Résolution de grands systèmes linéaires creux

Méthodes directes et méthodes itératives.

Thierry Dumont

Institut Camille Jordan
Université Lyon 1 et CNRS.

16 Novembre 2010.

Le conditionnement d'un système linéaire

Le conditionnement d'un système linéaire

Normes de matrices et conditionnement :

$A \in \mathbb{R}^{n \times n}$.

$\|x\|$ une norme sur $\mathbb{R}^{n \times n}$.

Le conditionnement d'un système linéaire

Normes de matrices et conditionnement :

$A \in \mathbb{R}^{n \times n}$.

$\|x\|$ une norme sur $\mathbb{R}^{n \times n}$.

Exemples :

- $\|x\|_\infty = \max |x_i|$
- $\|x\|_1 = \sum_{i=1}^{n} |x_i|$,
- la norme euclidienne $\|x\|_2 = (\sum_{i=1}^{n} x_i^2)^{1/2}$;

Le conditionnement d'un système linéaire

Normes de matrices et conditionnement :

$A \in \mathbb{R}^{n \times n}$.

$\|x\|$ une norme sur $\mathbb{R}^{n \times n}$.

Exemples :

- $\|x\|_\infty = \max |x_i|$
- $\|x\|_1 = \sum_{i=1}^{n} |x_i|$,
- la norme euclidienne $\|x\|_2 = (\sum_{i=1}^{n} x_i^2)^{1/2}$;

$$\|A\| = \max \frac{\|A\ x\|}{\|x\|}$$

définit une norme sur l'ensemble des matrices $n \times n$. On dit qu'il s'agit d'une norme *subordonnée* à la norme définie sur $\mathbb{R}^n$.

Le conditionnement d'un système linéaire

Définition : $\kappa(A) = \|A^{-1}\| \cdot \|A\|$

Le conditionnement d'un système linéaire

Définition : $\kappa(A) = \|A^{-1}\| \cdot \|A\|$

δA perturbation de A et δb perturbation de b une perturbation δb.

δx ?

Le conditionnement d'un système linéaire

Définition : $\kappa(A) = \|A^{-1}\| \cdot \|A\|$

δA perturbation de A et δb perturbation de b une perturbation δb.

δx ?

$$\frac{\|\delta x\|}{\|x\|} \leq \frac{\kappa(A)}{1 - \kappa(A)\|\delta A\|/\|A\|} \left(\frac{\|\delta A\|}{\|A\|} + \frac{\|\delta b\|}{\|b\|} \right).$$

Le conditionnement d'un système linéaire

Définition : $\kappa(A) = \|A^{-1}\| \cdot \|A\|$

δA perturbation de A et δb perturbation de b une perturbation δb.

δx ?

$$\frac{\|\delta x\|}{\|x\|} \leq \frac{\kappa(A)}{1 - \kappa(A)\|\delta A\|/\|A\|} \left(\frac{\|\delta A\|}{\|A\|} + \frac{\|\delta b\|}{\|b\|} \right).$$

Les normes $\|\cdot\|_\infty$ et $\|\cdot\|_1$ sont faciles à calculer :

- $\|A\|_\infty = \max_{1 \leq i \leq n} (\sum_{j=1}^{n} |A_{ij}|)$
- $\|A\|_1 = \max_{1 \leq j \leq n} (\sum_{i=1}^{n} |A_{ij}|)$.

Le conditionnement d'un système linéaire

Définition : $\kappa(A) = \|A^{-1}\| \cdot \|A\|$

δA perturbation de A et δb perturbation de b une perturbation δb.

δx ?

$$\frac{\|\delta x\|}{\|x\|} \leq \frac{\kappa(A)}{1 - \kappa(A)\|\delta A\|/\|A\|} \left(\frac{\|\delta A\|}{\|A\|} + \frac{\|\delta b\|}{\|b\|} \right).$$

Les normes $\|\cdot\|_\infty$ et $\|\cdot\|_1$ sont faciles à calculer :

- $\|A\|_\infty = \max_{1 \leq i \leq n} (\sum_{j=1}^{n} |A_{ij}|)$
- $\|A\|_1 = \max_{1 \leq j \leq n} (\sum_{i=1}^{n} |A_{ij}|)$.
- $\|A\|_2$ ne s'obtient pas facilement ! $\|A\|_2 = \sqrt{\rho(A^t \cdot A)}$. (matrice symétrique : quotient des modules des valeurs propres extrêmes).

Le diable...

Mauvais conditionnement ($\kappa(A)$ grand) =>

- imprécision des solutions,
- convergence lente des méthodes itératives (solution de systèmes, valeurs propres).

Le diable...

Mauvais conditionnement ($\kappa(A)$ grand) =>

- imprécision des solutions,
- convergence lente des méthodes itératives (solution de systèmes, valeurs propres).

Tous les systèmes *de la vie réelle* sont mal conditionnés !

Le diable...

Mauvais conditionnement ($\kappa(A)$ grand) =>

- imprécision des solutions,
- convergence lente des méthodes itératives (solution de systèmes, valeurs propres).

Tous les systèmes *de la vie réelle* sont mal conditionnés !

Exemple : matrice de $-\Delta + \lambda I$ en différences finies de pas h : $\kappa(A) = O(h^2)$.

Note : indépendant de la dimension.

