Université de Liège
Faculté des sciences appliquées

Méthodes itératives

pour la résolution de grands systèmes non linéaires

Travail de fin d'études présenté par

Romain BOMAN

en vue de l'obtention du grade d'ingénieur civil physicien

Année académique 1996-97

1. Introduction

1.1 Première description du problème

Ce travail se place dans le cadre de l'étude des grandes déformations en mécanique du solide. Pour obtenir des programmes de simulation puissants, la recherche dans ce domaine est divisée principalement en deux parties complémentaires. La première s'occupe de la mécanique du solide proprement dite, c'est-à-dire l'étude détaillée du comportement des matériaux, l'établissement des lois constitutives, la création et l'étude d'éléments finis appropriés, etc. La seconde permet de mettre en œuvre les concepts théoriques d'une manière efficace en particularisant les méthodes de l'analyse numérique aux problèmes rencontrés. Il s'agit, par exemple, d'optimiser un schéma d'intégration, de calculer efficacement des valeurs propres de matrices ou, comme ce travail, de résoudre d'une manière économique (en temps de calcul et en mémoire) un système linéarisé.
Ces deux aspects ne doivent jamais être dissociés. C'est pourquoi, nous analyserons soigneusement l'origine du système à résoudre pour en tirer un profit maximum.

Nous travaillerons directement dans le code de METAFOR [P4], un programme de simulation par éléments finis des grandes transformations des solides, développé par le service de thermomécanique des milieux continus de l'université de Liège.

A l'heure actuelle, les ordinateurs permettent la simulation des phénomènes physiques observés couramment en entreprise lors de la mise en forme des matériaux (l'emboutissage de tôles, ...) tant que ceux-ci sont de petite taille (peu de degrés de liberté). On entend par là des phénomènes qui peuvent être étudiés à deux dimensions (axisymétriques, état plan de contrainte ou de déformation). Pour ce type de problème, la méthode de Gauss, qui est utilisée dans METAFOR depuis sa naissance, est certainement la plus simple et la plus fiable. Cependant, si on veut s'approcher au maximum de la réalité, il faut, dans beaucoup de cas,

passer à un modèle à trois dimensions. Il suffit d'avoir en tête la forme de pièces mécaniques courantes pour s'en convaincre. A partir de ce moment, le nombre de degrés de liberté à prendre en compte devient très vite énorme. On atteint très facilement les dix mille, voire cent mille degrés de liberté. Les matrices résultantes sont beaucoup trop grosses pour être contenues entièrement en mémoire centrale. Une méthode couramment employée pour remédier à ce problème consiste à utiliser une mémoire périphérique (disques durs) pour stocker temporairement certaines données et ne garder en mémoire que le strict minimum. En ce qui concerne la résolution de systèmes d'équations, on peut utiliser, par exemple, les méthodes frontales [I1] qui permettent de travailler successivement avec des groupes d'équations dites 'actives', les autres étant laissées de côté. Bien qu'il soit possible de réduire la taille du stockage de la matrice du système par le format 'ligne de ciel' (SKYLINE) [P3], il est indispensable de garder en mémoire de nombreux zéros qui deviendront non nuls lors de la factorisation LU de la matrice.

Un autre problème surgit parallèlement au premier : les méthodes traditionnelles de résolution de systèmes sont basées sur l'élimination de Gauss. Celle-ci devient impraticable pour des grands systèmes parce qu'elle requiert un nombre d'opérations de l'ordre de N^2m (si N est la taille et m la largeur de bande de la matrice). Le calcul devient interminable, surtout si le système est stocké en partie sur disque.

Pour remédier à cela, de nouvelles méthodes de résolution sont apparues : les méthodes itératives. Elles ont l'avantage d'utiliser uniquement la matrice du système en tant d'opérateur linéaire, c'est-à-dire par son action sur un vecteur. Il est donc possible de réduire le stockage de la matrice à ses éléments non nuls. Le gain de mémoire par rapport au solveur direct est d'autant plus élevé que le système est grand et tridimensionnel, c'est-à-dire que la matrice possède une grande largeur de bande. Le revers de la médaille : ces solveurs sont extrêmement sensibles aux propriétés de la matrice du système et en particulier à son nombre de conditionnement.

Leur rapidité dépendra donc en grande partie du préconditionnement du système. Cela consiste à rechercher une approximation de l'inverse de la matrice qui rende le système préconditionné le plus proche possible de la matrice identité. Dans le cas extrême ou il y a égalité, les méthodes itératives fournissent la solution en une seule itération !

Le but de ce travail se résume donc comme ceci : il s'agit de résoudre d'une manière itérative le système linéaire provenant de la linéarisation des équations d'équilibre d'un corps en grandes déformations dans le cadre d'un algorithme de Newton-Raphson. Au fil de ces pages, nous allons essayer de répondre à la question suivante : est-il intéressant de remplacer le solveur direct actuel du programme METAFOR par un solveur itératif ?

Ce premier chapitre introductif décrit le problème à partir des équations d'équilibre de la mécanique des milieux continus. Nous expliquerons le fonctionnement de METAFOR et nous montrerons l'origine du système à résoudre dans le processus de Newton-Raphson. Les propriétés de la matrice du système nous seront très utiles pour orienter nos recherches parmi les méthodes existantes.

Le chapitre 2 montrera comment il est possible de gagner de la mémoire en utilisant une méthode itérative. Comme nous l'avons déjà annoncé, ces méthodes permettent de se

débarrasser de la grande quantité de zéros contenus sous la ligne de ciel de la matrice. Nous verrons plus tard que ce gain de mémoire peut représenter plusieurs dizaines de Mo lorsque l'on traite des gros problèmes tridimensionnels.

Les deux chapitres suivants forment la partie théorique de ce travail. Les méthodes itératives sont déduites de la théorie des projections dans le chapitre 3. A ce stade, il sera déjà intéressant de comparer d'un point de vue théorique les diverses méthodes. On pourra essayer de prédire leurs avantages et inconvénients pour résoudre le système qui nous intéresse.
Le chapitre 4 est consacré aux méthodes de préconditionnement. Bien qu'il existe beaucoup de méthodes nous focaliserons notre attention sur les algorithmes de factorisation incomplète parce qu'il semble qu'elles sont applicables dans beaucoup de domaines. Elles ont par exemple fourni des intéressant en mécanique des fluides [D1] et en élasticité linéaire [P2, S4, S5, S6].

Une grande moitié de ce rapport (le chapitre 6) est consacrée à la description des nombreux tests effectués pour étudier le comportement des solveurs utilisés pour résoudre des problèmes rencontrés couramment en mécanique du solide.

Le lecteur trouvera à la fin de ce chapitre une brève synthèse des résultats permettant de choisir le solveur qui est le mieux adapté pour un problème donné.

Ce travail de fin d'études ne constitue qu'une première approche rapide des méthodes itératives. Les perspectives dans le domaine sont très nombreuses vu le nombre d'algorithmes et le nombre d'ouvrages théoriques qui voient le jour chaque année.

L'idéal serait d'obtenir une 'boîte noire' permettant à l'utilisateur de ne pas se soucier de la méthode employée pourvu qu'il ait la solution exacte le plus économiquement et le plus rapidement possible. Personne n'y est encore arrivé !

1.2 Equations d'équilibre (grandes déformations)

Dans cette section, nous décrivons brièvement le programme METAFOR et en particulier la linéarisation des équations d'équilibre conduisant au système à résoudre. Les relations suivantes sont tirées de [P6].

1.2.1 Equations d'équilibre

Sous l'hypothèse de régime quasi-statique, l'équilibre d'un corps quelconque subissant des grands déplacements et des grandes déformations peut se résumer à l'équation d'apparence (!) très simple exprimant l'égalité des forces internes et des forces externes :

$$\vec{F}^{int} = \vec{F}^{ext}$$

avec, les expressions des forces

$$\vec{F}^{\,\mathrm{int}} = \int_{V(t)} [B]\{\sigma\}\, dV$$
$$\vec{F}^{ext} = \int_{V(t)} \rho [H]^T \{b\}\, dV + \int_{S(t)} [H]^T \{t\}\, dS$$

où $[B]$ est le tenseur reliant les déformations aux déplacements,
$[H]$ est une matrice relative aux fonctions de forme,
$\{\sigma\}$ est le tenseur contrainte mis sous forme vectorielle,
$\{b\}$ et $\{t\}$ sont respectivement les charges volumiques et surfaciques imposées.

1.2.2 Résolution de l'équation d'équilibre

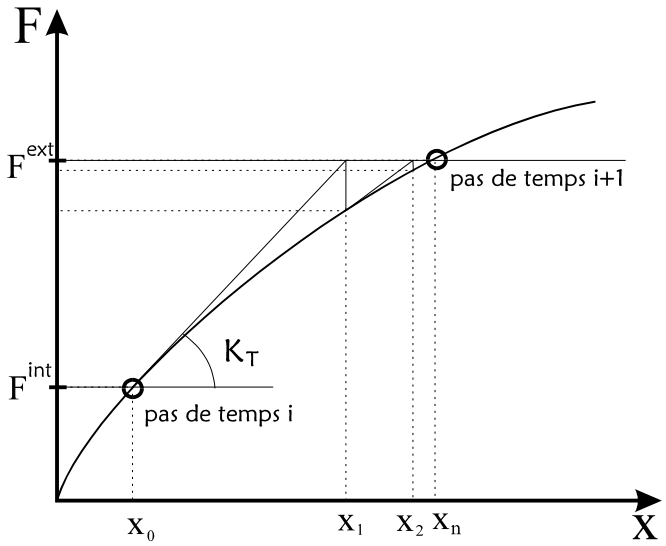
Résoudre ce système d'équations non linéaires directement est une tâche très difficile. On peut par exemple essayer de minimiser une fonctionnelle (énergie de déformation par exemple) par une méthode itérative d'optimisation, comme le gradient conjugué, mais cette fonctionnelle n'existe pas toujours, et lorsqu'elle existe, la présence de plusieurs minima et l'impossibilité de choisir le seul qui est physique posent un gros problème. C'est pourquoi on utilise habituellement une méthode incrémentale : on fait varier la charge de la valeur initiale nulle à la valeur finale fixée par l'utilisateur de METAFOR. Ceci permet de parcourir le chemin physique réellement suivi par le corps en déformation.

Supposons qu'une configuration d'équilibre soit connue. On incrémente la charge (on passe au pas de temps[1] suivant). Le corps étudié n'est plus en équilibre. Des forces caractéristiques de ce défaut d'équilibre apparaissent :

$$F_n^{HE} = F_n^{int} - F_n^{ext} \qquad \vec{F}^{HE} \neq \vec{0}$$

[1] Le temps est ici un pseudo-temps, sauf pour les phénomènes visqueux et dynamiques pour lesquels la déformation finale dépend de la vitesse de chargement.

On développe alors le défaut d'équilibre (appelé aussi résidu) en série de Taylor pour effectuer des itérations de Newton-Raphson et converger vers un nouvel état d'équilibre.



Le dessin ci-contre schématise la situation à une dimension. Pour simplifier les choses, nous avons considéré les forces externes indépendantes du déplacement.

$$\vec{F}^{HE(i+1)} = \vec{F}^{HE(i)} + \left(\frac{\partial \vec{F}^{HE(i)}}{\partial \vec{x}} \right)_{\vec{x}^{(i)}} \Delta \vec{x}$$
$$= \vec{F}^{HE(i)} + K_T \, \Delta \vec{x}$$

La dérivée du résidu d'équilibre par rapport aux positions nodales constitue la matrice de raideur tangente K_T :

$$K_T = \left(\frac{\partial \vec{F}^{HE}}{\partial \vec{x}} \right)_{\vec{x}^{(i)}} = \left(\frac{\partial \vec{F}^{int}}{\partial \vec{x}} \right)_{\vec{x}^{(i)}} - \left(\frac{\partial \vec{F}^{ext}}{\partial \vec{x}} \right)_{\vec{x}^{(i)}}$$

Les itérations de Newton-Raphson sont donc décrites par les équations matricielles

$$\begin{cases} \Delta \vec{x} = -K_T^{-1} \; \vec{F}^{HE(i)} \\ \vec{x}^{(i+1)} = \vec{x}^{(i)} + \Delta \vec{x} \end{cases}$$

Le but du travail est d'effectuer cette opération d'inversion matricielle en utilisant, si possible, moins de temps et d'espace mémoire qu'un solveur direct traditionnel.

Dans un code éléments finis, deux algorithmes se partagent presque tout le temps CPU utilisé par la machine.

Le premier est l'évaluation de la matrice K_T elle-même. Elle peut se faire de deux manières différentes : soit analytiquement lorsqu'une expression mathématique est disponible pour la loi de comportement considérée ; soit numériquement. Dans ce dernier cas, chaque terme de chaque matrice élémentaire est évalué en perturbant la position les nœuds de l'élément dans la configuration courante pour calculer la variation des forces internes et externes élémentaires résultante. Pour chaque perturbation, il faut intégrer la loi constitutive du matériau autant de fois que le nombre de points de Gauss de l'élément pour évaluer les contraintes. Une fois calculée, la matrice élémentaire est assemblée dans la grande matrice structurale K_T.

Le nombre d'opérations nécessaires à un tel calcul est proportionnel au nombre d'éléments de la structure.

La résolution du système linéaire résultant constitue le deuxième calcul coûteux en temps CPU. Celui-ci est proportionnel à N^2m (où N est la taille du système et m sa largeur de bande) pour le solveur actuel et pourra être réduit (dans certains cas) par l'utilisation d'autres solveurs.

Pour de très gros problèmes, la quantité de temps passé à résoudre le système devient de plus en plus importante par rapport à l'opération de construction de la matrice. La réduction de temps de calcul du système linéarisé par les méthodes itératives sera donc amplifiée. Ceci constitue un nouvel argument en faveur des solveurs itératifs.

1.3 Caractéristiques de la matrice de raideur tangente

1.3.1 Introduction

Les caractéristiques de la matrice à inverser vont influencer fortement le choix de l'algorithme de résolution. Calculons tout d'abord la dérivée des forces internes par rapport aux positions nodales dans le cas d'un élément quadrangulaire isoparamétrique 2D [P6] :

$$\left(d_g\, F_{k,i}^{Int}\right) = \iint \left(d_g\, \sigma_{ij}\right)\left[\frac{\partial \phi_k}{\partial x_j} \det J\right] d\xi\, d\eta + \iint \sigma_{ij}\, d_g\left[\frac{\partial \phi_k}{\partial x_j} \det J\right] d\xi\, d\eta$$

où les indices *k* et *i* représentent respectivement le nœud et le direction considérée,
ϕ_i sont les fonctions de forme de l'élément,
$\det J = \frac{\partial x_1}{\partial \xi}\frac{\partial x_2}{\partial \eta} - \frac{\partial x_1}{\partial \eta}\frac{\partial x_2}{\partial \xi}$.

1.3.2 Non symétrie de la matrice

Bien qu'elle possède une structure symétrique, la matrice de raideur tangente est en général NON SYMETRIQUE. On peut citer plusieurs causes qui expliquent cette propriété :

- Linéarisation de la dérivée de Jaumann[2] :

Pour évaluer la variation du tenseur contrainte par rapport aux inconnues, il faut effectuer une linéarisation de la loi constitutive du matériau. La linéarisation de la dérivée objective utilisée dans cette équation provoque l'apparition de termes non symétriques qui sont dus notamment aux rotations du matériau. Dans le cas de la dérivée objective de Jaumann, utilisée dans METAFOR, la matrice obtenue est faiblement non symétrique. On peut montrer que l'expression de K_T peut être écrite sous la forme :

$$K_T = \int_{V(t)} B^T\, M^{**}\, B\, dV \qquad M^{**}_{ijkl} = M_{ijkl} + \sigma_{ij}\, \delta_{kl}$$

avec *M*, le tenseur matériel continu caractérisant le matériau considéré,
B, le tenseur reliant les déformations aux déplacements.

On voit que les termes non symétriques ($\sigma_{ij}\, \delta_{kl}$) sont proportionnels aux contraintes et sont petits devant les paramètres du matériau contenus dans le tenseur *M* tant que les déformations sont petites devant l'unité.

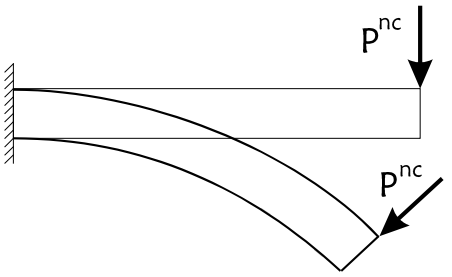
[2] La théorie étant très complexe, nous nous contenterons de ce paragraphe d'explications fortement résumées. Le lecteur trouvera plus de détails dans [P6].

Si on néglige les termes non symétriques, on peut montrer que cela revient à modifier la loi constitutive du matériau en remplaçant la dérivée de Jaumann des contraintes de Cauchy par la dérivée de Jaumann des contraintes de Kirchhoff.

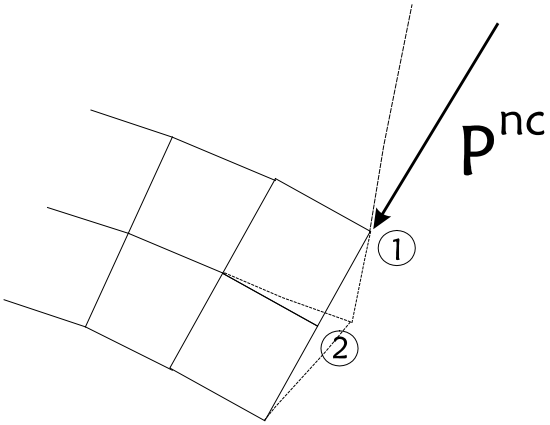
Notons enfin que l'expression ci-dessus peut être rendue totalement symétrique en choisissant d'autres dérivées objectives (par exemple la dérivée de Truesdell des contraintes de Cauchy). Cependant, les schémas d'intégration les plus simples sont basés sur la dérivée de Jaumann des contraintes de Cauchy, ce qui justifie son emploi dans METAFOR. De plus, nous allons voir qu'il existe d'autres sources d'asymétrie.

- Les forces non conservatives :

L'asymétrie de la matrice est provoquée aussi par la présence de forces non conservatives (forces dont l'intensité et la direction dépendent de la déformation du corps). On citera comme exemple les forces de pression et les charges suiveuses.



En effet, lors du calcul de K_T, il faut évaluer la variation des forces externes suite à une variation de configuration du corps étudié.



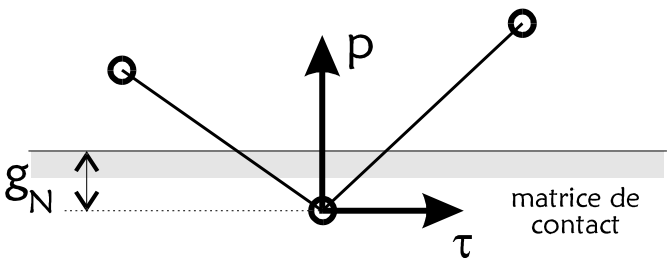
Par exemple, dans le cas d'une force qui est en permanence alignée sur un segment et appliquée à une extrémité (extrémité 1 sur la figure ci-contre) de celui-ci, n'importe quelle variation des extrémités du segment provoquera une variation de force appliquée en 1. Par contre, la variation de la force externe appliquée en 2 sera toujours nulle puisqu'il n'y en a pas.

$$\frac{\partial F_1^{ext}}{\partial x_2} \neq 0 \qquad \frac{\partial F_2^{ext}}{\partial x_1} = 0$$

La contribution de ces termes non symétriques peut être arbitrairement grande.

- Le contact avec frottement :

Le programme METAFOR utilise la méthode de la pénalité pour modéliser le contact entre solides. Ce phénomène induit automatiquement des termes non-symétriques lorsqu'il y a du frottement.



D'une manière générale, on peut écrire la loi constitutive de contact sur la forme matricielle (dans des axes locaux qui tournent avec la matière) :

$$\begin{bmatrix} \dot{\tau} \\ \dot{p} \end{bmatrix} = \begin{bmatrix} \dfrac{h\,K_T}{h+\alpha\,K_T} & \dfrac{\alpha\,\mu\,sign(\tau)\,K_T\,K_N}{h+\alpha\,K_T} \\ 0 & K_N \end{bmatrix} \begin{bmatrix} \dot{g}_T \\ \dot{g}_N \end{bmatrix}$$

où h est un coefficient caractérisant l'écrouissage ($h = 0$ si pas d'écrouissage),
τ et p sont les forces tangentielles et normales de contact,
g_T et g_N sont les 'gaps' tangentiels et normaux,
α est un coefficient valant 1 pour un contact glissant et 0 pour un contact collant,
μ est le coefficient de frottement,
K_T et K_N sont les coefficients de pénalité tangentielle et normale.

On voit que l'effort normal dépend uniquement du gap normal, c'est-à-dire de la pénétration du nœud considéré dans la matrice de contact. Par contre, l'effort tangentiel dépend des deux gaps s'il y a du frottement.

1.3.3 Symétrisation de la matrice de raideur

Nous avons vu que l'asymétrie de la matrice de raideur est provoquée principalement par trois facteurs : la linéarisation de la dérivée de Jaumann, le contact avec frottement et les forces non conservatives. Cependant, puisque le but de l'inversion de celle-ci est simplement de trouver une nouvelle configuration pour laquelle le résidu a diminué, il est souvent commode de la symétriser. On obtient alors un Δx différent de celui obtenu avec la matrice non symétrique mais le temps de calcul et l'espace mémoire utilisé pour l'inversion sont divisés approximativement par deux (dans le cas de l'algorithme de Gauss).

Deux procédures de symétrisation sont pour l'instant disponible : la première consiste à calculer uniquement de la partie triangulaire supérieure de la matrice en ignorant la partie triangulaire inférieure. La seconde réalise une moyenne entre les termes diagonalement opposés (meilleure approximation).

Il existe des cas particuliers, habituellement des problèmes de contact avec forces non conservatives, pour lesquels la symétrisation entraîne une divergence du processus de Newton-Raphson. Pour élaborer un solveur qui couvre tous les problèmes de grandes déformations rencontrés dans METAFOR, il est donc absolument nécessaire de considérer K_T comme étant non symétrique en général et de la traiter en tant que telle.

Cependant, nous n'abandonnons pas complètement l'idée d'une symétrisation de la matrice et il sera intéressant de voir à l'œuvre les solveurs itératifs spécialisés dans ce type de matrices.

Nous verrons plus tard que le choix d'un solveur itératif dépend en grande partie des propriétés de la matrice du système. Il est donc intéressant de se poser la question suivante : la matrice diagonalisée est-elle définie positive ? Si la réponse est oui, il sera possible de résoudre le système par la méthode bien connue du gradient conjugué. Celle-ci a maintenant

fait ses preuves en élastostatique. Pour ce cas particulier, la matrice est symétrique et définie positive parce qu'il existe une énergie de déformation interne quadratique qui est, par définition, positive.

Malheureusement, dans le cas de METAFOR, les phénomènes sont non linéaires (grandes rotations, grandes déformations, plasticité,...) et aucune information ne peut être déduite des relations exprimant K_T. Cependant, il sera intéressant d'essayer ce type de solveur dans les cas où les non symétries sont faibles (contact sans frottement, forces conservatives) vu les résultats impressionnants obtenus récemment en élasticité linéaire [S4, S5, S6].

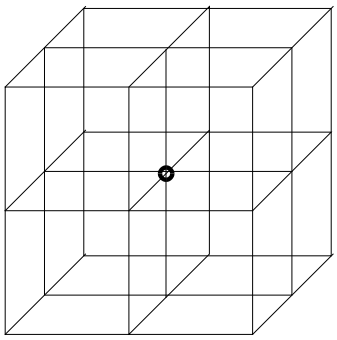
1.3.4 La matrice est creuse (sparse matrix)

Si les propriétés précédentes ne laissent pas présager de bons résultats, le fait que la matrice de raideur tangente soit très creuse est un atout majeur pour la résolution du système par des méthodes itératives. Comme nous allons le voir lors des développements théoriques, la seule opération matricielle utilisée lors de l'exécution d'un solveur itératif est une multiplication matrice par vecteur. Le nombre d'opérations nécessaires est égal au nombre d'éléments non nuls de la matrice. En conséquence, un solveur itératif effectuera une itération d'autant plus vite que la matrice est creuse.

- Evolution de la structure de la matrice lors d'un raffinement de maillage :

La matrice de raideur a approximativement un nombre constant de termes non nuls par ligne. Ce nombre ne dépend pas de la finesse de discrétisation de la structure.

En effet, considérons par exemple un corps 3D soumis à des contraintes mécaniques (3 inconnues par nœud) et maillé en transfini à l'aide d'hexaèdres. Un nœud quelconque intérieur au maillage est couplé uniquement aux nœuds des éléments dont il fait partie. Le schéma ci-contre représente la situation. Il faut donc considérer 3 * 9 nœuds et, puisqu'il y a 3 inconnues par nœud, cela fait 81 termes non nuls par ligne (ou par colonne).



Bien sûr, les nœuds de surface sont couplés avec un nombre plus faibles. De plus, il est aussi possible de mailler la structure différemment et d'avoir, même pour un nœud intérieur, un nombre différent de 81. Mais ce qui est important, c'est que ce nombre soit indépendant du nombre de degrés de liberté de la structure.

On observe donc un nombre d'éléments non nuls variant linéairement avec le nombre de degrés de liberté de la structure.

En conséquence, lorsque l'on raffine le maillage d'un corps, sa matrice de raideur tend à devenir de plus en plus creuse (on atteint facilement moins d'un pour-cent de termes non nuls).

→ Le nombre d'opérations nécessaires pour effectuer une itération varie donc linéairement avec la taille du système (à comparer avec N^2m pour Gauss)

1.4 A l'heure actuelle dans METAFOR

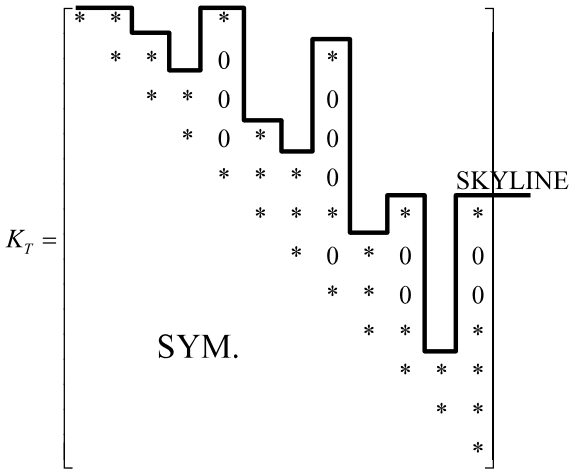
1.4.1 Résolution du système

METAFOR résout le système par décomposition LU (cas non symétrique) ou LDL^T (cas symétrique). Cette méthode possède l'avantage d'être très simple et très robuste. Pour réduire le nombre d'opérations au minimum, l'algorithme de réarrangement de SLOAN [S7] est exécuté lors de l'allocation dynamique de mémoire (la structure de la matrice de raideur est connue avant tout calcul puisqu'elle n'est fonction que de la géométrie du maillage et des conditions aux limites). Cet algorithme minimise la largeur de bande de la matrice. Il est, d'après son auteur, plus performant que les algorithmes classiques de réarrangement tels que 'Reverse Kuthill McKee' [S1].
Contrairement à ce que l'on pourrait penser, cet algorithme sera aussi utilisé dans le cas des solveurs itératifs pour optimiser le calcul du préconditionneur.

1.4.2 Méthode de stockage

Si la matrice K_T est très creuse, il n'en n'est pas de même pour les deux matrices triangulaires L et U définissant sa factorisation.
Pour économiser l'espace mémoire au maximum, le format SKYLINE [P3] est utilisé. C'est le seul format compatible avec l'élimination de Gauss. Il consiste à définir la ligne de ciel de la matrice comme sur la figure ci-dessous et de stocker les éléments sous cette ligne dans deux grands vecteurs de réels, SITL pour la partie triangulaire inférieure et SITU pour la supérieure.

Un vecteur d'entiers (LOCSIT) contient la hauteur de la ligne de ciel à partir de l'élément diagonal. Lors de l'élimination, seuls les termes sous cette ligne devront être modifiés (il suffit de regarder les relations de l'élimination de Gauss pour s'en convaincre). On effectue l'élimination directement dans ces vecteurs, si bien qu'à la fin du processus SITL et SITU contiennent respectivement les matrices L et U au format SKYLINE.



L'inconvénient majeur de la méthode est la nécessité de stocker tous les zéros de la matrice sous la ligne de ciel. Pour des structures à trois dimensions, cela peut représenter une quantité de mémoire très importante (trois à quatre fois supérieure à celle utilisée pour stocker les termes non nuls).

Ce genre de stockage n'est pas optimum pour les solveurs itératifs puisqu'on veut profiter du fait que la matrice de raideur est très creuse pour réduire le nombre d'opérations. Il faudra donc examiner les autres méthodes de stockage disponibles.

1.4.3 Avantages du solveur actuel

Les trois points suivants expliquent le succès de la méthode Gauss + SKYLINE pour résoudre les systèmes linéarisés.

- Robustesse :

C'est la principale qualité de ce type de solveur. La méthode fournit toujours une solution (sauf si l'on rencontre un pivot nul) quel que soit le problème rencontré. Celle-ci est peut-être inexacte par l'accumulation d'erreurs d'arrondis si la matrice du système est très mal conditionnée. Dans ce cas, l'algorithme de Newton-Raphson peut tout de même être poursuivi et parfois même converger. En particulier, puisque c'est une méthode directe, on n'a jamais de divergence.
Nous verrons plus loin que le mauvais conditionnement du système provoque d'autres difficultés pour les méthodes itératives qui peuvent être partiellement contournées à l'aide du choix adéquat d'une méthode de préconditionnement de la matrice K_T.

- Pas de paramètres à ajuster :

Contrairement aux solveurs itératifs, l'algorithme de Gauss peut être exécuté aveuglément, comme une boîte noire.

- Coût (C.P.U., mémoire) prévisible :

Lorsqu'on utilise l'algorithme de Gauss de METAFOR, on sait que chaque résolution de système va être effectuée en un temps bien défini, indépendant de l'itération en cours (mis à part les problèmes de contact avec réactualisation de la ligne de ciel). De plus, la quantité de mémoire nécessaire est déterminée une fois pour toute lors de l'allocation dynamique de mémoire.
Pour les solveurs itératifs, le nombre d'itérations nécessaires pour trouver la solution est déterminé par le spectre des valeurs propres de la matrice qui est inconnu a priori et qui change d'une itération de Newton-Raphson à l'autre. La mémoire nécessaire n'est pas non plus prévisible. Il faut se fixer une borne maximum.

1.4.4 Inconvénients du solveur actuel

Les deux points suivants résument pourquoi, aujourd'hui, on s'intéresse à d'autres méthodes de résolution.

- Nombre d'opérations trop élevé :

Le nombre d'opérations est proportionnel à N^2m. (N : dimension du système, m : largeur de bande) alors que l'on s'attend à une dépendance presque linéaire pour les solveurs itératifs. Il faudra déterminer le point d'intersection de ces deux courbes.

- Trop gourmand en mémoire :

En particulier, la nécessité de stocker en mémoire beaucoup de zéros est fort gênante surtout pour de gros problèmes.

Conclusion : la méthode est inadaptée pour traiter de gros problèmes 3D mais possède des avantages non négligeables pour des petits problèmes.

1.5 Types de problèmes à envisager

Nous nous intéresserons à tous les types de problèmes possibles et actuellement réalisables avec METAFOR, à savoir les problèmes 2D, 3D, mécaniques, thermiques, thermomécaniques (avec et sans couplage) ainsi que les problèmes de contacts entre solides.
Pour ce dernier point, il y a un petit problème : la résolution des problèmes de contact 3D est en cours de développement. Il n'est donc pas possible d'essayer le solveur sur des gros problèmes de contact avec des matrices fortement non symétriques.

1.6 Autres travaux sur la question

1.6.1 Brève historique

Bien que l'idée de résoudre des systèmes d'équations par des méthodes itératives ne soit pas du tout récente (Jacobi, Gauss-Seidel, ...), ce n'est que depuis quelques dizaines d'années que des méthodes performantes pouvant rivaliser avec les méthodes directes ont été imaginées et étudiées en profondeur par des mathématiciens (fin des années 60).

Même si la méthode du gradient conjugué à été développée pour la première fois en 1952, il a fallu attendre la fin des années 80 pour que de nouveaux et puissants préconditionneurs soient implantés et testés. Les récents progrès dans le matériel informatique ont contribué énormément au développement de nouveaux solveurs (les PC actuels sont au moins aussi puissants que les stations de travail d'il y a dix ans).

Aujourd'hui, des théorèmes existent et garantissent la convergence dans des certains cas particuliers (si la matrice est symétrique est définie positive). Les méthodes itératives peuvent entrer en compétition avec les anciennes méthodes directes.

Les chercheurs commencent à s'orienter maintenant vers les problèmes encore plus compliqués caractérisés par des matrices non symétriques et non définies positives. C'est le cas de la mécanique du solide en grandes déformations.

1.6.2 En élasticité linéaire

Beaucoup de travaux [P2, S4, S5, S6 pour ne citer qu'eux] ont été effectués pour inclure des algorithmes de gradients conjugués dans des codes d'élastostatique. En général, les résultats sont intéressants. On observe des diminutions de temps de calcul parfois assez impressionnantes. De plus, les problèmes pour lesquels le gradient conjugué reste impuissant sont assez bien définis.
Il est donc intéressant de s'inspirer de ces méthodes et de les adapter aux problèmes non linéaires.

1.6.3 En grandes déformations

Lorsqu'on entre dans le domaine non linéaire, les travaux sur la question se font beaucoup plus rares. Jongler avec des matrices qui ne sont ni symétriques ni définies positives est bien plus difficile que dans le cas où elles le sont. Il n'existe pas beaucoup de théorèmes relatifs à la convergence ou au préconditionnement de tels systèmes. Chaque chercheur a ses propres convictions et ses propres recettes de cuisine pour les problèmes qui le préoccupent.

Il est intéressant de constater que l'on trouve aussi ce type de matrice en mécanique des fluides (discrétisation en volumes finis) [D1]. On pourra donc se baser sur les résultats obtenus tout en gardant en tête que les équations sont discrétisées différemment.

En mécanique du solide, les programmeurs de WARP3D et ABAQUS [H1] ont sorti récemment une nouvelle version de leur code incluant un gradient conjugué comme solveur itératif (en se limitant à des cas où il est possible d'obtenir une matrice symétrique définie positive).

Ils présentent des résultats assez prometteurs mais, bien sûr, moins impressionnants qu'en linéaire. En effet, lorsqu'on traite des problèmes linéaires par la méthode des éléments finis, la résolution du système est le seul gros calcul à effectuer. Par exemple, l'évaluation de la matrice de raideur est, de loin, beaucoup plus rapide qu'en non linéaire.

1.6.4 La méthode EBE

Les nouveaux programmes tels que WARP3D et ABAQUS utilisent aussi une autre méthode de résolution leur permettant de ne jamais assembler la matrice de raideur : la méthode 'element by element' (EBE). Celle-ci permet aussi de tirer profit des ordinateurs multiprocesseurs. Des recherches intensives [S1] sont focalisées pour le moment sur la parallélisation des solveurs itératifs et leur insertion sur de tels ordinateurs.

Par exemple, il est possible de préconditionner non pas la matrice du système mais bien chaque matrice tangente élémentaire prise séparément. On peut aussi effectuer l'opération de base matrice fois vecteur beaucoup plus rapidement en confiant le calcul de chaque composante du vecteur résultat à un processeur différent.

En ce qui concerne METAFOR, il est installé sur une station de travail à processeur séquentiel. Nous ne pourrons donc pas tirer profit de cette méthode, d'autant plus qu'elle nécessiterait un remaniement profond du code et un travail de programmation assez grand.

1.7 A propos de la résolution du système non linéaire

1.7.1 Introduction

Par sa concision, le titre de ce travail peut paraître déroutant : on fait allusion à la résolution d'un système non linéaire de manière itérative. En fait, il faut comprendre que l'on résout un système linéaire provenant de la linéarisation d'équations d'équilibre non linéaire. Cependant le terme non linéaire est indispensable pour caractériser les matrices rencontrées (non symétriques,...).

Remarquons qu'il existe de nombreuses autres méthodes que Newton-Raphson pour résoudre itérativement un système non linéaire. Nous citons ici quelques-unes d'entre elles qui pourraient faire l'objet d'améliorations futures de METAFOR. Celles-ci sont totalement indépendantes du type de solveur linéaire utilisé.

1.7.2 Quasi-Newton

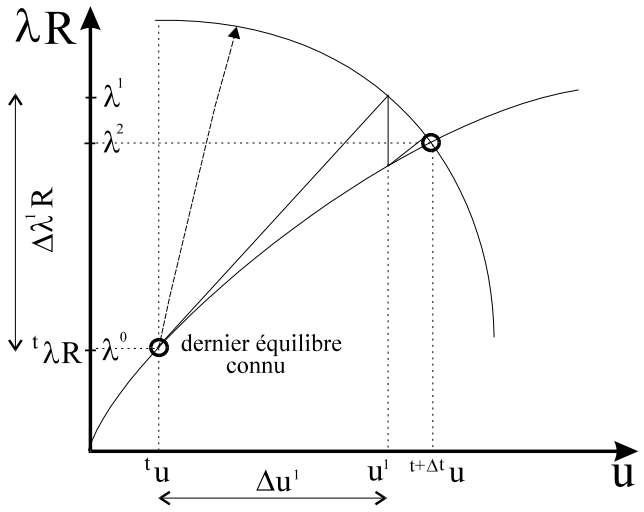
Les méthodes de Quasi-Newton [K1] sont couramment utilisées en optimisation et peuvent être étendues à la résolution de systèmes non linéaires. Elles consistent à actualiser la matrice de raideur tangente du système à l'aide de formules de récurrence. Les méthodes BFGS, DFP font partie de cette catégorie.

Grâce à ce type de méthodes, l'évaluation de la matrice de raideur devient beaucoup moins coûteuse. Cependant, on perd la propriété principale de l'algorithme de Newton-Raphson (appelé aussi Newton exact), c'est-à-dire la convergence quadratique vers la solution.

1.7.3 Arc length

Ce type de méthode est particulièrement bien adapté pour suivre des courbes d'équilibre au delà de points où la structure s'adoucit, c'est-à-dire au delà des extrema de la fonction donnant la force interne d'après le déplacement. Elle a été développée notamment par M. Crisfield [C2, R1].

Le dessin ci-contre montre le principe de la méthode



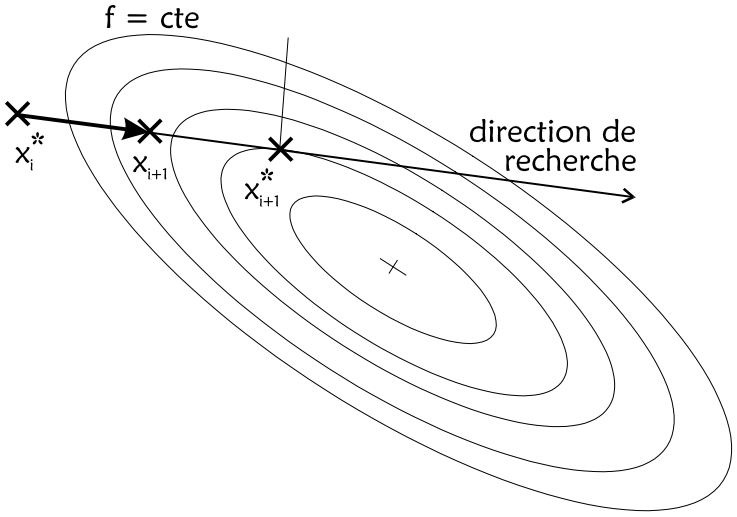
Au lieu d'utiliser un incrément de charge extérieure constant à chaque itération, celui-ci devient une inconnue au même titre que le déplacement. La méthode revient à tracer une hyper-sphère (un cercle lorsqu'on travaille à une dimension) représentant la charge extérieure en fonction du déplacement et de rechercher le point d'intersection de celle-ci avec la fonction force interne - déplacement pour pouvoir progresser d'un point d'équilibre à l'autre.

1.7.4 Line search

Cette technique [C2] ne permet pas de résoudre le système non linéaire mais elle peut, lorsqu'elle est combinée à une autre méthode, accélérer la convergence significativement.
Elle consiste à définir une direction de recherche par la formule

$$s^{(i+1)} = x^{(i+1)} - x^{(i)} = \Delta x^{(i)}$$

Une fois celle-ci définie, on essaye de minimiser le résidu (ou une autre fonction *f*, suivant le type de problème) dans l'espace des déplacements selon cette direction. Il est possible alors d'obtenir une meilleure solution que celle obtenue par le solveur (notons que celui-ci peut être itératif ou direct).



Le dessin ci-contre représente la situation dans un espace à deux dimensions :

- x_i^* : positions à l'itération précédente.
- x_{i+1} : solution donnée par le solveur.
- x_{i+1}^* : solution optimale minimisant la fonction *f*.

2. Méthodes de stockage

2.1 Introduction

Comme indiqué dans le premier chapitre, la méthode de stockage SKYLINE n'est pas la meilleure pour une implantation optimale des méthodes itératives. En effet, la puissance de celles-ci provient en grande partie du fait qu'elles ne modifient pas la matrice du système. Les seules opérations nécessaires sont des multiplications de matrices par vecteurs et des produits scalaires. Il faut donc pouvoir mémoriser uniquement les éléments non nuls de la matrice, car ce sont eux et, uniquement eux, qui interviendront dans les opérations de calcul.

Nous verrons aussi plus tard que le succès d'une méthode itérative dépend du préconditionnement du système. Celui-ci intervient dans le calcul par l'intermédiaire d'une autre matrice creuse qu'il faudra stocker en mémoire à côté de la matrice de raideur. Celle-ci intervient, quant à elle, par le calcul du produit de son inverse par un vecteur. On utilisera donc des formats légèrement différents pour obtenir une implémentation optimale.

Il existe des formats spéciaux pour tenir compte de la symétrie d'une matrice et ne stocker que la moitié de celle-ci. Ils ne seront pas utilisés dans ce travail pour plusieurs raisons :

- La matrice à stocker est en général non symétrique sauf si on la diagonalise.
- Si la matrice est symétrique, le préconditionneur ne l'est pas forcément.
- Les opérations matricielles deviennent plus compliquées à programmer et plus lentes à l'exécution, en particulier la recherche de l'approximation de l'inverse pour le préconditionnement.

Il est utile de définir, pour la suite, quelques variables

Notons *NNZ* (‘*N Non zero*’), le nombre d'éléments non nul de la matrice creuse à stocker et *N*, la dimension de la matrice du système (*NSYS* dans METAFOR).

Les méthodes suivantes ont principalement été tirées du manuel de SPARSKIT développé par Y. Saad [S2]. Elles sont utilisées non seulement dans tous les codes de calcul de mécanique (mis à part les codes basés sur la méthode element by element) mais aussi en électricité pour la résolution des grands réseaux électriques et en mécanique des fluides.

2.2 Le format CSR (Compressed Sparse Row)

C'est le format que nous utiliserons pour stocker la matrice de raideur. Son principal avantage est la facilité avec laquelle on peut calculer l'action de la matrice sur un vecteur.

On décompose la matrice en trois vecteurs :

- Le vecteur de réels *A* contient tous les éléments non nuls de la matrice creuse, considérée ligne par ligne. Sa longueur vaut *NNZ*.
- Le vecteur d'entiers *JA* contient les indices des colonnes de chaques éléments du vecteur *A*. Sa taille est aussi *NNZ*.
- Le vecteur d'entiers *IA* contient des pointeurs vers le début de chaque ligne dans les vecteurs *A* et *IA*. Par exemple, *IA(i)* contient le début de la $i^{ième}$ ligne de la matrice dans les vecteurs *A* et *IA*. Sa taille vaut *N+1* ; l'élément *N+1* pointant vers la ligne fictive *N+1* de la matrice.

Pour mieux visualiser la méthode, voici un petit exemple d'une matrice 5 x 5 :

0.2	0.1	0	0	0
0.8	0.75	0	0.3	0
0	0	1.7	0	0
0	0.6	0	2.1	0.4
0	0	0	1.3	1.1

En reprenant les notations de METAFOR, on a

NNZ = 11, *NSYS* = 5

Pour une matrice pleine, le format n'est , bien sûr pas avantageux. Remarquons que déjà dans ce cas-ci, on économise de la mémoire (si les réels sont codés avec 8 octets et les entiers avec 4 octects)

A	0.2	0.1	0.8	0.75	0.3	1.7	0.6	2.1	0.4	1.3	1.1
JA	1	2	1	2	4	3	2	4	5	4	5
IA	1	3	6	7	10	12					

On peut remarquer que l'ordre des éléments non nuls d'une même ligne n'est pas important. Cependant, on peut les ordonner. Cela accélérera certains calculs (notamment celui du préconditionneur).

La symétrie de structure de la matrice est aussi perdue. Il serait possible de gagner encore de la mémoire en tenant compte de cette propriété. Cependant, les opérations matricielles deviendraient plus difficiles à programmer et plus lentes à l'exécution.

2.3 Le format MSR (Modified Sparse Row)

Variation intéressante du format CSR, le format MSR tient compte du fait que la plupart des matrices considérées dans la suite possèdent une diagonale pleine. C'est le format que nous utiliserons pour le stockage du préconditionneur. Ce choix est justifié par le rôle particulier des éléments diagonaux lors d'une double substitution (avant et arrière).

On ne garde plus que deux vecteurs *A* et *JA*. La diagonale est stockée dans les *N* premiers éléments de *A* (l'élément *N+1* de *A* n'est pas utilisé) et les éléments non nuls de la matrice sont placés à partir de la position *N+2* dans ce vecteur. Les *N+1* premiers éléments de *JA* contiennent alors un vecteur équivalent à *IA* du format CSR et les indices des colonnes sont mémorisés à partir de l'élément *N*+2 de *JA*.

A	0.2	0.75	1.7	2.1	1.1	?	0.1	0.8	0.3	0.6	0.4	1.3
JA	7	8	9	9	11	11	2	1	4	2	5	4

Pour accélérer les opérations avec le préconditionneur, nous utiliserons aussi un troisième vecteur d'entier (que l'on appellera *IA*) de longueur *N* contenant des pointeurs vers le début de la partie triangulaire supérieure pour chaque ligne. On supposera alors que les éléments du vecteur A sont organisés de telle manière à ce que, pour chaque ligne, les éléments sous la diagonale viennent avant la partie supérieure.

2.4 Autres formats

Notons que d'autres méthodes existent mais elles sont parfois moins adaptées aux éléments finis qui nous intéressent :

CSC : (Compressed Sparse Column) : Ce format est identique au premier si on considère les colonnes de la matrice à la place des lignes. Le stockage CSC d'une matrice *A* correspond au stockage CSR de cette même matrice transposée A^T. Il serait tout à fait possible de choisir ce format à la place de CSR.

DIA : (Diagonal format) : Mémorise la matrice sous la forme d'une succession de diagonales plus au moins creuses. Cette méthode est principalement employée pour la résolution de systèmes provenant de la discrétisation d'équations aux dérivées partielles elliptiques.

