Résolution de systèmes linéaires creux par des méthodes directes

J. Erhel

Janvier 2014

1 Stockage des matrices creuses

Dans de nombreuses simulations numériques, la discrétisation du problème aboutit à manipuler une matrice de très grande taille (d'ordre pouvant aller jusqu'à 10^8) mais dont la plupart des coefficients sont nuls. Dans ce cas, on considère un stockage creux (par abus, on parle d'une matrice creuse) qui permet de ne pas stocker une grande partie des coefficients nuls et de réduire considérablement la complexité des résolutions de systèmes.

Les stockages courants sont :

le stockage bande : on stocke toutes les diagonales situées entre les deux diagonales contenant les éléments non nuls les plus éloignés de la diagonale principale comme colonnes d'une matrice rectangulaire dont le nombre de lignes est égal à l'ordre de la matrice creuse tandis que le nombre de colonnes de la matrice est égal à la largeur de la bande.

le stockage profil : Il peut arriver que les éléments non nuls extrêmes de chaque ligne soient à des distances de la diagonale très différentes suivant les lignes. Un stockage bande oblige alors à stocker un grand nombre d'éléments non nuls. Pour diminuer cet effet, on peut stocker les portions de chaque ligne situées entre ces éléments extrêmes.

le stockage par coordonnées (COO): dans un tableau à une dimension, on range tous les éléments non nuls de la matrice. Les indices de ligne et de colonne de chacun de ces éléments sont stockés dans le même ordre dans deux tableaux d'entiers à une dimension. C'est le stockage de référence dans MATLAB.

le stockage compact par lignes (CSR) : dans un tableau à une dimension on range tous les éléments non nuls par ligne, une ligne après l'autre. Les indices de colonnes et les limites de lignes sont retenus dans deux tableaux d'entiers.

le stockage compact par colonnes (CSC) : ce format est le même que le précédent, mais en remplaçant le rôle des lignes par celui des colonnes.

Il existe encore d'autres stockages compacts, par exemple par diagonales creuses ou en définissant des blocs.

La bibliothèque LAPACK permet le stockage bande. Pour les autres stockages creux, il faut utiliser une autre bibliothèque. Ces solveurs directs ou itératifs utilisent en général un stockage compact, par lignes ou par colonnes.

2 Produit matrice-vecteur

Dans les méthodes itératives de résolution, les deux opérations les plus coûteuses sont le produit matrice-vecteur et le préconditionnement. Les stockages creux permettent de réduire la complexité du produit matrice-vecteur. Dans ce qui suit, on considère un stockage compact par lignes.

On suppose que la matrice A d'ordre n a $nz(A)$ éléments non nuls que l'on a rangés dans le tableau `a(1:nz)` en les énumérant par lignes. Dans le tableau `ja(1:nz)` on range les indices de colonnes de ces éléments dans le même ordre et dans le tableau `ia(1:n+1)` on indique la liste des indices des démarrages de lignes de la matrice A stockée dans `a` avec par convention la valeur `nz+1` dans `ia(n+1)`. Etant donné deux vecteurs $x,\ y\ \in \mathbb{R}^n$, l'opération

$$y := y + A\ x$$

s'exprime par l'algorithme 1.

ALGORITHM 1: Produit matrice vecteur (stockage compact par lignes)

```
do i = 1, n
    do k = ia(i), ia(i+1) - 1
        y(i) = y(i) + a(k) * x(ja(k))
    enddo
enddo
```

La boucle interne est un produit scalaire creux; il met en jeu un *gather* (lecture indirecte). Si on avait considéré le stockage compact par colonnes, on serait arrivé à une boucle comportant un *scatter* (écriture indirecte). Ces opérations sont optimisées sur les calculateurs d'aujourd'hui.

La complexité passe de $O(n^2)$ pour l'opération BLAS2 standard à $O(nz(A))$ pour le produit en mode creux.

3 Factorisation de Cholesky

Soit A une matrice symétrique définie positive d'ordre n. On veut résoudre le système linéaire $Ax = b$. On sait qu'on peut appliquer la factorisation de Cholesky $A = LL^T$.

