Solveur linéaire par la méthode multifrontale MULT_FRONT

Résumé :

La méthode multifrontale MULT_FRONT est une méthode directe particulièrement adaptée à la résolution des systèmes linéaires dont la matrice est creuse. Cette méthode comprend une phase préliminaire de renumérotation destinée à minimiser le remplissage de la matrice au cours de la factorisation.

Cette phase permet aussi de regrouper les variables en "super-variables" ou "super-nœuds". La factorisation, quant à elle, est effectuée sous la forme d'une suite d'élimination de super-nœuds, dans des matrices pleines. Ces matrices pleines permettent l'utilisation de routines optimisées comme les BLAS, qui obtiennent les meilleures performances sur machines vectorielles ou scalaires.

Disposant de la matrice factorisée, chaque résolution de système ne nécessitera plus qu'une "descente/remontée" peu coûteuse.

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

Table des matières

Table des Matières

1 Description de la méthode....3
1.1 Méthode LDLT pour les matrices pleines....3
1.2 Matrice creuse et remplissage....6
1.3 Méthode multifrontale....7
1.4 Descente - Remontée....13
2 Implantation et utilisation dans Code_Aster....14
3 Bibliographie....15
4 Description des versions du document....15
Annexe 1 Documentation de référence de la méthode « Approximate Minimum degree »....17
Annexe 2 Documentation de référence METIS....17

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

1 Description de la méthode

La méthode multifrontale est une méthode **directe** de résolution de systèmes linéaires, qui est particulièrement adaptée aux systèmes ayant une matrice creuse. Elle factorise des matrices symétriques ou non, non nécessairement définies positives. Dans le cas le plus général, la méthode multifrontale fait appel à la méthode de Gauss avec recherche de pivots.

La méthode implantée dans *Code_Aster* factorise des matrices réelles ou complexes, symétriques ou non (une matrice non symétrique est acceptée dans la mesure où sa structure reste symétrique), et utilise l'algorithme dit LDL^T ou $L.U$, sans recherche de pivot.

La résolution d'un système linéaire s'effectue en trois étapes :

- renumérotation des inconnues,
- factorisation de la matrice,
- descente/remontée.

Dans le cas où plusieurs systèmes linéaires, de même matrice, sont à résoudre, seules les descentes/remontées sont à effectuer. De même si plusieurs matrices de même structure sont à factoriser, la renumérotation des inconnues ne sera pas à refaire. Cette phase préliminaire, qui assurera la performance de la factorisation, a un coût non négligeable, cependant son poids relatif (en temps de calcul), diminue avec la taille des matrices à factoriser.

Nous suivrons le plan suivant :

[1] présentation de la méthode LDL^T classique adaptée aux matrices symétriques pleines. Notion d'élimination,
[2] extension de la méthode aux matrices creuses, notion de remplissage,
[3] présentation de la méthode multi-frontale.

1.1 Méthode LDLT pour les matrices pleines

Soit $\mathbf{A}$, une matrice inversible, on sait qu'il existe une matrice triangulaire inférieure $\mathbf{L}$ à diagonale unité et une matrice triangulaire supérieure $\mathbf{U}$, telles que $\mathbf{A}=\mathbf{L}\cdot\mathbf{U}$.
L'ordre de ces matrices sera n .

Si $\mathbf{A}$ est symétrique, cette décomposition peut s'écrire :

$$\mathbf{A}=\mathbf{L}\mathbf{D}\mathbf{L}^{\mathbf{T}} \qquad \text{éq 1.1-1}$$

où $\mathbf{A}$ est une matrice diagonale et $\mathbf{L}^T$ la matrice transposée de $\mathbf{L}$, à diagonale unité.

Soient $(i,j)\in[1,n]^2$; de [éq 1.1-1] on déduit l'expression des coefficients de $\mathbf{L}$ et $\mathbf{D}$. En effet on peut écrire :

$$\mathbf{A}_{ij}=\sum_{k=1}^{n}\sum_{l=1}^{n}\mathbf{L}_{ik}\mathbf{D}_{kl}\mathbf{L}_{lj}^{T} \qquad \text{éq 1.1-2}$$

$\mathbf{L}$ étant triangulaire inférieure à diagonale unité, et $\mathbf{D}$ diagonale, [éq 1.1-2] devient :

$$\mathbf{A}_{ij}=\sum_{k=1}^{\min(i,j)}\mathbf{L}_{ik}\mathbf{L}_{jk}\mathbf{D}_{kk} \qquad \text{éq 1.1-3}$$

Il y a de nombreuses façons de calculer les coefficients de $\mathbf{L}$ et $\mathbf{D}$ à partir de [éq 1.1-3]. Nous allons voir la méthode dite "par ligne" utilisée habituellement dans le *Code_Aster*, et la méthode "par colonne" utilisée dans la méthode multifrontale.

Méthode par ligne

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

Les lignes de $\mathbf{L}$ et $\mathbf{D}$ sont calculées conjointement les unes après les autres. Supposons ces lignes connues jusqu'à l'ordre $(i-1)$, et supposons aussi que les coefficients de la ligne i sont connus jusqu'à l'ordre $j-1$; [éq 1.1-3] s'écrit, avec $j<i$:

$$\mathbf{A}_{ij}=\sum_{k=1}^{j-1}\mathbf{L}_{ik}\mathbf{L}_{jk}\mathbf{D}_{kk}+\mathbf{L}_{ij}\mathbf{D}_{jj}\mathbf{L}_{jj} \qquad \text{éq 1.1-4}$$

On a ainsi :

