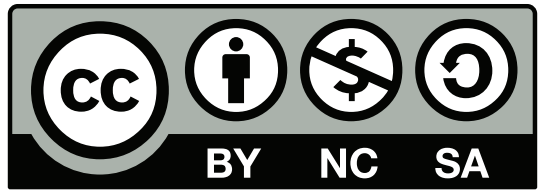
Résolution de systèmes linéaires

Enseignement spécialisé "éléments finis" – S6133/5

Christophe Bovet (christophe.bovet@onera.fr)

Onera – The French Aerospace Lab F-92322 Chatillon, France



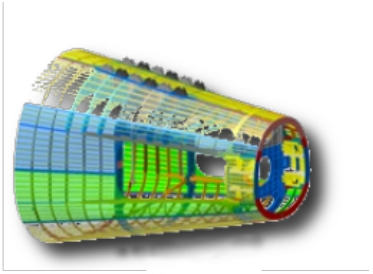

```
# Pour récupérer les supports de cours
> git clone https://gitlab.com/chrb1/ef.git
```

Contexte

Modélisation

$$\begin{cases} \mathbf{f}_{acc} = \mathbf{f}_{int} + \mathbf{f}_{ext} \\ \boldsymbol{\sigma} = \mathbf{f}(\boldsymbol{\epsilon}) \end{cases}$$

Discrétisation



Problème linéaire ?

oui

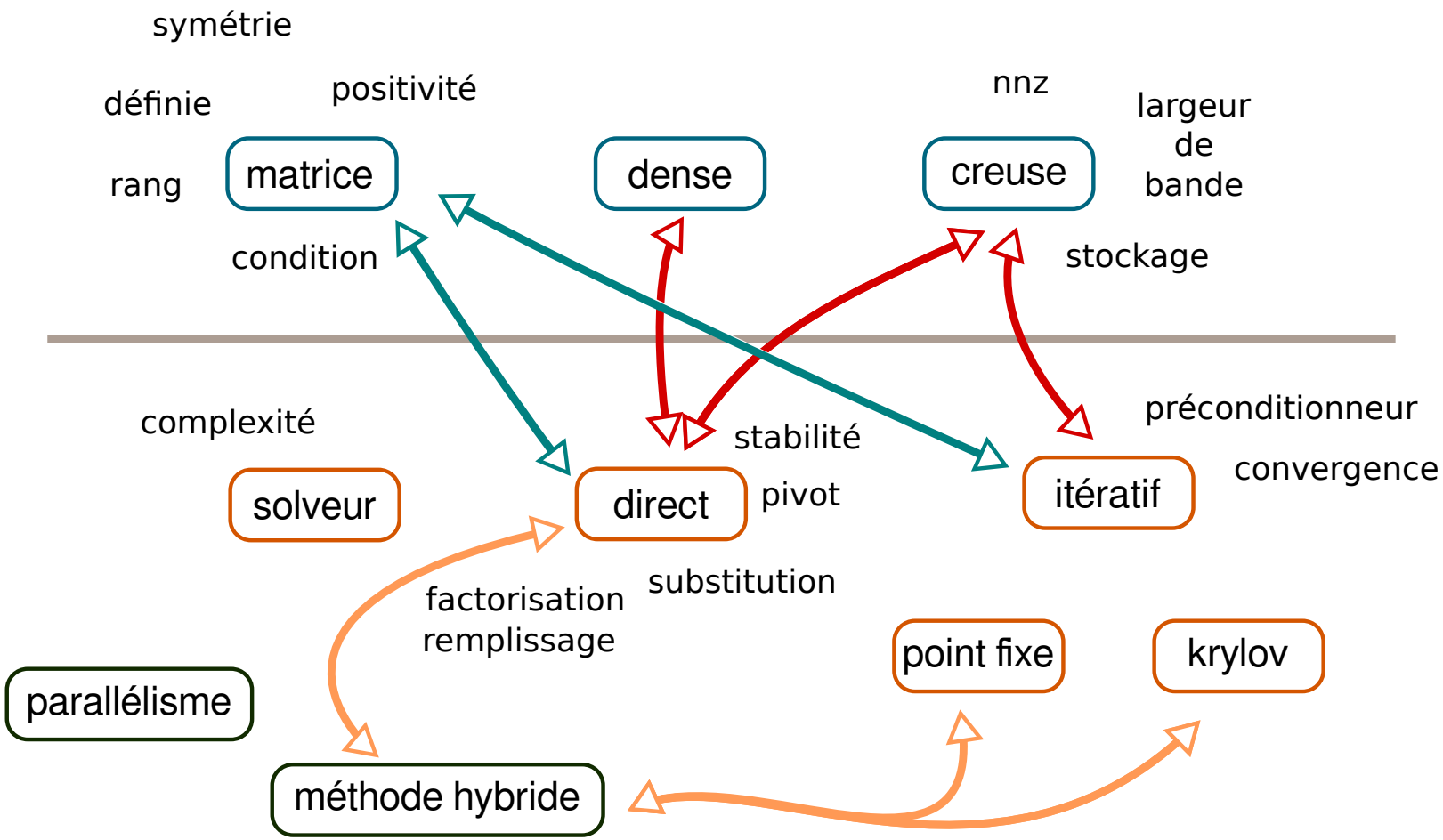
non

Système linéaire $\mathbf{Ax} = \mathbf{b}$

Méthode de Newton

Résoudre efficacement les grands systèmes linéaires issus de la discrétisation EF est indispensable

Schéma synoptique

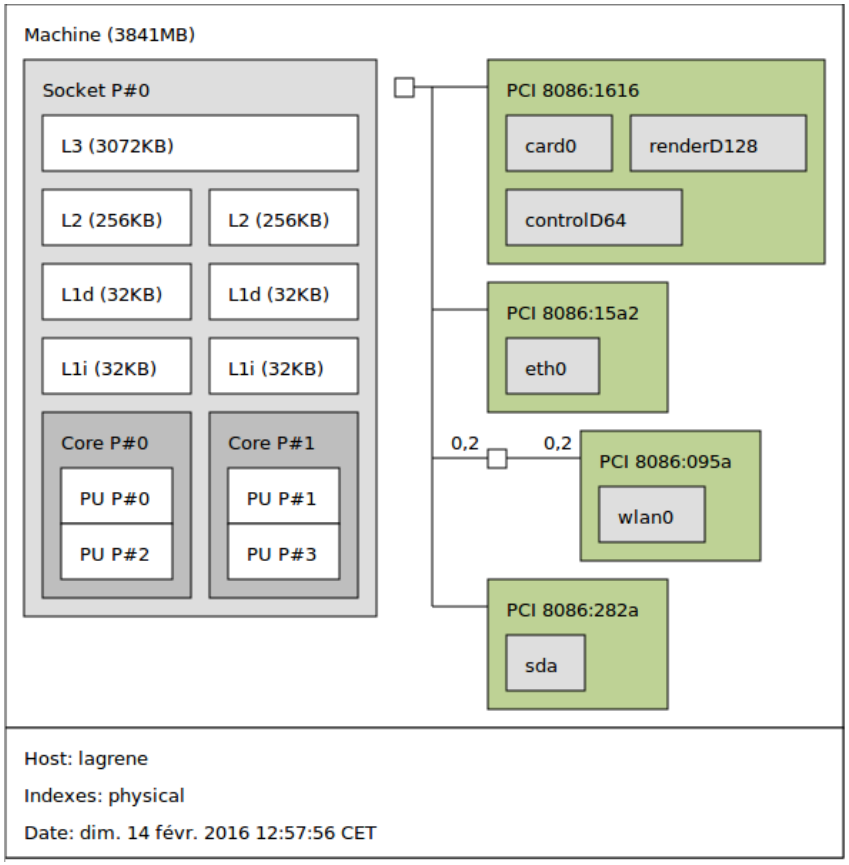


Objectifs

- Avoir un aperçu des méthodes de résolutions existantes
- Connaître les avantages & défauts des $\neq$ méthodes
- Être sensibilisé aux problématiques du monde réel : algèbre linéaire numérique (complexité, stabilité & précision, parallélisme, . . .)
- Avoir des notions des mots clefs du schéma précédent

Architecture d'un ordinateur

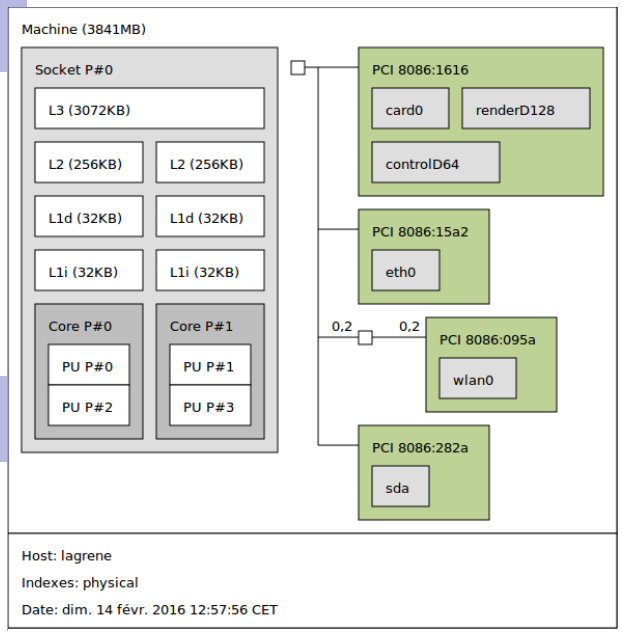
- **Unités de calculs** :
  noeud / processeur / cœur
- **Mémoire** :
  cache L1/L2/L3, RAM,
  disque dur
- **Communication** :
  bus / réseau (gigabit,
  infiniband)



Architecture d'un ordinateur

Mémoire partagée (symmetric multiprocessing)

Échange rapide (accès direct)
Besoin de protéger les données (mutex/semaphore)
⇒ multithreading, protocole OpenMP



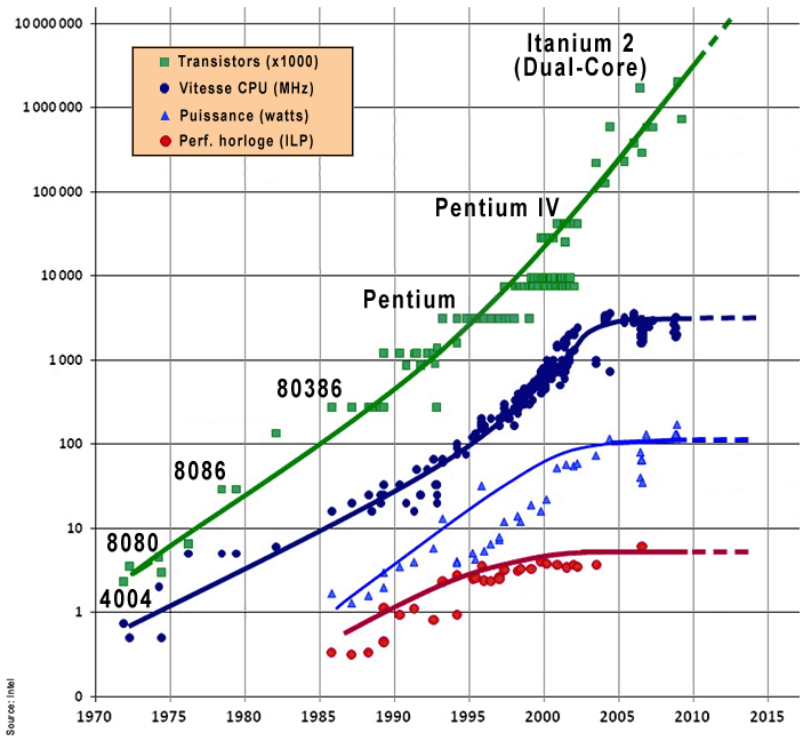
Mémoire séparée

Données protégées
Il faut s'occuper d'échanger les données.
⇒ Protocole MPI (message passing interface)

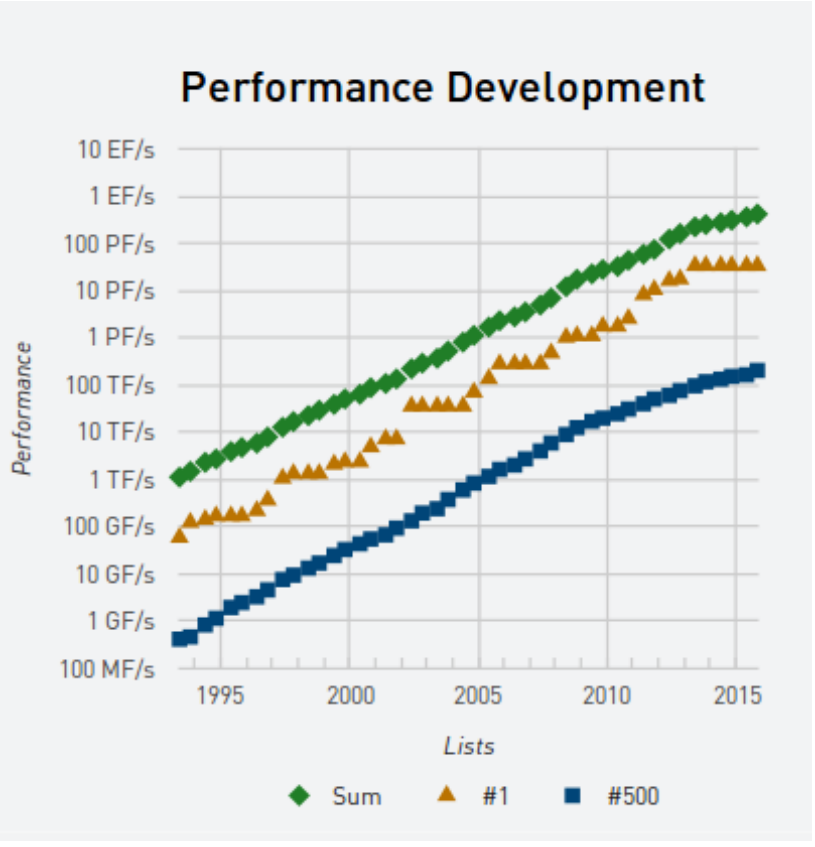
Architectures actuelles

Architectures mixtes plusieurs nœuds en réseau avec plusieurs cœurs à mémoire partagée.

