Généralités sur le gradient conjugué : GCPC *Aster* et utilisation de PETSc

Résumé :

Dans le cadre des simulations thermo-mécaniques, l'essentiel des coûts calcul provient souvent de la construction et de la résolution des systèmes linéaires. Le choix du bon solveur linéaire est donc primordial, d'une part pour sa rapidité, mais aussi pour sa robustesse et la place mémoire qu'il requiert. A chaque fois, un compromis est à opérer entre ces contraintes.
Depuis 60 ans deux types de solveurs se disputent la suprématie dans le domaine, les directs et ceux itératifs. D'où, par précaution, une offre diversifiée des codes de mécanique en la matière. *code_aster* n'échappe pas à la règle. Il propose trois solveurs directs (Gauss [R6.02.01], multifrontale native [R6.02.02] et le produit externe MUMPS [R6.02.03]), des solveurs itératifs de type Krylov (GCPC Aster et tous ceux de la librairie PETSc).

Dans ce document, **on détaille** d'un point de vue théorique, algorithmique et modélisations *code_aster*, **les fondamentaux du GCPC et des autres solveurs de Krylov implantés dans PETSc**. En particulier, on met en lumière leurs liens avec les autres solveurs linéaires du code et les méthodes d'optimisation continue. On poursuit par leurs difficultés d'implantation, leurs paramétrages, leurs périmètres ainsi que quelques conseils d'utilisation.

Pour plus de détails et de conseils sur l'emploi des solveurs linéaires on pourra consulter les notices d'utilisation spécifiques [U4.50.01]/[U2.08.03]. Les problématiques connexes d'amélioration des performances (RAM/CPU) d'un calcul et, de l'utilisation du parallélisme, font aussi l'objet de notices détaillées: [U1.03.03] et [U2.08.06].

Table des Matières

1 Problématique

En simulation numérique de phénomènes physiques, **un coût calcul important provient souvent de la construction et de la résolution de systèmes linéaires**. La mécanique des structures n'échappe pas à la règle ! Le coût de la construction du système dépend du nombre de points d'intégration et de la complexité des lois de comportement, tandis que celui de la résolution est lié au nombre d'inconnues, aux modélisations retenues et à la topologie (largeur de bande, conditionnement). Lorsque le nombre d'inconnues explose, la seconde étape devient prépondérante[1] et c'est donc cette dernière qui va principalement nous intéresser ici. D'ailleurs, lorsqu'il est possible d'être plus performant sur cette phase de résolution, grâce à l'accès à une machine parallèle, cet atout peut se propager à la phase de constitution du système proprement dite (calculs élémentaires et assemblages, cf. [U2.08.06]).
Ces **inversions de systèmes linéaires sont en fait omniprésentes dans les codes et souvent enfouies au plus profond d'autres algorithmes numériques**: schéma non-linéaire, intégration en temps, analyse modale…. On cherche, par exemple, le vecteur des déplacements nodaux (ou des incréments de déplacement) u vérifiant un système linéaire du type

$$Ku = f \tag{1-1}$$

avec K une matrice de rigidité et f un vecteur traduisant l'application de forces généralisées au système mécanique.

De manière générale, la **résolution de ce type de problème requiert un (plus) large questionnement** qu'il n'y parait :

- A-t-on accès à la matrice ou connaît-on simplement son action sur un vecteur ?
- Cette matrice est elle creuse ou dense ?
- Quelles sont ses propriétés numériques (symétrie, définie positive…) et structurelles (réelle/complexe, par bandes, par blocs..) ?
- Veux-t-on résoudre un seul système (1-1), plusieurs en simultané[2] ou de manière consécutive[3] ? Voire plusieurs systèmes différents et successifs dont les matrices sont très proches[4] ?
- Dans le cas de résolutions successives, peut-on réutiliser des résultats précédents afin de faciliter les prochaines résolutions (cf. technique de redémarrage, factorisation partielle) ?
- Quel est l'ordre de grandeur de la taille du problème, de la matrice et de sa factorisée par rapport aux capacités de traitements des CPU et des mémoires associées (RAM, disque) ?
- Veut-on une solution très précise ou simplement une estimation (cf. solveurs emboîtés) ?
- A-t-on accès à des librairies d'algèbre linéaire (et à leurs pré-requis MPI, BLAS, LAPACK…) ou doit-on faire appel à des produits «maison» ?

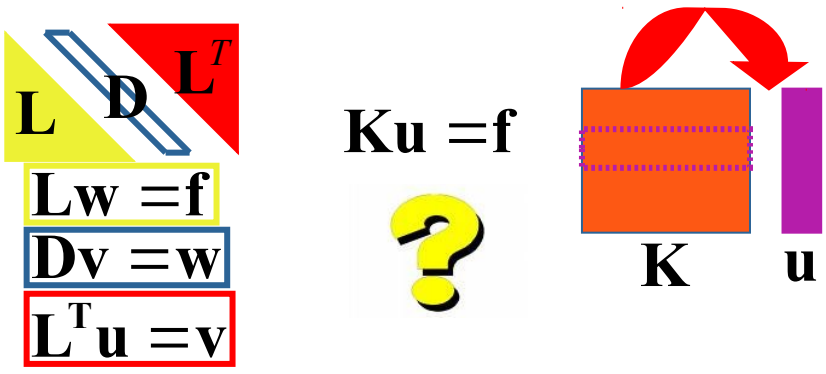
Dans *Code_Aster* , on construit explicitement la matrice et on la stocke au format MORSE[5] . Avec la plupart des modélisations, la matrice est creuse (du fait de la discrétisation par éléments finis), potentiellement mal conditionnée[6] et souvent réelle, symétrique et indéfinie[7] . En non linéaire, en modal ou lors de chaînages thermo-mécaniques, on traite souvent des problèmes de type «multiples seconds membres». Les méthodes discrètes de contact-frottement tirent profit de facultés de factorisation partielle. D'autre part, *Code_Aster* utilise