BSR : (Block Sparse Row format) : Mémorise la matrice sous forme d'une succession de blocs, ceux-ci étant comprimés au format CSR. On peut utiliser ce format pour traiter ensemble toutes les inconnues d'un même nœud (par exemple des blocs 3x3 pour les trois déplacements d'un même nœud). Les divisions par un élément de matrice deviennent une inversion de bloc. Ce format est utilisé en mécanique des fluides par [D1].

2.5 Assemblage de la matrice de raideur

2.5.1 Problème de l'assemblage de la matrice de raideur

A chaque itération de Newton-Raphson, la matrice de raideur tangente est évaluée dans la configuration courante. La procédure est assez simple : on parcourt tous les éléments de la structure. Pour chacun de ceux-ci, une matrice de raideur analytique ou numérique (au choix de l'utilisateur et aux possibilités du code) est calculée, symétrisée ou non et enfin assemblée dans la matrice structurale à l'aide des vecteurs de localisation adéquats.

Lors de l'assemblage dans la matrice structurale, les matrices de raideur élémentaires sont considérées dans un ordre arbitraire (suivant leurs numéros attribués d'après les choix de l'utilisateur et du programme de pré-traitement). Ceci implique que la matrice structurale doit être remplie de cette même manière arbitraire. Cependant, aucune des méthodes discutées dans la section précédente ne permet l'insertion aisée d'un élément n'importe où dans la matrice.

Il faut donc utiliser, lors de l'assemblage, une autre méthode de stockage remédiant à ce défaut au prix d'une quantité de mémoire utilisée plus importante. On utilise une liste liée (forward linked list) complètement décrite par G. Warzee et P. Saint-Georges [S4] dans leurs travaux similaires en élasticité linéaire.

Une fois l'assemblage terminé, un algorithme permet de transformer la liste liée au format CSR.

2.5.2 Méthode de stockage temporaire : la liste liée

2.5.2.1 Description

Nous décrivons dans cette section la procédure de remplissage de la liste liée lors de l'assemblage de la matrice de raideur. Cette méthode n'est pas très compliquée en soi mais son explication est assez ardue parce qu'elle utilise des pointeurs et même des pointeurs de pointeurs. C'est pourquoi, nous présentons ci-dessous un exemple d'assemblage se rapportant à la matrice 5x5 définie pour la présentation des méthodes de stockage.

Initialisation :

IA	*LINK*	*JA*	*A*
-1			
-2			
-3			
-4			
-5			
1			

Nouvelle entrée $A_{2,5} = 0.1$

IA	*LINK*	*JA*	*A*
-1	-2	5	0.1
1			
-3			
-4			
-5			
1 + 1 = 2			

Nouvelle entrée $A_{4,4} = 2.1$

IA	*LINK*	*JA*	*A*
-1	-2	5	0.1
1	-4	4	2.1
-3			
2			
-5			
2 + 1 = 3			

Nouvelle entrée $A_{2,1} = 0.4$

IA	*LINK*	*JA*	*A*
-1	3	5	0.1
1	-4	4	2.1
-3	-2	1	0.4
2			
-5			
3 + 1 = 4			

Nouvelle entrée $A_{2,5} = 0.2$

IA	*LINK*	*JA*	*A*
-1	3	5	0.1 + 0.2 = 0.3
1	-4	4	2.1
-3	-2	1	0.4
2			
-5			
4			

On utilise donc 4 vecteurs. Les vecteurs *A*, *IA* et *JA* possèdent les mêmes caractéristiques que pour le format CSR. On ajoute un vecteur supplémentaire d'entiers *LINK* qui contiendra des pointeurs (c'est le vecteur lien de la liste liée).

- L'initialisation est simple :

Elle consiste à remplir le vecteur *IA* par une suite de nombres négatifs valant le numéro de ligne changé de signe. Le dernier élément (position *N*+1) du vecteur joue un rôle particulier : il contient le numéro de la prochaine place 'vide' dans le vecteur *LINK* (il est donc initialisé à 1 dans un premier temps lorsque celui-ci est entièrement vide). Les vecteurs *A* et *JA* n'ont pas besoin d'être initialisés à 0.

- La liste liée sera construite de la manière suivante :

Ajoutons l'élément $A_{2,5} = 0.1$. On regarde tout d'abord dans la colonne *IA* le nombre en $2^{ème}$ place (puisqu'il s'agit de la $2^{ème}$ ligne). Un nombre négatif (-2) signifie que la ligne est vide pour l'instant. On va donc créer une nouvelle entrée dans la liste liée. IA(2) va contenir la place de la nouvelle entrée dans le vecteur *LINK* (donnée par *IA*(*N*+1)). L'élément LINK(IA(N+1)) va contenir le nombre négatif anciennement contenu dans *IA*(2). Les places correspondantes dans *JA* et *A* sont remplies avec le numéro de la colonne et la valeur de $A_{2,5}$. Enfin, on incrémente la valeur de IA(N+1). La liste est prête pour la prochaine insertion.
L'insertion de l'élément $A_{4,4}$ est similaire.

Lorsqu'on veut ajouter un élément se trouvant sur la même ligne ($A_{2,1}$ ou une nouvelle fois $A_{2,5}$), on suit la procédure suivante : *IA*(2) contient maintenant un nombre positif signifiant que la matrice de raideur possède déjà des éléments sur cette ligne. On parcourt alors le vecteur *LINK* aux positions *LINK*(*IA*(2)), *LINK*(*LINK*(*IA*(2)), ... tant que l'on y trouve des nombres positifs et tant que la colonne de l'élément correspondant n'est pas égale à celle de l'élément à introduire dans la liste. Si on trouve un élément négatif, on doit créer une nouvelle entrée dans la liste pour la ligne correspondante. On suit la procédure précédemment décrite en veillant à ce que l'ancien dernier élément de la ligne pointe bien vers le nouveau. Le nouveau contenant le nombre négatif précédemment trouvé.
Si on trouve un élément de la liste dont la colonne est identique à celle de l'élément à introduire, on ajoute simplement la valeur de celui-ci à l'ancienne.
Lorsque la procédure est terminée, *IA*(*N*+1) contient *NNZ*+1.

2.5.2.2 Passage de la liste liée au format CSR

Une fois la liste liée complètement remplie, repasser au format CSR est une nécessité.
Pour remplir les vecteurs *A*, *IA* et *JA* du format CSR, il suffit de parcourir la liste liée comme on le fait pour insérer un nouvel élément. Cependant, il est impossible d'utiliser les mêmes vecteurs pour la liste liée et pour le stockage final de la matrice.

Pour remédier à ce problème, on peut par exemple utiliser les vecteurs nécessaires pour stocker le préconditionneur pour construire la liste liée (nous verrons que ceux-ci sont toujours au moins aussi grands que ceux de la matrice de raideur). Nous allons voir que ce n'est pas la solution optimale au problème. En effet, à chaque itération de Newton-Raphson, on doit reconstruire entièrement la liste liée. Or ce n'est pas nécessaire puisque la structure de la matrice peut être déterminée une fois pour toute lorsque la géométrie du maillage et les conditions limites sont connues.

La seule objection à cette affirmation provient des problèmes de contact entre solides déformables. Dans ce cas bien particulier, la structure de la matrice change à chaque itération parce que certains nœuds du maillage peuvent entrer en contact avec d'autres nœuds du même maillage provoquant ainsi un couplage entre nœuds non voisins au départ.

Il est a priori impossible de définir quels nœuds vont entrer en contact lors du calcul. METAFOR est déjà actuellement confronté à ce problème parce qu'il doit définir lors de l'allocation dynamique de mémoire, la hauteur de la ligne de ciel. La solution utilisée consiste à considérer, lors de la réservation de mémoire, que tous les nœuds définis par l'utilisateur comme des ''nœuds de contact'' sont susceptibles de rentrer en contact et de générer en conséquence une raideur entre eux. Lors du calcul, à chaque itération, METAFOR effectue une actualisation de la ligne de ciel en fonction des nœuds qui sont réellement en contact dans la configuration courante.

2.5.3 Pré-assemblage de la matrice de raideur

2.5.3.1 Motivations

Dans le cas du format CSR, l'assemblage est une tâche beaucoup plus compliquée comparée au format SKYLINE. Il prend plus de temps en créant une liste liée et plus de mémoire en utilisant des vecteurs auxiliaires indispensables.
Le problème est partiellement[3] résolu en effectuant un pré-assemblage de la matrice de raideur tangente lors de l'allocation dynamique de mémoire, par analogie à la détermination de la hauteur de la ligne de ciel pour le SKYLINE.

- Avantages :

Cette façon de procéder à beaucoup d'avantages notamment l'élimination des vecteurs auxiliaires pour former la liste liée.
Cependant l'intérêt majeur est, bien sûr, la détermination du nombre exact d'éléments non nuls dans la matrice de raideur avant tout calcul. Ceci permet de réserver la quantité de mémoire juste nécessaire pour le bon fonctionnement du calcul. Il ne reste plus qu'un seul problème de mémoire : la détermination du nombre d'éléments non nuls du préconditionneur. Nous en reparlerons dans le chapitre consacré à celui-ci.

- Inconvénients :

Cependant, il sera très difficile de faire une actualisation de la structure de la matrice pour des problèmes de contact entre corps déformables[4]. Ce n'est pas un très gros inconvénient en pratique. En effet, une modification de structure dans une matrice stockée au format SKYLINE entraîne souvent la nécessité de prendre en considération tous les zéros en-dessous de celui-ci. En conséquence, le nombre d'opérations lors de l'élimination de Gauss peut être multiplié par un facteur non négligeable s'ils sont 'mal placés'. Pour le format CSR, nous

[3] L'assemblage est toujours plus lent qu'en utilisant le format SKYLINE.
[4] L'implémentation du contact entre corps déformables est en cours à l'heure actuelle. Nous ne serons donc pas confrontés à ce problème.

devrons traîner lors de tout le calcul certains éléments nuls mais ils seront toujours largement inférieurs en nombre qu'avec le format SKYLINE.

En conclusion : pour le format CSR, on ne mémorisera plus uniquement les éléments non nuls mais plutôt tous les éléments susceptibles de devenir non nuls lors du calcul.

2.5.4 Principe et implémentation dans METAFOR

L'implémentation a été assez difficile à mettre en œuvre. En effet, pour la méthode SKYLINE, les tableaux contenant la matrice (*SITL* et *SITU*) ne sont rempli que lors de l'assemblage de la matrice. Ils étaient donc utilisés comme vecteurs temporaires pour des calculs annexes (vecteurs de localisation, algorithme de SLOAN). Lorsque ces vecteurs ont été remplacés par *IA* et *JA*, ces opérations démolissaient tout le travail de pré-assemblage.

Voici l'enchaînement des opérations pour le pré-assemblage :
- On passe en revue tous les éléments de la structure (y compris les éléments de contact) en considérant des matrices élémentaires de raideur pleines.
- On assemble les matrices élémentaires grâce à une liste liée (les valeurs de A sont inconnues, nous n'avons donc pas besoin de celui-ci).
- La mémoire restante est testée à chaque instant, c'est-à-dire chaque fois qu'une nouvelle entrée est ajoutée dans la liste. Si la mémoire est insuffisante, on retourne un message d'erreur. Il est cependant impossible de savoir la quantité de mémoire qu'il manque (à ce stade, il doit en manquer beaucoup vu que les matrices de réels ne sont pas encore intervenues).
- On transforme une fois pour toute la liste liée au format CSR (les vecteurs IA et JA sont calculés une fois pour toutes). Le nombre d'éléments non nuls maximum (NNZ) est connu.
- On dimensionne correctement les vecteurs caractérisant K_T (*A*, *IA*, *JA*) et le préconditionneur (*ALU*, *JLU*, *JU*)

2.5.5 Amélioration possible ?

Lorsque les routines de contact seront totalement opérationnelles (surtout celles concernant les problèmes à trois dimensions), il sera intéressant de quantifier l'influence des zéros superflus et de voir dans quelles conditions il est préférable de réactualiser les vecteurs *IA* et *JA* de la matrice de raideur pour gagner du temps lors du calcul.

On effectuerait ceci en trois temps :
- Effectuer le pré-assemblage (SKYPRE) pour déterminer NNZ_{max} (indispensable).
- Déterminer si un nouvel assemblage est nécessaire.
- Effectuer l'assemblage aux itérations voulues en utilisant, par exemple, la mémoire réservée pour le préconditionneur pour créer la liste liée nécessaire.

2.6 Opérations matricielles sur des matrices creuses

2.6.1 Introduction

Gagner de la place en mémoire centrale n'est pas le seul but des méthodes de stockage précédemment énumérées. En effet, elles permettent d'éliminer la prise en considération des nombreux éléments nuls lors des opérations matricielles de base. La rapidité d'une méthode sera aussi fonction de la manière dont les routines de multiplication d'une matrice par un vecteur et de produit scalaire seront optimisées.

- Note sur la communication inverse :

On verra que les méthodes itératives sont indépendantes de la manière dont on calcule le produit d'une matrice par un vecteur, c'est pourquoi les nouvelles implantations (QMRPACK,...) [N2] des solveurs itératifs utilisent la méthode le la 'communication inverse'. En deux mots, cela veut dire que la routine du solveur rend la main à la routine appelante chaque fois qu'elle a besoin d'une opération matricielle. Celle-ci lui rend le résultat pour qu'elle puisse continuer. Cette méthode rend le solveur complètement indépendant de la méthode de stockage utilisée. Ce type de routines simplifie la vie aux personnes voulant utiliser un solveur sans se soucier de l'implantation, mais elle est beaucoup plus difficile à programmer. Elle ne convient donc pas pour ce travail.

2.6.2 Multiplication y = A x

La routine de multiplication matricielle est utilisée par tous les solveurs itératifs sans exception. Voici le code FORTRAN utilisé :

```
DO i=1,N
     t=0.0d0
   DO 1010 j=IA(i), IA(i+1)-1
      t=t+A(j)*x(JA(j))
   ENDDO
   y(i)=t
ENDDO
```

Les lignes de la matrice sont parcourues suivant leurs éléments non nuls. Ceux-ci sont multipliés par la composante de x adéquate et chaque contribution est sommée dans un scalaire temporaire t. On effectue donc *NNZ* 'additions + multiplications' à comparer avec N^2 dans le cas d'une matrice pleine

2.6.3 Multiplication $y = A^T x$

Certains solveurs itératifs utilisent non seulement *A* mais aussi sa transposée pour résoudre le problème. La multiplication par la transposée n'est pas immédiate.

Voici le code FORTRAN correspondant :

```
DO i=1,N
   c(i)=0.0D0
ENDDO
DO i=1,N
   DO j=IA(i), IA(i+1)-1
      k=JA(j)
      c(k)=c(k)+A(j)*b(i)
   ENDDO
ENDDO
```

Il faut remplir le vecteur résultat à chaque instant et dans un ordre différent de celui utilisé plus haut : on parcourt de nouveau les lignes de *A*, c'est-à-dire les colonnes de A^T et on ajoute la contribution de l 'élément courant à la composante de *y* correspondant à son numéro de colonne. Pour cette opération, il ne faut pas oublier d'initialiser le vecteur *y* à 0 avant toutes choses.

3. Les algorithmes existants

3.1 Introduction

Le but de ce chapitre est de décrire les algorithmes actuellement disponibles pour résoudre le système d'équations linéarisé découlant des itérations de Newton-Raphson. Nous nous intéresserons principalement aux méthodes instationnaires, les autres méthodes étant beaucoup moins performantes.

Lorsqu'on regarde la quantité de solveurs itératifs disponibles à l'heure actuelle, on pourrait penser qu'il existe plusieurs théories fort différentes pour résoudre le problème. En fait, tous les algorithmes repris dans ce travail sont déduits d'une méthode générale de projection.

L'idée principale de ces algorithmes est de projeter le problème, c'est-à-dire la matrice de raideur tangente, dans un espace réduit, défini par une série de vecteurs orthogonaux bien choisis. On résout alors un deuxième système beaucoup moins gros que le premier correspondant au système projeté. Une solution approchée du système à résoudre est déduite de ce calcul auxiliaire par le retour aux axes de l'espace initial. A chaque itération, on incrémente la dimension de l'espace de projection pour tendre vers la solution exacte.

Les méthodes se différencient par le type de projection (orthogonale ou oblique) utilisée. Parallèlement[5], se développent des versions simplifiées de certains algorithmes pour pouvoir traiter plus simplement des matrices spéciales (symétriques, définies positives).

Dans un premier temps, nous expliquerons la théorie des projections. Chaque méthode itérative sera déduite de celle-ci en détails par la suite. On pourra ainsi comprendre l'origine des forces et des faiblesses de chacune d'entre elles. Nous ferons alors certains choix et

[5] Historiquement, c'est bien sûr l'inverse qui s'est produit : du plus simple au plus compliqué.

certaines prédictions quant à l'efficacité possible de chaque solveur dans le contexte des éléments finis non linéaires. L'abandon de certaines méthodes à ce stade sera largement commenté et justifié.

Dans ce chapitre, nous ne faisons jamais référence à un préconditionnement quelconque du système. L'accélération de la convergence sera introduite dans le prochain chapitre vu la quantité de méthodes existantes.

Pour être en accord avec les manuels d'analyse numérique et pour simplifier les expressions mathématiques, nous changeons momentanément les notations utilisées :
Soit le système à résoudre

$$A\,x = b$$

avec *A*, matrice non singulière de dimension *N x N*.

Les développement théoriques de ce chapitre sont inspirés principalement du nouvel ouvrage de Y. Saad sur le sujet [S1].

3.2 Méthodes stationnaires et instationnaires

3.2.1 Méthodes stationnaires

Elles sont citées ici uniquement pour être complet. On regroupe sous le nom de méthodes stationnaires les méthodes itératives dont le mécanisme (opérations matricielles) ne varie pas d'une itération à une autre.

Les plus connues sont :
- Jacobi
- Gauss-Seidel
- SSOR (Successive Symetric Over Relaxation)

Ces méthodes possèdent, en général, des vitesses de convergence beaucoup trop faibles pour être considérées ici. Cependant, nous verrons qu'elles peuvent être utiles lors du préconditionnement (on couple alors une méthode stationnaire avec une méthode instationnaire).

3.2.2 Méthodes instationnaires (méthodes de projection)

Les méthodes instationnaires, quant à elles, utilisent toute l'information disponible aux itérations précédentes pour converger le plus rapidement possible vers la solution. Elles sont basées, comme nous l'avons déjà dit, sur des projections orthogonales et obliques.

Ces méthodes sont très intéressantes lorsqu'on doit résoudre de gros systèmes. Inversement, pour de petits systèmes, elles peuvent être parfois extrêmement lentes vis-à-vis des solveurs directs traditionnels. Bien que certaines méthodes ne soient pas toutes récentes (le gradient conjugué date des années 50), on les a considérées pendant longtemps comme des résultats mathématiques sans intérêt pratique pour la résolution de systèmes. Ce n'est que depuis quelques années, grâce aux progrès de l'informatique, que l'on peut espérer les utiliser avec profit.

3.3 Théorie des projections (Petrov-Galerkin)

3.3.1 Algorithme général

L'idée de toutes les méthodes itératives peut se résumer à une formulation très simple en terme de projections.
Imaginons que l'on veut approximer (projeter) le problème initial dans un certain sous-espace K de dimension m. Pour cela, on va imposer m contraintes qui vont définir un autre sous-espace L de dimension m. En général, on impose comme condition l'orthogonalité du résidu du système ($r = b - A\,x$) par rapport à ce sous-espace. La solution ne sera pas recherchée dans l'espace vectoriel K mais plutôt dans un espace analogue translaté de x_0, l'estimation initiale de la solution du système. Le problème devient :

$$\boxed{\text{Trouver } \tilde{x} \in x_0 + K \text{ tel que } b - A\,\tilde{x} \in L}$$

qui peut être reformulé sous la forme très générale (formulation de Petrov-Galerkin) :

$$\begin{cases} \tilde{x} = x_0 + \delta \quad \delta \in K \\ r_0 - A\,\delta \perp L \end{cases}$$

Remarque : pour une projection orthogonale, on choisit $L = K$ et les conditions précédentes sont appelées conditions de Galerkin.

Essayons de formuler ces conditions sous forme matricielle :
Choisissons les deux bases $V = [v_1, v_2, ..., v_m]$ et $W = [w_1, w_2, ..., w_m]$ respectivement pour K et L. La solution approchée peut s'écrire

$$x = x_0 + V\,y$$

On écrit ensuite la condition d'orthogonalité

$$W^T A V\,y = W^T r_0$$

En faisant l'hypothèse que la matrice $W^T AV$ n'est pas singulière, on obtient finalement une expression pour la solution approchée dans le sous espace K

$$\boxed{\tilde{x} = x_0 + V\left(W^T A V\right)^{-1} W^T\ r_0}$$

- Non singularité de la matrice projetée :

Dans les deux cas suivants, cette dernière hypothèse est justifiée :
– A est définie positive et $L = K$
– A est non singulière et $L = A\,K$
Autrement dit, il faudra éviter les projections orthogonales (méthode FOM par exemple) dans le cas général.

On peut montrer qu'utiliser une méthode de projection avec $L = K$ revient à minimiser la norme A de l'erreur (x_{approx} - x_{exact}) dans le sous espace K. Par contre, si on prend $L = A\,K$, on minimise la norme euclidienne du résidu.

- Algorithme général d'un schéma itératif pour la résolution de $A\,x = b$:

On peut alors représenter le schéma de ces méthodes de projection par l'algorithme suivant :

Algorithme 1 : Méthodes de projection

Jusqu'à convergence,
1. Sélection des sous-espaces K et L.
2. Choix (construction) des bases $V = [v_1, v_2, ..., v_m]$ et $W = [w_1, w_2, ..., w_m]$ pour K et L.
3. Calculer $\begin{cases} r := b - A\,x \\ y := \left(W^T A\,V\right)^{-1}\ W^T r \\ x := x + V\ y \end{cases}$

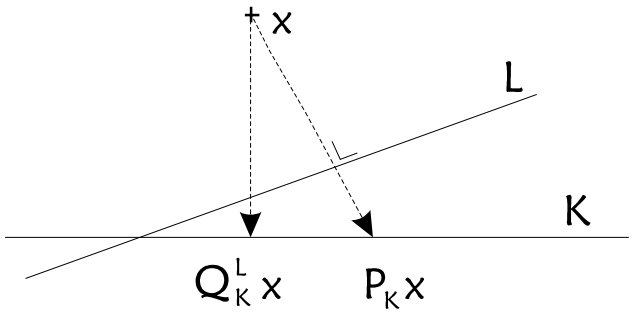
3.3.2 Interprétation en terme de projecteurs

Appelons P_K le projecteur orthogonal sur le sous-espace K et $Q_{K,L}$ le projecteur oblique sur le sous espace K orthogonalement à L. Il sont définis par

$$P_K\ x \in K, \qquad x - P_K\ x \perp K$$
$$Q_K^L\ x \in K, \qquad x - Q_K^L\ x \perp L$$

Le problème peut se présenter sous la forme

$$Q_K^L\left(b - A\,\tilde{x}\right) = 0$$



ou d'une manière équivalente, rappelant la formulation initiale :

$$A_m \, \tilde{x} = Q_K^L \, b \quad \text{avec} \quad x \in K \quad \text{et} \quad A_m = Q_K^L \, A \, P_K$$

Cette dernière équation montre bien que l'on a approché la résolution d'un système (*n x n*) par un système projeté (*m x m*) de dimension moindre.

3.3.3 Théorèmes de convergence des méthodes de projection

Dans cette section, nous avons regroupé les théorèmes fondamentaux qui permettent de mieux comprendre la manière avec laquelle va évoluer la solution approchée par une méthode de projection et, en particulier, la condition à réaliser pour obtenir la solution exacte (si tous les calculs étaient réalisés avec une précision infinie) en un nombre fini d'itérations.

- Théorème 1

Si *K* est invariant sous l'action de *A* et que *b* appartient à *K* alors, la solution approchée obtenue par une méthode de projection (orthogonale ou oblique) sur *K* est exacte.

- Théorème 2

Si *b* appartient à *K* et $x_0 = 0$ alors, la solution exacte x^* du problème original est telle que

$$\left\| b - A_m \, x^* \right\|_2 \le \gamma \left\| \left(I - P_K \right) x^* \right\|_2 \text{ avec } \gamma = \left\| Q_K^L \, A \left(I - P_K \right) \right\|_2$$

- Théorème 3

Si *A* est symétrique définie positive et $L = K$, $\tilde{x}$ étant le résultat d'une projection orthogonale sur *K* avec un vecteur initial x_0, alors $\tilde{x}$ minimise la norme *A* de l'erreur sur $x_0 + K$, c'est-à-dire,

$$E(\tilde{x}) = \min_{x \in x_0 + K} E(x) \text{ avec } E(x) = \left(A \left(x - x^* \right), \left(x - x^* \right) \right)^{1/2}$$

- Théorème 4

Si *A* est une matrice arbitraire non singulière, $L = A K$ et que $\tilde{x}$ est le résultat d'une projection oblique sur *K* orthogonalement à *L* avec un vecteur initial x_0, alors $\tilde{x}$ minimise la norme 2 du vecteur résidu $r = b - A x$ sur $x_0 + K$, c'est-à-dire,

$$R(\tilde{x}) = \min_{x \in x_0 + K} R(x) \text{ avec } R(x) = \left\| b - A \, x \right\|_2$$

Ce dernier théorème montre l'intérêt des méthodes basées sur des projections du type $L = A K$ puisque nous avons la garantie que le résidu diminue à chaque itération.

3.3.4 Projections à une dimension

Nous n'utiliserons pas ces types de projection parce qu'elles sont trop simples (on n'utilise pas l'information obtenue à une itération précédente) ; cependant, les méthodes telles que GMRES et CG ne sont que des généralisations de celles-ci où l'espace K sur lequel on projette gagne une dimension à chaque itération. La première itération sera donc inévitablement une projection unidimensionnelle.

Dans ce cas, on a $K = span\{d\}$ [6] et $L = span\{e\}$ où d et e sont deux vecteurs bien choisis. Le problème se pose sous la forme :

$$\begin{cases} x := x + \alpha\, d \\ r - A\, d \perp e \end{cases} \quad \Rightarrow \quad \alpha = (r,e) \big/ (A\, d, e)$$

Les méthodes qui en découlent sont :

Steepest descent
: A doit être symétrique et définie positive. $d = r$, $e = r$.

Minimal Residual
: A doit être définie positive $d = r$, $e = A\, r$.

Residual Norm Steepest Descent
: A doit être non singulière $d = A^T r$, $e = A\, r$.

3.4 Méthodes des espaces de Krylov

3.4.1 Définition

Les solveurs itératifs étudiés dans ce travail utilisent des méthodes de projection orthogonale et oblique sur des sous-espaces de Krylov. Ceux-ci sont définis par

$$K_m(A, r_0) = span\{r_0, A\, r_0, A^2\, r_0, \ldots, A^{m-1}\, r_0\} \text{ avec } r_0 = b - A\, x_0$$

avec $r_0 = b - A\, x_0$ le vecteur résidu initial du système (il est souvent égal à b puisque l'on prend en général $x_0 = 0$).

Ce sont donc des sous-espaces engendrés par des vecteurs de la forme $p(A)r_0$ où p est un polynôme de degré $m-1$. Ces méthodes approchent $A^{-1}b$ par $p(A)r_0$, où p est un 'bon' polynôme.

[6] On note *span*{a, b, c, ...}, l'espace formé par l'ensemble de toutes les combinaisons linéaires possibles des vecteurs a, b, c, ...

Les méthodes se différencient par le choix de L_m. Comme nous l'avons dit plus haut, les choix courants sont : $L_m = K_m$ et $L_m = A\,K_m$.

3.4.2 Propriétés

Dans la suite, nous aurons besoin de la définition suivante :

Le polynôme minimal d'un vecteur relatif à la matrice A est le polynôme non nul de plus bas degré tel que $p(A)v = 0$. Le degré de ce polynôme est appelé le **'grade' de *v* vis-à-vis de *A***. d'après Caley-Hamilton, le grade de *v* ne dépasse jamais *n*.

Les deux propositions suivantes sont immédiates :

- Théorème 1 :

Si μ est le grade de *r* alors K_μ est invariant sous l'action de *A* et $K_m = K_\mu \quad \forall\, m \geq \mu$

- Théorème 2 :

Le sous-espace de Krylov K_m est de dimension *m* si et seulement si le grade de *r* vis-à-vis de *A* est plus grand que *m-1*, c'est-à-dire,

$$\dim\left(K_m\right) = m \quad \Leftrightarrow \quad grade(r) \geq m$$
$$\dim\left(K_m\right) = \min\left\{m, grade(r)\right\}$$

En conséquence, on peut conclure qu'un solveur itératif fournira la solution du système en *N* itérations au maximum. Mais il est possible que celle-ci soit trouvée bien avant.
En pratique, les erreurs d'arrondis interviennent et cette borne supérieure théorique peut être dépassée.

Nous voyons aussi l'intérêt de rechercher des méthodes qui diminueront le grade de *r* puisqu'en pratique, il est peu souhaitable d'effectuer *N* itérations.

3.5 Orthogonalisation d'Arnoldi

L'algorithme d'Arnoldi est une méthode permettant de construire un sous-espace de Krylov par l'intermédiaire d'une base orthonormée. Le principe est simple : on part du résidu initial normé, les vecteurs de l'espace de Lanczos sont déduits les uns après les autres par multiplication par la matrice du système et orthonormés grâce à la procédure de Gram-Schmidt.

Nous utilisons ici une version modifiée, plus stable et moins sensible aux erreurs d'arrondis.
Une autre méthode consiste à utiliser l'algorithme de Householder qui provoque moins d'erreurs d'arrondis mais plus d'opérations (flops)

Algorithme d'Arnoldi - Orthogonalisation de Gram-Schmidt modifiée (MGS)

Choisir un vecteur v_1 de norme unitaire
DO $j = 1,m$
$w := A\, v_j$
    DO $i = 1,j$
        $h_{ij} = (w, v_i)$
        $w := w - h_{ij}\, v_i$
    ENDDO
    $h_{j+1,j} = \text{norm}(w_j)$
    IF $h_{j+1,j} = 0$ THEN STOP
    $v_{j+1} = w_j / h_{j+1,j}$
ENDDO

- Propriétés des vecteurs calculés :

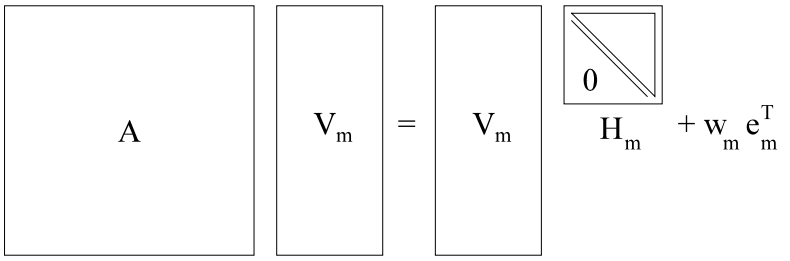
1. Supposons qu'au $m^{\text{ième}}$ pas, l'algorithme ne s'arrête pas, les vecteurs $v_1, v_2, ..., v_m$ forment une base orthonormée du sous-espace de Krylov

$$K_m = span\left\{v_1, A\, v_1, \ldots, A^{m-1}\, v_1\right\}$$

2. Si V_m est la matrice (n x m) dont les colonnes sont les vecteurs $v_1, v_2, ..., v_m$; $\overline{H}_m$, la matrice de Hessenberg définie par l'algorithme, et H_m la matrice carrée obtenue à partir de $\overline{H}_m$ en effaçant sa dernière ligne, on a les relations fondamentales suivantes :

$$\begin{aligned} A\, V_m &= V_m\, H_m + w_m\, e_m^T \\ &= V_{m+1}\, H_m \\ V_m^T A\, V_m &= H_m \end{aligned}$$

   Autrement dit, cet algorithme calcule la projection de la matrice A dans l'espace défini par les vecteurs de Krylov.



3. L'algorithme d'Arnoldi stoppe au pas *j* si et seulement si le polynôme minimum de v_1 et de degré *j*. Dans ce cas, le sous espace K_j est invariant. On obtient alors la solution exacte du système.

- Perte d'orthogonalité :

A cause des erreurs d'arrondis, les vecteurs *v* tendent à devenir de moins en moins orthogonaux. Il faut donc, dans des cas extrêmes, procéder à une réorthogonalisation des vecteurs. Pour ce faire, on peut suivre la démarche proposée par Pralett (double orthogonalisation) :
On calcule la norme du vecteur *w* et on la compare à celle de *A v*. Si la taille du vecteur a diminué trop fortement en taille lors de l'opération d'orthogonalisation, on effectue une seconde orthogonalisation. On peut montrer qu'une troisième est inutile.
Cette amélioration est utilisée pour le GMRES que nous développons dans la section suivante.

3.6 Méthode GMRES

3.6.1 Déduction de l'algorithme

La méthode du GMRES (Generalized Minimum Residual) est, comme son nom l'indique, une généralisation de la méthode unidimensionnelle du résidu minimum décrite brièvement dans une section antérieure. Cette méthode est sûrement la plus générale puisqu'elle ne fait aucune hypothèse sur la matrice du système mis à par sa non-singularité. De plus, elle est infaillible en théorie (en pratique sa puissance est limitée par la mémoire de la machine).

- Minimisation du résidu :

On choisit $K = K_m$ et $L = A K_m$ où K_m est le $m^{\text{ième}}$ sous-espace de Krylov.
Le but de cette méthode est de minimiser la norme du résidu sur tous les vecteurs x_0+K_m (voir les propriétés des projections de ce type).

Un vecteur quelconque de $x_0 + K_m$ peut s'écrire sous la forme

$$x = x_0 + V_m\, y$$

Détaillons l'expression de la norme du résidu en tenant compte de la forme de la solution approchée

$$J(y) = \|b - A\,x\|_2 = \|b - A\,(x_0 + V_m\,y)\|_2$$

On obtient alors successivement

$$\begin{aligned} J(y) &= \|r_0 - A\,V_m\,y\|_2 \\ &= \|\beta\,v_1 - V_{m+1}\,\overline{H}_m\,y\|_2 \\ &= \|\,V_{m+1}\,(\beta\,e_1 - \overline{H}_m\,y)\|_2 \\ &= \|\beta\,e_1 - \overline{H}_m\,y\|_2 \end{aligned}$$

La méthode revient donc à minimiser la fonction *J(y)*. La solution y_m de cette équation est assez simple à trouver puisqu'il s'agit de résoudre par les moindres carrés un système de dimension *(m+1) x m*.

Tout le problème est de résoudre ce système très rapidement. Pour cela, on peut exploiter la structure particulière de la matrice H_m. En effet, celle-ci a la forme d'une matrice triangulaire supérieure dans laquelle on a ajouté une diagonale sous la diagonale principale.

Un autre problème se pose pour la détermination de *m*, ce est-à-dire la dimension du sous-espace de recherche. En effet, tel qu'il est présenté plus haut, l'algorithme ne calcule la solution x_m qu'à la fin du processus d'orthogonalisation. Il est donc apparemment impossible d'obtenir une estimation de la norme du résidu lors du processus d'Arnoldi et donc de programmer un critère qui arrêterait la recherche de nouveaux v_m si cette norme est inférieure à une certaine valeur.

Une solution simple (mais peu efficace) serait de calculer la solution x_m à intervalles réguliers de manière à pouvoir calculer le résidu et tester un critère de convergence.

- <u>Factorisation QR du système projeté par rotations de Givens :</u>

La méthode suivante permet de remédier aux deux problèmes en résolvant le système surdéterminé à l'intérieur de la boucle d'Arnoldi. On transforme la matrice de Hessenberg en une matrice triangulaire supérieure en utilisant une succession de rotations planes (de Givens). La matrice de rotation se présente sous la forme :

$$\Omega_i = \begin{pmatrix} 1 & & & & & & \\ & \ldots & & & & & \\ & & 1 & & & & \\ & & & c_i & s_i & & \leftarrow \text{ligne i} \\ & & & -s_i & c_i & & \leftarrow \text{ligne i} + 1 \\ & & & & & 1 & \\ & & & & & & \ldots \\ & & & & & & 1 \end{pmatrix}$$

où $c_i^2 + s_i^2 = 1$.

Le but est d'éliminer $h_{i+1,i}$ en appliquant cette matrice au système.

On trouve, en effectuant le produit matriciel :

$$s_i = \frac{h_{i+1,i}}{\sqrt{\left|h_{i,i}\right|^2 + \left|h_{i+1,i}\right|^2}} \quad \text{et} \quad c_i = \frac{h_{i,i}}{\sqrt{\left|h_{i,i}\right|^2 + \left|h_{i+1,i}\right|^2}}$$

En définissant $Q_m = \Omega_m \, \Omega_{m-1} \ldots \Omega_1$

$$\overline{R}_m = \overline{H}_m^{(m)} = Q_m \, \overline{H}_m$$

$$\overline{g}_m = Q_m \left(\beta \, e_1\right) = \left(\gamma_{1,} \; \gamma_{2,} \ldots, \gamma_{m+1}\right)^T$$

on obtient un nouvelle formulation pour notre problème de minimum :

$$\boxed{\min \left\|\beta \, e_1 - \overline{H}_m y\right\|_2 = \min \left\|\overline{g}_m - \overline{R}_m y\right\|_2}$$

Ce dernier minimum peut être trouvé très facilement en résolvant le système triangulaire résultant en ignorant la dernière ligne de la matrice $\overline{R}_m$.

- Estimation du résidu :

Le résidu n'est rien d'autre que le dernier élément du vecteur $\overline{g}_m$, c'est-à-dire γ_{m+1}.
Plus précisément, on a

$$b - A \, x = V_{m+1} \, Q_m^T \left(\gamma_{m+1} \, e_{m+1}\right) \quad \text{et} \quad \left\|b - A \, x_m\right\|_2 = \left|\gamma_{m+1}\right|$$

Il est donc possible d'obtenir le résidu sans effectuer la multiplication matricielle ; ceci est très avantageux pour évaluer le critère d'arrêt rapidement. Cependant, il faudra faire attention aux erreurs d'arrondis et à la perte d'orthogonalité des vecteurs de Lanczos. La norme de la matrice *V* n'étant plus unitaire, le résidu obtenu par cette méthode risque d'être erroné.

- Echec de la méthode GMRES :

La seule cause d'échec possible se trouve dans la boucle d'Arnoldi, quand $h_{j+1,j} = 0$ à l'itération *j*. Cependant, on peut montrer que dans ce cas, le résidu devient nul et l'algorithme délivre la solution exacte à cette itération.

Nous sommes donc assurés d'obtenir la solution exacte après un maximum de *N* itérations quelle que soit la matrice non singulière considérée.

- Algorithme

L'algorithme de la méthode GMRES telle que présentée dans cette section n'est pas intéressant puisqu'inapplicable à de grands systèmes. En effet, la quantité de mémoire nécessaire pour stocker la matrice V_m (non creuse) augmente à chaque itération et devient très vite importante. De plus, l'orthogonalité des vecteurs *v* tend à se détruire par les erreurs d'arrondis successives. Il faut donc revoir la méthode pour l'adapter à la spécificité du système d'équations qui nous intéresse ici.

3.6.2 Variante de GMRES : Restarting GMRES

Cette variante effectue simplement une réinitialisation des vecteurs *v* après un nombre *m* d'itérations. Celui-ci peut être calculé par exemple pour tenir compte des limitations de l'ordinateur au point de vue mémoire mais aussi (et surtout) en fonction du nombre de conditionnement du système (plus le système est mal conditionné, plus le nombre *m* doit être pris grand).

Algorithme : GMRES(*m*)

1. x_0 donné, calculer $r_0 = b - A\,x_0$, $\beta = \|r_0\|_2$, $v_1 = r_0/\beta$
2. Génération de la base d'Arnoldi et de la matrice $\overline{H}_m$ en utilisant l'algorithme d'Arnoldi avec v_1 comme vecteur initial.
3. Calcul de y_m qui minimise $\|\beta\, e_1 - \overline{H}_m y\|_2$ et $x_m = x_0 + V_m\, y_m$
4. Calcul de la norme du résidu.
   Si le critère d'arrêt est satisfait, STOP, sinon, $x_0 := x_m$ et retour en 1.

Il subsiste cependant un problème : l'algorithme ainsi créé peut ne plus converger quand la matrice n'est pas définie positive (dans ce cas, le résidu stagne à une valeur déterminée). Pour des systèmes extrêmement mal conditionnés, on doit prendre $m = N$, ce qui est impraticable. C'est à ce niveau qu'interviennent les préconditionneurs.

3.6.3 Version tronquée (Quasi - GMRES)

- Principe :

L'idée est de garder en mémoire uniquement les *k* derniers vecteurs *v* dans un but, encore un fois, d'économie de mémoire.

Seul le point 2 de l'algorithme précédent est remplacé par une 'orthogonalisation incomplète', c'est-à-dire que pour calculer v_{m+1} lors du processus d'Arnoldi, on utilise uniquement les *k* derniers vecteurs calculés. Les autres points de l'algorithme restent inchangés.
La matrice $\overline{H}_m$ devient, dans le cas d'une orthogonalisation incomplète, une matrice bande dont la largeur de bande vaut $k + 1$.

Cette méthode n'a pas été implantée principalement pour trois raisons :
– la convergence n'est plus assurée pour une matrice quelconque (comme pour la version restart)
– l'implémentation est beaucoup plus compliquée que pour l'algorithme précédent.
– le résidu n'est plus calculé exactement.

Dans le paragraphe suivant, nous détaillons quand même la méthode utilisée pour mettre à jour la solution parce qu'elle sera utilisée aussi pour dériver le gradient conjugué dans le cas symétrique et défini positif.

- Modification de l'algorithme

Le seul problème est que nous avons de nouveau besoin de tous les vecteurs *v* lors du calcul final de la solution. Il faut donc calculer la solution à chaque étape du processus d'orthogonalisation.
Etudions le processus de mise à jour de la solution :

$$x_m = x_0 + V_m\ R_m^{-1}\ g_m = x_0 + P_m\ g_m \quad \text{avec}\ \ P_m = V_m\ R_m^{-1}$$

Si on appelle p_m la colonne m de la matrice P, il suffit alors d'effectuer à chaque itération

$$p_m = \frac{1}{h_{m,m}}\left(v_m - \sum_{i=m-k+1}^{m-1} h_{i,m}\ p_i\right) \qquad g_m = \begin{bmatrix} g_{m-1} \\ \gamma_m \end{bmatrix} = \begin{bmatrix} g_{m-1} \\ c_m\ \gamma_m^{(m-1)} \end{bmatrix}$$

La mise à jour de la solution prend alors la forme simple

$$x_m = x_{m-1} + \gamma_m\ p_m$$

Remarque : Cet algorithme fait donc une minimisation approchée puisque les vecteurs v sont uniquement orthogonaux si on en prend k consécutifs. Par conséquent, $|\gamma_{m+1}|$ n'est plus qu'une estimation du résidu. Plus précisément, on peut montrer que

$$\left\| b - A\ x_m \right\|_2 \le \sqrt{m+1}\ |\gamma_{m+1}|$$

3.6.4 Convergence

Les théorèmes donnant une indication sur la vitesse de convergence du GMRES en fonction des caractéristiques de la matrice sont peu nombreux. Ceux qui existent sont inapplicables en pratique et ils ont donc un intérêt purement théorique (les pertes d'orthogonalité et les erreurs d'arrondis ne sont pas prises en compte).

- Cas particulier des matrices définies positives :

Si A est une matrice définie positive, alors l'algorithme GMRES(m) converge quelque soit $m \ge 1$. A chaque itération, on a :

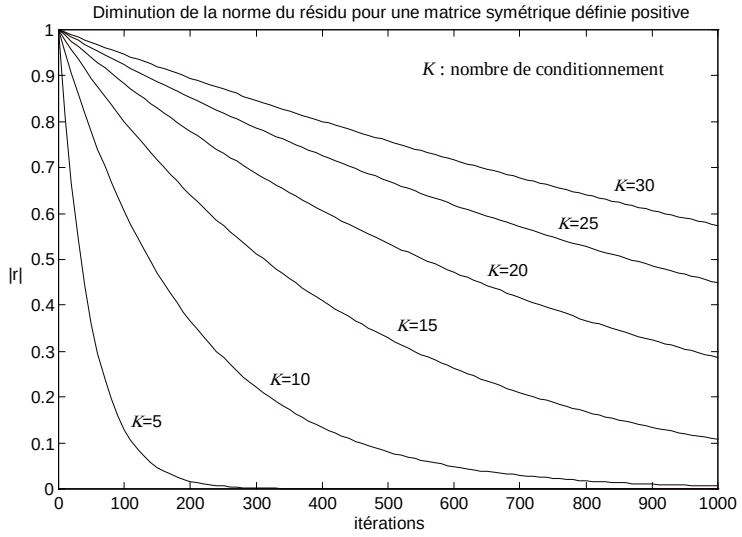
$$\left\| r_{k+1} \right\|_2 \le \sqrt{1 - \frac{\mu^2}{\sigma^2}}\ \left\| r_k \right\|_2 \qquad \text{avec} \quad \mu = \lambda_{\min}\left(\frac{A + A^T}{2}\right) \quad \text{et} \quad \sigma = \left\| A \right\|_2$$

Si la matrice est symétrique, on a :

$$\left\| r_{k+1} \right\|_2 \le \sqrt{\frac{\kappa^2 - 1}{\kappa^2}}\ \left\| r_k \right\|_2$$

où $\kappa = \dfrac{\lambda_{\max}(A)}{\lambda_{\min}(A)}$ est le nombre de conditionnement de la matrice du système.

En particulier, si le nombre de conditionnement est unitaire, la solution est trouvée en une itération par le GMRES.
Le graphique ci-dessous représente la diminution de la norme du résidu en fonction du nombre d'itération. Bien sûr, c'est une borne supérieure et donc souvent assez pessimiste.



- Cas général :

Il n'existe pas de relations aussi simple permettant de calculer la vitesse de convergence dans la cas non symétrique non défini positif. Cependant, nous vérifierons que le processus itératif sera d'autant plus rapide que la matrice du système est bien conditionnée.

3.7 L'algorithme de Lanczos (Arnoldi symétrique)

Cet algorithme n'est rien d'autre qu'une adaptation du processus d'Arnoldi dans le cas où la matrice *A* est symétrique. Des algorithmes simplifiés tels que le gradient conjugué (CG) ou SYMMLQ vont en découler. On utilise aussi couramment cet algorithme pour rechercher les valeurs propres d'une matrice symétrique.

Dans le cas d'une matrice *A* diagonale, remarquons que la matrice $\overline{H}_m$ ainsi créée est aussi symétrique et elle possède en plus la propriété très intéressante d'être tridiagonale. On la notera donc

$$T_m = \begin{pmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & \\ & \dots & \dots & \dots & \\ & & \beta_{m-1} & \alpha_{m-11} & \beta_m \\ & & & \beta_m & \alpha_m \end{pmatrix}$$

Cette structure particulière découle de l'existence d'une formule de récurrence pour construire les vecteurs successifs de la base de Lanczos :

$$v_{j+1} = A\, v_j - \alpha_j\, v_j - \beta_j\, v_{j-1}$$

où α_j et β_j sont déterminés pour obtenir l'orthogonalité des vecteurs calculés.

Autrement dit, nous n'avons besoin que des deux derniers vecteurs v_j calculés pour former le suivant, ce qui entraîne un gain d'espace mémoire très important (quelques vecteurs contre *m* pour le GMRES(*m*)).