Constat sur les architectures



Source : www.astrosurf.com/luxorion/loi-moore.html



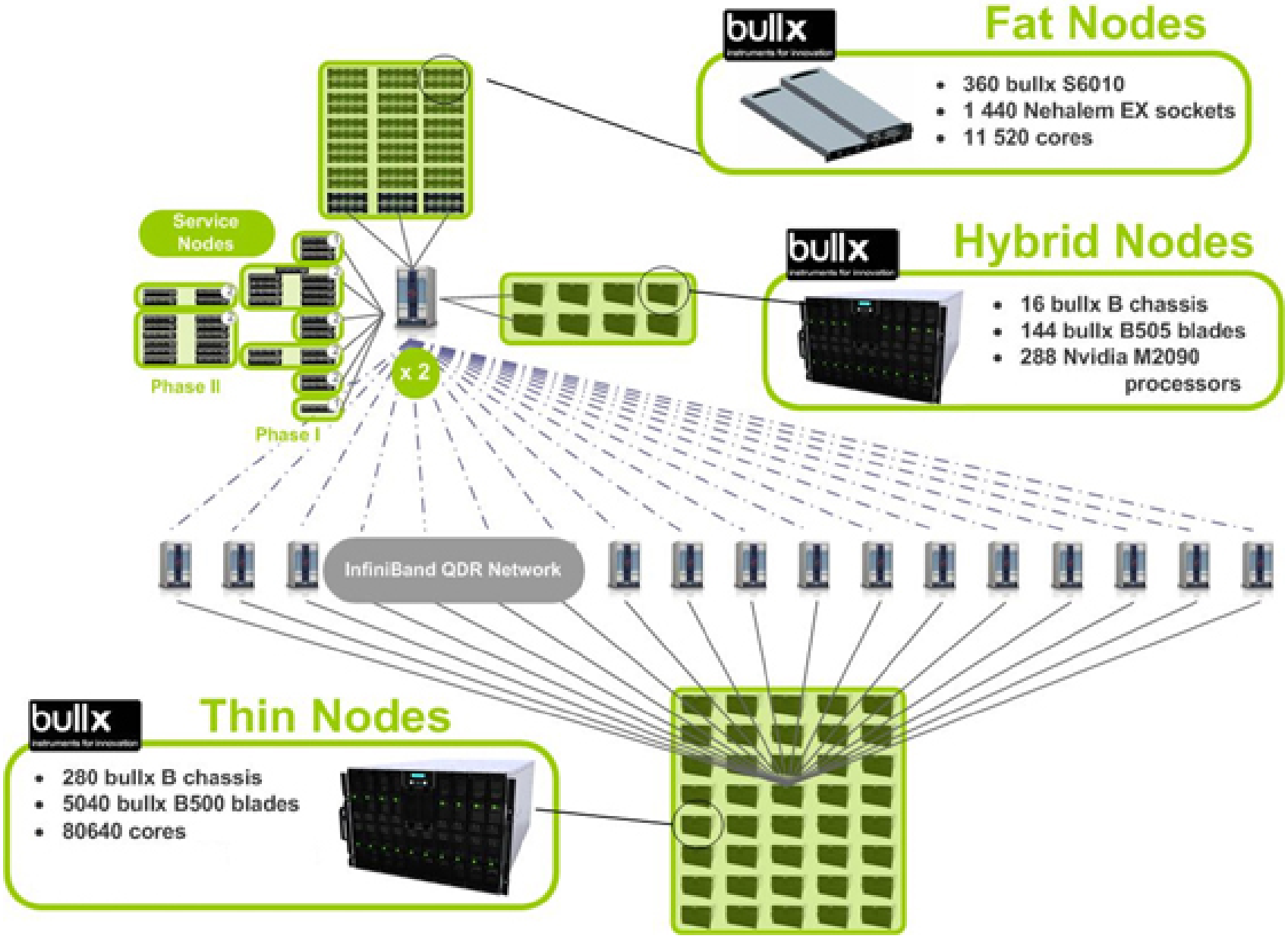
Source : www.top500.org

Flop/s = floating-point operation per second, ($péta = 10^{15}$)

Flop/s théorique = cœurs $\times$ fréquence $\times$ (Flop/cycle)

Présentation d'un Cluster

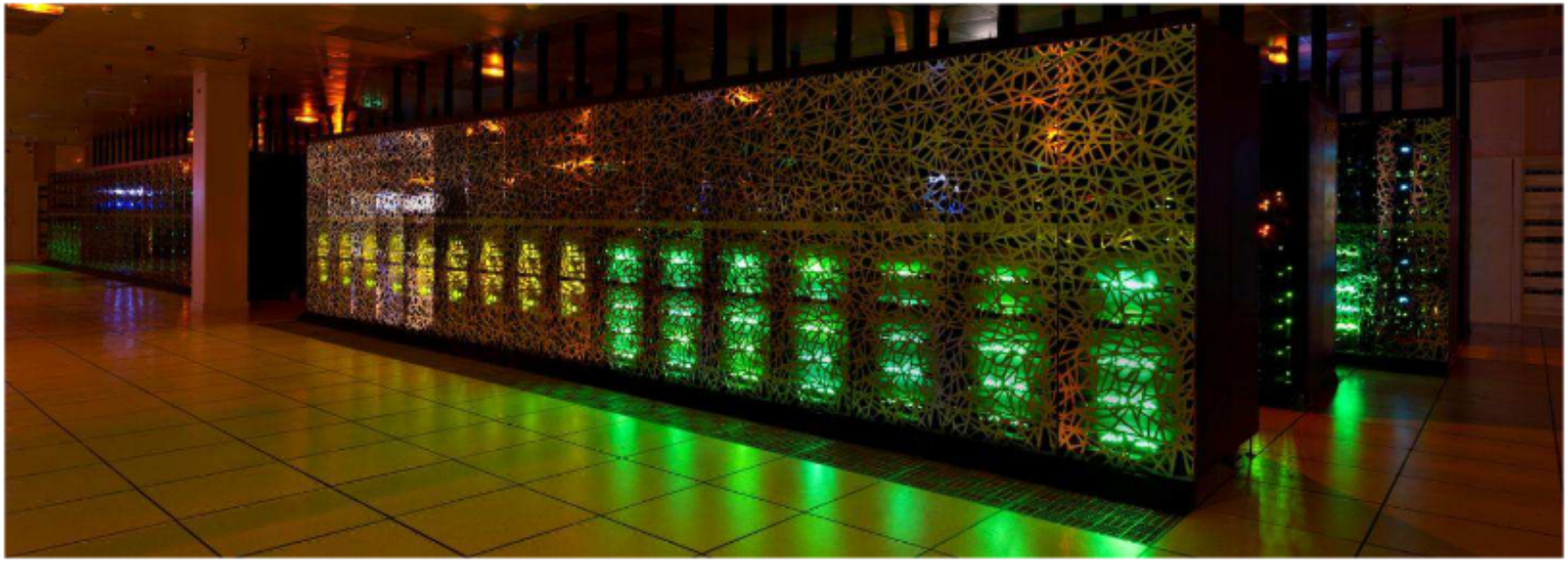
Le calculateur Curie du TGCC



Présentation d'un Cluster

Le calculateur Curie du TGCC

Fat nodes	Thin nodes
360 BullX-S6010	5040 BullX B500
Intel NH EX 2,26 Ghz	Intel Sandy Bridge 2,7Ghz
11 520 cœurs	80 640 cœurs
128 cœurs/nœud	16 cœurs/nœuds
512 Go/nœud	4 Go/cœurs
105 TFlops	1 740 TFlops



Plan

Plan

Contexte

- Équivalence système linéaire & calcul matriciel :

$$\begin{array}{cccc} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 & & & \\ \vdots \quad\quad \vdots & & \vdots \quad\quad \vdots & \Leftrightarrow \quad \boldsymbol{Ax} = \boldsymbol{b} \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n & & & \end{array}$$

- $\boldsymbol{A}$ et $\boldsymbol{b}$ (2^{nd} membre, *right hand side)* sont donnés
- On suppose $\boldsymbol{A}$ réelle d'ordre n et inversible ($\det(\boldsymbol{A}) \neq 0$)
- On cherche à calculer $\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}$, comment faire ?
- Vocabulaire : $\boldsymbol{A}$ est dite **dense** s'il y a peu de a_{ij} nuls sinon elle est creuse

Rappels

- $\boldsymbol{A}$ est symétrique si $\boldsymbol{A}^\top = \boldsymbol{A}$
- $\boldsymbol{A}$ est positive si $\boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x} \geq 0 \quad \forall \boldsymbol{x} \in \mathbb{R}^n$
- $\boldsymbol{A}$ est définie si pour $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x} = 0 \Rightarrow \boldsymbol{x} = \boldsymbol{0}$
- Norme matricielle :

$$\|\boldsymbol{A}\| = \max_{\|\boldsymbol{x}\|=1} \|\boldsymbol{A}\boldsymbol{x}\| = \max_{\|\boldsymbol{x}\|\neq 0} \frac{\|\boldsymbol{A}\boldsymbol{x}\|}{\|\boldsymbol{x}\|}$$

- Condition de $\boldsymbol{A}$ (inversible) (*condition number*) :

$$\kappa(\boldsymbol{A}) = \|\boldsymbol{A}\| \left\|\boldsymbol{A}^{-1}\right\| = \left(\max_{\|\boldsymbol{x}\|=1} \|\boldsymbol{A}\boldsymbol{x}\|\right) / \left(\min_{\|\boldsymbol{y}\|=1} \|\boldsymbol{A}\boldsymbol{y}\|\right)$$

On a toujours $\kappa(\boldsymbol{A}) \geq 1$

Élimination de Gauss

- Opération sur les lignes jusqu'à obtenir un syst. tri. sup. $\rightarrow$ succession de systèmes lin. $(\boldsymbol{A}^{(k)}, \boldsymbol{b}^{(k)})_k$ on pose $\boldsymbol{A}^{(1)} = \boldsymbol{A} \quad \boldsymbol{b}^{(1)} = \boldsymbol{b}$
- On pose $m_{i1} = a_{i1}^{(1)} / a_{11}^{(1)}$, on réalise ligne $i - m_{i1}$ ligne 1 (hyp $a_{11} \neq 0$.)

$$\begin{array}{l} a_{11}^{(1)} x_1 + a_{12}^{(1)} x_2 + \cdots + a_{1n}^{(1)} x_n = b_1^{(1)} \\ a_{21}^{(1)} x_1 + a_{22}^{(1)} x_2 + \cdots + a_{2n}^{(1)} x_n = b_2^{(1)} \\ \vdots \qquad\quad \vdots \qquad\qquad\quad \vdots \qquad\quad \vdots \\ a_{n1}^{(1)} x_1 + a_{n2}^{(1)} x_2 + \cdots + a_{nn}^{(1)} x_n = b_n^{(1)} \end{array} \Rightarrow \begin{array}{l} a_{11}^{(1)} x_1 + a_{12}^{(1)} x_2 + \cdots + a_{1n}^{(1)} x_n = b_1^{(1)} \\ a_{22}^{(2)} x_2 + \cdots + a_{2n}^{(2)} x_n = b_2^{(2)} \\ \vdots \qquad\qquad\quad \vdots \qquad\quad \vdots \\ a_{n2}^{(2)} x_2 + \cdots + a_{nn}^{(2)} x_n = b_n^{(2)} \end{array}$$

- À l'étape k

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - m_{ik} a_{kj}^{(k)} = a_{ij}^{(k)} - \frac{a_{ik}^{(k)} a_{kj}^{(k)}}{a_{kk}^{(k)}} \quad i, j = k+1, \ldots, n$$

$$\boldsymbol{b}_i^{(k+1)} = \boldsymbol{b}_i^{(k)} - m_{ik} \boldsymbol{b}_k^{(k)} \quad i = k+1, \ldots, n \qquad a_{kk}^{(k)} \text{ est appelé } \textbf{pivot}$$

- À l'étape n on a un système tri. sup., $\boldsymbol{A}^{(n)} \boldsymbol{x} = \boldsymbol{U} \boldsymbol{x} = \boldsymbol{b}^{(n)}$

Élimination de Gauss : changement de pivot

- Si $a_{kk}^{(k)} = 0$, il faut permuter des lignes, on parle de changement de pivot

$$\begin{array}{cccc} a_{11}^{(1)}x_1 + a_{12}^{(1)}x_2 + \cdots + a_{1n}^{(1)}x_n = b_1^{(1)} \\ a_{21}^{(1)}x_1 + a_{22}^{(1)}x_2 + \cdots + a_{2n}^{(1)}x_n = b_2^{(1)} \\ \vdots \quad\quad \vdots \quad\quad\quad\quad \vdots \quad\quad \vdots \\ a_{n1}^{(1)}x_1 + a_{n2}^{(1)}x_2 + \cdots + a_{nn}^{(1)}x_n = b_n^{(1)} \end{array} \Rightarrow \begin{array}{cccc} a_{11}^{(1)}x_1 + a_{12}^{(1)}x_2 + \cdots + a_{1n}^{(1)}x_n = b_1^{(1)} \\ a_{22}^{(2)}x_2 + \cdots + a_{2n}^{(2)}x_n = b_2^{(2)} \\ \vdots \quad\quad\quad\quad \vdots \quad\quad \vdots \\ a_{n2}^{(2)}x_2 + \cdots + a_{nn}^{(2)}x_n = b_n^{(2)} \end{array}$$

- Remarque : la condition $a_{ii} \neq 0$ n'est pas suffisante pour empêcher l'apparition de pivots nuls.
- Exemple :

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

Élimination de Gauss : changement de pivot

- Si $a_{kk}^{(k)} = 0$, il faut permuter des lignes, on parle de changement de pivot

$$\begin{array}{l} a_{11}^{(1)}x_1 + a_{12}^{(1)}x_2 + \cdots + a_{1n}^{(1)}x_n = b_1^{(1)} \\ a_{21}^{(1)}x_1 + a_{22}^{(1)}x_2 + \cdots + a_{2n}^{(1)}x_n = b_2^{(1)} \\ \vdots \qquad\quad \vdots \qquad\qquad \vdots \qquad\quad \vdots \\ a_{n1}^{(1)}x_1 + a_{n2}^{(1)}x_2 + \cdots + a_{nn}^{(1)}x_n = b_n^{(1)} \end{array} \Rightarrow \begin{array}{l} a_{11}^{(1)}x_1 + a_{12}^{(1)}x_2 + \cdots + a_{1n}^{(1)}x_n = b_1^{(1)} \\ a_{22}^{(2)}x_2 + \cdots + a_{2n}^{(2)}x_n = b_2^{(2)} \\ \vdots \qquad\quad \vdots \qquad\quad \vdots \\ a_{n2}^{(2)}x_2 + \cdots + a_{nn}^{(2)}x_n = b_n^{(2)} \end{array}$$

- Remarque : la condition $a_{ii} \neq 0$ n'est pas suffisante pour empêcher l'apparition de pivots nuls.
- Exemple :

$$\boldsymbol{A} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 7 & 8 & 9 \end{bmatrix} \quad \Rightarrow \quad \boldsymbol{A}^{(2)} = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 0 & -1 \\ 0 & -6 & -12 \end{bmatrix}$$