1 Pour *Code_Aster*, voir l'étude de 'profiling' conduite par N.Anfaoui[Anf03].
2 Même matrice mais plusieurs seconds membres indépendants; Cf. construction d'un complément de Schur.
3 Problème de type multiples seconds membres: même matrice mais plusieurs seconds membres successifs et interdépendants; Cf. algorithme de Newton sans réactualisation de la matrice tangente.
4 Problème de type multiples premiers membres : plusieurs matrices et seconds membres successifs et interdépendants, mais avec des matrices «spectralement» proches ; Cf. algorithme de Newton avec réactualisation de la matrice tangente.
5 Dit encore SCC pour 'Symmetric Compressed Column storage'.
6 En mécanique des structures le conditionnement $\eta(K)$ est connu pour être assez mauvais. Il peut varier, typiquement, de 10^5 à 10^{12} et le fait de raffiner le maillage, d'utiliser des éléments étirés ou des éléments structuraux a des conséquences dramatiques sur ce chiffre (cf. B.Smith. *A parallel implementation of an iterative substructuring algorithm for problems in 3D.* SIAM J.Sci.Comput., **14** (1992), pp406-423. §3.1 ou I.FRIED. *Condition of finite element matrices generated from nonuniform meshes*. AIAA J., **10** (1972), pp219-221.)
7 Le caractère indéfini plutôt que défini positif est du à l'adjonction de variables supplémentaires (dites «de Lagrange») pour imposer des conditions limites de Dirichlet simples ou généralisées[R3.03.01].

aussi des scénarios de résolutions simultanées (compléments de Schur du contact et de la sous-structuration…).

Quant aux tailles des problèmes, même si elles enflent d'année en année, elles restent modestes par rapport à la CFD : de l'ordre du million d'inconnues mais pour des centaines de pas de temps ou d'itérations de Newton.
Par ailleurs, d'un **point de vue «middleware et hardware»** , le code s'appuie désormais sur de nombreuses librairies optimisées et pérennes (MPI, BLAS, LAPACK, METIS, SCOTCH, PETSc, MUMPS…) et s'utilise principalement sur des clusters de SMP (réseaux rapides, grande capacité de stockage RAM et disque). On cherche donc surtout à optimiser l'utilisation des solveurs linéaires dans cette optique.

Depuis 60 ans, deux types de techniques se disputent la suprématie dans le domaine, les solveurs directs et ceux itératifs (cf. [Che05][Dav06][Duf06][Gol96][LT98][Liu89][Meu99][Saa03]).
Les premiers sont robustes et aboutissent en un nombre fini d'opérations (théoriquement) connu par avance. Leur théorie est relativement bien achevée et leur déclinaison suivant moult types de matrices et d'architectures logicielles est très complète. En particulier, leur algorithmique multiniveau est bien adaptée aux hiérarchies mémoires des machines actuelles. Cependant, ils requièrent des capacités de stockage qui croissent rapidement avec la taille du problème ce qui limite l'extensibilité de leur parallélisme[8] . Même si ce parallélisme peut se décomposer en plusieurs strates indépendantes, démultipliant ainsi les performances.
En revanche, les **méthodes itératives** sont plus «scalables» lorsque l'on augmente le nombre de processeurs. Leur théorie regorge de nombreux «problèmes ouverts», surtout en arithmétique finie. En pratique, leur convergence en un nombre «raisonnable» d'itérations, n'est pas toujours acquise, elle dépend de la structure de la matrice, du point de départ, du critère d'arrêt… Ce type de solveurs a plus de mal à percer en mécanique des structures industrielles où l'on cumule souvent hétérogénéités, non-linéarités et jonctions de modèles qui gangrènent le conditionnement de l'opérateur de travail. D'autre part, elles ne sont pas adaptées pour résoudre efficacement les problèmes de type «multiples seconds membres». Hors ceux-ci sont très fréquents dans l'algorithmique des simulations mécaniques.
Contrairement à leurs homologues directs, il n'est pas possible de proposer LE solveur itératif qui va résoudre n'importe quel système linéaire. L'adéquation du type d'algorithme à une classe de problèmes se fait au cas par cas. Ils présentent, néanmoins, d'autres avantages qui historiquement leur ont donné droit de cité pour certaines applications. A gestion mémoire équivalente, ils en requièrent moins que les solveurs directs, car on a juste besoin de connaître l'action de la matrice sur un vecteur quelconque, sans devoir véritablement la stocker. D'autre part, on n'est pas soumis au «diktat» du phénomène de remplissage qui détériore le profil des matrices, on peut exploiter efficacement le caractère creux des opérateurs et contrôler la précision des résultats[9] .
Dans la littérature en analyse numérique, on accorde souvent une place prépondérante aux solveurs itératifs en général, et, aux variantes du gradient conjugué en particulier. **Les auteurs les plus chevronnés s'accordent à dire que, même si son utilisation gagne au fil des ans et des variantes, des «parts de marché», certains domaines restent encore réfractaires. Parmi ceux-ci, la mécanique des structures, la simulation de circuit électronique…**
Pour paraphraser ces auteurs, l'utilisation de solveurs directs relève du domaine de la technique alors que le choix du bon couple méthode itérative/préconditionneur est plutôt un art ! En dépit de sa simplicité biblique sur le papier, la résolution d'un système linéaire, même symétrique défini positif, n'est pas «un long fleuve tranquille». Entre deux maux, remplissage/pivotage et préconditionnement, il faut choisir !



8 On parle aussi de «scalabilité» ou de passage à l'échelle.
9 Ce qui peut être très intéressant dans le cadre de solveurs emboîtés (par ex. Newton + GCPC), cf. V.Frayssé. *The power of backward error analysis*. HDR de l'Institut National Polytechnique de Toulouse (2000).

Figure 1-1._ Deux classes de méthodes pour résoudre un système linéaire: les directes et les itératives.