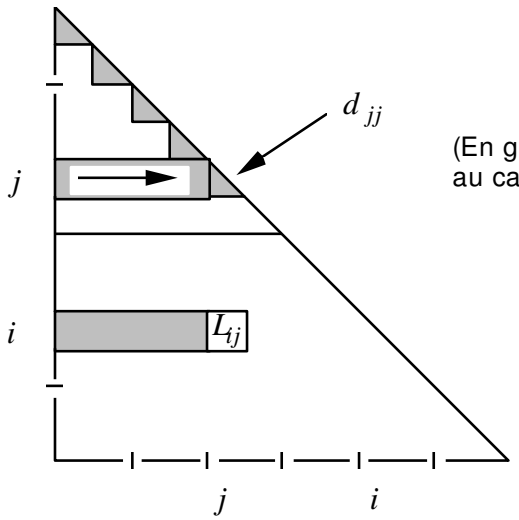
$$\mathbf{L}_{ij}=\left(\mathbf{A}_{ij}-\sum_{k=1}^{j-1}\mathbf{L}_{ik}\mathbf{L}_{jk}\mathbf{D}_{kk}\right)\Big/\mathbf{D}_{jj} \qquad \text{éq 1.1-5}$$

et de façon similaire :

$$\mathbf{D}_{ii}=\mathbf{A}_{ii}-\sum_{k=1}^{j-1}\mathbf{L}_{ik}\mathbf{L}_{ik}\mathbf{D}_{kk} \qquad \text{éq 1.1-6}$$

Avec cette méthode, chaque coefficient est calculé en une **seule fois** (opération [éq 1.1-5]), en allant chercher les coefficients précédemment calculés, et en faisant le produit scalaire de $\left(\mathbf{L}_{jk},k=1,j-1\right)$ et $\left(\mathbf{T}_{k},k=1,j-1\right)$, avec $T_k=L_{ik}.D_{kk}$

Cet algorithme est illustré par [Figure 1.1-a].



(En gris et hachuré, les termes nécessaires au calcul de L_{ij})

Figure 1.1-a

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

Méthode par colonnes

Examinons la figure [Figure 1.1-b] suivante :

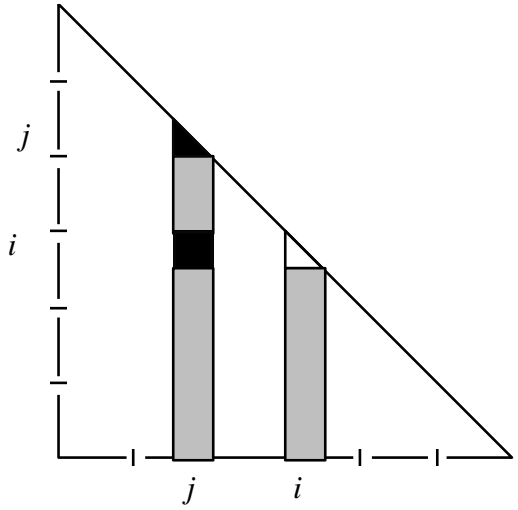


Figure 1.1-b

On verra ceci en détails plus loin (Cf. Figure 1.3-g).
Supposons que la colonne j, c'est-à-dire les termes $\mathbf{D}_{jj}$ et $\left(\mathbf{L}_{ij}, i=j+1,n\right)$, soit connus et effectuons l'algorithme suivant :

$$\left|\begin{array}{ll} \text{pour } i=j+1 \text{ à } n \text{, faire} & \textbf{éq 1.1-7} \\ \quad \mathbf{D}_{\text{ii}}=\mathbf{D}_{ii}-\mathbf{L}_{ij}^{2}\mathbf{D}_{jj} & \\ \quad \left|\begin{array}{ll} \text{pour } k=i+1 \text{ à } n \text{, faire} & \textbf{éq 1.1-8} \\ \quad \mathbf{L}_{ki}=\mathbf{L}_{ki}-\mathbf{L}_{kj}\mathbf{L}_{ij}\mathbf{L}_{jj} \quad \text{(saxpy)} & \end{array}\right. & \end{array}\right.$$

Faisons trois remarques :

1) les opérations [éq 1.1-7] s'appellent **l'élimination** de l'inconnue j. En effet, après [éq 1.1-7], on **ne fera plus jamais** appel aux termes de la colonne j dans la suite de l'algorithme. La méthode par colonne est parfois qualifiée de "looking forward method" ; dès qu'ils sont calculés, les termes de la matrice agissent sur les termes suivants. En revanche, les méthodes par ligne sont appelées "looking backward methods" ; on va chercher les termes précédemment calculés à chaque nouveau calcul,
2) l'opération [éq 1.1-7] est une opération de type "saxpy", on soustrait au vecteur $\left(\mathbf{L}_{ki}, k=i+1,n\right)$, le produit de la constante $\mathbf{L}_{ij}\cdot\mathbf{D}_{jj}$ et du vecteur $\left(\mathbf{L}_{kj}, k=i+1,n\right)$,
3) ayant effectué [éq 1.1-7] pour j fixé, voyons ce qu'il reste à faire pour connaître la colonne $(j+1)$,
   - $\mathbf{D}_{\text{j+1,j+1}}$ est connu (On le vérifie facilement),
   - il suffit de diviser la colonne $\left(\mathbf{L}_{k,j+1}, k=j+2,n\right)$ par $\mathbf{D}_{\text{j+1,j+1}}$ pour avoir la valeur définitive de celle-ci.

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

(On a abusivement confondu ci-dessus les termes $\mathbf{L}_{ki}$ définitifs et leur nom de programmation qui contient les valeurs de $\mathbf{L}_{ki}$ modifiées au cours des éliminations).

On en déduit donc l'algorithme général de factorisation LDL^T par colonnes.

pour $j=1$ à n, **faire**

pour $k=i+1$ à n, faire /normalisation/
$\mathbf{L}_{kj}=\mathbf{L}_{kj}/\mathbf{D}_{jj}$

éq 1.1-9

pour $i=j+1$ à n, **faire** /élimination/
$\mathbf{D}_{ii}=\mathbf{D}_{ii}-\mathbf{L}_{ij}^2\mathbf{D}_{jj}$

pour $k=i+1$ à n, **faire**
$\mathbf{L}_{ki}=\mathbf{L}_{ki}-\mathbf{L}_{kj}\mathbf{L}_{ij}\mathbf{L}_{jj}$ /saxpy/

Avant de passer à la notion de remplissage, il convient de faire dès maintenant une remarque utile pour la suite. Si l'on regarde [éq 1.1-9], il apparaît que l'on peut éliminer l'inconnue j, même si les termes $\left(\mathbf{L}_{ki}, k=i+1,n\right)$ et $\mathbf{D}_{ii}$ ne sont pas encore disponibles. En effet, il suffit de conserver les termes $\left(-\mathbf{L}_{kj}\mathbf{L}_{ij}\mathbf{D}_{jj}\right)$, et de les ajouter **ensuite** aux termes $\mathbf{L}_{ki}$. Ces termes $\left(\mathbf{L}_{kj}\mathbf{L}_{ij}\mathbf{D}_{jj}\right)$, i variant de $j+1$ à n et k de $i+1$ à n forment une matrice associée à l'élimination de l'inconnue j, que l'on appellera par la suite la **matrice frontale** j.