Une structure de données “universelle”

Format CSL :

$$\begin{vmatrix} 1 & 2 & 0 & 0 \\ 0 & 3 & 9 & 0 \\ 0 & 1 & 4 & 0 \end{vmatrix}$$

Une structure de données “universelle”

Format CSL :

$$\begin{vmatrix} 1 & 2 & 0 & 0 \\ 0 & 3 & 9 & 0 \\ 0 & 1 & 4 & 0 \end{vmatrix}$$

$$A = \begin{vmatrix} 1 & 2 & 3 & 9 & 1 & 4 \end{vmatrix}$$

Une structure de données “universelle”

Format CSL :

$$\begin{vmatrix} 1 & 2 & 0 & 0 \\ 0 & 3 & 9 & 0 \\ 0 & 1 & 4 & 0 \end{vmatrix}$$

$$A = \begin{vmatrix} 1 & 2 & 3 & 9 & 1 & 4 \end{vmatrix}$$
$$JA = \begin{vmatrix} 0 & 1 & 1 & 2 & 1 & 2 \end{vmatrix}$$

Une structure de données “universelle”

Format CSL :

$$\begin{vmatrix} 1 & 2 & 0 & 0 \\ 0 & 3 & 9 & 0 \\ 0 & 1 & 4 & 0 \end{vmatrix}$$

$$\begin{aligned} A &= \begin{vmatrix} 1 & 2 & 3 & 9 & 1 & 4 \end{vmatrix} \\ JA &= \begin{vmatrix} 0 & 1 & 1 & 2 & 1 & 2 \end{vmatrix} \\ IA &= \begin{vmatrix} 0 & 1 & 4 & 6 \end{vmatrix} \end{aligned}$$

Une structure de données "universelle"

Format CSL :

$$\begin{vmatrix} 1 & 2 & 0 & 0 \\ 0 & 3 & 9 & 0 \\ 0 & 1 & 4 & 0 \end{vmatrix}$$

$$\begin{aligned} A &= \begin{vmatrix} 1 & 2 & 3 & 9 & 1 & 4 \end{vmatrix} \\ JA &= \begin{vmatrix} 0 & 1 & 1 & 2 & 1 & 2 \end{vmatrix} \\ IA &= \begin{vmatrix} 0 & 1 & 4 & 6 \end{vmatrix} \end{aligned}$$

Le format CSR s'obtient en classant les coefficients par colonnes.

Une structure de données "universelle"

Format CSL :

$$\begin{vmatrix} 1 & 2 & 0 & 0 \\ 0 & 3 & 9 & 0 \\ 0 & 1 & 4 & 0 \end{vmatrix}$$

$$\begin{aligned} A &= \begin{vmatrix} 1 & 2 & 3 & 9 & 1 & 4 \end{vmatrix} \\ JA &= \begin{vmatrix} 0 & 1 & 1 & 2 & 1 & 2 \end{vmatrix} \\ IA &= \begin{vmatrix} 0 & 1 & 4 & 6 \end{vmatrix} \end{aligned}$$

Le format CSR s'obtient en classant les coefficients par colonnes.
!!! $y = Ax$ **et** $y = A^t x$!!!

Construire une matrice CSL (CSR) : une astuce C++

Standard template library : `map` et `pair`.

Construire une matrice CSL (CSR) : une astuce C++

Standard template library : `map` et `pair`.

- `map` : modélise une application d'un ensemble ordonné (E) dans un ensemble (X),
- `pair` : les paires d'objets ordonnés sont munies de l'ordre lexicographique.

```
map<pair<int,int>,double> M;
M[make_pair(i,j)]=1.0;
```

Construire une matrice CSL (CSR) : une astuce C++

Standard template library : `map` et `pair`.

- `map` : modélise une application d'un ensemble ordonné (E) dans un ensemble (X),
- `pair` : les paires d'objets ordonnés sont munies de l'ordre lexicographique.

```
map<pair<int,int>,double> M;
M[make_pair(i,j)]=1.0;
```

Ensuite, l'itérateur associé permet de parcourir la map `M` dans l'ordre ad'hoc => construction facile de la matrice CSR (ou CSL).

Derrière : arbres B (B-trees, arbres équilibrés)=> performant.
Note : on peut changer l'ordre sur `pair`.

Construire une matrice CSL (CSR) : une astuce C++

Standard template library : `map` et `pair`.

- `map` : modélise une application d'un ensemble ordonné (E) dans un ensemble (X),
- `pair` : les paires d'objets ordonnés sont munies de l'ordre lexicographique.

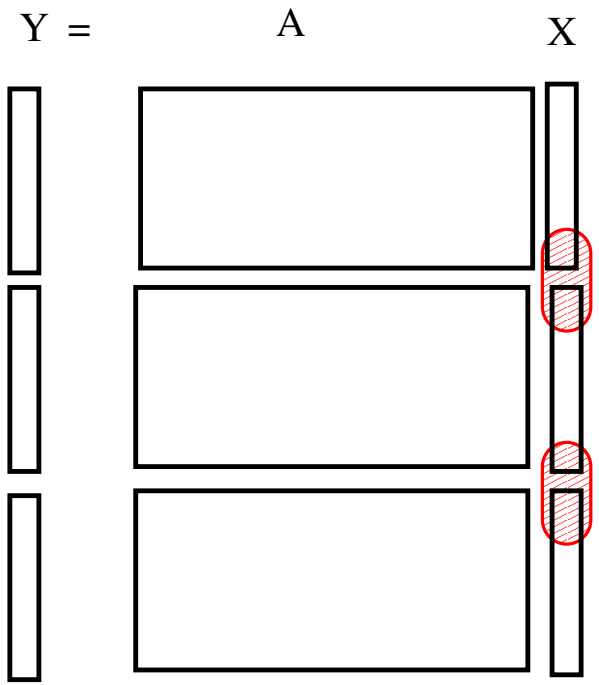
```
map<pair<int,int>,double> M;
M[make_pair(i,j)]=1.0;
```

Ensuite, l'itérateur associé permet de parcourir la map `M` dans l'ordre ad'hoc => construction facile de la matrice CSR (ou CSL).