Dernièrement l'arrivée des solveurs hybrides[Boi08], qui mixent les deux approches pour tirer partie de leurs avantages respectifs, bouleverse un peu la donne. Sur les «simulations frontières» comportant des dizaines de millions d'inconnues, le gradient conjugué peut se voire concurrencer par ces nouvelles approches (surtout pour des problèmes de type multiple second-membres).

Mais en fait il est généralement présent en tant que solveur d'interface de ces hybrides. Ceux-ci peuvent être vus comme des préconditionneurs particuliers (multiniveaux, multigrille, DD) d'un gradient conjugué ou de toutes autres méthodes de Krylov (GMRES, BiCG-Stab…). D'autre part, compte-tenu de sa simplicité de mise en œuvre (en séquentiel comme en parallèle) et de sa faible occupation mémoire, il reste souvent le seule alternative pour aborder les très gros problèmes[10] .

Remarques :

- *Les deux grandes familles de solveurs doivent plus être vues comme complémentaires que comme concurrentes. On cherche souvent à les mixer: méthodes DD[Boi08], préconditionneur par factorisation incomplète (cf. § 4.2) ou de type multigrille[Tar09], procédure de raffinement itératif en fin de solveur direct[Boi09]…*
- *Pour plus de détails sur les librairies d'algèbre linéaire implémentant ces méthodes et quelques résultats de benchmarks les confrontant, on pourra consulter la documentation [Boi09]. Mais pour vraiment tirer partie de ces produits, il faut les exploiter en mode parallèle. La documentation [Boi08] illustre quelques aspects du calcul parallèle (hardware, méthodologie, langages, indicateurs de performances) et précise les différents niveaux de parallélisme exploitables dans un code tel que Code_Aster.*

Abordons maintenant l'ensemble des chapitres autour desquels va s'articuler cette note. Après avoir explicité les **différentes formulations du problème** (système linéaire, minimisations de fonctionnelle) et afin de mieux faire sentir au lecteur les subtilités implicites des méthodes de type gradient, on propose un survol rapide de leurs fondamentaux : la classique «steepest-descent» puis les **méthodes de descente générales** ainsi que leurs liens avec le GC. Ces rappels étant faits, le **déroulement algorithmique du GC** devient clair (du moins on l'espère !) et **ses propriétés théoriques** de convergence, d'orthogonalité et de complexité en découlent. Des compléments sont rapidement brossés pour mettre en perspective le GC avec des notions récurrentes en analyse numérique : projection de Petrov-Galerkin et espace de Krylov, problème d'orthogonalisation, équivalence avec le méthode de Lanczos et propriétés spectrales, solveurs emboîtés et parallélisme.
Puis on détaille le «mal nécessaire» que constitue le **préconditionnement de l'opérateur de travail**, quelques stratégies souvent usitées et celles retenues pour le GCPC natif de *Code_Aster* et pour les solveurs de PETSc.
On conclut par **les difficultés d'implantation particulières dans le code**, les questions de **paramétrage** et de **périmètre** ainsi que par quelques **conseils d'utilisation**.

Nota :

- *L'implantation effective et la maintenance des solveurs itératifs dans Code_Aster est le fruit d'un travail d'équipe : J.P.Grégoire, J.Pellet, T.DeSoza, N.Tardieu, O.Boiteau…*
- *Dans ce document, bon nombre de figures ont été empruntées au papier introductif de J.R. Shewchuk [Sh94] avec son aimable autorisation : ©1994 by Jonathan Richard Shewchuk.*

10 Tout est relatif. Les limites actuelles sont plutôt de quelques millions d'inconnues pour un PC et de plusieurs centaines de millions pour une machine de centre de calcul. Les grands codes de CFD (pour EDF : *Code_Saturne*, TELEMAC…) sont ainsi complètement adossés à des solveurs itératifs de type gradient conjugué. Ils leurs permettent de distribuer des flots de données/traitements sur des milliers de processeurs et de gérer ainsi, *via* le parallélisme, des problèmes de très grande taille.

2 Les méthodes de descente

2.1 Positionnement du problème

Soit $\mathbf{K}$ la matrice de rigidité (de taille N) à inverser, si elle a le «bon goût» d'être symétrique définie positive (SPD dans la littérature anglo-saxonne), ce qui est souvent[11] le cas en mécanique des structures, on montre par simple dérivation que les problèmes suivants sont équivalents :

- Résolution du système linéaire habituel avec $\mathbf{u}$ vecteur solution (des déplacements ou incréments de déplacements, resp. en température….) et f vecteur traduisant l'application de forces généralisées au système thermo-mécanique

$$\left(P_1\right) \quad \mathbf{Ku}=f \tag{2.1-1}$$

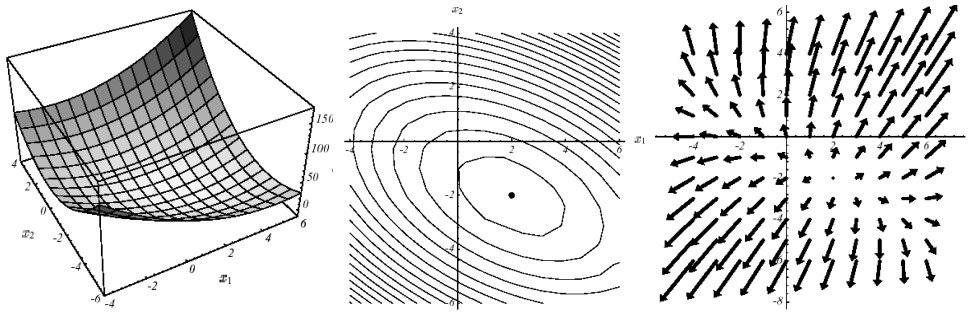
- Minimisation de la fonctionnelle quadratique représentant l'énergie du système, avec $<,>$ le produit scalaire euclidien usuel,

$$\left(P_2\right) \quad u=Arg\min_{v\in\Re^N} J\left(v\right)$$
$$\text{avec } J\left(v\right):=\frac{1}{2}\left\langle \mathbf{v},\mathbf{Kv}\right\rangle-\left\langle f,v\right\rangle=\frac{1}{2}v^T\mathbf{Kv}-f^T v \tag{2.1-2}$$