1.2 Matrice creuse et remplissage

On rappelle que si la matrice initiale comporte des termes nuls, les éliminations successives provoquent du remplissage, c'est-à-dire que certains termes $\mathbf{L}_{ki}$ sont différents de zéro alors que le terme initial $\mathbf{A}_{ki}$ l'est.

Supposons qu'avant l'élimination de l'inconnue j, le terme $\mathbf{L}_{ki}$ est nul ; si $\mathbf{L}_{kj}$ et $\mathbf{L}_{ij}$ sont **tous deux** non nuls, [éq 1.1-9] montre que $\mathbf{L}_{ki}$ deviendra lui aussi non nul. Ce remplissage a une interprétation graphique. Supposons dans ce cas que toutes les inconnues sont représentées par les nœuds d'un graphe : on reliera les nœuds k et i si et seulement si le terme $\mathbf{A}_{ki}$ de la matrice initial est non nul. Si $\mathbf{A}_{ki}$ est nul avec i relié à j et k relié à j, l'élimination du point de vue graphique de j consistera à relier alors i et k.

La figure [Figure 1.2-a] illustre cette interprétation. Les arêtes en pointillé sont celles crées par l'élimination de j. Elles correspondent à de nouveaux termes non nuls de la matrice L.

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

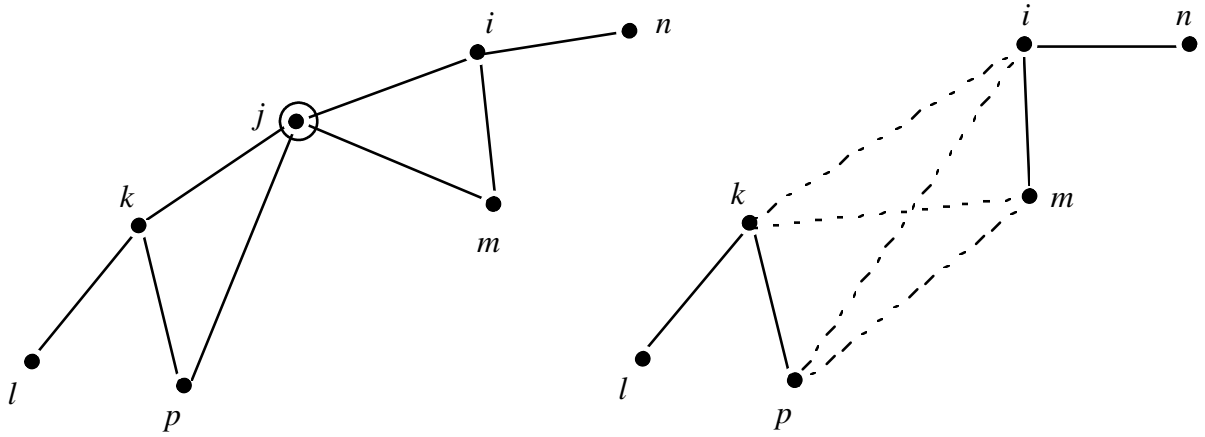


Figure 1.2-a : L'élimination du nœud j relie tous ses voisins entre eux

1.3 Méthode multifrontale

La méthode multifrontale est une méthode directe, de type Gauss, qui vise à exploiter au maximum le creux de la matrice à factoriser. Elle cherche, d'une part, à minimiser le remplissage en utilisant une renumérotation optimale, d'autre part, elle extrait de la structure creuse de la matrice l'information lui permettant **d'éliminer** (cf. page 5 remarque (1)) des inconnues indépendamment les unes des autres.

Examinons le cas simple de la figure [Figure 1.3-a], où la matrice a un seul terme nul, $\mathbf{A}_{21}$.

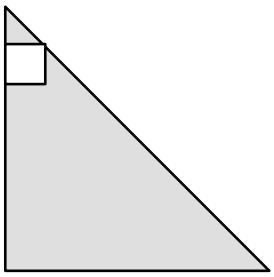


Figure 1.3-a

La colonne 2 de $\mathbf{L}$ ne subit pas les effets de l'élimination de l'inconnue 1, car le coefficient $\mathbf{A}_{21}=0$, puis $\mathbf{L}_{21}=0$ (cf. [éq 1.1-9] vue précédemment). Les contributions aux colonnes 3 et 4 de $\mathbf{L}$, des inconnues 1 et 2 sont indépendantes de leur ordre d'élimination (il faut regarder en détail [éq 1.1-9]). De cette constatation, on peut introduire la notion d'arbre d'élimination présentée par I. Duff [bib1].

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

L'arbre d'élimination de la matrice $\mathbf{A}$ peut être représenté par la figure [Figure 1.3-b].

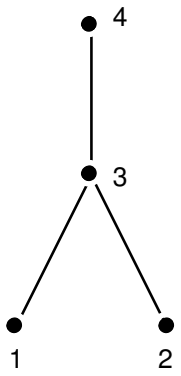


Figure 1.3-b

Cette arborescence contient deux notions :

- l'indépendance de certaines éliminations (ici les variables 1 et 2), qui conduira à un parallélisme possible des opérations,
- la minimisation des opérations à effectuer, (on voit sur l'arborescence que le terme $\mathbf{L}_{21}$ n'est pas à calculer).