La matrice L a aussi une structure creuse, qu'on va utiliser pour réduire le nombre d'opérations. Du fait que la factorisation de Cholesky ne nécessite pas de pivot, donc pas de permutation des lignes ou colonnes en cours de calcul, il est possible de prévoir à l'avance la structure creuse du facteur de Cholesy L.

3.1 Opérations cdiv et cmod dans Cholesky

Introduisons les opérations $cdiv(j)$ et $cmod(j,k)$ définies par les algorithmes 2 et 3:

ALGORITHM 2: opération cdiv(j)

$L(j,j) = \sqrt{L(j,j)}$
do $i = j+1, n$
  $L(i,j) = L(i,j)/L(j,j)$
enddo

ALGORITHM 3: opération cmod(j,k)

do $i = k, n$
  $L(i,k) = L(i,k) - L(k,j) * L(i,j)$
enddo

L'algorithme de Cholesky, version right-looking, s'écrit alors, pour une matrice pleine 4:

ALGORITHM 4: Cholesky right-looking avec opérations cdiv et cmod

do $j = 1, n$
  $cdiv(j)$
  do $k = j+1, n$
    $cmod(j,k)$
  enddo
enddo

3.2 Cholesky avec un stockage creux compact

Nous allons voir que la structure de L est creuse mais plus dense que celle de A. Nous définissons la structure de la colonne j par

$$L_{*j} = \{i, j+1 \leq i \leq n, L(i,j) \neq 0\}.$$

Nous écrivons l'algorithme right-looking en utilisant cette structure.

L'opération cdiv ne se fait que pour les indices i tels que $L(i,j)$ est non nul, donc pour $i \in L_{*j}$.

ALGORITHM 5: opération cdiv(j) avec L creuse

$L(j,j) = \sqrt{L(j,j)}$
do $i \in L_{*j}$
  $L(i,j) = L(i,j)/L(j,j)$
enddo

Maintenant, l'opération cmod ne modifie pas la valeur de $L(i,k)$ si $L(k,j)$ est nul. Donc on ne fait l'opération cmod(j,k) que pour $k \in L_{*j}$. Aussi, l'opération cmod ne modifie pas la valeur de $L(i,k)$ si $L(i,j)$ est nul. Donc on ne fait l'opération cmod(j,k) que pour $i \in L_{*j}$, $k \leq i \leq n$.

Réciproquement, si $L(k,j) \neq 0$ et $L(i,j) \neq 0$, alors a priori, $L(i,k) \neq 0$. Donc

$$k \in L_{*j} \text{ et } i \in L_{*j} \text{ et } k \leq i \leq n \Rightarrow i \in L_{*k}.$$

ALGORITHM 6: opération cmod(j,k) avec $k \in L_{*j}$ et L creuse

do $i \in L_{*j}$, $k \leq i \leq n$
  $L(i,k) = L(i,k) - L(k,j) * L(i,j)$
enddo

ALGORITHM 7: Cholesky right-looking sur matrice creuse

do $j = 1, n$
  $cdiv(j)$
  do $k \in L_{*j}$
    $cmod(j, k)$
  enddo
enddo

On réduit ainsi fortement le nombre d'opérations. La complexité est liée au nombre d'éléments non nuls dans L, noté $nz(L)$.

3.3 Remplissage et factorisation symbolique

La structure creuse de L n'est pas celle de A, car des éléments nuls dans A deviennent non nuls dans L. En effet, dans l'opération $cmod(j, k)$, le terme $L(i, k)$ peut devenir non nul si le produit $L(k, j) * L(i, j)$ est non nul. Le remplissage est contaminant: un terme $L(i, k)$ qui devient non nul peut induire un nouveau terme non nul dans une colonne $l > k$.