Du fait de la «définie-positivité» de la matrice qui rend J strictement convexe, le vecteur annulant[12] ∇J correspond à l'unique[13] minimum global $\mathbf{u}$. Cela s'illustre par la relation suivante, valable quelle que soit $\mathbf{K}$ symétrique,

$$J\left(\mathbf{v}\right)=J\left(\mathbf{u}\right)+\frac{1}{2}\left(\mathbf{v}-\mathbf{u}\right)^T\mathbf{K}\left(\mathbf{v}-\mathbf{u}\right) \tag{2.1-3}$$

Ainsi, pour tout vecteur $\mathbf{v}$ différent de la solution $\mathbf{u}$, le caractère défini positif de l'opérateur rend strictement positif le second terme et donc $\mathbf{u}$ est aussi un minimum global.



(a) (b) (c)

Figure 2.1-1 . Exemple de J quadratique en $N=2$ dimensions avec $K:=\begin{bmatrix}3 & 2\\ 2 & 6\end{bmatrix}$ et $f:=\begin{bmatrix}2\\ -8\end{bmatrix}$:
graphe (a), lignes de niveaux (b) et vecteurs gradient (c).
Le spectre de l'opérateur est $(\lambda_1;v_1)=(7;[1,2]^T)$ et $(\lambda_2;v_2)=(2;[-2,1]^T)$.

Ce résultat très important en pratique s'appuie entièrement sur cette fameuse définie-positivité propriété un peu «éthérée» de la matrice de travail. Sur un problème à deux dimensions ($N=2$!) on peut s'en faire une

11 Pour une simulation *via Code_Aster*, la matrice est souvent indéfinie du fait de l'adjonction de variables supplémentaires (dites «de Lagranges») pour imposer des conditions limites[Pel01].

12 L'annulation de la dérivée est un cas particulier au cas convexe et sans contrainte des fameuses relations de Kuhn-Tucker caractérisant l'optimum d'un problème d'optimisation différentiable. Elle est appelée «équation d'Euler».

13 Sans cette convexité, on est pas assuré de l'unicité. Il faut alors composer avec des minima locaux !

représentation limpide (cf. figure 2.1-1) : la forme paraboloïde qui focalise l'unique minimum au point $[2,-2]^T$ de pente nulle.

- Minimisation de la norme en énergie de l'erreur $e(v)=v-u$, plus parlante pour les mécaniciens,

$$(P_3)\quad u=Arg\min_{v\in\Re^N} E(v)$$
$$\text{avec } E(v):=\langle \mathbf{K}e(v), e(v)\rangle=(\mathbf{K}e(v))^T e(v)=\|e(v)\|_K^2 \tag{2.1-4}$$

D'un point de vue mathématique, ce n'est rien d'autre qu'une norme matricielle (licite puisque $\mathbf{K}$ est SPD). On préfère souvent l'exprimer *via* un résidu $r(v)=f-\mathbf{K}v$

$$E(v)=\langle r(v), K^{-1}r(v)\rangle=r(v)^T K^{-1} r(v) \tag{2.1-5}$$

Remarques :

- *Le périmètre d'utilisation du gradient conjugué peut en fait s'étendre à tout opérateur, pas forcément symétrique ou défini positif voire même carré ! Pour ce faire on définit la solution de (P_1) comme étant celle, au sens des moindres carrés, du problème de minimisation*

$$(P_2)'\quad u=Arg\min_{v\in\Re^N}\|\mathbf{K}v-f\|^2 \tag{2.1-6}$$

Par dérivation on aboutit aux équations dites «normales» dont l'opérateur est carré, symétrique et positif

$$(P_2)''\quad \underbrace{\mathbf{K}^\mathbf{T}\mathbf{K}}_{\tilde{\mathbf{K}}}\mathbf{u}=\mathbf{K}^\mathbf{T}\mathbf{f} \tag{2.1-7}$$

On peut donc lui appliquer un GC ou une steepest-descent sans trop d'encombres.

- *La solution du problème (P_1) est à l'intersection de N hyperplans de dimension $N-1$. Par exemple, pour le cas de la figure 2.1-1, elle s'exprime trivialement sous forme d'intersections de droites:*

$$\begin{cases} 3x_1+2x_2=2 \\ 2x_1+6x_2=-8 \end{cases} \tag{2.1-8}$$

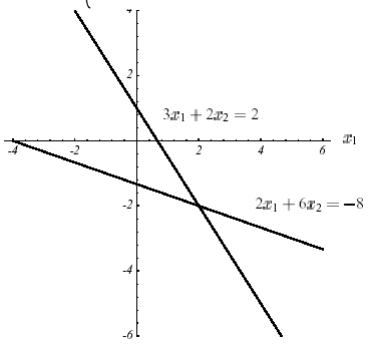


Figure 2.1-2. Résolution de l'exemple n°1 par intersection de droites.