Derrière : arbres B (B-trees, arbres équilibrés)=> performant.
Note : on peut changer l'ordre sur `pair`.
En fortran : double parcours du graphe.

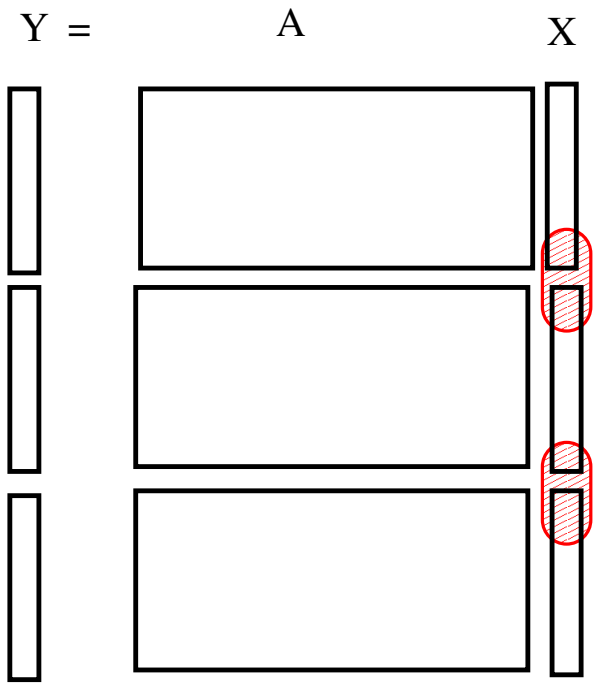
Parallélisation

Le produit matrice x vecteur :



Parallélisation

Le produit matrice x vecteur :



Plus délicat : résolution de systèmes triangulaires.

Rappel : différences finies pour l'équation de Poisson

$$-\Delta u = f.$$

Maillage à pas constant h en dimension 2 ou 3.

u_{ij}

Rappel : différences finies pour l'équation de Poisson

$$-\Delta u = f.$$

Maillage à pas constant h en dimension 2 ou 3.

u_{ij}

Dérivée par rapport à x : $\frac{u_{i+1,j}-u_{i,j}}{h}$.

Rappel : différences finies pour l'équation de Poisson

$$-\Delta u = f.$$

Maillage à pas constant h en dimension 2 ou 3.

u_{ij}

Dérivée par rapport à x : $\frac{u_{i+1,j}-u_{i,j}}{h}$.

Dérivée par rapport à x : $\frac{u_{i+1,j}-2u_{i,j}+u_{i-1,j}}{h^2}$.

Rappel : différences finies pour l'équation de Poisson

$$-\Delta u = f.$$

Maillage à pas constant h en dimension 2 ou 3.

u_{ij}

Dérivée par rapport à x : $\frac{u_{i+1,j}-u_{i,j}}{h}$.

Dérivée par rapport à x : $\frac{u_{i+1,j}-2u_{i,j}+u_{i-1,j}}{h^2}$.

$-\Delta$ en 2d :

$$\frac{-u_{i+1,j} - u_{i,j+1} + 4u_{i,j} - u_{i-1,j} - u_{i,j-1}}{h^2}$$

Rappel : différences finies pour l'équation de Poisson

$$-\Delta u = f.$$

Maillage à pas constant h en dimension 2 ou 3.

u_{ij}

Dérivée par rapport à x : $\frac{u_{i+1,j}-u_{i,j}}{h}$.

Dérivée par rapport à x : $\frac{u_{i+1,j}-2u_{i,j}+u_{i-1,j}}{h^2}$.

$-\Delta$ en 2d :

$$\frac{-u_{i+1,j} - u_{i,j+1} + 4u_{i,j} - u_{i-1,j} - u_{i,j-1}}{h^2}$$

Matrice creuse : $O(n)$ termes non nuls (n : ordre de la matrice).

Rappel : différences finies pour l'équation de Poisson

$$-\Delta u = f.$$

Maillage à pas constant h en dimension 2 ou 3.

u_{ij}

Dérivée par rapport à x : $\frac{u_{i+1,j}-u_{i,j}}{h}$.

Dérivée par rapport à x : $\frac{u_{i+1,j}-2u_{i,j}+u_{i-1,j}}{h^2}$.

$-\Delta$ en 2d :

$$\frac{-u_{i+1,j} - u_{i,j+1} + 4u_{i,j} - u_{i-1,j} - u_{i,j-1}}{h^2}$$

Matrice creuse : $O(n)$ termes non nuls (n : ordre de la matrice).

$5n$ coefficients non nuls en 2d, $7n$ en 3d.

