

Solving linear systems

P. Berger, M. Daydé, R. Guivarch, E. Simon (INP-ENSEEIH- IRIT),
A. Buttari (CNRS, INP-ENSEEIH-IRIT),
P. Amestoy, J.-Y. L'Excellent (Mumps Technologies),
A. Guermouche (Univ. Bordeaux-LaBRI),
F.-H. Rouet (Ansys, USA),
Bora Uçar (INRIA-CNRS/LIP-ENS Lyon) and
L. Giraud (INRIA)
2024 - ModIA

Ronan Guivarch

Ronan.Guivarch@toulouse-inp.fr

- Enseignant au département Sciences du Numériques de l'INP-ENSEEIH : Calcul Scientifique, Probabilités, Programmation des cartes FPGA, Calcul Parallèle
- Chercheur au laboratoire IRIT, équipe APO (Algorithmes Parallèles et Optimisation) - Mathématiques Numériques / Calcul Parallèle

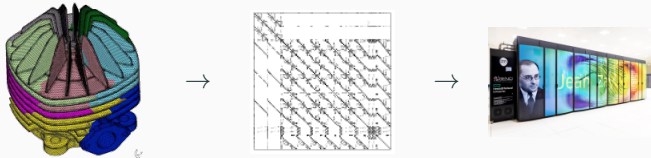
Résolution de systèmes linéaires issus des EDP (Moodle)

L'ensemble du cours sur la période Mars

- 12 CTD (avec Carola Kruse (CERFACS))
- 6 TP (matlab + FreeFem)
- 1 exam
- rendus de TP

Complété en Juin par la matière HPC (High Performance Computing)
[Algorithmes Parallèles,

Programmation Parallèle avec OpenMP (mémoire partagée et GPU) et
MPI (mémoire distribuée)] avec Alfredo Buttari et Ronan Guivarch.



Linear System $Ax = b$

At the foundations of many **scientific computing applications** (discretization of PDEs, step of an optimization method, ...).

Large-scale computations...

Up to few **billions (10^9) of unknowns**, applications asking TeraBytes (10^{12}) of memory and Exaflops (10^{18}) of computation.

...require large-scale computers.

Increasingly **large numbers of cores** available, high **heterogeneity in the computation** (CPU, GPU, FPGA, TPU, etc), and high **heterogeneity in data motions** (RAM to cache, out-of-core, node to node transfer, etc).

The sparse case

Matrix sparsity

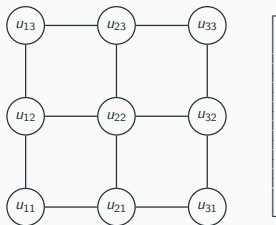
A **sparse matrix** is "any matrix with enough zeros that it pays to take advantage of them" (Wilkinson)

Physical problems are one of the main supplier for sparse linear systems. Let's consider the solution of the Poisson's PDE:

$$-\Delta u(x, y) = f$$

Finite-difference discretization gives the discrete equation:

$$(-\Delta u)_{i,j} \approx \frac{1}{h^2}(-u_{i+1,j} - u_{i-1,j} - u_{i,j+1} - u_{i,j-1} + 4u_{i,j}) = f_{i,j}$$


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

The sparse case

Matrix sparsity

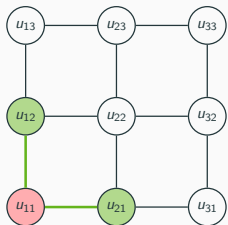
A **sparse matrix** is "any matrix with enough zeros that it pays to take advantage of them" (Wilkinson)

Physical problems are one of the main supplier for sparse linear systems. Let's consider the solution of the Poisson's PDE:

$$-\Delta u(x, y) = f$$

Finite-difference discretization gives the discrete equation:

$$(-\Delta u)_{i,j} \approx \frac{1}{h^2} (-u_{i+1,j} - u_{i-1,j} - u_{i,j+1} - u_{i,j-1} + 4u_{i,j}) = f_{i,j}$$


$$\begin{bmatrix} 4 & -1 & -1 \end{bmatrix} \begin{bmatrix} u_{11} \end{bmatrix} = \begin{bmatrix} -h^2 f_{11} \end{bmatrix}$$

The sparse case

Matrix sparsity

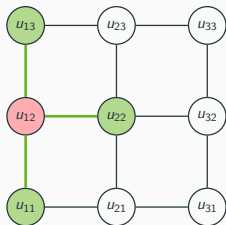
A **sparse matrix** is "any matrix with enough zeros that it pays to take advantage of them" (Wilkinson)

Physical problems are one of the main supplier for sparse linear systems. Let's consider the solution of the Poisson's PDE:

$$-\Delta u(x, y) = f$$

Finite-difference discretization gives the discrete equation:

$$(-\Delta u)_{i,j} \approx \frac{1}{h^2}(-u_{i+1,j} - u_{i-1,j} - u_{i,j+1} - u_{i,j-1} + 4u_{i,j}) = f_{i,j}$$



$$\begin{bmatrix} 4 & -1 & -1 \\ -1 & 4 & -1 \\ -1 & -1 & 4 \end{bmatrix}$$

$$\begin{bmatrix} u_{11} \\ u_{12} \\ u_{21} \end{bmatrix} = \begin{bmatrix} -h^2 f_{11} \\ -h^2 f_{12} \\ -h^2 f_{21} \end{bmatrix}$$

The sparse case

Matrix sparsity

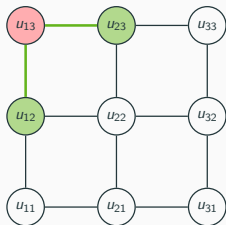
A **sparse matrix** is "any matrix with enough zeros that it pays to take advantage of them" (Wilkinson)

Physical problems are one of the main supplier for sparse linear systems. Let's consider the solution of the Poisson's PDE:

$$-\Delta u(x, y) = f$$

Finite-difference discretization gives the discrete equation:

$$(-\Delta u)_{i,j} \approx \frac{1}{h^2}(-u_{i+1,j} - u_{i-1,j} - u_{i,j+1} - u_{i,j-1} + 4u_{i,j}) = f_{i,j}$$



$$\begin{bmatrix} 4 & -1 & -1 \\ -1 & 4 & -1 \\ -1 & 4 & -1 \end{bmatrix}$$

$$\begin{bmatrix} u_{11} \\ u_{12} \\ u_{13} \end{bmatrix} = \begin{bmatrix} -h^2 f_{11} \\ -h^2 f_{12} \\ -h^2 f_{13} \end{bmatrix}$$

The sparse case

Matrix sparsity

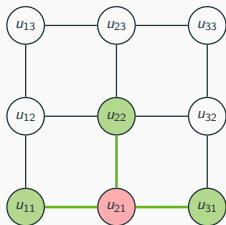
A **sparse matrix** is "any matrix with enough zeros that it pays to take advantage of them" (Wilkinson)

Physical problems are one of the main supplier for sparse linear systems. Let's consider the solution of the Poisson's PDE:

$$-\Delta u(x, y) = f$$

Finite-difference discretization gives the discrete equation:

$$(-\Delta u)_{i,j} \approx \frac{1}{h^2}(-u_{i+1,j} - u_{i-1,j} - u_{i,j+1} - u_{i,j-1} + 4u_{i,j}) = f_{i,j}$$



$$\begin{bmatrix} 4 & -1 & -1 & & \\ -1 & 4 & -1 & & \\ & -1 & 4 & -1 & \\ -1 & & & 4 & -1 \\ & & & -1 & 4 \end{bmatrix} \begin{bmatrix} u_{11} \\ u_{12} \\ u_{13} \\ u_{21} \\ u_{22} \end{bmatrix} = \begin{bmatrix} -h^2 f_{11} \\ -h^2 f_{12} \\ -h^2 f_{13} \\ -h^2 f_{21} \\ -h^2 f_{22} \end{bmatrix}$$

The sparse case

Matrix sparsity

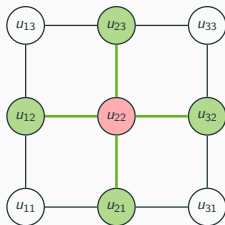
A **sparse matrix** is "any matrix with enough zeros that it pays to take advantage of them" (Wilkinson)

Physical problems are one of the main supplier for sparse linear systems. Let's consider the solution of the Poisson's PDE:

$$-\Delta u(x, y) = f$$

Finite-difference discretization gives the discrete equation:

$$(-\Delta u)_{i,j} \approx \frac{1}{h^2}(-u_{i+1,j} - u_{i-1,j} - u_{i,j+1} - u_{i,j-1} + 4u_{i,j}) = f_{i,j}$$



$$\begin{bmatrix} 4 & -1 & & -1 & & \\ -1 & 4 & -1 & & -1 & \\ & -1 & 4 & & -1 & \\ -1 & & & 4 & -1 & -1 \\ & -1 & & -1 & 4 & -1 \\ & & & & -1 & 4 \end{bmatrix} \begin{bmatrix} u_{11} \\ u_{12} \\ u_{13} \\ u_{21} \\ \mathbf{u_{22}} \\ u_{23} \\ u_{31} \\ u_{32} \\ u_{33} \end{bmatrix} = \begin{bmatrix} -h^2 f_{11} \\ -h^2 f_{12} \\ -h^2 f_{13} \\ -h^2 f_{21} \\ -h^2 f_{22} \\ -h^2 f_{23} \\ -h^2 f_{31} \\ -h^2 f_{32} \\ -h^2 f_{33} \end{bmatrix}$$

The sparse case

Matrix sparsity

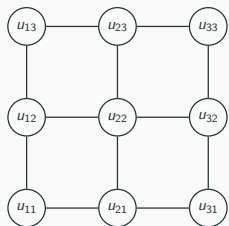
A **sparse matrix** is "any matrix with enough zeros that it pays to take advantage of them" (Wilkinson)

Physical problems are one of the main supplier for sparse linear systems. Let's consider the solution of the Poisson's PDE:

$$-\Delta u(x, y) = f$$

Finite-difference discretization gives the discrete equation:

$$(-\Delta u)_{i,j} \approx \frac{1}{h^2}(-u_{i+1,j} - u_{i-1,j} - u_{i,j+1} - u_{i,j-1} + 4u_{i,j}) = f_{i,j}$$


$$\begin{bmatrix} 4 & -1 & -1 & & & \\ -1 & 4 & -1 & -1 & & \\ & -1 & 4 & & -1 & \\ -1 & & & 4 & -1 & -1 \\ & -1 & -1 & -1 & 4 & -1 \\ & & -1 & & -1 & 4 & -1 \\ & & & -1 & & -1 & 4 \end{bmatrix} \begin{bmatrix} u_{11} \\ u_{12} \\ u_{13} \\ u_{21} \\ u_{22} \\ u_{23} \\ u_{31} \\ u_{32} \\ u_{33} \end{bmatrix} = \begin{bmatrix} -h^2 f_{11} \\ -h^2 f_{12} \\ -h^2 f_{13} \\ -h^2 f_{21} \\ -h^2 f_{22} \\ -h^2 f_{23} \\ -h^2 f_{31} \\ -h^2 f_{32} \\ -h^2 f_{33} \end{bmatrix}$$

What are the ways to solve a $Ax = b \in \mathbb{R}^n$ on computers ?

Iterative solvers

Compute a sequence of x_k converging towards x .

Examples: Gauss-Seidel, SOR, Krylov subspace methods, etc.

- Low computational cost and memory consumption if the convergence is quick (about $\mathcal{O}(n^2)$ (dense case)) operations per iteration)...
- BUT convergence depends on the matrix properties.

Direct solvers

Compute a factorization of A followed by forward and backward substitutions.

Examples: LDL^T , LU, QR, etc.

- High computational cost and memory consumption...
- BUT they are robust and easy to use.

What are the ways to solve a **sparse** $Ax = b \in \mathbb{R}^n$ on computers ?

Iterative solvers

Compute a sequence of x_k converging towards x .

Examples: Gauss-Seidel, SOR, **Krylov subspace methods**, etc.

- **Low computational cost** and **memory consumption** if the convergence is quick (about $\mathcal{O}(nnz(A))$ (sparse case)) operations per iteration)...
- BUT convergence **depends on the matrix properties**.

Direct solvers: dense/sparse matrix \Rightarrow dense/sparse solver

Compute a factorization of A followed by forward and backward substitutions.

Examples: LDL^T , LU, QR, etc.

- **High computational cost** and **memory consumption**...
- BUT they are **robust** and **easy to use**.

Quality of a solution: conditioning and errors

Dense Direct Solvers

Iterative Methods: Basic Methods

Iterative Methods: Krylov Methods

Iterative Methods: Symmetric Krylov Solvers

Iterative Method: Conjugate Gradient

Iterative Methods: Algebraic preconditioning techniques

Sparse Direct Solvers - Ordering

Sparse Direct Solvers - Multifrontal Method

Quality of a solution: conditioning and errors

Quality of a solution: conditioning and errors

Conditionnement

Conditionnement - Sensibilité du problème

- Soit A :

$$\begin{bmatrix} .780 & .563 \\ .913 & .659 \end{bmatrix} \text{ matrice presque singulière}$$

- Soit A' :

$$\begin{bmatrix} .780 & .563001095 \\ .913 & .659 \end{bmatrix} \text{ matrice singulière}$$

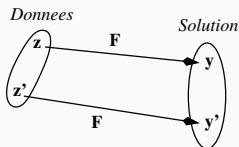
→ une perturbation des entrées de la matrice d'ordre $O(10^{-6})$ rend le problème insoluble.

- Autre situation si A est presque singulière : une petite perturbation sur A et/ou b (mesure, numérique, ...) → grandes perturbations sur la solution

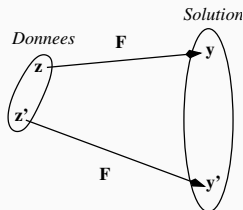
Ceci est propre au problème,
et est indépendant de l'algorithme de résolution utilisé

Conditionnement d'un problème

Le conditionnement peut être mesuré par le changement relatif solution /
changement relatif données $\frac{\| \frac{F(z') - F(z)}{F(z)} \|}{\| \frac{z' - z}{z} \|}$



Bien conditionné



Mal conditionné

Définition (Problème bien conditionné)

Un problème est bien conditionné si une faible variation des données entraîne une faible variation de la solution (c.à.d. $\|z - z'\|$ petit $\Rightarrow \|F(z) - F(z')\|$ petit)

$$Ax = b$$

Les coefficients de A et b sont des variables calculées, numériquement incorrectes.



Influence de ces erreurs sur le résultat du système :

$$(A + \Delta A) \cdot (x + \Delta x) = b + \Delta b$$



Recherche d'un majorant de $\|(x + \Delta x) - x\|/\|x\| = \|\Delta x\|/\|x\|$

Avec une norme induite, résultat généralement sous la forme :

$$K(A) = \|A\| \cdot \|A^{-1}\| \text{ nombre de conditionnement de } A$$

$$\|\Delta x\|/\|x\| \leq K(A) (\|\Delta A\|/\|A\| + \|\Delta b\|/\|b\|)$$

- Plus $K(A)$ est faible, meilleur est le conditionnement du système
- Valeur minimale de $K(A)$? $K(A) = \|A\| \|A^{-1}\| \geq \|A \cdot A^{-1}\| = 1$
- Selon la norme, $K(A)$ prend des valeurs différentes
- Si toutes les valeurs propres de A sont réelles (par ex., A symétrique) :

$$|\lambda_1| \geq \dots \geq |\lambda_n| \quad K(A) = \|A\|_2 \|A^{-1}\|_2 = |\lambda_1|/|\lambda_n|$$

Conditionnement

- Si A est orthogonale, $K(A) = 1$
- $K(A)$ élevé ne prouve que l'existence d'un risque de mauvais comportement lors de la résolution
- **Comment** évaluer $K(A)$? Ne jamais évaluer A^{-1} et utiliser de préférence des propriétés liées aux valeurs propres
- **Quand** calculer $K(A)$? Si l'on envisage d'utiliser une méthode de résolution connue pour être sensible au conditionnement

Et si $K(A) \gg 1$?? Changer de méthode ou **préconditionner** :

$$A \cdot x = b \rightarrow C \cdot A \cdot x = C \cdot b \text{ avec } K(C \cdot A) < K(A)$$

- Choix idéal de C : A^{-1} ????
- Choix réaliste de C : une approximation de A^{-1}

Quality of a solution: conditioning and errors

Forward and Backward Errors

- Considérons le système linéaire :

$$\begin{bmatrix} .780 & .563 \\ .913 & .659 \end{bmatrix} x = \begin{bmatrix} .217 \\ .254 \end{bmatrix}$$

- Supposons que deux algorithmes différents donnent les deux solutions suivantes :

$$x_1 = \begin{bmatrix} -20.568 \\ 28.881 \end{bmatrix} \text{ and } x_2 = \begin{bmatrix} 0.999 \\ -1.00 \end{bmatrix}$$

De x_1 et x_2 , quelle est la meilleure solution ?

- Résidus :

$$b - Ax_1 = \begin{bmatrix} -37 \\ -5 \end{bmatrix} \times 10^{-5} \text{ and } b - Ax_2 = \begin{bmatrix} -78 \\ -91 \end{bmatrix} \times 10^{-4}$$

- x_1 est la meilleure solution car son résidu est le plus petit
- Solution exacte :

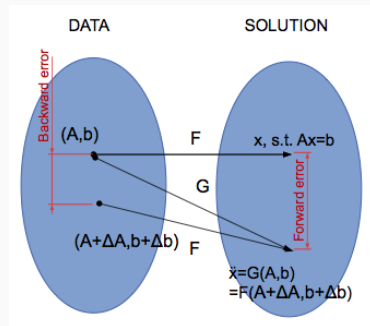
$$x^* = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

- en fait, x_2 est plus proche de la solution exacte

La notion de bonne solution est un concept ambigu

Erreurs directe et inverse

- F fait correspondre (précision exacte) les données (A, b) à la solution x de $Ax = b$
- G est un algorithme implémentant F et travaillant en précision finie
- \tilde{x} est telle que
 $\tilde{x} = G(A, b) = F(A + \Delta A, b + \Delta b)$



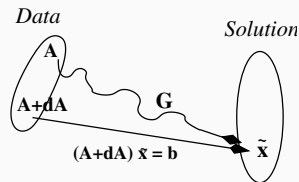
solution calculée (algorithme G) = solution exacte d'un problème perturbé

- Amplitude des perturbations (**erreur inverse**) associée à l'erreur dans la solution (**erreur directe**)
- La solution calculée sera satisfaisante si elle est la solution exacte d'un problème perturbé avec des perturbations acceptables pour l'utilisateur (ordre des erreurs d'arrondi, incertitude sur les données, ...)

Erreur inverse d'un algorithme résolvant $Ax = b$

soit \tilde{x} la solution approchée calculée par l'algorithme G .

Objectif : trouver les plus petites perturbations ΔA et Δb telles que $(A + \Delta A)\tilde{x} = b + \Delta b$



Théorème (Erreur Inverse (Rigal and Gaches, 1967))

Si nous définissons *l'erreur inverse (backward error)*

$$\eta_{A,b}^N(\tilde{x}) = \min \{ \varepsilon > 0 \text{ tel que } \|\Delta A\|_2 \leq \varepsilon \|A\|_2, \|\Delta b\|_2 \leq \varepsilon \|b\|_2, \\ \text{et } (A + \Delta A)\tilde{x} = b + \Delta b \}$$

Alors on peut prouver que :

$$\eta_{A,b}^N(\tilde{x}) = \frac{\|b - A\tilde{x}\|_2}{\|A\|_2 \|\tilde{x}\|_2 + \|b\|_2}$$

Erreur Inverse ne faisant intervenir que b

$$\begin{aligned}\eta_b^N(\tilde{x}) &= \min \{ \varepsilon > 0 \text{ tel que } \|\Delta b\|_2 \leq \varepsilon \|b\|_2, \\ &\quad \text{et } A\tilde{x} = b + \Delta b \} \\ &= \frac{\|b - A\tilde{x}\|_2}{\|b\|_2} \\ &= \frac{\|r\|_2}{\|b\|_2} \quad (r \text{ est le résidu.})\end{aligned}$$

- $\eta_{A,b}^N$ et η_b^N sont des erreurs inverses "**Normwise**" et sont utilisées dans le critère d'arrêt des méthodes itératives

Erreur Inverse "*Componentwise*"

en considérant les perturbations au niveau des composantes, nous définissons :

$$\begin{aligned}\eta_{A,b}^C(\tilde{x}) &= \min \{ \varepsilon > 0 \text{ tel que } |\Delta A| \leq \varepsilon |A|, |\Delta b| \leq \varepsilon |b|, \\ &\quad \text{et } (A + \Delta A)\tilde{x} = b + \Delta b \} \\ &= \max_{i=1,\dots,n} \frac{|b - A\tilde{x}|_i}{(|A||\tilde{x}| + |b|)_i}\end{aligned}$$

avec la convention $\frac{0}{0} = 0$.

Cette erreur inverse convient bien aux problèmes présentant des différences de magnitude entre les composantes.

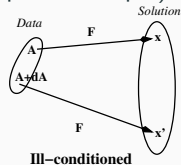
Quality of a solution: conditioning and errors

Synthèse (nombre de conditionnement
et erreur inverse)

- **Conditionnement** (de manière générale Δb doit être pris en compte) :

Approx. premier ordre $\frac{\|\Delta x\|}{\|x\|} \leq \kappa(A) \frac{\|\Delta A\|}{\|A\|}$

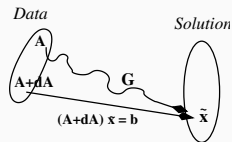
avec $\kappa(A) = \|A^{-1}\| \|A\|$



- **Erreur inverse :**

$$\eta_{A,b}^N(\tilde{x}) = \frac{\|A\tilde{x} - b\|}{\|A\| \|\tilde{x}\| + \|b\|}$$

(mesure la **qualité** d'un algorithme)



→ à comparer à l' ε -machine ou l'incertitude sur les données en entrée

- **Prédiction de l'erreur directe :**

Erreur Directe \leq **Nombre de Cond.** \times **Erreur Inverse**

Dense Direct Solvers

Dense Direct Solvers

**Gaussian Elimination and LU
factorization**

System of linear equations?

Example:

$$2x_1 - 1x_2 + 3x_3 = 13$$

$$-4x_1 + 6x_2 + 5x_3 = -28$$

$$6x_1 + 13x_2 + 16x_3 = 37$$

can be written under the form:

$$\mathbf{Ax} = \mathbf{b},$$

$$\text{with } \mathbf{A} = \begin{pmatrix} 2 & -1 & 3 \\ -4 & 6 & -5 \\ 6 & 13 & 16 \end{pmatrix}, \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}, \text{ and } \mathbf{b} = \begin{pmatrix} 13 \\ -28 \\ 37 \end{pmatrix}$$

Gaussian Elimination

Example:

$$2x_1 - x_2 + 3x_3 = 13 \quad (1)$$

$$-4x_1 + 6x_2 - 5x_3 = -28 \quad (2)$$

$$6x_1 + 13x_2 + 16x_3 = 37 \quad (3)$$

With $2 * (1) + (2) \rightarrow (2)$ and $-3 * (1) + (3) \rightarrow (3)$ we obtain:

$$2x_1 - x_2 + 3x_3 = 13 \quad (4)$$

$$0x_1 + 4x_2 + x_3 = -2 \quad (5)$$

$$0x_1 + 16x_2 + 7x_3 = -2 \quad (6)$$

Gaussian Elimination (cont.)