Etant donné une matrice creuse, **dont on connaît le remplissage**, l'arbre d'élimination peut être construit comme suit :

- toutes les feuilles de l'arbre (extrémités inférieures) correspondent aux inconnues j, telles que $\mathbf{A}_{ji} = 0$ pour $i = 1$ à $j - 1$. Ici 1 est bien sur une feuille car il n'y a pas de terme $\mathbf{A}_{1i}$ pour $i < 1$, 2 est aussi une feuille car $\mathbf{A}_{21} = 0$,
- un nœud j a pour père i, si i est le plus petit numéro de ligne telle que $\mathbf{A}_{ij} \neq 0$. Ici, 3 est le père de 1 et de 2.

Remarques :

1) *on emploie à partir de maintenant le vocabulaire de la théorie des graphes, arbre, feuille, nœud... Ici l'arbre est retourné, ses feuilles sont en bas,*
2) *on se reportera à [bib1] pour plus de détails et les démonstrations de la validité de la méthode,*
3) *dans l'exemple ci-dessus, l'ordre d'élimination entre les inconnues (3) et (4) est fixé par la numérotation initiale, on aurait pu permuter les lignes et les colonnes de la matrice et avoir 4 comme père de 1 et 2,*
4) *il faut remarquer que la fabrication de cette arborescence doit prendre en compte les **termes non nuls obtenus par remplissage** au cours de l'élimination. (On verra plus de détails sur ce sujet dans [bib2]). On ne peut fabriquer l'arbre d'élimination uniquement à partir de la matrice creuse initiale : il faut connaître aussi les termes de remplissage comme déjà mentionné précédemment. La factorisation numérique de la méthode multifrontale est précédée d'une phase importante : la simulation des éliminations et ainsi, de la création des termes non nuls. On appelle aussi cette simulation, l'élimination logique ou symbolique. Cette simulation a lieu au cours de la première phase : la renumérotation. On va voir les quatre phases de la méthode multifrontale, les plus importantes étant la première et la quatrième.*

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

Première phase : La renumérotation du "minimum degree" (degré minimum).

Cette renumérotation a pour **premier but** de minimiser le nombre d'opérations à effectuer lors de la factorisation. Pour cela on simule l'élimination des nœuds et on choisit comme candidat à l'élimination le nœud du graphe ayant le plus faible nombre de voisins. On utilise ici la notion de graphe vue à la figure [Figure 1.2-a]. La matrice creuse initiale définit le graphe sur lequel on travaille. Ce dernier est ensuite remis à jour à chaque élimination de nœuds (création de liens). La simulation du remplissage permet d'atteindre le **second but** qui est la fabrication de l'arbre d'élimination vue au paragraphe précédent. Le **troisième but** atteint, est la création des **super-nœuds**. C'est une notion importante que nous allons développer.

Un super-nœud (SN) est formé de l'ensemble des nœuds qui, au cours de l'élimination, ont les mêmes voisins au sens du graphe d'élimination. Au cours de la simulation de l'élimination, on détecte que par exemple, les nœuds i , j et k sont :

- d'un part, voisins entre eux (les termes $\mathbf{L}_{ij}$, $\mathbf{L}_{jk}$, $\mathbf{L}_{ik}$ ... sont non nuls),
- d'autre part, ils ont pour voisins communs les nœuds : $\left\{ l, m, p, q, r, s, t \right\}$ (voir [Figure 1.3-c]).

Ils forment alors le super-nœud $\left\{ i, j, k \right\}$ et seront éliminés tous ensemble lors de la factorisation numérique.

La **MM-F** est une méthode de factorisation par blocs, lorsqu'elle utilise la notion de "super-nœud".

Cette notion a le double avantage suivant :

- elle diminue le coût (non négligeable) de la renumérotation,
- elle diminue le coût de la factorisation en regroupant les calculs (utilisation de routines d'algèbre linéaire de type `blas-2` ou `blas-3`).

On voit sur [Figure 1.3-c] la structure des colonnes $\left\{ i, j, k \right\}$ (termes non nuls en grisé) dans une matrice globale creuse **virtuelle**. En fait, une telle matrice creuse **n'est jamais assemblée**. On travaille dans des matrices locales **pleines** que l'on appelle les **matrices frontales**. On a une matrice frontale par SN (On reverra cette notion au paragraphe "Factorisation numérique").

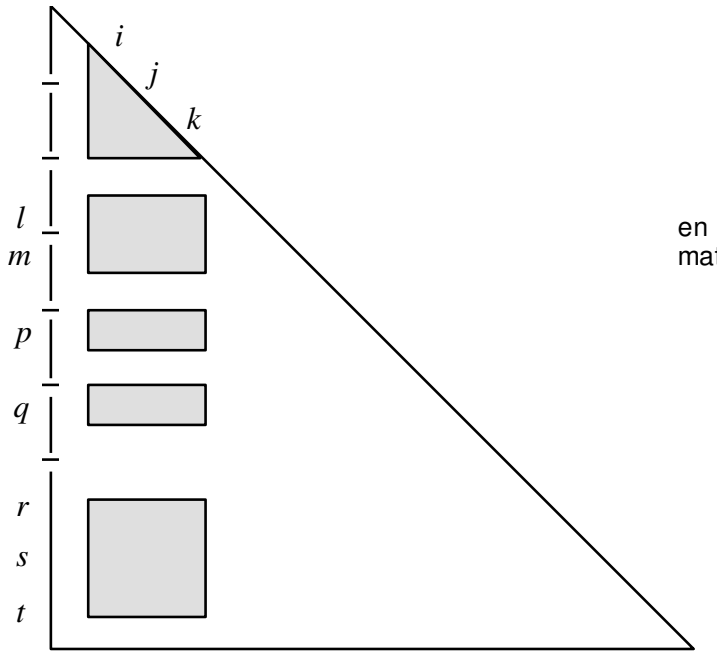


Figure 1.3-c

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

En résumé, la première phase de la méthode multifrontale (la renumérotation par le "minimum degree") consiste en les trois actions suivantes :

- une renumérotation des inconnues à éliminer, afin de minimiser le remplissage. On simule les éliminations en mettant à jour à chaque fois les listes de voisins des nœuds,
- conjointement à l'action précédente, on construit l'arbre d'élimination, et,
- on détecte les "super-nœuds", que l'on peut qualifier de classe d'équivalence pour la relation "ont les mêmes voisins".