En analysant à chaque étape le remplissage, on peut ainsi prédire la structure creuse de L. On ignore les éliminations numériques possibles et on base l'analyse sur les deux constats suivants:

- si le terme $L(i, k)$ est non nul, il reste non nul;
- si les deux termes $L(k, j)$ et $L(i, j)$ sont non nuls, alors $L(i, k)$ est non nul.

On réalise cette analyse avant de faire les calculs, de façon à prédire les structures L_{*j} par colonnes. On utilise le graphe de la matrice et la notion de graphe d'élimination.

Definition 3.1 *Soit $A \in \mathbb{R}^{n \times n}$ une matrice symétrique creuse. On définit alors son graphe par :*

- *un ensemble de sommets : $X = \{1, \cdots, n\}$,*
- *un ensemble d'arcs : $G = \{(i, j) \in X^2 \mid i > j \text{ et } \alpha_{ij} \neq 0\}$*

Le degré d'un sommet $x \in X$ est égal au nombre de ses voisins dans le graphe, c'est-à-dire au nombre de sommets $y \in X$ tels que $(y, x) \in G$. Cela représente le nombre de coefficients non nuls dans la colonne d'indice x sous la diagonale de la matrice A.

Chaque étape de l'algorithme de Cholesky correspond à un processus d'élimination des inconnues. La figure 1 montre l'influence de cette élimination sur les inconnues suivantes dont les indices $i1$, $i2$, $i3$ correspondent aux indices de ligne des

coefficients non nuls de la colonne j : cela crée les trois éléments indiqués sur la matrice par un gros point noir.

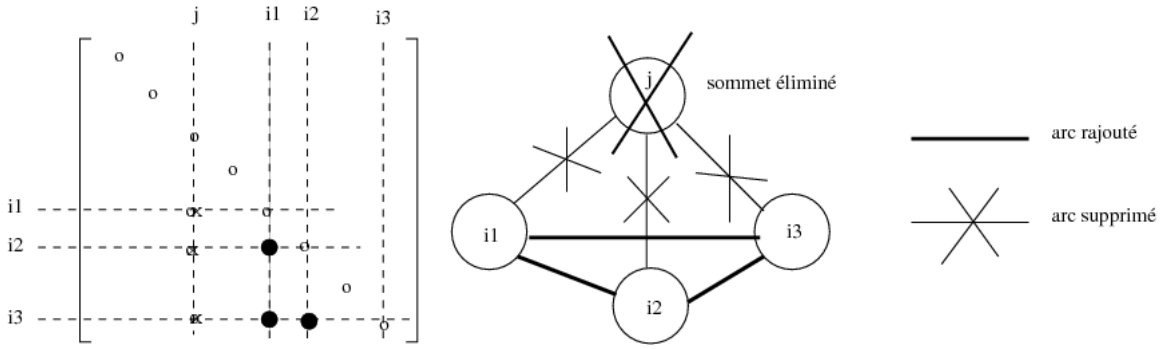

Figure 1: Remplissage après élimination d'un nœud.

Sur le graphe, cela correspond à supprimer les arcs entre j et ses voisins et à créer tous les arcs qui n'existent pas encore entre les voisins. On constate donc que dès que les voisins d'un sommet sont nombreux, son élimination entraîne un grand remplissage de la partie restante de la matrice.

Le processus complet conduit à la construction d'un arbre d'élimination. Avant d'effectuer les calculs, une phase préliminaire, basée sur cet arbre d'élimination, définit la structure creuse de L et prépare les tableaux pour le stockage compact.

3.4 Renumérotation et minimisation du remplissage

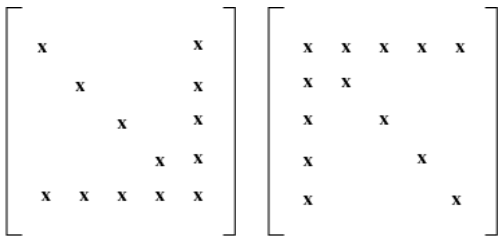

Figure 2: Matrices flèches: le remplissage dans L dépend de la numérotation.