Numériquement, on observe une dégradation assez rapide de l'orthogonalité des vecteurs formés et il faut recourir à des méthodes de réorthogonalisation selon certains critères. Cette solution nécessite le stockage d'un plus grand nombre de vecteurs et l'algorithme perd alors son attrait principal. L'orthogonalisation de Lanczos combiné avec une méthode de réorthogonalisation sélective ressemble alors à l'algorithme d'Arnoldi.
Pour METAFOR, nous acceptons la perte d'orthogonalité pour pouvoir gagner un espace mémoire maximum (avec le risque d'obtenir des temps de calculs légèrement plus longs).
La procédure de Lanczos s'écrit généralement de la manière suivante :

Algorithme : Orthogonalisation de Lanczos

DO $j = 1, ..., m$
  $w_j := A\, v_j - b_j\, v_{j-1}$
  $\alpha_j := (w_j, v_j)$
  $w_j := w_j - \alpha_j\, v_j$
  $\beta_{j+1} := \text{norm}(w_j)$
  $v_{j+1} := w_j / \beta_{j+1}$
ENDDO

- Algorithme de Lanczos :

De ce processus d'orthogonalisation, on peut déduire l'algorithme de Lanczos de la même manière que l'on peut déduire FOM du processus d'Arnoldi. La méthode Lanczos est donc une particularisation de FOM au cas symétrique. En conséquence, elle possède les mêmes inconvénients. Nous ne l'utiliserons donc pas pour METAFOR.

3.8 Gradient conjugué

On peut dériver la méthode du gradient conjugué de la manière suivante :
On essaye, comme pour le QGMRES, de modifier la formule de remise à jour de la solution pour ne pas devoir garder en mémoire les vecteurs *v*.

Posons $T_m = L_m U_m$. La mise à jour s'écrit :

$$x_m = x_0 + V_m U_m^{-1} L_m^{-1} \beta e_1$$

Si on appelle $\begin{cases} P_m = V_m U_m^{-1} \\ z_m = L_m^{-1} \beta e_1 \end{cases}$, on obtient $x_m = x_0 + P_m z_m = x_{m-1} + \zeta_m p_m$

Cette déduction nous apprend plusieurs choses :
- Le gradient conjugué est équivalent à la méthode FOM pour le cas non symétrique. La convergence est donc garantie uniquement dans le cas d'une matrice définie positive.
- Le gradient conjugué effectue implicitement une décomposition *LU* de la matrice tridiagonale. Lorsque la matrice est symétrique définie positive, l'existence de celle-ci (pas de pivots nuls) est toujours garantie. Nous allons voir qu'il existe d'autres méthodes de décomposition (SYMMLQ, par exemple) pour éviter cette restriction sur le type de la matrice du système.

- Propriétés :

1. $r_m = \sigma_m v_m$, en conséquence, les résidus sont orthogonaux entre eux.
2. Les vecteurs p_i sont conjugués dans la métrique de la matrice *A*, c'est-à-dire,

$$\left(A p_i, p_j\right) = 0 \text{ pour } i \neq j .$$

- Algorithme :

On peut déduire le gradient conjugué par une autre voie plus traditionnelle qui consiste à minimiser la fonction à *N* variables :

$$f(x) = \tfrac{1}{2} x^T A x - x^T b$$

où *A* est une matrice symétrique et définie positive.

Algorithme : le gradient conjugué (CG)

$r_0 = b - A\,x_0$, $p_0 := r_0$
DO $j = 0, 1, ...$
$\alpha_j := (r_j, r_j) / (A\,p_j, p_j)$
$x_{j+1} := x_j + \alpha_j\,p_j$
$r_{j+1} := r_j - \alpha_j\,A\,p_j$
$\beta_j := (r_{j+1}, r_{j+1}) / (r_j, r_j)$
$p_{j+1} := r_{j+1} + \beta_j\,p_j$
ENDDO

- Convergence

On peut montrer que

$$\left\| x_j - x^* \right\|_A \le 2\,\alpha^j \left\| x_0 - x^* \right\|_A, \qquad \alpha = \frac{\left(\sqrt{\kappa_2} - 1\right)}{\left(\sqrt{\kappa_2} + 1\right)}$$

où κ_2 est le nombre de conditionnement du système relatif à la norme 2 : $\kappa_2 = \frac{\lambda_{\max}}{\lambda_{\min}}$

Le gradient conjugué possède une convergence super-linéaire (taux de convergence qui augmente à chaque itération) lorsque les valeurs propres extrêmes de la matrice sont bien séparées.

3.9 Méthode SYMMLQ - MINRES

Cette méthode est basée sur le processus d'orthogonalisation de Lanczos et peut être vue comme une extension de l'algorithme du gradient conjugué pour des matrices symétriques mais non définies positives.

Il existe aussi une méthode, qui particularise le GMRES au cas non symétrique, appelée MINRES pour résoudre ce genre de problème. Celle-ci n'est qu'une variante de SYMMLQ.
La méthode utilisée est très simple, par contre, son implémentation est beaucoup plus ardue parce que les matrices qui interviennent dans le calcul ne sont jamais formées explicitement ; ce qui permet une économie de mémoire non négligeable.
Si la matrice est définie positive, on obtient les mêmes itérations que pour l'algorithme du gradient conjugué avec légèrement plus d'opérations par itération.

Contrairement au CG qui effectue implicitement une décomposition $L\,U$ de la matrice T_m (voir plus haut), la méthode SYMMLQ résout le système par une décomposition LQ où L est une triangulaire inférieure et Q une matrice orthogonale.

$$T_m = L_m\,Q_m$$

Grâce à cela, le problème des pivots nuls est évité et les systèmes non définis positifs peuvent entrer en jeu.

- Rotations planes :

Pour obtenir une approximation de la solution à chaque itération, il est nécessaire (comme pour le GMRES) de mettre à jour la factorisation LQ chaque fois que l'on introduit un nouveau vecteur dans la base. Cela s'effectue en appliquant une matrice de rotation plane du type.

$$Q = \begin{bmatrix} \cos\alpha & \sin\alpha \\ \sin\alpha & -\cos\alpha \end{bmatrix} \qquad Q^T\, Q = I \qquad \det Q = -1$$

Le produit de ces matrices donnerait finalement Q (on n'en a jamais besoin, elle ne sera donc jamais formée).
A l'itération m, on ajoute une colonne supplémentaire dans la matrice T_m.

$$T_m\, Q'_1\, Q'_2 \ldots Q'_m = T_m\, Q_m = \begin{bmatrix} * & & & & \\ * & \ddots & & & \\ * & \ddots & * & & \\ & \ddots & * & \overline{g} & \beta^* \\ & & * & \overline{d} & \alpha \\ & & & 0 & \beta \end{bmatrix} \underbrace{\begin{bmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & & c & s \\ & & & s & -c \end{bmatrix}}_{Q'_m} = L_m$$

Le but de l'opération est de rendre nul l'élément β*, la matrice résultante sera ainsi triangulaire inférieure. En effectuant le produit matriciel, on obtient les relations :

$$c = \overline{g}/\gamma \qquad s = \beta^*/\gamma \qquad \gamma = \sqrt{\overline{g}^2 + \beta^{*2}}$$

qui caractérisent la rotation plane et

$$\delta = c\,\overline{d} + s\,\alpha \qquad \overline{g}' = s\,\overline{d} - c\,\alpha$$
$$\varepsilon = s\,\beta \qquad \overline{d}' = -c\,\beta$$

qui sont les éléments du produit matriciel :

$$T_m\, Q_m = \begin{bmatrix} * & & & & \\ * & \ddots & & & \\ * & \ddots & * & & \\ & \ddots & * & \gamma & 0 \\ & & * & \delta & \overline{g}' \\ & & & \varepsilon & \overline{d}' \end{bmatrix} = L_m$$

- Mise à jour de la solution :

La formule permettant de calculer la solution approximée est toujours la même :

$$x_{m+1} = x_0 + V_m\, T_m^{-1} \left(\| r_0 \| e_1 \right)$$

Remplaçons la matrice triangulaire par sa factorisation.

$$x_{m+1} = x_0 + V_m \; Q_m \; L_m^{-1} \left(\| r_0 \| e_1 \right) = x_0 + V_m \; Q_m \; z_m$$

Contrairement à l'algorithme GMRES, la matrice Q est appliquée à la matrice V_m.
Imaginons que les m-1 premières rotations ont été effectuées successivement lors des itérations précédentes. Il faut maintenant appliquer la $m^{\text{ième}}$ rotation à la matrice résultante. Cette matrice peut se diviser en 3 parties :

$$x_{m+1} = x_0 + \underbrace{\left[V'_m \quad w_m \quad v_m \right] Q'_m}_{\textit{traité séparement}} \; \underbrace{L_m^{-1} \left(\| r_0 \| e_1 \right)}_{\textit{système à résoudre}}$$

Une première appelée V'_m ne sera pas modifiée par la rotation. Les deux autres sont des vecteurs qui seront modifiés (v_m est le nouveau vecteur de Lanczos et w_m est le vecteur de Lanczos de l'itération précédente, modifié par la dernière rotation). On cherche donc comment va se transformer la dernière colonne de cette matrice (c'est-à-dire la formule de récurrence de w_m). La formule suivante donne la réponse :

$$\left[V'_m \quad w_m \quad v_m \right] Q'_m = \left[V'_m \quad c \, w_m + s \, v_m \quad s \, w_m - c \, v_m \right]$$

D'un autre côté, il faut résoudre le système projeté. On se donne une solution de la forme [z 0] et on transforme le système (surdéterminé) en un système équivalent :

$$\begin{bmatrix} \gamma & 0 \\ \delta & \overline{g} \\ \varepsilon & \overline{d} \end{bmatrix} \begin{bmatrix} z \\ 0 \end{bmatrix} = \begin{bmatrix} RHS1 \\ 0 \\ 0 \end{bmatrix} \quad \leftrightarrow \quad \begin{bmatrix} \gamma & 0 \\ 0 & \overline{g} \\ 0 & \overline{d} \end{bmatrix} \begin{bmatrix} z \\ 0 \end{bmatrix} = \begin{bmatrix} RHS1 \\ -z \, \delta + RHS1 \\ -z \, \varepsilon \end{bmatrix}$$

et on trouve $z = RHS1/\gamma$ avec *RHS1*, la norme du résidu initial.
Pour les itérations suivantes, on effectue la même transformation pour obtenir :

$$\begin{bmatrix} * & & & & \\ & \ddots & & & \\ & & * & & \\ & & & \gamma & \\ & & & \delta & \overline{g} \\ & & & \varepsilon & \overline{d} \end{bmatrix} \begin{bmatrix} [z_m] \\ \\ z \\ 0 \end{bmatrix} = \begin{bmatrix} [RHS_m] \\ \\ RHS1 \\ 0 \end{bmatrix} \quad \leftrightarrow \quad \begin{bmatrix} * & & & & \\ & \ddots & & & \\ & & * & & \\ & & & \gamma & \\ & & & 0 & \overline{g} \\ & & & 0 & \overline{d} \end{bmatrix} \begin{bmatrix} [z_m] \\ \\ z \\ 0 \end{bmatrix} = \begin{bmatrix} [RHS_m] \\ \\ -z \, \delta + RHS1 \\ -z \, \varepsilon \end{bmatrix}$$

Finalement, la formule de mise à jour s'écrit :

$$x_{m+1} = \underbrace{x_0 + V'_m \; z_m}_{x_m} + \left(w_m \; c + v_m \; s \right) z$$

Il est possible de retomber sur les itérations du gradient conjugué lorsque la matrice est symétrique définie positive en effectuant le calcul supplémentaire [P1] :

$$x_m^{CG} = x_m^{LQ} + \lambda_m \; v_1$$

$\lambda_0 = 0$, $\quad \lambda_m = \lambda_{m-1} + z \; c_m \prod_{i=1}^{m} s_i$ (où s_i est le sinus de l'itération i)

3.10 Algorithme de Lanczos non symétrique

- Algorithme :

Nous abordons maintenant un autre type de méthodes de résolution de systèmes basées sur des projections différentes. Ces techniques vont nous permettre de traiter des systèmes non symétriques en utilisant une quantité de mémoire réduite.

Malheureusement, d'autres problèmes apparaissent et les solutions apportées sont très compliquées à mettre en œuvre. On peut prouver mathématiquement qu'il est impossible de mettre au point une méthode de projection infaillible qui permettrait de traiter des matrices quelconques avec une mémoire réduite. Il faut donc choisir : soit la robustesse, soit l'économie de mémoire.

La base des méthodes présentées dans cette section est une extension du processus d'orthogonalisation de Lanczos au cas non symétrique.

On construit deux sous espaces de Lanczos par l'intermédiaire de deux bases biorthonormées appartenant respectivement aux deux sous-espaces de Krylov suivants

$$K_m(A, v_1) = span\{v_1, A\,v_1, A^2\,v_1, \ldots, A^{m-1}\,v_1\}$$
$$K_m(A^T, w_1) = span\{w_1, (A^T)\,w_1, (A^T)^2\,w_1, \ldots, (A^T)^{m-1}\,w_1\}$$

Les solveurs qui vont suivre sont étroitement liées aux projections obliques. En effet, on pourra, dans la suite, projeter le problème dans le premier espace engendré par la matrice A à l'aide du deuxième espace engendré par la transposée A^T. Il faudra cependant vérifier à tout moment que cet opérateur de projection existe.

Algorithme : Orthogonalisation de Lanczos Symétrique

Choix des vecteurs v_1 et w_1 tel que $(v_1, w_1) = 1$,
Poser $\beta_1 = 0$, $w_0 = v_0 = 0$
DO $j = 1, m$

$\alpha_j := (A\,v_j, w_j)$
$v_{j+1} := A\,v_j - \alpha_j\,v_j - \beta_j\,v_{j-1}$
$w_{j+1} := A^T w_j - \alpha_j\,w_j - \delta_j\,w_{j-1}$
$\delta_{j+1} = \left|(v_{j+1}, w_{j+1})\right|^{1/2}$
$\beta_{j+1} = (v_{j+1}, w_{j+1}) / \delta_{j+1}$
$w_{j+1} := w_{j+1} / b_{j+1}$
$v_{j+1} := v_{j+1} / d_{j+1}$

ENDDO

La matrice H_m est tridiagonale et peut s'écrire

$$T_m = \begin{pmatrix} \alpha_1 & \beta_2 & & & \\ \delta_2 & \alpha_2 & \beta_3 & & \\ & \cdots & \cdots & \cdots & \\ & & \delta_{m-1} & \alpha_{m-11} & \beta_m \\ & & & \delta_m & \alpha_m \end{pmatrix}$$

- Propriétés :

Enumérons les propriétés des matrices ainsi créées. Elles sont semblables aux relations définissant l'orthogonalisation d'Arnoldi

$$\left(v_j, w_i\right) = \delta_{i,j} \quad 1 \le i, j \le m$$
$$A V_m = V_m T_m + \delta_{m+1} v_{m+1} e_m^T$$
$$A^T W_m = W_m T_m^T + \beta_{m+1} w_{m+1} e_m^T$$
$$W_m^T A V_m = T_m$$

Autrement dit, la matrice T_m est obtenue par une projection oblique sur $K_m(A, v_1)$ orthogonalement à $K_m(A^T, w_1)$ puisque la dernière relation est similaire à cette trouvée pourexprimer la projection d'une matrice (voir section 3.3)

- Défaillances du processus - look ahead :

Comme nous allons le voir, l'économie de mémoire se fait au détriment de la robustesse. L'algorithme provoque une division par zéro lorsque la relation suivante est vérifiée :

$$\left(v_{j+1}, w_{j+1}\right) = 0,$$

ce qui signifie, concrètement, que le projecteur n'existe pas (la direction de projection est parallèle à l'espace sur lequel on veut projeter la matrice).

Y. Saad distingue

- le 'lucky breakdown' : $v_{j+1} = 0$ et $w_{j+1} = 0$
- le 'serious breakdown' : $v_{j+1} \neq 0$ et $w_{j+1} \neq 0$

Dans le cas du 'lucky breakdown', la situation est la même que celle discutée pour le processus d'Arnoldi : l'espace de Krylov ne peut plus être agrandi et le théorème de convergence des méthodes de projection nous garantit que la solution exacte est trouvée en même temps.
Par contre, le deuxième cas est bien plus ennuyeux puisqu'il est totalement imprévisible. Il ne dépend pas des propriétés de la matrice (par exemple, le nombre de conditionnement) mais plutôt des vecteurs de départ choisis pour engendrer les deux espaces.

D'habitude, si on rencontre un cas pathologique, on y remédie en utilisant la stratégie 'look ahead' présentée par [N1], c'est-à-dire que l'on passe à l'itération suivante en sautant la paire

de vecteurs à problème et on continue l'algorithme comme si de rien n'était. La matrice T_m devient tridiagonale par blocs. Cependant, rien ne garantit que les vecteurs suivants ne seront pas, eux aussi, orthogonaux. Il existe même des cas où il est impossible de trouver deux vecteurs pour continuer les itérations.

Pour ce travail, nous n'avons jamais utilisé cette stratégie pour plusieurs raisons :
1. La première est fort terre à terre : le look ahead est extrêmement compliqué à mettre en œuvre. La programmation nécessite de travailler avec des matrices formées de blocs de taille variables et énormément de pointeurs[7].
2. La mémoire nécessaire au bon fonctionnement du processus devient totalement imprévisible puisque le problème ne dépend pas directement de la matrice du système.
3. En pratique, le produit scalaire n'est jamais nul. Il faut cependant éviter une division par un nombre trop petit pour ne pas perdre l'orthogonalité des vecteurs. Il faut donc utiliser un critère (lequel ?) pour choisir si, lors de l'introduction d'un nouveau vecteur dans les deux bases, on utilise la version avec ou sans look ahead.

- Perte d'orthogonalité :

Les vecteurs *v* et *w* tendent à perdre leur propriété fondamentale (bi-orthogonaux). Comme nous l'avons vu, ce problème existe aussi pour le processus d'Arnoldi mais il est beaucoup plus important ici puisqu'on utilise une brève formule de récurrence pour calculer chaque nouveau vecteur. Un remède est d'effectuer une nouvelle orthogonalisation mais c'est difficilement praticable si on veut garder une quantité de mémoire utilisée limitée..

- Temps de calcul du processus :

L'algorithme de Lanczos utilise deux produits matrice - vecteur (opération la plus coûteuse) contrairement à Arnoldi qui n'en utilise qu'un. C'est un peu comme si on résolvait deux systèmes simultanément (un avec *A*, l'autre avec A^T). Par contre, l'introduction d'un vecteur dans la base ne nécessite pas une orthogonalisation avec tous les autres grâce à la formule de récurrence. En conséquence, pour engendrer des petits espaces (quelques dizaines de vecteurs), le processus d'Arnoldi sera plus rapide que Lanczos.

Vu les nombreux problèmes décrits ci-dessus, nous ne développerons que les méthodes les plus simples pour METAFOR (Bi-CG, Bi-CGSTAB, CGS). Ces solveurs pourront être mis en parallèle avec le GMRES et nous verrons si le gain de mémoire se justifie face à la perte potentielle de robustesse et de rapidité.

A l'heure actuelle, de nombreuses personnes commencent à s'intéresser de plus près à ce type de solveurs et on peut espérer, prochainement, la découverte de nouvelles techniques comblant les déficiences de la méthode de Lanczos (le look ahead est déjà un grand pas en avant).

[7] Il existe des routines tels que QMRPACK [N2] qui utilisent le 'look ahead' combiné avec une méthode QMR mais nous nous sommes interdit d'utiliser de telles routines en temps que boîtes noires.

3.10.1 Méthode des gradients bi-conjugués (Bi-CG)

- Introduction :

Cette méthode est dérivée du processus de Lanczos non symétrique comme on a dérivé le gradient conjugué du processus de Lanczos symétrique.

Pour déduire l'algorithme, on va, encore une fois, modifier la formule de mise à jour de x_m.
On effectue tout d'abord une décomposition de Choleski de la matrice T_m.

$$T_m = L_{w,m}\ D_m\ L_{v,m}^T$$

Des méthodes différentes peuvent être déduites en effectuant d'autres factorisations (par exemple, QMR utilise une factorisation QR).

On définit ensuite la matrice $P_m = V_m\ L_{v,m}^{-T}$
La formule de mise à jour devient dans ce cas précis :

$$\begin{aligned} x_m &= x_0 + V_m\ T_m^{-1}\left(\beta\, e_1\right) \\ &= x_0 + V_m\ L_{v,m}^{-T}\ D_m^{-1}\ L_{w,m}^{-1}\left(\beta\, e_1\right) \\ &= x_0 + P_m\ D_m^{-1}\ L_{w,m}^{-1}\left(\beta\, e_1\right) \end{aligned}$$

- Remarques :

  - On utilise la méthode habituelle pour déduire la solution x_m à partir de x_{m-1} (voir à ce sujet QGMRES).
  - Les vecteurs r_j et r_j* ont la même direction que v_{j+1} et w_{j+1} ; ils sont donc bi-conjugués également.

- Propriétés :

  - $\left(P_m^*\right)^T A\ P_m = D_m$, d'où le nom 'gradients bi-conjugués'.
  - $\left(r_j, r_i^*\right) = 0$ et $\left(A\ p_j, p_i^*\right) = 0 \quad \forall\ i \neq j$
  - On constate habituellement une convergence irrégulière vers la solution (en effet, on n'a pas, comme pour le GMRES, une propriété de minimisation du résidu après chaque itération.

Cette méthode à l'air intéressante pour METAFOR et nous la testerons dans la suite.

Algorithme : Bi-CG

$r_0 = b - A\,x_0$; r_0^* arbitraire (p.expl $r_0^* = r_0$)
$p_0 = r_0$; $p_0^* = r_0^*$
DO $j = 1,...$
$\alpha_j = (r_j, r_j^*)/(A\,p_j, p_j^*)$
$x_{j+1} = x_j + \alpha_j\,p_j$
$r_{j+1} = r_j - \alpha_j\,A\,p_j$
$r_{j+1}^* = r_j^* - \alpha_j\,A^T\,p_j^*$
$\beta_j = (r_{j+1}, r_{j+1}^*) / (r_j, r_j^*)$
$p_{j+1} = r_{j+1} + \beta_j\,p_j$
$p_{j+1}^* = r_{j+1}^* + \beta_j\,p_j^*$
IF (convergence) THEN STOP
ENDDO

3.10.2 Conjugate Gradient Squared (CGS)

- Déduction de l'algorithme :

Pour déduire cette méthode, on cherche à former des itérations relatives non pas au résidu mais au carré de celui-ci. On cherche, bien sûr, à accélérer la convergence.

Ecrivons le résidu sous la forme :

$$r_i = \phi_i(A)\,r_0 \qquad \text{où } \phi i(A) \text{ est appelé 'polynôme résiduel'} \quad (\phi_i(0) = 1)$$

On cherche donc à écrire des itérations sur $r_i' = \phi_i^2(A)\,r_0$.

Posons, d'une manière analogue

$$p_i = \pi_i(A)\,p_0$$

Recalculons α_i

$$\alpha_i = \frac{\left(\phi_i(A)\,r_0, \phi_i\left(A^T\right) r_0\right)}{\left(A\,\pi_i(A)\,r_0, \pi_i\left(A^T\right) r_0\right)} = \frac{\left(\phi_i^2(A)\,r_0, r_0\right)}{\left(A\,\pi_i^2(A)\,r_0, r_0\right)}$$

On voit directement que la transposée n'apparaît plus explicitement.
Les itérations du gradient bi-conjugué peuvent s'écrire en terme de polynômes ϕ et π (d'après les formules de mise à jour de r et p) :

$$\phi_{j+1}(t) = \phi_j(t) - \alpha_j\,t\,\pi_j(t)$$
$$\pi_j(t) = \phi_j(t) + \beta_{j-1}\,\pi_{j-1}(t)$$

Posons maintenant

$$r_j = \phi_{j+1}^2(A)\, r_0$$
$$p_j = \pi_j^2(A)\, r_0$$
$$q_j = \phi_{j+1}\, \pi_j(A)\, r_0$$

Les itérations recherchées s'écrivent, en termes des nouveaux vecteurs

$$\begin{cases} r_{j+1} = r_j - \alpha_j\, A\left(2\, r_j + 2\, \beta_{j-1}\, q_{j-1} - \alpha_j\, A\, p_j\right) \\ p_j = r_j + 2\, \beta_{j-1}\, q_{j-1} + \beta_j\, p_{j-1} \\ q_j = r_j + \beta_{j-1}\, q_{j-1} - \alpha_j\, A\, p_j \end{cases}$$

- Remarques :

  - Les erreurs d'arrondis sont encore la cause de problèmes liés à la perte d'orthogonalité des vecteurs calculés.
  - Vu que tous les calculs se réfèrent au carré de la norme du résidu, on peut rencontrer un overflow (on converge deux fois plus vite mais on diverge deux fois plus vite aussi !).
  - Cette méthode est parfois utilisée lorsque A^T n'est pas disponible.

Algorithme : CGS

$r_0 = b - A\, x_0$; r_0^* arbitraire (par exemple $r_0^* = r_0$)
$p_0 = r_0$; $u_0 = r_0$
DO j = 1,...
  $v = A\, p_j$
  $\alpha_j = (r_{j-1}, r_0) / (v, r_0)$
  $q_j = u_{j-1} - \alpha_j\, v$
  $x_j = x_{j-1} + \alpha_j\, (u_{j-1} + q_j)$
  $r_j = r_{j-1} - \alpha_j A\, (u_{j-1} + q_j)$
  $\beta = (r_j, r_0) / (r_{j-1}, r_0)$
  $u_j = r_j + \beta\, q_j$
  $p_j = u_j + \beta\, (q_j + \beta\, p_{j-1})$
  IF (convergence) THEN STOP
ENDDO

Cet algorithme sera utilisé pour les tests du chapitre 6. Il sera intéressant de voir si les prévisions théoriques peuvent être retrouvées lors de la résolution de problèmes dans METAFOR.

3.10.3 Bi-CGSTAB

On peut appliquer une méthode similaire à celle employée ci-dessus pour le CGS dans le but d'améliorer l'algorithme du Bi-CG, c'est-à-dire éliminer sa convergence irrégulière caractéristique.
On cherche à formuler de nouvelles itérations pour l'expression

$$r_i' = \psi_i(A)\,\phi_i(A)\,r_0$$

où $\psi_i(A)$ est un polynôme ayant pour but de stabiliser ou lisser la convergence et
$\phi_i(A)$ est le polynôme du résidu.

Des calculs analogues au CGS peuvent être menés pour dériver une méthode (Bi-CG STABilised) qui donne généralement de meilleurs résultats que le simple Bi-CG.

Pratiquement, on choisit la relation de récurrence suivante

$$\psi_{j+1}(t) = \left(1 - \omega_j\, t\right)\psi_j(t)$$

où ω_j est un paramètre additionnel déterminé pour minimiser le résidu.

Algorithme : Bi-CGSTAB

$r_0 = b - A\,x_0$; r_0^* arbitraire (par exemple $r_0^* = r_0$)
$p_0 = r_0$;
DO j = 1,...
$\alpha_j = (r_j, r_0^*) / (A\,p_j, r_0^*)$
$sj = rj - \alpha j\, A\, pj$
$\omega_j = (A\,s_j, s_j) / (A\,s_j, A\,s_j)$
$r_{j+1} = s_j - \omega_j\, A\, s_j$
$\beta_j = (r_{j+1}, r_0^*) / (r_j, r_0^*) * \alpha_j / \omega_j$
$p_{j+1} = r_{j+1} + \beta_j\,(p_j - \omega_j\, A\, p_j)$
$x_{j+1} = x_j + \alpha_j\, p_{j+1} + \omega_j\, s_j$
IF (convergence) THEN STOP
ENDDO

3.10.4 QMR : Quasi-Minimal Residual

La méthode QMR utilise une factorisation QR pour résoudre le système projeté au sens des moindres carrés. La démarche est la même que pour le GMRES. Cependant, la solution à chaque itération ne minimise plus le résidu dans le sous-espace de Krylov (cette propriété est valable uniquement pour $L = A\,K$). On considère que c'est un 'quasi-minimum'. C'est de là que vient le nom de la méthode.

Cette méthode possède des propriétés de convergence plus intéressantes que le Bi-CG et tend à converger parfois aussi vite que le GMRES.

Malgré cela, nous n'utiliserons pas cet algorithme dans METAFOR parce qu'il est plus difficile à programmer que les autres de la famille de Lanczos et que, comme eux, il est sujet à des défaillances provenant du processus d'orthogonalisation.

3.11 Méthodes relatives aux équations normales

Le but recherché dans cette section est de transformer un problème non symétrique en problème symétrique défini positif, vu la simplicité et la puissance des méthodes pour résoudre ce type de problème. Cependant, nous verrons que cette approche rend le système très mal conditionné. Elle ne pourrait donc être utilisée que dans des cas très précis (système bien conditionné).

3.11.1 Les équations normales

Il existe principalement deux formulations couramment utilisées en pratique :

- Normal Residual (NR)

On multiplie les deux membres de l'équation matricielle par A^T.

$$A^T\,A\,x = A^T\,b$$

Cette façon d'aborder le système d'équation est équivalente au problème de minimisation de la fonction

$$f(x) = \tfrac{1}{2}\left\|b - A\,x\right\|_2$$

- Normal Error (NE)

Le système initial est transformé en introduisant un vecteur auxiliaire y.

$$A\,A^T\,y = b\,, \qquad x = A^T\,y$$

La fonction associée à minimiser s'écrit

$$f(y) = \tfrac{1}{2}\left\|x^* - A^T\,y\right\|_2$$

D'autres formulations existent (passage à un système à 2*n* inconnues par exemple) mais ne sont pas considérées par la suite.
Le système résultant est d'habitude traité par gradients conjugués (d'où les méthodes CGNE et CGNR)

3.11.2 Problème majeur de la méthode

Calculons le nombre de conditionnement en norme 2 du nouveau système relatif aux équations normales :

$$\mathrm{Cond}_2\left(A^T\ A\right) = \|A\|_2^2 \left\|A^{-1}\right\|_2^2 = \mathrm{Cond}_2^2\left(A\right)$$

Nous voyons que si le système initial est déjà très mal conditionné, la nouvelle formulation ne fait qu'empirer les choses. Le progrès fait sur un pas est détruit pas le bruit dû aux erreurs numériques.

En pratique, les auteurs [P1,S1,V1] déconseillent d'utiliser ces méthodes sauf si on est certain que l'on peut y gagner quelque chose. Dans notre cas, nous n'avons aucune information sur le nombre de conditionnement de la matrice de raideur (celui-ci dépend de beaucoup de paramètres) et nous utiliserons donc les solveurs des chapitres précédents.

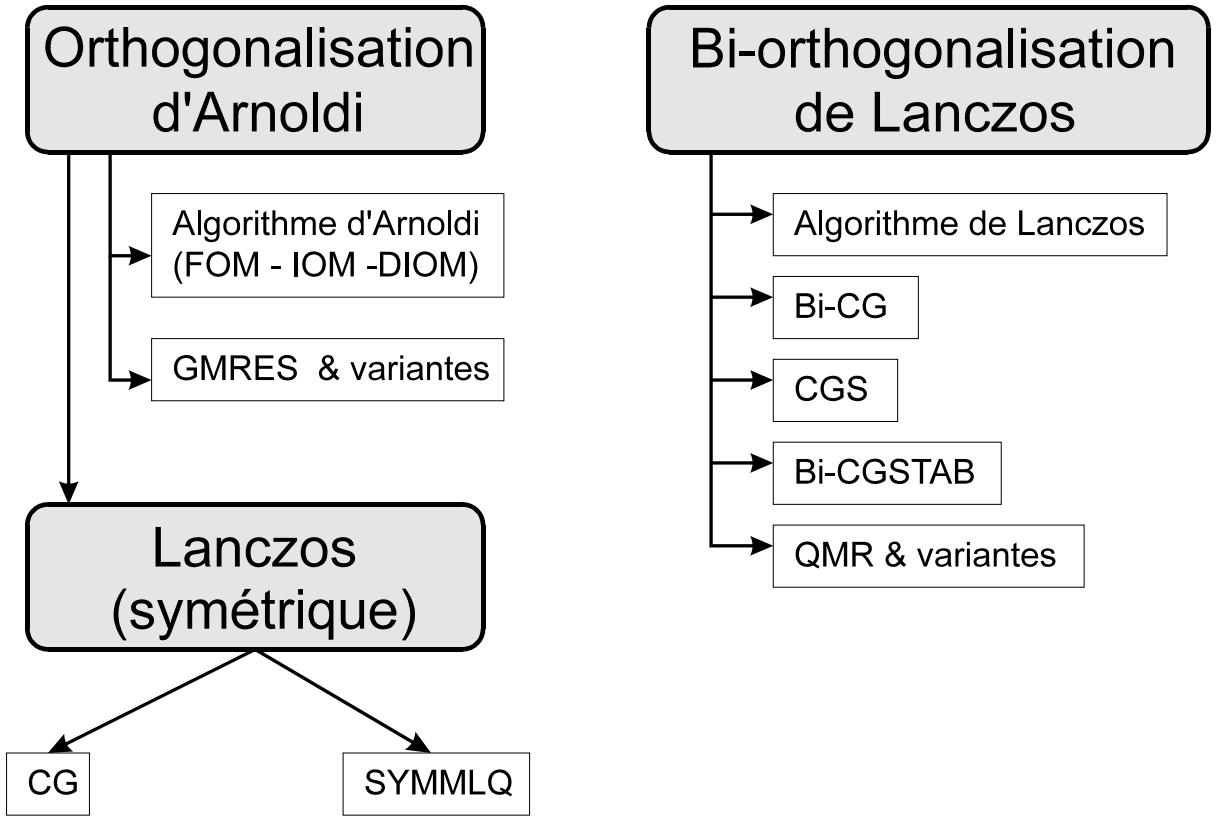
3.12 Résumé des différentes méthodes itératives

Résumons rapidement les méthodes itératives choisies :

Méthode	Avantages	Inconvénients
GMRES(m)	Applicable dans tous les cas Propriété de minimisation du résidu à chaque itération.	Nécessite le stockage de m vecteurs de la taille du système.
Bi-CG	Applicable dans tous les cas. Utilise peu de mémoire (10 vecteurs).	Breakdown possible. Convergence irrégulière.
Bi-CGSTAB	Applicable dans tous les cas. Utilise peu de mémoire (10 vecteurs). Convergence stabilisée.	Breakdown possible.
CGS	Applicable dans tous les cas . Utilise peu de mémoire (11 vecteurs). Converge 2 fois plus vite que Bi-CG.	Convergence très irrégulière. Breakdown possible. Diverge 2 fois plus vite Bi-CG
SYMMLQ	Utilise peu de mémoire (1/2 matrice).	Matrice symétrique. Préconditionneur = matrice M. (*)
CG	Utilise peu de mémoire (1/2 matrice).	Matrice symétrique et définie positive. Préconditionneur = matrice M.(*)

(*) voir chapitre 4

Le tableau ci-dessous rassemble les méthodes et indique clairement d'où elles sont issues.



3.13 Critères d'arrêt

- Introduction :

Pour obtenir de bons résultats, il est essentiel de sélectionner un bon critère d'arrêt adapté au type de problème envisagé.

Idéalement, il faudrait utiliser la formule

$$\left\| e^{(i)} \right\| = \left\| x^{(i)} - x \right\| \leq tol$$

mais l'erreur n'est pas calculable puisqu'on ne connaît pas la solution exacte de problème. Il est donc utile de remplacer celle-ci par la norme du résidu du système. On peut trouver une relation entre ces deux valeurs :

$$r^{(i)} = b - A\,x^{(i)} = \underbrace{b - A\,x}_{0} - A\,x^{(i)} + A\,x = -A\,e^{(i)} \quad \rightarrow \left\| e^{(i)} \right\| = \left\| A^{-1} \right\| \left\| r^{(i)} \right\|$$

où, bien sûr, la norme de l'inverse de la matrice du système est inconnue.

Il existe plusieurs critères possibles relatifs à la norme du résidu. Citons par exemple

1. $\left\|r^{(i)}\right\| \leq \varepsilon \left(\|A\| \left\|x^{(i)}\right\| + \|b\|\right)$: la norme de x n'est pas toujours calculable (voir GMRES).
2. $\left\|r^{(i)}\right\| \leq \varepsilon \|b\|$: le plus couramment utilisé.
3. $\left\|r^{(i)}\right\| \leq \varepsilon \left\|r^{(0)}\right\|$: se ramène à 2. Si $x_0 = 0$.
4. ...

- Avantage de l'utilisation de 2 :

Si on se rappelle que le second membre représente le déséquilibre des forces dans la structure à l'itération de Newton-Raphson considérée, l'utilisation du deuxième critère est intéressante. En effet, lorsqu'on est loin de l'équilibre (début d'un pas de temps), la norme du résidu d'équilibre est élevé et on est amené par ce critère à résoudre le système de manière assez imprécise. Ce n'est pas grave, puisqu'on n'a pas besoin d'une grande précision sur x. On profite donc du fait que la résolution du système fait partie d'un algorithme de Newton-Raphson qui converge généralement bien même si on lui fournit une solution approchée.
Par contre, lors des dernières itérations de Newton-Raphson, on veut une précision maximale puisque l'on se rapproche d'une configuration qui sera considérée équilibrée et le point de départ d'un nouveau pas de temps.

- Nombre d'itérations maximum autorisé :

Parallèlement à la précision e, nous utiliserons un paramètre it_{max} qui fixera le nombre maximum d'itérations du solveur. Si celui-ci est atteint, on donne à l'algorithme de Newton-Raphson la solution trouvée à la dernière itération et on espère que celle-ci sera assez précise pour permettre une diminution de la norme du résidu d'équilibre.
Le critère d'arrêt peut donc être modifié par l'utilisateur grâce aux paramètres ε et it_{max}. Il se résume aux deux relations suivantes :

$$\left\|r^{(i)}\right\| \leq \varepsilon \|b\|$$
$$it \leq it_{\max}$$

Il est important de comprendre que la précision ε peut ne pas être atteinte (si le nombre d'itérations maximum du solveur est atteint).

4. Préconditionnement du système

4.1 Introduction

Préconditionner un système d'équations, c'est simplement le transformer en un autre qui possède la même solution mais qui est plus simple à résoudre par des méthodes itératives, c'est-à-dire qui mettra moins d'itérations pour converger vers la solution. Le préconditionneur agit sur le spectre de la matrice *A* pour réduire au maximum son extension. En d'autres termes, il essaye de diminuer le nombre de conditionnement du système.

Un préconditionneur particulier peut être fantastique pour résoudre un type de problème donné et avoir des conséquences désastreuses pour un autre type de problème. De plus, il n'existe pas encore de théorèmes généraux garantissant le bon fonctionnement d'un préconditionneur en pratique. Chaque théoricien a ses propres convictions et on trouve rarement deux auteurs qui présentent une même méthode comme méthode optimale.

Les techniques de préconditionnement ont pour but de permettre aux solveurs itératifs d'atteindre l'efficacité et la robustesse des solveurs directs tout en diminuant considérablement le coût du calcul.

Dans un premier temps, nous nous intéresserons à la manière de prendre en compte le préconditionneur dans les algorithmes décrits au chapitre précédent.
Ensuite, nous verrons en détail les principales méthodes existantes pour créer un préconditionneur efficace à partir de la matrice du système lorsque celle-ci est non symétrique et non définie positive en général. L'efficacité est très importante parce que le calcul du préconditionneur peut demander autant de temps que la résolution du système préconditionné

par le solveur itératif. C'est le temps total (préconditionnement + résolution) qui devra être comparé au temps de calcul du solveur direct actuel.
Nous particulariserons les développements dans le cas symétrique pour pouvoir utiliser les solveurs CG et SYMMLQ. Ces derniers sont déjà utilisés en élasticité linéaire [S4] avec beaucoup de succès.
Comme pour le chapitre précédent, la plupart des méthodes proviennent de [S1]. Nous y renvoyons le lecteur pour plus de détails.

4.2 Algorithmes préconditionnés

- Types de préconditionnement :

Chaque algorithme présenté dans le chapitre précédent peut être modifié pour tenir compte d'un préconditionnement quelconque. Heureusement, la plupart des préconditionneurs peuvent se représenter sous la forme d'une matrice *M*. On peut ainsi écrire des algorithmes sans faire référence à un préconditionneur particulier. C'est ce que nous allons faire dans cette section.

La forme générale d'un système préconditionné peut apparaître sous trois variantes :
Le préconditionnement à gauche

$$M^{-1}\ A\ x = M^{-1}\ b$$

Le préconditionnement à droite

$$A\ M^{-1}\ y = b \quad x = M^{-1}\ y$$

La technique du 'splitting' (cas symétrique)

$$M = L\ L^T \quad ;\ \ L^{-1}\ A\ L^{-T}\ y = L^{-1}\ b$$

- Remarques :

Si on veut utiliser un certain solveur itératif, if faut que la matrice du système préconditionné ait les propriétés requises par celui-ci. Par exemple, la méthode SYMMLQ sera intéressante uniquement dans le cas d'une matrice de raideur symétrisée et un préconditionneur tel que cette symétrie soit préservée. Dans ce cas, il est utile d'utiliser la technique du 'splitting' et un préconditionneur défini positif ! (la factorisation $L\ L^T$ existe uniquement pour ce type de matrices). Ceci réduit fortement l'intérêt de l'algorithme face aux solveurs non symétriques.

Pour écrire un algorithme qui tient compte d'un préconditionneur, il suffit simplement de remplacer dans toutes les formules de récurrence la matrice *A* et le vecteur *b* par la matrice et le second membre du système préconditionné ($M^{-1}\ A$, ...). C'est pourquoi nous épargnons au lecteur la présentation détaillée de ces algorithmes. Cependant, pour le gradient conjugué et le GMRES, on peut profiter de certaines propriétés qui vont nous faciliter la vie.
Pour les autres solveurs (Bi-CG, CGS, ...), l'application du préconditionneur est immédiate et ne fait intervenir généralement qu'un vecteur auxiliaire.

Il est intéressant de noter que le produit $M^{-1} A$ n'est jamais calculé explicitement.

4.2.1 Le gradient conjugué préconditionné (PCG)

Grâce à ce qui va suivre, nous allons voir qu'il est possible d'utiliser le gradient conjugué avec les techniques de préconditionnement à gauche et à droite qui, d'habitude, détruisent la symétrie du système.
Le gradient conjugué s'appliquant essentiellement à des matrices symétriques et définies positives, il faut que la matrice M le soit aussi. De plus, pour pouvoir déduire l'algorithme préconditionné de la même manière que nous l'avons fait pour l'algorithme non préconditionné, il faut trouver un certain produit scalaire pour lequel la matrice du système est auto-adjointe, c'est à dire

$$\left(\left(M^{-1} A\right) x, y\right)_{?} = \left(x, \left(M^{-1} A\right) y\right)_{?}$$

- Préconditionnement à gauche

Puisque, par hypothèse, $M = M^T$, on a

$$\left(\left(M^{-1} A\right) x, y\right)_M = \left(A x, y\right) = \left(x, A y\right) = \left(x, M \left(M^{-1} A\right) y\right) = \left(x, \left(M^{-1} A\right) y\right)_M$$

de même

$$\left(\left(M^{-1} A\right) x, y\right)_A = \left(A M^{-1} A x, y\right) = \left(x, A M^{-1} A y\right) = \left(x, \left(M^{-1} A\right) y\right)_A$$

Appelons $r_j = b - A x_j$ le résidu des équations sans préconditionneur et

$z_j = M^{-1} r_j$ le résidu des équations préconditionnées

L'algorithme est obtenu à partir de l'algorithme du gradient conjugué en remplaçant directement

$$\begin{aligned} r_j &\rightarrow z_j \\ A &\rightarrow M^{-1} A \end{aligned}$$

Puisque $\left(z_j, z_j\right)_M = \left(r_j, z_j\right)$ et $\left(M^{-1} A p_j, p_j\right)_M = \left(A p_j, p_j\right)$, il n'y a jamais besoin de calculer les produits scalaires dans la métrique de M.

- Préconditionnement à droite

On remarque que l'on doit utiliser le produit scalaire dans la métrique de M^{-1}. On retombe sur le même algorithme. Il n'y a donc pas de choix possible.

4.2.2 Le GMRES préconditionné

Nous allons voir que le choix du type de préconditionnement est dicté par la facilité de calculer le résidu du système non préconditionné.

- Préconditionnement à gauche :

La boucle d'Arnoldi construit le sous-espace de Krylov

$$span\left\{r_0,\ M^{-1}\ A\,r_0,\ \ldots,\left(M^{-1}\ A\right)^{m-1}\ r_0\right\}$$

et les résidus calculés correspondent bien évidemment aux résidus des équations préconditionnées, c'est-à-dire

$$z_m = M^{-1}\ r_m = M^{-1}\left(b - A\,x_m\right)$$

Cependant, pour ce type de préconditionnement, il n'est pas simple de calculer les résidus non préconditionnés ; d'où l'intérêt de chercher une autre méthode.

- Préconditionnement à droite :

$$A\ M^{-1}\ u = b, \quad u = M\ x$$

La variable *u* n'apparaît pas explicitement dans l'algorithme.
Dans ce cas, le résidu du système initial est connu sans calcul supplémentaire puisque les résidus des deux systèmes sont égaux :

$$r_m = b - A\,x_m = b - A\ M^{-1}\ M\ u_m = b - A\,u_m$$

C'est, bien sûr, cette méthode qui sera utilisée pour METAFOR.

- Préconditionnement par 'splitting'

Il n'a pas d'intérêt ici puisque la matrice n'a pas besoin d'être symétrique. De plus, personne n'a prouvé que cette méthode donnait de meilleurs résultats.

4.3 Techniques de préconditionnement

4.3.1 Introduction

D'un point de vue technique, un préconditionneur peut être vu comme n'importe quel solveur approximatif auxiliaire à une méthode itérative. Le problème qui se pose ici est l'approximation de l'inverse d'une matrice.
Pour des raisons d'efficacité, la matrice *M* doit satisfaire certains critères :

- *M* doit être facile à manipuler :

Pour cela, elle doit être creuse. On doit aussi pouvoir calculer facilement son inverse puisqu'elle apparaît à chaque itération sous la forme d'une résolution de système.
Si on garde en mémoire que l'on essaye de concurrencer un solveur direct, on se rend compte facilement des restrictions qui en découlent. Par exemple, si on effectue 10 itérations, on devra résoudre 10 ou 20 systèmes *N x N* suivant que l'on utilise le GMRES ou une méthode de la famille du Bi-CG.

- Le nombre d'itération de la méthode doit diminuer :

C'est d'autant plus justifié que le nombre d'opérations par itération va être augmenté par l'utilisation du préconditionneur (construction et utilisation). L'efficacité de celui-ci se mesura donc par la diminution du nombre d'itérations mais aussi, et surtout, par la diminution du temps de calcul total.

- *M* doit être le plus proche de *A* possible :

En effet, dans le cas limite où $M = A$, le système à résoudre est

$$I\ x = A^{-1}\ b$$

et le premier espace de Krylov, formé par le résidu initial (avec $x_0 = 0$), contient la solution.
On converge donc en une seule itération !

4.3.2 SOR & SSOR

Revenons momentanément aux méthodes itératives stationnaires. Chacune d'entre elles peut être utilisée comme préconditionneur. La plus puissante d'entre elles est la méthode SSOR (successive symetric over relaxation). Les quelques lignes suivantes montrent comment déduire la matrice *M* d'un tel solveur.