Remarque :

L'arbre d'élimination fourni par la renumérotation est exprimé en termes de super-nœuds, car l'élimination numérique à suivre se fera par super-nœuds.

Deuxième phase : La factorisation symbolique

Cette phase n'est pas aussi fondamentale que la précédente. C'est une phase technique destinée à construire certains pointeurs. Ce sont en particulier les tableaux d'indices globaux et locaux qui établissent les correspondances entre les inconnues lors de l'assemblage des matrices frontales.

Troisième phase : La séquence d'exécution

C'est aussi une phase technique. On a vu au paragraphe précédent que la méthode consistait à parcourir l'arbre d'élimination, en effectuant une élimination à chaque nœud de l'arbre. Le résultat de cette élimination est une **matrice frontale**. L'ordre de cette matrice est le nombre de voisins du SN éliminé. Le stockage des matrices frontales est coûteux en occupation de la mémoire.

La matrice frontale j (résultat de l'élimination du SNj) sera assemblée dans la matrice frontale du nœud i, où i est le **père de** j. On verra cette phase plus en détail lors de la factorisation numérique). Toutes les matrices frontales doivent être conservées, jusqu'à ce qu'elles soient utilisées lors de l'élimination du "SN père". On peut alors ranger la matrice du "SN père" à la place de celles des "SN fils".

Il y a plusieurs façons de parcourir l'arbre en respectant la contrainte : "le fils doit être éliminé avant le père". L'objet de la troisième phase est de trouver l'ordre de parcours qui minimise la place en mémoire nécessaire pour le rangement des matrices frontales ([bib2], page 2.12).

Quatrième phase : La factorisation numérique

Cette phase est la factorisation effective, c'est-à-dire le calcul des matrices $\mathbf{L}$ et $\mathbf{D}$. Par la suite, on confondra ces deux matrices et d'un point de vue informatique $\mathbf{D}$ sera vu comme les termes diagonaux de $\mathbf{L}$.

La factorisation numérique consiste à parcourir l'arbre d'élimination ; pour chaque « Super-nœud », on effectue :

- l'assemblage, dans la matrice frontale "mère", des matrices frontales "fille",
- l'élimination des colonnes du super-nœud.

Voyons l'exemple suivant avec le graphe et l'arbre d'élimination de [Figure 1.3-d] (les nœuds 8,9, 10 ne figurent pas sur le dessin).

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

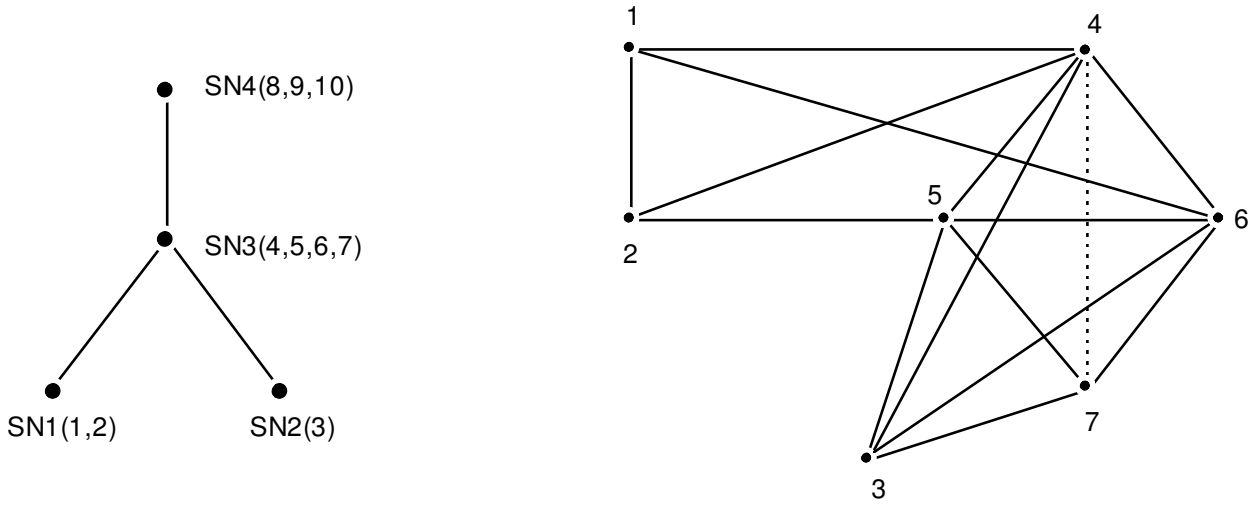


Figure 1.3-d : Exemple de graphe et d'arbre d'élimination

Entre parenthèses, on lit les numéros des inconnues du SN.

L'élimination du SN1 consiste en :

1) l'assemblage des colonnes 1 et 2 de la **matrice initiale**, dans une matrice de travail, dite matrice frontale avant élimination. Cette matrice est d'ordre 5, liée aux inconnues $(1,2,4,5,6)$. (Car 1 et 2 sont liés à $(4,5,6)$ uniquement),
2) l'élimination du SN1 (des colonnes 1 et 2, selon les formules [éq 1.1-7] et [éq 1.1-8] vues précédemment),
3) le rangement des deux colonnes 1 et 2 de la matrice, dans un tableau FACTOR, qui contient les colonnes de la matrice factorisée globale,
4) le rangement de la matrice frontale 1 d'ordre 3 liée aux inconnues $(4,5,6)$.

On fait la même chose avec le SN2.

Ces deux éliminations sont illustrées par la figure [Figure 1.3-e], où on voit en grisé avec hachure les deux matrices frontales.