- *Les méthodes de type gradient se démarquent de cette philosophie, elles s'inscrivent naturellement dans le cadre de minimisation d'une fonctionnelle quadratique, dans lequel elles ont été développées et intuitivement comprises.*

2.2 Steepest Descent

2.2.1 Principe

Cette dernière remarque préfigure la philosophie de la méthode dite «de la plus forte pente», plus connue sous sa dénomination anglo-saxonne de 'Steepest Descent': on construit la suite d'itérés $\mathbf{u}^i$ en suivant la direction

suivant laquelle J décroît le plus, du moins localement, c'est-à-dire $d^i=-\nabla J^i=r^i$ avec $J^i:=J\left(u^i\right)$ et $r^i:=f-\mathbf{K}\mathbf{u}^i$. A la $i^{ème}$ itération, on va donc chercher à construire $\mathbf{u}^{i+1}$ tel que :

$$u^{i+1}:=u^i+\alpha^i r^i \tag{2.2-1}$$

et

$$J^{i+1}<J^i \tag{2.2-2}$$

Grâce à cette formulation, on a donc transformé un problème de minimisation quadratique de taille N (en J et $\mathbf{u}$) en une minimisation unidimensionnelle (en G et α)

$$\begin{aligned}&\text{Trouver } \alpha^i \text{ tel que } \alpha^i=Arg\min_{\alpha\in[\alpha_m,\alpha_M]} G^i(\alpha)\\&\text{avec } G^i(\alpha):=J\left(u^i+\alpha\ r^i\right)\end{aligned} \tag{2.2-3}$$

Les figures suivantes illustrent le fonctionnement de cette procédure sur l'exemple n°1 : partant du point $u^0=[-2,-2]^T$ (cf. (a)) on cherche le paramètre de descente optimal, α^0 , suivant la ligne de plus grande pente r^0 ; ce qui revient à chercher un point appartenant à l'intersection d'un plan vertical et d'une paraboloïde (b), signifiée par la parabole (c). Trivialement ce point annule la dérivée de la parabole (d)

$$\frac{\partial G^0(\alpha^0)}{\partial\alpha}=0\Leftrightarrow\langle\nabla J(u^1),r^0\rangle=0\Leftrightarrow\langle r^1,r^0\rangle=0\Leftrightarrow\alpha^0:=\frac{\|r^0\|^2}{\langle r^0,\mathbf{K}\mathbf{r}^0\rangle} \tag{2.2-4}$$

Cette orthogonalité entre deux résidus successifs (i.e gradients successifs) produit un cheminement caractéristique, dit en «zigzag», vers la solution (e). Ainsi, dans le cas d'un système mal conditionné produisant des ellipses étroites et allongées[14], le nombre d'itérations requises peut être considérable (cf. figure 2.2-1).

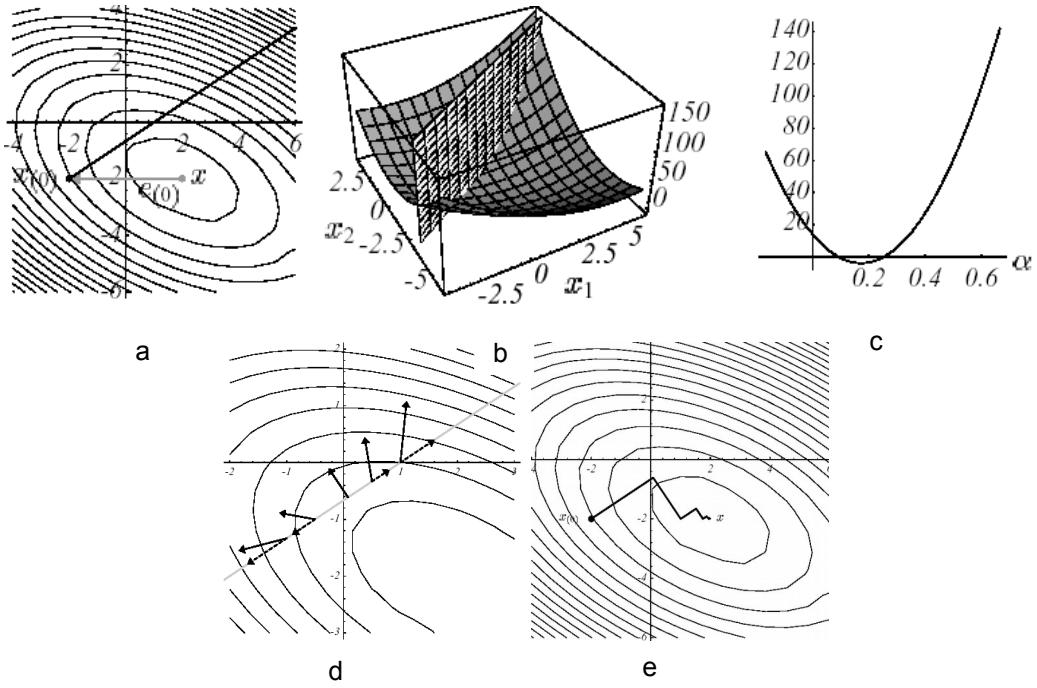


Figure 2.2-1. Illustration de la Steepest Descent sur l'exemple n°1 : direction de descente initiale (a), intersection de surfaces (b), parabole correspondante (c), vecteurs gradient et leur projection le long de la direction de descente initiale (d) et processus global jusqu'à la convergence (e).

2.2.2 Algorithme

14 Le conditionnement de l'opérateur K s'écrit comme le rapport de ses valeurs propres extrêmes $\eta(K):=\frac{\lambda_{\max}}{\lambda_{\min}}$ qui sont elles-mêmes proportionnelles aux axes des ellipses. D'où un lien direct et visuel entre mauvais conditionnement matriciel et vallée étroite et tortueuse où la minimisation est malmenée.

D'où le déroulement

Initialisation u^0 donné
Boucle en i
(1) $r^i = f - \mathbf{K}\mathbf{u}^i$ (nouveau résidu)
(2) $\alpha^i = \dfrac{\|r^i\|^2}{\langle r^i, \mathbf{K}\mathbf{r}^i \rangle}$ (paramètre optimal de descente)
(3) $u^{i+1} = u^i + \alpha^i r^i$ (nouvel itéré)
(4) Test d'arrêt via $J^{i+1} - J^i$ (par exemple)

Algorithme 1 : Steepest Descent.

Pour économiser un produit matrice-vecteur il est préférable de substituer à l'étape (1), la mise à jour du résidu suivante

$$r^{i+1} = r^i - \alpha^i \mathbf{K}\mathbf{r}^i \tag{2.2-5}$$

Toutefois, afin éviter des accumulations d'arrondis intempestives, on recalcule périodiquement le résidu avec la formule initiale (1).