Variations sur la décomposition LU

La méthode de Gauß : transformer la matrice A (d'ordre n) pour faire apparaître des coefficients nuls sous la diagonale de la première colonne, puis de la deuxième colonne et ainsi de suite.

Variations sur la décomposition LU

La méthode de Gauß : transformer la matrice A (d'ordre n) pour faire apparaître des coefficients nuls sous la diagonale de la première colonne, puis de la deuxième colonne et ainsi de suite.

La décomposition LU :

$$L_{n-1} \dots L_2 L_1 A = U.$$

Variations sur la décomposition LU

La méthode de Gauß : transformer la matrice A (d'ordre n) pour faire apparaître des coefficients nuls sous la diagonale de la première colonne, puis de la deuxième colonne et ainsi de suite.

La décomposition LU :

$$L_{n-1} \dots L_2 L_1 A = U.$$

$$L = (L_{n-1} \dots L_2 L_1)^{-1}$$

Variations sur la décomposition LU

La méthode de Gauß : transformer la matrice A (d'ordre n) pour faire apparaître des coefficients nuls sous la diagonale de la première colonne, puis de la deuxième colonne et ainsi de suite.
La décomposition LU :

$$L_{n-1} \dots L_2 L_1 A = U.$$

$$L = (L_{n-1} \dots L_2 L_1)^{-1}$$

$$A = LU$$

L est triangulaire inférieure à diagonale unité.

Variations sur la décomposition LU

La méthode de Gauß : transformer la matrice A (d'ordre n) pour faire apparaître des coefficients nuls sous la diagonale de la première colonne, puis de la deuxième colonne et ainsi de suite.
La décomposition LU :

$$L_{n-1} \dots L_2 L_1 A = U.$$

$$L = (L_{n-1} \dots L_2 L_1)^{-1}$$

$$A = LU$$

L est triangulaire inférieure à diagonale unité.
On a négligé les problèmes de pivot : $A = PLU$.

Décomposition LU : problèmes et solutions

- la factorisation (= le calcul de L et U) fait apparaître de nouveaux termes non nuls => taille mémoire importante. Renumérotations (P) nécessaire pour minimiser le nombre de nouveaux termes.

Décomposition LU : problèmes et solutions

- la factorisation (= le calcul de L et U) fait apparaître de nouveaux termes non nuls => taille mémoire importante. Renumérotations (P) nécessaire pour minimiser le nombre de nouveaux termes.
- coût de la factorisation.

Décomposition LU : problèmes et solutions

- la factorisation (= le calcul de L et U) fait apparaître de nouveaux termes non nuls => taille mémoire importante. Renumérotations (P) nécessaire pour minimiser le nombre de nouveaux termes.
- coût de la factorisation.

Années 70–80 : abandon au profit des méthodes itératives.
Retour en grâce relatif des méthodes directes.

Décomposition LU : le retour !

Toute la difficulté est dans la factorisation !

Décomposition LU : le retour !

Toute la difficulté est dans la factorisation !

1. algorithmes de renumérotation plus efficace (mais problème np-complet),
2. fabrication de sous blocs pleins et appel des `BLAS`.
3. technique multifrontale.
4. etc.

Décomposition LU : le retour !

Toute la difficulté est dans la factorisation !

1. algorithmes de renumérotation plus efficace (mais problème np-complet),
2. fabrication de sous blocs pleins et appel des `BLAS`.
3. technique multifrontale.
4. etc.

`BLAS` Basic Linear Algebra Subroutine : exemple ATLAS, GOTO.

Décomposition LU : SuperLU, un programme à tout faire

Première implantation moderne, améliorée constamment.

Décomposition LU : SuperLU, un programme à tout faire

Première implantation moderne, améliorée constamment.

+

- en C, interfaces Fortran.

Décomposition LU : SuperLU, un programme à tout faire

Première implantation moderne, améliorée constamment.

- en C, interfaces Fortran.
- disponible dans les distributions Linux,
- utilisée par ne nombreux logiciels (Matlab, Scipy...)
- très fiable.
- mode “matrice symétrique”.

Décomposition LU : SuperLU, un programme à tout faire

Première implantation moderne, améliorée constamment.

+

- en C, interfaces Fortran.
- disponible dans les distributions Linux,
- utilisée par ne nombreux logiciels (Matlab, Scipy...)
- très fiable.
- mode “matrice symétrique”.

-

- interface désagréable (très “C”),
- variations de l'interface d'une version à l'autre, ainsi que des “include files”.

Décomposition LU : SuperLU, retour d'expérience

- performances remarquables, en tout cas en séquentiel.
- parfait pour résoudre une suite de systèmes identiques.
- Exemple de $\Delta U = F$:
  - 2d : ok jusqu'à 150 000 inconnues ou plus.
  - 3d, ou 10^6 inconnues : trop lent, trop de mémoire.
- la dernière version calcule aussi des factorisations incomplètes.

Décomposition LU : SuperLU, retour d'expérience

- performances remarquables, en tout cas en séquentiel.
- parfait pour résoudre une suite de systèmes identiques.
- Exemple de $\Delta U = F$:
  - 2d : ok jusqu'à 150 000 inconnues ou plus.
  - 3d, ou 10^6 inconnues : trop lent, trop de mémoire.
- la dernière version calcule aussi des factorisations incomplètes.