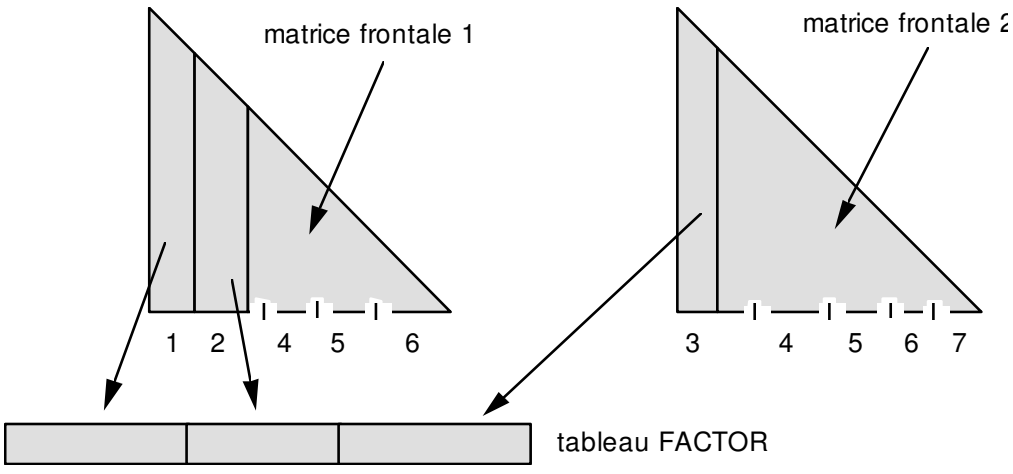


Figure 1.3-e

L'élimination du SN3 consiste en :

- l'ajout des colonnes $(4,5,6,7)$ de la matrice initiale, à la matrice de travail d'ordre 7, liée aux inconnues $(4,5,6,7,8,9,10)$,

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

- l'assemblage des matrices frontales 1 et 2 dans cette matrice de travail,
- l'élimination des colonnes $(4,5,6,7)$,
- l'obtention dans FACTOR de quatre colonnes supplémentaires,
- l'obtention de la matrice frontale 3 d'ordre 3 (colonnes $8,9,10$).

On remarque que la matrice frontale 3 peut se ranger à la place des matrices frontales 1 et 2 . Le rangement de ces matrices nécessite une structure de pile où l'on empile à la fin d'une élimination, et où l'on dépile pendant l'assemblage. C'est la longueur maximale de cette pile qui est minimisée lors de la phase de séquence d'exécution.

La figure [Figure 1.3-f] illustre l'élimination du SN3.

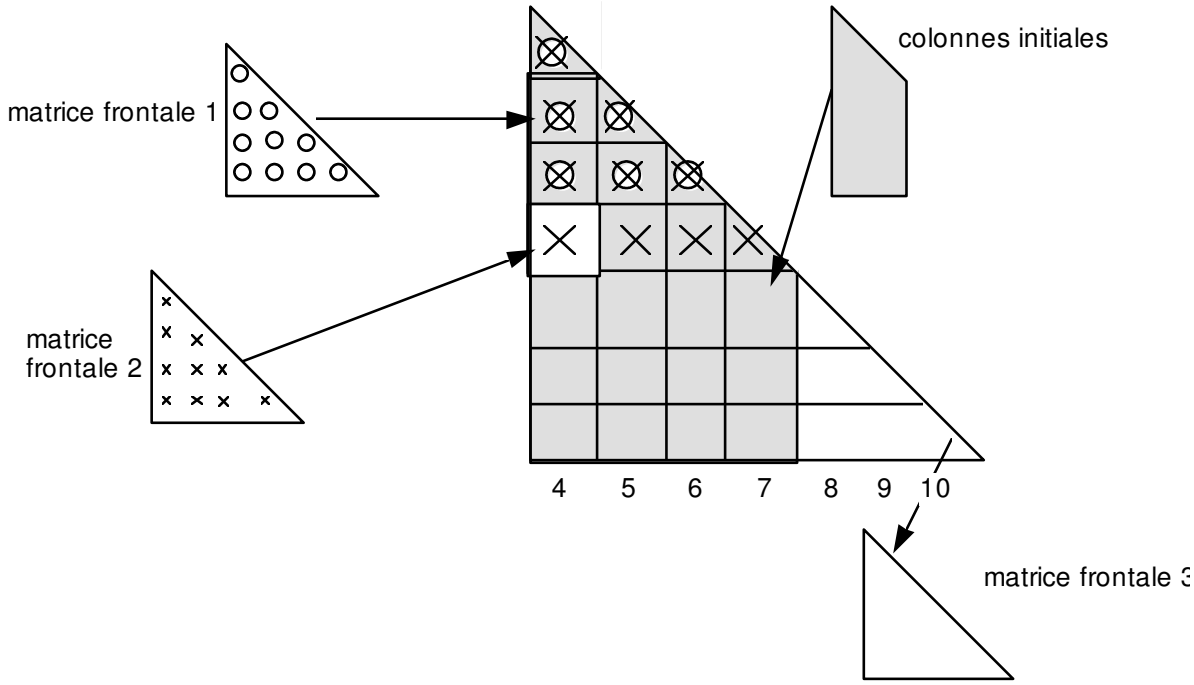


Figure 1.3-f : Élimination du SN3

On remarque que le coefficient $\mathbf{L}_{74}$ (carré blanc sur la figure [Figure 1.3-f]), provient de l'élimination du SN3, et que le terme initial $\mathbf{A}_{74}$ est nul (lien en pointillé sur la figure [Figure 1.3-d].

On a vu au paragraphe 1.1 que l'élimination d'une colonne consistait en une opération de type "saxpy", ajout à un vecteur du produit d'un vecteur par un scalaire.

On voit facilement que l'élimination d'un super-nœud, groupe de colonnes, consiste en une opération de type "produit matrice-vecteur". Ces opérations consomment la plus grande part en temps de calcul, du travail de factorisation de la matrice. Elles sont réalisées par des sous-programmes de la bibliothèque BLAS, fournies après avoir été optimisées sur la plupart des calculateurs. On voit sur la figure [Figure 1.3-g] la colonne j mise à jour par le produit de la matrice $\mathbf{A}[j+1:n,1:m]$ et du vecteur $\mathbf{A}[j,1:m]$.