Considérons par exemple la matrice de droite dans la figure 2. Le facteur L est dans ce cas une matrice pleine. Par contre, pour la matrice A de gauche, la structure de L est celle de A. Entre ces deux cas extrêmes, il peut y avoir plus ou moins de remplissage.

Ces exemples montrent que l'ordre d'élimination modifie fortement le niveau de remplissage. Des techniques de renumérotation, où on permute les lignes et les colonnes pour garder la symétrie, visent à minimiser le remplissage. Soit P la matrice de permutation associée. On effectue alors la factorisation de Cholesky

sur la matrice P^TAP, qui est toujours symétrique définie positive. Puis on résout le système $P^TAP(P^{-1}x) = P^Tb$.

3.5 Cholesky avec un stockage bande ou profil

Le facteur de Cholesky d'une matrice bande est une matrice bande de même largeur de bande.

La complexité passe de $O(n^3)$ pour l'opération LAPACK en mode plein à $O(nl^2)$ en mode bande, où l est la demi-largeur de bande.

De même, le facteur de Cholesky d'une matrice profil a le même profil.

Les algorithmes de renumérotation visent donc à minimiser la largeur de bande ou le profil de la matrice.

3.6 Algorithme du degré minimal

Un algorithme efficace et beaucoup utilisé est l'algorithme du degré minimal. Le principe en est de construire par récurrence une suite de graphes selon la procédure suivante :

1. $(X_0, G_0) = (X, G)$
2. Construction de (X_{i+1}, G_{i+1}) à partir de (X_i, G_i) par :
   (a) trouver $x \in X_i$ de degré minimal dans G_i (solution non nécessairement unique)
   (b) permuter x et $i+1$;
   (c) construire G_{i+1}) à partir de G_i en éliminant $i+1$.

3.7 Algorithme des dissections emboîtées

Il existe un autre algorithme qui a aussi de bons effets sur le remplissage : la dissection emboîtée. De plus, il a l'avantage d'introduire du parallélisme dans la factorisation et dans les résolutions des systèmes triangulaires qui en découlent.

L'algorithme est du type *diviser pour régner*. On suppose que la matrice A est irréductible, c'est-à-dire qu'il n'existe pas de permutation des indices qui décompose la matrice en deux ou plusieurs blocs diagonaux, ou encore dit autrement, que le graphe de la matrice A est connexe (il existe toujours un chemin entre deux sommets différents). Si A n'est pas irréductible, on applique l'algorithme à chacune des sous-matrices irréductibles qui la composent.

Le principe de l'algorithme est de partitionner récursivement le graphe. A chaque graphe (X, G) que l'on considère (le graphe entier à la première étape, des sous-graphes ensuite), on recherche une partie X_1 de X, appelée séparateur, telle que si on la supprime de (X, G) avec les arcs correspondants, on aboutisse à un graphe à deux composantes connexes et donc à deux sous-graphes indépendants.

En numérotant les sommets des deux sous-graphes avant ceux du séparateur, on obtient une matrice de type flèche (par blocs) du type

$$\begin{pmatrix} A_1 & & C_1 \\ & A_2 & C_2 \\ {C_1}^t & {C_2}^t & B \end{pmatrix}.$$

Les étapes suivantes consistent à donner la même structure aux matrices A_1 et A_2 qui sont aussi des matrices symétriques définies positives si A l'est. Nous avons vu dans l'exemple de la figure 2 l'intérêt de telles structures.

3.8 Factorisation numérique

Comme dans le cas plein, il existe 6 variantes selon l'ordre des 3 niveaux de boucle. Les 2 versions qui mixtent les accès lignes et colonnes sont peu utilisées. Les 2 versions par lignes sont bien adaptées au stockage profil. La plupart des logiciels de Cholesky utilisent un stockage compact par colonnes et une des deux versions par colonnes, left-looking ou right-looking.

Ces deux versions, avec les opérations *cdiv* et *cmod*, n'utilisent que du BLAS1. Pour introduire des opérations BLAS3, il faut définir des blocs, comme dans le cas plein. Il existe deux approches, dites supernodale et multifrontale.