Considérons le système général $A\,x = b$ et décomposons la matrice A sous la forme $A = M - N$ (splitting de A).

La méthode stationnaire associée peut s'écrire

$$x^{(k+1)} = M^{-1}\, N\, x^{(k)} + M^{-1}\, b$$
$$x^{(k+1)} = G\, x^{(k)} + f \quad \text{où} \quad G = M^{-1}\, N = I - M^{-1}\, A$$

Le processus tend à résoudre le système modifié

$$(I - G)\, x = f$$

c'est-à-dire, en remplaçant G et f par leurs valeurs

$$\boxed{M^{-1}\, A\, x = M^{-1}\, b}$$

qui est le système préconditionné associé au splitting M - N.

D'une autre manière, si on écrit la décomposition de A sous la forme

$A = D - E - F$ avec E la partie triangulaire inférieure de A,
F la partie triangulaire supérieure de A,

on a, d'après la théorie des itérations de Gauss-Seidel

$$M_{SSOR} = (D - E)\, D^{-1}\, (D - \omega\, F)$$
$$M_{SOR} = (D - E)\, D^{-1}\, (D - F)$$

D'un autre point de vue, ces expressions sont des factorisations approchées de la matrice A.

- Avantages :

  - L'utilisation d'une telle formule est immédiate : on utilise successivement deux substitutions arrière et avant pour appliquer le préconditionneur à un vecteur.
  - On peut économiser une quantité de mémoire non négligeable en effectuant les substitutions à partir de A. Dans ce cas, aucune mémoire auxiliaire n'est nécessaire.
  - Ces préconditionneurs ont eu un grand succès en mécanique des fluides (pour des matrices formées de quelques diagonales découlant de la discrétisation d'équations aux dérivées partielles par différences finies).

- Inconvénients :

  - La factorisation obtenue peut être médiocre pour une matrice arbitraire donnée.
  - Le paramètre ω optimal n'est pas facile à évaluer (souvent fonction des valeurs propres de la matrice).

4.3.3 Factorisation LU incomplète

- Introduction :

Nous avons vu qu'il est intéressant de former un préconditionneur sous la forme d'une factorisation pour faciliter le calcul du produit de son inverse. Utiliser une factorisation *LU* vient donc directement à l'esprit. Cependant, si la matrice *A* est creuse, les matrices *L* et *U* sont, en général, pleines (tous les éléments sous la ligne de ciel deviennent non nuls). Il faut donc choisir de garder certains éléments lors de l'élimination de Gauss et d'en éliminer d'autres.

Cette technique a été créée au départ pour les matrices de type *M*[8]. Elle consiste à décomposer la matrice du système suivant

$$A = L\,U - R$$

Un théorème (Ky Fan) garantit que pour des matrices de type *M*, le procédé décrit ci-dessus produit un splitting régulier de *A* ($A = L\,U - R$). Dans les autres cas, on peut rencontrer un pivot nul ou négatif. Ce qui pose certains problèmes.
Il faut alors modifier la matrice (par exemple $A + \alpha I$) ou effectuer un pivotage pour éviter des divisions par zéro.

Dans la suite, nous notons *NZ*(*A*) l'ensemble des paires d'indices (*i*, *j*) correspondant à des éléments non nuls.

4.3.3.1 ILU(0) : factorisation d'ordre zéro

Pour obtenir ce préconditionneur, on annule tous les éléments qui tombent à des positions d'éléments nuls dans la matrice *A* lors de la factorisation. Si *A* est une matrice de type *M* alors il en sera de même pour sa factorisation approchée $M = L\,U$, elle aussi symétrique et définie positive.

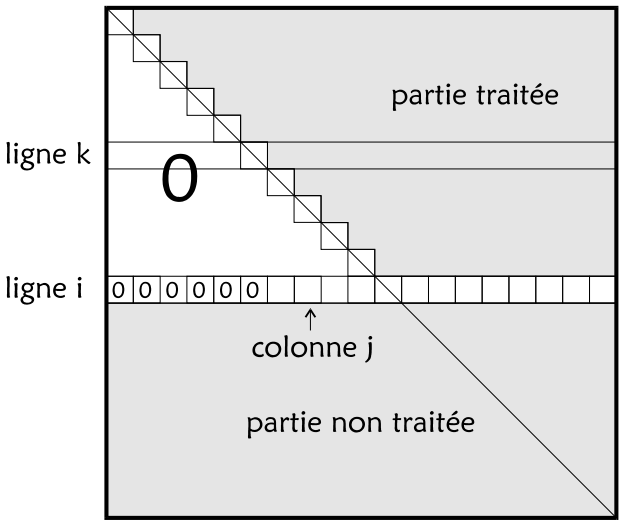
- Par définition :

  Une factorisation *LU* incomplète sans remplissage (fill-in), ou ILU(0), est donnée par deux matrices *L* et *U* satisfaisant les conditions :
  1. *L* est triangulaire inférieure et $(i, j) \in \mathrm{NZ}(L)$ ssi $i \geq j$ et $(i, j) \in \mathrm{NZ}(A)$
  2. *U* est triangulaire supérieure et $(i, j) \in \mathrm{NZ}(U)$ ssi $i \leq j$ et $(i, j) \in \mathrm{NZ}(A)$
  3. $(LU)_{ij} = a_{ij}$ pour tout (i, j) c $\mathrm{NZ}(A)$

- Algorithme & programmation :

[8] Une matrice *A* est de type *M* si elle est symétrique et définie positive,

$$A_{i,j} \leq 0 \text{ si } i \neq j \text{ et}$$
$$A_{i,j} > 0 \text{ si } i = j$$



La seule difficulté lors de l'implémentation vient de la méthode de stockage utilisée pour la matrice A et le préconditionneur. Pour stocker ce dernier, on utilise le format MSR défini dans le chapitre 2.

Pour rendre l'algorithme plus efficace, on utilise une variante de l'algorithme de Gauss qui permet d'effectuer l'élimination ligne après ligne (puisque c'est dans cet ordre que la matrice est stockée).

Ci-dessous, l'algorithme de la méthode :

Algorithme : ILU(0)

$u_{1,*} = a_{1,*}$
DO i = 2, N
  $u_{i,*} = a_{i,*}$
  DO k = 1, i-1
    IF (i, k) $\in$ NZ(A) THEN
      $l_{i,k} = u_{ik} / u_{kk}$
      DO j = k+1, N
        IF (i, j) $\in$ NZ(A) THEN $u_{i,j} = u_{i,j} - l_{i,k}\, u_{k,j}$
      ENDDO
    ENDIF
  ENDDO
ENDDO

Pour effectuer la combinaison linéaire de la ligne courante (indice i) par la ligne supérieure (indice k), on effectue la boucle sur les éléments non nuls de la ligne k. Pour une question de rapidité, il est indispensable de garder en mémoire l'emplacement des éléments non nuls de la ligne i. En effet, il est impossible d'obtenir la valeur de l'élément $u_{i,j}$ sans tester tous les éléments de la ligne, à cause du format de stockage.

On utilise alors un vecteur auxiliaire d'entiers de taille N qui va être une image de la ligne i décomprimée (c'est-à-dire avec tous ses zéros). On le construit de la manière suivante :
- Si $A_{i,j} = 0$ alors l'élément j du vecteur est nul.
- Si $A_{i,j} \neq 0$ alors l'élément j du vecteur pointe vers la position de $A_{i,j}$ dans les vecteurs A et JA du format CSR.

- Variante MILU0 (Modified ILU) :

Au lieu d'ignorer l'élément, on le soustrait des éléments diagonaux correspondants (la somme des éléments de chaque ligne et de chaque colonne est ainsi préservée). Cette méthode peut éviter l'apparition de pivots nuls dans certains cas.

4.3.3.2 ILU(*p*) : factorisation d'ordre *p*

Pour améliorer le préconditionneur, il est possible de choisir de garder d'autres éléments que ceux définis par la structure de la matrice *A*. Les méthodes qui suivent deviennent beaucoup plus compliquées à programmer parce qu'on ne connaît plus la structure de la matrice de préconditionnement à l'avance.

La première méthode consiste à tolérer les remplissages d'ordre *p* maximum.
Par définition, on appelle 'ordre de remplissage' (level of fill) un nombre associé à chaque élément de matrice lors de l'élimination de Gauss et calculé de la manière suivante :

$$f_{ij} = -l_{ik}\ u_{kj} \rightarrow \text{level}\left(f_{ij}\right) = \text{level}\left(l_{ik}\right) + \text{level}\left(u_{kj}\right) + 1.$$

Au début du calcul, tous les niveaux de remplissage sont initialisés à zéro.

- Inconvénients :

Cette méthode possède plus d'inconvénients que d'avantages. Elle ne sera donc pas utilisée dans le cadre de METAFOR.
- Méthode coûteuse en temps de calcul.
- Il est nécessaire de garder en mémoire tous les niveaux de remplissage de chaque élément de la matrice. Le gain de mémoire dû à l'utilisation d'un solveur itératif est diminué.
- Le nombre d'éléments annulés est imprévisible. Il est donc difficile de réserver une quantité limitée de mémoire pour le préconditionneur.
- Nous n'avons aucun contrôle sur la grandeur des éléments gardés et éliminés. En général, on constate que les plus grands sont les plus importants.
- Des tests ont été effectués pour la méthode des volumes finis par Delanaye [D1] et montrent que le gain obtenu par la qualité du préconditionnement est faible vis-à-vis du travail nécessaire pour la construction de la factorisation incomplète.

4.3.3.3 ILUT (stratégie de seuil)

Ce préconditionneur est un des plus puissants à l'heure actuelle pour la résolution de systèmes quelconques non singuliers. La présence de plusieurs paramètres permet de le rendre aussi puissant que l'on veut. Notons qu'il n'inclut pas ILU(0) comme cas limite.

Cette méthode consiste à ignorer les éléments d'après leur taille plutôt que d'après leur position. Ce choix n'a aucun fondement théorique ; cependant, on peut montrer qu'en pratique, pour une quantité d'éléments non nuls donnés, ILUT est plus efficace que ILU(*p*).

- Algorithme :

La routine utilisée pour METAFOR provient de SPARSKIT 2.0 de Saad [S2]. Elle est totalement optimisée pour effectuer un nombre d'opérations minimal.

L'algorithme est à peu près identique à celui de ILU(0). Seule change la règle de sélection des éléments de remplissage à garder.

Algorithme ILUT(*lfil*, *droptol*)

row(1 :*N*) = 0
DO *i* = 2, *N*
    row(1:*N*) = *A*(i,1:*N*)
    norm = norme moyenne des éléments de la ligne *i*.
    DO *k* = 1, *i*-1
        IF *row*(*k*) $\neq$ 0 THEN
            row(*k*) = *row*(*k*) / *A*(*k*,*k*)
            IF |*row*(*k*)| < *droptol* * *norm* THEN *row*(*k*) = 0
            IF *row*(*k*) ≠ 0 THEN *row*(*k*+1:*N*) = *row(k*+1,*N)* - *row*(*k*) * *U*(*k*,*k*+1:*N*)
        ENDIF
    ENDDO
    On retient les *p* plus grand éléments en norme dans *row* (QSPLIT).
    L(*i*,1:*i*-1) = *row*(1:*i*-1)
    U(*i*,*i*:*N*) = *row*(*i*:*N*)
ENDDO

Vu que la structure de la matrice de préconditionnement est inconnue à priori, on utilise un vecteur auxiliaire (appelé *row*) dans lequel on effectue les combinaisons linéaires successives. Pour gagner de la mémoire, Saad utilise un format particulier pour stocker le vecteur *row* :
- *r* : un vecteur de réels contenant tous les éléments non nuls de la ligne.
- *jr* : un vecteur d'entiers de la même taille que *r* et contenant le numéro de colonne de l'élément correspondant dans *r*.
- *ir* : un vecteur d'entiers de taille *N* analogue au vecteur auxiliaire utilisé pour ILU(0). Ses éléments non nuls pointent vers l'élément correspondant dans *r* et *jr*.

Après le calcul du pivot, on compare sa norme avec la norme moyenne de la ligne avant élimination multipliée par un facteur '*droptol*' fixé par l'utilisateur. Si elle est inférieure, on l'ignore et on n'effectue pas la combinaison linéaire.

Lorsque l'élimination de la ligne *i* est terminée, on trie le vecteur *row* par un algorithme dérivé du 'quick sort' qui sépare les *lfil* plus grands éléments des autres (en réalité, on garde *lfil* éléments dans *L* et *lfil* éléments dans *U*). Ces derniers sont ignorés (remplacés par 0) et on peut alors construire la ligne *i* des matrices *L* et *U* à partir de ceux qui restent.

- Inconvénients de ILUT :

  Bien qu'il soit très puissant, ce préconditionneur n'est pas la solution ultime :
  - On peut rencontrer un pivot nul : le remède consiste à inclure une technique de permutation de colonnes (ILUTP). Dans ce cas, si la matrice est symétrique, cette propriété est perdue. De plus, on perd l'avantage du réarrangement de la matrice pour un algorithme tel que SLOAN (la ligne de ciel est modifiée). Cependant, cette solution donne généralement de meilleurs résultats que le remplacement de l'élément nul par un nombre arbitraire. C'est pourquoi nous l'utilisons dans METAFOR.
  - L'algorithme est parfois instable et la convergence n'est pas toujours assurée.

4.3.3.4 Factorisation LL^T incomplète (IC(0))

La factorisation de Choleski. est utilisée pour des matrices symétriques définies positives. Nous expliquerons plus loin comment on peut étendre les résultats aux cas de matrices non définies positives.
L'algorithme utilisé pour METAFOR est dérivé de celui de N.M. Nachtigal [N2]. Dans ce dernier, on parcourt la matrice de la même manière que pour ILU(0). Seules les formules changent.

Algorithme IC(0)

DO i = 1, N
  $RSUM = 0$
  DO j = 1, i-1
    $$ZSUM = \frac{1}{L_{j,j}}\left(A_{i,j} - \sum_{k=1}^{j} L_{i,k}\, L_{j,k} \right)$$
    IF $(i, j) \in$ NZ(A) THEN
      $L_{i,j} = ZSUM$
    ENDIF
    $RSUM = RSUM + ZSUM^2$
  ENDDO
  $L_{i,i} = \sqrt{A_{i,i} - RSUM}$
ENDDO

- Stieltjes reduction [S4]:

Remarquons, en particulier, que l'élément $L_{i,i}$ est calculé par une racine carrée. Dans le cas d'une factorisation complète, le radicand est toujours positif uniquement lorsque la matrice A est symétrique et définie positive. Par contre, pour une factorisation incomplète, la condition est bien plus sévère : il faut que la matrice soit de type M.
Il faut donc transformer la matrice A en une matrice de type M (aussi appelée matrice de Stieltjes, d'où le nom de la procédure).

On utilise : Si $A_{i,j} > 0$ ($i \neq j$), alors $A_{i,j} = 0$ et $A_{i,i} \rightarrow A_{i,i} + A_{i,j}$
Si $A_{i,j} < 0$ ($i \neq j$), alors aucun changement.

Cette procédure produit toujours une matrice de type *M* si la matrice de départ est symétrique et définie positive. Si ce n'est pas le cas, on peut ajouter un terme positif sur la diagonale pour rendre la matrice de départ définie positive.

- Variante MIC(0) :

Tout comme pour ILU(0), on peut écrire une variante de IC(0) en ajoutant les éléments ignorés multipliés par un certain facteur (habituellement proche de 1) à l'élément diagonal correspondant. Ces méthodes donnent de très bons résultats en élasticité linéaire

4.3.4 Autres techniques

Cette section décrit très brièvement d'autres méthodes de préconditionnement. Elles sont en général moins employées que les factorisations *LU* parce que trop récentes, trop théoriques ou moins adaptée au cas qui nous intéresse.

- Approximation de l'inverse [S4,B1] :

Un autre moyen de préconditionner le système consiste à trouver une matrice (creuse) qui se rapproche de l'inverse de la matrice *A*. On peut par exemple considérer le problème en minimisant de la norme de Frobenius de la matrice $I - A\,M^{-1}$

$$\left\| I - A\,M^{-1} \right\|_F^2 = tr\left[\left(I - A\,M^{-1} \right)^T \left(I - A\,M^{-1} \right) \right] = \sum_{j=1}^{N} \left\| e_j - A\,M^{-1}\,e_j \right\|_2^2$$

Ce qui revient à résoudre approximativement au sens des moindres carrés le système

$$A\,m_j^{-1} = e_j$$

Pour cela, deux méthodes sont possibles :
- Soit choisir à l'avance les éléments non nul de M^{-1} (non zero pattern) et résoudre le système. Ce qui revient à effectuer une multiplication matricielle.
- Soit choisir m_j, un vecteur creux et faire quelques itérations de CGNR ou de steepest descent en utilisant des critères pour garder uniquement certains éléments.

Cette méthode est particulièrement intéressante pour des ordinateurs multi-processeurs, chaque colonne de la matrice étant calculée indépendamment des autres.

- Préconditionnement des équations normales :

Les matrices rencontrées étant symétriques et définies positives, ILU(0) est remplacé par une factorisation incomplète de Choleski IC(0).

- ILQ (Gram-Schmidt) [S3] :

Factorisation $A = L\,Q$ incomplète (on ne garde que certains éléments dans chaque colonne pour obtenir une matrice *M* creuse). Utile pour préconditionner les équations normales.

- Préconditionnement polynômial [B1] :

On essaye d'approcher l'inverse par une série tronquée.

5. Exemples numériques

5.1 Introduction

Nous avons constaté d'un point de vue théorique que l'efficacité des solveurs itératifs est maximale pour des gros problèmes caractérisés par une matrice de raideur tangente très creuse. Nous nous proposons de le prouver pratiquement en utilisant les routines et les préconditionneurs décrits dans les sections précédentes sur des problèmes variés (contact, chargement par pressions non conservatives, utilisation de différentes lois de comportement,...).

Ce ne sera pas aussi simple que cela en à l'air. En effet, la principale difficulté pour utiliser efficacement un solveur itératif est la quantité impressionnante de paramètres qui entrent en jeu. Citons par exemple le paramètre de restart du GMRES, le nombre maximum d'itérations admissible, le paramètre de remplissage du préconditionneur, la précision du processus de Newton-Raphson, etc. Il est donc nécessaire de s'habituer aux phénomènes rencontrés pour ne pas choisir une mauvaise méthode de résolution

De plus, le succès d'un solveur itératif est fonction des caractéristiques de la matrice du système à résoudre. Si la matrice est bien conditionnée, il ne faudra que quelques itérations pour trouver la solution avec une grande précision. Dans le cas contraire, le nombre d'itérations nécessaires peut devenir très élevé, et parfois même il n'est plus possible d'obtenir une solution acceptable. Ces solveurs sont donc beaucoup moins robustes que leurs homologues traditionnels, comme celui utilisé dans METAFOR.

Il n'est pas facile de deviner a priori ce qui va entraîner une convergence rapide de l'algorithme de Newton-Raphson couplé à un solveur itératif du type GMRES ou Bi-CG. En effet, très peu d'auteurs utilisent ce genre de méthode en mécanique du solide pour des

grandes déformations non linéaires. Par contre, le gradient conjugué commence à prendre une place importante en mécanique linéaire et nous nous baserons sur les résultats des récents travaux dans ce domaine pour avoir une première idée des phénomènes que l'on peut rencontrer.

P. Saint-Georges [S4] cite quelques facteurs qui influencent la matrice de raideur en élasticité linéaire :

- L'aspect des mailles E.F. : plus le maillage comporte des éléments aplatis, plus difficile sera la résolution du problème par une méthode itérative. Il est parfois utile de raffiner le maillage à ces endroits critiques pour faciliter la résolution. Dans ce cas, l'augmentation du temps de calcul provoquée par l'accroissement de la taille du système est compensée par la diminution du nombre d'itérations du solveur itératif. Au total, le temps C.P.U. est réduit.
- Des fortes variations spatiales de la taille des éléments : on peut y remédier de la même manière.
- Les propriétés matérielles (discontinuité du module de Young) : dans le cas non linéaire, ce facteur va intervenir souvent. Par exemple, si un matériau plastifie, sa raideur va diminuer fortement dans la zone en question. Nous verrons que les lois hyperélastiques incompressibles de METAFOR posent aussi des problèmes.
- Le type d'éléments et le degré des fonctions de forme : ceci ne nous intéresse pas puisque METAFOR utilise toujours des quadrangles du premier degré.

On retrouvera certains de ces phénomènes dans le cas non linéaire. Cependant, nous verrons qu'ils sont souvent amplifiés par les non linéarités du problème (par exemple, la taille des mailles varie au cours du calcul, contrairement au cas de petites déformations). M. Papadrakakis [P2] explique dans son travail que les grands déplacements subits par les mailles constituent une source de difficultés supplémentaires pour la résolution du système linéarisé par solveur itératif.

Le premier exemple numérique va nous permettre d'effectuer une première comparaison de toutes les méthodes (solveurs et préconditionneurs) décrits précédemment. Il s'agit de la flexion d'une poutre élastoplastique soumise à une charge répartie uniformément sur sa longueur. Nous étudierons simultanément l'influence de l'aspect des éléments du maillage sur leur efficacité. Dans un deuxième temps, nous traiterons le problème à trois dimensions pour déterminer à partir de quel moment les solveurs itératifs deviennent avantageux.

Le deuxième test consiste à comprimer un cube hyperélastique. Ce type de matériau entraîne souvent un très mauvais conditionnement de la matrice de raideur ; ce qui se ressentira lors de la résolution. Nous utiliserons aussi cette géométrie dans un cas favorable aux solveurs itératifs pour montrer les performances impressionnantes que l'on peut obtenir lorsque la matrice de raideur du système possède une grande largeur de bande.

Le troisième test est tout aussi intéressant parce qu'il nous a permis de traiter un problème impossible à résoudre par METAFOR jusqu'à ce jour à cause de la trop grande quantité de mémoire nécessaire (presque 20 000 degrés de libertés). Il s'agit d'un problème de compaction de poudre. Nous en profiterons pour étudier sur des plus petits cas l'influence de la précision du solveur. En effet, puisque la résolution du système d'équation fait partie d'un algorithme de Newton-Raphson, on peut se permettre de trouver une solution approchée dans la plupart des cas.

Les tests suivants envisagent le cas des problèmes de contact entre solides et matrices rigides. Malheureusement, les lois 3D modélisant ce phénomène ne sont pas encore toutes au point. Seul le contact sans frottement entre un plan et un solide peut être envisagé sans problème. Nous nous contenterons donc d'une étude rapide en 2D pour couvrir les autres cas.

L'étude de l'écrasement d'un cylindre en acier permettra de comparer les différents solveurs entre eux. Les résultats permettront de confirmer ceux obtenus dans le cas bidimensionnel de la poutre en flexion.

Pour étudier les lois de contact collant et avec frottement, nous considérerons le formage d'un boulon par écrasement et celui d'un composant aéronautique par superplasticité.

Enfin, un dernier test sera consacré au calcul de la déformation de deux plaques collées par un joint. C'est un problème 3D qui possède des mailles très aplaties.

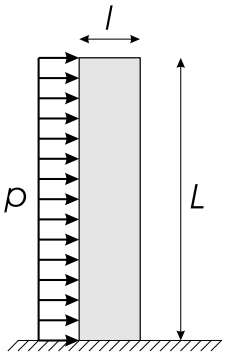
En résumé, le but de cette partie du travail est d'analyser tout ce qui peut influencer un algorithme itératif dans le cadre de l'étude des grandes déformations de solides. On déterminera également les qualités et les faiblesses des solveurs itératifs par rapport aux solveurs directs. A la fin de ce chapitre, nous résumerons les résultats sous la forme d'un organigramme permettant de choisir la meilleure méthode pour un type de problème donné. Nous expliquerons également le raisonnement à suivre pour fixer correctement les paramètres suivants le but recherché (rapidité, gain de mémoire, robustesse, ...).

Trouver des exemples numériques pour ce genre de travail n'est pas très simple. En effet, pour l'instant, la plupart des problèmes traités sont en 2D à cause de l'absence des lois de contact et des forces non conservatives en 3D. Les problèmes qui 'tournent' longtemps sont souvent des problèmes de taille modeste pour lesquels il est nécessaire d'effectuer de nombreux pas de temps afin de converger vers la solution.
Avec la mise au point imminente des lois de contact en 3D, on s'attend à traiter plus fréquemment des gros problèmes et les solveurs itératifs pourront alors entrer en jeu !

5.2 La poutre encastrée chargée uniformément sur sa longueur

5.2.1 Influence de la forme des mailles

5.2.1.1 Description du problème

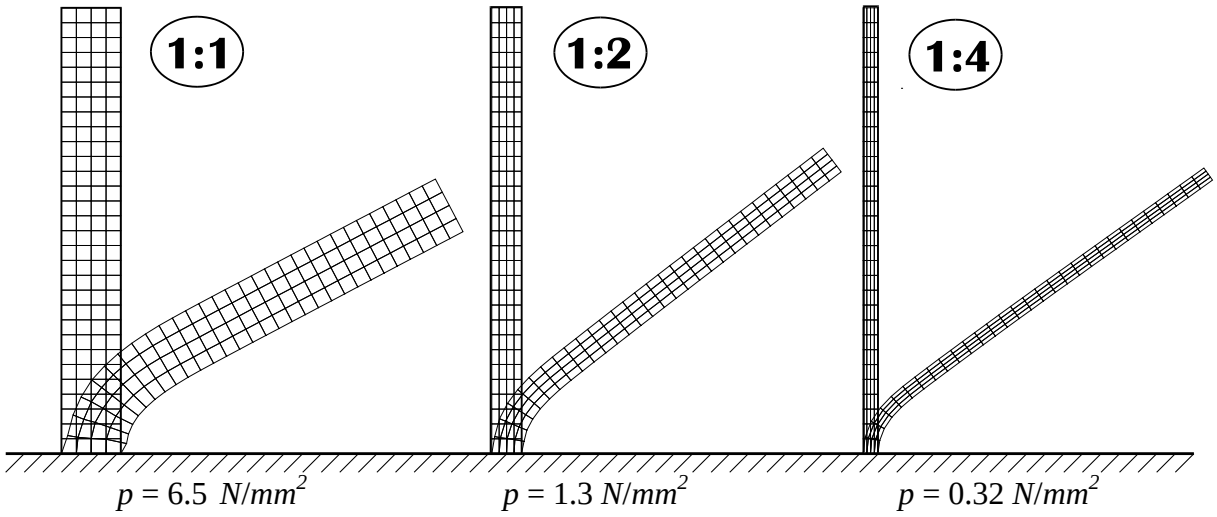


Considérons trois poutres en acier de longueurs identiques mais de largeurs différentes. On impose une charge (p) uniforme répartie sur la longueur de celles-ci de telle manière à ce que les déformées soient similaires. A tout instant, les charges nodales équivalentes vont rester perpendiculaires à la poutre. Comme nous l'avons vu, ce type de forces provoquent une augmentation de la non symétrie de la matrice de raideur.

Nous considérerons trois poutres caractérisées par une largeur différente :

$L = 300\ mm$ l = variable (40, 20 ou 10 mm)

Les trois poutres sont maillées de la même manière, c'est-à-dire 30 éléments selon la longueur et 4 selon la largeur. Elles possèdent donc un nombre de degrés de liberté identique. Ce nombre n'interviendra pas dans les résultats qui suivent.



On a donc des rapports de longueurs 1:1, 1:2 et 1:4. Nous devrions obtenir un nombre d'itérations (du solveur) croissant lorsque les mailles deviennent plus plates.

On utilisera les valeurs suivantes pour le matériau :

$$E = 200000\ N/mm^2 \qquad \nu = 0.3$$
$$\sigma_e = 250\ N/mm^2 \qquad h = 1000\ N/mm^2$$

où σ_e est la limite élastique du matériau,
h est le coefficient d'écrouissage.

Pourquoi choisir un problème 2D ?
Insistons encore une fois sur le fait que notre but n'est pas d'obtenir de meilleurs résultats que la méthode directe mais simplement de regarder l'influence du conditionnement de la matrice de raideur sur la résolution itérative. D'ailleurs, nous montrerons qu'il est impossible de concurrencer le solveur direct pour ce genre de problème. L'avantage principal de choisir des problèmes 2D pour effectuer les premières comparaisons est le nombre important de tests que l'on peut faire en un temps relativement court. Pour effectuer ce genre de tests sur des problèmes plus importants, il faudrait compter plusieurs semaines voire plusieurs mois de calculs.

Des tests similaires ont été publiés par les auteurs d'ABAQUS [H1] dans le cadre d'une publicité pour illustrer les phénomènes que l'on peut rencontrer en utilisant un solveur itératif. Cependant, ils ne donnent aucune indication sur les caractéristiques du matériau utilisé. C'est pourquoi, nous avons choisi une loi élastoplastique à écrouissage linéaire (nous pourrons ainsi voir l'effet des non linéarités de la loi de comportement).

Nous allons tout d'abord utiliser le solveur direct pour pouvoir comparer les résultats. Ensuite, nous considérerons chaque méthode itérative l'une après l'autre dans le but de comparer les méthodes de préconditionnement. Une section spéciale sera consacrée à la comparaison des solveurs entre eux.

5.2.1.2 Résultats du solveur direct

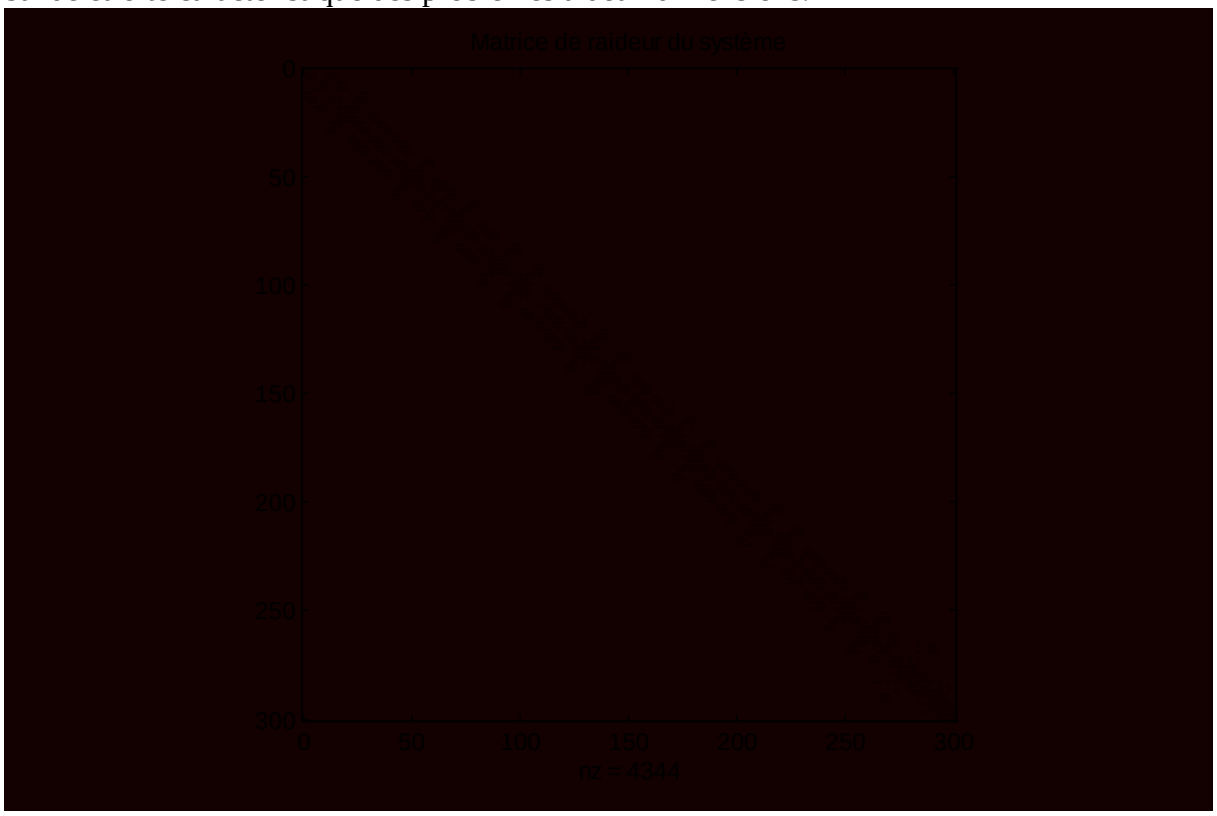
La taille du système à résoudre à chaque itération de Newton-Raphson est assez petite. On manipule ici des matrices de 300 *x* 300.
Puisque le problème est bidimensionnel, nous nous attendons à obtenir une faible largeur de bande, ce qui va favoriser sa résolution par la méthode de Gauss.

Largeur de bande[9] : $h_{max} = 26$, $h_{moy} = 12$.
Nombre de termes retenus : 7020 (7.8 %).
Mémoire utilisée : 52853 flottants double précision (codés sur 8 octets)

Pour les essais suivants, la matrice de raideur n'est pas symétrisée.

[9] Pour caractériser la largeur de bande, nous utilisons la hauteur de colonne moyenne du format SKYLINE (h_{moy}) et la hauteur de colonne maximale (h_{max}). Ce n'est donc pas rigoureusement le nombre défini en analyse numérique. La largeur de bande moyenne vaut donc 2 h_{moy}.

Ci-dessous, on a représenté la structure de la matrice de raideur des trois problèmes. Chaque point noir représente un élément non nul. On observe que l'algorithme de réarrangement des inconnues (SLOAN) a effectué un bon travail : la structure de la matrice se réduit à une fine bande étroite caractéristique des problèmes à deux dimensions.



Le tableau suivant regroupe les résultats pour les trois poutres :

Aspect	temps CPU [*s*]	Pas	Itérations N-R	CPU_{moyen} [10^{-2} *s*]
1 : 1	4.45	51	125	3.560
1 : 2	5.77	67	169	3.414
1 : 4	5.88	71	172	3.418

Dans la suite, nous désignons par CPU moyen le temps CPU total divisé par le nombre d'itérations de Newton-Raphson total.

- Observations :

La résolution du problème envisagé est influencée par l'aspect des mailles :
Plus les mailles sont aplaties, plus il faut effectuer de pas de temps pour converger vers la solution. Ce phénomène s'observe fréquemment et n'est pas particulier à la flexion d'une poutre. Pour comparer les résultats, il faudra donc considérer des moyennes, notamment pour le temps CPU. En effet, il serait trop facile de constater une augmentation du temps CPU total et de conclure à un mauvais conditionnement du système alors que le nombre de résolutions (nombre d'itérations de Newton-Raphson) n'est pas le même dans tous les cas.

Le temps CPU moyen est à peu près constant :
Théoriquement, si tout le temps de calcul était consacré à l'algorithme de Gauss, ces nombres devraient être égaux. Il existe aussi une certaine erreur dans le temps de calcul mesuré : si on effectue plusieurs fois le même test, le temps donné par METAFOR varie d'un à deux dixièmes de seconde. Ceci aussi peut expliquer les différences.

5.2.1.3 Résolution par GMRES(m)

- Introduction et remarques préliminaires :

Choix du paramètre de restart :
Dans un premier temps, nous utilisons un paramètre de restart (*m=80*) très élevé pour éliminer l'effet d'une perte brutale d'orthogonalité des vecteurs de Lanczos. En effet, si on fixait ce paramètre trop bas, les préconditionneurs de mauvaise qualité entraîneraient des résultats encore pires. L'effet du paramètre *m* sur la vitesse de convergence sera discuté plus tard.
On se place donc dans des conditions avantageuses pour les mauvais préconditionneurs. En pratique, pour des gros problèmes 3D possédant plusieurs milliers de degrés de liberté, il est impensable d'utiliser un rapport *m* / *N* si élevé (0.26).

Mémoire utilisée :
Le problème étant petit, la matrice de raideur n'est pas très creuse comparée aux gros problèmes à trois dimensions pour lequel on atteint facilement moins d'un pourcent d'éléments non nuls. On compte ici (*NNZ* =) 4576 éléments non nuls, c'est-à-dire 5.08 % par rapport à une matrice pleine.
En général, pour les problèmes mécaniques 2D, on a en moyenne entre 20 et 30 éléments non nuls par ligne dans la matrice de raideur. Lorsqu'on utilise un grand paramètre de restart, il faut garder en tête que la place utilisée en mémoire pour stocker les vecteurs de Lanczos représente approximativement entre *m*/30 et *m*/20 fois celle de la matrice de raideur. Si on ajoute à cela la place du préconditionneur, on constate qu'il faut considérer de très gros problèmes pour gagner au total de la mémoire par rapport à la méthode directe et son format SKYLINE.

Choix des paramètres it_{max} et ε :
Nous fixons aussi un nombre d'itérations maximum (it_{max} = 220) pour limiter le temps de calcul. Nous verrons que celui-ci peut influencer la solution obtenue en fin de calcul.
La précision du solveur itératif (ε) est fixée à 10^{-6} (le solveur essaye de diminuer la norme du résidu de cette valeur).

Remarque sur le préconditionnement :
Pour le préconditionneur ILUT, nous utilisons une tolérance de rejet (*droptol*) nulle pour être certain d'obtenir dans les trois cas une proportion d'éléments non nuls constante. Rappelons que la notation ILUT(*lfil*) signifie que l'on effectue une factorisation incomplète de la matrice de raideur en conservant dans chaque ligne de *L* et chaque ligne de *U* un nombre *lfil* d'éléments.

C'est pourquoi, pour estimer la mémoire nécessaire au préconditionnement, on utilise la formule :

$$NNZ2 = 2 * lfil * N$$

où *N* est le nombre d'équations et
lfil est le niveau de remplissage ('level of fill').
Cette valeur est toujours une borne supérieure.

Les autres types de préconditionnement (SSOR et MILU(0)) seront testés à part, vu qu'ils sont moins puissants et ne donnent pas de bons résultats.
Passons maintenant aux résultats numériques :

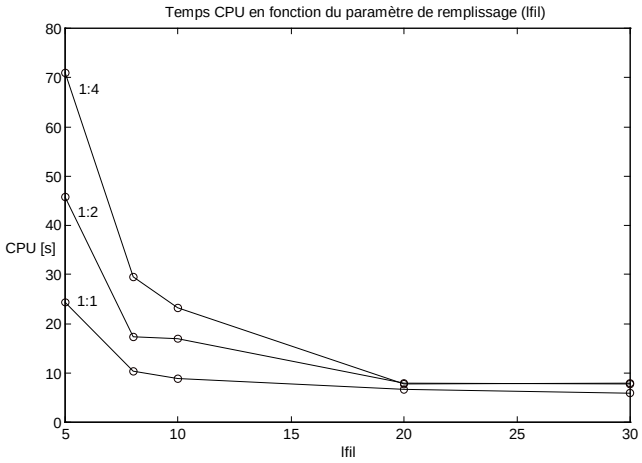
- Temps CPU :

[*s*]	ILU(0)	ILUT(5)	ILUT(8)	ILUT(10)	ILUT(20)	ILUT(30)
1 : 1	10.83	24.33	10.27	8.87	6.68	5.92
1 : 2	16.32	45.77	17.32	16.95	7.90	7.70
1 : 4	38.68	71.02	29.52	23.30	7.82	7.98

Le temps CPU est toujours plus élevé que celui obtenu par la méthode directe :
Au mieux, on utilise 35 % de temps en plus. Il faudra donc considérer des problèmes plus grands pour espérer obtenir un gain par rapport à ce dernier.

Variation en fonction de *lfil* :
Remarquons que le temps CPU est fortement sensible au paramètre de remplissage du préconditionneur ILUT : en doublant le nombre d'éléments de celui-ci (en utilisant ILUT(10) au lieu de ILUT(5)), on divise par trois le temps de calcul.

Le choix d'un mauvais préconditionneur peut entraîner des temps de calcul très grands (jusqu'à 12 fois plus que le solveur direct pour ILUT(5).

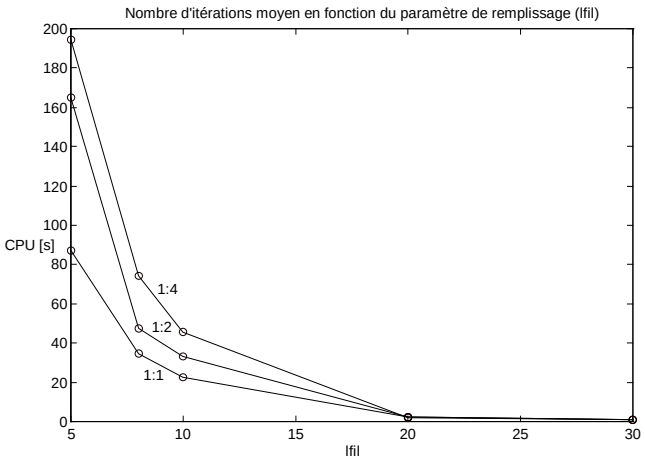


- Nombre d'itérations moyen pour résoudre un système :

	ILU(0)	ILUT(5)	ILUT(8)	ILUT(10)	ILUT(20)	ILUT(30)
1 : 1	39.78	87.02	34.78	22.64	2.18	1
1 : 2	58.04	164.9	47.39	33.09	2.19	1
1 : 4	105.2	194.7	73.97	45.63	2	1

Comparaison des préconditionneurs :
On constate que, pour un préconditionneur donné, le nombre d'itérations moyen du solveur itératif est d'autant plus grand que les mailles sont aplaties.

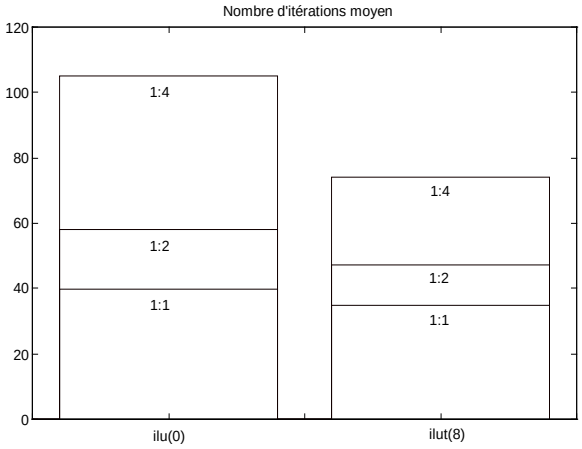
L'affirmation de P. St Georges est donc valable aussi dans le domaine non linéaire.



Les préconditionneurs ILUT(20) et ILUT(30) n'obéissent pas à la règle. Cependant, vu le petit nombre d'itérations, on suppose que l'on obtient, grâce à ces méthodes, une très bonne approximation de l'inverse de la matrice et que celui-ci devient indépendant de l'aspect des mailles. On peut vérifier, grâce à MATLAB[10], que ILUT(30) donne la décomposition LU exacte de la matrice K_T. Ceci permet de vérifier l'exactitude des prédictions théoriques et de la programmation en FORTRAN de la méthode utilisée.

En général, le tableau précédent montre que la puissance d'un préconditionneur est une fonction croissante de la mémoire qu'on lui attribue. On voit, par exemple, que si on passe de ILUT(5) à ILUT(10), on divise le nombre d'itérations moyen par un facteur légèrement inférieur à 4.

Comparaison ILU(0) et ILUT :
Il est très intéressant de comparer ILU(0) avec ILUT(8) parce qu'ils occupent une même quantité de mémoire (car $8*(2*lfil)*N \cong NNZ$). On remarque que la stratégie de seuil utilisée dans ILUT est plus efficace que la sélection des éléments suivant leurs positions (ILU(0)).
De plus, la méthode ILUT a l'air d'être légèrement moins sensible aux variations d'aspect des mailles que ILU(0).



Remarques :
Il faut cependant faire très attention aux conclusions que l'on pourrait tirer du tableau précédent. En effet :

1. Pour de gros problèmes, il est impensable d'utiliser l'inverse exact de la matrice comme préconditionneur. Cela coûterait beaucoup trop d'espace mémoire (voir tableau suivant). De plus, la méthode reviendrait à utiliser un algorithme de Gauss d'une manière très peu intéressante (il suffit de comparer le temps avec celui du solveur direct).

[10] MATLAB ® (© The MathWorks, Inc) est l'outil idéal pour étudier les propriétés des matrices. Tous les graphes de ce travail ont été tracés à l'aide de ce programme.

2. On pourrait penser que, plus le préconditionneur est puissant, plus vite sera effectué le calcul. C'est vrai dans ce cas particulier pour lequel le système possède peu de degrés de liberté mais c'est faux en général ! Pour des plus gros problèmes, nous verrons que le temps de calcul du préconditionneur devient très important si on choisit mal le paramètre *lfil*.
3. Parallèlement aux nombres du tableau précédent, il faut considérer le temps nécessaire pour effectuer une itération. En effet, celui-ci est d'autant plus grand que la matrice de préconditionnement est remplie. Plus haut, nous donnons le temps total pour effectuer le calcul. Regardons par exemple le cas de la poutre maillée par des éléments carrés. On constate que ILUT(30) effectue environ 40 fois moins d'itérations que ILU(0) ; pourtant, il n'y a qu'un facteur 2 entre les temps CPU respectifs.

- Mémoire et taille du préconditionneur :

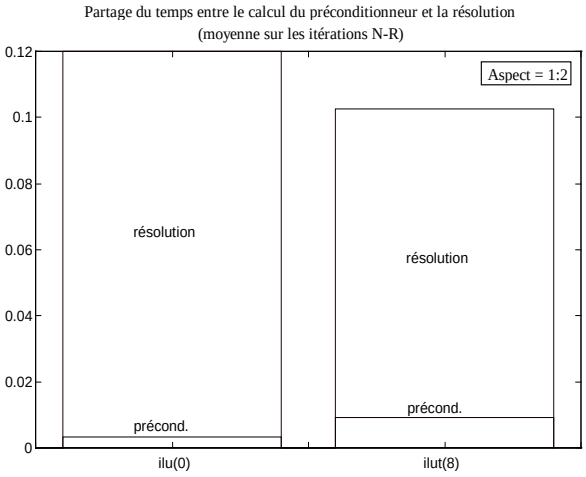
	ILU(0)	ILUT(5)	ILUT(8)	ILUT(10)	ILUT(20)	ILUT(30)
Mémoire	92508	89808	92508	94308	103308	112308
$NNZ2_{moyen}$	4576	2943	4584	5610	7278	7297
% él. nuls	5.08	3.27	5.09	6.23	8.09	8.11

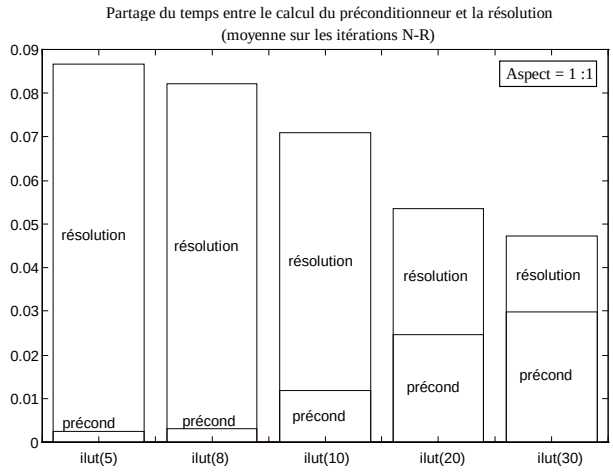
Un aspect important de ce travail est la diminution de l'espace mémoire nécessaire par l'utilisation d'un solveur itératif. Ici, nous voyons qu'aucune méthode ne permet de descendre en-dessous de la quantité utilisée par le solveur direct. Ceci est dû principalement à la faible taille du système et à la très faible largeur de bande de la matrice de raideur.
Remarquons que l'utilisation du préconditionneur optimum, ILUT(30), oblige de réserver le double de la mémoire utilisée par le solveur direct.