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

La factorisation étant terminée, on dispose de la matrice factorisée sous la forme d'un tableau compacté des termes non nuls uniquement. C'est un stockage de type "morse". Le tableau d'indice **global** évoqué dans la factorisation symbolique indique, pour chaque colonne de $\mathbf{L}$, les numéros de ligne de chaque coefficient stocké.

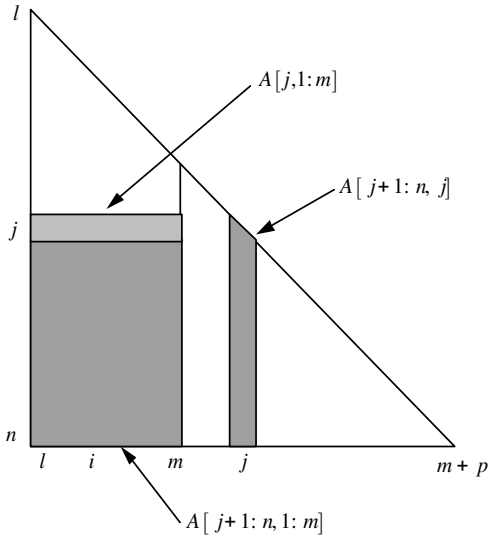


Figure 1.3-g : Mise à jour de la colonne j par un produit matrice*vecteur

1.4 Descente - Remontée

Les colonnes de $\mathbf{L}$ étant stockées de façon compactée, la descente est de type "saxpy" et la remontée de type produit scalaire, ces deux opérations étant toutes deux indexées. C'est-à-dire que l'algorithme de descente est grossièrement le suivant, (en ayant préalablement initialisé x par le second membre du système) :

$$\begin{array}{l}\textbf{pour } i=1 \textbf{ à } n \textbf{ faire}\\ \quad \textbf{pour } k\in \text{colonne } i \textbf{ faire}\\ \quad \mathbf{x}(global(k))=\mathbf{x}(global(k))-\mathbf{L}_{ki}\times\mathbf{x}(i) \qquad (\text{saxpy indexé})\\ \textbf{pour } i=1 \text{ à } n \textbf{ faire}\\ \quad \mathbf{x}(i)=\mathbf{x}(i)/\mathbf{D}_{ii}\end{array}$$

La remontée, elle, s'écrit sous la forme :

$$\begin{array}{l}\textbf{pour } i \textbf{ variant de } r \text{ à } 1 \textbf{ faire}\\ \quad x(i)=x(i)-\sum_{k\in colonne\, i} L_{\mathrm{ik}}\cdot x(global(k)) \quad : \quad \text{produit scalaire indexé}\end{array}$$

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

2 Implantation et utilisation dans *Code_Aster*

L'utilisation de la méthode multi-frontale est accessible par l'opérateur NUME_DDL, de la façon suivante :

```
nu = NUME_DDL (matr_rigi = matr, stockage = 'MORSE',
                                 renum = 'MD'
                              ou renum = 'MDA',
                              ou renum = 'METIS') ;
```

Deux autres méthodes de renumérotation sont disponibles : Minimum Degré approché (MDA), qui est une variante de la méthode du minimum degré, et une méthode de dissection emboîtée (METIS). Elles sont décrites brièvement en annexes 1 et 2. Il suffit de remplacer la valeur de renum par MDA, ou METIS.

Cette méthode est également directement disponible sous le mot-clé SOLVEUR dans MECA_STATIQUE, STAT_NON_LINE, DYNA_NON_LINE, THER_LINEAIRE, THER_NON_LINE avec la même logique.

Ensuite, on utilisera l'opérateur FACTORISER, l'appel étant le même que dans le cas d'une matrice stockée suivant le mode profil.

Dans NUME_DDL, on indique que la matrice à factoriser est rangée suivant le mode MORSE et que l'on demande une des deux renumérotations du "minimum degree", ou une renumérotation par dissection emboîtée (METIS).

Dans ce cas, NUME_DDL effectue les trois premières phases vues précédemment :

1) renumérotation,
2) factorisation symbolique,
3) séquence d'exécution.

Depuis un récent développement, la renumérotation s'effectue sur les nœuds (au sens de l'interpolation), et non plus sur les degrés de liberté. Cela permet un gain de temps de calcul et assure le respect de l'ordre initial des degrés de liberté. à l'intérieur des nœuds (cela est parfois nécessaire dans le cas incompressible où la pression doit être numérotée après les déplacements). La contrainte de numérotation des "double-lagrange" encadrant les degrés de liberté. affectés par la condition aux limites est aussi respectée.

En outre, NUME_DDL prépare le découpage par blocs de la matrice factorisée. En effet, on a vu qu'à chaque élimination, on rangeait dans un tableau les colonnes de la matrice factorisée. Ces colonnes ne serviront que lors de la descente/remontée. Elles ne servent jamais au calcul d'autres colonnes. Il n'y a donc aucun intérêt à les avoir toutes en mémoire. Elles sont rangées, dans le *Code_Aster*, sous la forme d'une collection d'objets JEVEUX de longueur variable. Ces objets, blocs de colonnes, doivent néanmoins satisfaire la contrainte suivante : chaque bloc correspond à un nombre entier de "supernœuds". On ne peut donc avoir les colonnes d'un même "supernœud" sur plusieurs blocs.