Thus x_1 is eliminated from (5) and (6). With $-4 * (5) + (6) \rightarrow (6)$:

$$2x_1 - x_2 + 3x_3 = 13$$

$$0x_1 + 4x_2 + x_3 = -2$$

$$0x_1 + 0x_2 + 3x_3 = 6$$

The linear system is solved by backward ($x_3 \rightarrow x_2 \rightarrow x_1$) substitution:

$$x_3 = \frac{6}{3} = 2$$

$$x_2 = \frac{1}{4}(-2 - x_3) = -1$$

$$x_1 = \frac{1}{2}(13 - 3x_3 + x_2) = 3$$

LU Factorization

- Find **L** unit lower triangular and **U** upper triangular such that:

$$\mathbf{A} = \mathbf{L} \times \mathbf{U}$$

$$\mathbf{A} = \begin{bmatrix} 2 & -1 & 3 \\ -4 & 6 & -5 \\ 6 & 13 & 16 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 3 & 4 & 1 \end{bmatrix} \times \begin{bmatrix} 2 & -1 & 3 \\ 0 & 4 & 1 \\ 0 & 0 & 3 \end{bmatrix}$$

- Procedure to solve $\mathbf{Ax} = \mathbf{b}$
 - $\mathbf{A} = \mathbf{LU}$
 - Solve $\mathbf{Ly} = \mathbf{b}$ (*forward elimination, down*)
 - Solve $\mathbf{Ux} = \mathbf{y}$ (*backward substitution, up*)

$$\mathbf{Ax} = (\mathbf{LU})\mathbf{x} = \mathbf{L}(\mathbf{Ux}) = \mathbf{Ly} = \mathbf{b}$$

From Gaussian Elimination to LU Factorization

$$\mathbf{A} = \mathbf{A}^{(1)}, \mathbf{b} = \mathbf{b}^{(1)}, \mathbf{A}^{(1)}\mathbf{x} = \mathbf{b}^{(1)}:$$

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} \quad \begin{array}{l} (2) \leftarrow (2) + (1) \times (-a_{21}/a_{11}) \text{ [2]} \\ (3) \leftarrow (3) + (1) \times (-a_{31}/a_{11}) \text{ [-3]} \end{array}$$

$$\mathbf{A}^{(2)}\mathbf{x} = \mathbf{b}^{(2)}$$

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} \\ 0 & a_{32}^{(2)} & a_{33}^{(2)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2^{(2)} \\ b_3^{(2)} \end{pmatrix} \quad \begin{array}{l} b_2^{(2)} = b_2 - a_{21}b_1/a_{11} \\ a_{32}^{(2)} = a_{32} - a_{31}a_{12}/a_{11} \end{array}$$

Finally $(3) \leftarrow (3) + (2) \times (-a_{32}/a_{22})$ [-4] gives $\mathbf{A}^{(3)}\mathbf{x} = \mathbf{b}^{(3)}$

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} \\ 0 & 0 & a_{33}^{(3)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2^{(2)} \\ b_3^{(3)} \end{pmatrix} \quad a_{33}^{(3)} = a_{33}^{(2)} - a_{32}^{(2)}a_{23}^{(2)}/a_{22}^{(2)}$$

Typical Gaussian elimination at step k :

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} a_{kj}^{(k)}, \text{ for } i > k$$

$$(\text{and } a_{ij}^{(k+1)} = a_{ij}^{(k)} \text{ for } i \leq k)$$

From Gaussian Elimination to LU factorization

$$\begin{cases} a_{ij}^{(k+1)} = a_{ij}^{(k)} - \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} a_{kj}^{(k)}, & \text{for } i > k \\ a_{ij}^{(k+1)} = a_{ij}^{(k)}, & \text{for } i \leq k \end{cases}$$

- One step of Gaussian elimination can be written: $\mathbf{A}^{(k+1)} = \mathbf{L}^{(k)} \mathbf{A}^{(k)}$ (and $b^{(k+1)} = \mathbf{L}^{(k)} b^{(k)}$), with

$$\mathbf{L}^k = \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & -l_{k+1,k} & \ddots & \\ & & \vdots & & \ddots \\ & & -l_{n,k} & & & 1 \end{pmatrix} \text{ and } l_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}.$$

- After $n-1$ steps, $\mathbf{A}^{(n)} = \mathbf{U} = \mathbf{L}^{(n-1)} \dots \mathbf{L}^{(1)} \mathbf{A}$ gives $\boxed{\mathbf{A} = \mathbf{LU}}$, with

$$\begin{aligned} \mathbf{L} &= [\mathbf{L}^{(1)}]^{-1} \dots [\mathbf{L}^{(n-1)}]^{-1} = \begin{pmatrix} 1 & & & \\ l_{21} & 1 & & \\ \vdots & & \ddots & \\ l_{n1} & & & 1 \end{pmatrix} \dots \begin{pmatrix} 1 & & & \\ & \ddots & & \\ & & 1 & \\ & & l_{n,n-1} & 1 \end{pmatrix} \\ &= \begin{pmatrix} 1 & & & 0 \\ l_{21} & 1 & & \\ \vdots & & \ddots & \\ l_{n1} & \dots & l_{n,n-1} & 1 \end{pmatrix} \end{aligned}$$

LU Factorization Algorithm

1. Overwrite matrix **A**: we store $a_{ij}^{(k)}$, $k = 2, \dots, n$ in $A(i, j)$

```
do k = 1, n-1
```

```
  L(k, k) = 1
```

```
  do i = k+1, n
```

```
    L(i, k) = A(i, k)/A(k, k)
```

```
    do j = k+1, n ! avoid building the zeros under  
                  the diagonal
```

```
      A(i, j) = A(i, j) - L(i, k) * A(k, j)
```

```
    end do
```

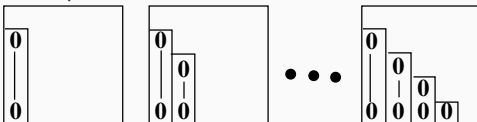
```
  end do
```

```
end do
```

```
L(n, n) = 1
```

▪ In the end, $\mathbf{A} = \mathbf{A}^{(n)} = \mathbf{U}$

Matrix **A** at each step:



2. Use lower triangle of array **A** to store $\mathbf{L}_{i,k}$ multipliers

- diagonal 1 of **L** is not stored:

```

do k = 1, n-1
  do i = k+1, n
    A(i, k) = A(i, k)/A(k, k)
    do j = k+1, n
      A(i, j) = A(i, j) - A(i, k) * A(k, j)
    end do
  end do
end do
end do
    
```


3. More compact array syntax (Matlab, scilab):

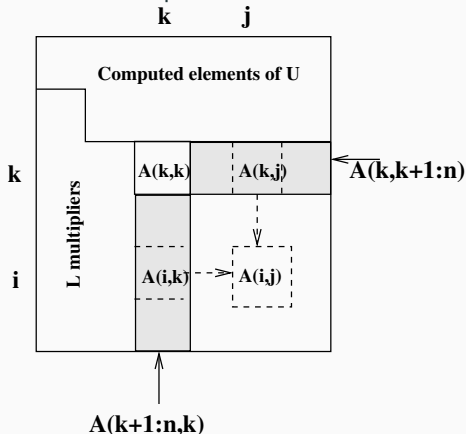
```
do k = 1, n-1
```

$$A(k+1:n, k) = A(k+1:n, k) / A(k, k)$$

$$A(k+1:n, k+1:n) = A(k+1:n, k+1:n) - A(k+1:n, k) * A(k, k+1:n)$$

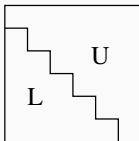
end do

- corresponds to a rank-1 update:



What we have computed

- we have stored the **L** and **U** factors in **A**:
 - $\mathbf{A}_{i,j}$, $i > j$ corresponds to l_{ij}
 - $\mathbf{A}_{i,j}$, $i \leq j$ corresponds to u_{ij}
 - with $l_{ii} = 1, i = 1, n$
- Finally,



after factorization: $\text{memory_zone}(\mathbf{A}) = \mathbf{L} + \mathbf{U} - I$

LU factorization : summary

- Step by step columns of \mathbf{A} are set to zero and \mathbf{A} is updated $\mathbf{L}^{(n-1)} \dots \mathbf{L}^{(1)} \mathbf{A} = \mathbf{U}$ leading to
 $\mathbf{A} = \mathbf{L}\mathbf{U}$ where $\mathbf{L} = [\mathbf{L}^{(1)}]^{-1} \dots [\mathbf{L}^{(n-1)}]^{-1}$
- At each step $\mathbf{A}(k, k)$ is referred to as the **pivot**
 - zero entries in column of \mathbf{A} can be replaced by entries in \mathbf{L}
 - row entries of \mathbf{U} are stored in corresponding locations of \mathbf{A}

for $k = 1, n - 1$ **do**

if $|\mathbf{A}(k, k)|$ too small **then**

 exit (small pivots are not allowed)

end if

$\mathbf{A}(k+1:n, k) = \mathbf{A}(k+1:n, k) / \mathbf{A}(k, k)$

$\mathbf{A}(k+1:n, k+1:n) = \mathbf{A}(k+1:n, k+1:n) - \mathbf{A}(k+1:n, k) * \mathbf{A}(k, k+1:n)$

end for

Algorithm 1: LU factorization

When $|\mathbf{A}(k, k)|$ is too small, one could consider other pivots: **numerical pivoting strategies** will be introduced later.

Existence and uniqueness of LU decomposition

Theorem

$\mathbf{A} \in \mathbb{R}^{n \times n}$ has an LU factorization (where \mathbf{L} is unit lower triangular and \mathbf{U} is upper triangular) iff $\det(\mathbf{A}(1:k, 1:k)) \neq 0$ for all $k \in \{1 \dots n-1\}$. If the LU factorization exists, then it is unique and $\det(\mathbf{A}) = u_{11} \dots u_{nn}$.

Theorem

For each nonsingular matrix \mathbf{A} , there exists a permutation matrix \mathbf{P} such that \mathbf{PA} possesses an LU factorization $\mathbf{PA} = \mathbf{LU}$.

Definition

$\mathbf{A} \in \mathbb{R}^{n \times n}$ is **strictly diagonally dominant** iff $|a_{ii}| > \sum_{j=1, j \neq i}^n |a_{ij}|$ for all $i = 1, \dots, n$.

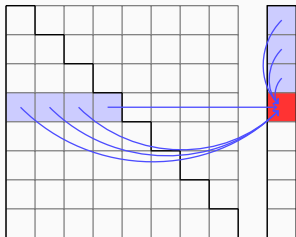
Theorem

If \mathbf{A}^T is strictly diagonally dominant then \mathbf{A} is non-singular and \mathbf{A} has an LU factorization and $|l_{ij}| \leq 1$.

Solution phase : $Lx = b$ (Left-Looking and Right-looking)

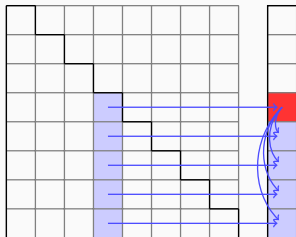
```
x = b
for j = 1, n do
  for i = 1, j - 1 do
     $x_j = x_j - l_{ji}x_i$ 
  end for
   $x_j = \frac{x_j}{l_{jj}}$ 
end for
```

Algorithm 2: LL



```
x = b
for j = 1, n do
   $x_j = \frac{x_j}{l_{jj}}$ 
  for i = j + 1, n do
     $x_i = x_i - l_{ij}x_j$ 
  end for
end for
```

Algorithm 3: RL



Number of floating-point operations (flops)

- In forward elimination ($Ly = b$), computing the k^{th} unknown

$$y_k = b_k - \sum_{j=1}^{k-1} L_{kj}y_j$$

leads to $(k-1)$ multiplications and $(k-1)$ additions, for $1 \leq k \leq n$
 $n^2 - n$ flops overall

- Idem for $Ux = y$ and at worst n divisions ($U_{kk} \neq 1$)

Number of floating-point operations (flops) (cont.)

```
for  $k = 1, n - 1$  do
  if  $|A(k, k)|$  too small then
    exit (small pivots are not allowed)
  end if
   $A(k+1:n, k) = A(k+1:n, k) / A(k, k)$ 
   $A(k+1:n, k+1:n) = A(k+1:n, k+1:n) - A(k+1:n, k) * A(k, k+1:n)$ 
end for
```

Algorithm 4: LU factorization

- Number of flops during factorization:
 - $n - k$ divisions
 - $(n - k)^2$ multiplications, $(n - k)^2$ additions
 - $k = 1, 2, \dots, n - 1$
 - total: $\approx \frac{2 \times n^3}{3}$
(Strassen's algorithm (recursive algo with the multiplication of 2 matrices of size $n \times n$) can reduce this to $\Theta(n^{\log_2 7}) \simeq \Theta(n^{2.8})$)

Exercise (How to compute \mathbf{x} such that $\mathbf{x} = (\mathbf{A}^2)^{-1} \mathbf{b}$)

Let \mathbf{A} be a non singular matrix of order n with the properties of the first theorem of slide 31 (i.e. it exists \mathbf{L} and \mathbf{U} such that $\mathbf{A} = \mathbf{LU}$); note that \mathbf{A}^2 is also non singular.

1. Compare the computational complexity of solving $\mathbf{A}^2 \mathbf{x} = \mathbf{b}$ with the following two algorithms:
 - 1.1 Compute $\mathbf{B} = \mathbf{A}^2$, factor \mathbf{B} and solve $\mathbf{Bx} = \mathbf{b}$
 - 1.2 Factor \mathbf{A} and use the factored form to solve $\mathbf{A}^2 \mathbf{x} = \mathbf{b}$
2. Explain why computing directly $\mathbf{C} = (\mathbf{A})^{-1}$ and performing $\mathbf{x} = \mathbf{Cb}$ is not a method of choice.

Dense Direct Solvers

LU Factorization with partial pivoting

LU Factorization: some numerical issues

$$\text{Consider } A = \begin{bmatrix} \varepsilon & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \frac{1}{\varepsilon} & 1 \end{bmatrix} \times \begin{bmatrix} \varepsilon & 1 \\ 0 & 1 - \frac{1}{\varepsilon} \end{bmatrix}$$

$$\kappa_2(A) = \frac{\lambda_{\max}}{\lambda_{\min}} = \frac{1 + \varepsilon + \sqrt{5 + \varepsilon^2 - 2\varepsilon}}{-1 - \varepsilon + \sqrt{5 + \varepsilon^2 - 2\varepsilon}} \simeq 2.6$$

If one solves:

$$\begin{bmatrix} \varepsilon & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 + \varepsilon \\ 2 \end{bmatrix}$$

Exact solution $x^* = (1, 1)$.

LU Factorization: some numerical issues

$$A = \begin{bmatrix} \varepsilon & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \frac{1}{\varepsilon} & 1 \end{bmatrix} \times \begin{bmatrix} \varepsilon & 1 \\ 0 & 1 - \frac{1}{\varepsilon} \end{bmatrix}$$

ε	$\frac{\ x^* - x\ }{\ x^*\ }$	$\kappa_2(A)$
10^{-3}	6×10^{-16}	2.621
10^{-6}	2×10^{-11}	2.618
10^{-9}	9×10^{-8}	2.618
10^{-12}	9×10^{-5}	2.618
10^{-15}	7×10^{-2}	2.618

Table 1: Relative error as a function of ε .

- Even if A is well conditioned, Gaussian elimination may introduce errors

LU Factorization: some numerical issues

$$A = \begin{bmatrix} \varepsilon & 1 \\ 1 & \mathbf{1} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \frac{1}{\varepsilon} & 1 \end{bmatrix} \times \begin{bmatrix} \varepsilon & 1 \\ 0 & \mathbf{1} - \frac{1}{\varepsilon} \end{bmatrix}$$

ε	$\frac{\ x^* - x\ }{\ x^*\ }$	$\kappa_2(A)$
10^{-3}	6×10^{-16}	2.621
10^{-6}	2×10^{-11}	2.618
10^{-9}	9×10^{-8}	2.618
10^{-12}	9×10^{-5}	2.618
10^{-15}	7×10^{-2}	2.618

Table 1: Relative error as a function of ε .

- Explanation: pivot ε is too small and leads to a large element growth (**growth factor**) in L and U : $\frac{1}{\varepsilon}$ in L leads to a loss of information/accuracy in $\mathbf{1} - \frac{1}{\varepsilon}$

LU Factorization: some numerical issues

$$A = \begin{bmatrix} \varepsilon & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \frac{1}{\varepsilon} & 1 \end{bmatrix} \times \begin{bmatrix} \varepsilon & 1 \\ 0 & 1 - \frac{1}{\varepsilon} \end{bmatrix}$$

- Let us try to exchange rows 1 and 2 of A and b :

$$\begin{bmatrix} 1 & 1 \\ \varepsilon & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 + \varepsilon \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 \\ \varepsilon & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \varepsilon & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & 1 - \varepsilon \end{bmatrix}$$

→ Multipliers are bounded: $\forall i = k + 1 : n, \frac{|a_{i,k}^{(k)}|}{|a_{k,k}^{(k)}|} \leq 1$

→ terms of original matrix remain significant in LU factors

→ perfect accuracy obtained !

Partial Pivoting

- Partial pivoting: choose at each step the largest element of the column as the pivot
- avoids large elements in factors matrix (**growth factor**)
- Then (P : permutation), $PA = LU$, $Ly = Pb$, $Ux = y$
 - LU with partial pivoting is generally *backward stable*

$$\frac{\|Ax - b\|}{\|A\| \times \|x\| + \|b\|} \approx \varepsilon \quad (1)$$

$$\frac{\|x - x^*\|}{\|x^*\|} \approx \varepsilon \times \kappa(A) \quad (2)$$

- (1) small backward error (and small residual) independently of the conditioning
- (2) accuracy depends on conditioning
if $\varepsilon \approx 10^{-q}$ et $\kappa(A) \approx 10^p$ then x has approximatively $(q - p)$ correct digits

LU factorization with partial pivoting

Next algorithm computes \mathbf{L} and \mathbf{U} such that $\mathbf{PA} = \mathbf{LU}$, and computes \mathbf{Pb} .

for $k = 1, n - 1$ **do**

Pivot search : Find index i of largest entry in $\mathbf{A}(k : n, k)$

if $|A(i, k)| \leq \varepsilon \|\mathbf{A}\|$ **then**

 exit since \mathbf{A} is numerically singular

end if

 Swap rows i and k of \mathbf{A} and \mathbf{b}

$\mathbf{A}(k+1:n, k) = \mathbf{A}(k+1:n, k) / \mathbf{A}(k, k)$

$\mathbf{A}(k+1:n, k+1:n) = \mathbf{A}(k+1:n, k+1:n) - \mathbf{A}(k+1:n, k) * \mathbf{A}(k, k+1:n)$

end for

Algorithm 5: LU factorization with partial pivoting

To solve $\mathbf{Ax} = \mathbf{b}$ one should thus perform $\mathbf{LU} = \mathbf{Pb}$. Permutation \mathbf{P} is either applied on \mathbf{b} during Algorithm 5 or needs be saved to be applied later.

Dense Direct Solvers

Symmetric matrices

Symmetric matrices

- Assumption: A has a LU factorization
- A symmetric: only store lower or upper triangle
- $A = LU$ and $A^T = A \Rightarrow LU = U^T L^T$,
thus $LU(L^T)^{-1} = U^T \Rightarrow U(L^T)^{-1} = L^{-1}U^T = D$ diagonal
and $U = DL^T$,
finally $A = L(DL^T) = LDL^T$, with $D = \text{Diag}(U)$
- Example:

$$\begin{bmatrix} 4 & -8 & -4 \\ -8 & 18 & 14 \\ -4 & 14 & 25 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -1 & 3 & 1 \end{bmatrix} \times \begin{bmatrix} 4 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix} \times \begin{bmatrix} 1 & -2 & -1 \\ 0 & 1 & 3 \\ 0 & 0 & 1 \end{bmatrix}$$

- Solution of $Ax = b$: with $A = LDL^T$:
 1. $Ly = b$ then $Dz = y$ followed by
 2. $L^T x = z$

Properties of LDL^T Algorithm

We have shown that if \mathbf{A} symmetric and $A = LU$ exists then $\exists L$ and $D (= \text{Diag}(U))$ such that $A = LDL^T$. **LU** Algorithm 1 thus already computes all we need: L and D .

Proposition (LDL^T Algorithm)

Given a symmetric matrix A for which an LU factorisation exists, the LU algorithm 1 can be adapted to compute LDL^T factorization.

Proposition (Complexity of LDL^T factorization)

If only the lower triangular part of the matrix (including diagonal) is used/updated and if only L and D matrices are stored, then the cost of LDL^T factorisation is $\approx \frac{n^3}{3}$

LDL^T Algorithm for symmetric matrices

- let l_k be column k of L and u_k be row k of U then in Algorithm 1,
 $A(k+1:n, k+1:n) = A(k+1:n, k+1:n) - l_k * u_k^T$
- since $l_k = u_k / u_{kk}$, when U is not stored, one must temporarily save u_k to perform the update.

for $k = 1, n - 1$ **do**

if $|A(k, k)|$ too small exit (small pivots)

$\mathbf{v}_k = A(k+1:n, k)$ (corresponds to u_k in **LU** Algorithm)

$A(k+1:n, k) = A(k+1:n, k) / A(k, k)$

for $j = k + 1, n$ **do**

$A(j:n, j) = A(j:n, j) - A(j:n, k) * \mathbf{v}_k(j)$

end for

end for

Algorithm 6: LDL^T factorization

Complexity of LDL^T factorization

```
for  $k = 1, n - 1$  do
  if  $|A(k, k)|$  too small exit (small pivots)
   $\mathbf{v}_k = A(k+1:n, k)$  (corresponds to  $u_k$  in LU Algorithm)
   $A(k+1:n, k) = A(k+1:n, k) / A(k, k)$ 
  for  $j = k + 1, n$  do
     $A(j:n, j) = A(j:n, j) - A(j:n, k) * \mathbf{v}_k(j)$ 
  end for
end for
```

$$\blacksquare \text{ flops}(LDL^T) = 2 \sum_{k=1}^n \left(\sum_{i=1}^{n-k} i \right) = 2 \sum_{k=1}^n \left(\frac{(n-k)(n-k-1)}{2} \right)$$

$$\text{flops}(LDL^T) \approx \sum_{k=1}^n (n-k)^2 \quad (\text{thus } \frac{1}{2} \text{flops}(\mathbf{LU}))$$

$$LDL^T \approx \frac{n^3}{3} \text{floating point operations}$$

Symmetric matrices and pivoting

- Diagonal pivoting preserves symmetry but is insufficient for stability
- In general one looks for a permutation P such that:

$$PAP^T = LDL^T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ x & 1 & 0 & 0 \\ x & 0 & 1 & 0 \\ x & x & x & 1 \end{bmatrix} \times \begin{bmatrix} x & 0 & 0 & 0 \\ 0 & x & x & 0 \\ 0 & x & x & 0 \\ 0 & 0 & 0 & x \end{bmatrix} \times \begin{bmatrix} 1 & x & x & x \\ 0 & 1 & 0 & x \\ 0 & 0 & 1 & x \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- D : matrix of diagonal 1×1 and 2×2 blocks

Examples of 2×2 pivots: $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$, $\begin{bmatrix} \varepsilon_1 & 1 \\ 1 & \varepsilon_2 \end{bmatrix}$

- Pivot choice more complex: 2 columns at each step Let

$PAP^T = \begin{bmatrix} E & C^T \\ C & B \end{bmatrix}$. If E is a 2×2 pivot, form E^{-1} to get:

$$PAP^T = \begin{bmatrix} I & 0 \\ CE^{-1} & I \end{bmatrix} \begin{bmatrix} E & 0 \\ 0 & B - CE^{-1}C^T \end{bmatrix} \begin{bmatrix} I & E^{-1}C^T \\ 0 & I \end{bmatrix}$$

Dense Direct Solvers

Cholesky Factorization

Cholesky Factorization

- **A** positive definite if $\mathbf{x}^T \mathbf{A} \mathbf{x} > 0 \quad \forall \mathbf{x} \neq 0$
- **A** symmetric positive definite \Rightarrow Cholesky factorization
 $\mathbf{A} = \mathbf{L} \mathbf{L}^T$ with L lower triangular, **L** is unique
- By identification :

$$\begin{bmatrix} a_{11} & a_{21} & a_{31} \\ a_{21} & a_{22} & a_{32} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \times \begin{bmatrix} l_{11} & l_{21} & l_{31} \\ 0 & l_{22} & l_{32} \\ 0 & 0 & l_{33} \end{bmatrix}$$

- It follows: $a_{11} = \rho_{11} \quad a_{21} = l_{21} \times l_{11} \quad a_{31} = l_{31} \times l_{11}$
 $a_{22} = \rho_{21} + \rho_{22} \quad a_{32} = l_{31} \times l_{21} + l_{32} \times l_{22} \quad a_{33} = \rho_{31} + \rho_{32} + \rho_{33}$

Thus:

$$\begin{array}{lll} l_{11} = \sqrt{a_{11}} & l_{21} = a_{21}/l_{11} & l_{31} = a_{31}/l_{11} \\ a_{22}^{(1)} = a_{22} - \rho_{21} & a_{32}^{(1)} = a_{32} - l_{31} \times l_{21} & a_{33}^{(1)} = a_{33} - \rho_{31} \\ l_{22} = \sqrt{a_{22}^{(1)}} & a_{32}^{(2)} = a_{32}^{(1)} - l_{32} \times l_{22} & a_{33}^{(2)} = a_{33}^{(1)} - \rho_{32} \\ l_{33} = \sqrt{a_{33}^{(2)}} & & \end{array}$$

Cholesky factorization

Cholesky Factorization

```
do k=1, n
  A(k,k)=sqrt(A(k,k))
  A(k+1:n,k) = A(k+1:n,k)/A(k,k)
  do j=k+1, n
    A(j:n,j) = A(j:n,j) - A(j:n,k) A(j,k)
  end do
end do
```

- Cholesky Factorization is backward stable (without pivoting)
- Factorization: $\approx \frac{n^3}{3}$ flops
- Similar to LU , but only the lower part is updated.
- **LU** Factorization (internal loop):

$$A(k+1:n, k) = A(k+1:n, k) / A(k, k)$$

$$A(k+1:n, k+1:n) = A(k+1:n, k+1:n) - \&$$

$$A(k+1:n, k) * A(k, k+1:n)$$

Make your own summary on direct methods for dense matrices

Iterative Methods: Basic Methods

Principles:

- Generates a sequence of approximates $x^{(k)}$ to the solution
- Essentially involves matrix-vector products
- Often linked to preconditioning techniques: $Ax = b \rightarrow MAx = Mb$
- Evaluation of a method \Leftrightarrow speed of convergence

Direct method vs. Iterative method

Direct

- Very general technique
 - High numerical accuracy
 - Sparse matrices with irregular patterns
- Factorization of A
 - May be costly in terms of memory for factors
 - Factors can be reused for successive/multiple right-hand sides

Iterative

- Efficiency depends on the type of the problem
 - Convergence preconditioning
 - Numerical properties structure of A
- Requires the product of A by a vector
 - Less costly in terms of memory and possibly flops
 - Solutions with successive right-hand sides can be problematic

Iterative Methods: Basic Methods

Stationary methods

Basic iterative methods (stationary methods)

Definition

An iterative method is called **stationary** if $x^{(k+1)}$ can be expressed as a function of $x^{(k)}$ only.