Un outil de base parfait

Décomposition LU : SuperLU, retour d'expérience

- performances remarquables, en tout cas en séquentiel.
- parfait pour résoudre une suite de systèmes identiques.
- Exemple de $\Delta U = F$:
  - 2d : ok jusqu'à 150 000 inconnues ou plus.
  - 3d, ou 10^6 inconnues : trop lent, trop de mémoire.
- la dernière version calcule aussi des factorisations incomplètes.

Un outil de base parfait

A utiliser :
-en mode mise au point (plutôt que des méthodes itératives),
-pour des problèmes de taille raisonnable.

Décomposition LU : autres implantations

Solveurs parallèles :
-MUMPS : `http://graal.ens-lyon.fr/MUMPS/`

-PASTIX : `http://dept-info.labri.u-bordeaux.fr/~ramet/pastix/` ; basé sur le partitionneur de graphes SCOTCH.

-PARDISO : `http://www.pardiso-project.org/`, gratuit mais *pas libre.*

Méthode historique de Gauß–Seidel

$$A = L + U$$

$$LX_{n+1} = -AX_n + b.$$

Méthode historique de Gauß–Seidel

$$A = L + U$$

$$LX_{n+1} = -AX_n + b.$$

Convergence extrêmement lente, mais programmation facile. A oublier pour la résolution de systèmes, mais on en verra des applications intéressantes.

Méthodes de Krylov

x_0.

Méthodes de Krylov

x_0.

$g_0 = Ax_0 - b$.

Méthodes de Krylov

x_0.

$g_0 = Ax_0 - b$.

Espace de Krylov d'ordre p :

$$K_p = Vect\{g_0, Ag_0, A^2 g_0, \ldots, A^{p-1} g_0\}.$$

Méthodes de Krylov

x_0.

$g_0 = Ax_0 - b$.

Espace de Krylov d'ordre p :

$$K_p = Vect\{g_0, Ag_0, A^2 g_0, \ldots, A^{p-1} g_0\}.$$

Idée : x_n minimiser *quelque chose* sur K_n .

GC et GMRES

1. la méthode du gradient conjugué (GC) : uniquement pour les problèmes symétriques définis positifs .
   $e_{n+1} = b - Ax_{n+1}$,

GC et GMRES

1. la méthode du gradient conjugué (GC) : uniquement pour les problèmes symétriques définis positifs .
   $e_{n+1} = b - Ax_{n+1}$, $E_{n+1} = e_{n+1}^t A e_{n+1}$,

GC et GMRES

1. la méthode du gradient conjugué (GC) : uniquement pour les problèmes symétriques définis positifs .
   $e_{n+1} = b - Ax_{n+1}$, $E_{n+1} = e_{n+1}^t A e_{n+1}$,
   $x_{n+1} \in x_0 + K_n$ qui minimise E_{n+1}.

   Note : $(x^t A x)^{1/2}$ est une norme.

GC et GMRES

1. la méthode du gradient conjugué (GC) : uniquement pour les problèmes symétriques définis positifs .
   $e_{n+1} = b - Ax_{n+1}$, $E_{n+1} = e_{n+1}^t A e_{n+1}$,
   $x_{n+1} \in x_0 + K_n$ qui minimise E_{n+1}.

   Note : $(x^t A x)^{1/2}$ est une norme.
2. la méthode du résidu minimal généralisé pour les problèmes non symétriques :

$$x_{n+1} = \mathrm{argmin}\ \|\mathrm{b} - \mathrm{A}\ \mathrm{x}\|_2;\ \mathrm{x} \in \mathrm{x}_0 + \mathrm{K}_\mathrm{n}.$$

GC et GMRES

Propriété importante : les seules expressions ou A intervient sont des produits matrices $y = Ax$.

Conséquences :

- Parallélisation MPI relativement facile,
- On n'a pas forcément besoin de connaître A, mais seulement l'action de A sur un vecteur.

Convergence

1. GC : $E_m = \sqrt{g_m^t A g_m}$ (erreur dans la norme définie par A).

$$E_m \leq E_0 \, (\frac{\sqrt{\kappa - 1}}{\sqrt{\kappa + 1}})^m.$$

Convergence

1. GC : $E_m = \sqrt{g_m^t A g_m}$ (erreur dans la norme définie par A).

$$E_m \leq E_0 \ (\frac{\sqrt{\kappa - 1}}{\sqrt{\kappa + 1}})^m.$$

2. GMRES : étude beaucoup plus délicate ! Si A est diagonalisable $A = X \Lambda X^{-1}$:

$$||r_m||_2 \leq ||r_0||_2 \ \kappa_2(X) \ \epsilon(m),$$

$$\epsilon(m) = \min_{p \in P_m, p(0)=1} \ \max_{i=1,\ldots,n} |p(\lambda_i|.$$

Préconditionnement

Idée : trouver une matrice K telle que :

1. $K^{-1}A$ soit bien conditionnée,

Préconditionnement

Idée : trouver une matrice K telle que :

1. $K^{-1}A$ soit bien conditionnée,
2. le système $Ky = F$ soit facile à résoudre.
3. ... et résoudre $K^{-1}Ax = K^{-1}b$.

Préconditionnement

Idée : trouver une matrice K telle que :

1. $K^{-1}A$ soit bien conditionnée,
2. le système $Ky = F$ soit facile à résoudre.
3. ... et résoudre $K^{-1}Ax = K^{-1}b$.

Différents préconditionneurs :

- factorisations incomplètes,
- solveurs approchés,
- préconditionnement ad'hoc.