Élimination de Gauss : changement de pivot

- Si $a_{kk}^{(k)} = 0$, il faut permuter des lignes, on parle de changement de pivot

$$\begin{array}{l} a_{11}^{(1)}x_1 + a_{12}^{(1)}x_2 + \cdots + a_{1n}^{(1)}x_n = b_1^{(1)} \\ a_{21}^{(1)}x_1 + a_{22}^{(1)}x_2 + \cdots + a_{2n}^{(1)}x_n = b_2^{(1)} \\ \vdots \qquad\quad \vdots \qquad\qquad \vdots \qquad\quad \vdots \\ a_{n1}^{(1)}x_1 + a_{n2}^{(1)}x_2 + \cdots + a_{nn}^{(1)}x_n = b_n^{(1)} \end{array} \Rightarrow \begin{array}{l} a_{11}^{(1)}x_1 + a_{12}^{(1)}x_2 + \cdots + a_{1n}^{(1)}x_n = b_1^{(1)} \\ a_{22}^{(2)}x_2 + \cdots + a_{2n}^{(2)}x_n = b_2^{(2)} \\ \vdots \qquad\quad \vdots \qquad\quad \vdots \\ a_{n2}^{(2)}x_2 + \cdots + a_{nn}^{(2)}x_n = b_n^{(2)} \end{array}$$

- Nombre d'opérations pour le passage de $\boldsymbol{A}^{(1)}$ à $\boldsymbol{A}^{(2)}$
  - $(n-1)^2$ multiplications, $(n-1)^2$ additions, $(n-1)$ divisions
- Pour toutes les étapes
  - $\sum_{k=1}^{n}(n-k)^2 = \sum_{k=1}^{n-1} k^2 = n^3/3 + n^2/2 + n/6$
  - $\sum_{k=1}^{n} k = n(n+1)/2 = n^2 + \ldots$
- Complexité algorithmique : $\simeq 2/3n^3 + o\left(n^3\right)$ flop

Résolution de systèmes triangulaires

- On arrive au système triangulaire supérieur

$$\begin{aligned} u_{11}x_1 + u_{12}x_2 + \cdots + u_{1n}x_n &= b_1 \\ u_{22}x_2 + \cdots + u_{2n}x_n &= b_2 \\ \ddots \quad \vdots \quad &\vdots \\ u_{nn}x_n &= b_n \end{aligned}$$

- Résolution par substitution rétrograde *(backward subst., row version)*

$$x_n = \frac{b_n}{u_{nn}}$$

$$x_i = \frac{1}{u_{ii}}\left(b_i - \sum_{j=i+1}^{n} u_{ij}x_j\right) \ , \ i = n-1, \ldots, 1$$

Résolution de systèmes triangulaires

- Résolution par substitution rétrograde *(backward subst., row version)*

$$x_n = \frac{b_n}{u_{nn}}$$
$$x_i = \frac{1}{u_{ii}}(b_i - \sum_{j=i+1}^{n} u_{ij}x_j) \ , \ i = n-1, \ldots, 1$$

- Nombre d'opérations
  - Pour x_i $(n-i-1)$ $\times$, 1 $\div$ $\Rightarrow$ pour tous les x_i $\sum_{i=1,n} i = n(n+1)/2$
  - Pour x_i $(n-i-2)$ $+$, 1 $-$. pour tous les x_i $\sum_{i=1,n-1} i = n(n-1)/2$
- Complexité algorithmique : n^2 flop

Plan

Theorem

Soit $\boldsymbol{A}$ inversible, l'élimination de Gauss donne

$$\boldsymbol{PA} = \boldsymbol{LU}$$

où $\boldsymbol{P}$ est une matrice de permutation, $\boldsymbol{L}$ est triangulaire inférieure à diagonale unitaire et $\boldsymbol{U}$ est triagulaire supérieure.

- Remarque : Matrice de permutation

$$\boldsymbol{P} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \Rightarrow \quad \boldsymbol{P} \begin{pmatrix} \ell_1 \\ \ell_2 \\ \ell_3 \end{pmatrix} = \begin{pmatrix} \ell_2 \\ \ell_1 \\ \ell_3 \end{pmatrix}$$

Élimination de Gauss & factorisation $\boldsymbol{LU}$

- Remarque : suivant les propriétés de $\boldsymbol{A}$, l'absence de pivot nul est garantie (matrice à diagonale dominante, matrice SDP)
- En l'absence de pivot nul, l'élimination de Gauss donne

$$\boldsymbol{A} = \boldsymbol{LU} \quad \text{avec} \quad \boldsymbol{L} = \begin{pmatrix} 1 & & & \\ m_{21} & 1 & & \\ \vdots & \ddots & \ddots & \\ m_{n1} & \dots & m_{nn-1} & 1 \end{pmatrix} \quad \text{et} \quad \boldsymbol{U} = \begin{pmatrix} u_{11} & \dots & \dots & u_{1n} \\ & u_{22} & \dots & u_{2n} \\ & & \ddots & \vdots \\ & & & u_{nn} \end{pmatrix}$$

- Étape k de l'élimination de Gauss

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} a_{kj}^{(k)} = a_{ij}^{(k)} - m_{ik} a_{kj}^{(k)} \quad i,j = k+1, \dots, n$$

- On pose $\boldsymbol{M}_k = \boldsymbol{I}_n - \boldsymbol{m}_k \boldsymbol{e}_k^\top$ avec $\boldsymbol{m}_k = [0, \dots, 0, m_{k+1,k}, \dots, m_{n,k}]^\top$

$$\boldsymbol{A}^{(k+1)} = \boldsymbol{M}_k \boldsymbol{A}^{(k)}$$

Élimination de Gauss & factorisation LU

Esprit de la démonstration :

- On a
$$(M_{n-1}M_{n-2}\cdots M_1)A = U$$
- L triangulaire inférieure car produit de matrice tri. inf.
$$A = (M_{n-1}M_{n-2}\cdots M_1)^{-1}U = LU$$
- On remarque $M_k^{-1} = I_n + m_k e_k^\top$
- En explicitant les M_i^{-1}
$$L = (I_n + m_1 e_1^\top)\ldots(I_n + m_{n-1}e_{n-1}^\top)$$
- Or $(m_i e_i^\top)(m_j e_j^\top) = 0 \quad \forall i \neq j$
- Au final
$$L = (I_n + \sum_i m_i e_i^\top)$$

Élimination de Gauss & factorisation $\boldsymbol{LU}$

En pratique, on veut souvent résoudre plusieurs second membres. La substitution à la volée de $\boldsymbol{b}$ n'est pas souhaitée

1. Calcul de la factorisation $\boldsymbol{LU}$

$$\boldsymbol{LU} = \boldsymbol{PA}$$

Alors

$$\boldsymbol{LUx} = \boldsymbol{PAx} = \boldsymbol{Pb}$$

2. Résolution d'un système tri. inf. à diagonale unitaire

$$\boldsymbol{Ly} = \boldsymbol{Pb} \qquad \text{(descente, } \textit{forward substitution}\text{)}$$

3. Résolution d'un système tri. supérieur

$$\boldsymbol{Ux} = \boldsymbol{y} \qquad \text{(remontée, } \textit{backward substitution}\text{)}$$

Élimination de Gauss & factorisation $\boldsymbol{LU}$

Theorem (Existence et unicité)

Soit la matrice $\boldsymbol{A} \in \mathbb{R}^{n \times n}$. La factorisation $\boldsymbol{LU}$ existe et est unique ssi toutes les sous-matrices $\boldsymbol{A}_i$ d'ordre $i = 1, \ldots, n-1$ sont inversibles.

- Si $\boldsymbol{A}$ est inversible, on peut toujours se ramener au cas ci-dessus grâce à des permutations.

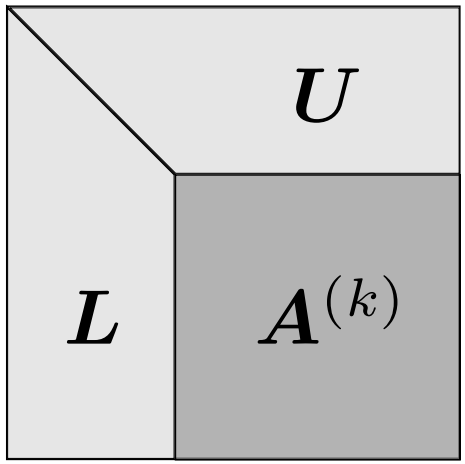
Complexité

- Coût du calcul de $\boldsymbol{LU} \simeq 2/3n^3 + o\left(n^3\right)$
- Coût de résolution d'un système triangulaire $\simeq n^2$
- Coût de stockage, en dense il suffit d'un vecteur suppl. pour $\boldsymbol{P}$.

Élimination de Gauss & factorisation *LU*

Algorithme LU de base sans changement de pivot

```
def lu_kji(A):
  """ Factorisation LU (ver. kji) sans pivot """
  n, m = A.shape
  if(n != m):
    raise ValueError("Only square matrix allowed")
  for k in xrange(0, n):
    if abs(A[k,k]) < 1.0e-20:
      raise ArithmeticError("Null pivot")
    A[k+1:,k] /= A[k,k] # i = k+1:n
    for j in xrange(k+1,n):
      A[k+1:,j] -= A[k+1:, k] * A[k,j]
  return A
```



Remarques : raisonnement simplifié

- Arithmétique exacte (spécifités du calcul sur ordinateur)
- Algèbre linéaire dense (spécifités des EDP)

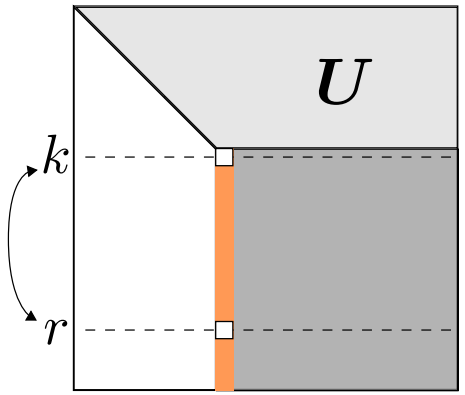
Pivot partiel $\boldsymbol{PA} = \boldsymbol{LU}$

- Exemple ($k \ll 1$)

$$\boldsymbol{A} = \begin{bmatrix} k & 1 & 0 \\ 10 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \boldsymbol{x}_{ex} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \quad \Rightarrow \quad \boldsymbol{b} = \begin{bmatrix} k+1 \\ 11 \\ 1 \end{bmatrix}$$

- Au lieu de se satisfaire de $a_{kk}^{(k)} \neq 0$, on permute la ligne i qui maximise $|a_{ik}^{(k)}|, i \geq k$

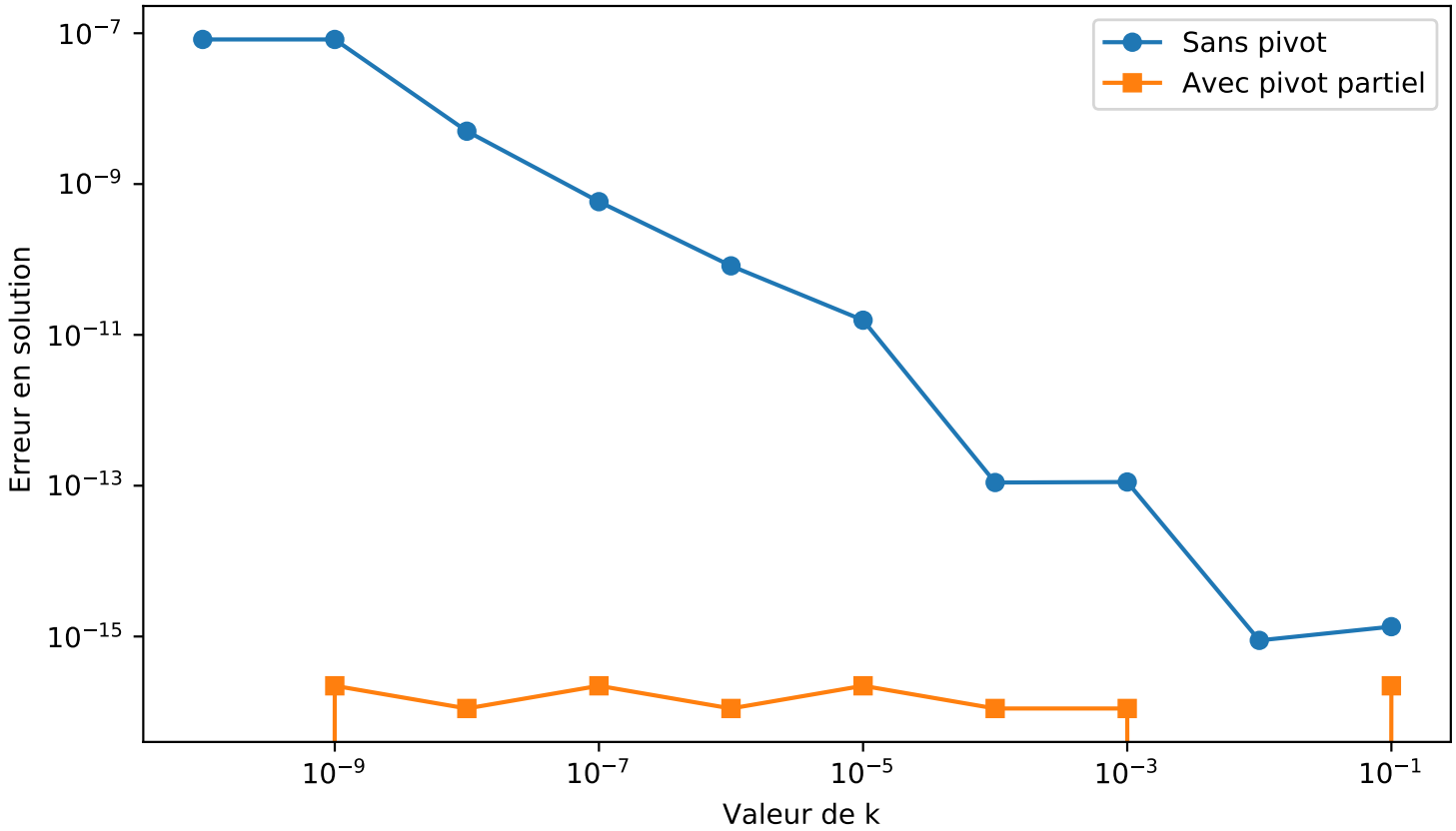
- Requiert $(n-k)$ tests
- N'affecte pas les lignes de $\boldsymbol{U}$ déjà factorisées



Élimination de Gauss et calculs à virgule flottante

- Exemple ($k \ll 1$)