- Residual at iteration k : $r^{(k)} = b - Ax^{(k)}$
- i^{th} component: $r_i^{(k)} = b_i - \sum_j a_{ij}x_j^{(k)} = b_i - \sum_{j \neq i} a_{ij}x_j^{(k)} - a_{ii}x_i^{(k)}$
- Idea: try to “reset” the r_i components to 0. This gives:

Do $i = 1, n$

$$x_i^{(k+1)} = \frac{1}{a_{ii}}(b_i - \sum_{j \neq i} a_{ij}x_j^{(k)})$$

EndDo

Basic iterative methods (stationary methods)

Definition

An iterative method is called **stationary** if $x^{(k+1)}$ can be expressed as a function of $x^{(k)}$ only.

- Residual at iteration k : $r^{(k)} = b - Ax^{(k)}$
- i^{th} component: $r_i^{(k)} = b_i - \sum_j a_{ij}x_j^{(k)} = b_i - \sum_{j \neq i} a_{ij}x_j^{(k)} - a_{ii}x_i^{(k)}$
- Idea: try to “reset” the r_i components to 0. This gives:

Do $i = 1, n$

$$x_i^{(k+1)} = \frac{1}{a_{ii}}(b_i - \sum_{j \neq i} a_{ij}x_j^{(k)})$$

EndDo

Jacobi iteration

- Jacobi:

Do $i = 1, n$

$$x_i^{(k+1)} = \frac{1}{a_{ii}}(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^n a_{ij}x_j^{(k)})$$

EndDo

- Remark that one does not use the latest information
- Gauss-Seidel iteration:

Do $i = 1, n$

$$x_i^{(k+1)} = \frac{1}{a_{ii}}(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^n a_{ij}x_j^{(k)})$$

EndDo

Stationary Methods: Matrix Approach

Decompose A as

$$A = L + D + U$$

where D is the diagonal of A , and L (resp. U) is the strictly lower (resp. upper) triangular part.

- Given a non-singular matrix M , we define the recurrence:

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

(Note: $y = M^{-1}z$ means “Solve $My = z$ for y ”)

- Jacobi iteration:

$$M = D$$

$$x^{(k+1)} = D^{-1}(b - (A - D)x^{(k)}) = x^{(k)} + D^{-1}r^{(k)}$$

- Gauss-Seidel iteration:

$$M = L + D$$

$$x^{(k+1)} = (D + L)^{-1}(b - (A - D - L)x^{(k)}) = x^{(k)} + (L + D)^{-1}r^{(k)}$$

Successive over-relaxation (SOR):

$$\begin{aligned}x^{(k+1)} &= \omega x_{\text{GaussSeidel}}^{(k+1)} + (1 - \omega)x^{(k)} \\ &= x^k + \omega(D + \omega L)^{-1}(b - Ax^{(k)})\end{aligned}$$

Choice of ω :

- Theoretical optimal values for limited classes of problems
- Problematic in general

Many other variants depending on the choice of the matrix M (block Jacobi, block Gauss-Seidel, ...)

Convergence properties

Previous methods follow the model:

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

Knowing that the solution x^* satisfies $Ax^* = b$, this gives:

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

Thus:

$$x^{(k+1)} - x^* = (I - M^{-1}A)(x^{(k)} - x^*)$$

Theorem

The sequence $(x^{(k)})_{k=1,2,\dots}$ defined by

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

converges for all $x^{(0)}$ to the solution x^ iff the spectral radius of $I - M^{-1}A$ satisfies the inequality $\rho(I - M^{-1}A) < 1$.*

Theorem

If $A \in \mathbb{C}^{n \times n}$ is strictly diagonal dominant, then the Jacobi iterations converge.

Theorem

If A is symmetric positive definite, then the Gauss-Seidel iterations converge (for any x_0).

Conclusion on stationary methods

- Relatively easy to implement and parallelize
- Often depend on parameters that are difficult to forecast (example: ω in SOR)
- Convergence difficult to guarantee in finite precision arithmetic
- other "basic" iterative methods: Gradient methods (Steepest Descent, A -conjugate Directions, Conjugate Gradient)

⇒ Krylov Methods

[Fin Séance 2 2023]

Iterative Methods: Krylov Methods

- Y. Saad, *Iterative Methods for Sparse Linear Systems*, SIAM, 2nd edition
- C.T. Kelley, *Iterative Methods for Linear and Nonlinear Equations*, SIAM
- Luc Giraud and Serge Gratton, "Introduction to Krylov subspace methods for the solution of linear systems" (pdf on Moodle)

Iterative Methods: Krylov Methods

Some background

Krylov methods: some background



Aleksei Nikolaevich Krylov

1863-1945: Russia, Maritime Engineer

His research spans a wide range of topics, including shipbuilding, magnetism, artillery, mathematics, astronomy, and geodesy. In 1904 he built the first machine in Russia for integrating ODEs.

In 1931 he published a paper on what is now called the "Krylov subspace".

(Most slides of this part are from Luc Giraud / INRIA)

Krylov methods: some background

Definition

Let $A \in \mathbb{R}^{n \times n}$ and $r \in \mathbb{R}^n$; the space denoted by $\mathcal{K}_m(A, r)$ (with $m \leq n$) and defined by

$$\mathcal{K}_m(A, r) = \text{Span}\{r, Ar, \dots, A^{m-1}r\}$$

is referred to as the Krylov space of dimension m associated with A and r .

Assumption

Let us assume $x_0 = 0$

No loss of generality,

because the situation $x_0 \neq 0$ can be transformed with a simple shift to the system $Ay = b - Ax_0 = \bar{b}$,

for which obviously $y_0 = 0$.

Why using this search space?

The **minimal polynomial** $q(t)$ of A is the unique monic polynomial of minimal degree such that $q(A) = 0$.

It is constructed from the eigenvalues of A as follows:

If the distinct eigenvalues of A are $\lambda_1, \dots, \lambda_\ell$ and if λ_j has index m_j (the size of the largest Jordan block associated with λ_j), then the sum of all indices is

$$m = \sum_{j=1}^{\ell} m_j, \text{ and } q(t) = \prod_{j=1}^{\ell} (t - \lambda_j)^{m_j}. \quad (7)$$

When A is diagonalizable, m is the number of distinct eigenvalues of A .

When A is a Jordan block of size n , then $m = n$.

Why using this search space?

If we write

$$q(t) = \prod_{j=1}^{\ell} (t - \lambda_j)^{m_j} = \sum_{j=0}^m \alpha_j t^j,$$

then the constant term is $\alpha_0 = \prod_{j=1}^{\ell} (-\lambda_j)^{m_j}$.

Therefore $\alpha_0 \neq 0$ iff A is nonsingular.

Furthermore, from

$$0 = q(A) = \alpha_0 I + \alpha_1 A + \dots + \alpha_m A^m, \quad (8)$$

it follows that

$$A^{-1} = -\frac{1}{\alpha_0} \sum_{j=0}^{m-1} \alpha_{j+1} A^j.$$

This description of A^{-1} portrays $x = A^{-1}b$ immediately as a member of the Krylov space of dimension m associated with A and b denoted by $\mathcal{K}_m(A, b) = \text{Span}\{b, Ab, \dots, A^{m-1}b\}$.

Taxonomy of the Krylov subspace approaches

The Krylov methods for identifying $x_m \in \mathcal{K}_m(A, b)$ can be distinguished in four classes:

- **The Galerkin approach (or Ritz-Galerkin approach) (FOM, CG,...):**
construct x_m such that the residual is orthogonal to the current subspace: $b - Ax_m \perp \mathcal{K}_m(A, b)$.
- **The minimum norm residual approach (GMRES,...):**
construct $x_m \in \mathcal{K}_m(A, b)$ such that $\|b - Ax_m\|_2$ is minimal that means that the residual is orthogonal to the subspace $A\mathcal{K}_m(A, b)$ (ie $b - Ax_m \perp A\mathcal{K}_m(A, b)$).
- **The Petrov-Galerkin approach:**
construct x_m such that $b - Ax_m$ is orthogonal to some other m -dimensional subspace.
- **The minimum norm error approach:**
construct $x_m \in A^T\mathcal{K}_m(A, b)$ such that $\|b - Ax_m\|_2$ is minimal.

Constructing a basis of $\mathcal{K}_m(A, b)$

- Obvious choice $b, Ab, \dots, A^{m-1}b$
 - not very attractive from the numerical point of view because vectors $A^j b$ become more and more colinear to the eigenvector associated to the largest eigenvalue.
 - In finite arithmetic, leads to a loss of rank: suppose A is diagonalizable $A = VDV^{-1}$, then $A^k b = VD^k(V^{-1}b)$.
 - A better choice is the Arnoldi procedure.

Iterative Methods: Krylov Methods

Arnoldi procedure



Walter Edwin Arnoldi

1917-1995: USA.

His main research subjects covered vibration of propellers, engines and aircraft, high speed digital computers, aerodynamics and acoustics of aircraft propellers, lift support in space vehicles and structural materials.

"The principle of minimized iterations in the solution of the eigenvalue problem" in Quart. of Appl. Math., Vol.9 in 1951.

The Arnoldi procedure

=> builds an orthonormal basis of $\mathcal{K}_m(A, b)$

```
1  $v_1 = b/\|b\|$ 
2 for  $j = 1, 2, \dots, m$  do
3   for  $i = 1, \dots, j$  do
4     | Compute  $h_{i,j} = v_i^T A v_j$ 
5   end
6    $w_j = A v_j - \sum_{i=1}^j h_{i,j} v_i$ 
7   Compute  $h_{j+1,j} = \|w_j\|$ 
8   exit if ( $h_{j+1,j} = 0$ )
9   Compute  $v_{j+1} = w_j/h_{j+1,j}$ 
10 end
```

Algorithm 7: Arnoldi procedure

Steps $\{3,4,5\}$ and 6 : Classical Gram-Schmidt (CGS)

The Arnoldi procedure properties

Proposition

If the Arnoldi procedure does not stop before the m^{th} step, the vectors v_1, \dots, v_m form an orthonormal basis of the Krylov subspace $\mathcal{K}_m(A, b)$.

Proof.

The vectors are orthogonal by construction and have a norm equal to 1.

They span $\mathcal{K}_{m-1}(A, b)$ follows from the fact that each vector v_j is of the form $q_{j-1}(A)v_1$, where q_{j-1} is a polynomial of degree $j-1$.

This can be shown by induction.

For $j=1$ it is true as $v_1 = q_0(A)v_1$ with $q_0 = 1$.

Assume that it is true for all j and consider v_{j+1} .

We have:

$$h_{j+1,j}v_{j+1} = Av_j - \sum_{i=1}^j h_{i,j}v_i = Aq_{j-1}(A)v_1 + \sum_{i=1}^j h_{i,j}q_{i-1}(A)v_1.$$

So v_{j+1} can be expressed as $q_j(A)v_1$ where q_j is of degree j .



The Arnoldi procedure properties

Proposition

Denote

- V_m the $n \times m$ matrix with column vector v_1, \dots, v_m
- \bar{H}_m the $(m+1) \times m$ Hessenberg matrix whose nonzero entries are h_{ij}
- H_m the square matrix obtained from \bar{H}_m by deleting its last row.

Then the following relations hold:

$$AV_m = V_m H_m + h_{m+1,m} v_{m+1} e_m^T \quad (9)$$

$$AV_m = V_{m+1} \bar{H}_m \quad (10)$$

$$V_m^T AV_m = H_m \quad (11)$$

$$A V_m = V_m H_m + v_{m+1} e_m^T$$

Proof.

Equality (9) can be shown for each column. Using Matlab notations:

For $j \neq m$

$$\begin{aligned} AV_m(:, j) &= V_m H(:, j) + w_m \times 0 \\ \Leftrightarrow Av_j &= \sum_{i=1}^{j+1} v_i h_{i,j} \\ \Leftrightarrow Av_j &= \sum_{i=1}^j h_{i,j} v_i + h_{j+1,j} v_{j+1} \end{aligned}$$

which corresponds to steps 3, 4 and 5 of the algorithm.

For $j = m$

$$\begin{aligned} AV_m(:, m) &= V_m H(:, m) + w_m \times 1 \\ \Leftrightarrow Av_m &= \sum_{i=1}^m v_i h_{i,m} + w_m \\ \Leftrightarrow w_m &= Av_m - \sum_{i=1}^m h_{i,m} v_i \end{aligned}$$

which corresponds to step 4 of the algorithm.



Proof.

To show equality (10), we first notice the structure of \bar{H}_m

$$\begin{pmatrix} H_m \\ h_{m+1,m}e_m^T \end{pmatrix}$$

Equality (9) writes

$$\begin{aligned} AV_m &= V_m H_m + h_{m+1,m} v_{m+1} e_m^T \\ &= V_m H_m + v_{m+1} (h_{m+1,m} e_m^T) \\ &= \begin{pmatrix} V_m v_{m+1} \end{pmatrix} \begin{pmatrix} H_m \\ h_{m+1,m} e_m^T \end{pmatrix} \\ &= v_{m+1} \bar{H}_m \end{aligned}$$



Proof.

Equality (11):

$$V_m^T A V_m = H_m$$

V_m is an orthonormal matrix: $V_m^T V_m = I_m$ (identity in \mathbb{R}^m).

By construction v_{m+1} is orthogonal to all the previous vectors:

$$V_m^T v_{m+1} = 0.$$

Multiplying Equation (9) by V_m^T on the left gives

$$V_m^T A V_m = \underbrace{V_m^T V_m}_{I_m} H_m + h_{m+1,m} \underbrace{V_m^T v_{m+1}}_{O_m} e_m^T$$



When $x_0 \neq 0$

- we assumed $x_0 = 0$ and we showed that we seek x_m in the affine subspace $\mathcal{K}_m(A, b)$.
- if $x_0 \neq 0$, then we can show that we seek x_m in the affine subspace $x_0 + \mathcal{K}_m(A, r_0)$ where $r_0 = b - Ax_0$.

Using Arnoldi procedure to solve a linear system

From an initial guess x_0 , we search $x_m \in x_0 + \mathcal{K}_m = x_0 + \mathcal{K}_m(A, r_0)$.

How to choose x_m ?

- Galerkin approach: $b - Ax_m \perp \mathcal{K}_m$

\Rightarrow FOM Solver (Full Orthogonalization Method)

- Minimum Norm Residual approach: $\|b - Ax_m\|_2$ minimal

\Rightarrow GMRES Solver (Generalised Minimal RESidual)

Iterative Methods: Krylov Methods

Galerkin Approach – FOM

- $x_m \in x_0 + \mathcal{K}_m$ means $\exists y_m \in \mathbb{R}^m$ such that $x_m = x_0 + V_m y_m$

Exercise

How y_m can be computed to verify $b - Ax_m \perp \mathcal{K}_m$?

```
1 Set the initial guess  $x_0$ 
2  $r_0 = b - Ax_0$ ;  $\beta = \|r_0\|$ ;  $v_1 = r_0/\beta$ ;
3 for  $j = 1, 2, \dots, m$  do
4      $w_j = Av_j$ ;
5     for  $i = 1, \dots, j$  do
6          $h_{i,j} = v_i^T w_j$ ;
7          $w_j = w_j - h_{i,j}v_i$ ;
8     end
9      $h_{j+1,j} = \|w_j\|$ ;
10    if  $h_{j+1,j} = 0$  then
11         $m = j$ ;
12        goto 16
13    end
14     $v_{j+1} = w_j/h_{j+1,j}$ 
15 end
16  $y_m = H_m^{-1}(\beta e_1)$ 
17  $x_m = x_0 + V_m y_m$ 
```

Algorithm 8: FOM - MGS variant - Y.Saad - Algo 6.4

steps 4 to 8 : Modified Gram-Schmidt (MGS)

Some remarks on this algorithm

- with this algorithm presented this way, we compute $x_m \in x_0 + \mathcal{K}_m$ for a given m .
- there is no guaranty that x_m is close to the exact solution ($\eta_{A,b}^N(\tilde{x}_m) < \varepsilon?$ or $\eta_b^N(\tilde{x}_m) < \varepsilon?$).
- we have to re-formulate the algorithm in order to stop when the wanted precision is achieved (or when we have reach a maximum number of iterations).

Exercise

re-write the algorithm

Improvements of FOM algorithm

Question: is it necessary to compute x_j and r_j at each iteration?
(computational cost: 2 matrix×vector products)

Improvements of FOM algorithm

Question: is it necessary to compute x_j and r_j at each iteration?
(computational cost: 2 matrix×vector products)

- a priori, yes: $\|r_j\|$ is present in the stopping criteria

Improvements of FOM algorithm

Question: is it necessary to compute x_j and r_j at each iteration?
(computational cost: 2 matrix×vector products)

- a priori, yes: $\|r_j\|$ is present in the stopping criteria
- we show that

$$b - Ax_j = -h_{j+1,j}e_j^T y_j v_{j+1}$$

and

$$\|b - Ax_j\| = h_{j+1,j}|e_j^T y_j|$$

Improvements of FOM algorithm

Question: is it necessary to compute x_j and r_j at each iteration?
(computational cost: 2 matrix×vector products)

- a priori, yes: $\|r_j\|$ is present in the stopping criteria
- we show that

$$b - Ax_j = -h_{j+1,j}e_j^T y_j v_{j+1}$$

and

$$\|b - Ax_j\| = h_{j+1,j}|e_j^T y_j|$$

Exercise

Prove these two relations

Improvements of FOM algorithm

Question: is it necessary to compute x_j and r_j at each iteration?
(computational cost: 2 matrix×vector products)

- a priori, yes: $\|r_j\|$ is present in the stopping criteria
- we show that

$$b - Ax_j = -h_{j+1,j}e_j^T y_j v_{j+1}$$

and

$$\|b - Ax_j\| = h_{j+1,j}|e_j^T y_j|$$

Exercise

Prove these two relations

⇒

we don't have to compute x_j and r_j at each iteration. We will compute these values once we have converged (or we estimate that we have converged).

Iterative Methods: Krylov Methods

Minimum Norm Residual approach –
GMRES

Minimum Norm Residual approach

- $x_m \in x_0 + \mathcal{K}_m$ means $\exists y_m \in \mathbb{R}^m$ such that $x_m = x_0 + V_m y_m$
- **Question:**
How y_m can be computed such that $\|b - Ax_m\|_2$ is minimal?

Exercise

Prove that $\|b - Ax_m\|_2 = \|\beta e_1 - \bar{H}_m y_m\|_2$

The GMRES iterate is the vector of $x_0 + \mathcal{K}_m$ such that

$$\begin{aligned}x_m &= x_0 + V_m y_m \\ y_m &= \arg \min_{y \in \mathbb{R}^m} \|\beta e_1 - \bar{H}_m y\|\end{aligned}$$

GMRES algorithm

```
1 Set the initial guess  $x_0$ 
2  $r_0 = b - Ax_0$ ;  $\beta = \|r_0\|$ ;  $v_1 = r_0/\beta$ ;
3 for  $j = 1, 2, \dots, m$  do
4      $w_j = Av_j$ ;
5     for  $i = 1, \dots, j$  do
6          $h_{i,j} = v_i^T w_j$ ;
7          $w_j = w_j - h_{i,j}v_i$ ;
8     end
9      $h_{j+1,j} = \|w_j\|$ ;
10    if  $h_{j+1,j} = 0$  then
11         $m = j$ ;
12        goto 16
13    end
14     $v_{j+1} = w_j/h_{j+1,j}$ 
15 end
16 Solve the least-squares problem  $y_m = \arg \min \|\beta e_1 - \bar{H}_m y\|$ 
17  $x_m = x_0 + V_m y_m$ 
```

Algorithm 9: GMRES - MGS variant - Y.Saad - Algo 6.9

Solution of the linear least-square problem

A stable solution technique to solve $\|\beta e_1 - \bar{H}_m y\|_2$ is to use Givens rotations to transform the Hessenberg matrix into upper triangular form.

$$Q_i = \begin{pmatrix} 1 & & & & & \\ & \dots & & & & \\ & & 1 & & & \\ & & & c_i & s_i & \\ & & & -s_i & c_i & \\ & & & & & 1 \\ & & & & & & \ddots \\ & & & & & & & 1 \end{pmatrix} \begin{array}{l} \leftarrow \text{row } i \\ \leftarrow \text{row } i + 1 \end{array}$$

with $c_i^2 + s_i^2 = 1$.

Computing $B = Q_i A$ copies all the rows but the i^{th} and $(i+1)^{\text{th}}$ from A in B and replaces those two rows by linear combinations of them such that $B_{i+1,i} = 0$ if $s_i = \frac{a_{i+1,i}}{\sqrt{a_{i,i}^2 + a_{i+1,i}^2}}$ and $c_i = \frac{a_{i,i}}{\sqrt{a_{i,i}^2 + a_{i+1,i}^2}}$.

Application of the first Givens rotation to \bar{H}_5

$$\text{Consider } \bar{H}_5 = \begin{pmatrix} h_{11} & h_{12} & h_{13} & h_{14} & h_{15} \\ h_{21} & h_{22} & h_{23} & h_{24} & h_{25} \\ & h_{32} & h_{33} & h_{34} & h_{35} \\ & & h_{43} & h_{44} & h_{45} \\ & & & h_{54} & h_{55} \\ & & & & h_{65} \end{pmatrix} \text{ and } \beta e_1 = \begin{pmatrix} \beta \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

Multiplying by the left by

$$Q_1 = \begin{pmatrix} c_1 & s_1 & & & \\ -s_1 & c_1 & & & \\ & & 1 & & \\ & & & 1 & \\ & & & & 1 \\ & & & & & 1 \end{pmatrix} \text{ with } \begin{cases} s_1 = \frac{h_{2,1}}{\sqrt{h_{1,1}^2 + h_{2,1}^2}} \\ c_1 = \frac{h_{1,1}}{\sqrt{h_{1,1}^2 + h_{2,1}^2}} \end{cases}.$$

Clearly $Q_1^T Q_1 = I$.