Factorisations incomplètes

Idée de base : La factorisation LU crée de nombreux termes non nuls.

On se donne un ensemble I de paires indices et on ne construit les coefficients $L_{i,j}$ et $U_{i,j}$ que pour $(i,j) \in I$.

Factorisations incomplètes

Idée de base : La factorisation LU crée de nombreux termes non nuls.
On se donne un ensemble I de paires indices et on ne construit les coefficients $L_{i,j}$ et $U_{i,j}$ que pour $(i,j) \in I$.

Le plus simple : $I = \{(i,j) \mid A_{i,j} \neq 0\}$.
Nombreuses améliorations en laissant grossir intelligemment I.

Factorisations incomplètes

Idée de base : La factorisation LU crée de nombreux termes non nuls.
On se donne un ensemble I de paires indices et on ne construit les coefficients $L_{i,j}$ et $U_{i,j}$ que pour $(i,j) \in I$.

Le plus simple : $I = \{(i,j) \mid A_{i,j} \neq 0\}$.
Nombreuses améliorations en laissant grossir intelligemment I.

- peu de coefficients => peu coûteux.
- peu de coefficients => taille mémoire faible.

Bibliothèques

(tous les codes sont libres (GPL, CCIL)).

- Codes de Y. Saad (Mr GMRES) :
  `http://www-users.cs.umn.edu/` : sparskit, parms , itsol.
- HIPS `http://hips.gforge.inria.fr/`.
- PETSC `http://www.mcs.anl.gov/petsc/petsc-as/`.
- HYPRE `http://acts.nersc.gov/hypre/`.

En bleu les codes parallèles.

Méthodes Multigrilles : ingrédient 1

Notion de lisseur.

Rappel : Méthode de Gauß–Seidel : $A = L + U$;

Méthodes Multigrilles : ingrédient 1

Notion de lisseur.

Rappel : Méthode de Gauß–Seidel : $A = L + U$;

$LX_{n+1} = -AX_n + b$.

Méthodes Multigrilles : ingrédient 1

Notion de lisseur.

Rappel : Méthode de Gauß–Seidel : $A = L + U$;

$LX_{n+1} = -AX_n + b$.

Application à la matrice de Δ en différences finies :

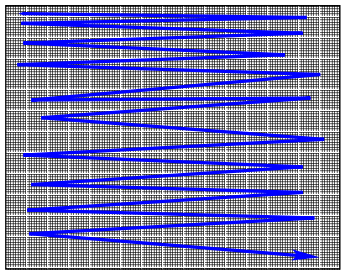
$$u_{i-1,j} + u_{i,j-1} - 4u_{i,j} + u_{i+1,j} + u_{i,j+1} = h^2 \cdot b_{i,j}.$$

Méthodes Multigrilles : ingrédient 1

Notion de lisseur.

Rappel : Méthode de Gauß–Seidel : $A = L + U$;
$LX_{n+1} = -AX_n + b$.
Application à la matrice de Δ en différences finies :

$$u_{i-1,j} + u_{i,j-1} - 4u_{i,j} + u_{i+1,j} + u_{i,j+1} = h^2 \cdot b_{i,j}.$$



$$u^*_{i,j} = h^2 \cdot b_{i,j} - \frac{1}{4}(u_{i-1,j} + u_{i,j-1} + u_{i+1,j} + u_{i,j+1}).$$

Méthodes Multigrilles : ingrédient 1

Lisseur : analyse de Fourier sur une grille régulière :

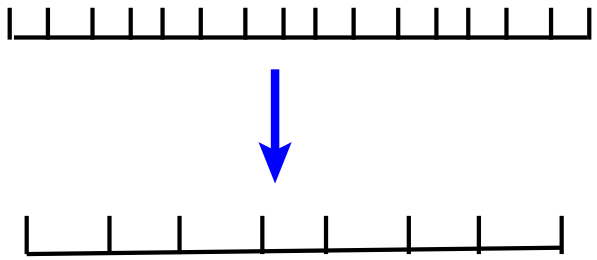
- les harmoniques spatiales de haute fréquence de l'erreur $b - Au$ tendent rapidement vers zéro,
- les harmoniques de basse fréquence tendent lentement vers zéro.

Méthodes Multigrilles : ingrédient 1

Lisseur : analyse de Fourier sur une grille régulière :

- les harmoniques spatiales de haute fréquence de l'erreur $b - Au$ tendent rapidement vers zéro,
- les harmoniques de basse fréquence tendent lentement vers zéro.

Après quelques itérations, on peut approcher l'erreur sur une grille plus grossière :



Méthodes Multigrilles : ingrédients 2 et 3

$$e = b - Ax.$$

$$x^* = x + A^{-1}e =$$

Méthodes Multigrilles : ingrédients 2 et 3

$$e = b - Ax.$$

$$x^* = x + A^{-1}e = A^{-1}b.$$

Méthodes Multigrilles : ingrédients 2 et 3

$$e = b - Ax.$$

$$x^* = x + A^{-1}e = A^{-1}b.$$

Dernier ingrédient (3) :

- restriction (exemple : moyenne) de la grille fine vers la grille grossière,
- opération inverse (prolongement). En général la transposée de la restriction.

Méthodes Multigrilles

Passage de x_n à x_{n+1}.

1. lissage sur la grille fine,

Méthodes Multigrilles

Passage de x_n à x_{n+1}.