2.2.3 Eléments de théorie

On montre que cet algorithme converge, pour tout point de départ u^0, à la vitesse[15]

$$\|e(u^{i+1})\|_K^2 = \omega^i \|e(u^i)\|_K^2$$
$$\text{avec } \omega^i := 1 - \frac{\|r^i\|^4}{\langle K^{-1} r^i, r^i \rangle \langle \mathbf{K}\mathbf{r}^i, r^i \rangle} \tag{2.2-6}$$

En développant l'erreur sur la base des modes propres $(\lambda_j ; v_j)$ de la matrice $\mathbf{K}$

$$e(u^i) = \sum_j \xi_j v_j \tag{2.2-7}$$

le facteur d'atténuation de l'erreur en énergie devient

$$\omega^i = 1 - \frac{\left(\sum_j \xi_j^2 \lambda_j^2\right)^2}{\left(\sum_j \xi_j^2 \lambda_j^3\right)\left(\sum_j \xi_j^2 \lambda_j\right)} \tag{2.2-8}$$

Dans (2.2-8), le fait que les composantes ξ_j soient au carré assure l'éviction prioritaire des valeurs propres dominantes. On retrouve ici une des caractéristiques des méthodes modales de type Krylov (Lanczos, Arnoldi cf. [Boi01] §5/§6) qui privilégient les modes propres extrêmes. Pour cette raison, la Steepest Descent et le gradient conjugué sont dits «grossiers» comparés aux solveurs itératifs classiques (Jacobi, Gauss-Seidel, SOR…) plus «lisses» car éliminant sans discrimination toutes les composantes à chaque itération.
Finalement, grâce à l'inégalité de Kantorovitch[16], on améliore grandement la lisibilité du facteur d'atténuation. Au terme de *i* itérations, au pire, la décroissance s'exprime sous la forme

$$\|e(u^i)\|_K \leq \left(\frac{\eta(K)-1}{\eta(K)+1}\right)^i \|e(u^0)\|_K \tag{2.2-9}$$

Elle assure la convergence linéaire[17] du processus en un nombre d'itérations proportionnel au conditionnement de l'opérateur. Ainsi, pour obtenir

15 La définie positivité de l'opérateur nous assure de la «bonne nature» du paramètre ω^i. C'est bien un facteur d'atténuation car $0 \leq \omega^i \leq 1$.

16 Quelque soit K matrice SPD et u vecteur non nul : $1 \leq \dfrac{\langle \mathbf{K}\mathbf{u}, u \rangle \langle K^{-1}u, u \rangle}{\|u\|_2^4} \leq \dfrac{\left(\eta(K)^{1/2} + \eta(K)^{-1/2}\right)^2}{4}$).

$$\frac{\|e(u^i)\|_K}{\|e(u^0)\|_K} \leq \varepsilon \ \ (\text{petit}) \tag{2.2-10}$$

il faut un nombre d'itérations de l'ordre de

$$i \approx \frac{\eta(K)}{4} \ln \frac{1}{\varepsilon} \tag{2.2-11}$$

Un problème mal conditionné ralentira donc la convergence du processus, ce que l'on avait déjà «visuellement» constaté avec le phénomène de « vallée étroite et tortueuse ».

Pour mieux appréhender l'implication du spectre de l'opérateur et du point de départ dans le déroulement de l'algorithme, simplifions la formule (2.2-8) en se plaçant dans le cas trivial où $N=2$. En notant $\kappa = \frac{\lambda_1}{\lambda_2}$ le conditionnement matriciel de l'opérateur et $\mu = \frac{\xi_2}{\xi_1}$ la pente de l'erreur à la $i^{\text{ème}}$ itération (dans le système de coordonnées des deux vecteurs propres), on obtient une expression plus lisible du facteur d'atténuation de l'erreur (cf. figure 2.2-2)

$$\omega^i = 1 - \frac{\left(\kappa^2+\mu^2\right)^2}{\left(\kappa+\mu^2\right)\left(\kappa^3+\mu^2\right)} \tag{2.2-12}$$

Comme pour les solveurs modaux, on constate que l'importance du conditionnement de l'opérateur est pondéré par le choix d'un bon point de départ : malgré un mauvais conditionnement, les cas (a) et (b) sont très différents ; Dans le premier, le point de départ engendre presque un espace propre de l'opérateur et on converge en deux itérations, sinon ce sont les «sempiternels zigzags».

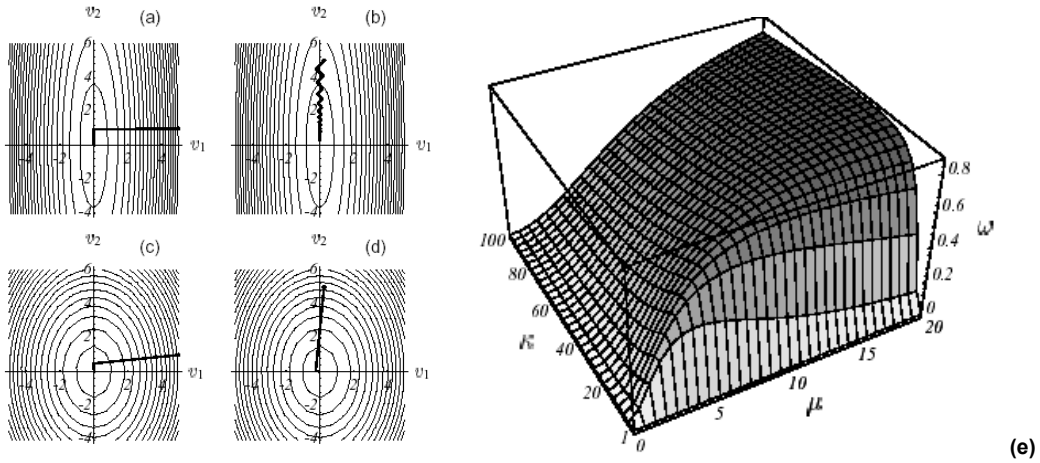


Figure 2.2-2. Convergence de la Steepest Descent sur l'exemple n°1 suivant les valeurs du conditionnement et du point de départ: κ grand et μ petit (a), κ et μ grand (b), κ et μ petit (c), κ petit et μ grand (d) et forme globale de $\omega = \omega(\kappa, \mu)$ (e).