- Rapport CPU(préconditionnement) / CPU(résolution) :

	ILU(0)	ILUT(5)	ILUT(8)	ILUT(10)	ILUT(20)	ILUT(30)
1 : 1	0.029	0.030	0.040	0.200	0.863	1.700
1 : 2	0.028	0.016	0.097	0.120	1.170	1.481
1 : 4	0.011	0.014	0.043	0.087	1.344	1.431

Ce tableau reflète le temps passé dans la routine GMRES par rapport à celui passé pour construire le préconditionneur. Pour des petits problèmes comme ceux-ci, le rapport qui minimise le temps CPU est plus grand que 1. On passe plus de temps à préconditionner le système qu'à le résoudre. C'est le signe que l'utilisation d'un solveur itératif n'est pas intéressant.





Les temps CPU moyens de préconditionnement et de résolution sont représentés sur les graphiques ci-dessus. Celui de droite est relatif à la poutre moyenne. On remarque que le calcul du préconditionneur ILUT(8) est plus lent (il fait intervenir un algorithme de tri) que ILU(0) mais son efficacité est bien meilleure[11].
Le deuxième graphique représente la même chose pour les différentes valeurs envisagées du paramètre de remplissage. Le temps de calcul total moyen par itération diminue avec celui-ci. Pour des plus gros problèmes, il s'agira de déterminer la bonne valeur du paramètre pour obtenir un temps minimum

- Pas et itérations N-R :

	ILU(0)	ILUT(5)	ILUT(8)	ILUT(10)	ILUT(20)	ILUT(30)
1 : 1	51/125	51/125	51/125	51/125	51/125	51/125
1 : 2	55/136	55/138	67/169	67/169	67/169	67/169
1 : 4	73/174	73/196	71/172	71/172	71/172	71/172

Dans ce tableau, on a regroupé le nombre de pas de temps total et le nombre d'itérations, c'est-à-dire le nombre de fois qu'il a fallu résoudre un système de 300 x 300.
Pour des mailles carrées (1ère ligne), on voit que l'on a toujours le même nombre d'itérations. Le système est résolu avec une précision comparable à un solveur direct.
Un nombre de pas de temps différent que celui obtenu avec le solveur direct ne signifie pas nécessairement que la solution est fausse. On observe parfois une augmentation (3ème ligne), parfois une diminution (2ème ligne).

Augmentation du nombre d'itérations :
Ces différences proviennent de la stratégie automatique d'incrémentation temporelle utilisée dans METAFOR : lorsque la précision voulue (ici 10^{-3}) sur la norme du résidu d'équilibre est atteinte dans la boucle de Newton-Raphson, on passe au pas de temps suivant. Pour calculer l'incrément de charge, on multiplie l'incrément précédent par un facteur qui

[11] Si on obtient un temps de calcul total plus petit pour ILU(0), c'est donc uniquement à cause de la différence entre le nombre d'itérations total de N-R (voir tableaux). En moyenne, ILUT(8) est le plus puissant des deux.

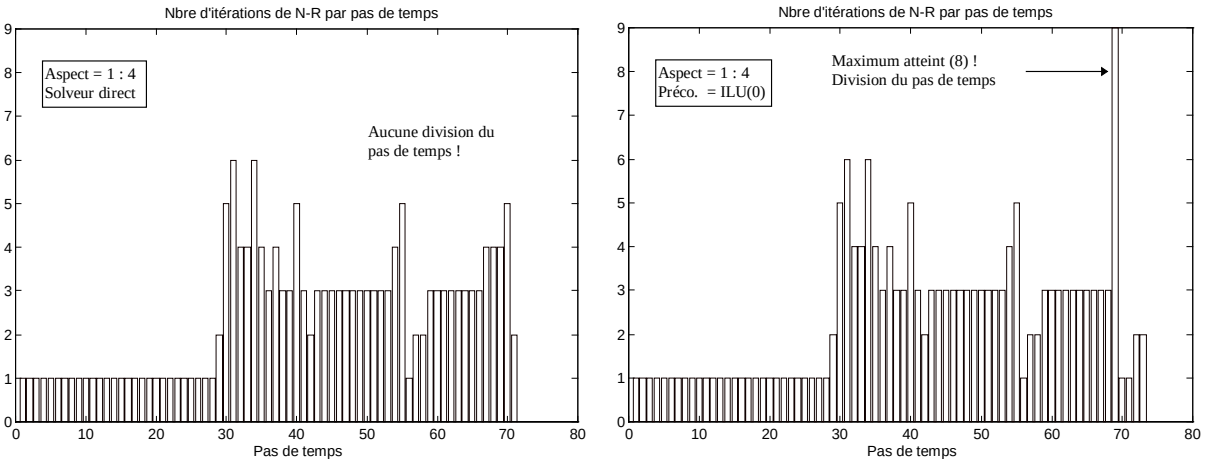
tient compte du nombre d'itérations du pas de temps précédent. Plus on a effectué d'itérations pour arriver à l'équilibre, moins grand sera cet incrément.

Dans le cas d'un solveur itératif, on n'essaye pas de résoudre exactement le système donné. Le but est de réduire la norme du résidu jusqu'à un certain point. Si ce point n'est pas atteint après un nombre d'itérations maximum (ici, fixé à 220), le solveur retourne la solution approximative qu'il possède et on continue le processus de Newton-Raphson.

En général, cette façon de faire augmente le nombre d'itérations de Newton-Raphson par rapport à celui obtenu à l'aide d'un solveur direct. Si on veut obtenir le même nombre de pas de temps et le même nombre d'itérations, il suffit généralement d'augmenter le nombre maximum d'itérations (it_{max}) du solveur et de diminuer la précision ε.

Il se peut aussi que le processus de Newton-Raphson ne converge plus (ou trop lentement) à cause de la résolution inexacte du système. Lorsque le nombre d'itérations de N-R dépasse une valeur fixée par l'utilisateur (ici 8), METAFOR effectue une division du pas de temps.

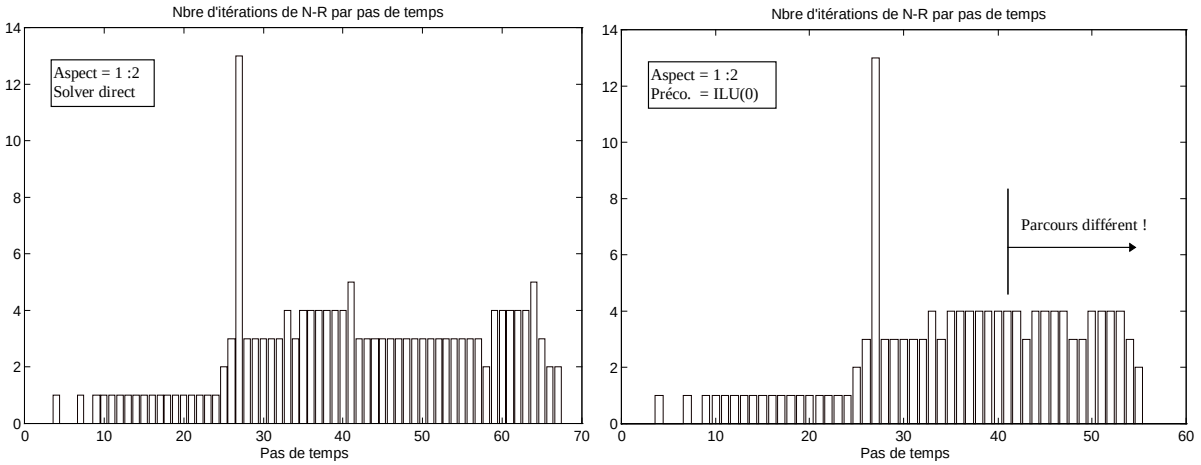
On voit très bien ce phénomène à la troisième ligne du tableau (73 pas à la place de 71). Les graphiques suivants illustrent le phénomène :



Au pas de temps 69, ILU(0) n'est pas assez puissant pour permettre au GMRES de diviser le résidu initial par 10^{-6} en moins de 220 itérations. Le processus de Newton Raphson ne converge plus assez rapidement et après la 8ème itération du pas de temps, la norme du résidu d'équilibre est toujours supérieure à la tolérance fixée. On recommence alors les calculs avec un pas de temps plus petit (ici il est divisé par trois). Il faut donc deux pas suppléméntaires pour arriver au bout du calcul.

Diminution du nombre d'itérations :

Dans d'autres cas, le nombre de pas de temps obtenu est surprenant : il est bien inférieur à celui effectué par le solveur direct (voir deuxième ligne du tableau, pour les deux premiers préconditionneurs, 55 au lieu de 67 !).



Malgré cette différence, la solution finale est identique à celle obtenue avec le solveur direct. On obtient trois décimales correctes pour toutes les grandeurs couramment étudiées. Nous verrons plus loin que ce parcours est aussi suivi lorsqu'on utilise une matrice symétrisée et le solveur direct.

- Influence du paramètre de restart (*m*) :

Le paramètre de restart peut influencer grandement la convergence vers la solution du problème. Pour les tests suivants, nous utilisons it_{max} = 1000 et le préconditionneur ILUT(8).

m	80	60	40	20	10
CPU [s]	10	10	10	101	/
it_{moy}	34.78	34.78	34.90	710.4	/
pas/it.	51/125	51/125	51/125	56/142	/

Aspect 1:1

m	80	60	40	20	10
CPU [s]	17	19	110	/	/
it_{moy}	47.39	47.43	425.4	/	/
pas/it.	67/169	67/169	69/173	/	/

Aspect 1:2

m	80	60	40	20	10
CPU [s]	30	135	/	/	/
it_{moy}	73.97	501.8	/	/	/
pas/it.	71/172	66/162	/	/	/

Aspect 1:4

Dans les tableaux ci-dessus, une barre oblique indique que METAFOR n'a pas trouvé de solution ou que la solution finale est très différente de celle trouvée avec le solveur direct (c'est le cas dans le troisième tableau). Ce phénomène est décrit en détail dans la prochaine section.

Choix d'après l'aspect des mailles :
On constate aussi que le paramètre de restart doit être pris d'autant plus grand que le problème est mal conditionné.

Difficultés dans le choix de m :
Les trois tableaux montrent bien la difficulté de choisir le paramètre de restart :
Si on le choisit trop petit (m = 10), on n'obtient jamais la solution avec ILUT(8). Si on veut garder un tel '*m*', le remède le plus simple consiste à augmenter la mémoire attribuée au préconditionneur.
Si le paramètre de restart est choisi trop grand (m>80), on gaspille inutilement de la mémoire parce que le GMRES n'effectue plus de restart.
Il faut donc utiliser un paramètre intermédiaire. Pour cette étude, nous avons choisi m = 80 pour pouvoir traiter les trois poutres avec le même paramètre. Cependant, si nous devions résoudre uniquement le premier problème, on aurait avantage à utiliser une valeur comprise entre 20 et 40.

Zone de transition :
Il existe une zone de transition entre la convergence et la divergence : lorsqu'on fait diminuer le paramètre de restart, on constate une augmentation impressionnante du temps de calcul et du nombre d'itérations moyen. En effet, le GMRES doit effectuer un grand nombre de restarts et le résidu diminue de moins en moins vite au cours des itérations du solveur.
Il est donc dangereux de se placer dans cette zone et il vaut mieux utiliser un grand paramètre de restart lorsque la mémoire le permet. D'autant plus que, pour un problème quelconque donné, on ne connaît pas la valeur limite inférieure qui assure la convergence.

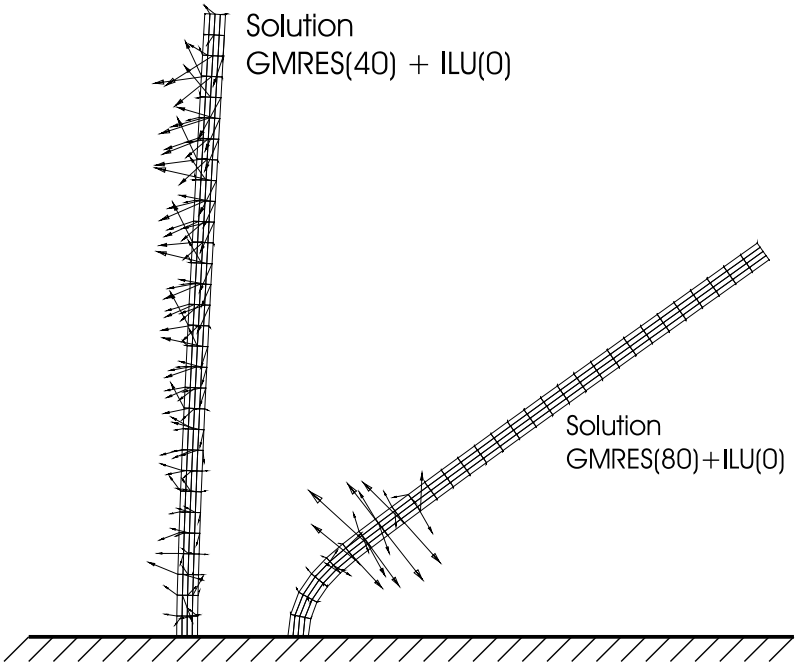
5.2.1.4 Convergence vers des solutions différentes

- Description du phénomène :

Lors des nombreux tests effectués pour cette étude, un phénomène très intéressant s'est manifesté à quelques reprises : dans des cas très particuliers, il arrive que le processus de Newton-Raphson converge vers une solution très différente de celle attendue.

Le dessin, page suivante, montre ce que l'on obtient.
La différence des deux déformées est très importante ; pourtant, elles vérifient toutes les deux le critère d'arrêt du processus de Newton-Raphson.
Nous avons tracé également les résidus d'équilibres correspondants. Pour la solution la plus fléchie, les résidus d'équilibres les plus importants sont très localisés. Pour l'autre, ils sont répartis ''uniformément'' sur toute la longueur.

Rien ne permet de désigner directement la solution physique du problème. On pourrait par exemple penser que la solution pour laquelle les vecteurs résidus sont les mieux répartis sur la structure constitue la meilleure approximation de la réalité. Cependant, la valeur moyenne de la norme des résidus d'équilibre est plus faible dans le cas fortement fléchi. Ce qui nous pousse à dire que celle-ci est une meilleure approximation de la réalité.



les 2 solutions obtenues pour le même problème (échelles des vecteurs identiques)

Considérons la poutre la plus mince et essayons de résoudre le problème en diminuant le paramètre *m* du solveur jusque 40. Vu les deux premiers tableaux de la section précédente, on s'attend à ce que METAFOR ne trouve plus la solution pour des valeurs proches de 40. Ce n'est pas ce qui se passe : le processus converge mais la solution finale est inattendue.

Analysons plus précisément ce phénomène étrange : pour essayer d'en trouver la cause, on a joué sur la tolérance du processus de N-R, sur le nombre d'itérations maximum du solveur et sur le paramètre de restart du GMRES :

Les deux dernières lignes du tableau suivant rappellent les résultats obtenus pour *m* = 80 et pour le solveur direct. Les valeurs en gras indiquent que l'on obtient la solution peu fléchie.

Solveur	it_{max}	*tol. N-R*	*pas/it.*	it_{moyen}
GMRES(40) + ILU(0)	220	10^{-3}	**34/38**	220
GMRES(40) + ILU(0)	220	10^{-6}	**0**	220
GMRES(40) + ILU(0)	5000	10^{-3}	**36/57**	3303
GMRES(40) + ILUT(8)	220	10^{-3}	**35/46**	214
GMRES(40) + ILUT(10)	220	10^{-3}	70/160	133
Solver direct	/	10^{-3} ou 10^{-6}	51/125	/
GMRES(80) + ILU(0)	220	10^{-3}	55/136	58.04

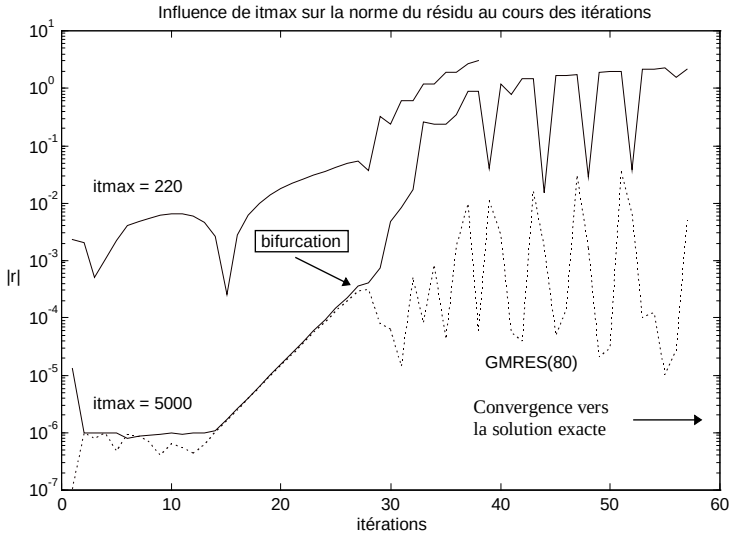
- Influence de la tolérance de N-R :

Si on utilise une tolérance 1000 fois plus faible (10^{-6}), la méthode de résolution utilisée ne converge plus. Par contre, le solveur direct n'est pas du tout influencé par ce changement. C'est donc la preuve que la solution donnée par le solveur direct est correcte. D'ailleurs, on la retrouve avec GMRES si on utilise un paramètre de restart plus important.

Nous verrons que les autres solveurs convergent toujours vers la bonne solution (lorsqu'ils convergent), quel que soit le préconditionneur utilisé.

- Influence de it_{max} :

La cause du problème est certainement le manque de précision obtenue lors de la résolution du système. On peut augmenter celle-ci en augmentant le nombre it_{max}.
Réduire le paramètre ε du solveur ne sert à rien. En effet, on obtiendrait une solution plus précise uniquement lorsque le processus itératif n'atteint pas un nombre it_{max} d'itérations.



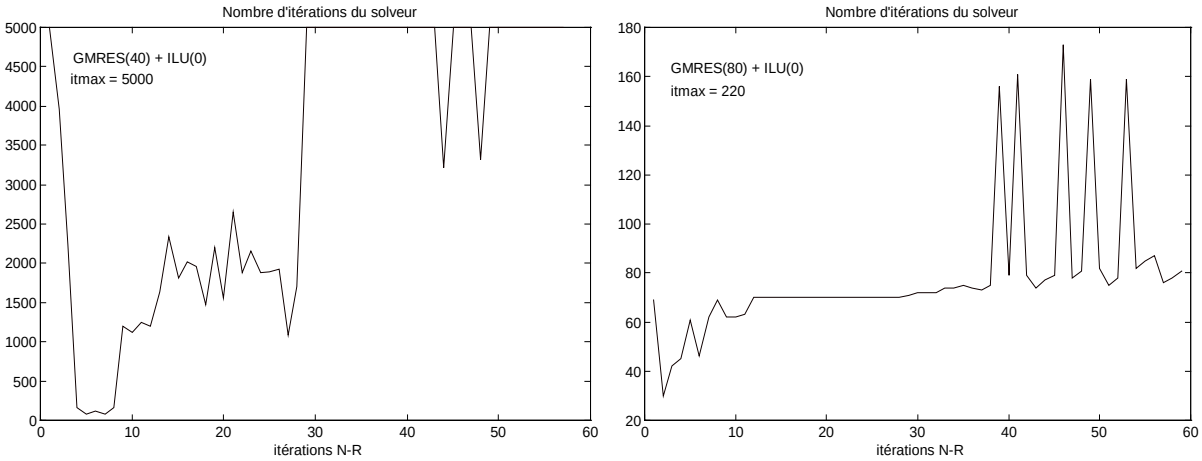
Le graphique ci-dessus montre que l'on obtient une meilleure approximation de la solution pour it_{max} = 5000. En particulier, pour la première itération, celle-ci est environ 100 fois plus précise.
On a représenté en pointillés l'évolution du résidu $\| r - K_T \Delta x \|$ pour GMRES(80) après chaque résolution itérative. Cette méthode fournit la solution exacte du problème (identique au solveur direct). On observe très distinctement un point de bifurcation survenant à la 28ème itération. C'est à ce moment que le système commence à fléchir fortement ou pas.

- Influence du paramètre de restart :

Les deux graphiques de la page suivante représentent le nombre d'itérations effectuées par le solveur pour chaque itération de Newton-Raphson pour le cas pathologique et le cas de référence.

On observe une augmentation brutale du nombre d'itération vers la 28ème itération de N-R. Les 5000 itérations sont très vite atteintes et le système cesse d'être résolu avec la précision demandée par l'utilisateur.
Ce phénomène est courant et, d'habitude, il n'y a pas lieu de s'inquiéter (si it_{max} est trop petit, le processus ne converge plus). Ici, malheureusement, le processus converge vers une autre solution !

Sur le graphique de droite, on remarque aussi la forte augmentation du nombre d'itérations mais, cette fois, la précision ε est atteinte en moins de 5000 itérations, vu le nombre plus important de vecteurs de Lanczos stockés en mémoire (m = 80).



Cet exemple montre encore l'utilité d'utiliser un paramètre de restart assez grand.

- Influence de la méthode utilisée :

Si on emploie un préconditionneur plus puissant comme ILUT(10), le problème est résolu. De plus, l'utilisation d'une autre méthode de résolution comme Bi-CG, par exemple, n'entraîne jamais à notre connaissance un tel phénomène pour les valeurs des paramètres utilisés dans ce travail (cfr. sections suivantes).

- Conclusions :

Il faut être prudent lorsqu'on obtient une solution par l'intermédiaire d'un solveur itératif parce que celui-ci ne résout pas toujours le système avec une grande précision (à cause du paramètre it_{max}). Si la tolérance du processus de Newton-Raphson est trop grande, on risque de converger vers une solution inacceptable qui peut passer inaperçue si on n'a pas une idée précise de la solution à obtenir.
Les mauvaises solutions apparaissent ici suite à l'utilisation du GMRES avec un trop petit paramètre de restart.

Le GMRES est un très bon solveur à condition de le munir d'un paramètre de restart assez grand. Dans le cas contraire, il ne peut pas rivaliser avec les autres méthodes.

Ce phénomène devra être étudié plus en détail si on veut inclure des méthodes de type quasi-Newton (BFGS, ...) dans METAFOR. Un remède pourra être apporté par la technique du LINE SEARCH décrite dans l'introduction de ce travail.

5.2.1.5 Résolution par BICG

Intéressons-nous maintenant aux solveurs provenant de la bi-orthogonalisation de Lanczos. Leur principal avantage est la mémoire réduite avec laquelle ils opèrent.

Nous utilisons les mêmes valeurs pour it_{max} et ε que pour le GMRES.

- Temps CPU :

[*s*]	ILU(0)	ILUT(5)	ILUT(8)	ILUT(10)	ILUT(20)	ILUT(30)
1 : 1	11.37	16.26	11.08	9.87	6.13	5.63
1 : 2	23.83	28.77	21.22	16.25	7.95	7.38
1 : 4	34.83	/	36.07	26.33	7.85	7.80

- Nombre d'itérations moyen pour résoudre un système :

	ILU(0)	ILUT(5)	ILUT(8)	ILUT(10)	ILUT(20)	ILUT(30)
1 : 1	40.94	90.34	36.5	23.97	2.27	1
1 : 2	79.98	133.8	55.58	37.63	2.302	1
1 : 4	128.6	/	130	77.23	2	1

Les observations faites à partir des résultats du GMRES sont aussi valables pour le Bi-CG.
Les résultats sont moins bons du point de vue du temps de calcul et du nombre d'itérations moyennes. De plus, il existe un problème qui n'a pas pu être résolu.

Question mémoire, on en utilise largement moins que pour le GMRES ; par exemple, pour ILU(0), on réserve 65868 flottants contre 92508 pour le GMRES. Cependant, on est encore loin des 52853 du solveur direct.

5.2.1.6 Résolution par BICGSTAB

La méthode BICGSTAB est censée améliorer les résultats du BICG. Nous allons voir que ce n'est pas toujours le cas.

- Temps CPU :

[*s*]	ILU(0)	ILUT(5)	ILUT(8)	ILUT(10)	ILUT(20)	ILUT(30)
1 : 1	/	15.60	25.13	/	6.02	5.70
1 : 2	35.62	39.95	19.68	15.45	7.48	7.38
1 : 4	50.62	/	45.20	33.52	7.57	8.35

- Nombre d'itérations moyen pour résoudre un système :

	ILU(0)	ILUT(5)	ILUT(8)	ILUT(10)	ILUT(20)	ILUT(30)
1 : 1	/	92.32	51.86	/	1.2	1
1 : 2	113.5	172.3	62.31	37.63	1.2	1
1 : 4	175	/	108.7	109	1	1

Pour le BICGSTAB, il y a trois problèmes qui ne peuvent plus être résolus et ceci, sans raison apparente. En effet, on peut comprendre que la méthode ait des difficultés pour résoudre le cas 1:4 avec le préconditionneur ILUT(5) puisqu'il est mal conditionné. Par contre, on ne voit pas pourquoi le cas 1:1 avec ILUT(10) et ILU(0) ne converge pas alors qu'il converge avec ILUT(5).

Ces constatations découlent des mauvaises propriétés de la bi-orthogonalisation de Lanczos. Il suffit que l'algorithme rencontre deux vecteurs presque orthogonaux pour perdre toute la précision gagnée lors des itérations précédentes du solveur.

Il serait intéressant de voir si la stratégie de 'look ahead' décrite dans la partie théorique du travail remédie à ce problème.

5.2.1.7 Résolution par CGS

La méthode CGS n'est qu'une variante des deux précédentes. Théoriquement, on s'attend à une convergence (ou une divergence) accélérée.

- Temps CPU :

[*s*]	ILU(0)	ILUT(5)	ILUT(8)	ILUT(10)	ILUT(20)	ILUT(30)
1 : 1	12.75	23.93	11.32	9.22	5.75	5.70
1 : 2	36.88	87.15	22.98	17.16	7.48	8.10
1 : 4	/	/	/	105.10	7.90	7.58

- Nombre d'itérations moyen pour résoudre un système :

	ILU(0)	ILUT(5)	ILUT(8)	ILUT(10)	ILUT(20)	ILUT(30)
1 : 1	50.57	113.10	38.83	23.46	1.48	1
1 : 2	136.6	189.8	77.05	47.33	1.2	1
1 : 4	/	/	/	159	1	1

Encore une fois, les conclusions relatives à l'aspect des mailles et l'efficacité des préconditionneurs sont identiques. Elles sont donc indépendantes de la méthode de résolution utilisée.

On peut remarquer que l'algorithme CGS est beaucoup plus sensible à l'aspect des mailles que les autres méthodes. En particulier, il est impossible de résoudre le cas 1:4 avec un paramètre de remplissage inférieur à 10.

5.2.1.8 Conditionnement de la matrice au cours des itérations

Nous avons vu dans la partie théorique du travail qu'il n'existe pas de théorèmes généraux caractérisant la convergence des solveurs itératifs si la matrice du système n'est pas symétrique et définie positive. Les matrices qui nous intéressent sont tout à fait quelconques : en effet, même dans le cas où on les symétrise, l'application du préconditionneur supprime directement cette propriété. Malgré cela, nous allons essayer de comprendre plus précisément les facteurs qui influencent la vitesse de convergence en se basant sur ces théorèmes.

Dans un premier temps, regardons s'il existe un rapport entre le nombre d'itérations du solveur et son nombre de conditionnement.

Pour éliminer l'influence du préconditionneur, on utilise l'algorithme GMRES non préconditionné.

- GMRES non préconditionné :

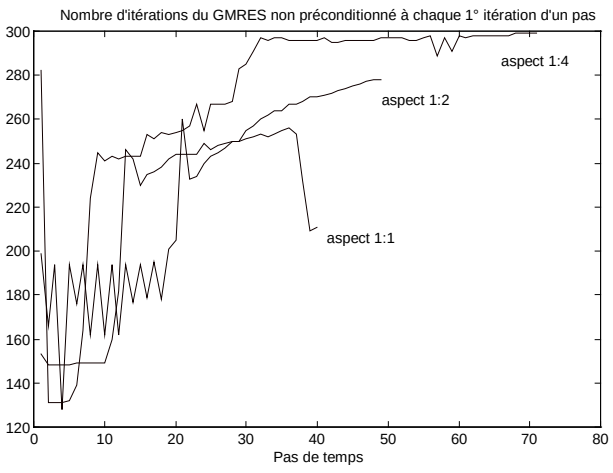
Pour obtenir la convergence, il faut utiliser un paramètre de restart maximum, c'est-à-dire 300.
Le tableau suivant regroupe les résultats :

	Temps CPU [s]	*pas/it.*	it_{moy}
1 : 1	65	51/125	224.1
1 : 2	81	55/136	244.3
1 : 4	112	71/171	283.7

Il est intéressant de voir que l'on arrive à obtenir la solution sans préconditionneur à condition de réserver un espace mémoire suffisant (équivalent à une matrice pleine de 300 x 300).
Ici aussi, on sent l'effet de l'aspect des mailles sur le nombre d'itérations moyen du solveur.

Ci-contre, on a représenté le nombre d'itérations du GMRES à chaque première itération d'un pas de temps.

L'aspect du problème a une grande influence. En particulier, on voit que dans le cas 1:4, la convergence est atteinte pour des valeurs très proches de 300.



Le tableau ci-dessous montre le nombre de conditionnement[12] en norme 2 de la matrice du système pour deux itérations particulières. L'itération critique est différente pour chaque cas : on a choisi l'itération de N-R pour laquelle on observe le plus grand nombre d'itérations du solveur (c'est-à-dire, le maximum de chaque courbe ci-dessus).
On a noté entre parenthèses le nombre d'itérations du solveur pour trouver la solution.

Aspect	Première itération	Itération critique
1 : 1	304 721 (153)	60 707 276 (257)
1 : 2	4 413 324 (199)	548 965 625 (278)
1 : 4	56 294 577 (282)	5 952 751 808 (299)

On voit très bien que le nombre de conditionnement du système dépend fortement de l'aspect des mailles utilisées. Pour la première itération, il y a un facteur 185 entre celui relatif aux mailles carrées et celui des mailles quatre fois plus longues que larges. Il est même possible de rencontrer des matrices dont le nombre de conditionnement dépasse le milliard ! La nécessité de préconditionner le système n'est plus à démontrer.

Le nombre de conditionnement n'est pas le seul facteur qui entre en jeu pour influencer le nombre d'itérations du solveur. Par exemple, dans le cas 1:1, à l'itération critique, on a une matrice moins bien conditionnée que pour la première itération du cas 1:4 et pourtant, le GMRES trouve plus facilement la solution.

- GMRES préconditionné :

Nous nous proposons d'étudier l'influence du préconditionnement du système d'équations par l'intermédiaire du calcul des valeurs propres. Dans le cas symétrique et défini positif, le nombre de conditionnement de la matrice peut être calculé par le rapport de ses valeurs propres extrêmes. Dans notre cas, on peut s'attendre à obtenir des valeurs propres complexes ou négatives.

Il faudrait donc essayer de trouver un nombre équivalent calculé grâce à celles-ci et qui mesure la facilité avec laquelle un système peut être résolu par la voie itérative.
Pour une matrice donnée, si on calcule le rapport $|\lambda_{max}| / |\lambda_{min}|$ (λ_i représentent les valeurs propres), on obtient une image de la dispersion des valeurs propres dans le plan complexe. Plus ce dernier sera proche de l'unité, plus les valeurs propres se regrouperont vers le point (1,0) et la matrice tendra vers la matrice unité.

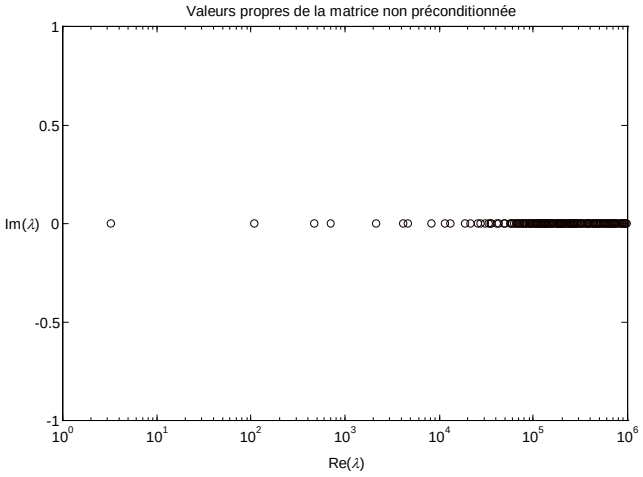
Les graphiques suivants montrent la distribution des valeurs propres de la matrice de raideur avant et après préconditionnement. Les 6 méthodes étudiés plus haut sont envisagées.

On considère le cas de la poutre la plus large (aspect 1:1) à la première itération du premier pas de temps.

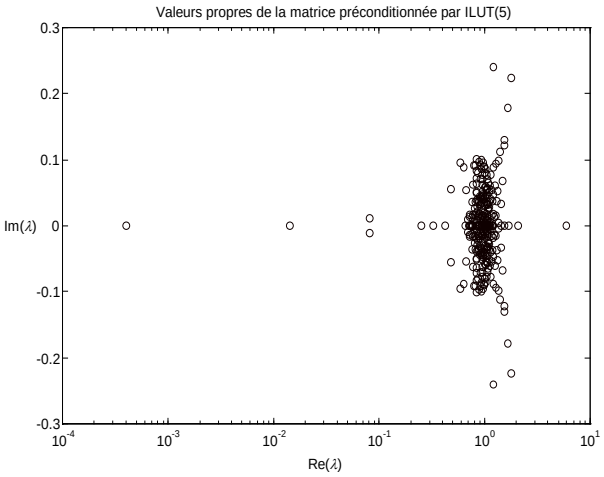
[12] Ils ont été calculés grâce à la fonction cond() de MATLAB

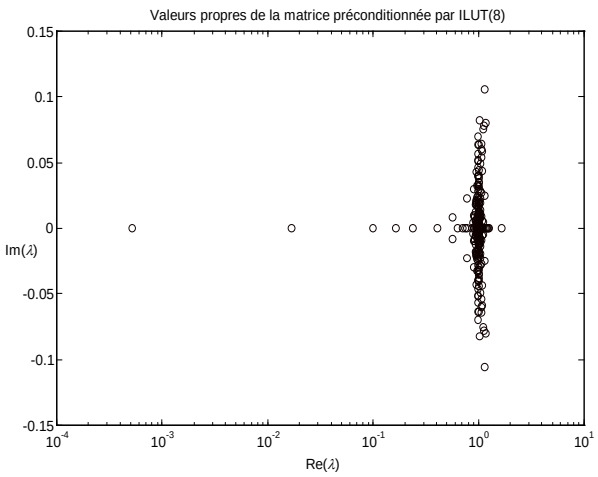
Les valeurs propres calculées sont toutes réelles et positives.
Elles sont réparties assez uniformément (l'axe x possède une échelle logarithmique) sur une très grande portion de l'axe réel.

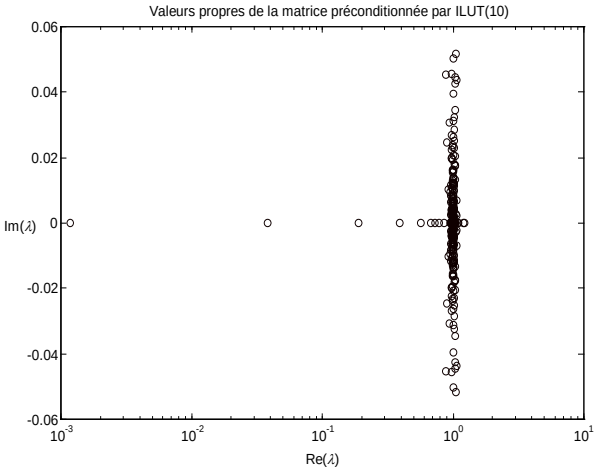
Le rapport des valeurs propres extrêmes vaut $3.05\ 10^5$.

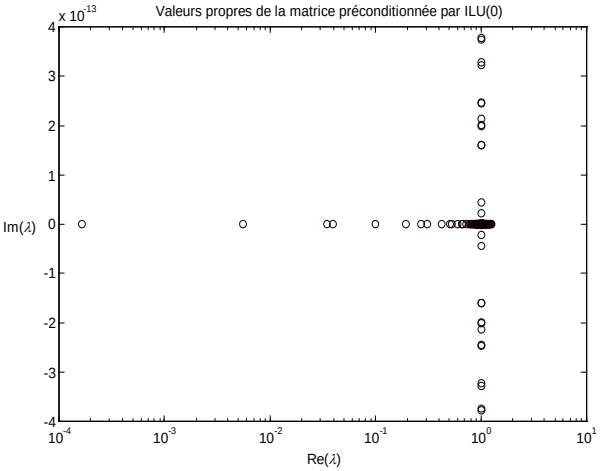


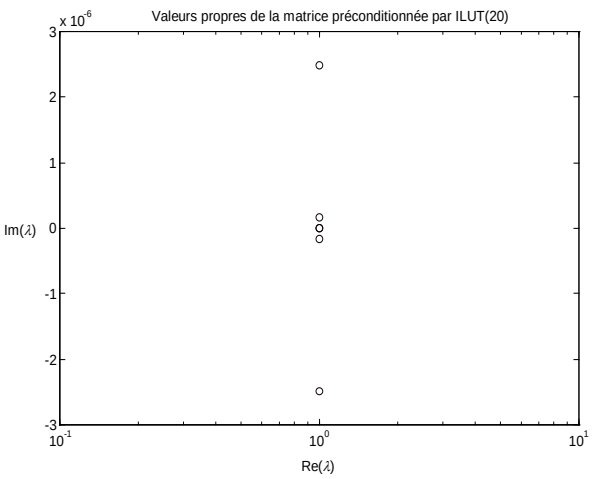
Intéressons-nous maintenant à la matrice préconditionnée $A\ (L\ U)^{-1}$:

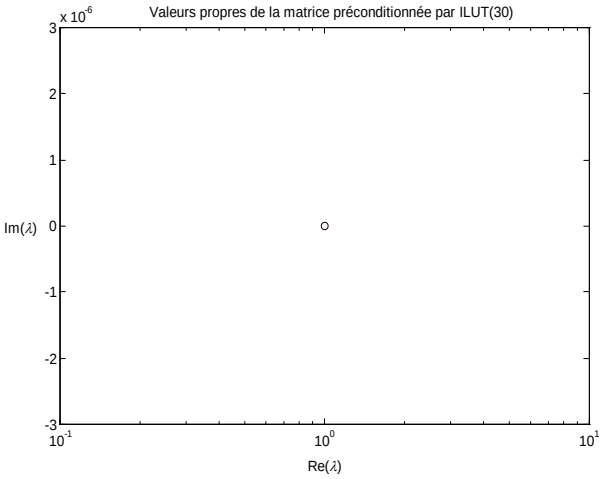










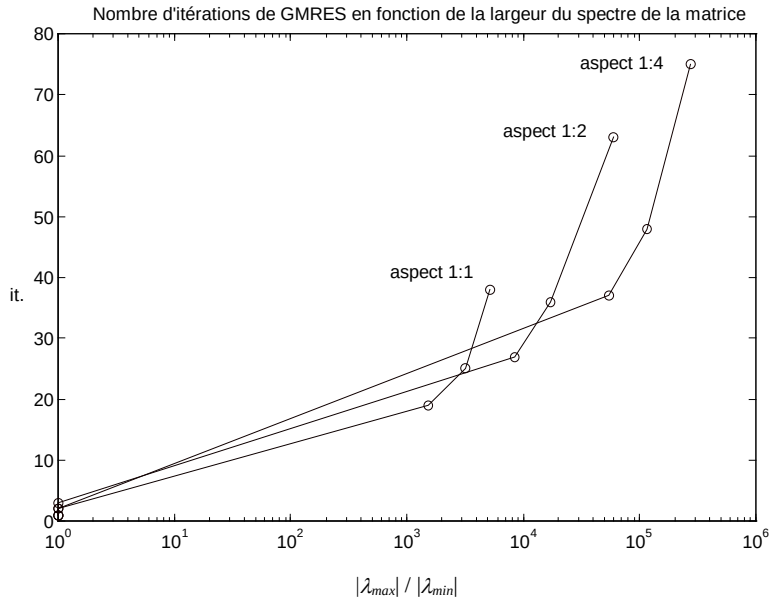


Le tableau suivant regroupe les résultats numériques relatifs aux 6 cas envisagés.

Méthode	ILU(0)	ILUT(5)	ILUT(8)	ILUT(10)	ILUT(20)	ILUT(30)
$\|\lambda_{max}\| / \|\lambda_{min}\|$	$7.66\ 10^3$	$5.21\ 10^3$	$3.16\ 10^3$	$1.51\ 10^3$	1	1
itérations	30	38	25	19	2	1

Comme annoncé, les préconditionneurs ont pour effet de réduire la largeur du spectre de la matrice. Les valeurs propres résultantes sont d'autant plus proches du point (1,0) que le préconditionneur est puissant. Dans le cas de l'utilisation de l'inverse de la matrice (ILUT(30)), toutes les valeurs propres sont confondues et unitaires.

On peut calculer $|\lambda_{max}| / |\lambda_{min}|$ pour les trois poutres étudiées et pour les préconditionneurs utilisés. Il suffit alors de mettre en rapport les nombres obtenus avec le nombre d'itérations du GMRES, ce qui fournit le graphe suivant :



En conclusion, le nombre d'itérations est une fonction croissante non linéaire de l'étendue des valeurs propres de la matrice du système dans le plan complexe.

5.2.1.9 Utilisation d'autres préconditionneurs

Pour les tests précédents, nous avons utilisé exclusivement les préconditionneurs ILU(0) et ILUT. Nous allons montrer brièvement que le choix des autres méthodes décrites dans le chapitre 4 n'améliore pas les résultats obtenus. Nous pourrons donc les ignorer dans la suite.

- Test de MILU(0) :

Pour rappel, cette méthode consiste à effectuer une factorisation incomplète ILU(0) en rejetant les termes ignorés sur la diagonale, pondérés par un facteur choisi par l'utilisateur.

Nombre moyen d'itérations (GMRES(80)) :

	ILU(0)	MILU(0)
1 :1	39.78	88.6
1 :2	58.04	251
1 :4	105.2	/

Le tableau ci-dessus montre l'inefficacité de la méthode vis-à-vis de la version non modifiée ILU(0). Nous avons pris un coefficient de pondération égal à 1 (conseillé par la plupart des auteurs). D'autres tests ont été effectués avec d'autres valeurs et ils ne sont pas plus concluants. Déjà pour la poutre maillée avec des éléments carrés, il faut plus du double d'itérations en moyenne pour trouver la solution du système. Pour des éléments deux fois plus longs que larges, nous avons dû augmenter le nombre d'itérations maximum admissible (it_{max}) jusqu'à 300 (on utilise 220 pour ILU(0)) afin d'obtenir la convergence du processus de Newton-Raphson. Cependant, celui-ci converge en 31 pas de temps au lieu de 67 : on retrouve le phénomène de bifurcation décrit dans la section 5.2.1.4. Pour le troisième cas, le préconditionnement est tellement mauvais que METAFOR ne passe jamais au deuxième pas de temps !

- Test de SSOR :

Les résultats fournis par cette méthode sont très décevants : pour les trois poutres, on n'obtient jamais une amélioration de la convergence. Autrement dit, même pour un maillage de carrés, on ne trouve jamais la solution du problème !

- Conclusion :

Les préconditionneurs SSOR et MILU(0) ne sont pas adaptés aux matrices rencontrées dans METAFOR.

5.2.1.10 Résolution du système symétrisé

Une pratique courante pour limiter l'espace mémoire utilisé est de symétriser la matrice de raideur. Pour le solveur direct, le temps de résolution est divisé approximativement par deux.
Utiliser une matrice de raideur symétrique ne garantit pas toujours la convergence du processus de Newton-Raphson. C'est le cas lorsque les contributions non symétriques sont trop importantes.

- Résultat du solveur direct :

Rappelons tout d'abord des résultats obtenus précédemment dans le cas symétrique

Aspect	temps CPU [*s*]	Pas	Itérations N-R	CPU_{moyen} [10^{-2} *s*]
1 : 1	4.45	51	125	3.560
1 : 2	5.77	67	169	3.414
1 : 4	5.88	71	172	3.418

En symétrisant la matrice, on obtient :

Aspect	temps CPU [*s*]	Pas	Itérations N-R	CPU_{moyen} [10^{-2} *s*]
1 : 1	3.68	51	125	2.95
1 : 2	4.20	55	136	3.08
1 : 4	5.43	73	178	3.04

La mémoire utilisée est réduite à 47140 flottants double précision.
Le temps CPU moyen n'est pas divisé exactement par deux parce que, pour des petits problèmes comme ceux-ci, les pré- et post-traitements prennent une part importante du temps de calcul, indépendamment du temps de résolution.

Le nombre de pas de temps et le nombre d'itérations sont différents du cas non symétrique. Pour la deuxième poutre, le processus de Newton-Raphson converge en 55 pas au lieu de 67. Cependant, si on compare les valeurs caractéristiques (déformation plastique équivalente, contraintes, ...), on retrouve les mêmes valeurs que dans le cas non symétrique à 10^{-3} près. Ces valeurs étaient déjà obtenues dans certains cas avec le GMRES et une matrice non symétrique.

- Résultats des solveurs itératifs :

L'utilisation simultanée de la matrice symétrisée et d'un solveur itératif ne permet pas de converger vers la solution. En particulier, il sera impossible de tester les solveurs symétriques. Nous reportons donc l'étude des algorithmes CG et SYMMLQ à plus tard (section 5.5.4)

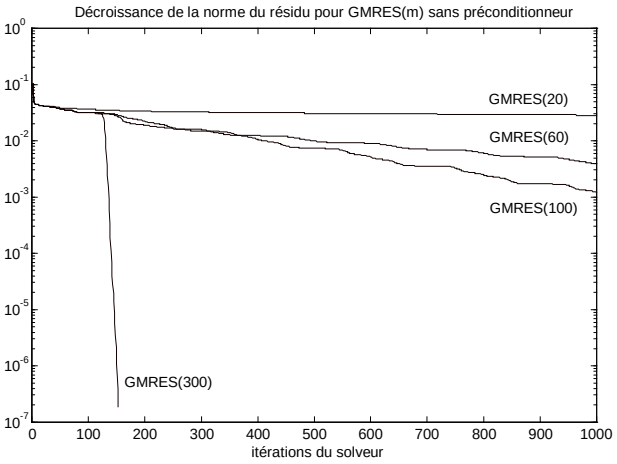
On peut essayer de résoudre le problème de cinq manières :
- Utiliser un autre solveur : tous les solveurs itératifs ont été utilisés, sans succès.
- Utiliser un préconditionneur puissant : même avec la méthode ILUT(30), on n'arrive pas à terminer le calcul.