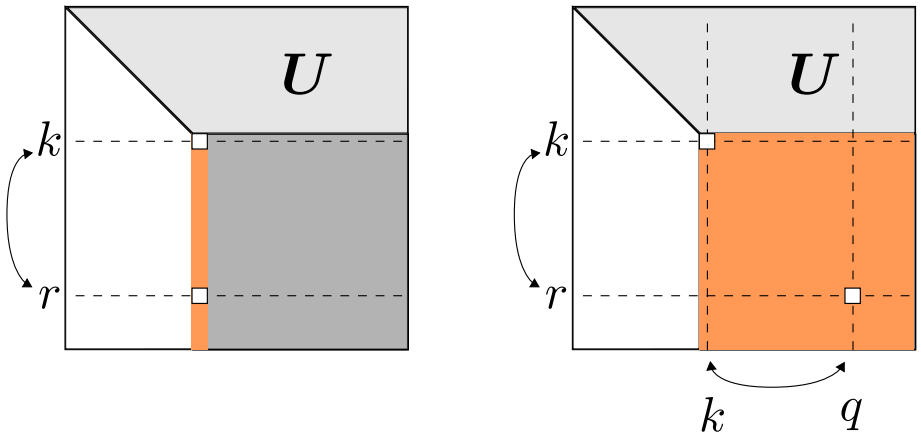
$$\boldsymbol{A} = \begin{bmatrix} k & 1 & 0 \\ 10 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \boldsymbol{x}_{ex} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \quad \Rightarrow \quad \boldsymbol{b} = \begin{bmatrix} k+1 \\ 11 \\ 1 \end{bmatrix}$$



Pivot total $\boldsymbol{PAQ = LU}$

- Si le pivot partiel n'est pas suffisant, il existe le pivotage total (ou complet).

$$\boldsymbol{PAQ}^\top = \boldsymbol{LU}$$



- Requiert $(n-k)^2$ tests

Calculs à virgule flottante : le scaling

- La présence de grands pivots ne suffit pas à garantir la précision de la solution.
- Exemple :

$$\boldsymbol{A} = \begin{bmatrix} -10^9 & 10^8 & 10^8 \\ 10^8 & 10^{-6} & 0 \\ 10^8 & 0 & 1 \end{bmatrix} \quad \boldsymbol{x}_{ex} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \Rightarrow \boldsymbol{b} = \boldsymbol{A}\boldsymbol{x} \quad \boldsymbol{x}_{num} = \begin{bmatrix} 1. \\ 0.9981378 \\ 1.0018622 \end{bmatrix}$$

- Le scaling consiste a utiliser des matrices diagonales $\boldsymbol{D}_1, \boldsymbol{D}_2$ inversible pour uniformiser les ordres de grandeurs

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \quad \Leftrightarrow \quad \begin{cases} \boldsymbol{D}_1 \boldsymbol{A} \boldsymbol{D}_2 \boldsymbol{x}^\star = \boldsymbol{D}_1 \boldsymbol{b} \\ \boldsymbol{x} = \boldsymbol{D}_2 \boldsymbol{x}^\star \end{cases}$$

Calculs à virgule flottante : le raffinement itératif

- Pour les matrices mal conditionnées, la méthode du raffinement itératif permet d'améliorer la qualité de la solution

Algorithme 1 : Raffinement itératif

Données : ϵ tolérance utilisateur, $\boldsymbol{A}$, $\boldsymbol{b}$
Entrées : $\boldsymbol{x}^{(0)} = \text{gauss}(\boldsymbol{A}, \boldsymbol{b})$, $i = 0$
répéter
  $\boldsymbol{r}^{(i)} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(i)}$
  $\boldsymbol{z} = \text{gauss}(\boldsymbol{A}, \boldsymbol{r}^{(i)})$
  $\boldsymbol{x}^{(i+1)} = \boldsymbol{x}^{(i)} + \boldsymbol{z}$
jusqu'à $\|\boldsymbol{z}\| \leq \epsilon \left\|\boldsymbol{x}^{(i+1)}\right\|$

- Exemple :

$$\boldsymbol{A} = \begin{bmatrix} k & 1 & 0 \\ 10 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \boldsymbol{b} = \begin{bmatrix} k+1 \\ 11 \\ 1 \end{bmatrix}$$

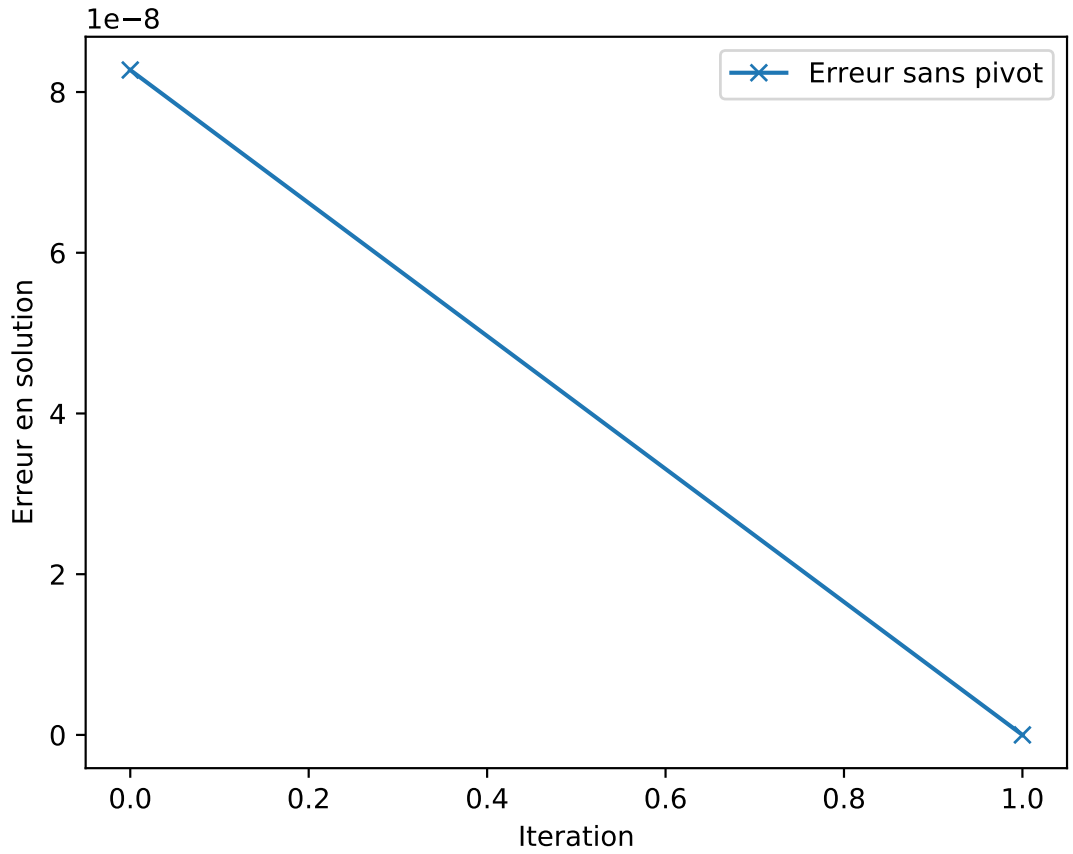
- Solution exacte :

$$\boldsymbol{x} = \begin{bmatrix} 1. \\ 1. \\ 1. \end{bmatrix}$$

Calculs à virgule flottante : le raffinement itératif

- Exemple :

$$\boldsymbol{A} = \begin{bmatrix} k & 1 & 0 \\ 10 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \boldsymbol{b} = \begin{bmatrix} k+1 \\ 11 \\ 1 \end{bmatrix} \quad \boldsymbol{x}_{ex} = \begin{bmatrix} 1. \\ 1. \\ 1. \end{bmatrix} \quad \boldsymbol{x}^{(0)} = \begin{bmatrix} 1.00000008 \\ 1. \\ 1. \end{bmatrix}$$



Autres factorisations classiques

- Variante de la factorisation $\boldsymbol{LU}$:

$$\boldsymbol{LDU}^\star = \boldsymbol{PA}$$

  Avec $\boldsymbol{D}$ diagonale et $\boldsymbol{U}^\star$ tri. sup. à diagonale unitaire.
- Ces factorisations ne profitent pas des propriétés de $\boldsymbol{A}$.
- Si $\boldsymbol{A}$ est symétrique $\rightarrow$ factorisation de Crout (pivotage symétrique)

$$\boldsymbol{LDL}^\top = \boldsymbol{PAP}^\top$$

  Coût du calcul $\simeq 1/3n^3$
- Si $\boldsymbol{A}$ est SDP, tous les termes $\boldsymbol{D}_{ii} > 0 \rightarrow$ factorisation de Cholesky.

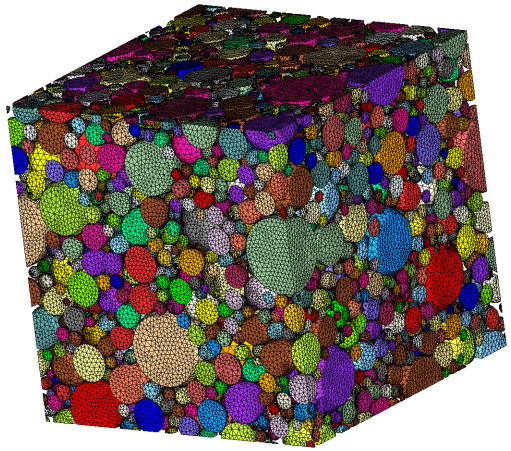
$$\boldsymbol{L}_c\boldsymbol{L}_c^\top = \boldsymbol{A}$$

  Coût du calcul $\simeq 1/3n^3$, pas besoin de pivoter *a priori*.

Est-ce que la complexité compte ?

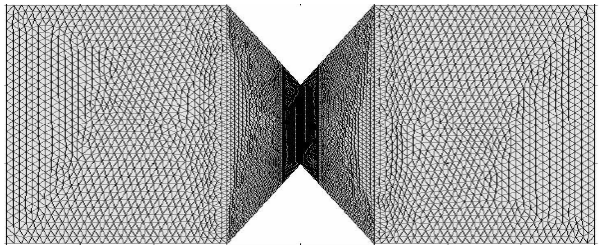
- En **dense**, coût d'une facto ***LU*** et temps de calcul

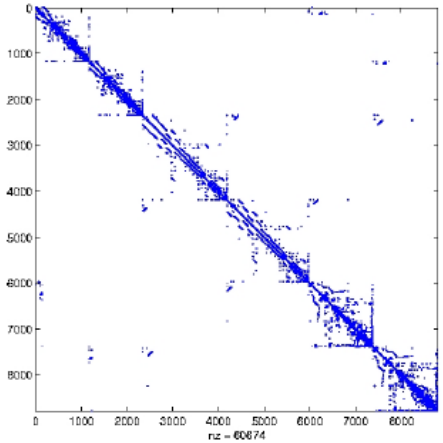
	Flop/s de l'ordinateur		
n	10^9	10^{12}	10^{15}
10^4	10 min	1 sec	1 μs
10^6	20 ans	7 mois	10 min
10^8	...	...	20 ans

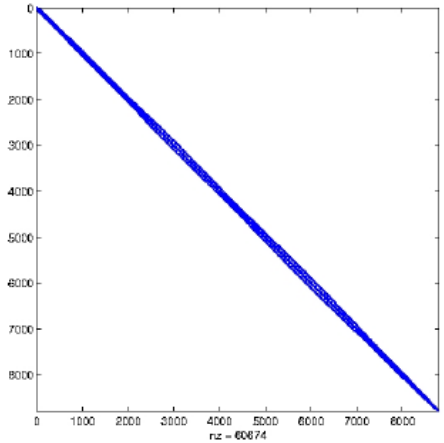


$\geq 10^8 dofs$

- Heureusement les systèmes sont souvent **creux**







Source : Analyse des solides déformables par la méthode des éléments finis, Bonnet, Frangi

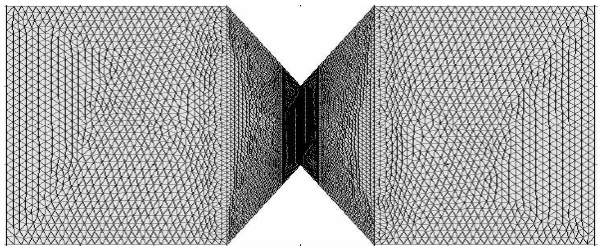
Remarques & références

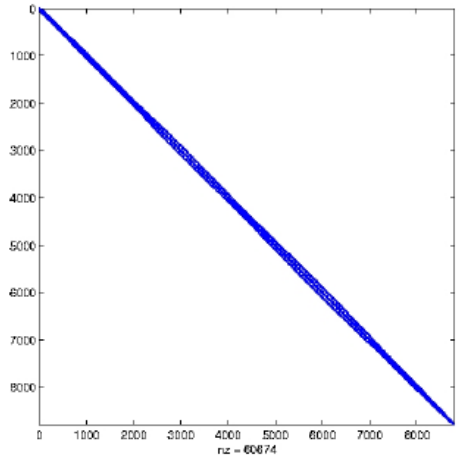
- Remarques
  - Sans info. supplémentaire, les méthodes `solve` utilisent une $\boldsymbol{LU}$
  - Si vous connaissez les propriétés de $\boldsymbol{A}$, utilisez les bonnes méthodes !
  - Ne codez pas vos propres méthodes (sauf si ça vous amuse)
- Quelques références
  - Langages interprétés
    Python Scipy : `scipy.linalg`
    Octave / Matlab : `lu`
  - Librairies compilées
    Eigen `http://eigen.tuxfamily.org/`
    GNU Scientific library `http://www.gnu.org/software/gsl/`
    LAPACK `http://www.netlib.org/lapack/`

Voir aussi `http://en.wikipedia.org/wiki/Comparison_of_linear_algebra_libraries`

Systèmes creux

- Les systèmes provenant d'EDPs sont souvent creux
- On parle de systèmes creux quand le nombre de valeur non nulle de $\boldsymbol{A}$ est en $O(n)$
- Exploiter le caractère creux du système permet de diminuer considérablement le coût de calcul d'une factorisation
- Certaines matrices creuses ont des structures particulières comme les matrices bandes ou blocs

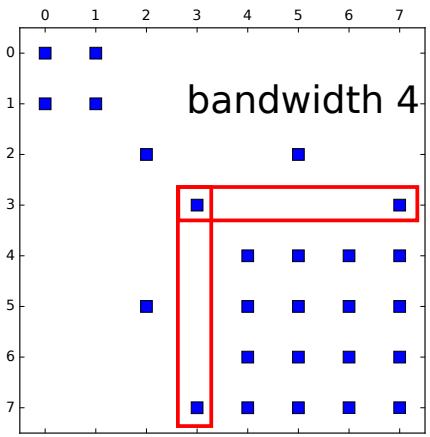




Source : Analyse des solides déformables par la méthode des éléments finis, Bonnet, Frangi