Étant donné que l'on ne connaît pas la place en mémoire disponible lors de la factorisation numérique, on décide dans NUME_DDL que la longueur maximum de chaque bloc sera le max. (sur tous les SN) de la somme des longueurs des colonnes du SN.

$$\mathbf{L}\, b_{max} = \underset{SNi}{Max}\left(\sum_{k \in SNi} longueur\left(col._k \right) \right) \quad , \quad \text{(en abrégé)}$$

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

L'opérateur FACTORISER utilise les pointeurs créés par NUME_DDL et la matrice initiale MORSE. Il crée la matrice factorisée sous la forme d'une collection d'objets JEVEUX, comme on l'a vu précédemment. Une structure de donnée **provisoire** est aussi nécessaire. Elle concerne les matrices frontales. Deux cas se présentent :

- la pile des matrices frontales peut tenir toute entière en mémoire (dans un tableau monodimensionnel), dans ce cas, on alloue un objet JEVEUX et l'algorithme multifrontal gère alors lui-même cet espace, en rangeant la matrice frontale "mère" à la place des matrices "filles",
- elle ne tient pas et, dans ce cas, elle est créée sous la forme d'une collection d'objets JEVEUX, chaque matrice frontale étant un objet. Pour l'élimination du "supernœud" i il faut simultanément en mémoire :
  - le bloc de la factorisée auquel appartient le SNi
  - l'objet matrice frontale i ainsi que toutes les matrices frontales "fille" de i qui seront **détruites** après leur utilisation.

La matrice frontale i sera, elle, conservée en mémoire jusqu'à son utilisation et sa destruction. On pourrait, comme il était fait auparavant, libérer au sens de JEVEUX chaque matrice frontale après sa création. Cela peut entraîner alors, en cas de faible mémoire disponible, un stockage sur disque rédhibitoire en volume. En effet, une destruction d'objet JEVEUX n'entraîne pas la destruction de son image sur disque, et la somme des longueurs des matrices frontales est énorme.

En résumé, il faut pouvoir ranger simultanément en mémoire :

- un bloc de la factorisée,
- la pile des matrices frontales, en un seul objet si cela tient, en plusieurs objets sinon.

Il faut donc **"suffisamment"** de mémoire pour utiliser la méthode multifrontale. La seconde manière de ranger la pile permet l'exécution quand la mémoire est suffisante, mais émiettée.

Quand la mémoire est insuffisante, il faut relancer l'exécution de FACTORISER avec une mémoire plus grande.

3 Bibliographie

1) I. DUFF : Parallel implementation of multifrontal scheme". Parallel computing 3, pp. 193-201 (1986)

2) C. ROSE : Une méthode multifrontale pour la résolution directe des systèmes linéaires. Note HI-76/93/008

3) An approximate minimum degree ordering algorithm, Tim DAVIS, Patrick AMESTOY, Iain DUFF, Technical Report TR 94-039, (SIAM J. Matrix Analysis & Applications, vol 17, no.4, (1996), pp. 886-905).

4) The evolution of the minimum degree ordering ordering, A. George, J. Liu, SIAM review, vol 31 1989. On peut aussi consulter les sites internet suivants : http ://www.cerfacs.fr - http ://ww.cise.ufl.edu/~davis

5) METIS : A software package for partitionning unstructured graphs,partitioning meshes, and computing fill oredering of sparse matrices Version 4.0. Georges Karypis &Vipin Kumar

6) A fast and high quality MultiLevel Scheme for partitioning irregular graphs. SIAM journal on Scientific Computing 1998, Vol 20 No 1. Georges Karypis & Vipin Kumar

4 Description des versions du document

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

Version Aster	Auteur(s) Organisme(s)	Description des modifications
5	C.ROSE-EDF/R&D/SINETICS	Texte initial
8,4	C.ROSE-EDF/R&D/SINETICS	

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

Annexe 1Documentation de référence de la méthode « Approximate Minimum degree »

L'algorithme du « minimum degree » consiste a numéroter les nœuds d'un graphe dans l'ordre croissant du nombre de leurs voisin. En voici une description simplifiée mais essentielle.

1) on forme le graphe initial associé à la matrice creuse à factoriser,
2) on calcule le nombre des voisins des nœuds (leur degré),
3) on numérote en premier (puis élimine du graphe), le nœud ayant le moins de voisins (degré minimum),
4) on remet à jour le graphe de l'étape 1,
5) on retourne à l'étape 2.

Le calcul du degré des nœuds est une opération coûteuse en temps de calcul. La méthode « Approximate Minimum degree » [bib3] se propose de réduire ce coût. Pour cela, au lieu de calculer le degré réel d'un nœud on se contente d'un majorant (souvent égal, d'après les auteurs, au vrai degré), de calcul plus facile. En effet au cours de l'élimination le vrai degré du nœud i est égal à d_i, cardinal de l'ensemble suivant : $A_i \cup \{L_e\}$, où l'union est faite sur l'ensemble des voisins e, précédemment éliminés, de i (termes de remplissage), et où A_i est l'ensemble des voisins initiaux. Dans la méthode approchée i est remplacé par $d_i = \mathrm{card}(A_i) + \sum \mathrm{card}\{L_e\}$, c'est-à-dire que l'on néglige l'intersection des L_e.
Cette méthode est décrite dans [bib3] et les différentes variantes de l'algorithme du « minimum degree » sont exposées dans [bib4].

Annexe 2Documentation de référence METIS

L'implémentation du module de renumérotation pour matrices creuses METIS est décrit dans la note [bib5], fournie dans le répertoire METIS-4.0/Doc du logiciel. L'algorithme utilisé est décrit dans [bib6]
Il est aussi possible de consulter l'adresse internet suivante : http ://www.cs.umn.edu/~karypis

METIS est principalement un outil de découpage de graphes (maillages), visant les 2 buts suivants :

- découper un graphe donné en p sous-graphes ayant des tailles les plus proches possibles,
- minimiser la taille des frontières entre les sous-graphes.
  Le premier but vise un bon équilibre de parallélisation sur p processeurs et le second vise à minimiser les communications.

METIS utilise un algorithme de partition de graphes à plusieurs niveaux, procédant de la façon suivante : à chaque niveau on cherche des séparateurs (ensemble de cotés), de tailles minimales coupant le graphe en parties égales. METIS applique ce principe à la renumérotation des nœuds d'un graphe en cherchant à chaque étape un séparateur divisant le graphe en 2 sous-graphes. On numérote en tête les nœuds du premier sous-graphe, ensuite ceux du second, puis ceux du séparateur. On applique ensuite le même algorithme aux 2 sous graphes de façon récursive.

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)