- Réduire la précision (ε) du solveur : des valeurs de 10^{-3} à 10^{-14} ont été testées. On observe aucune amélioration de convergence. Même dans le cas de très petits ε, le nombre d'itérations maximum n'est jamais atteint. Il est donc inutile de changer ce dernier.
- Augmenter le nombre d'itérations maximum du processus de N.-R. : Même en prenant une limite de 20, METAFOR continue à diviser indéfiniment le pas de temps à partir d'un certain moment parce que le résidu augmente d'une itération à l'autre.

5.2.1.11 Comparatif des différents solveurs entre eux

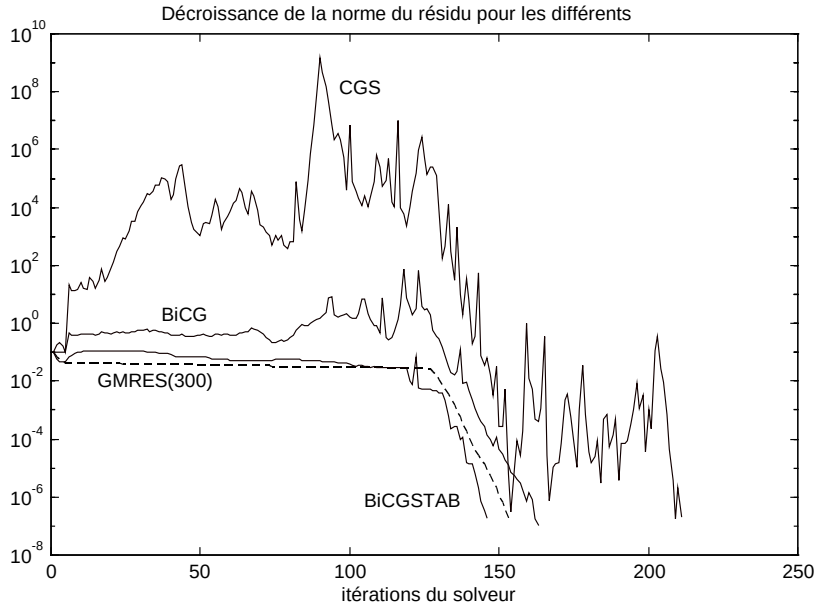
- Convergence au cours d'une itération quelconque :

Nous considérons la première itération du processus de Newton-Raphson lors de la résolution de la poutre la plus étroite (aspect des mailles 1:4). On n'utilise pas de préconditionneur.



On voit que la convergence dépend fortement du paramètre de restart. Au minimum, on obtient la convergence en 156 itérations de GMRES.
On voit très bien que la norme du résidu ne commence à diminuer significativement qu'à partir de 130 itérations. Si on utilise un paramètre de restart inférieur à cette valeur (par exemple 100), l'orthogonalité des vecteurs est perdue et cette diminution brutale ne s'observe pas.

Comparons maintenant GMRES avec les autres solveurs :

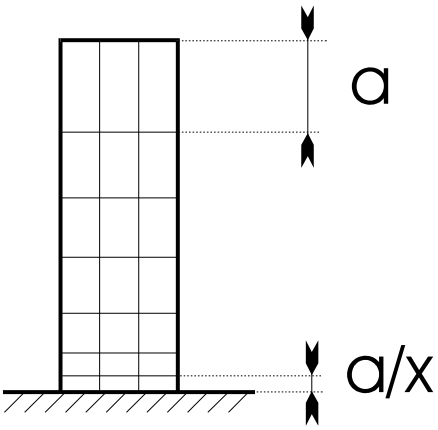


On remarque que la décroissance de la norme du résidu n'est plus monotone. Ceci était prédit par la théorie.
Le parcours suivit par CGS est très irrégulier : vers l'itération 150, il atteint presque la précision souhaitée mais il lui faudra plus de 50 nouvelles itérations pour franchir cette limite après une remontée jusqu'à 10^0 !
Le solveur Bi-CG est beaucoup moins irrégulier et, grâce à cela, on converge vers la solution bien plus tôt. Le nombre d'itérations obtenu est légèrement supérieur à celui du GMRES(300).
Pour ce cas particulier, la convergence la plus rapide est obtenue avec Bi-CGSTAB. Ce solveur possède une courbe de convergence stabilisée présentant encore moins d'oscillations, d'où le nom de la méthode.

N'oublions pas que cet exemple a été pris au hasard et on ne peut pas conclure que le Bi-CGSTAB est la meilleure méthode en se basant uniquement sur les courbes précédentes..

5.2.2 Influence des variations de grandeur de mailles

Nous allons maintenant tester l'influence de l'utilisation d'un maillage irrégulier pour notre poutre. Puisque les contraintes maximales se trouvent à la base de la poutre, près de son encastrement, il paraît logique de raffiner le maillage à cet endroit au détriment du reste.



Les dimensions de la poutre qui va nous servir de test sont les suivantes :

L = 300 mm
l = 40 mm

Nous utilisons de nouveau 30 mailles selon la longueur et 4 mailles selon la largeur.
On considère la méthode GMRES + ILUT(8).

Dans le tableau suivant, la notation 1:x signifie que les dernières mailles près de l'encastrement sont x fois plus aplaties que celles à l'extrémité de la poutre.

Rapport	1:1	1:3	1:10	1:30	1:80
it_{moy}	34.78	36.12	36.94	35.24	33.61

Contrairement aux essais précédents, on voit que le nombre d'itérations ne varie pratiquement pas en fonction de l'aspect des mailles. Ceci est certainement dû aux faibles rotations et aux faibles déplacements subis par les mailles les plus aplaties.

Si on raffine le maillage à l'extrémité des poutres au lieu de l'encastrement, on obtient des mailles aplaties qui vont subir un grand déplacement. Cependant, le même phénomène se produit : le nombre d'itérations moyen ne varie pas. On pourrait expliquer cela par la faible déformation subie par les mailles plates.

- Conclusion :

On ne peut pas conclure à un mauvais conditionnement de la matrice de raideur lorsque le problème présente des mailles aplaties. Seule l'expérience peut aider l'utilisateur d'un solveur itératif dans le choix des paramètres.

En général, le problème sera d'autant plus difficile à résoudre par un solveur itératif si le maillage contient des éléments aplatis qui subissent des grandes déformations et des grands déplacements.

5.2.3 Influence du nombre de degrés de liberté

- Introduction :

Traitons maintenant le problème en 3D pour pouvoir augmenter le nombre de degrés de liberté plus facilement. Pour étudier l'influence de celui-ci sans être gêné par les autres facteurs précédemment décrits, il est nécessaire de faire varier la profondeur de la poutre, un peu comme nous avons fait varier la section lors de l'étude de l'influence de l'aspect des mailles.
Nous allons devoir aussi modifier le type de sollicitation. En effet, les forces réparties non conservatives n'existent pas encore dans la version actuelle de METAFOR. Nous les remplacerons donc pas des fixations non nulles. Autrement dit, nous imposerons à un ensemble de points situés à l'extrémité de la poutre un déplacement donné. Ceci est équivalent à l'application d'une charge concentrée à l'extrémité de la poutre.

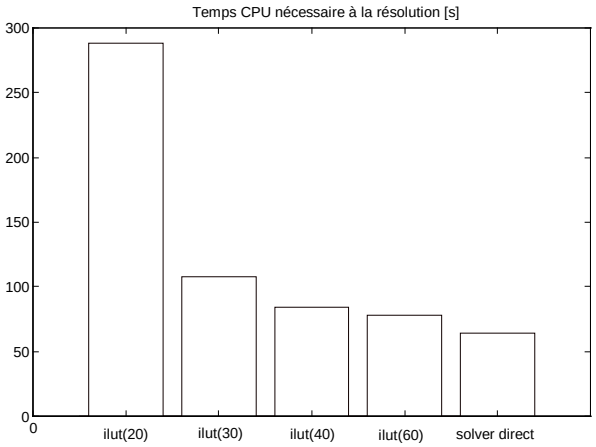
Pour ces tests, nous utilisons une poutre de 30 cm de long, 4 cm de large et une profondeur variable (1, 5 et 10 cm). On la maille avec des éléments cubiques de 1 cm de côté. Les trois essais successifs posséderont donc 1, 5 et 10 éléments sur la profondeur.

- Choix du préconditionneur :

La principale caractéristique des problèmes à trois dimensions est l'importance de la grande largeur de bande de la matrice de raideur. Celle-ci à tendance à augmenter fortement avec le nombre de degrés de liberté. Par contre, le nombre de termes non nuls par ligne reste toujours à peu près constant. Il en découle que le choix du paramètre de remplissage du préconditionneur ILUT n'est pas très facile. Une valeur intéressante en 2D peut être tout à fait inefficace en 3D.

Pour mieux comprendre ce phénomène, faisons varier le paramètre de remplissage dans le cas le la poutre de 1 cm de profondeur. Celle-ci constitue l'extension la plus simple du problème 2D traité précédemment.
Le tableau ci-dessous montre les résultats obtenus avec GMRES(60) :

lfil	20	30	40	60
temps CPU [s]	288	108	84	78
it_{moy}	208	48	14.28	1.98



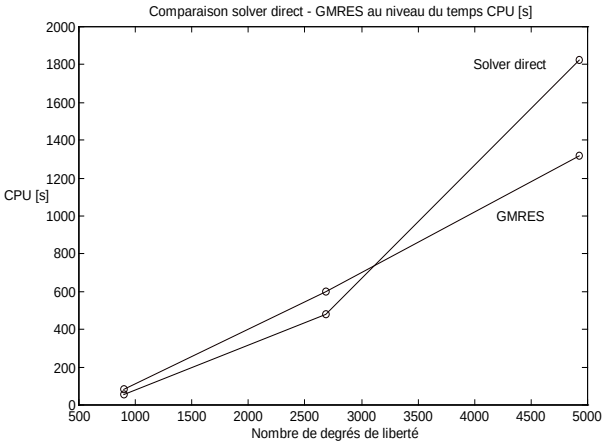
Nécessité de choisir *lfil* plus grand qu'à 2D :
Les préconditionneurs qui donnaient de bons résultats en 2D ne sont pas assez puissants pour résoudre un problème similaire en 3D. Par exemple, la méthode ILUT(20) permettaient d'effectuer une moyenne de deux itérations par résolution de système. Dans ce cas-ci, le GMRES doit en accomplir 208 !
La largeur de bande est la responsable : elle passe de 12 à 37.

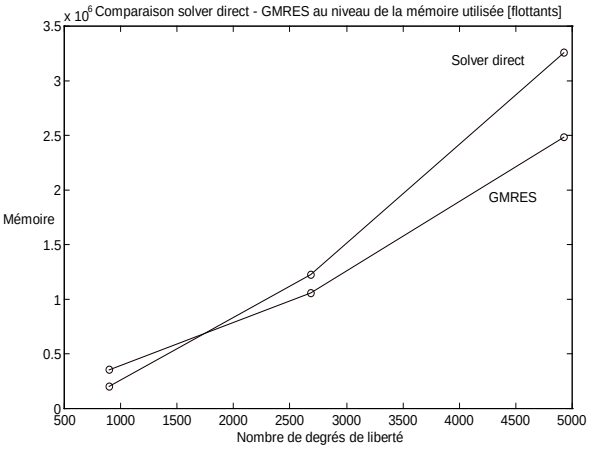
Incapacité de concurrencer le solveur direct :
Nous voyons que la structure ne possède pas encore assez de degrés de liberté pour obtenir de meilleurs résultats que le solveur direct. En effet, en augmentant le paramètre de remplissage, on tend vers l'inverse exacte de la matrice de raideur. Pour une valeur de 60, on n'effectue plus que deux itérations en moyenne. C'est la dernière valeur pour laquelle on peut encore parler de résolution itérative. Une valeur supérieure fournirait la vraie décomposition LU de la matrice et dans ce cas, l'algorithme direct est, bien sûr, plus rapide.

- <u>Influence du nombre de mailles sur la profondeur :</u>

Faisons varier la profondeur de la poutre. Nous utilisons la méthode GMRES(60) + ILUT(40). Celle-ci ne constitue peut être pas le meilleur choix. Cependant, nous allons voir que les résultats obtenus sont encourageants. Des graphiques similaires pourraient être obtenus avec d'autres méthodes. Nous étudierons cela plus précisément sur un autre problème (l'écrasement du cylindre épais).

Voici des courbes[13] qui nous nous encouragent à poursuivre plus loin cette étude des solveurs itératifs :





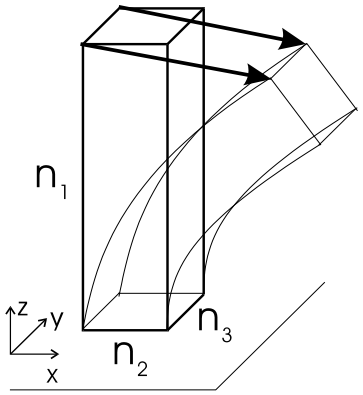
[13] Le mot 'courbes' est peut être mal choisi puisqu'on ne possède que trois points expérimentaux. Cependant, utiliser un nombre de points supérieur n'aurait changé en rien nos conclusions. Il était donc plus intéressant de consacrer notre temps à l'étude d'autres problèmes.

Intersections des courbes :
Il est très important de remarquer que les courbes ci-dessus se croisent pour un nombre de degrés de liberté donné. Si on augmente progressivement la taille du problème, le solveur itératif commence à rattraper le solveur direct. L'avantage de la méthode de stockage CSR se fait sentir vers 2000 d.d.l. A ce moment, la largeur de bande devient très importante et le nombre de zéros stockés par le solveur direct sous la ligne de ciel est approximativement égal à la place du préconditionneur et des vecteurs auxiliaires du solveur itératif.
Vers 3500 d.d.l., le GMRES devient plus rapide que le solveur direct et l'écart entre les deux ne va cesser de grandir lorsqu'on augmente encore la profondeur de la poutre.
A 5000 d.d.l., la matrice de raideur ne contient plus que 1.3 % d'éléments non nuls contre 5.2 % vers 1000 d.d.l.

En conséquence, grâce aux solveurs itératifs, il est possible de résoudre des problèmes largement plus gros qu'avec le solveur direct.

- Autres essais sur des poutres en 3D :

Tous les solveurs décrits dans ce travail ont été testés tout d'abord sur un problème similaire à celui qui vient d'être traité. Seules les dimensions de la poutre sont différentes.



La poutre étudiée à une longueur de 30 mm. Sa largeur et son épaisseur valent toutes les deux 10 mm.
Les nombres n_1, n_2 et n_3 indiquent le nombre d'éléments sur trois arêtes orthogonales

On impose à une de ses arêtes (voir dessin) un déplacement de 28 mm selon *x* et -10 selon *z*.

Nous regroupons dans le tableau ci-dessous quelques résultats intéressants :

n_1	n_2	n_3	d.d.l.	CPU GMRES	CPU direct	Mémoire GMRES	Mémoire direct
16	8	3	1724	11:15	11:24	887 825	849 499
25	10	8	7407	1h40:09	6h09:49	4 156 203	7 503 239
26	13	12	14170	2h43:00	/	8 183 359	23 405 154

Le dernier problème ne pouvait pas être résolu avant ce travail vu la quantité de mémoire requise (mémoire disponible sur la station de travail : 11 000 000 flottants double précision).

5.2.4 Conclusions

- Comparaison des solveurs :

Robustesse :
Au niveau de la robustesse, on constate que toutes les méthodes basées sur la bi-orthogonalisation de Lanczos présentent parfois des faiblesses imprévisibles. Ceci peut être très gênant puisqu'il est impossible a priori de savoir si la méthode va converger ou pas indépendamment du conditionnement du problème (il suffit que l'on rencontre deux vecteurs presque orthogonaux).
Le GMRES(*m*), quant à lui, est très intéressant à condition de choisir le paramètre de restart suffisamment grand . En effet, si celui-ci est trop petit, la vitesse de convergence devient vite médiocre. On préférera donc l'utilisation du GMRES avec un grand paramètre de restart lorsque la mémoire le permet.

Mémoire :
Du point de vue de la mémoire, les solveurs dérivés du Bi-CG sont très attractifs. Ils ne nécessitent que 8 ou 9 vecteurs à mémoriser en plus de la matrice du système et du préconditionneur.
Pour des problèmes 3D, les solveurs itératifs requièrent en général moins de mémoire que le solveur direct. Pour un ordinateur donné, de nouveaux problèmes deviennent accessibles !

Temps de calcul :
Pour tous les tests réalisés, le GMRES(*m*) est toujours le plus rapide des solveurs itératifs.
Nous avons vu qu'il existe une valeur du nombre de degrés de liberté pour laquelle les solveurs itératifs commencent à être intéressants vis-à-vis de leur homologue direct.
Pour des problèmes à deux dimensions, il ne faut pas espérer détrôner le solveur direct (il existe peut-être des cas particuliers mais nous ne les avons pas recherchés).

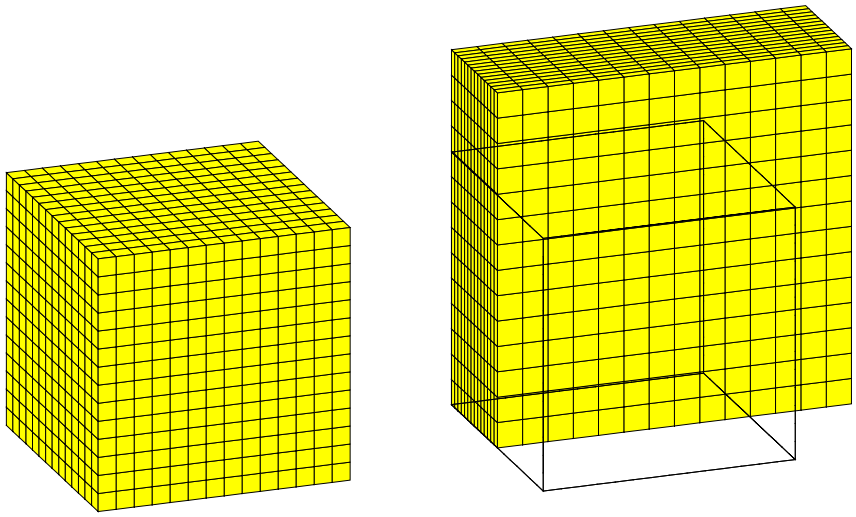
- Choix des paramètres :

Utiliser un solveur itératif n'est pas aussi facile qu'utiliser un solveur direct. Il est nécessaire de comprendre correctement l'influence de tous les paramètres pour choisir rapidement la meilleure méthode convenant au cas étudié.
Le GMRES(*m*) est désavantagé sur ce plan face aux méthodes de la famille du Bi-CG. Le choix d'un bon paramètre de restart (*m*) est difficile. En pratique, il vaut cependant mieux le choisir trop grand que trop petit.
Nous énoncerons à la fin de ce travail quelques règles simples pour fixer au mieux les paramètres du solveurs.

5.3 Compression d'un cube

5.3.1 Introduction

Ce problème va nous permettre de mettre en évidence les difficultés qui proviennent de la loi de comportement du matériau.
Nous avons déjà vu, lors de l'étude de la poutre, qu'un matériau élastoplastique pouvait poser des difficultés aux solveurs itératifs lorsque les zones plastiques commencent à s'étendre et à provoquer de grands déplacements. En effet, il existe de très grandes différences de raideur entre les régions élastiques et les régions plastiques. On observe alors une augmentation du nombre d'itérations du solveur.



On utilise donc un cube maillé uniformément comme représenté ci-dessus. On impose un déplacement à une des faces de manière à réduire son épaisseur d'un certain pourcentage.
Dans les tests qui suivent, nous utiliserons 80 % et 50%.
Le matériau utilisé possède une loi de comportement hyperélastique quasi incompressible.

5.3.2 Etude des matériaux hyperélastiques :

- Rappel théorique :

Pour rappel, les matériaux hyperélastique possède une loi de comportement où le tenseur contrainte est obtenu directement par dérivation d'une fonction potentielle [P5,P6]

$$\sigma = \frac{\partial \psi}{\partial C}$$

où σ est le tenseur de contrainte de Cauchy,
ψ est la densité d'énergie de déformation,
C est le tenseur de Cauchy-Green à droite.

En faisant l'hypothèse de quasi incompressibilité, on peut montrer que la densité d'énergie de déformation peut s'écrire sous la forme :

$$\psi = \psi\left(\bar{I}_1, \bar{I}_2, \bar{I}_3\right) = \psi^*\left(\bar{I}_1, \bar{I}_2\right) + c\, H(J) + \tfrac{1}{2}\lambda\left[G(J)\right]^2$$

où $\bar{I}_1, \bar{I}_2$ sont les deux premiers invariants du tenseur de Finger $F\,F^T$ (F est le tenseur gradient de déformation).
λ est le coefficient de pénalité qui permet d'approcher la transformation isochorique.
c est une constante
H et G sont des fonctions du jacobien du tenseur F (J = dét F).

METAFOR connaît quatre lois de matériaux hyperélastiques :
Modèle Néo-Hookien :

$$\psi^* = C_1\left(I_1 - 3\right)$$

Modèle Mooney-Rivlin :

$$\psi^* = C_1\left(I_1 - 3\right) + C_2\left(I_2 - 3\right)$$

Modèle de Hart-Smith amélioré :

$$\psi^* = C_1 \int e^{C_3\left(\bar{I}_1 - 3\right)^{C_4}}\, d\,\bar{I}_1 + 3\,C_2 \ln \bar{I}_2$$

Modèle de Gent-Thomas :

$$\psi^* = C_1\left(\bar{I}_1 - 3\right) + 3\,C_2 \ln \bar{I}_2$$

Nous considérerons, à titre de comparaison, une loi élastique avec un coefficient de poisson proche de 0.5 (ν = 0.49).

Les paramètres de chaque loi constitutive ont été calculés pour qu'elles représentent le même matériau (même comportement en petites déformations).

- <u>Difficultés pour les solveurs itératifs :</u>

La principale difficulté provenant de ces matériaux est issue du coefficient de pénalité λ. Celui-ci multiplie une fonction $[G(J)]^2$ qui vaut zéro lorsque J = 1 (déformation isochore) et qui croit rapidement lorsque J s'éloigne de cette valeur. Autrement dit, ce terme a pour effet d'amplifier le défaut d'incompressibilité par l'intermédiaire de l'énergie de déformation.
Si le paramètre λ est grand, on va voir apparaître des contributions très grandes dans la matrice de raideur et le nombre de conditionnement augmentera en conséquence.

Dans le cas élastique, on retrouve d'une manière naturelle le coefficient de pénalité [P6] :
La loi de Hooke s'écrit :

$$\dot{\sigma} = H\, D$$

où H, le tenseur de Hooke, peut se décomposer en

$$G\, H^D + \lambda\, H^V \text{ avec } \lambda = \frac{2\,\nu}{1 - 2\,\nu}\, G$$

et G est le module de cisaillement du matériau.

L'énergie de déformation peut alors se mettre sous la forme :

$$\psi = \psi^D + \frac{\lambda}{2}\left(\text{variation de volume}\right)^2$$

On retrouve une formule qui ressemble fortement à l'expression de l'énergie de déformation définie plus haut dans la théorie des matériaux hyperélastiques.

Les matériaux incompressibles vont donc certainement poser quelques problèmes lors d'une résolution par les méthodes itératives.

5.3.3 Test des différents modèles

Nous utilisons dans un premier temps la méthode GMRES(80) et ILUT(30) pour résoudre un cube de 6 éléments de côté comprimé à 80 %. La matrice de raideur est symétrisée.

	Solveur direct		Solveur itératif	
Modèle	pas/it.	CPU	pas/it.	CPU
Elastique	26/33	0:57	arrêt en t = 0 !	
Gent-Thomas	25/27	1:07	31/43	2:47
Hart-Smith amélioré	25/29	1:21	arrêt en t = 0.23 !	
Moonley-Rivlin	25/27	1:04	arrêt en t = 0.62 !	
Néo-Hookien	25/27	0:53	32/44	2:20

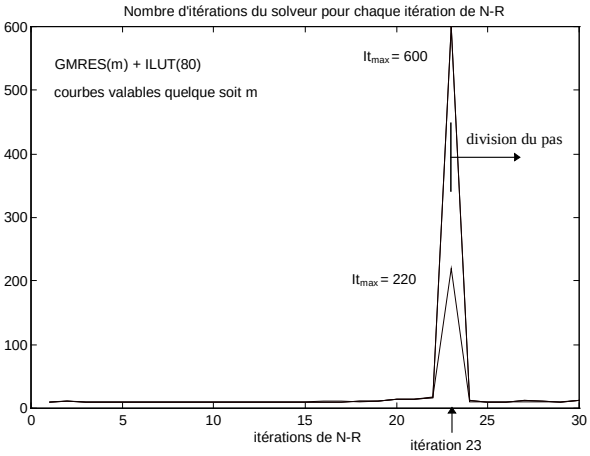
Les résultats du solveur itératif ne sont pas fameux. Seuls les deux modèles les plus simples (Gent-Thomas et Néo-Hookien) permettent d'obtenir une solution. Il faut cependant beaucoup plus d'itérations de N-R comparé au solveur direct. Les trois autres modèles provoquent beaucoup trop de divisions du pas de temps et l'incrément temporel devient trop petit pour pouvoir continuer le calcul. Le cas élastique est particulier : METAFOR n'arrive jamais à passer au deuxième pas de temps.

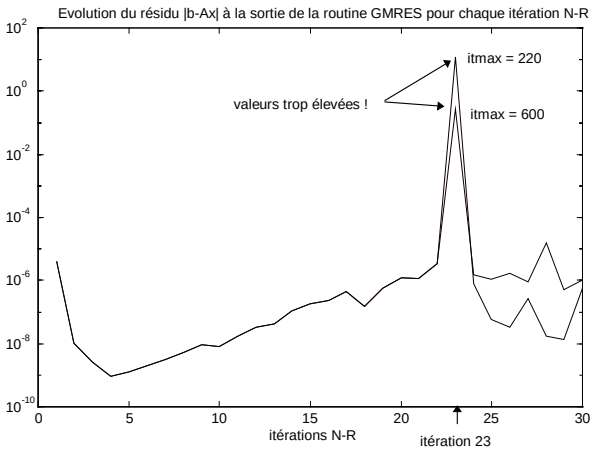
- Défaillance du préconditionneur ILUT :

Focalisons-nous sur le cube élastique. Si nous arrivons à le résoudre par voie itérative, les autres cas ne poseront pas de problèmes.
Pour comprendre ce qui se passe, nous allons augmenter la puissance du solveur itératif au détriment de la mémoire utilisée.
Utilisons cette fois ILUT(80). Dorénavant nous ne symétrisons plus la matrice de raideur.

Quel que soit le paramètre de restart utilisé (de 80 à 883), le calcul est totalement identique (la convergence n'est toujours pas obtenue).

Les graphiques suivants vont nous aider pour déterminer la cause de cette défaillance des solveurs itératifs.





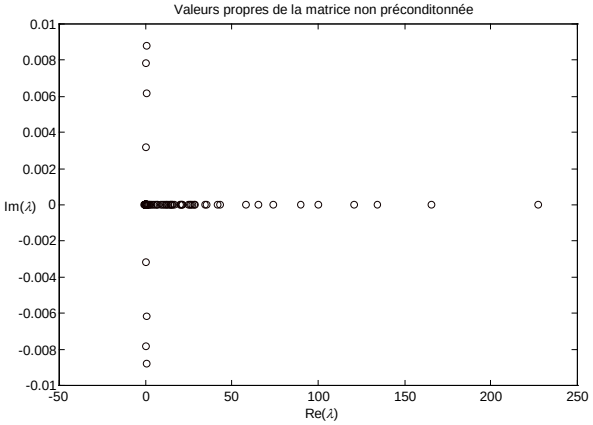
Remarquons tout d'abord que le problème ne venait pas de la symétrisation de la matrice de raideur.
On voit très bien sur les graphiques qui précèdent que le solveur itératif est incapable de trouver la solution du système de la 23° itération de Newton-Raphson. En effet, même en augmentant le nombre d'itérations maximum admissible, le résidu du système d'équations reste très grand. Si on augmente encore ce nombre, on s'aperçoit que le résidu estimé à chaque itération du GMRES ne correspond pas au résidu vrai $|b - A\,x|$ à cause de l'accumulation d'erreurs d'arrondis. Le GMRES diverge et si on ne l'arrête pas à temps, l'algorithme provoque un 'overflow'.

Ces dernières constatations prouvent que la matrice n'est pas assez bien conditionnée. Nous allons donc essayer de trouver le meilleur préconditionneur en faisant varier le paramètre de remplissage de celui-ci (on utilise m = 500):

lfil	0	5	80	100	> 150
pas/it.	25/28	/	/	/	25/28

Pour réussir à résoudre le problème de manière itérative, il faut donc utiliser soit un préconditionneur très puissant (>150) soit aucun préconditionneur (0) ! Toutes les valeurs du paramètre de remplissage comprises entre ces deux nombres provoquent une dégradation de la convergence du GMRES.

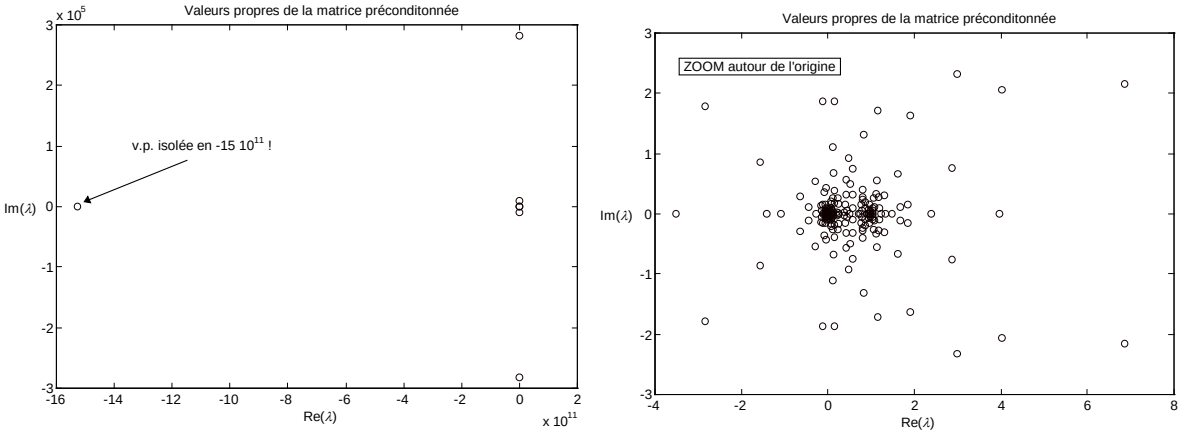
Analysons ce phénomène grâce aux valeurs propres de la matrice du système avant et après préconditionnement :



Le premier graphique est relatif à la matrice non préconditionnée. On remarque des valeurs propres négatives (difficile à voir sur ce dessin) et complexes. Ceci prouve bien que la matrice de raideur tangente est tout à fait quelconque.
On peut calculer :

$$|\lambda_{max}| / |\lambda_{min}| = 1.5\ 10^{6}$$

Les deux graphes ci-dessous se rapportent à la matrice préconditionnée : $A\,(L\,U)^{-1}$



On remarque deux phénomènes particuliers :

- Tout d'abord la présence d'une valeur propre isolée très loin de l'origine (-15 10^{11}). On calcule alors $|\lambda_{max}| \,/\, |\lambda_{min}| = 6.2\ 10^{15}$. Autrement dit, le spectre de la matrice préconditionnée est bien plus large que celui de la matrice initiale.
- Près de l'origine, les valeurs propres se rassemblent autour du point (1,0) comme nous l'avions remarqué lors de l'étude des différents préconditionneurs, mais aussi autour de l'origine.

En conclusion, le préconditionneur ILUT n'est pas infaillible. Contrairement à ce qu'on pourrait croire intuitivement, l'application d'une méthode de préconditionnement peut, dans certains cas très particuliers, dégrader les performances des solveurs itératifs.

5.3.4 Influence du coefficient de Poisson

- Introduction :

Nous étudions dans cette section des matériaux élastiques avec différentes valeurs du coefficient de Poisson. Nous venons de montrer que la valeur $\nu = 0.49$ entraîne des difficultés lorsqu'on fait subir au cube une compression de 80 %. Il faudrait tout d'abord s'assurer que le coefficient de poisson est bien la cause du problème.

- Influence de ν pour un cube de 6 éléments par côté :

Pour les tests suivants, nous comprimons le cube à 50% pour ne pas devoir diminuer excessivement la valeur du coefficient de Poisson.
Le tableau de la page suivante regroupe les résultats des tests.

	GMRES(60) ILUT(50)					solveur direct
ν	0.45	0.46	0.47	0.48	0.49	$\forall\ \nu$ [14]
pas/it.	23/18	25/22	30/29	/	/	23/18
it_{moy}	15.8	27.4	95.8	/	/	
CPU/it. [*s*]	2.7	3.5	4.5	/	/	3.2
CPU total	0:50	1:17	2:11			0:57 / 0.31*

* suivant que l'on symétrise ou non la matrice de raideur

Influence néfaste de l'incompressibilité :
Lorsque le coefficient de Poisson se rapproche de 0.5, le solveur itératif résout le problème de plus en plus difficilement. Le nombre d'itérations de Newton-Raphson et le nombre d'itérations du solveur augmente très fort. Pour $\nu = 0.48$, on observe des divisions incessantes du pas de temps et METAFOR ne converge plus.

Temps de calcul :
Pour $\nu = 0.45$, l'utilisation du solveur itératif entraîne un temps de calcul total plus petit que le solveur direct uniquement si on ne symétrise pas la matrice de raideur. Pour $\nu = 0.46$, le temps de résolution devient plus grand que dans le cas du solveur direct d'une part parce que le temps de résolution de chaque système d'équations est plus élevé et d'autre part parce que la résolution imprécise du GMRES entraîne un nombre d'itérations de N-R plus grand. Et cela empire avec l'augmentation de ν.

Mémoire :
La résolution itérative nécessite plus de mémoire (464019 flottants au lieu de 449986) vu le préconditionneur et le paramètre de restart utilisé. Si on effectue une symétrisation de la matrice de raideur, cette différence s'accentue.

- Influence du maillage pour $\nu = 0.49$:

Pour les tests suivants, ν est fixé 0.49 et on augmente progressivement le nombre d'éléments finis qui discrétisent le cube. Celui-ci est toujours comprimé à 50%.

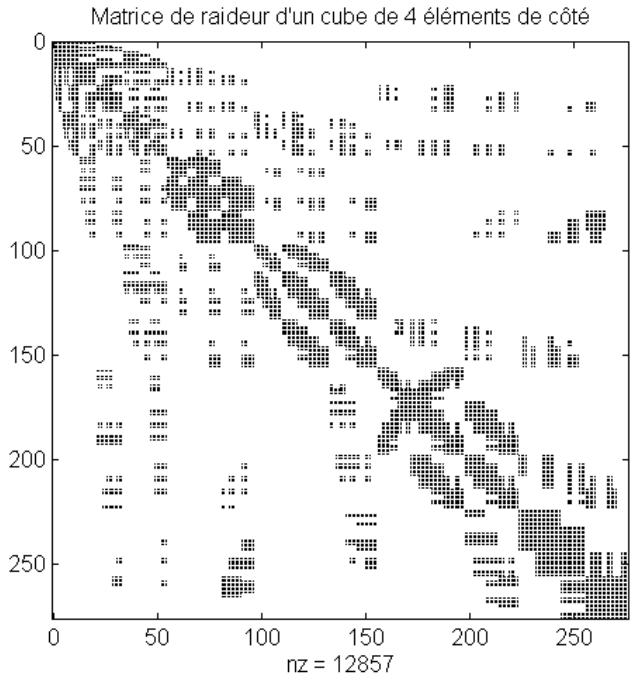
Soit *n* le nombre d'éléments par arêtes.

	GMRES(60) ILUT(50)			solveur direct
n	3	4	5	$\forall\ n$
pas/it.	25/26	29/35	/	25/26
it_{moy}	3.7	116	/	

Le nombre d'éléments finis joue aussi un rôle dans la convergence du GMRES. Plus celui-ci est grand, plus la résolution est difficile. Par contre, le solveur direct est assez robuste pour résoudre les trois cas considérés ci-dessus sans aucune variation du nombre d'itérations total de Newton-Raphson.

[14] En pratique , le solveur direct permet de choisir des valeurs de ν jusqu'à 0.499.

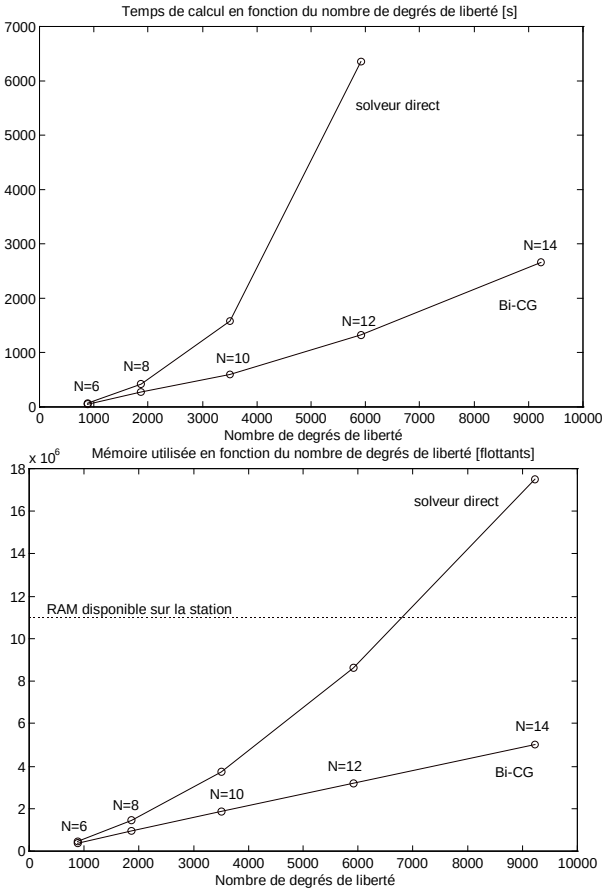
5.3.5 Influence du nombre de degrés de liberté



Le cube constitue la géométrie idéale pour montrer l'efficacité des solveurs itératifs. Si on le maille régulièrement, la matrice de raideur tangente résultante possède une très grande largeur de bande (la plus grande pour un nombre de degrés de liberté donné).

Le problème considéré est un cube hyperélastique. On utilise le modèle Néo-Hookien décrit dans le rappel théorique. Ceci nous permet d'éviter les problèmes rencontrés dans les sections précédentes. Le cube est comprimé à 50%.

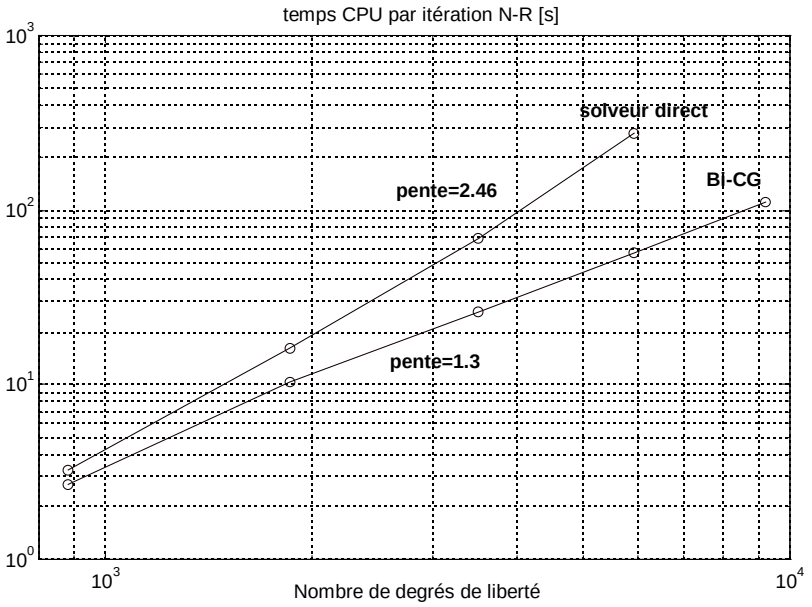
Nous utilisons cette fois-ci Bi-CG. Cela nous épargne le choix d'un paramètre de restart.
Appelons *N* le nombre d'éléments sur chaque arrête. On obtient, après plusieurs heures de calculs les graphiques suivants résumant les résultats :

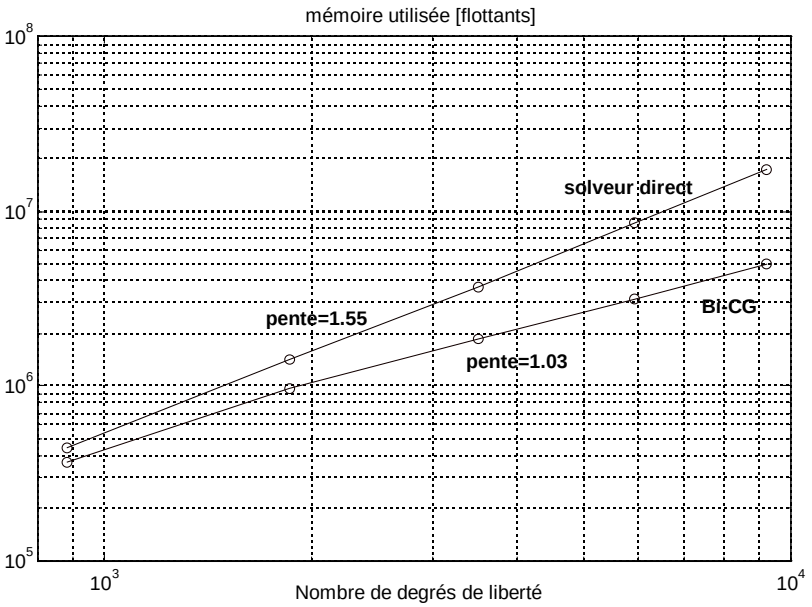


Les valeurs obtenues sont très satisfaisantes. Le Bi-CG effectue la résolution avec rapidité et économie de mémoire.

Deux remarques s'imposent :

1. Dans le cas du cube, comme nous l'avions dit, le solveur direct est désavantagé par la grande largeur de bande de la matrice de raideur. On constate que les points d'intersection des courbes de temps CPU et de mémoire utilisée apparaissent pour des tailles de problèmes assez petites (moins de 1000 d.d.l.) comparé au problème de la poutre 3D étudiée précédemment.
2. Un bref calcul montre qu'il serait possible de traiter comme cas limite un problème à 19950 d.d.l., c'est-à-dire, au maximum, un cube de 17 éléments de côté !
3. On voit très bien que la mémoire utilisée par le solveur itératif est une fonction linéaire de la taille du problème. Pour obtenir plus précisément la loi de variation du temps CPU et de la mémoire utilisée en fonction du nombre de degrés de liberté, on peut tracer les courbes précédentes en axes logarithmiques :





Considérons le temps CPU : on a bien une pente comprise entre 2 et 3 pour le solveur direct comme annoncé en théorie (N^2m opérations où N est la taille du système et m la largeur de bande). Quant au Bi-CG, sa pente est légèrement supérieure à 1.

Pour la mémoire du solveur direct on a une dépendance en $N^{1.55}$ (ce serait N^2 si on stockait toute la matrice). Par contre, pour le solveur itératif, la dépendance linéaire est confirmée.

5.4 Compaction de poudres

5.4.1 Introduction

- But de ce test :

Le problème traité dans cette section a été résolu la première fois pour la mise au point des lois de comportement des poudres dans METAFOR [C1]. Jusqu'à présent, il n'était pas possible de vérifier les résultats des tests de compaction simulés à deux dimensions par un modèle à trois dimensions à moins de considérer un maillage grossier. Nous allons donc tirer profit du gain de mémoire obtenu par l'utilisation d'un solveur itératif pour raffiner au maximum le problème 3D. On essaye donc de repousser les limites de la station de travail utilisée (11 millions de flottants double précision).

Ce problème permettra aussi de montrer l'importance du choix du paramètre de remplissage du préconditionneur. Les exemples décrits précédemment (voir la poutre 2D) pouvaient laisser croire que plus cette valeur est grande plus vite le solveur itératif converge vers la solution. En fait ceci n'est vrai que dans le cas de petits problèmes et c'est justement le signe que l'emploi d'un solveur itératif ne permettra jamais, dans ce cas, de concurrencer valablement le solveur direct.

- Théorie des poudres dans METAFOR :

Pour modéliser les poudres METAFOR utilise une loi de comportement élastoplastique avec le critère de Shima et Oyane définissant une surface de charge. Si ρ est la densité de la poudre, on écrit

$$f = \tfrac{3}{2}\, c(\rho)\, J_2 + f(\rho)\, I_1^2 - \sigma_0^2$$

où $J_2 = s_{ij}\, s_{ij}$ (s_{ij} est le déviateur du tenseur contrainte)
$I_1 = \sigma_{mm}$

On définit les deux fonctions $c(\rho)$ et $f(\rho)$ par les relations

$$c(\rho) = 1 + \beta \frac{1-\rho}{\rho - \rho_0} \quad \text{et} \quad f(\rho) = \alpha \frac{1-\rho}{\rho - \rho_0}$$

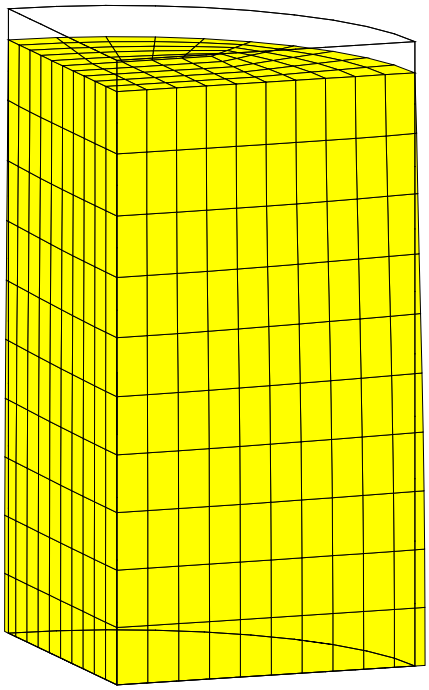
avec ρ_0 la densité de la poudre sous l'effet de la gravité,
α et β des paramètres fixés par l'expérience et caractérisant la poudre étudiée.

Ce critère est une généralisation de la surface de charge de Von-Mises. En effet, on la retrouve en imposant $\rho = 1$.

- Description du problème envisagé :

Pour effectuer les tests suivants, utiliserons les paramètres :

$E = 10000\ N/mm^2$ $\quad \nu = 0.25$ $\quad \rho_{initial} = 0.8$ $\quad \alpha = 0.2$
$\sigma_0 = 450\ N/mm^2$ $\quad \rho_0 = 0.3$ $\quad \beta = 2$



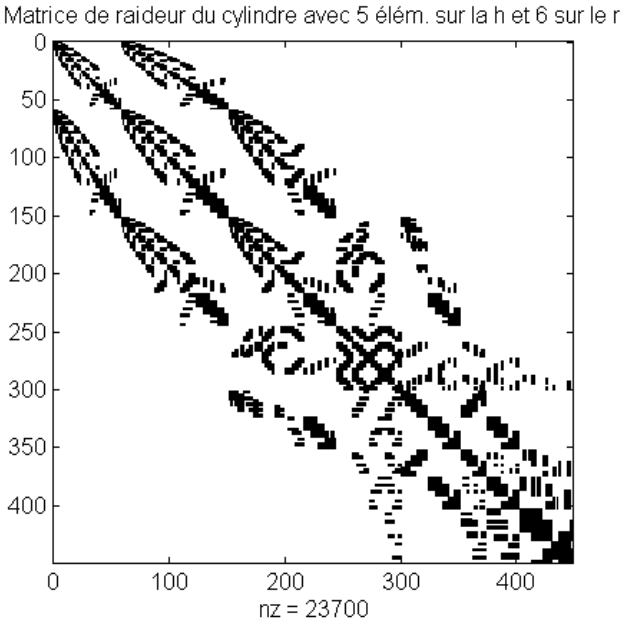
Le problème consiste à comprimer un cylindre rempli de poudre. Celle-ci est retenue par une membrane en caoutchouc très souple que nous ne modéliserons pas.
On impose un déplacement de la face supérieure du cylindre en l'empêchant toute modification de sa surface. Cette sollicitation provoque un état de contrainte non homogène.