Result after the first rotation

$$\bar{H}_5^{(1)} = \begin{pmatrix} h_{11}^{(1)} & h_{12}^{(1)} & h_{13}^{(1)} & h_{14}^{(1)} & h_{15}^{(1)} \\ 0 & h_{22}^{(1)} & h_{23}^{(1)} & h_{24}^{(1)} & h_{25}^{(1)} \\ & h_{32} & h_{33} & h_{34} & h_{35} \\ & & h_{43} & h_{44} & h_{45} \\ & & & h_{54} & h_{55} \\ & & & & h_{65} \end{pmatrix} \quad \text{and} \quad \bar{g}_1^{(1)} = \begin{pmatrix} c_1\beta \\ -s_1\beta \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Result after 5 rotations

This elimination process is continued until the 5th rotation is applied which leads to

$$\bar{H}_5^{(5)} = \begin{pmatrix} h_{11}^{(1)} & h_{12}^{(1)} & h_{13}^{(1)} & h_{14}^{(1)} & h_{15}^{(1)} \\ 0 & h_{22}^{(2)} & h_{23}^{(2)} & h_{24}^{(2)} & h_{25}^{(2)} \\ & 0 & h_{33}^{(3)} & h_{34}^{(3)} & h_{35}^{(3)} \\ & & 0 & h_{44}^{(4)} & h_{45}^{(4)} \\ & & & 0 & h_{55}^{(5)} \\ & & & & 0 \end{pmatrix} \quad \text{and} \quad \bar{g}^{(5)} = \begin{pmatrix} \gamma_1^{(1)} \\ \gamma_2^{(2)} \\ \gamma_3^{(3)} \\ \gamma_4^{(4)} \\ \gamma_5^{(5)} \\ \gamma_6^{(5)} \end{pmatrix}$$

QR Factorisation of \bar{H}_m

Generally, the rotation Q_i is defined by

$$s_i = \frac{h_{i+1,i}}{\sqrt{\left(h_{i,i}^{(i-1)}\right)^2 + h_{i+1,i}^2}} \text{ and } c_i = \frac{h_{i,i}^{(i-1)}}{\sqrt{\left(h_{i,i}^{(i-1)}\right)^2 + h_{i+1,i}^2}}. \quad (12)$$

Define, the unitary matrix

$$Q_m = Q_m \dots Q_1 \in \mathbb{R}^{(m+1) \times (m+1)}$$

and let

$$\begin{aligned} \bar{R}_m &= \bar{H}_m^{(m)} = Q_m \bar{H}_m, \\ \bar{g}_m &= \beta Q_m e_1 = (\gamma_1^{(1)} \dots \gamma_m^{(m)} \gamma_{m+1}^{(m)})^T. \end{aligned}$$

Because Q_m is unitary we have

$$\min \|\beta e_1 - \bar{H}_m y\| = \min \|Q_m (\beta e_1 - \bar{H}_m y)\| = \min \|\bar{g}_m - \bar{R}_m y\|$$

- The solution of the least-squares problem is obtained by solving the triangular system

$$R_m y = g_m,$$

where $R_m \in \mathbb{R}^{m \times m}$ is obtained from \bar{R}_m by deleting its last row and similarly $g_m \in \mathbb{R}^m$ is obtained from \bar{g}_m by discarding its last component.

The residual of this least-squares problem is the last component of \bar{g}_m .

- QR factorisation in the general case
- particular case of \bar{H}_m

Some properties of GMRES

Proposition

1. *The rank of AV_m is equal to the rank of R_m .
In particular if $r_{mm} = 0$, then A must be singular.*
2. $y_m = \operatorname{argmin}_{y \in \mathbb{R}^m} \|\beta e_1 - \bar{H}_m y\|$ is given by

$$y_m = R_m^{-1} g_m.$$

3. *The residual vector is*

$$\begin{aligned} b - Ax_m = V_{m+1}(\beta e_1 - \bar{H}_m y_m) &= V_{m+1} Q_m^T (Q_m \beta e_1 - Q_m \bar{H}_m y_m) \\ &= V_{m+1} Q_m^T (\gamma_{m+1}^{(m)} e_{m+1}) \end{aligned}$$

then

$$\|b - Ax_m\| = |\gamma_{m+1}^{(m)}| \quad (13)$$

Incremental Factorisation of \bar{H}_m

The QR factorization of \bar{H}_{m+1} can be cheaply computed from the QR factorization of \bar{H}_m .

$$\bar{H}_{m+1} = \left(\begin{array}{c|c} \bar{H}_m & \begin{matrix} h_{1,m+1} \\ \vdots \\ h_{m+1,m+1} \end{matrix} \\ \hline 0 & h_{m+2,m+1} \end{array} \right)$$

It is enough to apply the m Givens rotations computed to factor \bar{H}_m to the **last column** of \bar{H}_{m+1} and build and apply a $(m+1)^{th}$ rotation to zero the **$h_{m+2,m+1}$** entry.

Incremental Factorisation of \bar{H}_m

$$\bar{H}_6^{(5)} = \left(\begin{array}{ccccc|c} h_{11}^{(1)} & h_{12}^{(1)} & h_{13}^{(1)} & h_{14}^{(1)} & h_{15}^{(1)} & h_{16}^{(1)} \\ 0 & h_{22}^{(2)} & h_{23}^{(2)} & h_{24}^{(2)} & h_{25}^{(2)} & h_{26}^{(2)} \\ & 0 & h_{33}^{(3)} & h_{34}^{(3)} & h_{35}^{(3)} & h_{36}^{(3)} \\ & & 0 & h_{44}^{(4)} & h_{45}^{(4)} & h_{46}^{(4)} \\ & & & 0 & h_{55}^{(5)} & h_{56}^{(5)} \\ & & & & 0 & h_{66} \\ & & & & 0 & h_{76} \end{array} \right) \text{ and } \bar{g}_6^{(5)} = \left(\begin{array}{c} \gamma_1^{(1)} \\ \gamma_2^{(2)} \\ \gamma_3^{(3)} \\ \gamma_4^{(4)} \\ \gamma_5^{(5)} \\ \gamma_6^{(5)} \\ 0 \end{array} \right)$$

We then apply both to $\bar{H}_6^{(5)}$ and $\bar{g}_5^{(6)}$ a 6th Givens rotation defined by

$$s_6 = \frac{h_{76}}{\sqrt{(h_{66}^{(5)})^2 + h_{76}^2}} \text{ and } c_6 = \frac{h_{66}^{(5)}}{\sqrt{(h_{66}^{(5)})^2 + h_{76}^2}}$$

to get \bar{R}_6 and \bar{g}_6 .

Incremental Factorisation of \bar{H}_m

In particular we have $\bar{g}_6 =$
$$\begin{pmatrix} \gamma_1^{(1)} \\ \gamma_2^{(2)} \\ \gamma_3^{(3)} \\ \gamma_4^{(4)} \\ \gamma_5^{(5)} \\ c_6 \gamma_6^{(5)} \\ -s_6 \gamma_6^{(5)} = \gamma_7^{(6)} \end{pmatrix}$$
 with

$$\|b - Ax_6\| = |\gamma_7^{(6)}| = |-s_6 \gamma_6^{(5)}| = |s_6| |\gamma_6^{(5)}| = |s_6| \|b - Ax_5\|.$$

By induction it can be shown that

$$\|b - Ax_{m+1}\| = |s_m| \|b - Ax_m\|.$$

Remark:

1. The norm of the residual is monotonically decreasing,
2. If $s_m = 0$, the solution is found at step m .

"Happy Breakdown" of GMRES

It exists one possible breakdown in the GMRES algorithm if $h_{k+1,k} = 0$ which prevents to increase the dimension of the search space.

Proposition

Let A be a nonsingular matrix. Then the GMRES algorithm breaks down at step k (i.e. $h_{k+1,k} = 0$) iff the iterate x_k is the exact solution of $Ax = b$.

Iterative Methods: Krylov Methods

Unsymmetric Krylov solver based on
Arnoldi procedure – the costs

Computational cost and memory storage of FOM and GMRES

- n , dimension of the problem,
- $nnz(A)$, number of non-zeros of A ,
- m , size of the Krylov subspace where we found a solution x_m that suits us.

FOM Algorithm

```
1  Set the initial guess  $x_0$ 
2   $r_0 = b - Ax_0$ ;  $\beta = \|r_0\|$ ;  $v_1 = r_0 / \beta$ ;
3  for  $j = 1, 2, \dots, m$  do
4       $w_j = Av_j$ ;
5      for  $i = 1, \dots, j$  do
6           $h_{i,j} = v_i^T w_j$ ;
7           $w_j = w_j - h_{i,j}v_i$ ;
8      end
9       $h_{j+1,j} = \|w_j\|$ ;
10     if  $h_{j+1,j} = 0$  then
11          $m = j$ ;
12         goto 16
13     end
14      $v_{j+1} = w_j / h_{j+1,j}$ 
15 end
16  $y_m = H_m^{-1}(\beta e_1)$ 
17  $x_m = x_0 + V_m y_m$ 
```

Algorithm 10: FOM - MGS variant - Y.Saad - Algo 6.4

FOM:

FLoating Point OPerations (Flops)

- m matrix \times vector products: $\simeq 2m \times nnz(A)$ operations
- iteration j : $\simeq 4 \times j \times n$ operations (GS : ddot : $\simeq 2n$, daxpy : $\simeq 2n$)
 $\Rightarrow \simeq 2m^2n$ for m iterations
- Total : $\simeq m(2 \times nnz(A) + 2mn)$
+ solution $H_m y = \beta e_1$ and matrix \times vector product $V_m \cdot y_m$

Computational cost and memory storage of FOM and GMRES

FOM:

FLoating Point Operations (Flops)

- m matrix \times vector products: $\simeq 2m \times nnz(A)$ operations
- iteration j : $\simeq 4 \times j \times n$ operations (GS : ddot : $\simeq 2n$, daxpy : $\simeq 2n$)
 $\Rightarrow \simeq 2m^2n$ for m iterations
- Total : $\simeq m(2 \times nnz(A) + 2mn)$
+ solution $H_m y = \beta e_1$ and matrix \times vector product $V_m \cdot y_m$

Memory storage

- base V_m , vectors x_0, x_m, w_j , matrix \bar{H}_m
- Total : $\simeq (m + 3)n + \frac{m^2}{2}$

Computational cost and memory storage very similar for **GMRES**.

Goal: Reduce the computational cost and/or the memory storage

- restarted versions: $\text{FOM}(\mathbf{m})$ and $\text{GMRES}(\mathbf{m})$
- truncated versions: $\text{DIOM}(\mathbf{m})$ and $\text{DQGMRES}(\mathbf{m})$

Restarted versions

- the idea of these versions is to fix a maximum size for the Krylov subspace m ,
- if the solution x_m in this subspace does not suit us, we start again and build a new subspace taking the solution x_m as the initial guess x_0 ,
- and this as many times it is necessary to obtain a solution that verifies the convergence criteria.
- GMRES(m) algorithm

```
1 Set the initial guess  $x_0$ 
2 Call GMRES with stopping criteria ( $\eta_b^N(x_k) \leq \varepsilon$  or  $nbit = m$ )
3 if  $\eta_b^N(x_k) > \varepsilon$  then
4   |  $x_0 = x_m$ 
5   | goto 2
6 end
```

- m , maximum size is a parameter of this algorithm and depends of constraints of the target computer.

- the idea of these versions is to maintain orthogonality only among the last m "Arnoldi's" vectors
- that leads to
 - computational gains during orthogonalisation procedure,
 - sparse structure de \bar{H}_m that can be exploited
 - incremental factorisation **LU** for DIOM(m) ,
 - simplified incremental factorisation **QR** for DQGMRES(m) ,
 - memory storage gain of the basis V_m (we can exhibit a short recurrence between x_{m+1} and x_m) [see the idea when describing symmetric solvers].

Iterative Methods: Symmetric Krylov Solvers

Arnoldi for symmetric problems: the Lanczos method

Proposition

Assume that the Arnoldi's method is applied to a real symmetric matrix A . The matrix H_m is tridiagonal and symmetric.

Exercise

Prove the Proposition

The standard notation used to describe the Lanczos algorithm is to use T_m rather than H_m and $\alpha_j \equiv h_{j,j}$ and $\beta_j \equiv h_{j-1,j} = h_{j,j-1}$.

$$T_m = \begin{pmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_{m-1} & \alpha_{m-1} & \beta_m \\ & & & \beta_m & \alpha_m \end{pmatrix}.$$



Cornelius Lanczos

1893-1974: Hungary.

Lanczos worked on relativity and mathematical physics and invented what is now called the Fast Fourier Transform. He worked with Einstein in Berlin, for the Boeing Aircraft Company, at Purdue University and the Department at the Dublin Institute for Advance Study in Ireland.

Lanczos algorithm

- 1: Choose an initial vector v_1 of norm equal to one.
- 2: $\beta_1 = 0$ and $v_0 = 0$
- 3: **for** $j = 1, 2, \dots, m$ **do**
- 4: $w_j = Av_j - \beta_j v_{j-1}$
- 5: $\alpha_j = w_j^T v_j$
- 6: $w_j = w_j - \alpha_j v_j$
- 7: $\beta_{j+1} = \|w_j\|$
- 8: **If** $\beta_{j+1}=0$ **then** Stop.
- 9: $v_{j+1} = w_j / \beta_{j+1}$
- 10: **end for**

The following “Arnoldi” relations read:

$$AV_m = V_m T_m + \beta_{m+1} v_{m+1} e_m^T, \quad (14)$$

$$V_m^T AV_m = T_m \quad (15)$$

Arnoldi relations for a symmetric problem

Arnoldi relations:

$$\begin{aligned}AV_m &= V_m H_m + h_{m+1,m} v_{m+1} e_m^T \\AV_m &= V_{m+1} \bar{H}_m \\V_m^T AV_m &= H_m\end{aligned}$$

Arnoldi relations when the matrix is symmetric:

$$\begin{aligned}AV_m &= V_m T_m + \beta_{m+1} v_{m+1} e_m^T, \\V_m^T AV_m &= T_m\end{aligned}$$

From Arnoldi procedure to Lanczos algorithm

```
1:  $v_1 = b/\|b\|$ ;  
2: for  $j = 1, 2, \dots, m$  do  
3:    $w_j = Av_j$ ;  
4:   for  $i = 1, \dots, j$  do  
5:      $h_{i,j} = v_i^T w_j$ ;  
6:      $w_j = w_j - h_{i,j}v_i$ ;  
7:   end for  
8:   Compute  $h_{j+1,j} = \|w_j\|$ ;  
9:   exit if ( $h_{j+1,j} = 0$ );  
10:  Compute  $v_{j+1} = w_j/h_{j+1,j}$   
11: end for
```

Algorithm 11: Arnoldi
(MGS)

```
1: Choose an initial vector  $v_1$  of  
   norm equal to one.  
2:  $\beta_1 = 0$  and  $v_0 = 0$   
3: for  $j = 1, 2, \dots, m$  do  
4:    $w_j = Av_j - \beta_j v_{j-1}$   
5:    $\alpha_j = w_j^T v_j$   
6:    $w_j = w_j - \alpha_j v_j$   
7:    $\beta_{j+1} = \|w_j\|$   
8:   If  $\beta_{j+1} = 0$  then Stop.  
9:    $v_{j+1} = w_j/\beta_{j+1}$   
10: end for
```

Algorithm 12: Lanczos

Iterative Methods: Symmetric Krylov Solvers

Lanczos algorithm for symmetric linear
systems

Lanczos algorithm for symmetric linear systems

- Galerkin approach:
search $x_m \in x_0 + \mathcal{K}_m$ such that $b - Ax_m \perp \mathcal{K}_m$

Exercise

how express x_m ?

Lanczos algorithm for symmetric linear systems

- 1: $r_0 = b - Ax_0$; $\beta = \|r_0\|$; $v_1 = r_0/\beta$
- 2: **for** $j = 1, 2, \dots, m$ **do**
- 3: $w_j = Av_j - \beta_j v_{j-1}$ (If $j = 1$ set $\beta_1 v_0 = 0$)
- 4: $\alpha_j = w_j^T v_j$
- 5: $w_j = w_j - \alpha_j v_j$
- 6: $\beta_{j+1} = \|w_j\|$
- 7: **if** $\beta_{j+1} = 0$ **then**
- 8: $m = j$; goto 12
- 9: **end if**
- 10: $v_{j+1} = w_j/\beta_{j+1}$
- 11: **end for**
- 12: Set $T_m = \text{tridiag}(\beta_i, \alpha_i, \beta_{i+1})$ and $V_m = (v_1, \dots, v_m)$.
- 13: Compute $y_m = T_m^{-1}(\beta e_1)$ and $x_m = x_0 + V_m y_m$.

Exercise

Show that the residual associated with x_m is colinear to v_{m+1} ?

We have (third time?)

$$\begin{aligned}b - Ax_m &= b - A(x_0 + V_m y_m) \\&= r_0 - AV_m y_m \\&= V_m(\beta e_1) - (V_m T_m + \beta_{m+1} v_{m+1} e_m^T) y_m \text{ (from (14))} \\&= V_m \underbrace{(\beta e_1 - T_m y_m)}_0 - \beta_{m+1} v_{m+1} e_m^T y_m.\end{aligned}$$

1. T_m , 2 vectors
2. To compute v_{j+1} , we need v_{j-1} and v_j , 3 vectors?
=> NO, we still need the base V_m to compute x_m

Iterative Methods: Symmetric Krylov Solvers

The D-Lanczos variant

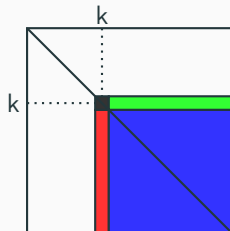
As T_m is tridiagonal, we can compute an incremental factorisation **L.U** of T_m and we will show that:

- factors can be updated at each Lanczos iteration,
- we do not have to store all the basis V_m to compute x_m .

Rappel: factorisation $L.U$

Factorisation $L.U$:

For k from 1 to $n - 1$:



1. Compute column k of L :

$$A(k+1:n, k) = A(k+1:n, k) / A(k, k)$$

2. Update the sub-block:

$$\begin{aligned} &A(k+1:n, k+1:n) \\ &= A(k+1:n, k+1:n) \\ &\quad - A(k+1:n, k) \times A(k, k+1:n) \end{aligned}$$

Factorisation $L.U$ of $T_m = L_m \times U_m$

Exemple : $T_5 = L_5 \times U_5$

$$T_5 = \begin{pmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & \\ & \beta_3 & \alpha_3 & \beta_4 & \\ & & \beta_4 & \alpha_4 & \beta_5 \\ & & & \beta_5 & \alpha_5 \end{pmatrix}$$

Factorisation $L.U$ of $T_m = L_m \times U_m$

We show that

- L and U are bi-diagonal matrices
- the coefficients of the upper diagonal of U_m are the coefficients β_i of T_m

$$T_5 = \begin{pmatrix} 1 & & & & \\ \lambda_2 & 1 & & & \\ & \lambda_3 & 1 & & \\ & & \lambda_4 & 1 & \\ & & & \lambda_5 & 1 \end{pmatrix} \times \begin{pmatrix} \eta_1 & \beta_2 & & & \\ & \eta_2 & \beta_3 & & \\ & & \eta_3 & \beta_4 & \\ & & & \eta_4 & \beta_5 \\ & & & & \eta_5 \end{pmatrix}$$

Factorisation LU of $T_m = L_m \times U_m$

Another way to represent the factors L and U in the same memory space A

$$\text{memory}(A) := \begin{pmatrix} \eta_1 & \beta_2 & & & \\ \lambda_2 & \eta_2 & \beta_3 & & \\ & \lambda_3 & \eta_3 & \beta_4 & \\ & & \lambda_4 & \eta_4 & \beta_5 \\ & & & \lambda_5 & \eta_5 \end{pmatrix}$$

(remember factorization L.U)

How compute coefficients of L_m and U_m

- Using a recurrence, we show that

$$\lambda_m = \frac{\beta_m}{\eta_{m-1}}, \quad m \in \{2, \dots\}$$

$$\eta_m = \alpha_m - \lambda_m \beta_m, \quad m \in \{1, \dots\}, \beta_1 = 0$$

The approximate solution is given by

$$x_m = x_0 + \underbrace{V_m U_m^{-1}}_{P_m} \underbrace{L_m^{-1}(\beta e_1)}_{z_m},$$

then

$$x_m = x_0 + P_m z_m.$$

Because of the structure of U_m , we can easily update P_m and compute its last column using the previous p_i 's and v_m .

Equating the last columns of $P_m U_m = V_m$ gives

$$\begin{aligned}\eta_m p_m + \beta_m p_{m-1} &= v_m \\ \Rightarrow p_m &= \eta_m^{-1} (v_m - \beta_m p_{m-1}).\end{aligned}$$

Expression of x_m

Similarly, we can derive an update for z_m exploiting the structure of L_m (bidiagonal matrix)

$$z_m = \begin{pmatrix} z_{m-1} \\ \xi_m \end{pmatrix},$$

where $\xi_m = -\lambda_m \xi_{m-1}$.

Consequently we have

$$x_m = x_0 + (P_{m-1}p_m) \begin{pmatrix} z_{m-1} \\ \xi_m \end{pmatrix} = \underbrace{x_0 + P_{m-1}z_{m-1}}_{x_{m-1}} + \xi_m p_m$$

- Short Relation between x_m et x_{m-1} : $x_m = x_{m-1} + \xi_m p_m$
 \Rightarrow no more storage of V_m
- Memory usage : a vector for p_m and some scalars

D-Lanczos algorithm

- 1: Set $r_0 = b - Ax_0$; $\xi_1 = \beta = \|r_0\|$ and $v_1 = r_0/\beta$
- 2: Set $\lambda_1 = \beta_1 = 0$, $p_0 = 0$
- 3: **for** $j = 1, \dots$ **do**
- 4: Compute $w_j = Av_j - \beta_j v_{j-1}$ and $\alpha_j = w_j^T v_j$
- 5: **if** $j > 1$ **then**
- 6: computed $\lambda_j = \frac{\beta_j}{\eta_{j-1}}$ and $\xi_j = -\lambda_j \xi_{j-1}$
- 7: **end if**
- 8: $\eta_j = \alpha_j - \lambda_j \beta_j$
- 9: $p_j = \eta_j^{-1} (v_j - \beta_j p_{j-1})$
- 10: $x_j = x_{j-1} + \xi_j p_j$
- 11: **if** converged **then**
- 12: Stop
- 13: **end if**
- 14: $w_j = w_j - \alpha_j v_j$
- 15: $\beta_{j+1} = \|w_j\|$, $v_{j+1} = w_j/\beta_{j+1}$
- 16: **end for**

Exercise

Show that: $\|b - Ax_m\| = \beta_{m+1} \left| \frac{\xi_m}{\eta_m} \right|$

Some important properties

Proposition

Let $r_m = b - Ax_m$ and p_m the vectors generated by the D-Lanczos algorithm. We have:

1. the residual vector r_m is colinear to v_{m+1} . As a consequence, the residual vectors are orthogonal to each other.
2. the vectors p_i are A -conjugate, that is: $\forall i \neq j, p_i^T A p_j = 0$.

Iterative Method: Conjugate Gradient

D-Lanczos algorithm

- 1: Set $r_0 = b - Ax_0$; $\xi_1 = \beta = \|r_0\|$ and $v_1 = r_0/\beta$
- 2: Set $\lambda_1 = \beta_1 = 0$, $p_0 = 0$
- 3: **for** $j = 1, \dots$ **do**
- 4: Compute $w_j = Av_j - \beta_j v_{j-1}$ and $\alpha_j = w_j^T v_j$
- 5: **if** $j > 1$ **then**
- 6: computed $\lambda_j = \frac{\beta_j}{\eta_{j-1}}$ and $\xi_j = -\lambda_j \xi_{j-1}$
- 7: **end if**
- 8: $\eta_j = \alpha_j - \lambda_j \beta_j$
- 9: $p_j = \eta_j^{-1} (v_j - \beta_j p_{j-1})$
- 10: $x_j = x_{j-1} + \xi_j p_j$ // new iterate function of the previous one
- 11: **if** converged **then**
- 12: Stop
- 13: **end if**
- 14: $w_j = w_j - \alpha_j v_j$
- 15: $\beta_{j+1} = \|w_j\|$, $v_{j+1} = w_j/\beta_{j+1}$
- 16: **end for**

D-Lanczos algorithm

- 1: Set $r_0 = b - Ax_0$; $\xi_1 = \beta = \|r_0\|$ and $v_1 = r_0/\beta$
- 2: Set $\lambda_1 = \beta_1 = 0$, $p_0 = 0$
- 3: **for** $j = 1, \dots$ **do**
- 4: Compute $w_j = Av_j - \beta_j v_{j-1}$ and $\alpha_j = w_j^T v_j$
- 5: **if** $j > 1$ **then**
- 6: computed $\lambda_j = \frac{\beta_j}{\eta_{j-1}}$ and $\xi_j = -\lambda_j \xi_{j-1}$
- 7: **end if**
- 8: $\eta_j = \alpha_j - \lambda_j \beta_j$
- 9: $p_j = \eta_j^{-1} (v_j - \beta_j p_{j-1})$ // we need to compute a new direction
- 10: $x_j = x_{j-1} + \xi_j p_j$ // new iterate function of the previous one
- 11: **if** converged **then**
- 12: Stop
- 13: **end if**
- 14: $w_j = w_j - \alpha_j v_j$
- 15: $\beta_{j+1} = \|w_j\|$, $v_{j+1} = w_j/\beta_{j+1}$
- 16: **end for**

D-Lanczos algorithm

- 1: Set $r_0 = b - Ax_0$; $\xi_1 = \beta = \|r_0\|$ and $v_1 = r_0/\beta$
- 2: Set $\lambda_1 = \beta_1 = 0$, $p_0 = 0$
- 3: **for** $j = 1, \dots$ **do**
- 4: Compute $w_j = Av_j - \beta_j v_{j-1}$ and $\alpha_j = w_j^T v_j$
- 5: **if** $j > 1$ **then**
- 6: computed $\lambda_j = \frac{\beta_j}{\eta_{j-1}}$ and $\xi_j = -\lambda_j \xi_{j-1}$
- 7: **end if**
- 8: $\eta_j = \alpha_j - \lambda_j \beta_j$
- 9: $p_j = \eta_j^{-1}(\textcolor{red}{cr}_{j-1} - \beta_j p_{j-1})$ // r_{j-1} and v_j are colinears
- 10: $x_j = x_{j-1} + \xi_j p_j$
- 11: **if** converged **then**
- 12: Stop
- 13: **end if**
- 14: $w_j = w_j - \alpha_j v_j$
- 15: $\beta_{j+1} = \|w_j\|$, $v_{j+1} = w_j/\beta_{j+1}$
- 16: **end for**

Conjugate Gradient can be derived from D-Lanczos

With a translation of p_j index to conform with standard notation, we express [step 10](#) of the algorithm:

$$x_{j+1} = x_j + \delta_j p_j \quad (16)$$

and as consequence

$$r_{j+1} = r_j - \delta_j A p_j. \quad (17)$$

Thus a first expression of δ_j

$$\delta_j = \frac{r_j^T r_j}{r_j^T A p_j}. \quad (18)$$

Conjugate Gradient can be derived from D-Lanczos (cont.)