Remarques :

- *Cette méthode de la plus forte pente a été initiée par Cauchy (1847) et remis au goût du jour par Curry (1944). Dans le cadre des opérateurs SPD, elle est aussi parfois appelée méthode du gradient à paramètre optimal (cf. [LT98] §8.2.1). Malgré ses faibles propriétés de convergence, elle a connu son «heure de gloire» pour minimiser des fonctions J quasi-quelconques, en fait seulement dérivables.*

17 C'est-à-dire $\lim_{i\to\infty} \frac{J(u^{i+1})-J(u)}{J(u^i)-J(u)} \leq \alpha := \left(\frac{\eta(K)-1}{\eta(K)+1}\right)^2 < 1$. Le taux de convergence asymptotique α est appelé rapport de Kantorovitch

Dans ce cas de figure plus général, elle ne permet alors d'atteindre que des points stationnaires, au mieux des minima locaux.

- *Pour éviter les effets zig-zag divers procédés d'accélération ont été proposés (cf. Forsythe 1968, Luenberger 1973 ou [Min08] §2.3) qui ont une vitesse de convergence similaire à celle du gradient conjugué mais pour une complexité calcul bien supérieure. Ils sont donc tombés progressivement en désuétude.*
- *Notons que des variantes de l'algorithme 1 ont été introduites pour traiter des cas non SPD : Itération du résidu minimum et Steepest Descent avec norme du résidu (cf. [Saa03]).*
- *On retrouve au travers de la Steepest Descent un concept clé très répandu en analyse numérique : celui de la résolution d'un problème, par projection d'itérés sur un sous-espace approximant, ici* $K_i := vect(r^i)$ *, perpendiculairement à un autre sous-espace, ici* $L_i := K_i$ *. On les appelle, respectivement, espace de recherche ou d'approximation et espace de contrainte. Pour la Steepest Descent, ils sont égaux et réduits à leur plus simple expression mais nous verrons que pour le gradient conjugué ils prennent la forme d'espaces particuliers, dit de Krylov.*

Formellement, à chaque itération i *de l'algorithme, on cherche ainsi un incrément* $\delta^i := \alpha^i r^i$ *tel que :*

$$\begin{cases} u^i := u^0 + \delta^i \quad \delta^i := \alpha^i r^i \in K_i \\ \langle r^0 - \boldsymbol{K}\,\delta^i, w \rangle = 0 \quad \forall\, w \in L_i = K_i \end{cases} \tag{2.2-13}$$

Ce cadre général constitue ce qu'on appelle les conditions de Petrov-Galerkin.

2.2.4 Complexité et occupation mémoire

La majeure partie du coût calcul de l'algorithme 1 réside dans la mise à jour (2.2-5) et, plus particulièrement, dans le produit matrice-vecteur $K\,r^i$. D'autre part, on a déjà mentionné que sa convergence était acquise et s'opérait en un nombre d'itérations proportionnel au conditionnement de la matrice (cf. (2.2-11)).
La complexité calcul de l'algorithme est donc, si on tient compte du caractère creux de l'opérateur, de l'ordre de $\Theta(\eta(K)\,cN)$ où c est le nombre moyen de termes non nuls par ligne de K.
Ainsi, avec des problèmes résultant de la discrétisation éléments finis d'opérateurs elliptiques du second ordre[18] (resp. du quatrième ordre), on a des conditionnements d'opérateurs en $\eta(K) = \Theta(N^{2/d})$ (resp. $\eta(K) = \Theta(N^{4/d})$) où d la dimension d'espace. La complexité calcul de la Steepest Descent dans ce cadre s'écrit ainsi $\Theta\left(cN^{\frac{2}{d}+1}\right)$ (resp. $\Theta\left(cN^{\frac{4}{d}+1}\right)$).
Pour ce qui est de l'occupation mémoire, seul le stockage de la matrice de travail est éventuellement requis[19] : $\Theta(cN)$. En pratique, la mise en place informatique du stockage creux impose la gestion de vecteurs d'entiers supplémentaires : par exemple, pour le stockage `MORSE` utilisé dans *Code_Aster*, vecteurs des indices de fin de ligne et des indices de colonnes des éléments non nuls du profil. D'où une complexité mémoire effective de $\Theta(cN)$ réels et $\Theta(cN+N)$ entiers.

2.3 Méthode de descente «générale»

2.3.1 Principe

18 Cas le plus souvent rencontré en mécanique des structures.
19 Dans l'absolu, la Steepest Descent comme le GC ne requiert que la connaissance de l'action de la matrice sur un vecteur quelconque et non pas son stockage *in extenso*. Cette facilité peut s'avérer précieuse pour des applications très gourmandes en degrés de liberté (CFD).