Matrices bandes

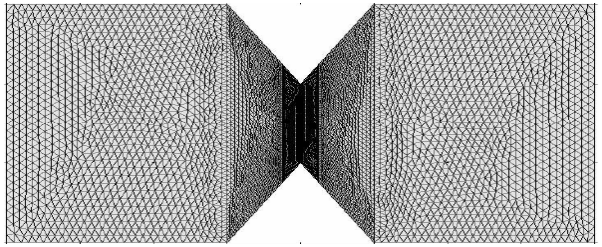
- $\boldsymbol{A}$ a une largeur de bande inférieure p si $a_{ij} = 0 \quad$ quand $\quad i > j + p$
- $\boldsymbol{A}$ a une largeur de bande supérieure q si $a_{ij} = 0 \quad$ quand $\quad j > i + q$
- Propriété :
  Soit $\boldsymbol{A}$ inversible., la factorisation LU existe et $\boldsymbol{L}$ a une largeur de bande inférieur p, $\boldsymbol{U}$ a une largeur de bande supérieure q

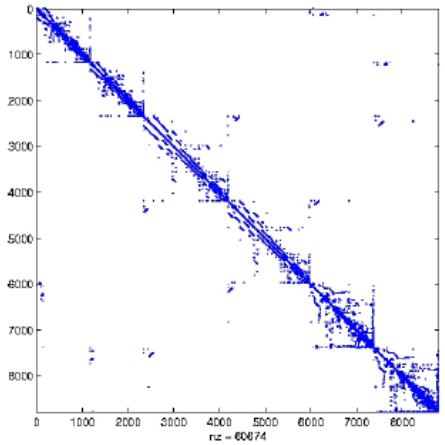


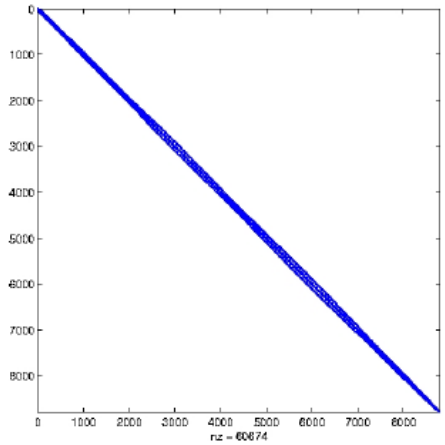
- On peut ne stocker que la bande
- Calcul uniquement des valeurs dans la bande
- Complexité de la factorisation en $O(2npq)$
- Complexité des substitutions en $O(np)$ et $O(nq)$

Stockage creux

- Pour une matrice bande, on stocke la bande dans une matrice dense







Source : Analyse des solides déformables par la méthode des éléments finis, Bonnet, Frangi

- Dans le cas général, $\neq$ stockages sont possibles (COO, CSR, CSC, ...)
- Exemple du stockage COO `(i,j,aij)`

```
i = [ 1 1 2 3 4]
j = [ 1 2 2 3 2]
aij = [ 11 45 6 22 3]
n = 4
m = 4
nnz = 5
```

$$\boldsymbol{A} = \begin{bmatrix} 11 & 45 & 0 & 0 \\ 0 & 6 & 0 & 0 \\ 0 & 0 & 22 & 0 \\ 0 & 3 & 0 & 0 \end{bmatrix}$$

- `nnz` nombre de valeurs non nulles

Factorisation et stockage creux

- On considère la factorisation LU de la matrice $\boldsymbol{A}$ creuse
- Les matrices $\boldsymbol{L}$ et $\boldsymbol{U}$ sont en général creuses également
- Elles sont par contre (beaucoup) plus remplies

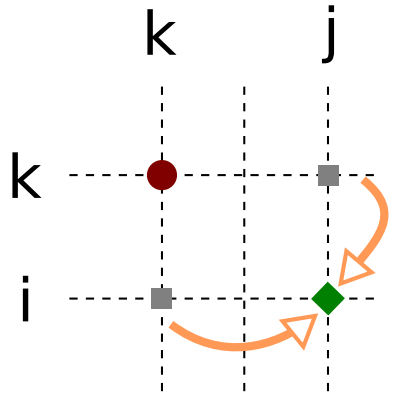
$$nnz(\boldsymbol{A}) \ll nnz(\boldsymbol{L}) \quad \text{et} \quad nnz(\boldsymbol{A}) \ll nnz(\boldsymbol{U})$$

- Le remplissage ou *fill-in* est la différence de *nnz* entre $\boldsymbol{A}$ et $\boldsymbol{L}$ ou $\boldsymbol{U}$
- Le *fill-in* augmente la mémoire nécessaire et le coût de la factorisation
- En plus de toutes les problématiques vues précédemment (changement de pivots, scaling, ...), il faut veiller à minimiser ce remplissage

Source du *fill-in*

- Retour sur l'élimination de Gauss

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

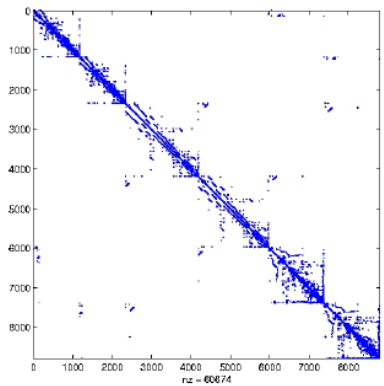


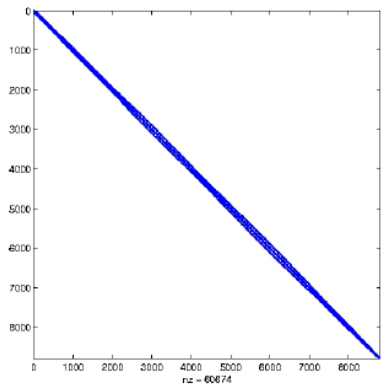
- À l'étape $k+1$, $a_{ij}^{(k+1)} \neq 0$ si $\begin{cases} a_{ij}^{(k)} \neq 0 \\ a_{ik}^{(k)} \neq 0 \text{ et } a_{kj}^{(k)} \neq 0 \text{ (structural fill-in)} \end{cases}$
- Exemple de la matrice flèche

$$\begin{pmatrix} x & x & x & x \\ x & x & & \\ x & & x & \\ x & & & x \end{pmatrix} \Rightarrow \begin{pmatrix} \star & \star & \star & \star \\ \star & \star & \star & \star \\ \star & \star & \star & \star \\ \star & \star & \star & \star \end{pmatrix} \quad \text{mais} \quad \begin{pmatrix} x & & & x \\ & x & & x \\ & & x & x \\ x & x & x & x \end{pmatrix} \Rightarrow \begin{pmatrix} \star & & & \star \\ & \star & & \star \\ & & \star & \star \\ \star & \star & \star & \star \end{pmatrix}$$

Analyse symbolique

- Permutater des lignes et des colonnes de $\boldsymbol{A}$ permet de réduire significativement le remplissage
- On factorise $\boldsymbol{A}^\star = \boldsymbol{P}_1 \boldsymbol{A} \boldsymbol{Q}_1$ où $\boldsymbol{P}_1$ et $\boldsymbol{Q}_1$ sont des matrices de permutations.
- La phase d'**analyse symbolique** vise à trouver $\boldsymbol{P}_1$ et $\boldsymbol{Q}_1$ pour minimiser le remplissage
- Cette phase s'appuie uniquement sur le profile de $\boldsymbol{A}$
- Elle fait appel à de notions complexes de théorie des graphes
- Exemple : algorithme de type Cuthill–McKee pour minimiser la largeur de bande de $\boldsymbol{A}^\star$





Complexité en $O(m \log(m) nnz)$ avec m le plus haut degré du graphe.

Factorisation et stockage creux

- Les permutations ont un rôle double
  - Garantir la stabilité et limiter la propagation des erreurs d'arrondis
  - Limiter le remplissage
- La factorisation de $\boldsymbol{A}^\star = \boldsymbol{P}_1 \boldsymbol{A} \boldsymbol{Q}_1$ peut nécessiter des changements de pivots
- Il faut veiller à ne augmenter trop le remplissage

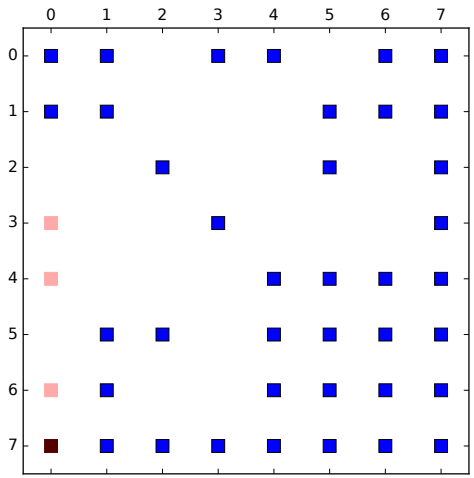
$$\boldsymbol{L}\boldsymbol{U} = \boldsymbol{P}_2 \boldsymbol{A}^\star \boldsymbol{Q}_2 = \boldsymbol{P}_2 \boldsymbol{P}_1 \boldsymbol{A} \boldsymbol{Q}_1 \boldsymbol{Q}_2$$

- Pivots candidats *(threshold pivot)* :
  au lieu de choisir

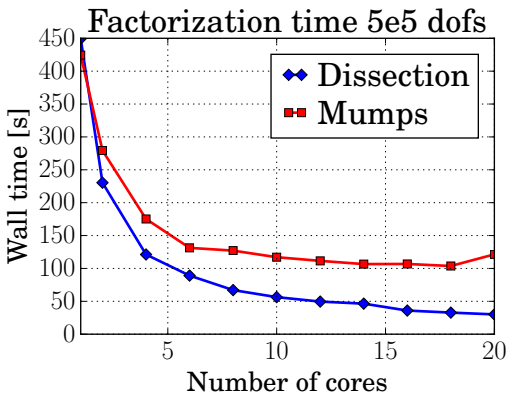
$$p = \underset{i \geq k}{\operatorname{argmax}}(|a_{ik}^{(k)}|)$$

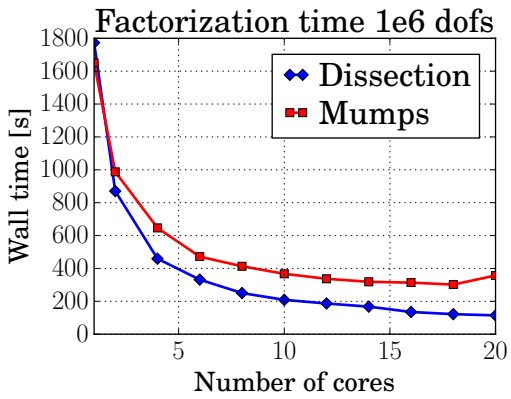
on choisit le pivot qui minimise le *fill-in*
parmi les pivots candidats vérifiant :

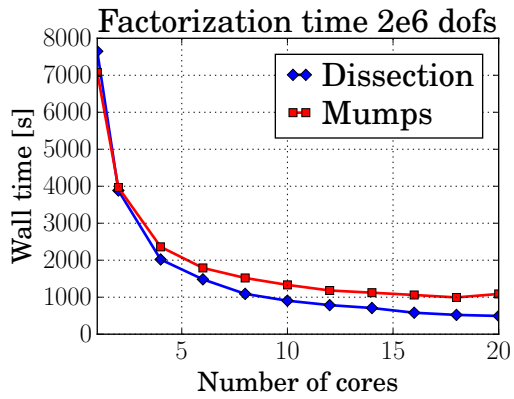
$$|a_{ik}^{(k)}| \geq \tau |\max a_{pk}^{(k)}|$$



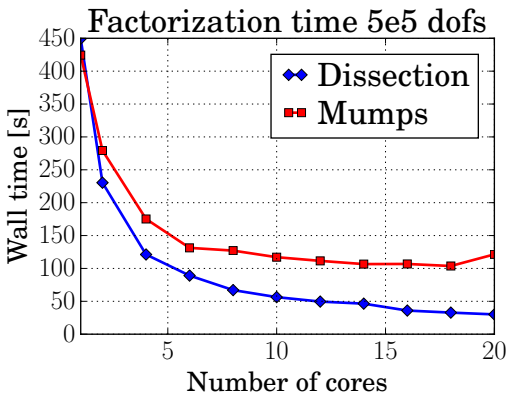
Exemple : cube élastique linéaire (c3d20)

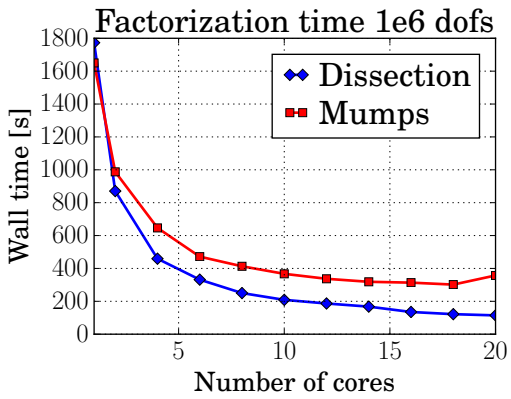


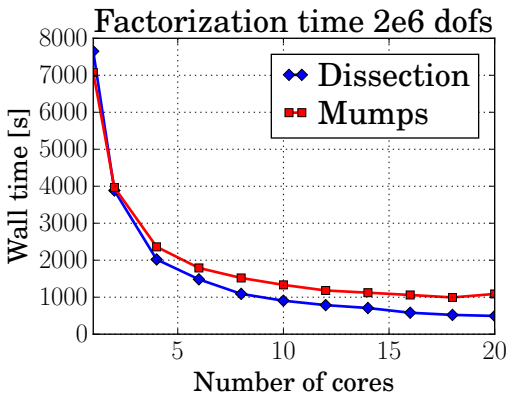


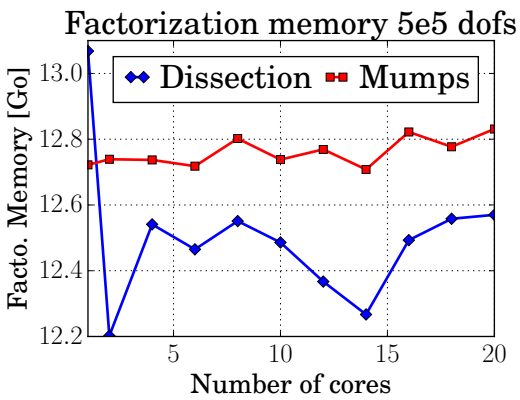


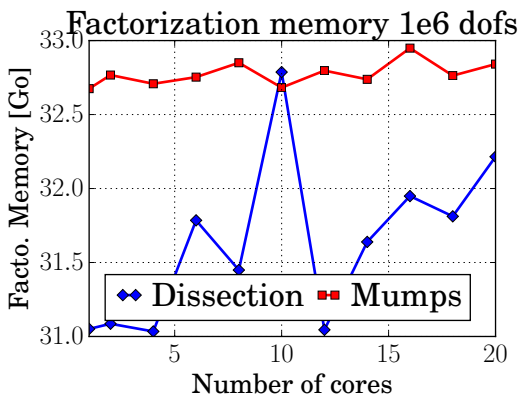
Exemple : cube élastique linéaire (c3d20)