Pour le calcul par la méthode des éléments finis, il est intéressant de modéliser ¼ de cylindre puisque la mise en charge possède deux plans de symétrie.

Dimensions : Hauteur = 20 *mm* ;
Rayon = 5 *mm*

Ce problème se prête très bien à une résolution par un solveur itératif :

- Les mailles sont cubiques et ne s'aplatissent pas trop au cours du calcul.
- La structure s'étend dans les trois directions de l'espace, ce qui laisse prévoir une grande largeur de bande (voir représentation ci-contre).

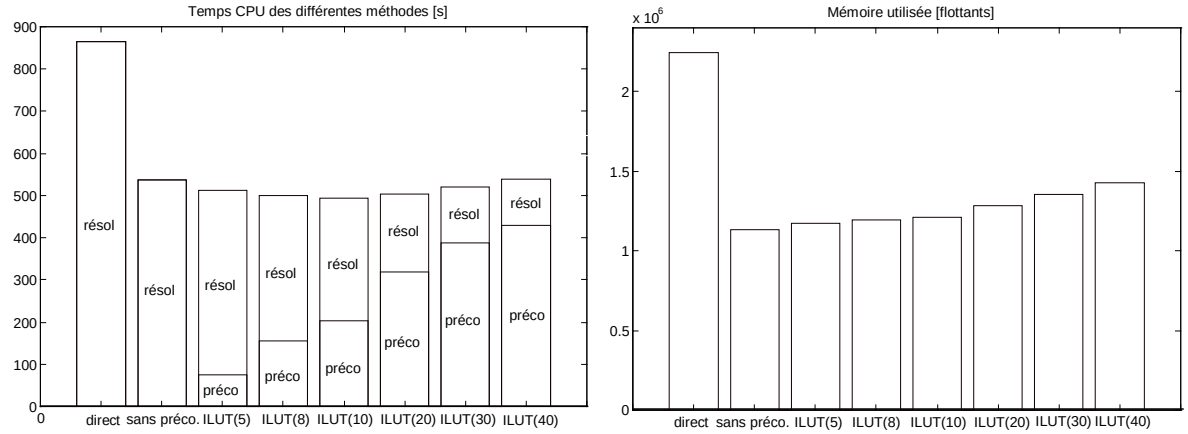


5.4.2 Influence du choix du paramètre de remplissage

Dans un premier temps, nous comprimons le cylindre de 5% (une compression plus importante ne change rien aux résultats à part le nombre de pas de temps). Le maillage utilisé est assez grossier. Le rayon et la hauteur sont tous les deux discrétisés par 10 segments. Cela nous donne 2419 dégrés de liberté et une hauteur de colonne moyene (SKYLINE) de 257.

Nous utilisons la méthode GMRES(60) + ILUT(*lfil*). Le paramètre de restart peut paraître excessivement élevé. Cependant, cette valeur permet de résoudre le système rapidement sans aucun préconditionnement.

Le graphique suivant montre le temps de calcul utilisé pour le préconditionnement et pour la résolution des systèmes d'équations.



Temps CPU :
On remarque qu'il existe un temps minimum pour une valeur intermédiaire du paramètre de remplissage (*lfil* = 10). Dans ce cas, les différences entre les valeurs ne sont pas très marquées parce que les systèmes d'équations rencontrés ne posent pas trop de problèmes. En particulier, on voit qu'il est possible de trouver la solution sans aucune méthode de préconditionnement
Si on considérait un problème plus gros, les différences s'accentueraient et le minimum serait plus marqué.
Ce minimum résulte de deux facteurs : d'une part, le temps de préconditionnement qui augmente au fur et à mesure que l'on augmente le paramètre de remplissage ; d'autre part le temps de résolution qui diminue avec la qualité du préconditionnement.

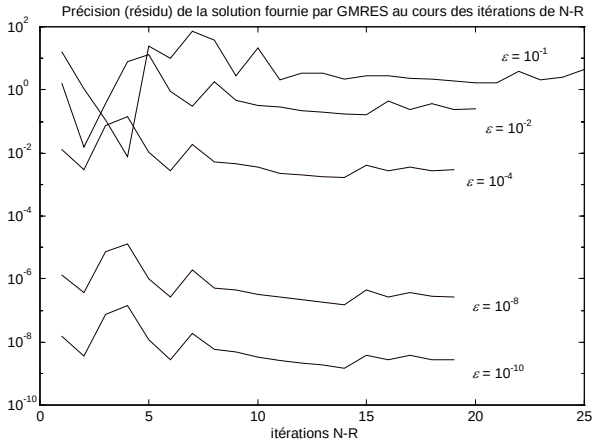
Mémoire :
Du point de vue de la mémoire, le GMRES utilise largement moins de mémoire que le solveur direct, quel que soit le paramètre *lfil*.
Le choix du paramètre de restart n'a pas beaucoup d'influence. En le réduisant à 30, on ne gagne que 100 000 flottants et on perd en moyenne 20 secondes sur le temps de calcul. En général, il vaut mieux se mettre en sécurité et utiliser un grand paramètre de restart.

5.4.3 Influence de la précision du solveur

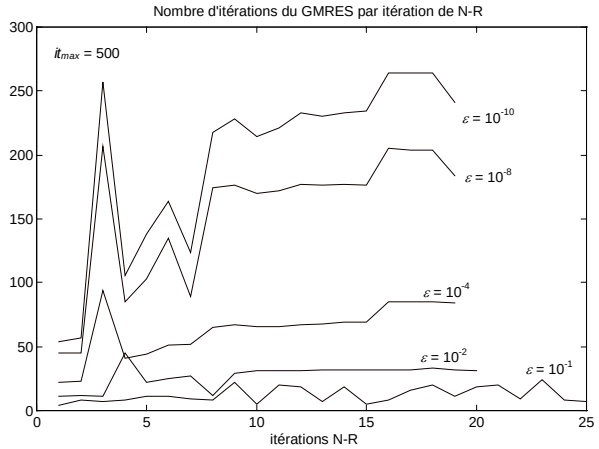
Le paramètre ε donne le rapport entre le résidu $|b - A\,x|$ initial et celui à la sortie de la routine du solveur pour autant que le nombre d'itérations maximal admissible (it_{max}) ne soit pas atteint.
Son choix est important : une grande valeur permettra une résolution rapide mais imprécise qui entraînera peut-être une divergence du processus de Newton-Raphson. Par contre, en utilisant une valeur très petite, on tend à trouver la solution du solveur direct et à entraîner une convergence identique (même nombre de pas de temps).

Remarquons que la précision de la solution finale (valeurs des contraintes en fin de calcul par exemple) est influencée par le choix de ε. Cela peut paraître étrange puisque la précision de la solution est une fonction de la tolérance sur le résidu d'équilibre (fixée par défaut à 10^{-3}). Cependant, pour une tolérance fixée, mieux on résout le système (par l'intermédiaire d'un ε très petit), plus le résidu d'équilibre à la dernière itération d'un pas de temps est faible. Par exemple, il n'est pas rare d'obtenir une norme de résidu d'équilibre de 10^{-5} ($<10^{-3}$) avec le solveur direct et tout juste 10^{-3} avec le solveur itératif muni d'un grand ε. Nous étudierons plus précisément ce phénomène dans le cas du godet superplastique (section 5.6).

Choisissons GMRES(30) et ILUT(10). Les graphiques suivants montrent l'histoire du calcul pour différentes valeurs de la précision. On utilise un nombre d'itérations maximum de 500 pour le solveur pour éliminer l'effet de ce paramètre.

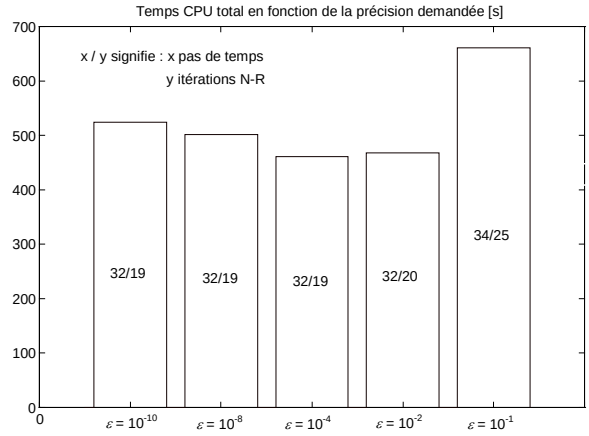


Courbes de précisions :
Les courbes sont tout à fait similaires à part celle relative à $\varepsilon = 10^{-1}$. Cette valeur est tellement élevée que METAFOR doit effectuer une division du pas de temps au début du calcul.



Nombre d'itérations du solveur :
On voit sur le graphique ci-contre que ce nombre est très influencé par la précision demandée.

Evidemment, demander une précision plus grande se paye plus cher mais on peut en tirer profit au niveau du nombre d'itérations total du processus de Newton-Raphson, comme le montre la figure suivante.



Temps CPU total :
Si la précision demandée est trop faible, le processus de Newton-Raphson ne converge plus aussi facilement et METAFOR doit effectuer des divisions du pas de temps.

Il existe donc une valeur optimale, inconnue a priori, pour le paramètre ε (ici 10^{-4}).

5.4.4 Comparaison des modèles 2D et 3D

- Introduction :

Utilisons maintenant le GMRES pour traiter un problème beaucoup plus gros. On s'attend à obtenir des résultats intéressants vu les temps de calculs obtenus pour le cube (section 5.3.5).
On aimerait comparer les solutions obtenues en traitant le problème de compaction tout d'abord en 2D par le solveur direct avec une hypothèse d'axisymétrie, ensuite en 3D avec un solveur itératif.

- Limites du solveur direct :

Considérons un premier cas : 25 éléments sur la hauteur 14 éléments sur la rayon.
On obtient un système de 11756 inconnues avec une largeur de bande moyenne de 508.

	It/pas	temps CPU	Mémoire
GMRES(60) ILUT(30)	32/17	43:41	6 559 168
solveur direct	/	/	18 256 902

Le solveur direct n'arrive pas à résoudre le problème par manque de mémoire. Il est obligé de stocker 4.3 % de la matrice de système contre 0.6% pour le solveur itératif. Cette différence est due aux zéros sous la ligne de ciel. Ceux-ci représentent 5.1 millions de nombres flottants double précision (ce qui représente environ 39 Mo de RAM !).

Le temps CPU est assez petit (moins d'une heure). Il serait intéressant de le comparer avec celui du solveur direct lors de l'acquisition d'une station de travail plus puissante.

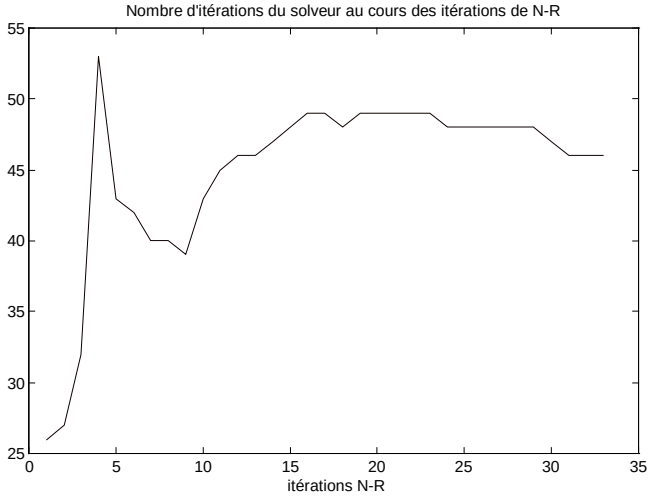
- Problème de taille maximale :

Pour comparer les solutions 2D et 3D, on utilise, bien entendu, une discrétisation équivalente dans les deux cas. Celle-ci a été déterminée pour que le problème occupe toute la mémoire disponible lors du test 3D.

On utilise le maillage suivant : 32 éléments sur la hauteur,
16 éléments sur le rayon.

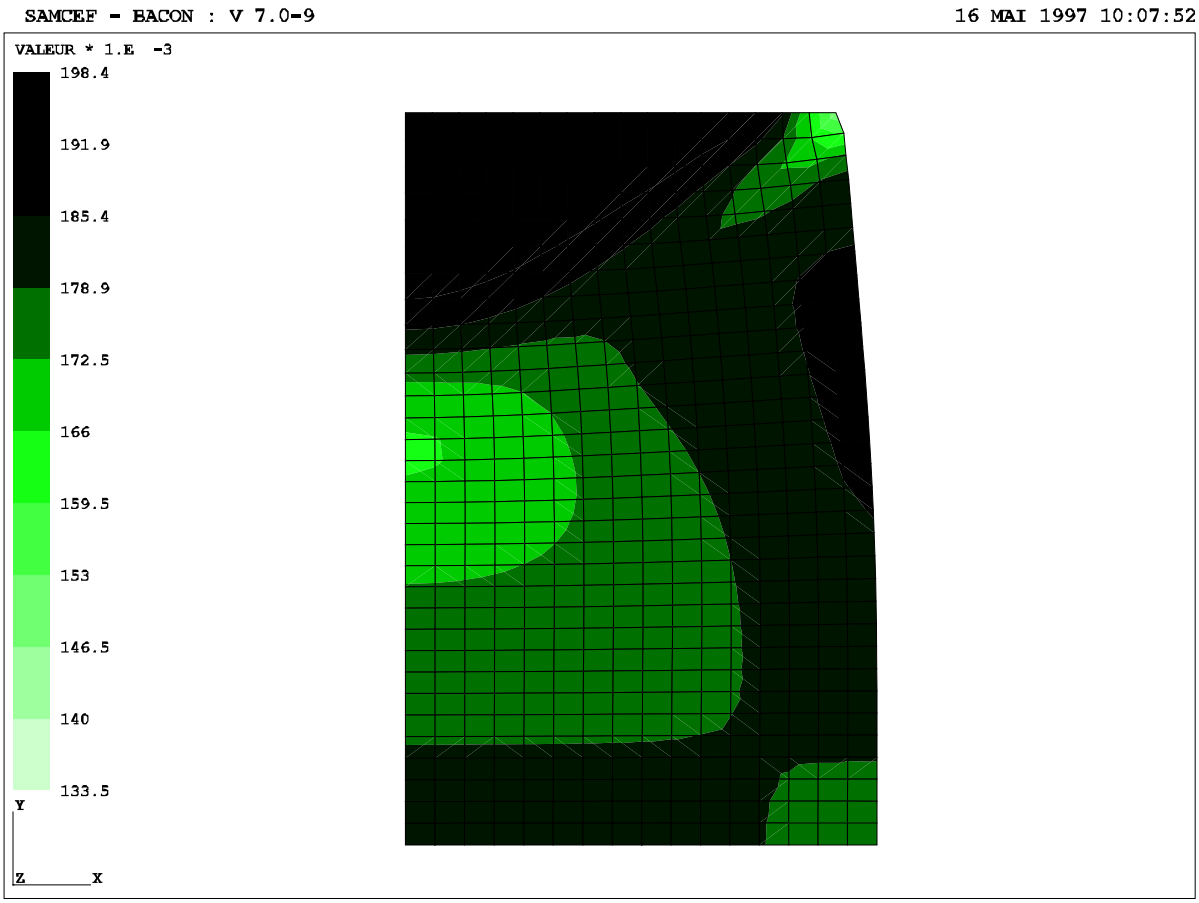
Comme solveur, on considère le GMRES(55) et ILUT(33). Ceci fournit un problème à 19527 d.d.l., c'est-à-dire le plus gros problème étudié dans ce travail.

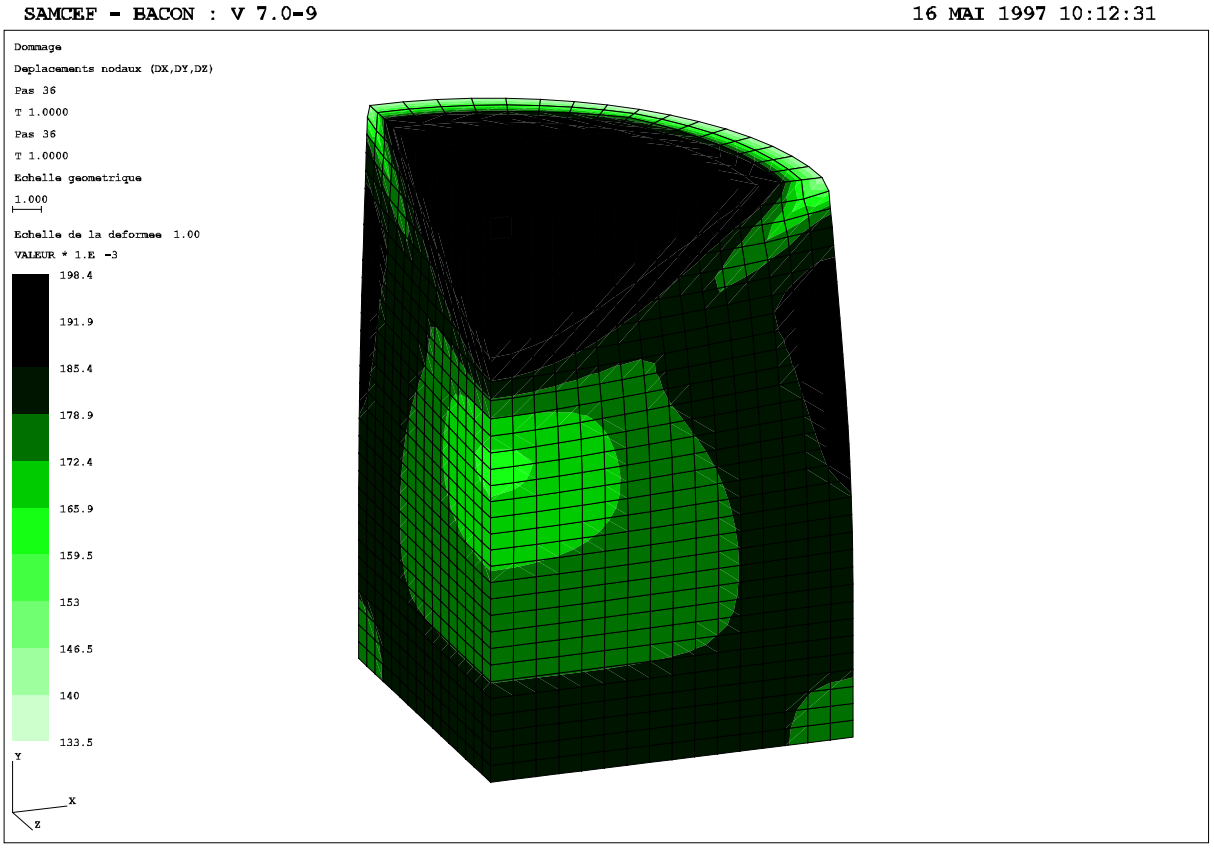
	It/pas	temps CPU	Mémoire
GMRES(60) ILUT(30)	36/33	2 h 32	10 995 677



Le graphique ci-dessus montre que le GMRES n'a effectué aucun restart (m = 55). On pourrait donc réduire la quantité de mémoire utilisée soit en diminuant m soit en diminuant *lfil*.
Dans ce dernier cas, on obtiendrait un plus grand nombre d'itérations du solveur à chaque résolution de système mais celles-ci s'effectueraient plus vite. Au total, il est impossible de prédire si ces modifications sont favorables sans relancer le calcul.

Les graphiques suivants permettent de comparer visuellement les deux solutions.
La solution 3D est bien axisymétrique au vu des solutions sur les bords de la pièce. Quant aux valeurs de la porosité (inverse de la densité) à la fin du calcul, elles coïncident très bien (voir échelle des valeurs) et les iso-valeurs ont la même forme dans les deux cas.


SAMCEF - BACON : V 7.0-9
16 MAI 1997 10:07:52
VALEUR * 1.E -3
198.4
191.9
185.4
178.9
172.5
166
159.5
153
146.5
140
133.5
Y
Z
X


SAMCEF - BACON : V 7.0-9
16 MAI 1997 10:12:31
Dommage
Deplacements nodaux (DX,DY,DZ)
Pas 36
T 1.0000
Pas 36
T 1.0000
Echelle geometrique
1.000
Echelle de la deformee 1.00
VALEUR * 1.E -3
198.4
191.9
185.4
178.9
172.4
165.9
159.5
153
146.5
140
133.5
Y
X
Z

5.5 Ecrasement d'un cylindre épais

5.5.1 Introduction

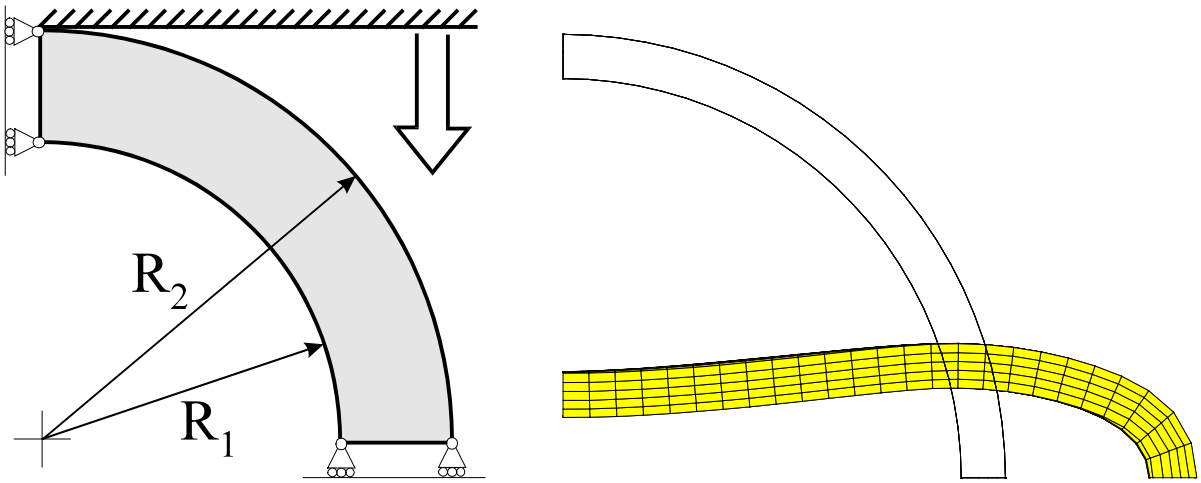
- But de ce test :

Rappelons que METAFOR ne traite pas encore le contact 3D avec frottement. Pour prouver que les solveur itératifs peuvent améliorer le temps de calcul et la quantité de mémoire utilisée pour des gros problèmes, nous nous contenterons donc d'une loi de contact sans frottement. Plus tard, il sera intéressant de voir si les résultats suivants peuvent être étendus à toutes les lois de contact.

Dans ce cas-ci, nous allons voir comment se comportent les différents solveurs par rapport au solveur direct lorsque nous augmentons le nombre de degrés de liberté. Nous considérerons rapidement le cas des solveurs symétriques (CG et SYMMLQ) pour prouver leur inefficacité face aux méthodes telles que Bi-CG ou GMRES.

- Description du problème - géométrie :

Considérons un cylindre épais en acier déposé entre les mâchoires d'une presse. Celles-ci sont progressivement resserrées pour obtenir un certain pourcentage de l'écartement initial (70%). On se propose de calculer grâce à METAFOR la déformée du cylindre au cours de l'écrasement



La loi de comportement de l'acier est identique à celle utilisée dans le cas de la flexion de poutres (voir section 5.2.1.1 pour les valeurs exactes).
On emploie une loi de contact glissant avec un faible coefficient de pénalité (10^3). On élimine ainsi les problèmes qui peuvent en découler. Plus tard, nous étudierons l'effet de ce coefficient indépendamment des autres paramètres du problème et du solveur.

Pour pouvoir étudier l'influence du nombre de degrés de liberté sans faire entrer en jeu d'autres facteurs tels que l'aspect des mailles, nous garderons la taille de celles-ci constante et nous ferons varier la hauteur du cylindre.

L'influence de la largeur de bande sera mise en évidence à l'aide de deux valeurs pour le rayon interne : 8 et 9 cm. Puisque l'on veut garder des éléments de taille constante, il faudra donc doubler leur nombre lorsque l'on double l'épaisseur du cylindre.

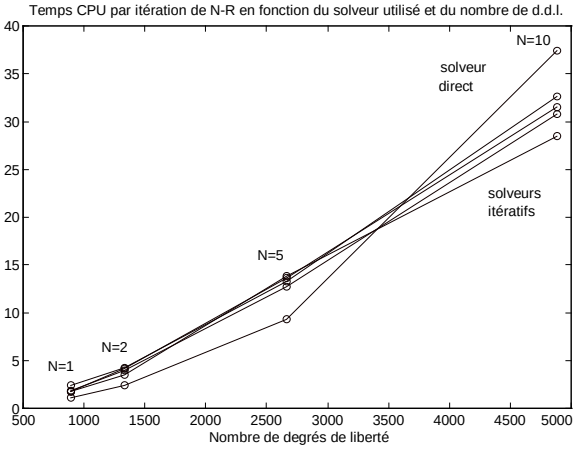
5.5.2 Comparaison des différentes méthodes de résolution

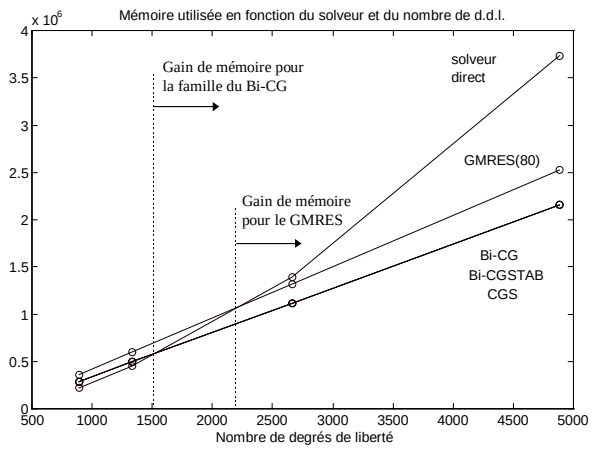
- Cylindre d'1 cm d'épaisseur :

Commençons tout d'abord par comparer les quatre solveurs non symétriques GMRES, Bi-CG, Bi-CGSTAB et CGS. Jusqu'à présent, nous ne considérions qu'un seul solveur à la fois. Il est donc intéressant de comparer leurs résultats pour des gros problèmes. Nous verrons si les conclusions des tests à deux dimensions sont toujours valables à trois dimensions.

Soit N le nombre d'éléments sur la hauteur du cylindre.

Nous utilisons le préconditionneur ILUT(30) et le paramètre de restart du GMRES(m) est fixé à 80. Cette valeur a été choisie car elle entraîne un temps de calcul optimal pour le cas $N = 1$. On obtient alors les graphiques suivants pour le cylindre d'épaisseur unitaire :





Temps CPU :
On constate que les différentes méthodes ont un comportement très semblable. Le graphique ci-dessus ne permet pas de départager une méthode itérative parmi les quatre étudiées. Nous reprenons dans le tableau ci-dessous les temps de calculs respectifs :

	$N = 1$	$N = 2$	$N = 5$	$N = 10$
GMRES(80)	3:46	7:20	28:51	57:31
Bi-CG	3:55	8:29	28:27	1h 03:35
Bi-CGSTAB	3:56	8:15	26:27	1h02:03
CGS	5:14	8:36	27:41	1h05:43
solveur direct	2:20	6:37	19:24	1h15:27

Les écarts entre les méthodes sont toujours de l'ordre de la minute.
Le GMRES conduit aux meilleurs résultats sauf dans le cas $N = 5$. Cependant, cet avantage est contrebalancé par la quantité de mémoire supplémentaire nécessaire à son bon fonctionnement.
En général, le CGS est la méthode qui fournit les résultats les plus décevants. Notons par exemple sa mauvaise performance dans le cas $N = 1$.
La méthode Bi-CG et sa version stabilisée se valent plus ou moins. On penchera vers ce type de méthode lorsque la mémoire fait défaut.

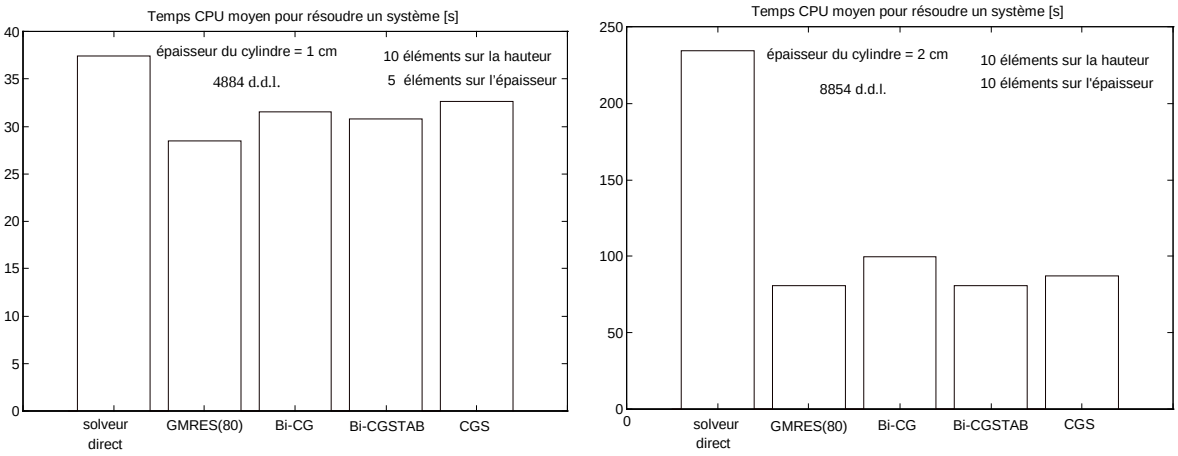
Point d'intersection et largeur de bande :
Comme précédemment, nous remarquons qu'il existe toujours un point d'intersection entre la courbe d'un solveur itératif et celle du solveur direct. Cependant, celle-ci a lieu pour des nombres de degrés de liberté assez élevé (environ 3500) si on les compare avec celles du cube traité précédemment. La différence provient de la différence de largeur de bande de la matrice de raideur tangente des deux problèmes. En effet, le cube de 6 éléments de côté possédait 883 d.d.l. et une hauteur de colonne moyenne de 123. Dans le cas du cylindre, le plus petit cas considéré possède 888 d.d.l., c'est-à-dire environ la même chose ; cependant la largeur de bande moyenne obtenue est de 24 !

Mémoire :
En ce qui concerne la mémoire (graphique de droite), les résultats sont classiques. On retrouve aussi le point d'intersection entre la courbe du solveur direct et celle d'un solveur itératif. Ce point correspond à un nombre de d.d.l. moins élevé que celui relatif au temps de calcul. Autrement dit, en augmentant la taille du problème, on gagne de la mémoire par rapport au solveur direct avant de gagner du temps de calcul.
Il est intéressant de visualiser la place occupée par les 80 vecteurs de Lanczos lors de l'utilisation du GMRES. Cette quantité de mémoire est représentée par l'espace entre les deux droites du graphique. L'écart à tendance à s'amplifier avec l'augmentation du nombre de d.d.l. (il vaut *m**(*nombre de d.d.l.*)).

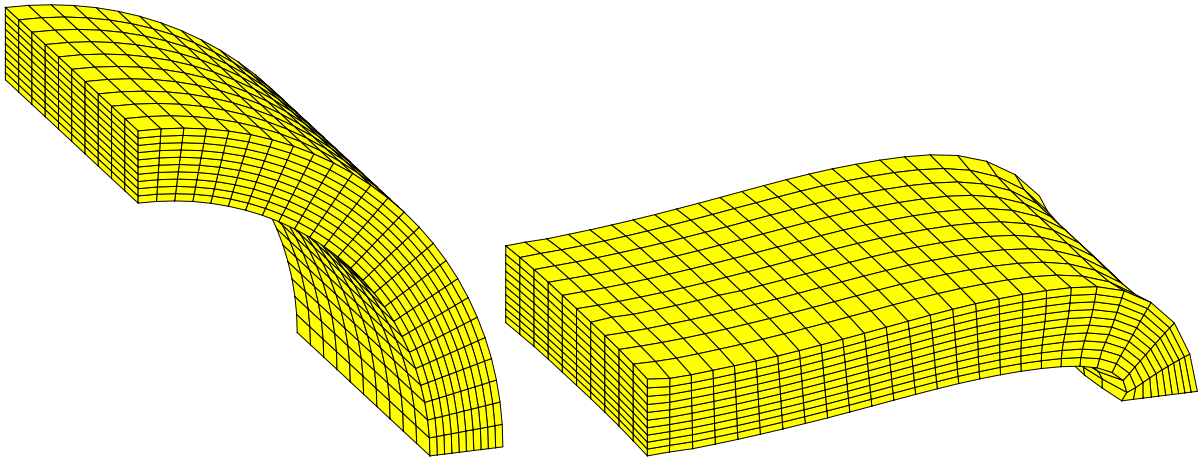
- Cylindre de 2 cm d'épaisseur :

Les deux graphiques ci-dessous permettent de voir clairement l'effet d'une augmentation du nombre de degrés de liberté. A la place de continuer à allonger le cylindre, nous augmentons son épaisseur pour augmenter plus significativement la largeur de bande.

Dans le cas du cylindre le plus épais, les solveurs itératifs permettent de diviser le temps de calcul par un nombre légèrement inférieur à 3 (dans le cas du GMRES le calcul dure 3h02 au lieu de 8h57).

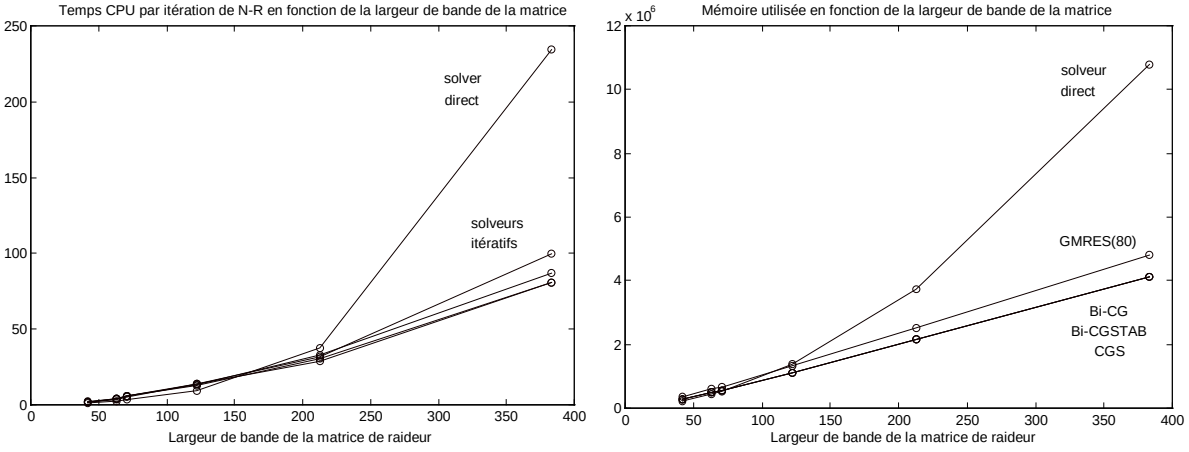


Voici une représentation de la structure initiale et déformée après écrasement :



- <u>Influence de la largeur de bande :</u>

Les deux graphiques suivants reprennent tous les tests effectués pour comparer le temps de calcul et la mémoire utilisée par les différents solveurs en fonction de la hauteur de colonne moyenne (image de la largeur de bande moyenne) de la matrice de raideur tangente.



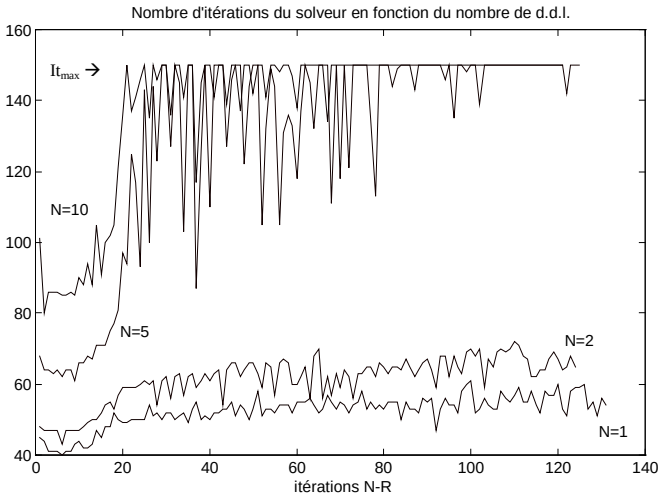
En utilisant comme référence la largeur de bande de la matrice au lieu du nombre de degrés de liberté, on obtient des courbes qui dépendent beaucoup moins du type de problème. On voit par exemple que les solveurs itératifs deviennent plus rapide que le solveur direct lorsqu'on atteint une valeur de 150. Ceci peut être vérifié dans le cas du cube.

5.5.3 Influence du nombre de degrés de liberté sur le nombre d'itérations moyen

Le nombre d'itérations moyen du solveur par résolution de systèmes n'est pas constant lorsqu'on augmente le nombre de degrés de liberté. En général, plus le système est grand, plus le solveur itératif devra effectuer un nombre important d'itérations pour obtenir la précision demandée.
Examinons ce phénomène de plus près dans le cas du GMRES :

	$N = 1$	$N = 2$	$N = 5$	$N = 10$
nombre de d.d.l.	888	1332	2664	4884
itmoy	52	61	129	135

Ici, nous avons fixé une limite de 150 itérations et cela donne de bons résultats dans tous les cas. La précision demandée par l'intermédiaire de ε pourrait être diminuée. En effet, dans le cas N=10, cette précision n'est pas atteinte à cause de la limite sur les itérations mais la convergence de Newton-Raphson n'est pas affectée (on effectue toujours le même nombre de pas de temps que le solveur direct).



Si on veut traiter des problèmes plus gros, il est préférable d'augmenter légèrement cette limite artificielle pour ne pas perturber la convergence du processus de Newton-Raphson. C'est ce que nous avons fait pour le plus gros cas (8954 d.d.l.) : on a choisi dans ce cas it_{max} = 220.

5.5.4 Utilisation des solveurs symétriques

- Introduction :

Le principal avantage des solveurs symétriques est, bien sûr, le gain de mémoire qu'ils permettent dans le stockage de la matrice de raideur tangente (on stocke uniquement la triangulaire supérieure).

Dans la littérature, les solveurs symétriques tels que le gradient conjugué (CG) sont fréquemment utilisés dans le cadre de l'élastostatique. Ils ont d'ailleurs permis la résolution de très gros problèmes à peu de frais. L'utilisation de tels solveurs peut-elle apporter quelque chose dans le domaine non linéaire des grandes déformations ?

Si c'est le cas, nous savons que de tels solveurs devront toujours être secondés par d'autres méthodes pour traiter les problèmes où le processus de Newton-Raphson ne converge plus lorsque l'on symétrise la matrice de raideur. C'est le cas des problèmes faisant intervenir du contact avec frottement ou des forces non conservatives.
De plus, il faut maintenant comparer les résultats obtenus avec ceux du solveur direct symétrique. Celui-ci est approximativement deux fois plus rapide que sa version non symétrique.

Ces premières réflexions montrent qu'il sera difficile de concurrencer le solveur direct avec une méthode itérative du type CG ou SYMMLQ.

- Résultats :

Essayons de résoudre le plus petit problème (1 élément de profondeur, 888 d.d.l.) avec ces méthodes qui ont tant de succès pour d'autres types de problèmes :

Méthode		Pas/it.	Temps CPU [s]
SYMMLQ	+ RIC(0)	65/170	9:48
CG	+ RIC(0)	68/177	9:51
Bi-CG	+ ILU(0)	61/171	8:51
Bi-CG	+ ILUT(30)	51/130	3:55
Bi-CG	+ ILUT(20)	/	/
Solveur direct	(symétrique)	51/130	2:00

Nous avons considéré, à titre de comparaison, le Bi-CG avec deux types de préconditionnement : ILU(0) qui est la version non symétrique de la factorisation incomplète de Choleski et ILUT(30) qui a déjà fait ses preuves tout au long de ce travail.

Seul le préconditionnement par ILUT(30) permet d'obtenir le même nombre d'itérations total que le solveur direct. Dans les autres cas, METAFOR effectue de nombreuses divisions de pas de temps et le temps de calcul total en est affecté fortement.

Pour le Bi-CG, le choix d'un paramètre *lfil* égal à 20 entraîne la divergence du processus de Newton-Raphson.

- Causes des mauvaises performances :

Mauvaise qualité du préconditionnement :
Le temps de résolution obtenu par les méthodes CG et SYMMLQ sont beaucoup trop importants. On atteint presque 5 fois le temps du solveur direct. Ce manque de puissance provient en grande partie du préconditionneur utilisé. Celui-ci est basé sur une factorisation de Choleski incomplète. Comme cet algorithme fonctionne uniquement dans le cas de matrices symétriques et définies positives, il est nécessaire d'effectuer une réduction de la matrice (expliquée clairement dans la partie théorique du travail). Cette réduction entraîne une perte d'information inévitable et on obtient en conséquence une mauvaise approximation de l'inverse de la matrice.

Difficultés provenant de la symétrisation :
En pratique, on constate que l'utilisation d'un solveur itératif et une matrice de raideur symétrisée pose des problèmes[15]. Le fait de symétriser la matrice constitue déjà une première approximation qui peut faire diverger le processus de Newton-Raphson dans certains cas. Le solveur itératif entraîne une deuxième erreur puisque le système n'est généralement pas résolu aussi précisément qu'avec le solveur direct. L'accumulation des ces deux approximations successives rend le problème souvent impossible à résoudre par cette voie.

- Conclusion :

Ces résultats déplorables justifient l'abandon définitif de ce type de solveur pour METAFOR.

[15] Voir à ce sujet les essais réalisés dans le cas de la flexion des poutres élastoplastiques (section 5.2). D'autres tentatives ont été réalisées sur le cube et le cylindre rempli de poudre. Lorsque les deux méthodes symétriques sont applicables, les résultats sont toujours décevants.

5.5.5 Influence du coefficient de pénalité normale

Etudions maintenant l'influence du coefficient de pénalité normale (K_N). On utilisera le cas du cylindre mince de 1 cm d'épaisseur avec le GMRES(80) + ILUT(30).

Nous regroupons les résultats le tableau suivant. La notation *x/y* signifie que le solveur itératif entraîne x itérations tandis que le solveur direct en entraîne *y*. Lorsque $x = y$, un seul nombre est répertorié.

K_N [N/mm^2]	10^3	10^4	10^5	10^6	10^7	10^8
itérations N.-R.	131	170	191	208	187/208	222/208
CPU (GMRES)	3:46	4:49	5:13	5:34	5:02	5:56
CPU/it.	1.72	1.70	1.63	1.60	1.61	1.60

Pour le cas qui nous intéresse, le temps moyen pour résoudre un système n'est pas une fonction croissante de la pénalité.

Le lecteur comprendra pourquoi nous avons fait les tests précédents avec une pénalité si faible (10^{-3}) : la résolution itérative n'est pas perturbée mais le temps de calcul est beaucoup plus long vu le nombre croissant d'itérations de Newton-Raphson ; de plus, les grandeurs caractéristiques (contraintes, déformations,...) ne nous intéressent pas dans le cadre de ce travail (tant qu'il n'y a pas de différences avec le solveur direct).

5.5.6 Conclusions

Ce problème nous a permis de montrer les similitudes du comportement des solveurs GMRES, Bi-CG, Bi-CGSTAB et CGS. Lorsque l'on augmente la taille du problème, ces derniers permettent d'obtenir une grande rapidité de calcul et une économie de mémoire non négligeable.

Les solveurs symétriques utilisés couramment en élasticité linéaire sont inadaptés au cas de grandes déformations.

Une grande valeur du coefficient de pénalité ne joue pas un rôle néfaste vis-à-vis d'un solveur itératif. C'est le processus de Newton-Raphson et non pas le solveur qui converge moins vite. Cette dernière conclusion devra être vérifiée par la suite dans un autre cas.

5.6 Formage superplastique d'un composant aéronautique

5.6.1 Introduction

- But de ce test :

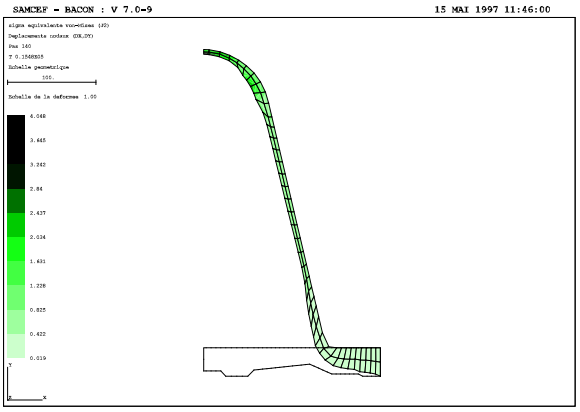
Cet exemple va nous permettre de considérer les lois de contact avec frottement. Malheureusement, il n'est pas encore possible de traiter ce type de problème à trois dimensions. Nous étudierons donc l'effet des paramètres tels que les coefficients de pénalités normale et tangentielle sans rechercher à obtenir des performances intéressantes.

- Description du processus et de la géométrie :

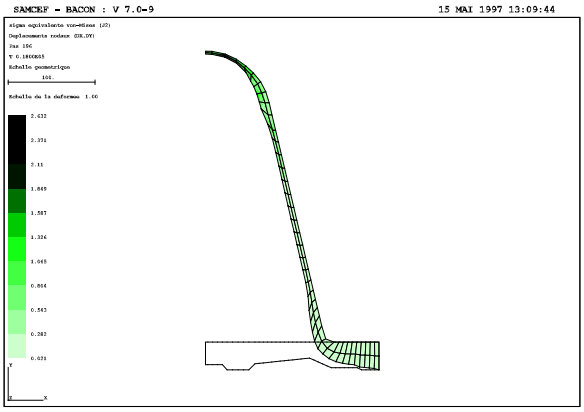
Il s'agit de simuler la fabrication d'un composant aéronautique par superplasticité. Ce procédé est extrêmement long (plusieurs heures) et ne permet pas une production en série. Cependant, grâce à cette façon de procéder, on peut atteindre des déformations très importantes (500 %) sans rupture du matériau. On l'utilise donc principalement dans l'industrie aéronautique.

Ce test a été réalisé la première fois par J.-P. Ponthot pour la mise au point d'une loi super-plastique dans METAFOR. Il décrit en détail dans son travail [P4] le procédé de formage d'une telle pièce et les propriétés du matériau utilisé.