We are looking for a new direction p_{j+1} as a linear combination of r_{j+1} et p_j (step 9 with translation of p_j index)

$$p_{j+1} = r_{j+1} + \gamma_j p_j, \quad (19)$$

We show that we can re-write δ_j

$$\delta_j = \frac{r_j^T r_j}{p_j^T A p_j}. \quad (20)$$

Then a first expression for γ_j is

$$\gamma_j = -\frac{r_{j+1}^T A p_j}{p_j^T A p_j}, \quad (21)$$

that can be eventually written as

$$\gamma_j = \frac{r_{j+1}^T r_{j+1}}{r_j^T r_j}. \quad (22)$$

Conjugate Gradient Algorithm

To use the standard notation, let's rename $\alpha_j = \delta_j$ and $\beta_j = \gamma_j$ and summarize the two previous slides:

1: $r_0 = b - Ax_0$, $p_0 = r_0$.

2: **for** $j = 0, 1, \dots$ **do**

3: $\alpha_j = (r_j^T r_j) / (p_j^T A p_j)$

4: $x_{j+1} = x_j + \alpha_j p_j$

5: $r_{j+1} = r_j - \alpha_j A p_j$

6: $\beta_j = (r_{j+1}^T r_{j+1}) / (r_j^T r_j)$

7: $p_{j+1} = r_{j+1} + \beta_j p_j$

8: **end for**

- Conjugate Gradient is a Krylov solver!
- Beware: α_j and β_j of GC have nothing to do with those of D-Lanczos algorithm.
- Memory usage: 4 vectors (x, p, Ap, r) .vs. 5 for D-Lanczos (v_m, v_{m-1}, w, p, x) .

Some important remarks

The algorithm relies on a LU factorization without pivoting of a symmetric tridiagonal matrix. This factorization might not exist and CG can break-down.

1. For symmetric positive definite (SPD) matrices, T_m is always non singular and a stable LU decomposition exists. **CG** is then the **method of choice** for this class of matrices.
2. For symmetric indefinite matrices a technique based on a **LQ** decomposition of T_m exists that is known as **SYMMLQ**.
3. For symmetric indefinite matrices, the solution of the linear system involving T_m can be replaced by the solution of a linear least-square involving \bar{T}_m . This method is known as **MINRES** and belong to the minimum norm error class of Krylov solvers.

Conjugate Gradient properties for SPD case (1)

Proposition

Because A is SPD, the bilinear form $x^T A y$ define an inner product.

The Galerkin condition $b - Ax_m \perp \mathcal{K}_m$ can be written $A(x_m - x^) \perp \mathcal{K}_m$ which also reads $(x_m - x^*) \perp_A \mathcal{K}_m$.*

This latter condition implies that

$$\|x_m - x^*\|_A$$

is minimal over \mathcal{K}_m

Conjugate Gradient properties for SPD case (2)

We have the following upper-bound for the convergence rate of CG in the SPD case.

Proposition

Let x_m be the m^{th} iterate generated by the CG algorithm then

$$\|x_m - x^*\|_A \leq 2 \cdot \left(\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1} \right)^m \|x_0 - x^*\|_A.$$

Iterative Methods: Algebraic preconditioning techniques

Iterative Methods: Algebraic preconditioning techniques

Some backgrounds

Some properties of Krylov solvers

- CG

$$\|x_m - x^*\|_A = \min_{p \in \mathbb{P}_m, p(0)=1} \|p(A)(x_0 - x^*)\|_A$$

- GMRES, MINRES

$$\|r_m\| = \min_{p \in \mathbb{P}_m, p(0)=1} \|p(A)(r_0)\|_2$$

- Bound on the rate of convergence of CG:

$$\|x_m - x^*\|_A \leq 2 \cdot \left(\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1} \right)^m \|x_0 - x^*\|_A.$$

Some properties of Krylov solvers

- Assumption: if A diagonalizable with m distinct eigenvalues then CG, GMRES, MINRES converges in at most m steps (Cayley-Hamilton theorem - minimal polynomial).
- Assumption: if r_0 has k components in the eigenbasis then CG, GMRES, MINRES converges in k steps.

Iterative Methods: Algebraic preconditioning techniques

Driving principles

Driving principles to design preconditioners

Find a non-singular matrix M such that $M.A$ has "better" properties v.s. the convergence behaviour of the selected Krylov solver:

- $M.A$ has less distinct eigenvalues
- $M.A \approx I$ in some sense

The preconditioner should

- be cheap to compute and to store,
- be cheap to apply,
- ensure a fast convergence.

With a good preconditioner the solution time for the preconditioned system should be significantly less than for the unpreconditioned system.

Iterative Methods: Algebraic preconditioning techniques

Preconditioned CG

The particular case of CG

- for CG, M must be symmetric (to keep a symmetric linear system)
- M be given in a factorized form CC^T
- CG can be applied to $\tilde{A}\tilde{x} = \tilde{b}$
 $\tilde{A} = C^TAC$, $C\tilde{x} = x$ and $\tilde{b} = C^Tb$ (Saad 9.4 with different notations).

Notations

- Let $x_k = C\tilde{x}_k$; $p_k = C\tilde{p}_k$; $\tilde{r}_k = C^T r_k$; $z_k = CC^T r_k = Mr_k$
- Conjugate Gradient algorithm for preconditioned system:

```
1:  $\tilde{r}_0 = \tilde{b} - \tilde{A}\tilde{x}_0$ ,  $\tilde{p}_0 = \tilde{r}_0$ .  
2: for  $k = 0, 1, \dots$  do  
3:    $\alpha_k = (\tilde{r}_k^T \tilde{r}_k) / (\tilde{p}_k^T \tilde{A}\tilde{p}_k)$   
4:    $\tilde{x}_{k+1} = \tilde{x}_k + \alpha_k \tilde{p}_k$   
5:    $\tilde{r}_{k+1} = \tilde{r}_k - \alpha_k \tilde{A}\tilde{p}_k$   
6:    $\beta_k = (\tilde{r}_{k+1}^T \tilde{r}_{k+1}) / (\tilde{r}_k^T \tilde{r}_k)$   
7:    $\tilde{p}_{k+1} = \tilde{r}_{k+1} + \beta_k \tilde{p}_k$   
8:   if Convergence Stop  
9: end for
```

- Let's re-arrange the algorithm \Rightarrow PCG algorithm

Writing the algorithm only using the unpreconditioned variables leads to:

Preconditioned Conjugate Gradient algorithm

1. Compute $r_0 = b - Ax_0$, $z_0 = Mr_0$ and $p_0 = z_0$
2. For $k=0,2, \dots$ Do
3. $\alpha_k = r_k^T z_k / p_k^T A p_k$
4. $x_{k+1} = x_k + \alpha_k p_k$
5. $r_{k+1} = r_k - \alpha_k A p_k$
6. $z_{k+1} = M r_{k+1}$
7. $\beta_k = r_{k+1}^T z_{k+1} / r_k^T z_k$
8. $p_{k+1} = z_{k+1} + \beta_k p_k$
9. if x_k accurate enough then stop
10. EndDo

Exercise

1. *A*-norm minimization

- *A*-norm of preconditioned system $\|\tilde{x}_k - \tilde{x}^*\|_{\tilde{A}}$ is minimal over $\mathcal{K}(\tilde{A}, \tilde{b}, k)$
- $\|x_k - x^*\|_A$?

2. Orthogonality

- p_k ?
- r_k ?

Iterative Methods: Algebraic preconditioning techniques

Preconditioner taxonomy and examples

Preconditioner taxonomy and examples

There are two main classes of preconditioners

- **Implicit preconditioners:**

approximate A with a matrix M such that solving the linear system $Mz = r$ is easy.

- **Explicit preconditioners:**

approximate A^{-1} with a matrix M and just perform $z = Mr$.

The governing ideas in the design of the preconditioners are very similar to those followed to define iterative stationary schemes. Consequently, all the stationary methods can be used to define preconditioners.

Stationary methods

Let x_0 be given and $M \in \mathbb{R}^{n \times n}$ a nonsingular matrix, compute

$$x_k = x_{k-1} + M(b - Ax_{k-1}).$$

Note that $b - Ax_{k-1} = A(x^* - x_{k-1}) \Rightarrow$ the best M is A^{-1} .

The stationary scheme converges to $x^* = A^{-1}b$ for any x_0 iff $\rho(I - MA) < 1$, where $\rho(\cdot)$ denotes the spectral radius.

Let $A = L + D + U$

- $M = I$: Richardson method,
- $M = D^{-1}$: Jacobi method,
- $M = (L + D)^{-1}$: Gauss-Seidel method.

Notice that M has always a special structure and the inverse must never be explicitly computed ($z = B^{-1}y$ reads *solve the linear system $Bz = y$*).

Several possibilities exist to solve $Ax = b$:

- Left preconditioner

$$MAx = Mb.$$

- Right preconditioner

$$AMy = b \text{ with } x = My.$$

- Split preconditioner if $M = M_1 M_2$

$$M_2 A M_1 y = M_2 b \text{ with } x = M_1 y.$$

Notice that the spectrum of MA , AM and $M_2 A M_1$ are identical (for any matrices B and C , the eigenvalues of BC are the same as those of CB)

Preconditioner location v.s. stopping criterion

The stopping criterion are based on backward error

$$\eta_{A,b}^N = \frac{\|b - Ax\|}{\|A\|\|x\| + \|b\|} < \varepsilon \text{ or } \eta_b^N = \frac{\|b - Ax\|}{\|b\|} < \varepsilon.$$

In PCG we can still compute η .

For GMRES, using a preconditioner means running GMRES on

Left precondition.	Right precondition.	Split precondition.
$MAx = Mb$	$AMy = b$	$M_2AM_1y = M_2b$

The free estimate of the residual norm of GMRES is associated with the preconditioned system.

Preconditioner location v.s. stopping criterion (cont)

	Left precondition.	Right precondition.	Split precondition.
$\eta_M(A, b)$	$\frac{\ MAx - Mb\ }{\ MA\ \ x\ + \ Mb\ }$	$\frac{\ AMy - b\ }{\ AM\ \ x\ + \ b\ }$	$\frac{\ M_2AM_1y - M_2b\ }{\ M_2AM_1\ \ y\ + \ M_2b\ }$
$\eta_M(b)$	$\frac{\ MAx - Mb\ }{\ Mb\ }$	$\frac{\ AMy - b\ }{\ b\ }$	$\frac{\ M_2AM_1y - M_2b\ }{\ M_2b\ }$

Using $\eta_M(b)$ for right preconditioned linear system will monitor the convergence as if no preconditioner was used.

Iterative Methods: Algebraic preconditioning techniques

Some classical algebraic preconditioners

Some classical algebraic preconditioner

- Incomplete factorization : IC , $ILU(p)$, $ILU(p, \tau)$
- SPAI (Sparse Approximate Inverse): compute the sparse approximate inverse by minimizing the Frobenius norm $\|MA - I\|_F$
- FSAI (Factorized Sparse Approximate inverse): compute the sparse approximate inverse of the Cholesky factor by minimizing the Frobenius norm $\|I - GL\|_F$
- AINV (Approximate Inverse): compute the sparse approximate inverse of the LDU or LDL^T factors using an incomplete biconjugation process

Incomplete factorizations

One variant of the LU factorization writes:

IKJ variant - Top looking variant

```
1.  for  $i = 2, \dots, n$  do  
2.    for  $k = 1, \dots, i - 1$  do  
3.       $a_{i,k} = a_{i,k} / a_{k,k}$   
4.    for  $j = k + 1, \dots, n$  do  
5.       $a_{i,j} = a_{i,j} - a_{i,k} * a_{k,j}$   
6.    end for  
7.  end for
```

Zero fill-in ILU - $ILU(0)$

Let denote $NZ(A)$ the set of (row,column) index of the nonzero entries of A .

ILU(0)	
1.	for $i = 2, \dots, n$ do
2.	for $k = 1, \dots, i - 1$ and $(i, k) \in NZ(A)$ do
3.	$a_{i,k} = a_{i,k} / a_{k,k}$
4.	for $j = k + 1, \dots, n$ and $(i, j) \in NZ(A)$ do
5.	$a_{i,j} = a_{i,j} - a_{i,k} * a_{k,j}$
6.	end for
7.	end for
8.	end for

Definition

The initial level of fill of an entry $a_{i,j}$ is defined by:

$$lev(i, j) = \begin{cases} 0 & \text{if } a_{i,j} \neq 0 \text{ or } i = j, \\ \infty & \text{otherwise.} \end{cases}$$

Each time this entry is modified in line 5 of the LU top looking algorithm, its level fill is updated by

$$lev(i, j) = \min\{lev(i, j), lev(i, k) + lev(k, j) + 1\}. \quad (23)$$

A first example

$$A = \begin{pmatrix} x & x & x & x \\ x & x & & \\ x & & x & \\ x & & & x \end{pmatrix} \quad lev = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & \infty & \infty \\ 0 & \infty & 0 & \infty \\ 0 & \infty & \infty & 0 \end{pmatrix} \rightarrow \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}.$$

ILU(p)

1. **for** all nonzero entries $a_{i,j}$ set $lev_{i,j} = 0$
2. **for** $i = 2, \dots, n$ **do**
3. **for** $k = 1, \dots, i - 1$ and $lev_{i,k} \leq p$ **do**
4. $a_{i,k} = a_{i,k} / a_{k,k}$
5. **for** $j = k + 1, \dots, n$ **do**
6. $a_{i,j} = a_{i,j} - a_{i,k} * a_{k,j}$
7. $lev(i, j) = \min\{lev(i, j), lev(i, k) + lev(k, j) + 1\}$
8. **if** $lev(i, j) > p$ **then** $a_{i,j} = 0$
9. **end for**
10. **end for**
11. **end for**

- Fix a drop tolerance τ and a number of fill p to be allowed in each row of the incomplete LU factors. At each step of the elimination process, drop all fill-ins that are smaller than τ times the 2-norm of the current row; for all the remaining ones keep only the p largest.
- Trade-off between amount of fill-in (construction time and application time for the preconditioner) and decrease of number of iterations.

Iterative Methods: Algebraic preconditioning techniques

Spectral preconditioners

Motivations

- The convergence of Krylov methods for solving the linear system often depends to a large extent on the eigenvalue distribution. In many cases, it is observed that “removing” the smallest eigenvalues can greatly improve the convergence
- Many preconditioners are able to cluster most of the eigenvalues close to one but still leave a few close to the origin
- “Moving” these eigenvalues by tuning the parameters that control these preconditioners is often difficult and might lead to very expensive preconditioners to set-up and to apply
- Performing a low rank update might enable to shift the smallest eigenvalues.

Sparse Direct Solvers - Ordering

A selection of references

- Books

- Duff, Erisman and Reid, Direct methods for Sparse Matrices, Clarendon Press, Oxford 1986.
- Dongarra, Duff, Sorensen and van der Vorst, Solving Linear Systems on Vector and Shared Memory Computers, SIAM, 1991.
- Davis, Direct methods for sparse linear systems, SIAM, 2006.
- George, Liu, and Ng, Computer Solution of Sparse Positive Definite Systems.
- Saad, Iterative methods for sparse linear systems, 2nd edition, SIAM, 2004.

- Articles

- Gilbert and Liu, Elimination structures for unsymmetric sparse LU factors, SIMAX, 1993.
- Liu, The role of elimination trees in sparse factorization, SIMAX, 1990.
- Heath, Ng and Peyton, Parallel Algorithms for Sparse Linear Systems, SIAM review 1991.

Sparse Direct Solvers - Ordering

Sparse matrices

Sparse matrices

Example:

$$\begin{array}{rclclcl} 3x_1 & + & 2x_2 & & = & 5 \\ & & 2x_2 & - & 5x_3 & = & 1 \\ 2x_1 & & & + & 3x_3 & = & 0 \end{array}$$

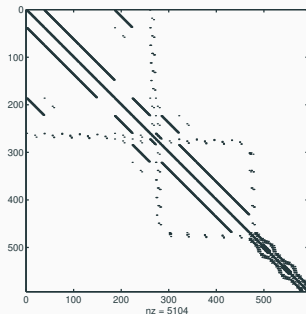
can be represented as

$$\mathbf{Ax} = \mathbf{b},$$

$$\text{where } \mathbf{A} = \begin{pmatrix} 3 & 2 & 0 \\ 0 & 2 & -5 \\ 2 & 0 & 3 \end{pmatrix}, \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}, \text{ and } \mathbf{b} = \begin{pmatrix} 5 \\ 1 \\ 0 \end{pmatrix}$$

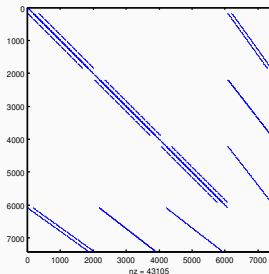
Sparse matrix: only nonzeros are stored.

Sparse matrix



Matrix dwt_592.rua ($n=592$, $nnz=5104$, 1.5%);
Structural analysis of a submarine

Matrix from Computational Fluid Dynamics;
(collaboration Univ. Tel Aviv)

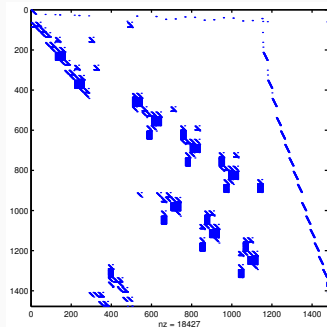


“Saddle-point” problem

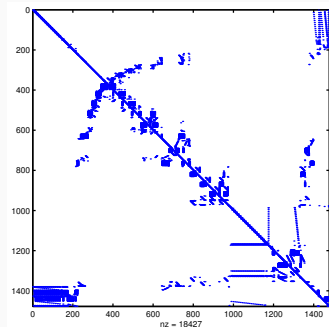
$$n \approx 7400, \text{ nnz} = 43105, 0.07\%$$

Preprocessing sparse matrices

Original ($A = \text{LHR01}$)



Preprocessed matrix ($A'(\text{LHR01})$)



Modified Problem: $A'x' = b'$ with $A' = P_n P D_r A D_c Q P^T Q_n$

Solution of $\mathbf{Ax} = \mathbf{b}$

- \mathbf{A} is unsymmetric :
 - \mathbf{A} is factorized as: $\mathbf{A} = \mathbf{LU}$, where \mathbf{L} is a lower triangular matrix, and \mathbf{U} is an upper triangular matrix.
 - Forward-backward substitution: $\mathbf{Ly} = \mathbf{b}$ then $\mathbf{Ux} = \mathbf{y}$
- \mathbf{A} is symmetric:
 - general $\mathbf{A} = \mathbf{LDL}^T$
 - positive definite $\mathbf{A} = \mathbf{LL}^T$

- Only non-zero values are stored
- Factors **L** and **U** have far more nonzeros than **A**
- Data structures are complex
- Computations are only a small portion of the code (the rest is data manipulation)
- Memory size is a limiting factor (\rightarrow *out-of-core solvers*)

Key numbers:

1- Small sizes :

- 500 MB matrix
- Factors = 5 GB
- Flops = 100 Gflops

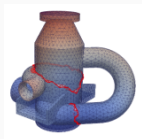
2- Example of 2D problem: Lab. Géosciences Azur, Valbonne

- Complex 2D finite difference matrix $n=16 \times 10^6$, 150×10^6 nonzeros
- Storage (single prec): 2 GB (12 GB with the factors)
- Flops: 10 Tflops

3- Example of 3D problem: EDF (Code_Aster, structural engineering)

- Real matrix finite elements $n = 10^6$, $nnz = 71 \times 10^6$ nonzeros
- Storage: 3.5×10^9 entries (28 GB) for factors, 35 GB total
- Flops: 21 Tflops

Sparse linear solvers: typical numbers



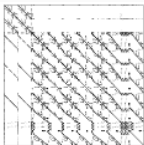
pump (credits:

Code_Aster)

Discretisation of physical problem

→ Solve $\mathbf{AX}=\mathbf{B}$ or $\mathbf{Ax}=\mathbf{b}$, \mathbf{A} large and sparse

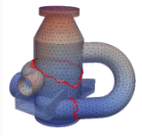
- Direct solution: factor $\mathbf{A}=\mathbf{LU}$ (or \mathbf{LDL}^T if symmetric) then $\mathbf{X}=\mathbf{U}^{-1}(\mathbf{L}^{-1}\mathbf{B})$
- Iterative solvers: $\mathbf{x}=\lim_k \mathbf{x}_k$



Typical numbers (illustration on the pump problem)

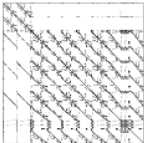
- \mathbf{A} : $n = 5.4 \times 10^6$ with $nnz = 2 \times 10^8$ nonzeros
- Target computer:
 - Peak performance: 518 Gflops/s (14 cores \times 37 Gflops/s/core)
 - Max memory bandwidth: $bw = 28$ GB/s

Sparse linear solvers: time analysis



pump (credits:

Code_Aster)



Discretisation of physical problem

→ Solve $\mathbf{AX}=\mathbf{B}$ or $\mathbf{Ax}=\mathbf{b}$, \mathbf{A} large and sparse

- \mathbf{A} : $n = 5.4 \times 10^6$ with $nnz = 2 \times 10^8$ nonzeros

Peak performance of computer: 518 Gflops/s (14 cores \times

37 Gflops/s/core)

Memory bandwidth: $bw = 28$ GB/s

Typical numbers (illustration on the pump problem)

- Factor $\mathbf{A} = \mathbf{LDL}^T \rightarrow \mathbf{L}$: 5×10^9 nonzeros (40 GB)! (fill-in)
- Direct solution: factor \mathbf{A} ($\mathbf{A} = \mathbf{LDL}^T$) \rightarrow 25 Tflops
Time(direct) ~ 175 s (Tera = 10^{12}); \rightarrow (143 Gflops/s)
- Iterative solution: key kernel is often \mathbf{Ay} : 5×10^8 flops
Min time $\mathbf{Ay} \sim 0.1$ s (7 Gflops/s)

$$\left(\frac{(2 \text{ flops per access to } \mathbf{A}) \times (bw \text{ in GB/s})}{(8 \text{ Bytes per element of } \mathbf{A})} \right)$$

$$\text{Time(iterative)} \sim 0.1 \text{ s} \times NbIter \times NbCol(\mathbf{B}) + t_{precond}$$

- Storage scheme depends on the pattern of the matrix and on the type of access required
 - band or variable-band matrices
 - “block bordered” or block tridiagonal matrices
 - general matrix
 - row, column or diagonal access

Data formats for a general sparse matrix \mathbf{A}

What needs to be represented

- Assembled matrices: $m \times n$ matrix \mathbf{A} with nnz nonzeros.
- Elemental matrices (unassembled): $m \times n$ matrix \mathbf{A} with NELT elements.
- Symmetric
→ store only part of the data.
- Distributed format ?
- Duplicate entries and/or out-of-range values ?