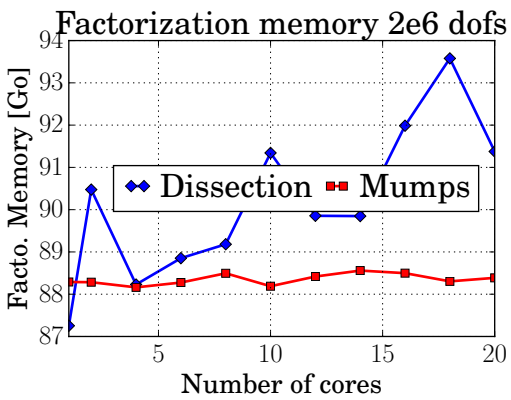












Synthèse

Les solveurs directs sont

- des variantes de l'élimination de Gauss
- ils fonctionnent en 3 phases

Fact. symbolique $\rightarrow$ Fact. numérique $\rightarrow$ descente-remontée

- robustes : ils fournissent la solution exacte (arithmétique exacte) en un nombre fini d'opération (qui peut être grand)
- coûteux en mémoire (les ressources croissent fortement avec la taille du problème)
- le conditionnement de $\boldsymbol{A}$ influe uniquement sur la qualité de la solution (remèdes : pivot, scaling, raffinement itératif)

Remarques & références

- Remarques
  - Si vous connaissez les propriétés de $\boldsymbol{A}$, utilisez les bonnes méthodes !
  - Ne codez pas vos propres méthodes (sauf si c'est votre métier)
- Quelques références
  - Langages interprétés
    - Python Scipy :
      `scipy.sparse` & `scipy.sparse.linalg`
    - Matlab :
      `http://fr.mathworks.com/help/matlab/sparse-matrices.html`
  - Librairies compilées
    - MUltifrontal Massively Parallel sparse direct Solver (MUMPS)
      `http://mumps.enseeiht.fr/`
    - Supernodal LU (SuperLU)
      `http://crd-legacy.lbl.gov/~xiaoye/SuperLU/`
    - Parallel Sparse matriX package (PaStiX)
      `http://pastix.gforge.inria.fr/`

Plan

Pourquoi utiliser des méthodes itératives ?

- Les solveurs directs sont robustes mais très gourmand en mémoire
- La parallélisation est possible mais pas évidente (échange de complément de Schur)
- Pour les très gros problèmes il faut penser aux méthodes itératives ou hybrides
- **En dense**, un produit MV en $O(n^2) \rightarrow$ uniquement utile pour le creux
- Solveurs itératifs presque *embarrassingly parallel*

$$\boldsymbol{A}\boldsymbol{x} \quad \Rightarrow \quad \begin{bmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} \\ \boldsymbol{A}_{21} & \boldsymbol{A}_{22} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{bmatrix}$$

Principe

- On souhaite résoudre $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ avec $\boldsymbol{A}$ très grande et creuse
- Construire une suite de vecteurs tels que $\lim_{k \to +\infty} \boldsymbol{x}_k = \boldsymbol{A}^{-1}\boldsymbol{b} = \boldsymbol{x}^\star$
- Soit $\boldsymbol{x}_k$ l'approximation au pas k

$$\boldsymbol{x}^\star = \boldsymbol{x}_k + \boldsymbol{e}_k \quad \text{(erreur)}$$

$$\boldsymbol{A}\boldsymbol{e}_k = \boldsymbol{A}\boldsymbol{x}^\star - \boldsymbol{A}\boldsymbol{x}_k := \boldsymbol{r}_k \quad \text{(résidu)} \quad \text{d'où} \quad \boldsymbol{x}^\star = \boldsymbol{x}_k + \boldsymbol{A}^{-1}\boldsymbol{r}_k$$

- Idée : remplacer $\boldsymbol{A}$ par une matrice proche mais facilement inversible
- Une méthode itérative est convergente ssi la suite $(\boldsymbol{x}_n)_n \to \boldsymbol{x}^\star \ \forall$ init. $\boldsymbol{x}_0$
- Choix du critère d'arrêt $\left| \begin{array}{l} \|\boldsymbol{r}_k\| \leq \epsilon \|\boldsymbol{r}_0\| \text{ (résidu)} \\ \|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\| \leq \epsilon \|\boldsymbol{x}_k\| \text{ (stagnation)} \end{array} \right.$

Principe

- Soit $\boldsymbol{M}$ une matrice inversible qui
  - est une bonne approximation de $\boldsymbol{A}$
  - soit facile à calculer
  - permette de résoudre facilement le système $\boldsymbol{Mz} = \boldsymbol{r}$
- $\boldsymbol{M}$ est appelé **préconditionneur**
- Au lieu de résoudre $\boldsymbol{x}^\star = \boldsymbol{x}_k + \boldsymbol{A}^{-1}\boldsymbol{r}_k$, on itère :

$$\begin{aligned} \boldsymbol{r}_k &= \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_k \\ \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \boldsymbol{M}^{-1}\boldsymbol{r}_k \end{aligned}$$

- Trois étapes :
  calcul de résidu $\rightarrow$ résolution du pb préconditionné $\rightarrow$ maj solution

Préconditionnements de type décomposition

- Décomposer la matrice $\boldsymbol{A} = \boldsymbol{M} - \boldsymbol{N}$ où $\boldsymbol{M}$ est « facilement inversible »
- Suivant le choix de $\boldsymbol{M}$ on obtient $\neq$ méthodes avec des propriétés de convergence différentes
- Exemples : si $\boldsymbol{A} = \boldsymbol{D} - \boldsymbol{E} - \boldsymbol{F}$ avec | $\boldsymbol{D}$ diagonal
$\boldsymbol{E}$ tri. inf. stricte
$\boldsymbol{E}$ tri. sup. stricte
- $\boldsymbol{M} = \boldsymbol{D} \rightarrow$ méthode de Jacobi (convergence garantie si A est à diag. dom. stricte)
- $\boldsymbol{M} = \boldsymbol{D} - \boldsymbol{E} \rightarrow$ méthode de Gauss-Seidel (convergence garantie si A est SPD)

Méthodes itératives de type Krylov

- Avec les méthodes itératives stationnaires précédentes

$$\boldsymbol{x}_k = \boldsymbol{x}_{k-1} + \boldsymbol{M}^{-1}\boldsymbol{r}_{k-1} = \boldsymbol{x}_{k-1} + \boldsymbol{z}_{k-1}$$

- $\boldsymbol{z}_k = Q_k(\boldsymbol{M}^{-1}\boldsymbol{A})\boldsymbol{z}_0$ où Q_k est le polynôme $Q_k(X) = (1-X)^k$

$$Q_k \in \mathcal{K}_k(\boldsymbol{M}^{-1}\boldsymbol{A}, \boldsymbol{z}_0) = \text{Vect}\left(\boldsymbol{z}_0, (\boldsymbol{M}^{-1}\boldsymbol{A})\boldsymbol{z}_0, \ldots, (\boldsymbol{M}^{-1}\boldsymbol{A})^{k-1}\boldsymbol{z}_0\right)$$

- $\mathcal{K}_k(\boldsymbol{M}^{-1}\boldsymbol{A}, \boldsymbol{z}_0)$ est l'espace de Krylov généré par le couple $(\boldsymbol{M}^{-1}\boldsymbol{A}, \boldsymbol{z}_0)$
- Les méthodes de type Krylov ajoutent une contrainte sur $\boldsymbol{z}_k$ tel que

$$\begin{cases} \boldsymbol{x}_k \in \ \boldsymbol{x}_0 + \mathcal{K}_k(\boldsymbol{M}^{-1}\boldsymbol{A}, \boldsymbol{z}_0) \\ \boldsymbol{z}_k \perp_? \mathcal{K}_k(\boldsymbol{M}^{-1}\boldsymbol{A}, \boldsymbol{z}_0) \end{cases}$$

Méthodes itératives de type Krylov

- Les méthodes de type Krylov ajoutent une contrainte sur $\boldsymbol{r}_m$ tel que

$$\begin{cases} \boldsymbol{x}_m \in \ \boldsymbol{x}_0 + \mathcal{K}_m(\boldsymbol{A}, \boldsymbol{r}_0) \\ \boldsymbol{r}_m \perp_? \ \mathcal{K}_m(\boldsymbol{A}, \boldsymbol{r}_0) \end{cases}$$

- Suivant les propriétés de $\boldsymbol{A}$, le choix du type l'orthogonalité permet de définir plusieurs approches (CG, GMRes, OrthoDir, BiCG, CGS, ...)
- Si $\boldsymbol{A}$ est SDP le Gradient Conjugué (CG) est la meilleure solution

$$\begin{cases} \boldsymbol{x}_m \in \ \boldsymbol{x}_0 + \mathcal{K}_m(\boldsymbol{K}, \boldsymbol{r}_0) \\ \boldsymbol{r}_m \perp \ \mathcal{K}_m(\boldsymbol{K}, \boldsymbol{r}_0) \end{cases}$$

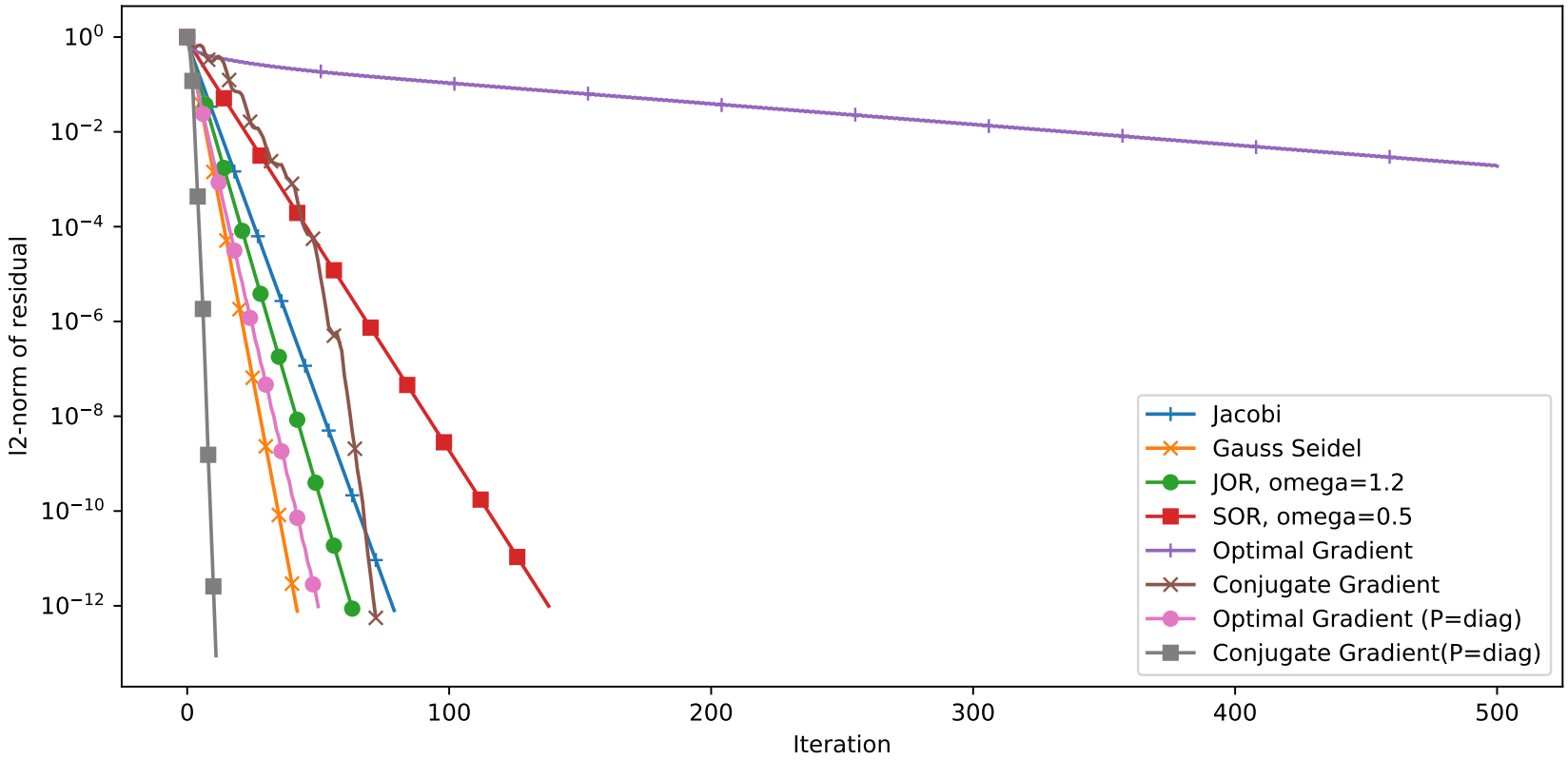
- Avec le CG, $\boldsymbol{x}_k$ minimise la $\boldsymbol{A}$-norme de l'erreur sur l'espace de Krylov.
- Dans le contexte parallèle, l'expression de la contrainte nécessite des communications globales

Quelques exemples

- Matrice symétrique

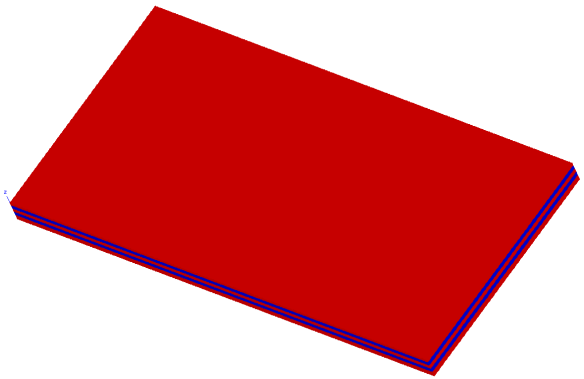
```
def example_generator(n=100, k=1e3, off=1.):
  """ Generate an example and plot the results """
  b = np.random.rand((n))
  L = np.random.rand((n*n)); L.shape = n, n
  A = - (L + L.transpose())
  A += k * np.diag(np.random.rand(n) + off)

  P = A.diagonal()
  def prec(r, z):
    z[:] = r/P; return z
```