1. lissage sur la grille fine,
2. calcul de l'erreur e sur la grille fine,

Méthodes Multigrilles

Passage de x_n à x_{n+1}.

1. lissage sur la grille fine,
2. calcul de l'erreur e sur la grille fine,
3. restriction de l'erreur vers la grille grossière e ,

Méthodes Multigrilles

Passage de x_n à x_{n+1}.

1. lissage sur la grille fine,
2. calcul de l'erreur e sur la grille fine,
3. restriction de l'erreur vers la grille grossière e ,
4. (!) résolution sur la grille grossière de $Ac = e$,

Méthodes Multigrilles

Passage de x_n à x_{n+1}.

1. lissage sur la grille fine,
2. calcul de l'erreur e sur la grille fine,
3. restriction de l'erreur vers la grille grossière e ,
4. (!) résolution sur la grille grossière de $Ac = e$,
5. prolongement de c à la grille fine (=c),

Méthodes Multigrilles

Passage de x_n à x_{n+1}.

1. lissage sur la grille fine,
2. calcul de l'erreur e sur la grille fine,
3. restriction de l'erreur vers la grille grossière e ,
4. (!) résolution sur la grille grossière de $Ac = e$,
5. prolongement de c à la grille fine (=c),
6. $x_{n+1} = x_n + c$ sur la grille fine.

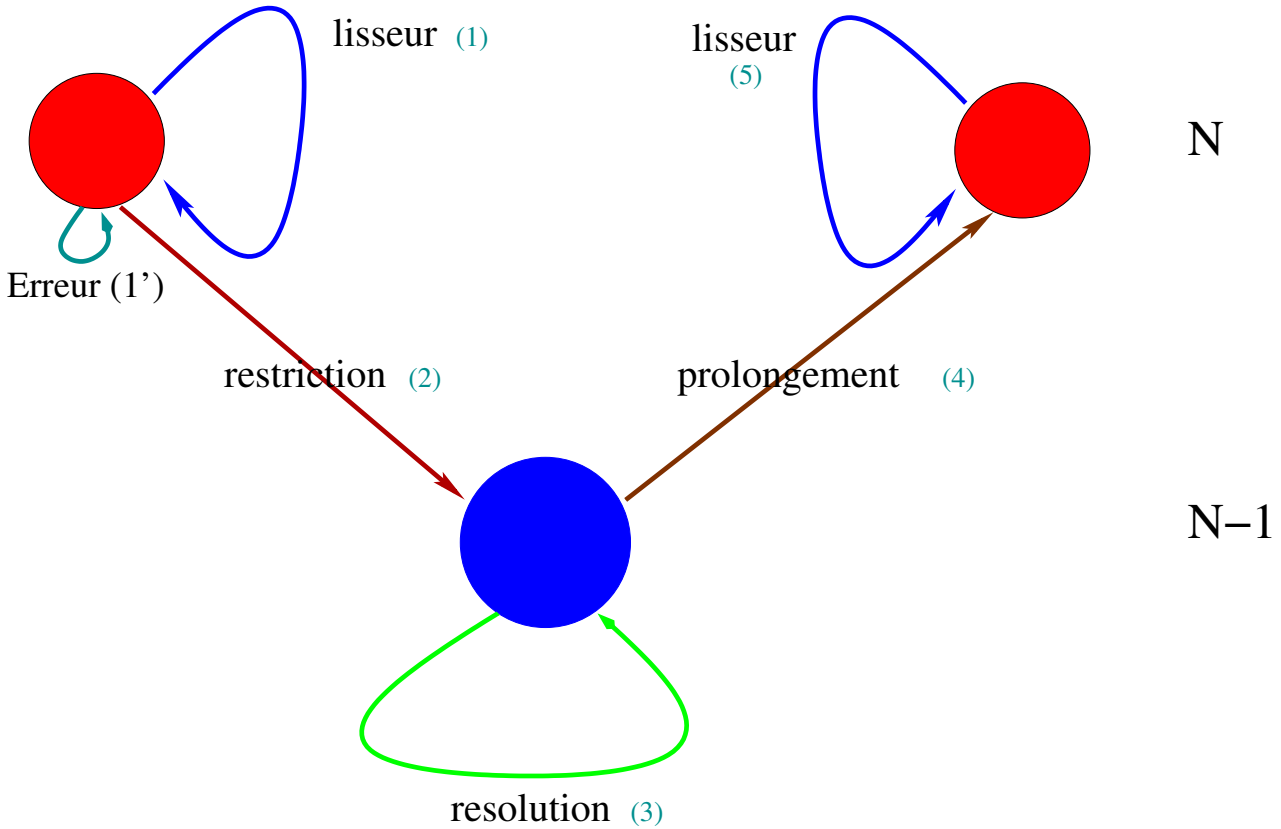
Méthodes Multigrilles

Passage de x_n à x_{n+1}.

1. lissage sur la grille fine,
2. calcul de l'erreur e sur la grille fine,
3. restriction de l'erreur vers la grille grossière e ,
4. (!) résolution sur la grille grossière de $Ac = e$,
5. prolongement de c à la grille fine (=c),
6. $x_{n+1} = x_n + c$ sur la grille fine.

Algorithme récursif : résolution de (4 !) par la même méthode....
jusqu'à une grille très grossière.