Classical Data Formats for Assembled Matrices

- Example of a 3x3 matrix with $nnz=5$ nonzeros

	1	2	3
1	a11		
2		a22	a23
3	a31		a33

- Coordinate format

IRN	[1 : nnz]	=	1	3	2	2	3
JCN	[1 : nnz]	=	1	1	2	3	3
VAL	[1 : nnz]	=	a_{11}	a_{31}	a_{22}	a_{23}	a_{33}

Classical Data Formats for Assembled Matrices

- Example of a 3x3 matrix with $nnz=5$ nonzeros

	1	2	3
1	a11		
2		a22	a23
3	a31		a33

- Coordinate format

IRN $[1 : nnz] = 1 \quad 3 \quad 2 \quad 2 \quad 3$

JCN $[1 : nnz] = 1 \quad 1 \quad 2 \quad 3 \quad 3$

VAL $[1 : nnz] = a_{11} \quad a_{31} \quad a_{22} \quad a_{23} \quad a_{33}$

- Compressed Sparse Column (CSC) format

IRN $[1 : nnz] = 1 \quad 3 \quad 2 \quad 2 \quad 3$

VAL $[1 : nnz] = a_{11} \quad a_{31} \quad a_{22} \quad a_{23} \quad a_{33}$

COLPTR $[1 : N + 1] = 1 \quad 3 \quad 4 \quad 6$

column J is stored in IRN/A locations COLPTR(J)...COLPTR($J+1$)-1

Classical Data Formats for Assembled Matrices

- Example of a 3x3 matrix with $nnz=5$ nonzeros

	1	2	3
1	a11		
2		a22	a23
3	a31		a33

- Coordinate format

IRN $[1 : nnz] = 1 \quad 3 \quad 2 \quad 2 \quad 3$
JCN $[1 : nnz] = 1 \quad 1 \quad 2 \quad 3 \quad 3$
VAL $[1 : nnz] = a_{11} \quad a_{31} \quad a_{22} \quad a_{23} \quad a_{33}$

- Compressed Sparse Column (CSC) format

IRN $[1 : nnz] = 1 \quad 3 \quad 2 \quad 2 \quad 3$
VAL $[1 : nnz] = a_{11} \quad a_{31} \quad a_{22} \quad a_{23} \quad a_{33}$
COLPTR $[1 : N + 1] = 1 \quad 3 \quad 4 \quad 6$
column J is stored in IRN/A locations COLPTR(J)...COLPTR(J+1)-1

- Compressed Sparse Row (CSR) format:

Similar to CSC, but row by row

Classical Data Formats for Assembled Matrices

- Example of a 3x3 matrix with nnz=5 nonzeros

	1	2	3
1	a11		
2		a22	a23
3	a31		a33

- Diagonal format (M=N):

NDIAG = 3

IDIAG = -2 0 1

$$\text{VAL} = \begin{bmatrix} na & a_{11} & 0 \\ na & a_{22} & a_{23} \\ a_{31} & a_{33} & na \end{bmatrix} \quad (na: \text{not accessed})$$

VAL(i,j) corresponds to A(i,i+IDIAG(j)) (for $1 \leq i + \text{IDIAG}(j) \leq N$)

Sparse Matrix-vector products $Y \leftarrow AX$

Algorithm depends on sparse matrix format:

- (Dense format)

```
DO I=1,M
  Yi=0
  DO J=1,N
    Yi = Yi + A(I, J)*X(J)
  END DO
  Y(I)=Yi
END DO
```

- Coordinate format:

- CSC format:

Sparse Matrix-vector products $Y \leftarrow AX$

Algorithm depends on sparse matrix format:

- (Dense format)

```
DO I=1,M
  Yi=0
  DO J=1,N
    Yi = Yi + A(I, J)*X(J)
  END DO
  Y(I)=Yi
END DO
```

- Coordinate format:

```
Y(1:M) = 0
DO k=1,NNZ
  Y(IRN(k)) = Y(IRN(k)) + VAL(k) * X(JCN(k))
ENDDO
```

- CSC format:

Sparse Matrix-vector products $Y \leftarrow AX$

Algorithm depends on sparse matrix format:

- (Dense format)

```
DO I=1,M
  Yi=0
  DO J=1,N
    Yi = Yi + A(I, J)*X(J)
  END DO
  Y(I)=Yi
END DO
```

- Coordinate format:

```
Y(1:M) = 0
DO k=1,NNZ
  Y(IRN(k)) = Y(IRN(k)) + VAL(k) * X(JCN(k))
ENDDO
```

- CSC format:

```
Y(1:M) = 0
DO J=1,N
  Xj=X(J)
  ! SAXPY
  DO k=COLPTR(J),COLPTR(J+1)-1
    Y(IRN(k)) = Y(IRN(k)) + VAL(k)*Xj
  ENDDO
ENDDO
```

Example of elemental matrix format

$$\mathbf{A}_1 = \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} \begin{pmatrix} -1 & 2 & 3 \\ 2 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}, \quad \mathbf{A}_2 = \begin{matrix} 3 \\ 4 \\ 5 \end{matrix} \begin{pmatrix} 2 & -1 & 3 \\ 1 & 2 & -1 \\ 3 & 2 & 1 \end{pmatrix}$$

$$\mathbf{A} = \begin{pmatrix} -1 & 2 & 3 & 0 & 0 \\ 2 & 1 & 1 & 0 & 0 \\ 1 & 1 & 3 & -1 & 3 \\ 0 & 0 & 1 & 2 & -1 \\ 0 & 0 & 3 & 2 & 1 \end{pmatrix} = \mathbf{A}_1 + \mathbf{A}_2$$

Example of elemental matrix format

$$\mathbf{A}_1 = \begin{matrix} & \begin{matrix} 1 & 2 & 3 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} & \begin{pmatrix} -1 & 2 & 3 \\ 2 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \end{matrix}, \quad \mathbf{A}_2 = \begin{matrix} & \begin{matrix} 2 & 3 \end{matrix} \\ \begin{matrix} 3 \\ 4 \\ 5 \end{matrix} & \begin{pmatrix} 2 & -1 & 3 \\ 1 & 2 & -1 \\ 3 & 2 & 1 \end{pmatrix} \end{matrix}$$

- $N=5$ $NELT=2$ $NVAR=6$ $\mathbf{A} = \sum_{i=1}^{NELT} \mathbf{A}_i$
 - ELTPTR [1:NELT+1] = 1 4 7
- ELTVAR [1:NVAR] = 1 2 3 3 4 5
 - ELTVAL [1:NVAL] = -1 2 1 2 1 1 3 1 1 2 1 3 -1 2 2 3 -1 1
- Remarks:
 - $NVAR = ELTPTR(NELT+1)-1$
 - Order of element i : $S_i = ELTPTR(i+1) - ELTPTR(i)$
 - $NVAL = \sum S_i^2$ (unsym) ou $\sum S_i(S_i + 1)/2$ (sym),
 - storage of elements in ELTVAL: by columns

- Standard ASCII format for files
- Header + Data (CSC format). key xyz:
 - x=[rcp] (real, complex, pattern)
 - y=[suhzr] (sym., uns., herm., skew sym. ($A = -A^T$), rectang.)
 - z=[ae] (assembled, elemental)
 - ex: M_T1.RSA, SHIP003.RSE
- Supplementary files: right-hand-sides, solution, permutations...
- Canonical format introduced to guarantee a unique representation (order of entries in each column, no duplicates).

File storage: Rutherford-Boeing

```
DNV-Ex 1 : Tubular joint-1999-01-17                                     M_T1
      1733710      9758      492558      1231394      0
rsa      97578      97578      4925574      0
(10I8)      (10I8)      (3e26.16)
      1      49      96      142      187      231      274      346      417      487
      556      624      691      763      834      904      973      1041      1108      1180
      1251      1321      1390      1458      1525      1573      1620      1666      1711      1755
      1798      1870      1941      2011      2080      2148      2215      2287      2358      2428
      2497      2565      2632      2704      2775      2845      2914      2982      3049      3115
...
      1      2      3      4      5      6      7      8      9      10
      11      12      49      50      51      52      53      54      55      56
      57      58      59      60      67      68      69      70      71      72
      223      224      225      226      227      228      229      230      231      232
      233      234      433      434      435      436      437      438      2      3
      4      5      6      7      8      9      10      11      12      49
      50      51      52      53      54      55      56      57      58      59
...
-0.2624989288237320E+10      0.6622960540857440E+09      0.2362753266740760E+11
0.3372081648690030E+08      -0.4851430162799610E+08      0.1573652896140010E+08
0.1704332388419270E+10      -0.7300763190874110E+09      -0.7113520995891850E+10
0.1813048723097540E+08      0.2955124446119170E+07      -0.2606931100955540E+07
0.1606040913919180E+07      -0.2377860366909130E+08      -0.1105180386670390E+09
0.1610636280324100E+08      0.4230082475435230E+07      -0.1951280618776270E+07
0.4498200951891750E+08      0.2066239484615530E+09      0.3792237438608430E+08
0.9819999042370710E+08      0.3881169368090200E+08      -0.4624480572242580E+08
```

- Example

```
%%MatrixMarket matrix coordinate real general
% Comments
5 5 8
1 1 1.000e+00
2 2 1.050e+01
3 3 1.500e-02
1 4 6.000e+00
4 2 2.505e+02
4 4 -2.800e+02
4 5 3.332e+01
5 5 1.200e+01
```

Examples of sparse matrix collections

- SuiteSparse Matrix Collection (formerly *The University of Florida Sparse Matrix Collection*) <https://sparse.tamu.edu/>
- Matrix Market <http://math.nist.gov/MatrixMarket/>
- Rutherford-Boeing
<http://www.cerfacs.fr/algor/Softs/RB/index.html>

Sparse Direct Solvers - Ordering

Gaussian elimination and sparsity

Gaussian elimination

$$\mathbf{A} = \mathbf{A}^{(1)}, \mathbf{b} = \mathbf{b}^{(1)}, \mathbf{A}^{(1)}\mathbf{x} = \mathbf{b}^{(1)}:$$

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} \quad \begin{array}{l} 2 \leftarrow 2 - 1 \times a_{21}/a_{11} \\ 3 \leftarrow 3 - 1 \times a_{31}/a_{11} \end{array}$$

$$\mathbf{A}^{(2)}\mathbf{x} = \mathbf{b}^{(2)}$$

$$\begin{pmatrix} a_{11} & a_{12}^{(2)} & a_{13}^{(2)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} \\ 0 & a_{32}^{(2)} & a_{33}^{(2)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1^{(2)} \\ b_2^{(2)} \\ b_3^{(2)} \end{pmatrix} \quad \begin{array}{l} b_2^{(2)} = b_2 - a_{21}b_1/a_{11} \dots \\ a_{32}^{(2)} = a_{32} - a_{31}a_{12}/a_{11} \dots \end{array}$$

$$\text{Finally } \mathbf{A}^{(3)}\mathbf{x} = \mathbf{b}^{(3)}$$

$$\begin{pmatrix} a_{11} & a_{12}^{(2)} & a_{13}^{(2)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} \\ 0 & 0 & a_{33}^{(3)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1^{(2)} \\ b_2^{(2)} \\ b_3^{(3)} \end{pmatrix} \quad a_{(33)}^{(3)} = a_{(33)}^{(2)} - a_{32}^{(2)}a_{23}^{(2)}/a_{22}^{(2)} \dots$$

Typical Gaussian elimination step k :

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

Relation with $\mathbf{A} = \mathbf{LU}$ factorization

- One step of Gaussian elimination can be written:

$$\mathbf{A}^{(k+1)} = \mathbf{L}^{(k)} \mathbf{A}^{(k)}, \text{ with}$$

$$\mathbf{L}^{(k)} = \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & -l_{k+1,k} & \ddots & \\ & & \vdots & \ddots & \\ & & -l_{n,k} & & 1 \end{pmatrix} \text{ and } l_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}.$$

- Then, $\mathbf{A}^{(n)} = \mathbf{U} = \mathbf{L}^{(n-1)} \dots \mathbf{L}^{(1)} \mathbf{A}$, which gives $\boxed{\mathbf{A} = \mathbf{LU}}$,

$$\text{with } \mathbf{L} = [\mathbf{L}^{(1)}]^{-1} \dots [\mathbf{L}^{(n-1)}]^{-1} = \begin{pmatrix} 1 & & 0 \\ & \ddots & \\ & & 1 \\ & & l_{i,j} & \\ & & & \ddots & \\ & & & & 1 \end{pmatrix}.$$

- In dense codes, entries of \mathbf{L} and \mathbf{U} overwrite entries of \mathbf{A} .
- Furthermore, if \mathbf{A} is symmetric, $\boxed{\mathbf{A} = \mathbf{LDL}^T}$ with $d_{kk} = a_{kk}^{(k)}$:
 $\mathbf{A} = \mathbf{LU} = \mathbf{A}^t = \mathbf{U}^t \mathbf{L}^t$ implies $(\mathbf{U})(\mathbf{L}^t)^{-1} = \mathbf{L}^{-1} \mathbf{U}^t = \mathbf{D}$ diagonal and $\mathbf{U} = \mathbf{DL}^t$, thus $\mathbf{A} = \mathbf{L}(\mathbf{DL}^t) = \mathbf{LDL}^t$

Dense LU factorization

- Step by step columns of \mathbf{A} are set to zero and \mathbf{A} is updated $\mathbf{L}^{(n-1)} \dots \mathbf{L}^{(1)} \mathbf{A} = \mathbf{U}$ leading to
 $\mathbf{A} = \mathbf{L}\mathbf{U}$ where $\mathbf{L} = [\mathbf{L}^{(1)}]^{-1} \dots [\mathbf{L}^{(n-1)}]^{-1}$
- - zero entries in column of \mathbf{A} can be replaced by entries in \mathbf{L}
- row entries of \mathbf{U} can be stored in corresponding locations of \mathbf{A}

```
1: for  $k = 1$  to  $n$  do
2:    $L(k:k) = 1$  ;  $L(k+1:n, k) = \frac{A(k+1:n, k)}{A(k, k)}$ 
3:    $U(k, k:n) = A(k, k:n)$ 
4:   for  $j = k+1$  to  $n$  do
5:     for  $i = k+1$  to  $n$  do
6:        $A(i, j) = A(i, j) - L(i, k) \times U(k, j)$ 
7:     end for
8:   end for
9: end for
```

Algorithm 13: Dense LU factorization

When $|A(k, k)|$ is relatively too small, numerical pivoting required

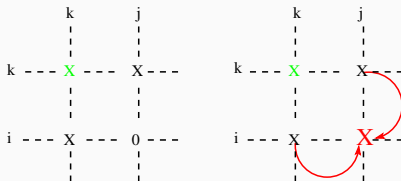
Gaussian elimination and Sparsity

Step k of **LU** factorization (a_{kk} pivot):

- For $i > k$ compute $l_{ik} = a_{ik}/a_{kk}$ ($= a'_{ik}$),
- For $i > k, j > k$ update remaining rows/cols in matrix

$$a'_{ij} = a_{ij} - l_{ik} \times a_{kj} = a_{ij} - \frac{a_{ik} \times a_{kj}}{a_{kk}}$$

- If $a_{ik} \neq 0$ and $a_{kj} \neq 0$ then $a'_{ij} \neq 0$
- If a_{ij} was zero \rightarrow its non-zero value must be stored



fill-in

- Idem for Cholesky Factorization

Gaussian elimination and sparsity

- Interest of permuting a matrix:

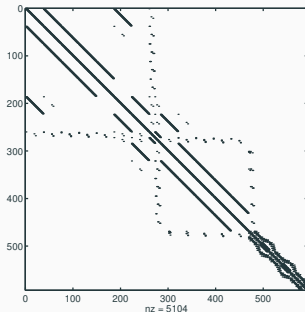
$$\begin{pmatrix} X & X & X & X & X \\ X & X & 0 & 0 & 0 \\ X & 0 & X & 0 & 0 \\ X & 0 & 0 & X & 0 \\ X & 0 & 0 & 0 & X \end{pmatrix} \quad 1 \leftrightarrow 5 \quad \begin{pmatrix} X & 0 & 0 & 0 & X \\ 0 & X & 0 & 0 & X \\ 0 & 0 & X & 0 & X \\ 0 & 0 & 0 & X & X \\ X & X & X & X & X \end{pmatrix}$$

- Ordering the variables has a strong impact on
 - fill-in
 - number of operations
 - shape of the dependency **graph (tree)** and parallelism
- Fill reduction is NP-hard in general [Yannakakis 81]

Illustration: Reverse Cuthill-McKee on matrix dwt_592.rua

Harwell-Boeing matrix: dwt_592.rua, structural computing on a submarine. NNZ(LU factors)=58202

Original matrix



Factorized matrix

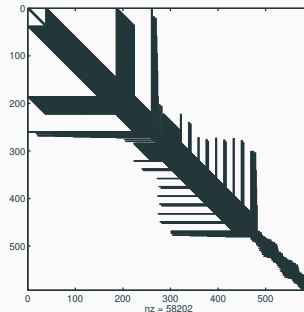
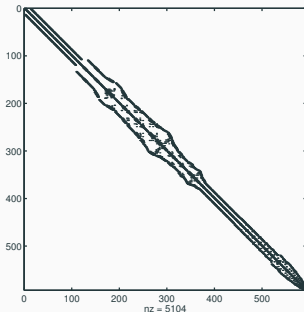


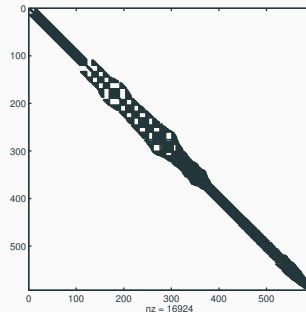
Illustration: Reverse Cuthill-McKee on matrix dwt_592.rua

$\text{NNZ}(\text{LU factors})=16924$

*Permuted matrix
(RCM)*



Factorized permuted matrix



Sparse LU factorization : a (too) simple algorithm

Only non-zeros are stored and operated on

```
1: Permute matrix A to reduce fill-in and flops (NP complete problem)
2: for  $k = 1$  to  $n$  do
3:    $L(k : k) = 1$  ; For nonzeros in column k:  $L(k + 1 : n, k) = \frac{A(k+1:n,k)}{A(k,k)}$ 
4:    $U(k, k : n) = A(k, k : n)$ 
5:   for  $j = k + 1$  to  $n$  limited to nonzeros in row  $U(k, :)$  do
6:     for  $i = k + 1$  to  $n$  limited to nonzeros in col.  $L(:, k)$  do
7:        $A(i, j) = A(i, j) - L(i, k) \times U(k, j)$ 
8:     end for
9:   end for
10: end for
```

Algorithm 14: Simple sparse **LU** factorization

Questions

Dynamic data structure for A to accommodate fill-in

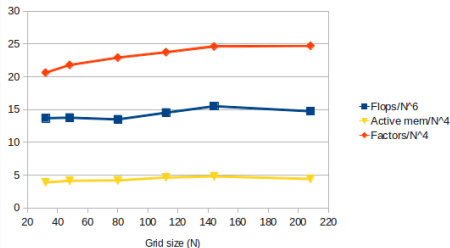
Data access efficiency; Can we predict position of fill-in ?

$|A(k, k)|$ too small \rightarrow numerical permutation needed !!!

Complexity of sparse direct methods

Regular problems (nested dissections)	2D $N \times N$ grid	3D $N \times N \times N$ grid
Nonzeros in original matrix	$\Theta(N^2)$	$\Theta(N^3)$
Nonzeros in factors	$\Theta(N^2 \log N)$	$\Theta(N^4)$
Floating-point ops	$\Theta(N^3)$	$\Theta(N^6)$

Exo : flops : 2D 1000×1000 ; 3D $100 \times 100 \times 100$; full 3D ?



3D example in earth science:
acoustic wave propagation,
27-point finite difference grid

Goal [Seiscope project:
<https://seiscope2.osug.fr/>]:
LU on complete earth

Extrapolation on a $n = N^3 = 1000^3$ grid:

55 Exaflops, 200 TBytes for factors, 40 TBytes of working memory!

Sparse Direct Solvers - Ordering

Three-phase scheme to solve $Ax = b$

Three-phase scheme to solve $Ax = b$

1. Analysis step

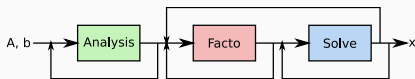
- Preprocessing of A (symmetric/unsymmetric orderings, scalings)
- Build the dependency graph (elimination tree, eDAG ...)
- $A_{pre} = P D_r A Q_c D_c P^T$,

2. Factorization ($A_{pre} = LU, LDL^T, LL^T, QR$)

Numerical pivoting

3. Solution based on factored matrices

- triangular solves: $Ly = b_{pre}$, then $Ux_{pre} = y$
- improvement of solution (iterative refinement), error analysis



Sparse Direct Solvers - Ordering

Permutation matrices

A **permutation matrix** is a square $(0, 1)$ -matrix where each row and column has a single 1.

If P is a permutation matrix, $PP^T = I$, i.e., it is an orthogonal matrix.

Permutation matrices

Let,

$$A = \begin{matrix} & \begin{matrix} 1 & 2 & 3 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} & \begin{pmatrix} \times & \times & \\ \times & & \times \\ & & \times \end{pmatrix} \end{matrix}$$

and suppose we want to permute columns as $[2, 1, 3]$.

Define $p_{2,1} = 1$, $p_{1,2} = 1$, $p_{3,3} = 1$ (if column j to be at position i , set $p_{ji} = 1$), and $B = AP$

$$B = \begin{matrix} & \begin{matrix} 2 & 1 & 3 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} & \begin{pmatrix} \times & \times & \\ & \times & \times \\ & & \times \end{pmatrix} \end{matrix} = \begin{matrix} & \begin{matrix} 1 & 2 & 3 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} & \begin{pmatrix} \times & \times & \\ \times & & \times \\ & & \times \end{pmatrix} \end{matrix} \begin{matrix} & \begin{matrix} 1 & 2 & 3 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} & \begin{pmatrix} & 1 & \\ 1 & & \\ & & 1 \end{pmatrix} \end{matrix}$$

Permutation matrices - Symmetric case

Let,

$$A = \begin{matrix} & \begin{matrix} 1 & 2 & 3 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} & \begin{pmatrix} \times & \times & \times \\ \times & \times & \\ \times & & \times \end{pmatrix} \end{matrix}$$

and suppose we want to permute lines and columns as $[3, 2, 1]$ to preserve symmetry.

Define $p_{3,1} = 1$, $p_{2,2} = 1$, $p_{1,3} = 1$, and $B = P^T A P$

$$B = \begin{matrix} & \begin{matrix} 3 & 2 & 1 \end{matrix} \\ \begin{matrix} 3 \\ 2 \\ 1 \end{matrix} & \begin{pmatrix} \times & & \times \\ & \times & \times \\ \times & \times & \times \end{pmatrix} \end{matrix} = \begin{matrix} & \begin{matrix} 1 & 2 & 3 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} & \begin{pmatrix} & & 1 \\ & 1 & \\ 1 & & \end{pmatrix} \end{matrix} \begin{matrix} & \begin{matrix} 1 & 2 & 3 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} & \begin{pmatrix} \times & \times & \times \\ \times & \times & \\ \times & & \times \end{pmatrix} \end{matrix} \begin{matrix} & \begin{matrix} 1 & 2 & 3 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} & \begin{pmatrix} & & 1 \\ & 1 & \\ 1 & & \end{pmatrix} \end{matrix}$$

Sparse Direct Solvers - Ordering

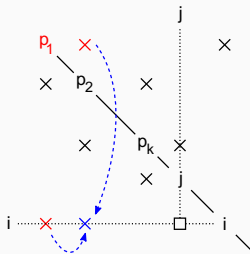
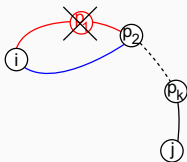
Fill-in characterization (symmetric matrices)

Fill-in characterization (proof intuition)

Let A be a symmetric matrix ($G(A)$ its associated graph), L the matrix of factors $A = LL^T$;

Fill path theorem, Rose, Tarjan, Leuker, 76

$l_{ij} \neq 0$ iff there is a path in $G(A)$ between i and j such that all nodes in the path have indices smaller than both i and j .

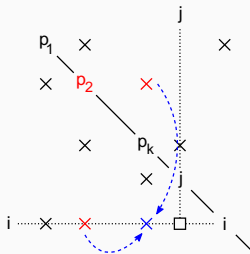
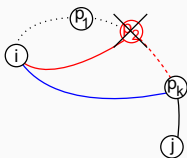


Fill-in characterization (proof intuition)

Let A be a symmetric matrix ($G(A)$ its associated graph), L the matrix of factors $A = LL^T$;

Fill path theorem, Rose, Tarjan, Leuker, 76

$l_{ij} \neq 0$ iff there is a path in $G(A)$ between i and j such that all nodes in the path have indices smaller than both i and j .

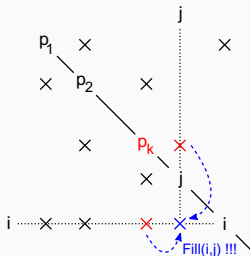
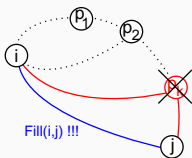


Fill-in characterization (proof intuition)

Let A be a symmetric matrix ($G(A)$ its associated graph), L the matrix of factors $A = LL^T$;

Fill path theorem, Rose, Tarjan, Leuker, 76

$l_{ij} \neq 0$ iff there is a path in $G(A)$ between i and j such that all nodes in the path have indices smaller than both i and j .



Sparse Direct Solvers - Ordering

**Graph definitions and relations to
sparse matrices**

Graph notations and definitions

A **graph** $G = (V, E)$ consists of a finite set V , called the vertex set and a finite, binary relation E on V , called the edge set.