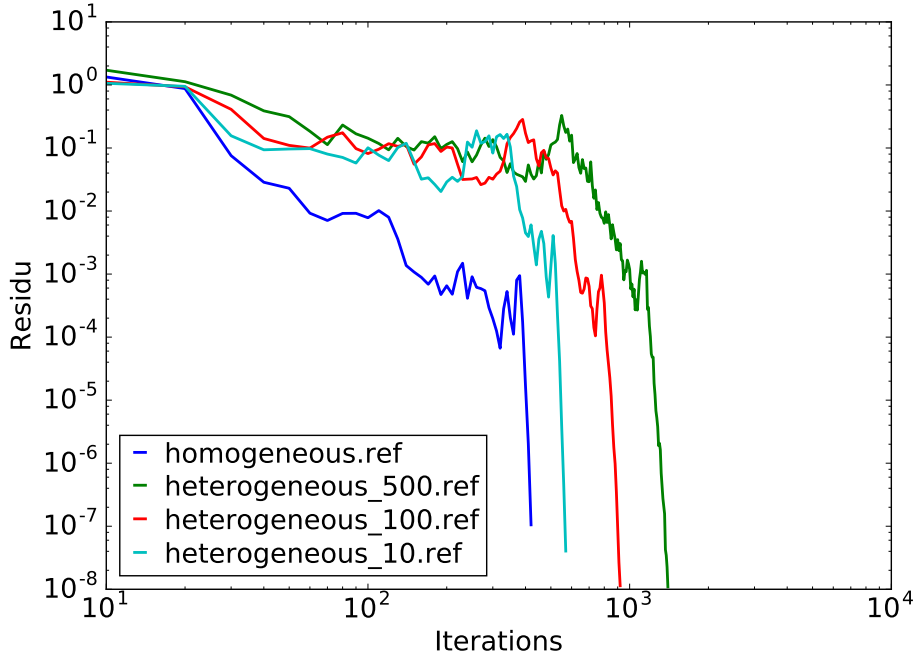
Quelques exemples



Exemple provenant d'une EDP et conditionnement

- Plaque lamifiée élastique linéaire $n \simeq 21\ 000$,
- Solveur : Gradient conjugué
- Préconditionneur : factorisation de cholesky incomplète





Synthèse

- Les méthodes itératives nécessitent très peu de mémoire
- Elles peuvent donc résoudre de très gros problèmes
- Choisir un bon préconditionneur n'est pas toujours évident
- La vitesse de convergence dépend du conditionnement de l'opérateur à résoudre
- Les méthodes de Krylov convergent généralement plus rapidement
- Elles requièrent cependant plus de communications

Plan

Méthodes hybrides

- Les méthodes modernes combinent méthodes directes et itératives
- Objectif : combiner les avantages de chaque approche
- Exemples :
  - Méthodes itératives par blocs
    solveur itératif stationnaire
    méthodes directes sur les sous-blocs
  - Méthodes itératives avec préconditionneur par facto. incomplète
  - Méthode de type décomposition de domaine
    solveurs directs pour les problèmes locaux
    solveurs de Krylov pour équilibrer l'interface
  - Méthodes multi-grilles
    solveurs itératifs stationnaires
    préconditionneur avec une méthode DD

Méthodes hybrides

- Les méthodes modernes combinent méthodes directes et itératives
- Objectif : combiner les avantages de chaque approche
- Exemples :
  - Méthodes itératives par blocs
    solveur itératif stationnaire
    méthodes directes sur les sous-blocs
  - Méthodes itératives avec préconditionneur par facto. incomplète
  - Méthode de type décomposition de domaine
    solveurs directs pour les problèmes locaux
    solveurs de Krylov pour équilibrer l'interface
  - Méthodes multi-grilles
    solveurs itératifs stationnaires
    préconditionneur avec une méthode DD

Méthodes de décomposition de domaine

- Système global

$$\boldsymbol{K}\boldsymbol{u} = \boldsymbol{f} \quad \text{avec} \quad \boldsymbol{K} \text{ SDP}$$

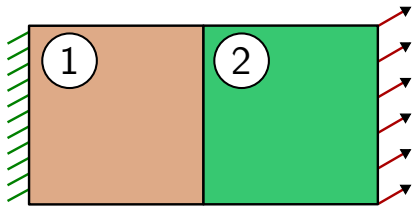
- Décompo. sans recouvrement

$$\boldsymbol{K}^{(s)}\boldsymbol{u}^{(s)} = \boldsymbol{f}^{(s)} + \boldsymbol{\lambda}^{(s)}, \; s = 1, 2$$

$$\boldsymbol{\lambda}_b^{(1)} + \boldsymbol{\lambda}_b^{(2)} = 0$$

$$\boldsymbol{u}_b^{(1)} - \boldsymbol{u}_b^{(2)} = 0$$

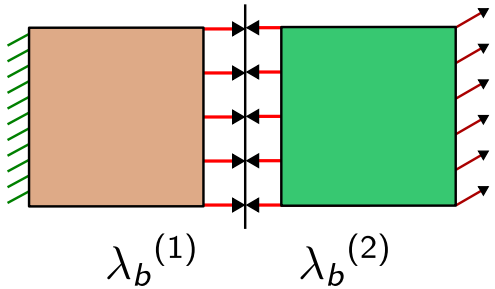
- Problème global $\Leftrightarrow$
  - Équilibre locaux
  - Équilibre interface
  - Continuité interface



Problème init.

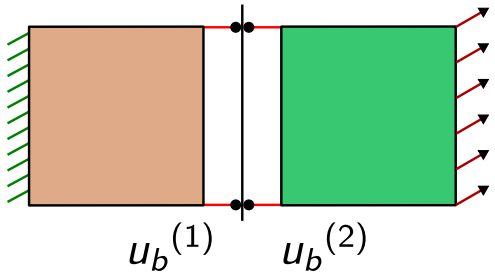
⇩

Décomposition



Équilibre

$\lambda_b{}^{(1)} = -\lambda_b{}^{(2)}$



Continuité

$u_b{}^{(1)} = u_b{}^{(2)}$

Décomposition en deux sous domaines

On suppose les $\boldsymbol{K}_{ii}^{s}$ inversibles.

$$\begin{pmatrix} \boldsymbol{K}_{ii}^{(1)} & 0 & \boldsymbol{K}_{ib}^{(1)} \\ 0 & \boldsymbol{K}_{ii}^{(2)} & \boldsymbol{K}_{ib}^{(2)} \\ \boldsymbol{K}_{bi}^{(1)} & \boldsymbol{K}_{bi}^{(2)} & \boldsymbol{K}_{bb}^{(1)} + \boldsymbol{K}_{bb}^{(2)} \end{pmatrix}$$

$$= \begin{pmatrix} \boldsymbol{I} & 0 & 0 \\ 0 & \boldsymbol{I} & 0 \\ \boldsymbol{K}_{bi}^{(1)} \boldsymbol{K}_{ii}^{(1)^{-1}} & \boldsymbol{K}_{bi}^{(2)} \boldsymbol{K}_{ii}^{(2)^{-1}} & \boldsymbol{I} \end{pmatrix} \begin{pmatrix} \boldsymbol{K}_{ii}^{(1)} & 0 & 0 \\ 0 & \boldsymbol{K}_{ii}^{(2)} & 0 \\ 0 & 0 & \boldsymbol{S} \end{pmatrix} \begin{pmatrix} \boldsymbol{I} & 0 & \boldsymbol{K}_{ii}^{(1)^{-1}} \boldsymbol{K}_{ib}^{(1)} \\ 0 & \boldsymbol{I} & \boldsymbol{K}_{ii}^{(2)^{-1}} \boldsymbol{K}_{ib}^{(2)} \\ 0 & 0 & \boldsymbol{I} \end{pmatrix}$$

Complément de Schur

$$\boldsymbol{S} = \boldsymbol{K}_{bb} - \boldsymbol{K}_{bi}^{(1)} \boldsymbol{K}_{ii}^{(1)^{-1}} \boldsymbol{K}_{ib}^{(1)} - \boldsymbol{K}_{bi}^{(2)} \boldsymbol{K}_{ii}^{(2)^{-1}} \boldsymbol{K}_{ib}^{(2)}$$

$$= \left(\boldsymbol{K}_{bb}^{(1)} - \boldsymbol{K}_{bi}^{(1)} \boldsymbol{K}_{ii}^{(1)^{-1}} \boldsymbol{K}_{ib}^{(1)} \right) + \left(\boldsymbol{K}_{bb}^{(2)} - \boldsymbol{K}_{bi}^{(2)} \boldsymbol{K}_{ii}^{(2)^{-1}} \boldsymbol{K}_{ib}^{(2)} \right)$$

Décomposition en deux sous domaines

$$\boldsymbol{K}\boldsymbol{u} = \boldsymbol{f}$$

$$\begin{pmatrix} \boldsymbol{K}_{ii}^1 & 0 & 0 \\ 0 & \boldsymbol{K}_{ii}^2 & 0 \\ 0 & 0 & \boldsymbol{S} \end{pmatrix} \begin{pmatrix} \boldsymbol{I} & 0 & \boldsymbol{K}_{ii}^{1\,-1}\boldsymbol{K}_{ib}^1 \\ 0 & \boldsymbol{I} & \boldsymbol{K}_{ii}^{2\,-1}\boldsymbol{K}_{ib}^2 \\ 0 & 0 & \boldsymbol{I} \end{pmatrix} \begin{pmatrix} \boldsymbol{u}_i^1 \\ \boldsymbol{u}_i^2 \\ \boldsymbol{u}_b \end{pmatrix} = \begin{pmatrix} \boldsymbol{I} & 0 & 0 \\ 0 & \boldsymbol{I} & 0 \\ -\boldsymbol{K}_{bi}^1\boldsymbol{K}_{ii}^{1\,-1} & -\boldsymbol{K}_{bi}^2\boldsymbol{K}_{ii}^{2\,-1} & \boldsymbol{I} \end{pmatrix} \begin{pmatrix} \boldsymbol{f}_i^1 \\ \boldsymbol{f}_i^2 \\ \boldsymbol{f}_b \end{pmatrix}$$

Algorithme 2 : Méthode de décomposition de domaine, principe de base

Factorisation parallèle de $\boldsymbol{K}_{ii}^{(s)}$

Calcul parallèle des RHS condensé $\boldsymbol{b}^s = \boldsymbol{f}_b^s - \boldsymbol{K}_{bi}^s\boldsymbol{K}_{ii}^{s\,-1}\boldsymbol{f}_i^s$ puis assemblage avec son voisin

Résolution du système d'interface $\boldsymbol{S}\boldsymbol{u}_b = \boldsymbol{b}$ via un solveur itératif

Descente remontée parallèle de $(\boldsymbol{K}_{ii}^{(s)})^{-1} \quad \Rightarrow \quad \boldsymbol{u}_i^s = \boldsymbol{K}_{ii}^{s\,-1}(\boldsymbol{f}_i^s - \boldsymbol{K}_{bi}^s\boldsymbol{u}_b)$

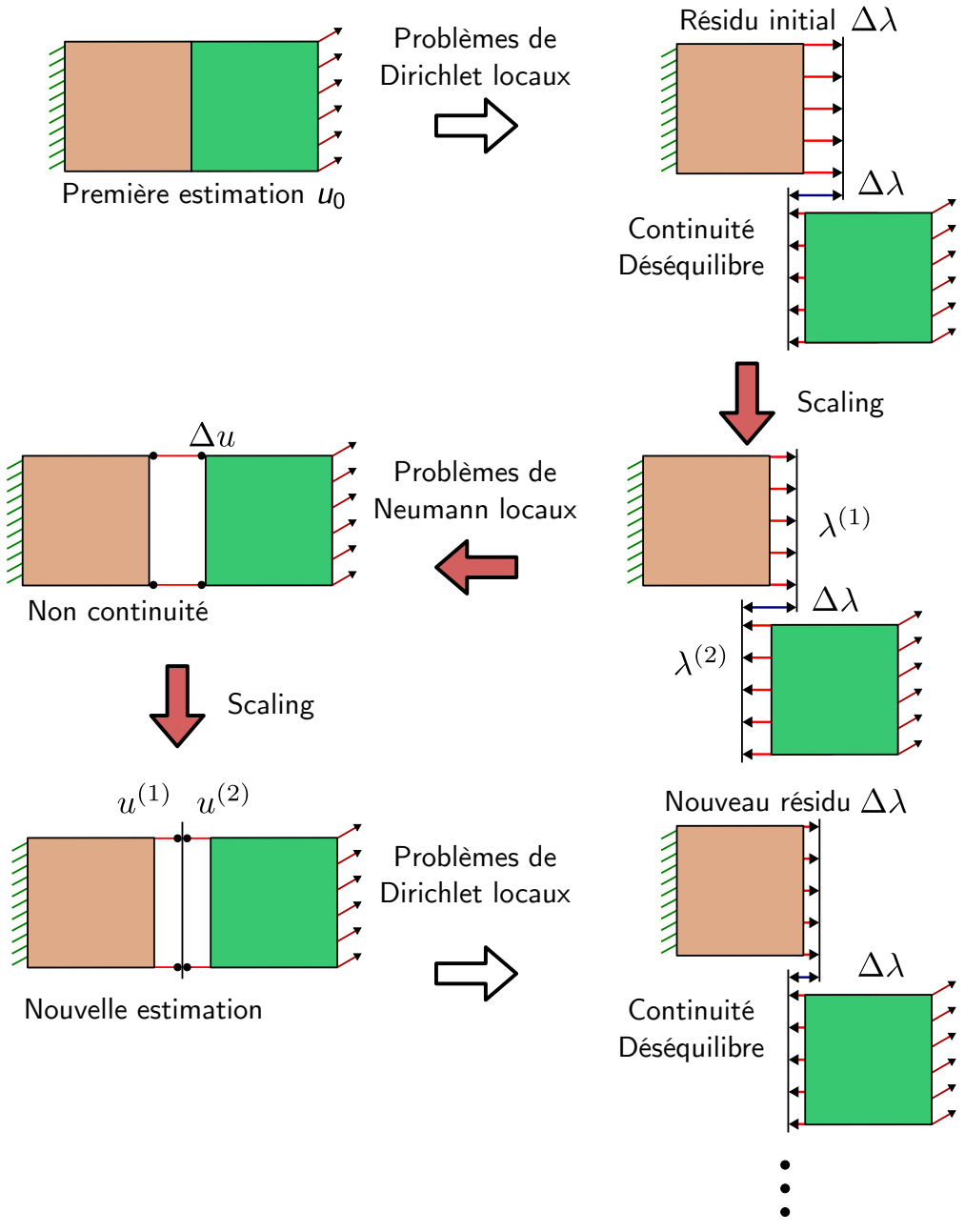
Méthode Balancing Domain Decomposition (BDD)