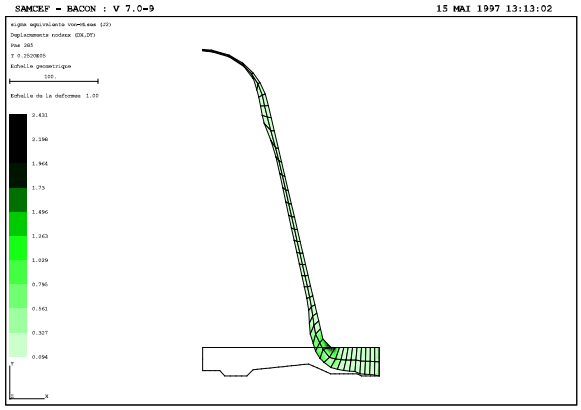
$\mu = 0$



$\mu = 0.15$



$\mu = 0.30$



Le matériau superplastique est déposé au-dessus d'un moule qui déterminera sa forme future. Une pression est appliquée uniformément sur le matériau jusqu'à ce qu'il soit en contact avec l'entièreté du moule. Pour la modélisation dans METAFOR, on utilise une représentation 2D axisymétrique puisque le moule est une surface de révolution.

- Loi de comportement utilisée :

Rappelons le critère de Von-Mises généralisé à un matériau superplastique utilisé par Ponthot. Les coefficients ont été fixés par l'expérience.

$$\overline{\sigma} = 0.0296 + 5267.62 \left(\dot{\overline{\varepsilon}}^{vp}\right)^{0.85}$$

tandis que les paramètres élastiques sont donnés par :

$$E = 12\,000\ N/mm^2 \quad ; \quad \nu = 0.25$$

Pour modéliser le contact, nous utiliserons donc successivement trois types de frottement caractérisés par un coefficient μ différent :

→ contact glissant : $\mu = 0$
→ contact avec frottement : $\mu = 0.15$ et $\mu = 0.30$

5.6.2 influence de la précision du solveur sur la solution finale

- Introduction :

La précision du solveur ε peut influencer la solution obtenue en fin de calcul. Nous allons le montrer sur cet exemple en utilisant $\mu = 0.15$ (des résultats analogues pourraient être obtenus avec d'autres coefficients de frottement).

Rappelons que Newton-Raphson considère que la pièce mécanique est en équilibre lorsque la norme du résidu d'équilibre est inférieure à une tolérance donnée. Par défaut, celle-ci est fixée à 10^{-3} .Ceci découle de nombreux tests qui tendent à prouver qu'une valeur supérieure conduit à une précision inutile et des temps de calculs très longs ; par contre, une valeur supérieure fournit une solution qui est beaucoup trop loin de l'équilibre pour représenter correctement la physique du phénomène . Dans la suite, nous appelons ce nombre *TOL*.

Nous allons monter que $TOL = 10^{-3}$ n'est pas toujours le meilleur choix lorsqu'on utilise un solveur itératif, même dans les cas où le solveur direct donne de bons résultats avec une telle tolérance.

Pour les comparaisons qui suivent, nous utilisons la contrainte équivalente de Von-Mises.

- Solution fournie par le solveur direct et $TOL = 10^{-3}$:

[N/mm^2]	$(\sigma_{eq})_{max}$	$(\sigma_{eq})_{min}$	Pas/it.
solveur direct	2.6317296	2.0614358 10^{-2}	196/516

- Solutions fournies avec le solveur itératif et avec $TOL = 10^{-3}$ et $\varepsilon = 10^{-4}$:

Utilisons maintenant le GMRES(60) avec $\varepsilon = 10^{-4}$. Comme préconditionneur, nous utilisons ILUT(*lfil*) avec des valeurs décroissantes du paramètre de remplissage *lfil*.

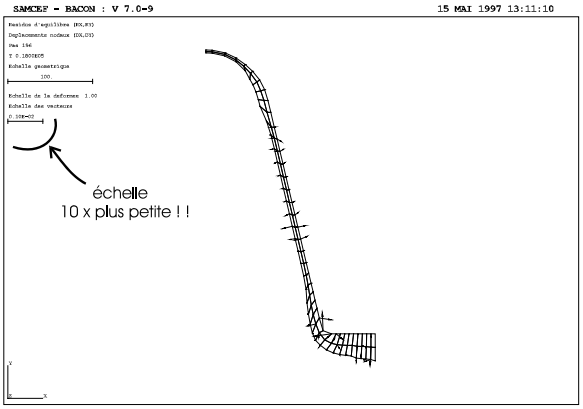
[N/mm^2]	$(\sigma_{eq})_{max}$	$(\sigma_{eq})_{min}$	Pas/it.
ILUT(20)	2.6317**370**	2.061**3890** 10^{-2}	196/516
ILUT(10)	2.62**98585**	2.07**34455** 10^{-2}	196/511
ILUT(9)	2.**8680513**	**1.5518852** 10^{-2}	192/501
ILUT(8)	**4.6953468**	**0.9040115** 10^{-2}	166/424
ILUT(7)	pas de convergence		
ILU(0)	pas de convergence		

Lorsqu'on utilise un préconditionneur assez faible, la solution finale est très différente des valeurs obtenues avec le solveur direct (erreur relative de 78% dans le pire des cas).

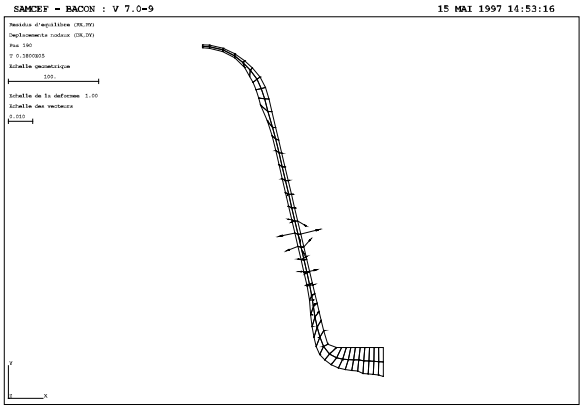
Si le paramètre de remplissage est égal à 20, on préconditionne le système avec l'inverse exacte de la matrice de raideur (sa largeur de bande moyenne vaut 8) et on converge en une seule itération. On obtient alors le même nombre de pas de temps que le solveur direct.

Remarquons qu'il n'y a plus de convergence pour des valeurs de *lfil* inférieure à 8.

Les deux graphiques suivants montrent la répartition des résidus d'équilibre sur la structure en fin de calcul pour le solveur direct d'une part et GMRES+ILUT(8) d'autre part.

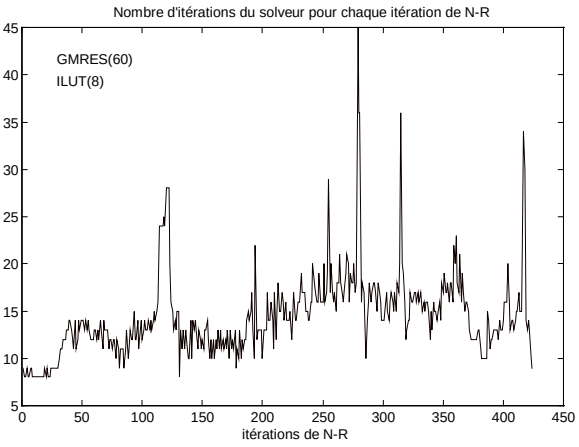


Solveur direct



GMRES(80) + ILUT(8)

Ces dessins prouvent que la solution fournie par le solveur direct est plus proche de l'équilibre que celle relative au GMRES (l'échelle des vecteurs est différente)



Pour effectuer ces tests, nous avons utilisé un nombre d'itérations maximal it_{max} = 200.

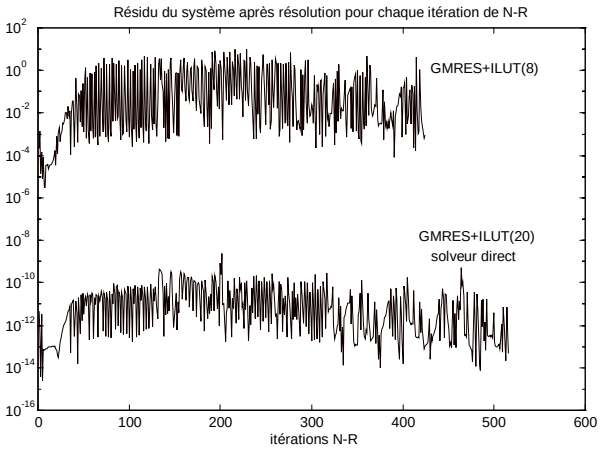
Le graphique ci-contre montre, dans le cas du préconditionneur le plus faible, qu'il n'est jamais atteint. Autrement dit, le résidu du système satisfait bien le critère d'arrêt après chaque itération de N-R.

On étudie donc bien le rôle de ε sans l'influence d'autres paramètres.

Lorsqu'on arrive à la dernière itération d'un pas de temps, le solveur direct permet généralement de diminuer plus fortement la norme résidu d'équilibre que le solveur itératif (ce dernier est plus proche de 10^{-3} que le solveur direct). Par exemple, pour la dernière itération, on obtient un résidu d'équilibre de 4.9 10^{-6} pour le solveur direct et 'seulement' 4.3 10^{-5} pour le GMRES avec ILUT(8). Ces deux valeurs sont inférieures à 10^{-3}.
Vu le nombre important de pas de temps effectués au cours du calcul, ces différences s'amplifient et les solutions finales sont différentes.

Le graphique suivant montre la norme du résidu $|b - A\,x|$ après chaque passage dans la routine du solveur.

La courbe inférieure correspond à ILUT(20). Elle est identique à celle du solveur direct puisque l'algorithme ne fait qu'une itération. On voit très bien que le solveur direct résout beaucoup mieux le système que ne le fait la GMRES lorsqu'il est couplé à un faible préconditionneur.



Les deux paragraphes suivants vont confirmer notre tentative d'explication. Nous allons montrer qu'on retrouve bien la solution donnée par le solveur direct en utilisant soit une précision plus grande pour le solveur soit une précision plus grande pour le processus de Newton-Raphson

- Calcul avec $TOL = 10^{-3}$ et $\varepsilon = 10^{-6}$:

Considérons toujours ILUT(8) avec une précision 100 fois plus importante. La différence avec les résultats du solveur direct devient très petite comme le montre le tableau suivant :

[N/mm^2]	$(\sigma_{eq})_{max}$	$(\sigma_{eq})_{min}$	Pas/it.
GMRES	2.631**2077**	2.06**48342** 10^{-2}	196/516

- Calcul avec $TOL = 10^{-5}$ et $\varepsilon = 10^{-4}$:

Diminuons la tolérance sur la norme du résidu d'équilibre. Les résultats du solveur direct sont exactement identiques au cas $TOL = 10^{-3}$. Cela confirme que la pièce mécanique était bien plus près de l'équilibre qu'en utilisant le solveur itératif. Les résultats de ce dernier sont résumés dans le tableau suivant :

[N/mm^2]	$(\sigma_{eq})_{max}$	$(\sigma_{eq})_{min}$	Pas/it.
GMRES	2.7**548515**	**1.879717** 10^{-2}	190/488

On trouve bien, comme on s'y attendait, des valeurs plus proches de celles du solveur direct.

5.6.3 Influence des coefficients de pénalité normale et tangentielle

- Introduction :

Les coefficients de pénalité peuvent introduire des contributions importantes dans la matrice de raideur tangente. En effet, si un noeud du maillage pénètre légèrement dans la matrice de contact, METAFOR crée une force qui est proportionnelle à la profondeur de pénétration et qui modélise la force de contact réelle.

Les coefficients de pénalité représentent le rapport entre l'intensité de la force de contact et les gaps normaux et tangentiels. Plus ces derniers sont élevés, plus la structure étudiée est proche de la réalité physique mais, dans ce cas, la matrice de raideur devient mal conditionnée. Remarquons qu'une pénétration nulle à l'équilibre demanderait des coefficients infinis.

Appelons K_T et K_N les deux coefficients de pénalité. Les tests précédents ont été effectués avec les valeurs $K_N = 2\ 10^5\ N/mm^2$ et $K_T = 2\ 10^4\ N/mm^2$.

Comme solveur, nous utilisons GMRES(60) et ILUT(8).

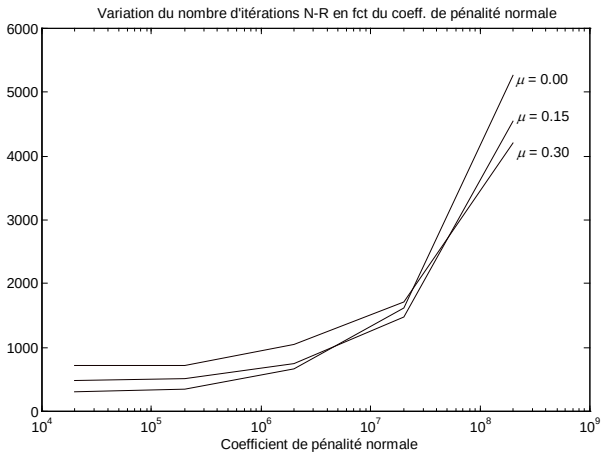
- Influence de K_N avec $K_T = 10^4$:

Nombre d'itérations N-R total :
Le tableau ci-dessous montre le nombre d'itérations pour effectuer le calcul avec le solveur direct et le GMRES. Lorsque les valeurs sont différentes, *x* / *y* signifie *x* itérations pour le solveurs direct et *y* pour le GMRES.

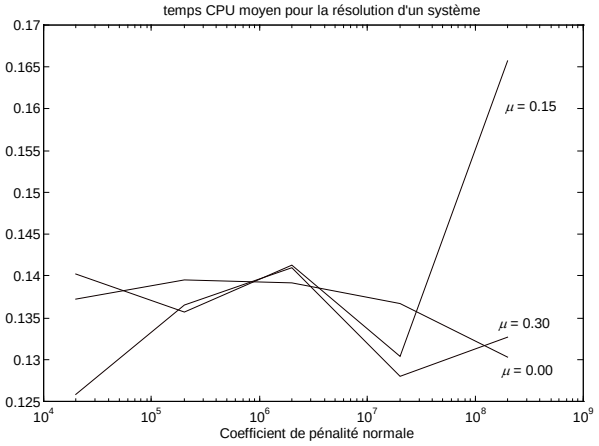
	$K_N = 2\ 10^4$	$K_N = 2\ 10^5$	$K_N = 2\ 10^6$	$K_N = 2\ 10^7$	$K_N = 2\ 10^8$
$\mu = 0.00$	306	344	661	1617	5265/5261
$\mu = 0.15$	485	516	743	1473/1474	4550/4569
$\mu = 0.30$	715	718	1057/1043	1719	4207/4128

Le nombre d'itérations est fortement influencé par le coefficient de frottement et la pénalité utilisée. Chaque fois que les deux nombres d'itérations ne coïncident pas, nous avons vérifié que la solution finale est identique dans les deux cas.

Le graphique ci-contre est relatif au solveur direct. Les trois valeurs de μ sont envisagées.



Temps CPU par itération de N-R :
Le solveur n'est pas affecté directement par le coefficient de pénalité : le graphique ci-contre montre le temps CPU moyen pour résoudre un système. Les courbes relatives à des valeurs de μ différentes ne sont pas du tout similaires.



Pour ce cas particulier, on ne peut donc pas conclure que K_N provoque des difficultés pour la résolution par un solveur itératif.

Temps CPU total :
Dans le cas de $K_N = 2\ 10^8$, il faut en moyenne 9 minutes pour effectuer le calcul avec le solveur direct et 11 minutes avec son homologue itératif. Par contre, pour $K_N = 2\ 10^4$, les deux solveurs convergent en moins de 4 minutes.

Conclusion :
La difficulté et la lenteur du calcul proviennent donc principalement du processus de Newton-Raphson et non pas de la résolution du système à chaque itération.

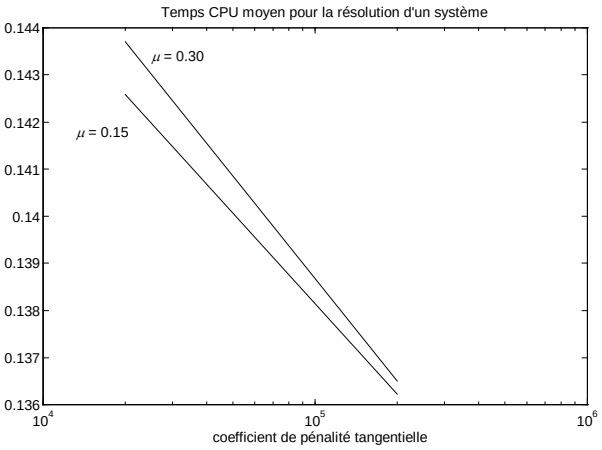
- Influence de K_T avec $K_N = 10^5$:

Considérons maintenant la pénalité tangentielle :

Nombre d'itérations N-R :
Le solveur itératif converge toujours avec le même nombre d'itérations que le solveur direct, comme le montre le tableau suivant.

	$K_T = 2\ 10^4$	$K_T = 2\ 10^5$	$K_T = 2\ 10^6$
$\mu = 0.15$	516	533	/
$\mu = 0.30$	718	947	/

Temps CPU par itération N-R :
Ici aussi, on constate que la pénalité n'influence pas défavorablement le solveur itératif.

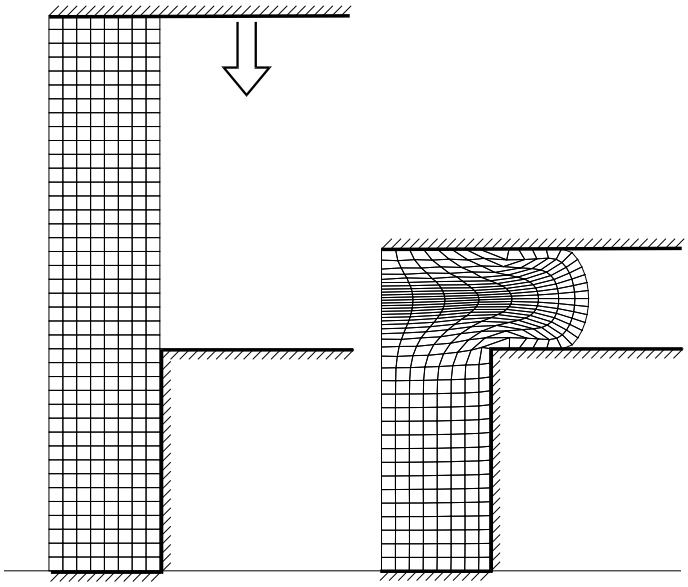


Il reste à savoir (ce n'est pas fait dans ce travail) si ces conclusions peuvent être généralisées dans d'autres cas

5.7 Formage d'un boulon

5.7.1 Introduction

Pour être complet, nous traitons dans cette section un cas 2D soumis à une loi de contact collant. Il s'agit d'une pièce cylindrique en acier dont on comprime la base supérieure. La partie inférieure de la pièce est coincée dans un moule rigide et ne peut donc pas se déformer axialement. A la fin du processus, on obtient une pièce qui ressemble fort à un boulon, d'où le titre de cet exemple.



La pièce initiale mesure 50 mm de haut et 10 mm de rayon. On la comprime de 20 mm.

5.7.2 Résultats

Il est assez difficile de faire varier les paramètres utilisés (pénalité, nombre de d.d.l.,...) parce que le problème devient impossible à résoudre, même avec le solveur direct. Nous nous contenterons donc de montrer que le solveur itératif parvient à résoudre le problème mais il lui faut beaucoup de temps.

Les pénalités sont fixées toutes les deux à 10^5 et la pièce est maillée à l'aide de 40 éléments sur la hauteur et 8 sur le rayon (nous ne sommes pas arrivés à faire converger METAFOR avec un maillage plus raffiné).

	Temps CPU	Mémoire	Pas/it.
solveur direct	0:29	165523	45/103
GMRES(20) + ILUTP(20)	1:46	213898	45/103

Il est intéressant de voir la différence entre les temps de calcul pour obtenir la même solution ! Pourtant, nous avons 688 d.d.l. Rappelons qu'un cube possédant plus ou moins le même nombre de degrés de liberté entraîne déjà un gain de mémoire par rapport au solveur direct et des temps de calcul des deux solveurs très voisins.

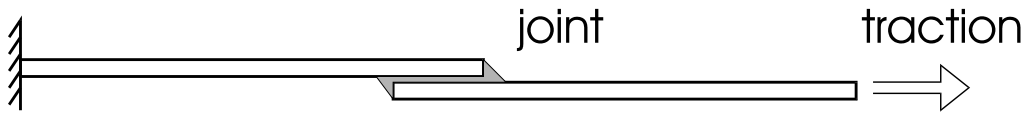
On remarque donc une nouvelle fois l'importance de la largeur de bande de la matrice de raideur. Celle-ci joue un rôle primordial pour le choix du solveur optimal.

5.8 JSR3D

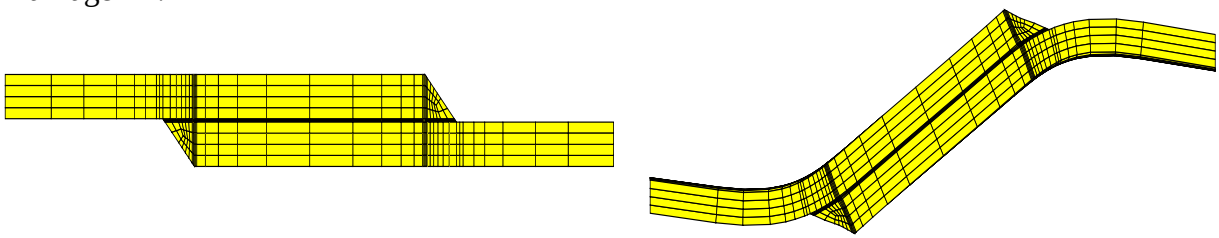
5.8.1 Introduction

Ce problème 3D a été étudié très précisément lors de l'élaboration de la loi de comportement des colles [T1]. L'intérêt d'un tel exemple dans le cadre de ce travail est simple : nous allons montrer que la présence de mailles aplaties n'exclut pas toujours l'emploi d'un solveur itératif. C'est d'autant plus vrai si le problème possède beaucoup de degrés de liberté et que sa résolution devient coûteuse par la voie traditionnelle.

Il s'agit de deux plaques métalliques collées par un joint (JSR = joint sans recouvrement). Une des deux est encastrée à son extrémité et on applique sur l'autre un effort de traction. Ci-dessous, nous avons représenté très schématiquement la situation :



Pour obtenir les résultats voulus (principalement la répartition des contraintes dans le joint), il est absolument nécessaire de mailler finement la région entre les deux plaques. Par contre, les parties extrêmes des deux plaques, c'est-à-dire l'encastrement et celle sur laquelle on exerce une traction, ne subissent pas de grandes variations de contraintes et il est inutile d'utiliser un maillage fin.



Ces deux dessins représentent un zoom sur le joint dans la configuration initiale et la configuration déformée (amplifiée 30 fois). On peut remarquer la finesse du maillage au niveau du joint.

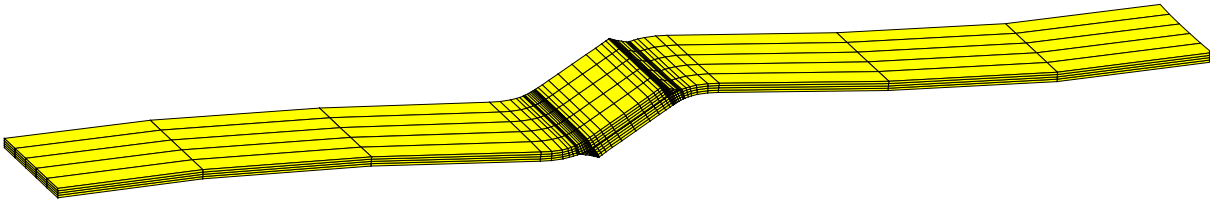
5.8.2 Résultats

Nous utilisons plusieurs méthodes différentes pour pouvoir déterminer la meilleure :

	Temps CPU	Mémoire	Pas/it.
Bi-CG + ILUT(80)	13 :01	4 615 269	57/9
Bi-CG + ILUT(60)	12 :50	3 941 019	57/9
Bi-CG + ILUT(40)	13 :08	3 266 770	57/9
GMRES + ILUT(60)	11 :58	4 856 497	57/9
solveur direct	14 :04	5 545 680	57/9

Temps de calcul :
Les temps de calcul sont fort semblables. Les solveurs itératifs sont légèrement plus rapides avec la GMRES en tête. Encore une fois, nous remarquons l'influence de la largeur de bande. Nous avons 7748 d.d.l. et pourtant les solveurs itératifs ne brillent pas par leur vitesse d'exécution.

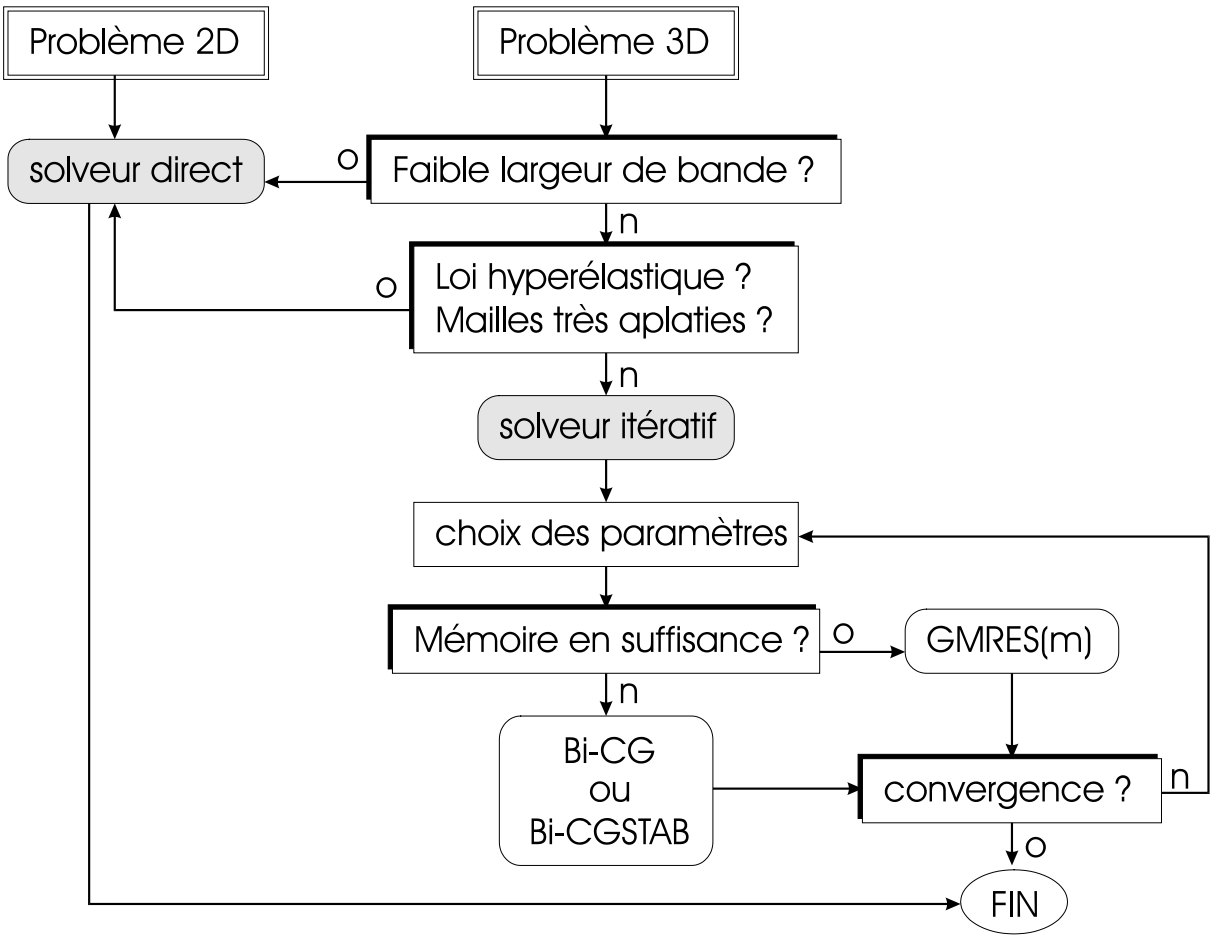
Gain de mémoire :
On voit très bien l'influence du choix du paramètre de remplissage non seulement sur le temps CPU mais aussi sur la mémoire utilisée. Le meilleur choix, dans ce cas-ci est d'utiliser la méthode Bi-CG + ILUT(40) vu l'économie de mémoire qu'elle procure.



5.9 Conclusions

5.9.1 Choix du solveur

Dans cette section, nous résumons la manière de choisir le solveur et ses paramètres en fonction du type de problème. L'organigramme ci-dessous montre les questions qu'il faut se poser avant d'effectuer son choix. On remarque qu'il reste encore beaucoup de cas qui doivent être traités avec le solveur direct. Il n'est donc pas avantageux d'oublier complètement celui-ci bien que la grande majorité des problèmes peuvent être résolus uniquement avec le GMRES et le Bi-CG.



5.9.2 Choix de paramètres du solveur itératif

Tous les tests de ce chapitre prouvent que le choix des paramètres n'est pas facile. Il nécessite de l'expérience et un certain 'feeling'. C'est d'ailleurs pour cette raison que nous ne sommes pas arrivés à programmer un algorithme qui choisirait les paramètres du solveur sans aucune intervention de l'utilisateur. La conception d'une telle boîte noire demanderait un travail

colossal et beaucoup plus de tests que ceux effectués ici. Nous regroupons ici quelques conseils pour guider un utilisateur débutant.

- Paramètre de restart (*m*) :

Si d'après l'organigramme ci-dessus le GMRES se révèle être le meilleur choix, il faut lui associer un paramètre de restart convenable. Une trop petite valeur provoquera une convergence beaucoup trop lente. Une valeur trop grande entraînera un gaspillage de la mémoire utilisée (et des calculs inutiles pour orthogonaliser chaque nouveau vecteur de Lanczos). Entre les deux, on trouve une valeur optimale qui minimise le temps de calcul.

Il vaut mieux utiliser un trop grand paramètre de restart qu'un trop petit. Habituellement des valeurs de *m* de l'ordre de 80 assure la convergence.

Pour des longs calculs, on peut utiliser l'astuce suivante : on lance METAFOR avec un paramètre élevé (80 par exemple) et on regarde les résultats de la première itération. Si on tient compte du fait que, généralement, les dernières itérations sont les plus pénibles pour le solveur itératif (maillage écrasé, zones plastifiées étendues,...), on peut alors ajuster le paramètre de restart en conséquence. Par exemple, si le solveur effectue déjà des restarts à la première itération, c'est le signe qu'on a choisi une valeur trop faible.

- Choix du paramètre de remplissage :

Les résultats obtenus avec ILUT montrent bien que ce préconditionneur est le choix idéal à l'heure actuelle pour résoudre les problèmes rencontrés dans METAFOR. Grâce à son paramètre de remplissage, on peut doser son efficacité et sa taille en mémoire.
D'habitude, il est intéressant d'utiliser une valeur légèrement supérieure à[16]

$$lfil = NNZ/(2*NSYS)$$

où *NNZ* est le nombre d'éléments non nuls dans la matrice de raideur (cette valeur est calculée lors du préassemblage et affichée dans le fichier résultat).
NSYS est la taille du système.

On obtient ainsi un préconditionneur légèrement plus rempli que ILU(0) et donc plus puissant. Pour des problèmes 3D, on doit donc utiliser des valeurs plus élevées qu'en 2D.

2D : $5 < lfil < 20$ 3D : $30 < lfil < 60$

Une autre astuce pour fixer correctement le paramètre *lfil* cette fois : comme pour le cas du paramètre de restart, on peut lancer le calcul et essayer d'obtenir, sur les premières itérations un temps de préconditionnement plus ou moins égal au temps de résolution. Cette méthode empirique donne souvent de bons résultats pour des problèmes de taille moyenne.

[16] Rappelons que le paramètre de remplissage représente le nombre d'éléments retenus par ligne dans matrice L et dans la matrice U (ceci explique le facteur 2 au dénominateur)

Pour des petits problèmes, nous avons vu que le temps de préconditionnement optimal correspond au calcul de l'inverse exact de la matrice.
Pour les très gros problèmes, on essayera de ne pas perdre trop de temps dans le préconditionnement (temps de résolution > temps de calcul du préconditionnement).

- Critère d'arrêt :

Le critère d'arrêt peut être modifié par l'intermédiaire de deux paramètres : ε et it_{max}. Le premier fixe le rapport entre la norme du résidu final du système d'équations et la norme du résidu initial. Le second permet de limiter le nombre d'itérations lorsque la convergence est trop lente. Si, à un instant donné, on atteint it_{max} itérations, la précision demandée par l'utilisateur n'est pas atteinte mais elle est peut-être suffisante pour continuer le processus de Newton-Raphson.
Les tests précédents montrent que l'utilisation de $\varepsilon = 10^{-6}$ est un bon choix.
Quant au nombre d'itérations maximum, il ne doit pas être choisi trop grand. Des valeurs de 200 à 500 conduisent souvent à de bons résultats.

$$\varepsilon = 10^{-6}$$
$$200 < it_{max} < 500$$

Si on veut à tout prix suivre le même chemin que le solveur direct, il faut prendre une valeur de it_{max} très grande pour éliminer son influence et une précision de l'ordre très petite (10^{-10})

- Précision de l'algorithme de Newton-Raphson (*TOL*) :

Nous avons vu que, dans la plupart des cas, une tolérance de 10^{-3} sur le résidu est suffisante mais qu'il existe des cas où une précision plus grande est nécessaire pour obtenir les résultats du solveur direct avec une valeur de *TOL* identique.

A *TOL* égale, utiliser le solveur direct conduit à des résultats plus précis

Insistons encore une fois sur le fait que ces règles ne sont pas absolues et chaque problème doit être analysé séparément.

- Les autres paramètres :

Remarquons qu'il existe d'autres paramètres qui n'ont pas été pris en compte ici :
– *droptol* est un paramètre qui permet de négliger les éléments les plus petits lors du calcul du préconditionneur ILUT. Celui-ci est décrit en détail dans le chapitre 4. Nous l'avons constamment pris à 0.0 ce qui permet un contôle total sur la mémoire et la qualité du préconditionneur résultant de la factorisation.
– *permtol* est la tolérance de permutation lorsqu'on effectue un pivotage lors du préconditionnement. Certains tests ont été calculés en utilisant *permtol* = 0.01. Autrement dit, on effectue un changement de pivot lorsqu'il existe sur la ligne courante un élément dont la valeur est 100 fois plus élevée que celle de l'élément diagonal. Cependant, on constate que l'algorithme n'effectue pratiquement jamais de permutations.

6. Conclusions

6.1 Derniers commentaires

Nous arrivons au terme de ce travail de fin d'études. Quelles conclusions pouvons nous en tirer ? Résumons tout d'abord brièvement la démarche effectuée :

Dans le premier chapitre, nous avons délimité le problème et fixé les buts de ce travail : au départ, il s'agissait de voir si les nouvelles méthodes de résolution de systèmes par voie itérative pouvaient être utilisées dans l'algorithme de Newton-Raphson de METAFOR pour accélérer la résolution de gros problèmes 3D. Ensuite, nous avons détaillé les caractéristiques du système d'équations et principalement les causes de l'absence de symétrie de la matrice de raideur tangente.

Grâce à la théorie des méthodes de projection décrite dans le chapitre 3, nous avons pu choisir les algorithmes les plus intéressants pour le problème donné. A ce moment-là, 6 solveurs sont candidats à une étude plus approfondie :

Tout d'abord le GMRES, découlant du processus d'orthogonalisation d'Arnoldi. C'est le solveur itératif le plus robuste que l'on puisse trouver : à chaque itération, il diminue le résidu du système quoi qu'il arrive. Cependant, il est très gourmand en mémoire et son paramètre de restart est difficile à fixer a priori.

Trois autres solveurs sont dérivés de la méthode de bi-orthogonalisation de Lanczos : le Bi-CG, caractérisé par une convergence irrégulière, le Bi-CGSTAB, sa version stabilisée et le CGS, théoriquement deux fois plus rapide que les deux précédents. Ils permettent de limiter les besoins de mémoire.

Les deux dernières méthodes considérées sont des solveurs symétriques pour traiter les cas où on symétrise la matrice de raideur. Il s'agit du gradient conjugué (CG), utilisé dans beaucoup de codes de calcul et SYMMLQ, la version symétrique du GMRES. Ces deux dernières méthodes ont deux inconvénients : le préconditionneur doit être symétrique et défini positif (or la matrice du système ne l'est pas) et de nombreux problèmes ne peuvent pas être résolus avec une matrice symétrisée.

Le chapitre 4 récapitule les différentes méthodes de préconditionnement les plus couramment utilisées. La plus grande famille de méthodes est basée sur une factorisation incomplète (ILU) de la matrice de raideur.

L'introduction de toutes ces méthodes dans le code de METAFOR n'a pas été facile parce qu'il fallait modifier la structure des données. En effet, si on veut gagner de la mémoire en utilisant un solveur itératif, il faut profiter du fait qu'ils ne font intervenir la matrice du système uniquement sous la forme d'un produit de celle-ci par un vecteur. Cela permet de garder en mémoire uniquement les éléments non nuls. On est donc obligé d'utiliser les formats de stockage particuliers décrits dans le chapitre 2.

Il ne reste plus qu'à tester tous ces algorithmes. C'est l'objet du chapitre 5.

D'après les essais réalisés, on peut dire que l'utilisation des solveurs itératifs est très avantageuse dans le cas de gros problèmes pour lesquels la largeur de bande de la matrice de raideur est très grande. Les structures cubiques maillées uniformément permettent d'obtenir des temps de calcul jusqu'à six fois plus petits que ceux fournis par le solveur direct. Grâce au format de stockage utilisé, on peut traiter des problèmes jusqu'à 20 000 degrés de liberté en utilisant moins de 11 millions de flottants double précision. Ces problèmes demanderaient une grande extension de mémoire pour pouvoir être traités par voie directe.

D'un autre côté, on trouve les problèmes 2D. Ils sont caractérisés par une faible largeur de bande. Le solveur direct donne alors le temps de calcul et l'occupation mémoire minimale. Il est donc inutile de s'acharner à essayer de résoudre plus vite ce genre de problème.

Entre ces deux extrêmes, le choix du solveur optimal n'est pas toujours facile. D'habitude, le solveur itératif devient assez vite attractif au point de vue de la mémoire lorsqu'on augmente le nombre de degrés de liberté de la structure. Si on veut concurrencer le solveur direct au niveau du temps de calcul, il faut augmenter encore la taille du problème.

Pour ces problèmes intermédiaires, il est difficile de choisir correctement la valeur des paramètres caractérisant le solveur, le préconditionneur et le critère d'arrêt. Un mauvais choix peut conduire à une utilisation abusive de mémoire et une convergence beaucoup trop lente. L'utilisateur doit se familiariser avec tous les phénomènes décrits dans ce travail pour pouvoir utiliser efficacement les méthodes itératives.

Si on compare maintenant les solveurs entre eux, on remarque que le GMRES est presque toujours la méthode donnant le temps de calcul le plus faible. Cependant, la grande quantité de mémoire requise pousse à utiliser des variantes moins gourmandes telles que le Bi-CG ou le Bi-CGSTAB lorsque l'on joue avec les limites de la machine (plus de 20 000 d.d.l.).

La métode CGS est relativement moins puissante en pratique et son utilisation est déconseillée.
Quant aux solveurs symétriques, CG et SYMMLQ, ils ne font pas le poids face aux méthodes itératives non symétriques pour les deux raisons évoquées plus haut.

En ce qui concerne le préconditionnement, la méthode la plus intéressante est ILUT(*lfil*) qui permet, grâce à son paramètre de remplissage, de contrôler la qualité du préconditionnement et le temps passé à calculer la factorisation incomplète.

6.2 Perspectives

Ce travail n'a pas la prétention d'avoir couvert toutes les méthodes existantes ainsi que tous les phénomènes en découlant. Citons, pour terminer, les améliorations que l'on pourrait apporter.

- Etude d'autres solveurs :

Il existe, bien sûr, d'autres méthodes qui pourraient être utilisées à la place des solveurs considérés dans ce travail. Par exemple, le QMR, Quasi-Minimal Residual, que nous avons décrit très brièvement dans la partie théorique. Cet algorithme fait l'objet de nombreuses recherches à l'heure actuelle pour lui donner la rapidité de convergence du GMRES en utilisant une mémoire réduite comme le Bi-CG. De plus, l'utilisation simultanée du 'look ahead' permet d'améliorer sa robustesse malgré le fait qu'il se base sur la dangereuse bi-orthogonalisation de Lanczos.

- Mise au point d'autres préconditionneurs :

Chaque année, de nouvelles méthodes de préconditionnement voient le jour. Cependant, elles ont toutes un domaine d'action plus ou moins réduit : par exemple, ILU(0) est souvent employé avec succès lors de la résolution d'équations aux dérivées partielles ; par contre, nous avons vu que cette méthode est souvent inadaptée pour résoudre les systèmes linéarisés de la mécanique du solide.
Nous n'avons pas eu le temps d'essayer toutes les méthodes. Il existe, par exemple, une méthode développée par Saad [S3] pour préconditionner les équations normales grâce à une décomposition LQ de la matrice du système.

- Modification du processus de Newton-Raphson :

Dans ce travail, nous nous sommes focalisés sur la résolution d'un système linéarisé. Comme nous l'expliquons dans l'introduction, il serait peut être intéressant de choisir une autre

méthode de résolution que l'algorithme de Newton-Raphson. Il existe beaucoup de variantes telles que toutes les méthodes de Quasi-Newton.

- Etude plus poussée des problèmes de contact à trois dimensions :

Lorsque les lois de contact 3D seront opérationnelles, il faudra analyser le comportement des solveurs itératifs face à celles-ci sur des problèmes de grande taille. En effet, la plupart des problèmes réels font intervenir des écrasements et des emboutissages. Il serait donc très décevant d'obtenir de mauvais résultats.
Grâce aux tests envisagés en 2D, nous pouvons être presque certains que l'utilisation des lois de contact ne posera pas de problèmes majeurs pour les solveurs itératifs.

- Recherche d'un critère d'arrêt plus simple à utiliser :

Le critère d'arrêt considéré ici nécessite le choix de deux paramètres et ce n'est pas toujours facile de déterminer des valeurs optimales. Cependant, il donne en général de bon résultats c'est pourquoi nous n'avons pas approfondi la question. D'autres critères permettraient peut-être d'éviter l'intervention de l'utilisateur.

- Inclure le solveur dans une boîte noire :

Lorsque l'influence des paramètres sera mieux connue, il sera peut-être possible de laisser choisir à METAFOR le type de solveur à utiliser selon certaines caractéristiques du problème (largeur de bande de la matrice de raideur, loi de comportement, ...).

7. Bibliographie

[B1] Barret R., M. Berry, Tony Chan, ... : *'Templates for the solution of linear systems : Building blocks for iterative methods'*.

[B2] Bruaset A.M. : *'A survey of preconditioned iterative methods'*, Editions Longman Scientific & Technical.

[C1] Colantonio L. : *'Finite element simulation of cold compaction of powders'*, Report TA-31, Ulg (1996).

[C2] Crisfield M.A. : *'An arc-lengh method including line searches and accelerations'*, International Journal for numerical methods in engineering, vol 19, pp 1269-1289 (1983)

[D1] Delanaye M. : 'Polynomial reconstruction finite volume schemes for the compressible Euler and Navier-Stokes equations on unstructured adaptative grids', Thèse de doctorat, Ulg (1997)

[H1] Hibbit, Karlsson & Sorensen : *'ABAQUS / News : The Newsletter for ABAQUS Users'*, (Winter 1995).

[H2] Hogge M. : *'Méthode des éléments finis'*, Notes de cours, Ulg (1994)

[I1] Irons B.M. : *'A frontal solution program for finite element analysis'*, International Journal for Numerical Methods in Engineering, vol 2, pp 5-32 (1970)

[K1] Kelley C.T. : *'Iterative methods for linear and nonlinear equations'*, North Carolina State University (1995)

[K2] Koppenhoefer, Gullerud, Ruggieri, Dodds : *'WARP3D-Release 9.4 : Dynamic Nonlinear Analysis of Solids Using a Preconditioned Conjugate Gradient Software Architecture'*, NASA-AMES research center Moffett field, California

[L1] Letniowski W. : *'An overview of preconditioned iterative methods for sparse matrix equations'*, Research Report, University of Waterloo, Canada (1989)

[L2] Litt F.-X. : *'Analyse numérique (1ère partie)'*, notes de cours, Ulg (1995).

[N1] Nachtigal N.M. : *'A look-ahead variant of the Lanczos algorithm and its application to the quasi-minimal residual method for non-Hermitian linear systems'*, Ph.D dissertation, Massachussets Institute of Technology, Cambrige (1991).

[N2] Nachtigal N. M. : QMRPACK, netlib

[P1] Paige and Saunders : *'Solution of sparse indefinite systems of linear equations'*, SIAM Journal of Numerical Analysis 12, pp 617-629 (1975).

[P2] Papadrakakis M. : *'Solution techniques for large-scale finite element analysis'*, National Technical Institute of Athens.

[P3] Ponthot J.-P.: *'Introduction à l'algorithme « SKYLINE » dans METAFOR'*, Ulg (1989).

[P4] Ponthot J.-P. : *'Traitement unifié de la mécanique des milieux continus solides en grandes transformations par la méthode des éléments finis'*, Thèse de doctorat, Ulg (1995)

[P5] Ponthot J.-P. : *'Mode d'emploi pour la version pilote de « METAFOR », module de calcul en grandes déformations'*, Rapport TF-24, Version 0.1 (1992)

[P6] Ponthot J.-P. : *'Modélisation des processus de formage secondaire des matériaux'*, Notes provisoires, Ulg (1995)

[R1] Ricks E. : *'An incremental approach to the solution of snapping and buckling problems'*, Int. J. Solids Structures, vol 15, pp 529-551 (1978)

[S1] Saad Y. : *'Krylov subspace techniques, conjugate gradients, preconditioning and sparse matrix solvers'*, University of Minesota, USA.

[S2] Saad Y. : *'SPARSKIT : a basic tool kit for sparse matrix computations (version 2)'*.

[S3] Saad Y. : 'Preconditioning techniques for non symmetric and indefinite linear systems', Journal of Computationnal and Applied Mathematics 24, pp 89-105 (1988)

[S4] Saint-Georges P. and Warzee G. : '*Developpement of an efficient iterative solver for linear systems in FE structural analysis'*, ULB.

[S5] Saint-Georges P., Warzee G., Notay Y., Beauwens R. : *'High performance solvers for FEM structural analysis'*, International Jounal for numerical methods in engineering, vol 39, pp 1313-1340 (1996).

[S6] Saint-Georges P., Warzee G., Notay Y., Beauwens R. : *'Fast iterative solvers for FE analysis in general and shell analysis in particular'*, Advances in Finite Element Technology, pp 273-282.

[S6] Sloan S.W. : *'A FORTRAN program for profile and wavefront reduction'*, University of Newcastle, Australia (1989)

[T1] Tsarnaoussis C. : *'Développement d'un élément fini à comportement non linéaire pour la modélisation des structures collées (critère de Raghava)'*, TFE, Ulg (1996).

[V1] Van der Vorst A. and Dekker K. : *'Conjugate gradient type methods and preconditionning'*, Delft University of Technology, Netherlands.

8. Table des matières