Three standard graph models

Undirected graph: The edges are unordered pair of vertices, i.e., $\{u, v\} \in E$ for some $u, v \in V$.

Directed graph: The edges are ordered pair of vertices, that is, (u, v) and (v, u) are two different edges.

Bipartite graph: $G = (U \cup V, E)$ consists of two disjoint vertex sets U and V such that for each edge $(u, v) \in E$, $u \in U$ and $v \in V$.

An **ordering** or **labelling** of $G = (V, E)$ having n vertices, i.e., $|V| = n$, is a mapping of V onto $1, 2, \dots, n$.

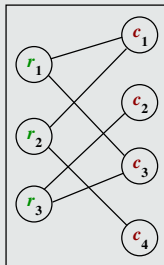
Matrices and graphs: rectangular matrices

The rows/columns and nonzeros of a given sparse matrix correspond (with natural labelling) to the vertices and edges, respectively, of a graph.

Rectangular matrices

$$A = \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} & \begin{pmatrix} \times & & \times & \\ \times & & & \times \\ & \times & \times & \end{pmatrix} \end{matrix}$$

Bipartite graph



The set of rows corresponds to one of the vertex set R , the set of columns corresponds to the other vertex set C such that for each $a_{ij} \neq 0$, (r_i, c_j) is an edge.

Matrices and graphs: square unsymmetric pattern

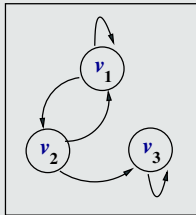
The rows/columns and nonzeros of a given sparse matrix correspond (with natural labelling) to the vertices and edges, respectively, of a graph.

Square unsymmetric pattern matrices

$$A = \begin{matrix} & \begin{matrix} 1 & 2 & 3 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} & \begin{pmatrix} \times & \times & \\ \times & & \times \\ & & \times \end{pmatrix} \end{matrix}$$

Graph models

- Bipartite graph as before.
- Directed graph



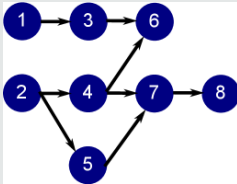
The set of rows/cols corresponds the vertex set V such that for each $a_{ij} \neq 0$, (v_i, v_j) is an edge. Transposed view possible too, i.e., the edge (v_i, v_j) directed from column i to row j . Usually self-loops are omitted.

Matrices and graphs: square unsymmetric pattern

A special subclass

Directed acyclic graphs (DAG):

A directed graphs with no loops (maybe except for self-loops).



DAGs

We can sort the vertices such that if (u, v) is an edge, then u appears before v in the ordering.

Question: What kind of matrices have a DAG structure ?

Matrices and graphs: symmetric pattern

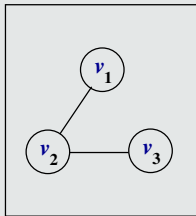
The rows/columns and nonzeros of a given sparse matrix correspond (with natural labelling) to the vertices and edges, respectively, of a graph.

Square symmetric pattern matrices

$$A = \begin{matrix} & \begin{matrix} 1 & 2 & 3 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} & \begin{pmatrix} & \times & \\ \times & \times & \times \\ & \times & \times \end{pmatrix} \end{matrix}$$

Graph models

- Bipartite and directed graphs as before.
- Undirected graph



The set of rows/cols corresponds the vertex set V such that for each $a_{ij}, a_{ji} \neq 0$, $\{v_i, v_j\}$ is an edge. No self-loops; usually the main diagonal is assumed to be zero-free.

Definitions: edges, degrees, and paths

Many definitions for directed and undirected graphs are the same. We will use (u, v) to refer to an edge of an undirected or directed graph to avoid repeated definitions.

- An edge (u, v) is said to **incident on** the vertices u and v .
- For any vertex u , the set of vertices in $\text{adj}(u) = \{v : (u, v) \in E\}$ are called the **neighbors** of u . The vertices in $\text{adj}(u)$ are said to be **adjacent** to u .
- The **degree** of a vertex is the number of edges incident on it.
- A **path** p of length k is a sequence of vertices $\langle v_0, v_1, \dots, v_k \rangle$ where $(v_{i-1}, v_i) \in E$ for $i = 1, \dots, k$. The two end points v_0 and v_k are said to be connected by the path p , and the vertex v_k is said to be **reachable** from v_0 .

Definitions: Components

- An undirected graph is said to be **connected** if every pair of vertices is connected by a path.
- The **connected components** of an undirected graph are the equivalence classes of vertices under the “is reachable” from relation.
- A directed graph is said to be **strongly connected** if every pair of vertices are reachable from each other.
- The **strongly connected components** of a directed graph are the equivalence classes of vertices under the “are mutually reachable” relation.

Sparse Direct Solvers - Ordering

Trees and spanning trees

Definitions: trees and spanning trees

A **tree** is a connected, acyclic, undirected graph.

Properties of trees

- Any two vertices are connected by a unique path.
- $|E| = |V| - 1$

A **rooted tree** is a tree with a distinguished vertex r , called the **root**.

There is a **unique path** from the root r to every other vertex v . Any vertex y in that path is called an **ancestor** of v . If y is an ancestor of v , then v is a **descendant** of y .

The **subtree rooted at** v is the tree induced by the descendants of v , rooted at v .

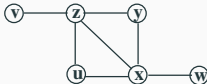
A **spanning tree** of a connected graph $G = (V, E)$ is a tree $T = (V, F)$, such that $F \subseteq E$.

Sparse Direct Solvers - Ordering

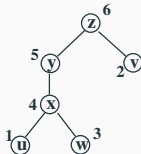
Ordering and tree traversal

Ordering of the vertices of a rooted tree

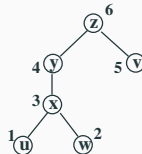
- A **topological ordering** of a rooted tree is an ordering that numbers children vertices before their parent.
- A **postorder** is a topological ordering which numbers the vertices in any subtree consecutively.



Connected graph G



Rooted spanning tree
with topological ordering



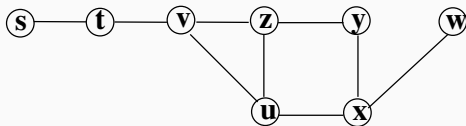
Rooted spanning tree
with postordering

How to explore a graph:

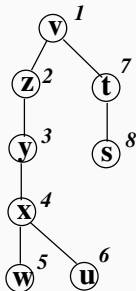
Two algorithms such that each edge is traversed exactly once in the forward and reverse direction and each vertex is visited. (Connected graphs are considered in the description of the algorithms.)

- **Depth first search** : Starting from a given vertex (*mark* it) follow a path (and mark each vertex in the path) until a vertex that has no neighbor or that is adjacent to only marked vertices. Return to previous vertex and continue.
- **Breadth first search** Select a vertex and put it into an initially empty queue of vertices *to be visited*. Repeatedly remove the vertex x at the head of the queue and place all neighbors of x that were not enqueue before into the queue.

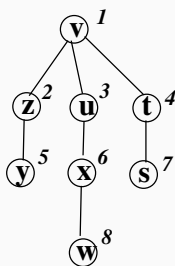
Illustration of DFS and BFS exploration



Depth-first spanning tree (v)



Breadth-first spanning tree (v)



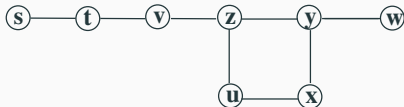
Each vertex is visited once.

Sparse Direct Solvers - Ordering

**Peripheral and pseudo-peripheral
vertices**

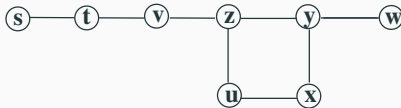
Graph shapes and peripheral vertices

- The **distance** between two vertices of a graph is the size of the shortest path joining those vertices.
- The **eccentricity** $l(v) = \max \{d(v, w) \mid w \in V\}$
- The **diameter** $\delta(G) = \max \{l(v) \mid v \in V\}$
- v is a **peripheral** vertex if $l(v) = \delta(G)$
- Example of connected graph with $\delta(G) = 5$ and peripheral vertices s, w, x .

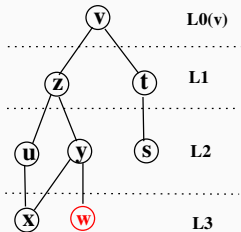


- The **rooted level structure** (v) of G is a partitioning of G into levels $L_0(v), \dots, L_{l(v)}(v)$ such that $L_0(v) = v$
 $L_i(v) = \text{Adj}(L_{i-1}(v)) \setminus L_{i-2}(v)$ (with $L_{-1} = \emptyset$)
- A **pseudo-peripheral** is a node with a large eccentricity.
- Algorithm to determine a pseudo-peripheral :
Let $i = 0$, r_i be an arbitrary node, and $nlevels = l(r_i)$. The algorithm iteratively updates $nlevels$ by selecting the vertex r_{i+1} of minimum degree of $L_{l(r_i)}$ and stopping when $l(r_{i+1}) \leq nlevels$.

Pseudo-peripheral vertices

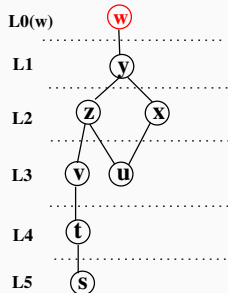


First step, select v



Level structure(v)

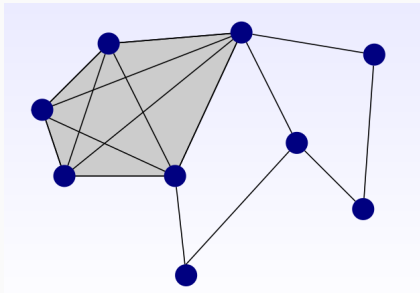
2nd step: select W in L3 (min. degree)



Cliques

In an undirected graph $G = (V, E)$, a set of vertices $S \subseteq V$ is a clique if for all $s, t \in S$, we have $(s, t) \in E$.

In a symmetric matrix A , a clique corresponds to a subset of rows R and the corresponding columns such that the matrix $A(R, R)$ is full.



Sparse Direct Solvers - Ordering

The elimination process in the graphs

Symmetric matrices and graphs

Predicting structure helps

1. in reducing the memory requirements,
2. in achieving high performance,
3. in simplifying the algorithms.

We will consider the **Cholesky factorization** $A = LL^T$. In this case, structural and numerical aspects are neatly separated.

For general case (e.g., LU factorization) **pivoting** is necessary and depends on the **actual numerical values**. There are combinatorial tools for these, but we will not cover.

Structure prediction algorithms should run, preferably, faster than the numerical computations that will follow.

Symmetric matrices and graphs

- Assumptions: \mathbf{A} symmetric and pivots are chosen from the diagonal
- Structure of \mathbf{A} symmetric represented by the graph $G = (V, E)$
 - Vertices are associated to columns: $V = \{1, \dots, n\}$
 - Edges E are defined by: $(i, j) \in E \leftrightarrow a_{ij} \neq 0$
 - G undirected (symmetry of \mathbf{A})

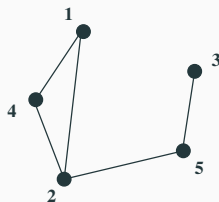
Symmetric matrices and graphs

- Remarks:

- Number of nonzeros in column $j = |\text{adj}_G(j)|$
- Symmetric permutation \equiv renumbering the graph

	1	2	3	4	5
1	X	X		X	
2	X	X		X	X
3			X		X
4	X	X		X	
5		X	X		X

Symmetric matrix



Corresponding graph

The elimination model for symmetric matrices

A **symmetric, positive definite** matrix can be factorized by means of the **Cholesky** algorithm

```
for  $k = 1, \dots, n$  do
```

$$l_{kk} = \sqrt{a_{kk}^{(k-1)}}$$

```
  for  $i = k + 1, \dots, n$  do
```

$$l_{ik} = a_{ik}^{(k-1)} / l_{kk}$$

```
    for  $j = k + 1, \dots, i$  do
```

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

```
    end for
```

```
  end for
```

```
end for
```

$$\begin{pmatrix} a_{11} & & & \\ a_{21} & a_{22} & & \\ a_{31} & a_{32} & a_{33} & \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

The elimination model for symmetric matrices

- ▶ Let A be a symmetric positive definite matrix of order n
- ▶ The LL^T factorization can be described by the equation:

$$\begin{aligned} A &= A_0 = \begin{pmatrix} d_1 & v_1^T \\ v_1 & A_1 \end{pmatrix} \\ &= \begin{pmatrix} \sqrt{d_1} & 0 \\ \frac{v_1}{\sqrt{d_1}} & I_{n-1} \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & A_1 \end{pmatrix} \begin{pmatrix} \sqrt{d_1} & \frac{v_1^T}{\sqrt{d_1}} \\ 0 & I_{n-1} \end{pmatrix} \\ &= L_1 A_1 L_1^T, \text{ where} \end{aligned}$$

$$A_1 = \overline{A_1} - \frac{v_1 v_1^T}{d_1}$$

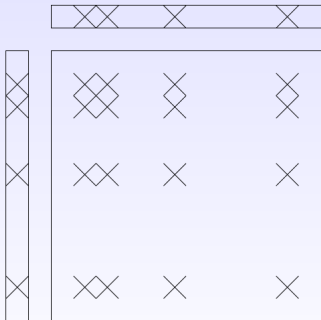
- ▶ The basic step is applied on $A_1 A_2 \cdots$ to obtain :

$$A = (L_1 L_2 \cdots L_{n-1}) I_n (L_{n-1}^T \cdots L_2^T L_1^T) = LL^T$$

The elimination model for symmetric matrices

The basic step: $A_1 = \overline{A_1} - \frac{v_1 v_1^T}{d_1}$

What is $v_1 v_1^T$ in terms of structure?



v_1 is a column of A , hence the neighbors of the corresponding vertex.

$v_1 v_1^T$ results in a dense sub-block in A_1 , i.e., the elimination of a node results in the creation of a **clique** that connects all the neighbors of the eliminated node.

If any of the nonzeros in dense submatrix are not in A , then we have fill-ins.

The elimination process in the graphs

$G_U(V, E) \leftarrow$ undirected graph of \mathbf{A}

for $k = 1 : n - 1$ **do**

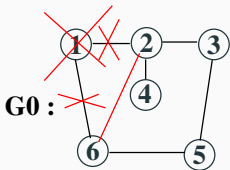
$V \leftarrow V - \{k\}$ {remove vertex k }

$E \leftarrow E - \{(k, \ell) : \ell \in \text{adj}(k)\} \cup \{(x, y) : x \in \text{adj}(k) \text{ and } y \in \text{adj}(k)\}$

$G_k \leftarrow (V, E)$ {for definition}

end for

G_k are the so-called **elimination graphs** (Parter, '61).

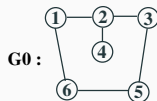


$H_0 =$

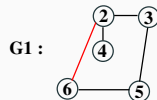
1	x			x	
x	2	x	x		x
	x	3		x	
	x		4		
		x		5	x
x	x			x	6

Red 'X' marks are placed over the first row and the first column of the matrix, and a red line connects the element at row 2, column 5 to the element at row 5, column 2.

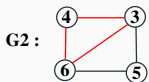
A sequence of elimination graphs



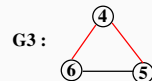
$$H0 = \begin{bmatrix} 1 & \times & & & & \times \\ \times & 2 & \times & \times & & \\ & \times & 3 & & \times & \\ & \times & & 4 & & \\ & & \times & & 5 & \times \\ \times & & & & \times & 6 \end{bmatrix}$$



$$H1 = \begin{bmatrix} 2 & \times & \times & & & + \\ \times & 3 & & \times & & \\ & \times & & 4 & & \\ & & \times & & 5 & \times \\ + & & & & \times & 6 \end{bmatrix}$$



$$H2 = \begin{bmatrix} 3 & + & \times & + \\ + & 4 & & + \\ \times & & 5 & \times \\ + & + & \times & 6 \end{bmatrix}$$



$$H3 = \begin{bmatrix} 4 & + & + \\ + & 5 & \times \\ + & \times & 6 \end{bmatrix}$$

$$\begin{pmatrix} l_{11} & & & & & \\ l_{21} & l_{22} & & & & \\ & l_{32} & l_{33} & & & \\ & l_{42} & l_{43} & l_{44} & & \\ & & l_{53} & l_{54} & l_{55} & \\ l_{61} & l_{62} & l_{63} & l_{64} & l_{65} & l_{66} \end{pmatrix}$$

Sparse Direct Solvers - Ordering

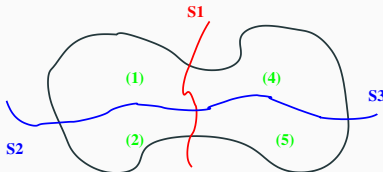
Fill-reducing heuristics

Fill-reducing heuristics

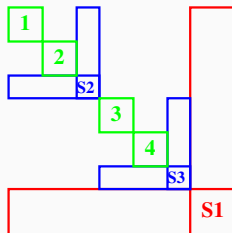
Three main classes of methods for minimizing fill-in during factorization

1. Global approach: The matrix is permuted into a matrix with a given pattern; **fill-in is restricted to occur within that structure**
 - **Cuthill-McKee** (block tridiagonal matrix)
(Remark: BFS traversal of the associated graph)
 - **Nested dissections** (“block bordered” matrix)
(Remark: interpretation using the fill-path theorem)

Graph partitioning



Permuted matrix



2. Local heuristics: at each step of the factorization, selection of the pivot that is likely to minimize fill-in.
Method is characterized by the way pivots are selected.
 - Markowitz criterion (for a general matrix).
 - **Minimum degree** or Minimum fill-in (for symmetric matrices).
3. Hybrid approaches: once the matrix is permuted to block structure, local heuristics are used within the blocks.

Sparse Direct Solvers - Ordering

**Cuthill-McKee - BFS traversal of the
associated graph**

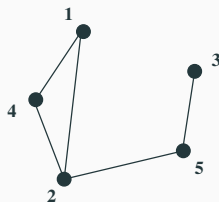
Symmetric matrices and graphs (matlab exo1.m)

- Remarks:

- Number of nonzeros in column $j = |\text{adj}_G(j)|$
- Symmetric permutation \equiv renumbering the graph

	1	2	3	4	5
1	X	X		X	
2	X	X		X	X
3			X		X
4	X	X		X	
5		X	X		X

Symmetric matrix



Corresponding graph

Breadth-First Search (BFS) of a graph to permute a matrix

Exercise (BFS traversal to order a matrix)

Given a symmetric matrix \mathbf{A} and its associated graph $G = (V, E)$

1. Derive from BFS traversal of G an algorithm to reorder matrix \mathbf{A}
2. Explain the structural property of the permuted matrix?
3. Explain why such an ordering can help reducing fill-in during LU factorisation?

CM: Algorithm

Level sets are built from the vertex of minimum degree. At any level, priority is given to a vertex with smaller number of neighbors.

Cuthill-McKee

pick a vertex v and order it as the first vertex

$S \leftarrow \{v\}$

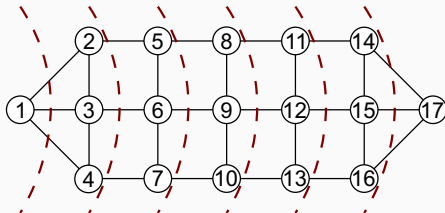
while $S \neq V$ **do**

$S' \leftarrow$ all vertices in $V \setminus S$ which are adjacent to S

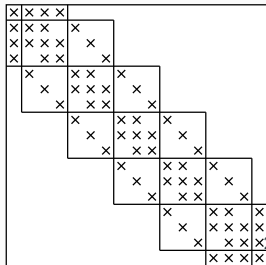
 order vertices in S' in increasing order of degrees

$S \leftarrow S \cup S'$

end while



(example from Duff, Erisman and Reid 86)

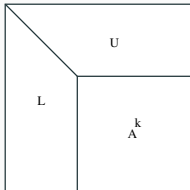


Sparse Direct Solvers - Ordering

Minimum degree

Reordering unsymmetric matrices: Markowitz criterion

- At step k of Gaussian elimination:

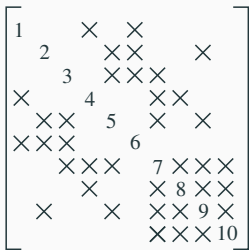


- r_i^k = number of non-zeros in row i of \mathbf{A}^k
 - c_j^k = number of non-zeros in column j of \mathbf{A}^k
 - a_{ij} must be large enough and should minimize $(r_i^k - 1) \times (c_j^k - 1) \quad \forall i, j > k$
- Minimum degree : Markowitz criterion for symmetric diagonally dominant matrices

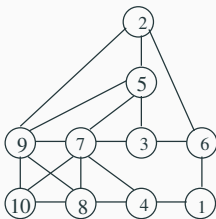
Minimum degree algorithm

- **Step 1:**

Select the vertex that possesses the smallest number of neighbors in G^0 .



(a) Sparse symmetric matrix

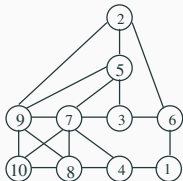


(b) Elimination graph

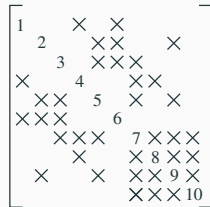
The node/variable selected is 1 of degree 2.

Illustration

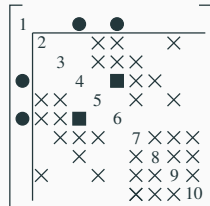
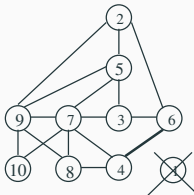
Step 1: elimination of pivot 1



(a) Elimination graph



(b) Factors and active submatrix



× Initial nonzeros

● Nonzeros in factors

■ Fill-in

Minimum degree algorithm applied to the graph:

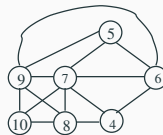
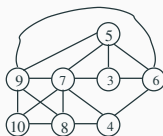
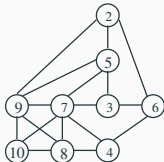
- Step k : Select the node with the smallest number of neighbors
- G^k is built from G^{k-1} by *suppressing the pivot* and *adding edges* corresponding to fill-in.

Minimum degree algorithm based on elimination graphs

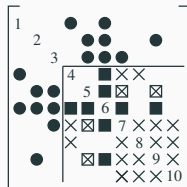
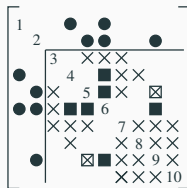
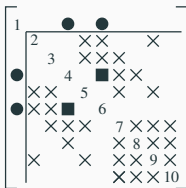
```
 $\forall i \in [1 \dots n] \quad t_i = |\text{Adj}_{G^0}(i)|$   
For  $k = 1$  to  $n$  Do  
     $p = \min_{i \in V_{k-1}} (t_i)$   
    For each  $i \in \text{Adj}_{G^{k-1}}(p)$  Do  
         $\text{Adj}_{G^k}(i) = (\text{Adj}_{G^{k-1}}(i) \cup \text{Adj}_{G^{k-1}}(p)) \setminus \{i, p\}$   
         $t_i = |\text{Adj}_{G^k}(i)|$   
    EndFor  
     $V^k = V^{k-1} \setminus p$   
EndFor
```

Illustration (cont'd)

Graphs G_1 , G_2 , G_3 and corresponding reduced matrices.



(a) Elimination graphs



(b) Factors and active submatrices

× Original nonzero

⊗ Original nonzero modified

■ Fill-in

● Nonzeros in factors

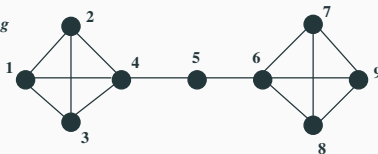
Minimum Degree does not always minimize fill-in !!!

Consider the following matrix

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

Remark: Using initial ordering

↓
No fill-in

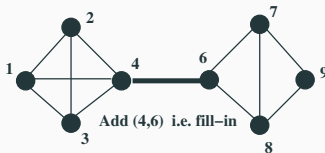


Corresponding elimination graph

Step 1 of Minimum Degree:

Select pivot 5 (minimum degree = 2)

Updated graph



Exercice

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

mat2

1. ordre naturel
2. minimum degree
3. CMK en partant de 1

CMK1 et CMK2

Sparse Direct Solvers - Multifrontal Method

Sparse Direct Solvers - Multifrontal Method

The elimination tree

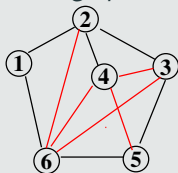
The elimination tree for symmetric matrices

The elimination tree provides structural information relevant to describing both Gaussian elimination of symmetric matrices and the orthogonal factorization.