- L'opérateur est une somme de compléments de Schur locaux

$$\boldsymbol{S}\boldsymbol{u}_b = (\boldsymbol{S}^{(1)} + \boldsymbol{S}^{(2)})\boldsymbol{u}_b$$

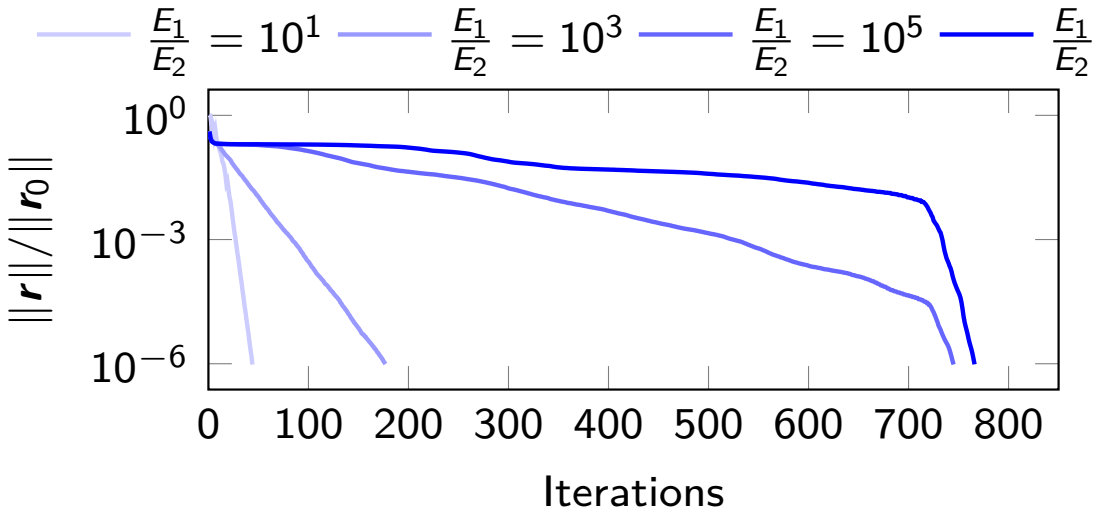
- Pour le préconditionneur, on approche l'inverse de la somme par la somme des inverses

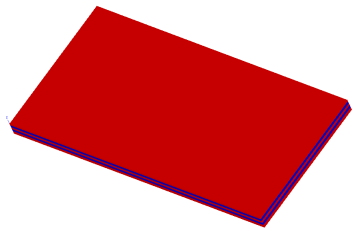
$$\boldsymbol{M} = (\boldsymbol{S}^{(1)+} + \boldsymbol{S}^{(2)+})$$



Méthode BDD

- Les méthodes DD classiques restent sensibles au conditionnement du problème
- Des solutions existent (multipreconditionnement, augmentation)

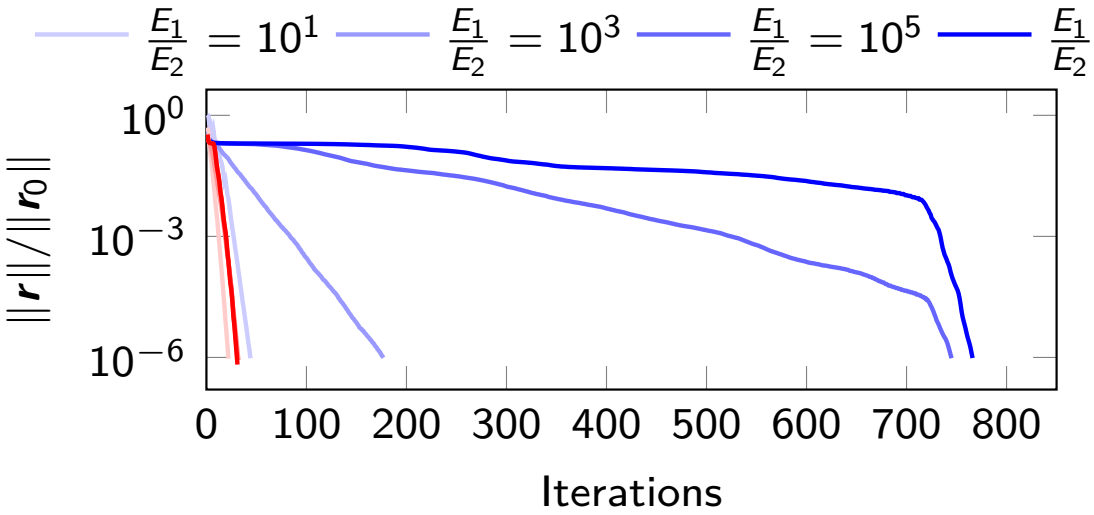


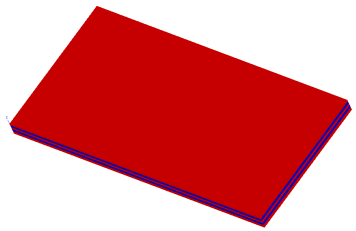


Plaque hétérogène

Méthode BDD

- Les méthodes DD classiques restent sensibles au conditionnement du problème
- Des solutions existent (multipreconditionnement, augmentation)



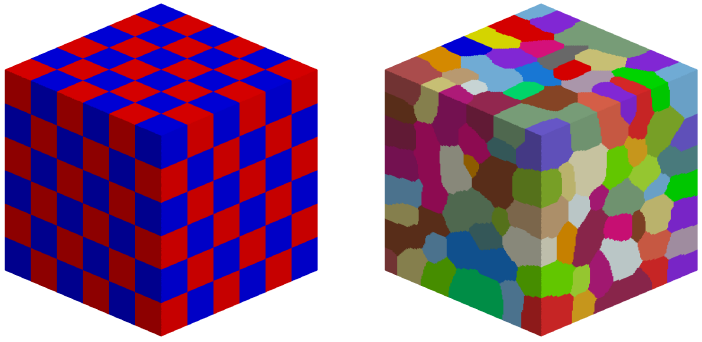


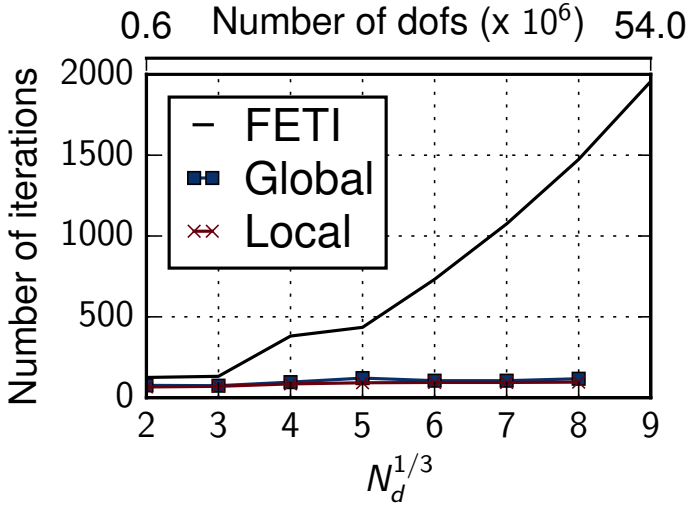
Plaque hétérogène

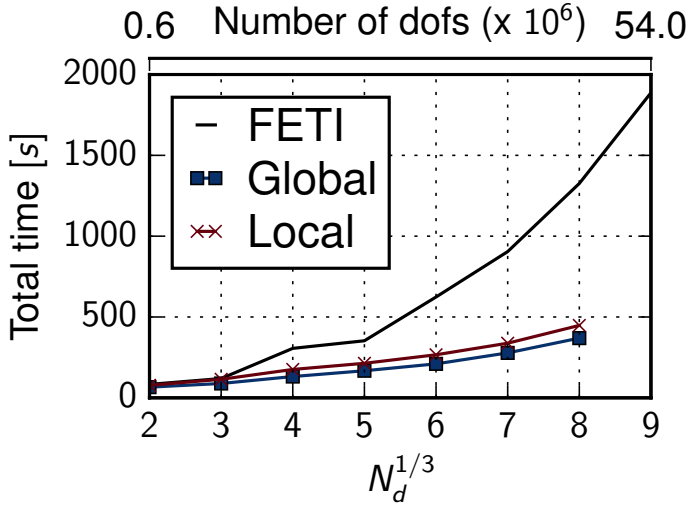
Étude d'extensibilité faible hétérogène (élasticité linéaire)

Paramètres $\tau = 0.01$, $E_1/E_2 = 10^6$, $H/h = 29$, (Occigen cluster, $4Gb/core$)

Adaptive Multipréconditioned FETI







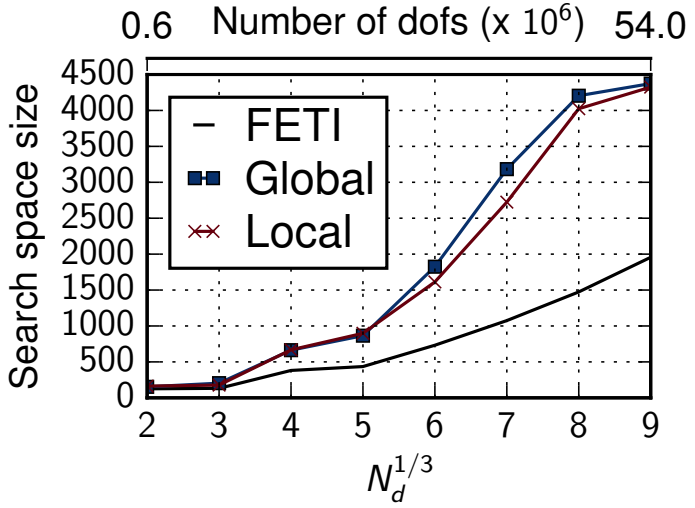
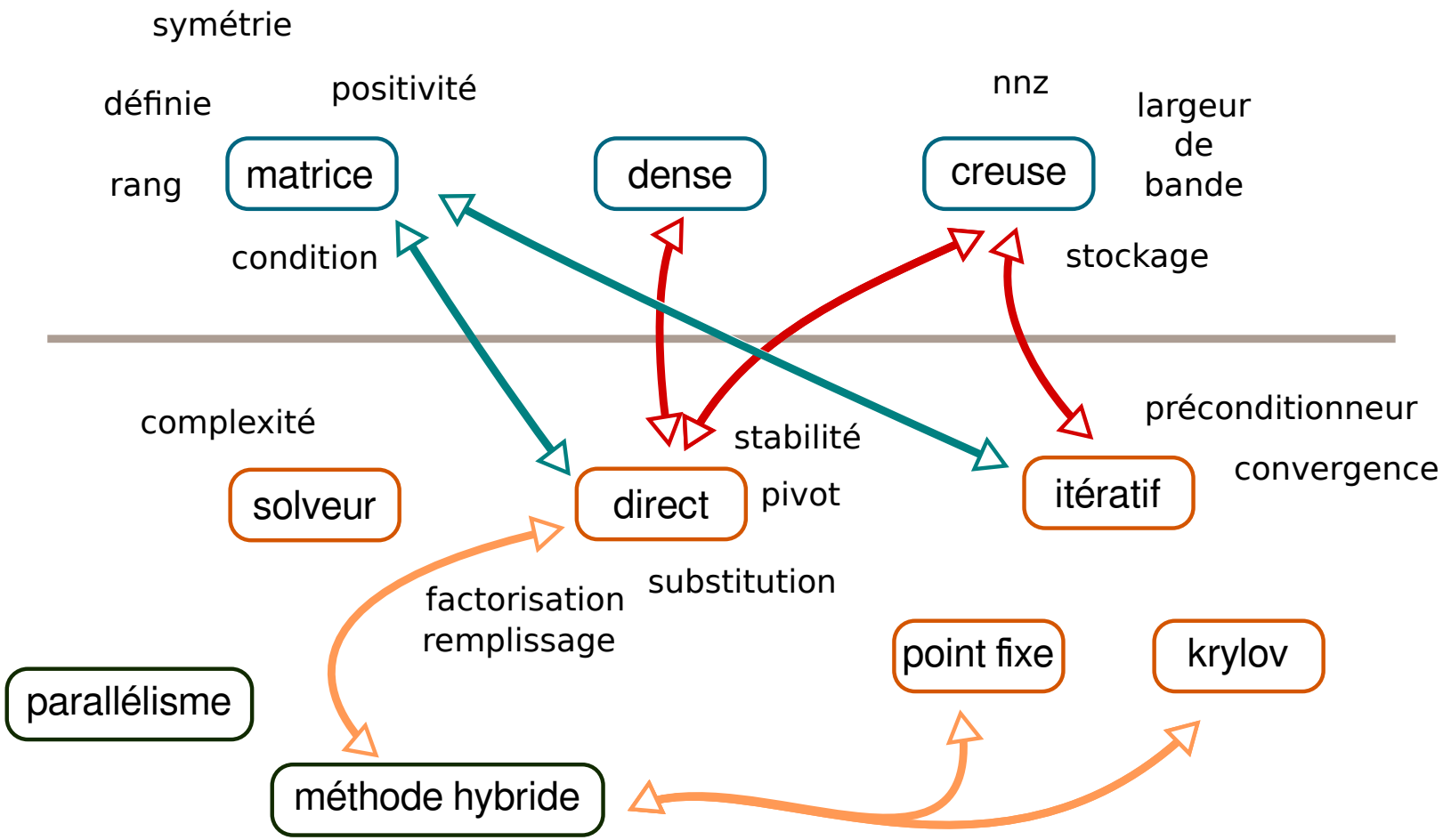


Schéma synoptique



Références

- Méthodes numériques en général
  - Quarteroni, A. M., Sacco, R., & Saleri, F. (2008). Méthodes numériques : algorithmes, analyse et applications. Springer Science & Business Media.
- Algèbre linéaire dense
  - Golub & van Loan : Matrix Computations, 3rd ed., Johns Hopkins, 1996.
- Solveurs directs creux
  - I. Duff, A. Erisman, J. Reid : Direct Methods for Sparse Matrices, Oxford University Press, 1986.
  - T. Davis : Direct Methods for Sparse Linear Systems, SIAM,2006.
- Solveurs de Krylov
  - Y. Saad : Iterative Methods for Sparse Linear Systems, 2nd ed., pp. 103–128, SIAM, 2003.
  - Van der Vorst, H. A. (2003). Iterative Krylov methods for large linear systems (Vol. 13). Cambridge University Press.