Méthodes Multigrilles



Méthodes Multigrilles

Nombreuses variantes :

- lisseur,
- parcours des grilles,
- cas de grilles non uniformes
- etc.

Importante littérature.

Insuffisance des méthodes multigrilles

-Très “différences finies”.
-Maillages structurés.
-Loin d'être des boites noires.

Peut-on généraliser les méthodes multigrilles à des problèmes “abstraits” $Ax = b$?

Méthodes Multigrilles Algébriques

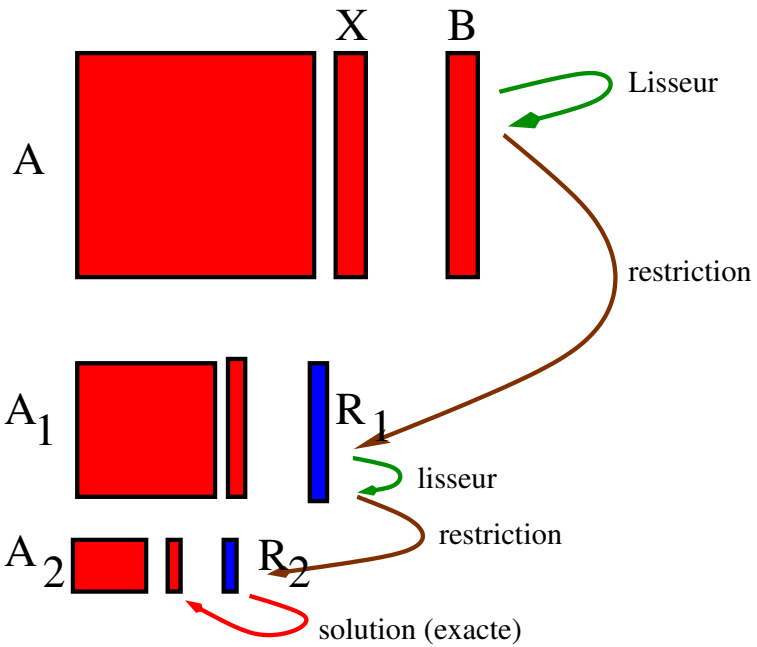
Définir une hiérarchie de matrices,

Méthodes Multigrilles Algébriques

Définir une hiérarchie de matrices, un lisseur,

Méthodes Multigrilles Algébriques

Définir une hiérarchie de matrices, un lisseur, une restriction et un prolongement.



Exemple : AGMG Y. Notay

AG gregation M ultiG rid.

1. Définir des agrégats disjoints de variables G_j.
2. Pour une matrice A, on définit A_c :

$$(A_c)_{i,j} = \sum_{k \in G_i} \sum_{l \in G_j} a_{kl}.$$

Soit encore en posant $P_{i,j} = 1$ ssi $i \in G_j$:

$$A_c = P^t A P.$$

3. Restriction : P^t ; Prolongement : P.
4. Lisseur : relaxation.

Exemple : AGMG Y. Notay

Mais comment fabriquer les agrégats ? sur quel critère ?

Exemple : AGMG Y. Notay

Mais comment fabriquer les agrégats ? sur quel critère ?

On définit l'ensemble S_i des nœuds fortement liés au nœud i :

$$G_i = \{j \neq i |\ a_{i,j} < -\beta \max_{a_{i,k}<0} |a_{i,k}|\}.$$

Typiquement $\beta = 0.25$.

Exemple : AGMG Y. Notay

Mais comment fabriquer les agrégats ? sur quel critère ?

On définit l'ensemble S_i des nœuds fortement liés au nœud i :

$$G_i = \{j \neq i |\ a_{i,j} < -\beta \max_{a_{i,k}<0} |a_{i,k}|\}.$$

Typiquement $\beta = 0.25$.
Dans la méthode AGMG, on "grossit" (coarsen) deux fois à chaque étage.

Exemple : AGMG Y. Notay

Mais comment fabriquer les agrégats ? sur quel critère ?

On définit l'ensemble S_i des nœuds fortement liés au nœud i :

$$G_i = \{j \neq i |\ a_{i,j} < -\beta \max_{a_{i,k}<0} |a_{i,k}|\}.$$

Typiquement $\beta = 0.25$.
Dans la méthode AGMG, on "grossit" (coarsen) deux fois à chaque étage.

On utilise cette méthode comme préconditionneurs du GC ou de GMRES.

Exemple : AGMG Y. Notay

Nombreux détails d'implantation !

Programme en Fortran.

Version parallèle (MPI) : en dessous d'une certaine taille de matrices, on utilise une méthode directe parallélisée (MUMPS).

Facile à utiliser

```
 subroutine
dagmg(n,a,ja,ia,f,x,ijob,iprint,nrest,iter,tol)
```

- `n,a,ja,ia` : la matrice.
- `f` : second membre.
- `x` : point de départ des itérations et solution après execution.
- `ijob` : type de travail : préprocessing et/ou solution.

Facile à utiliser

```
 subroutine
dagmg(n,a,ja,ia,f,x,ijob,iprint,nrest,iter,tol)
```

- `n,a,ja,ia` : la matrice.
- `f` : second membre.
- `x` : point de départ des itérations et solution après execution.
- `ijob` : type de travail : préprocessing et/ou solution.
- `nrest` : type de solveur (FGC ou autre).
- `iter` : nombre d'itérations (in/out).
- `tol` : test d'arrêt.