La complexité est difficile à calculer dans le cas général. Dans le cas d'une matrice issue d'une discrétisation d'un problème de diffusion dans un carré ou un cube, on peut prévoir le terme dominant. Dans ce cas, l'algorithme des dissections emboîtées est optimal et le terme dominant est le coût de la factorisation du premier séparateur. Ce séparateur est plein, donc le coût est celui de Cholesky pour matrice pleine. En 2D, le séparateur est une ligne de taille $n^{1/2}$, donc le coût est en $O(n^{3/2})$. En 3D, le séparateur est un carré de taille $n^{2/3}$, donc le coût est en $O(n^3)$.

3.9 Résolution

La résolution effectue une descente-remontée avec la matrice triangulaire inférieure L issue de la factorisation $P^T AP = LL^T$. Il faut aussi effectuer les permutations sur le second membre et sur la solution.

3.10 Synthèse des étapes

Pour résoudre un système linéaire creux symétrique défini positif, on applique ainsi les étapes décrites ci-dessous:

- Minimisation du remplissage par renumérotation avec la matrice de permutation P: $A' = P^T AP$ et $b' = P^T b$,
- factorisation symbolique de Cholesky: structure $L_{*j}, j = 1, n$,
- Factorisation numérique de Cholesky: $A = LL^T$,

- Résolution du système triangulaire: $Ly = b'$ (descente),
- Résolution du système triangulaire: $L^T x' = y$ (remontée),
- Renumérotation: $x = Px'$.

4 Factorisation LU

Comme pour la factorisation de Cholesky, la plupart des logiciels utilisent un stockage compact par colonnes et une version par colonnes. Là aussi, le calcul se décompose en quatre étapes : renumérotation, factorisation symbolique, factorisation numérique, résolution.

Mais dans le cas non symétrique, il est nécessaire d'appliquer une stratégie de pivot, avec une permutation des colonnes ou des lignes pour garantir la stabilité numérique de la factorisation LU. En général, la stratégie est dynamique, c'est-à-dire est appliquée en cours de calcul. La permutation risque alors de générer du remplissage. Le critère de choix du pivot est basé à la fois sur l'algorithme de degré minimal, pour minimiser le remplissage, et sur le pivot partiel, pour la stabilité numérique. Une autre possibilité est d'appliquer une stratégie de pivot statique, a priori, avant la factorisation numérique. Il faut alors souvent effectuer une correction itérative du résidu pour améliorer la précision.

Comme pour Cholesky, l'introduction de blocs et d'opérations BLAS3 se fait par une technique supernodale ou multifrontale.

5 Exemples de logiciels

Il existe de nombreux logiciels pour les renumérotations et les factorisations de Cholesy et de Gauss. Certains sont séquentiels, d'autres ont des versions parallèles. Une comparaison de ces solveurs est difficile, car les résultats varient fortement d'une matrice creuse à une autre. Nous citons ici quelques logiciels libres, qui sont fréquemment utilisés.

- La suite logicielle SuiteSparse inclut des solveurs creux pour matrices symétriques définies positives (CHOLMOD) ou matrices non symétriques (UMFPACK). Le logiciel utilise une technique multifrontale et un pivot dynamique. Il est séquentiel et intégré à Matlab.
- Le solveur MUMPS est aussi une version multifrontale avec pivot dynamique, qui peut traiter des matrices symétriques définies positives, non symétriques, symétriques indéfinies. Il existe une version séquentielle et une version parallèle. La renumérotation peut se faire soit avec le degré minimum, soit avec les dissections emboîtées, par le biais par exemple du logiciel METIS ou PARMETIS en parallèle.
- Le solveur SuperLU traite les matrices non symétriques, avec technique supernodale et pivot statique. La version SuperLU-DIST est parallèle. Il utilise aussi METIS.

- Le solveur PSPASES effectue Cholesky, avec technique multifrontale ; il est parallèle, avec un nombre de processus qui est une puissance de 2, valant au moins 2. Il utilise PARMETIS.