Its extension to unsymmetric matrices is possible.

Reminder

- A **spanning tree** of a connected graph $G = (V, E)$ is a tree $T = (V, F)$, such that $F \subseteq E$.
- A **topological ordering** of a rooted tree is an ordering that numbers children vertices before their parent.
- A **postorder** is a topological ordering which numbers the vertices in any subtree consecutively.
- Let $\mathbf{F} = \mathbf{L} + \mathbf{L}^T$ be the filled matrix, and $G(\mathbf{F})$ the *filled graph* of \mathbf{A} denoted by $G^+(\mathbf{A})$.



$$G^+(\mathbf{A}) = G(\mathbf{F})$$

$$\begin{bmatrix} 1 & \bullet & & & & \bullet \\ \bullet & 2 & \bullet & \bullet & & \bullet \\ & \bullet & 3 & \bullet & \bullet & \bullet \\ & \bullet & \bullet & 4 & \bullet & \bullet \\ & & \bullet & \bullet & 5 & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet & 6 \end{bmatrix}$$

$$\mathbf{F} = \mathbf{L} + \mathbf{L}^T$$

Let A be an $n \times n$ symmetric positive-definite and irreducible matrix, $A = LL^T$ its Cholesky factorization, and $G^+(A)$ its filled graph (graph of $F = L + L^T$).

A first definition

Since A is irreducible, each of the first $n - 1$ columns of L has at least one off-diagonal nonzero.

For each column $j < n$ of L , remove all the nonzeros in the column j except the first one below the diagonal.

Let L_t denote the remaining structure and consider the matrix $F_t = L_t + L_t^T$. The graph $G(F_t)$ is a tree called the **elimination tree**.

$$A = \begin{pmatrix} a & \bullet & & \bullet & & \\ & b & \bullet & & & \\ \bullet & & c & & & \\ & & & d & & \bullet \bullet \\ \bullet & & & & e & \bullet \bullet \\ & & & & & f \bullet \bullet \\ & & & & & & g \\ \bullet & & & & & & & h \bullet \bullet \\ & & \bullet \bullet & & & & & & i \\ \bullet & & \bullet \bullet & & & & & & & j \end{pmatrix}$$

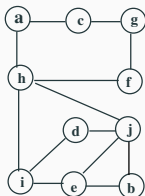
$$F = \begin{pmatrix} a & \bullet & & \bullet & & \\ & b & \bullet & & & \\ \bullet & & c & & \circ & \\ & & & d & & \bullet \bullet \\ \bullet & & & & e & \bullet \bullet \\ & & & & & f \bullet \bullet \\ & & & & & & g \circ \\ \bullet & & \circ & & & & & h \bullet \bullet \\ & & & \bullet \bullet & & & & & i \circ \\ \bullet & & \bullet \bullet & & & & & & \bullet \circ j \end{pmatrix}$$

$$F_t = \begin{pmatrix} a & \bullet & & \bullet & & \\ & b & \bullet & & & \\ \bullet & & c & & & \\ & & & d & & \bullet \\ \bullet & & & & e & \bullet \\ & & & & & f \bullet \\ & & & & & & g \circ \\ \bullet & & & & & & & h \bullet \\ & & & & & & & & i \circ \\ \bullet \bullet & & & & & & & & \bullet \circ j \end{pmatrix}$$

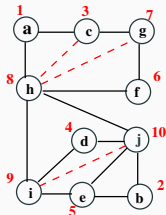
A first definition

The elimination tree of A is a spanning tree of $G^+(A)$ satisfying the relation $PARENT[j] = \min\{i > j : \ell_{ij} \neq 0\}$.

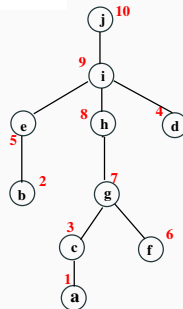
$$F = \begin{pmatrix} a & \bullet & & & & & & & & \\ & b & \bullet & & & & & & & \\ & & c & \bullet & & \circ & & & & \\ & & & d & \bullet & & \bullet & & & \\ & & & & e & \bullet & & & & \\ & & & & & f & \bullet & & & \\ & & & & & & g & \circ & & \\ & & & & & & & h & \bullet & \\ & & & & & & & & i & \circ \\ & & & & & & & & & j \end{pmatrix} \quad F_1 = \begin{pmatrix} a & \bullet & & & & & & & & \\ & b & \bullet & & & & & & & \\ & & c & \bullet & & & & & & \\ & & & d & \bullet & & & & & \\ & & & & e & \bullet & & & & \\ & & & & & f & \bullet & & & \\ & & & & & & g & \circ & & \\ & & & & & & & h & \bullet & \\ & & & & & & & & i & \circ \\ & & & & & & & & & j \end{pmatrix}$$



$G(A)$



$G^+(A) = G(F)$

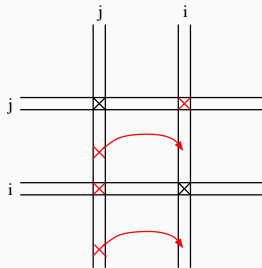


$T(A)$

A second definition: Represents column dependencies

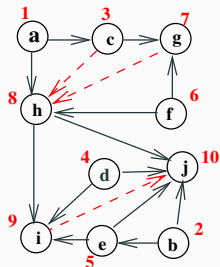
- Dependency between columns of L :

- Column $i > j$ depends on column j iff $\ell_{ij} \neq 0$
- Use a directed graph to express this dependency (edge from j to i , if column i depends on column j)
- Simplify redundant dependencies (transitive reduction)

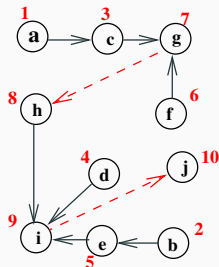


- The transitive reduction of the directed filled graph gives the elimination tree structure. Remove a directed edge (j, i) if there is a path of length greater than one from j to i .

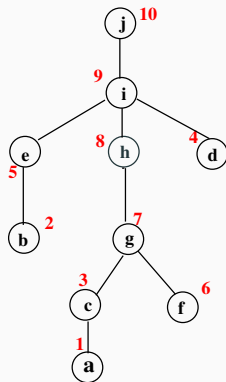
Directed filled graph and its transitive reduction



Directed filled graph



Transitive reduction


$$\mathbf{T}(\mathbf{A})$$

The elimination tree: a major tool to model factorization

The elimination tree and its generalisation to unsymmetric matrices are compact structures of major importance during sparse factorization

- Express the order in which variables can be eliminated: because the elimination of a variable only affects the elimination of its ancestors, any **topological order** of the elimination tree will lead to a **correct result** and to the **same fill-in**
- Express concurrency: because variables in separate subtrees do not affect each other, they can be eliminated in **parallel**
- **Efficient to characterize the structure of the factors** more efficiently than with elimination graphs

Sparse Direct Solvers - Multifrontal Method

**Elimination tree and Multifrontal
approach**

Elimination tree and Multifrontal approach

We recall that:

The elimination tree is the dependency graph of the factorization.

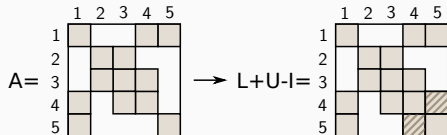
Building the elimination tree

- Permute matrix (to reduce fill-in) \mathbf{PAP}^T .
- Build filled matrix $\mathbf{A}_F = \mathbf{L} + \mathbf{L}^T$ where $\mathbf{PAP}^T = \mathbf{LL}^T$
- Transitive reduction of associated filled graph

Multifrontal approach

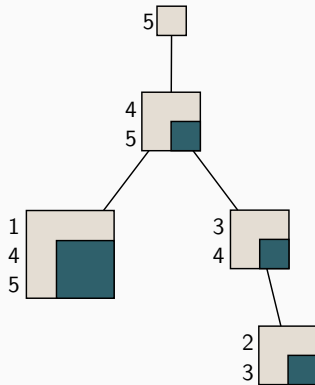
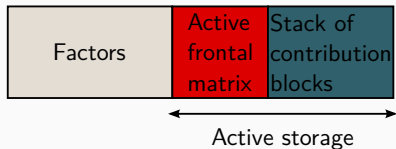
- Each column corresponds to a node of the tree
- (multifrontal) Each node k of the tree corresponds to the partial factorization of a **frontal matrix** whose row and column structure is that of column k of \mathbf{A}_F .

The multifrontal method [Duff & Reid '83]

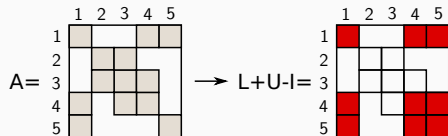


Storage is divided into two parts:

- Factors
- Active memory

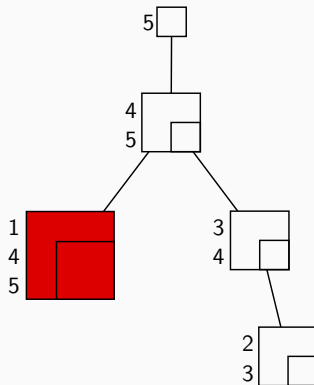


The multifrontal method [Duff & Reid '83]



Storage is divided into two parts:

- Factors
- Active memory



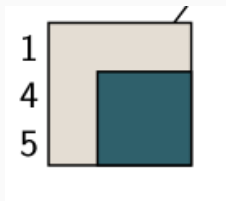
Elimination of variable 1

1. Assembly of the frontal matrix:

- a_{11} pivot, a_{14} and a_{15} unchanged,
- column 1

$$a_{41}^{(1)} = -\frac{a_{41}}{a_{11}}$$

$$a_{51}^{(1)} = -\frac{a_{51}}{a_{11}}$$



Terms of the frontal matrix are stored and will not be changed until the end of the factorization.

2. Compute the contribution matrix:

$$a_{44}^{(1)} = -a_{41}^{(1)} \times a_{14}$$

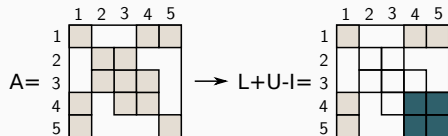
$$a_{45}^{(1)} = -a_{41}^{(1)} \times a_{15}$$

$$a_{54}^{(1)} = -a_{51}^{(1)} \times a_{14}$$

$$a_{55}^{(1)} = -a_{51}^{(1)} \times a_{15}$$

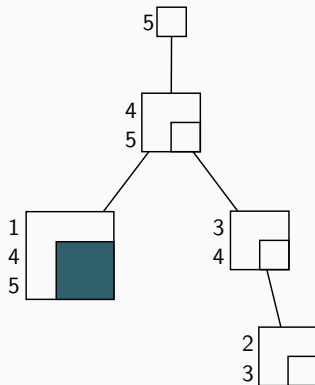
Terms of the contribution matrix are stored for later updates.

The multifrontal method [Duff & Reid '83]

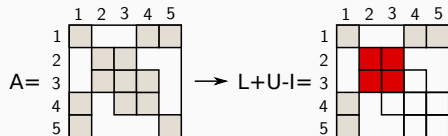


Storage is divided into two parts:

- Factors
- Active memory

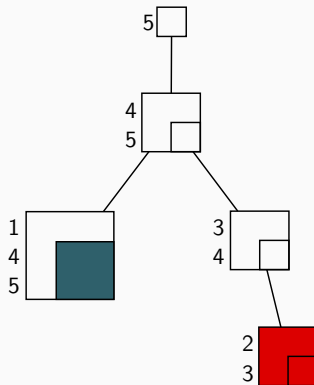


The multifrontal method [Duff & Reid '83]

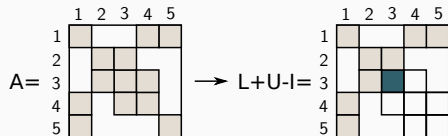


Storage is divided into two parts:

- Factors
- Active memory

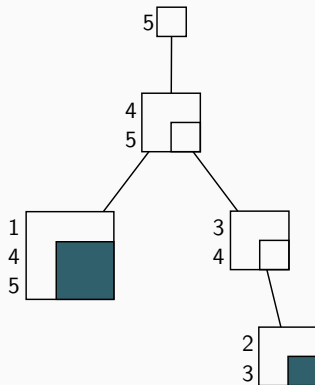


The multifrontal method [Duff & Reid '83]

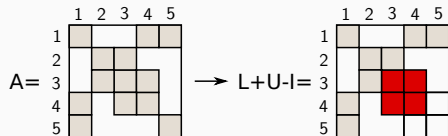


Storage is divided into two parts:

- Factors
- Active memory

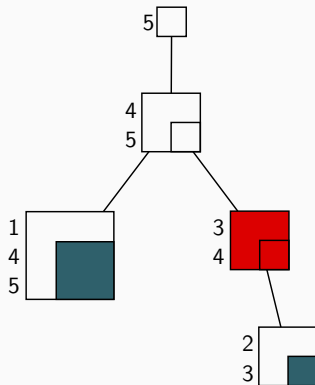


The multifrontal method [Duff & Reid '83]

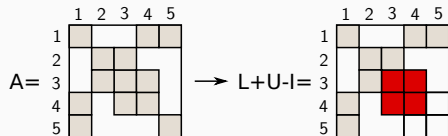


Storage is divided into two parts:

- Factors
- Active memory

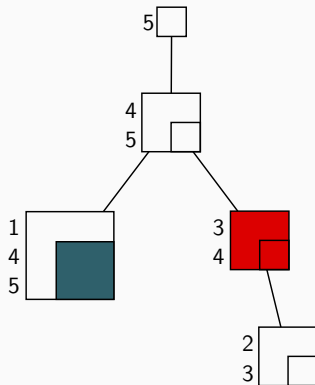


The multifrontal method [Duff & Reid '83]

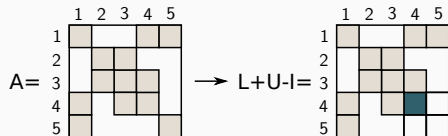


Storage is divided into two parts:

- Factors
- Active memory

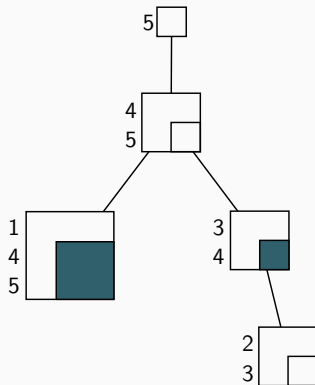


The multifrontal method [Duff & Reid '83]

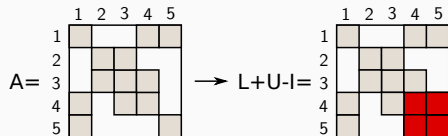


Storage is divided into two parts:

- Factors
- Active memory

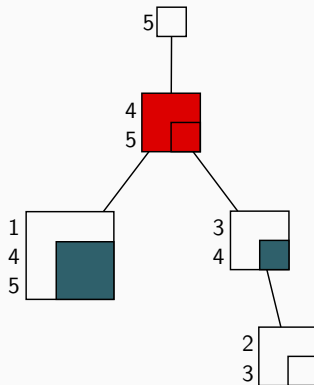


The multifrontal method [Duff & Reid '83]

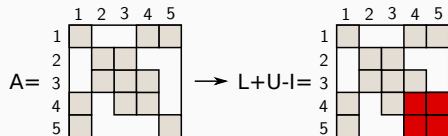


Storage is divided into two parts:

- Factors
- Active memory

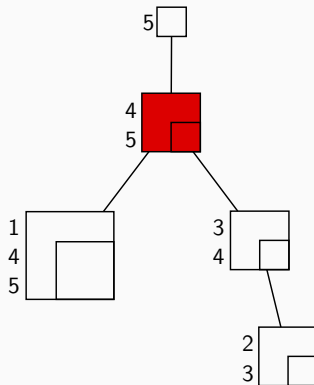


The multifrontal method [Duff & Reid '83]

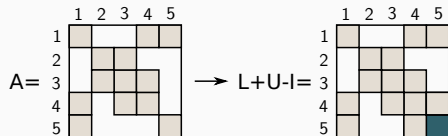


Storage is divided into two parts:

- Factors
- Active memory

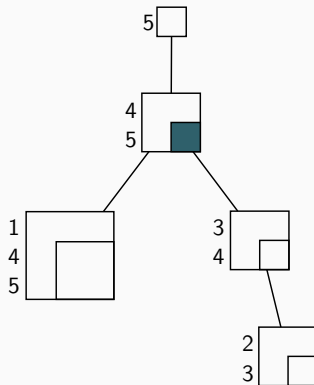


The multifrontal method [Duff & Reid '83]

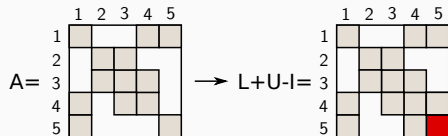


Storage is divided into two parts:

- Factors
- Active memory

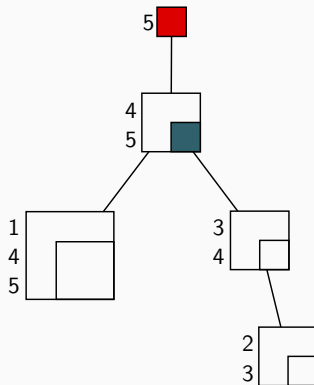


The multifrontal method [Duff & Reid '83]

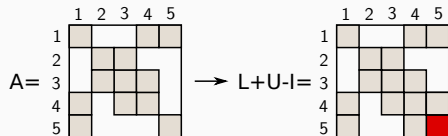


Storage is divided into two parts:

- Factors
- Active memory

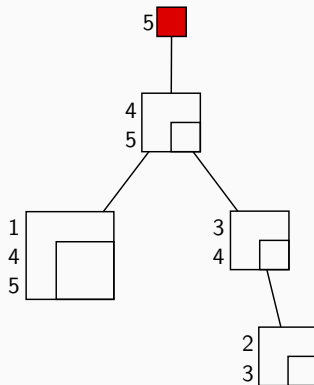


The multifrontal method [Duff & Reid '83]



Storage is divided into two parts:

- Factors
- Active memory



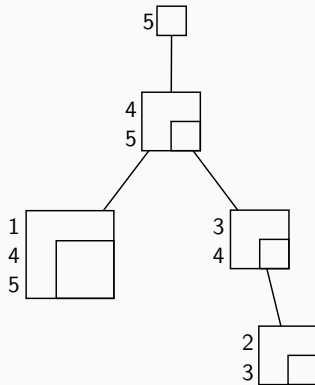
The multifrontal method [Duff & Reid '83]

- Factors are incompressible and usually scale fairly; they can optionally be written on disk.
- In sequential, the **traversal** that minimizes active memory is known [Liu'86].
- In parallel, active memory becomes dominant.

Example: share of active storage on the
AUDI matrix using MUMPS 4.10.0

1 processor: 11%

256 processors: 59%



Sparse Direct Solvers - Multifrontal Method

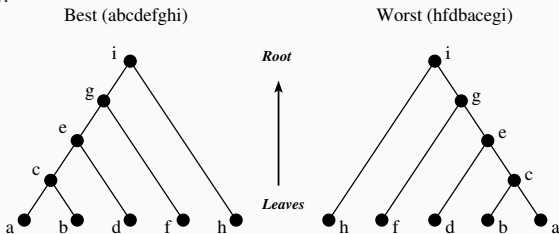
Postorderings and memory usage

Postorderings and memory usage

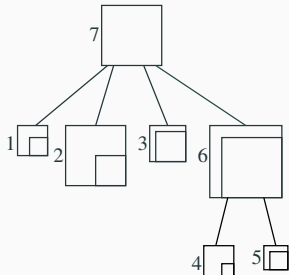
- Assumptions:
 - Tree processed from the leaves to the root
 - Parents processed as soon as all children have completed (postorder of the tree)
 - Each node produces and sends **temporary data** consumed by its father.
- **Exercise:** In which sense is a postordering-based tree traversal more interesting than a random topological ordering ?

Postorderings and memory usage

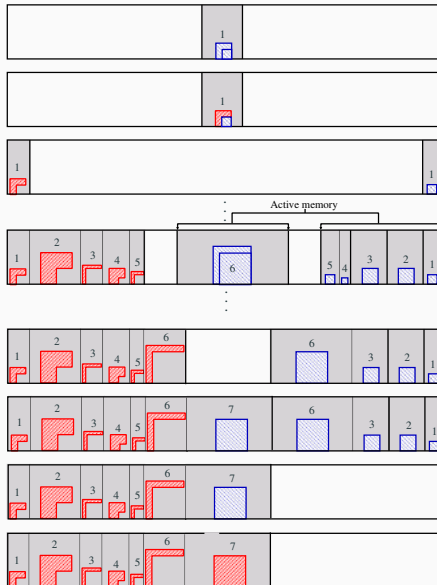
- Assumptions:
 - Tree processed from the leaves to the root
 - Parents processed as soon as all children have completed (postorder of the tree)
 - Each node produces and sends **temporary data** consumed by its father.
- Exercise:** In which sense is a postordering-based tree traversal more interesting than a random topological ordering ?
- Furthermore, memory usage also depends on the postordering chosen:



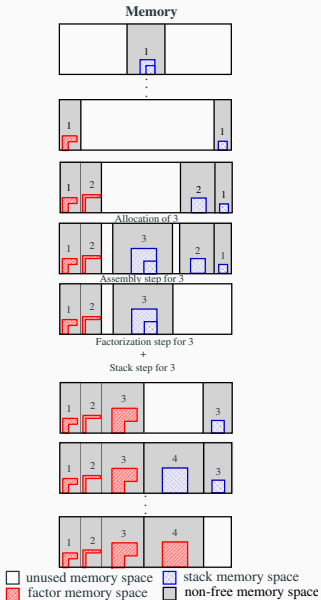
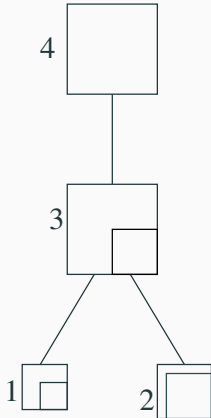
Example 1: Processing a wide tree



Memory

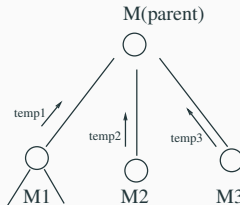


Example 2: Processing a deep tree



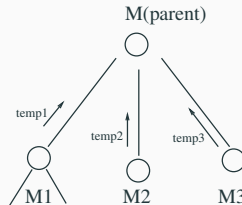
Minimizing the active memory: modelization of the problem

- M_i : memory peak for complete subtree rooted at i ,
- $temp_i$: temporary memory produced by node i ,
- m_{parent} : memory for storing the parent.



Minimizing the active memory: modelization of the problem

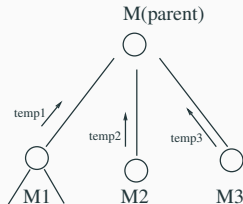
- M_i : memory peak for complete subtree rooted at i ,
- $temp_i$: temporary memory produced by node i ,
- m_{parent} : memory for storing the parent.



$$M_{parent} = \max(\max_{j=1}^{nbchildren} (M_j + \sum_{k=1}^{j-1} temp_k), m_{parent} + \sum_{j=1}^{nbchildren} temp_j) \quad (24)$$

Minimizing the active memory: modelization of the problem

- M_i : memory peak for complete subtree rooted at i ,
- $temp_i$: temporary memory produced by node i ,
- m_{parent} : memory for storing the parent.



$$M_{parent} = \max\left(\max_{j=1}^{nbchildren} \left(M_j + \sum_{k=1}^{j-1} temp_k \right), m_{parent} + \sum_{j=1}^{nbchildren} temp_j \right) \quad (24)$$

Objective: order the children to minimize M_{parent}

Memory-minimizing schedules

Theorem

[Liu,86] *The minimum of $\max_j(x_j + \sum_{i=1}^{j-1} y_i)$ is obtained when the sequence (x_i, y_i) is sorted in decreasing order of $x_i - y_i$.*

Corollary

An optimal child sequence is obtained by rearranging the children nodes in decreasing order of $M_i - temp_i$.

Interpretation: At each level of the tree, child with relatively large peak of memory in its subtree (M_i large with respect to $temp_i$) should be processed first.

⇒ Apply on complete tree starting from the leaves
(or from the root with a recursive approach)

Sparse Direct Solvers - Multifrontal Method

Concluding remarks

Concluding remarks

- Key parameters in selecting a method

1. Functionalities of the solver
2. Characteristics of the matrix
 - Numerical properties and pivoting.
 - Symmetric or general
 - Pattern and density
3. Preprocessing of the matrix
 - Scaling
 - Reordering for minimizing fill-in
4. Target computer (architecture)

- Substantial gains can be achieved with an adequate solver: in terms of numerical precision, computing and storage

- Good knowledge of matrix and solvers

- Many challenging problems

- Active research area