Une étape cruciale de ces méthodes est le choix de leur direction de descente[20]. Même si le gradient de la fonctionnelle en reste l'ingrédient principal, on peut tout à fait en choisir une autre version d^i que celle requise pour la Steepest Descent. A la $i^{ème}$ itération, on va donc chercher à construire u^{i+1} vérifiant

$$u^{i+1} := u^i + \alpha^i d^i \tag{2.3-1}$$

avec

$$\alpha^i = \underset{\alpha \in [\alpha_m, \alpha_M]}{Arg\min} J(u^i + \alpha\, d^i) \tag{2.3-2}$$

Ce n'est bien sûr qu'une généralisation de la 'Steepest Descent' vue précédemment et on montre que ses choix du paramètre de descente et sa propriété d'orthogonalité se généralisent

$$\alpha^i := \frac{\langle r^i, d^i \rangle}{\langle d^i, \mathbf{K}\mathbf{d^i} \rangle} \qquad \langle d^i, r^{i+1} \rangle = 0 \tag{2.3-3}$$

D'où le même effet «zigzag» lors du déroulement du processus et une convergence similaire à (2.2-6) avec un facteur d'atténuation de l'erreur minoré par :

$$\omega^i := 1 - \frac{\langle r^i, d^i \rangle^2}{\langle K^{-1} r^i, r^i \rangle \langle \mathbf{K}\mathbf{d^i}, d^i \rangle} \geq \frac{1}{\eta(K)} \left\langle \frac{r^i}{\|r^i\|}, \frac{d^i}{\|d^i\|} \right\rangle^2 \tag{2.3-4}$$

De ce résultat on peut alors formuler deux constats :

- Le conditionnement de l'opérateur intervient directement sur le facteur d'atténuation et donc sur la vitesse de convergence,
- Pour s'assurer de la convergence (condition suffisante) il faut, lors d'une itération donnée, que la direction de descente ne soit pas orthogonale au résidu.

La condition suffisante de ce dernier item est bien sûr vérifiée pour la Steepest Descent ($d^i = r^i$) et elle imposera un choix de direction de descente pour le gradient conjugué. Pour pallier au problème soulevé par le premier point, nous verrons qu'il est possible de constituer un opérateur de travail $\tilde{K}$ dont le conditionnement est moindre. On parle alors de préconditionnement.
Toujours dans le cas d'un opérateur SPD, le déroulement d'une méthode de descente «générale» s'écrit donc

Initialisation u^0, d^0 donnés, $r^0 = f - \mathbf{K}\mathbf{u^0}$
Boucle en i
(1) $z^i = \mathbf{K}\mathbf{d^i}$
(2) $\alpha^i = \frac{\langle r^i, d^i \rangle}{\langle d^i, z^i \rangle}$ (paramètre optimal de descente)
(3) $u^{i+1} = u^i + \alpha^i d^i$ (nouvel itéré)
(4) $r^{i+1} = r^i - \alpha^i z^i$ (nouvel résidu)
(5) Test d'arrêt via $J^{i+1} - J^i$, $\|d^i\|$ ou $\|r^{i+1}\|$ (par exemple)
(6) Calcul de la dd $d^{i+1} = d^{i+1}(\nabla J^k, \nabla^2 J^k, d^k \ldots)$

Algorithme 2 : Méthode de descente dans le cas d'une fonctionnelle quadratique.

Cet algorithme préfigure déjà bien celui du Gradient Conjugué (GC) que nous examinerons au chapitre suivant (cf. algorithme 4). Il montre bien que le GC n'est qu'une méthode de descente appliquée dans le cadre de fonctionnelles quadratiques et de directions de descente spécifiques. Finalement, seule l'étape (6) s'en trouvera étoffée.

Remarques :

- *En posant successivement comme directions de descente les vecteurs canoniques des axes de coordonnées de l'espace à N dimensions ($d^i = e_i$), on obtient la méthode de Gauss-Seidel.*

20 Par définition une direction de descente (dd) à la $i^{ème}$ étape, notée d^i, vérifie : $(\nabla J^i)^T d^i < 0$.

- *Pour éviter le surcoût calcul de l'étape de minimisation unidimensionnelle (2) (produit matrice-vecteur) on peut choisir de fixer le paramètre de descente arbitrairement : c'est la méthode de Richardson qui converge au mieux comme la Steepest Descent.*

2.3.2 Compléments

Avec une fonctionnelle J continue quelconque (cf. figure 2.3-1), on dépasse le cadre strict de l'inversion de système linéaire pour celui de l'optimisation continue non linéaire sans contrainte (J est alors souvent appelée fonction coût ou fonction objectif). Deux simplifications qui avaient court jusqu'ici deviennent alors illicites :

- La réactualisation du résidu : étape (4),
- Le calcul simplifié du paramètre de descente optimal : étape (2).

Leurs raisons d'être étant uniquement d'utiliser toutes les hypothèses du problème pour faciliter la minimisation unidimensionnelle (2.3-2), on est alors contraint d'effectuer explicitement cette recherche linéaire sur une fonctionnelle plus chahutée avec cette fois de multiples minima locaux. Heureusement, il existe toute une panoplie de méthodes suivant le degré d'information requis sur la fonction coût J :

- J (interpolation quadratique, dichotomie sur les valeurs de la fonction, de la section dorée, règle de Goldstein et Price…),
- J , ∇J (dichotomie sur les valeurs de la dérivée, règle de Wolfe, d'Armijo…),
- (J , ∇J , $\nabla^2 J$ Newton-Raphson…)
- …

Pour ce qui est de la direction de descente, là aussi de nombreuses solutions sont proposées dans la littérature (gradient conjugué non linéaire, quasi-Newton, Newton, Levenberg-Marquardt[21]…). De longue date, les méthodes dites de gradient conjugué non linéaire (Fletcher-Reeves (FR) 1964 et Polak-Ribière (PR) 1971) se sont avérées intéressantes : elles convergent superlinéairement vers un minimum local pour un coût calcul et un encombrement mémoire réduits (cf. Figures 2.3-1).

Elles conduisent au choix d'un paramètre supplémentaire β^i qui gère la combinaison linéaire entre les directions de descente

$$d^{i+1} = -\nabla J^{i+1} + \beta^{i+1} d^i \quad \text{avec} \quad \beta^{i+1} := \begin{cases} \dfrac{\|\nabla J^{i+1}\|^2}{\|\nabla J^i\|^2} & (\text{FR}) \\ \dfrac{\langle \nabla J^{i+1}, \nabla J^{i+1} - \nabla J^i \rangle}{\|\nabla J^i\|^2} & (\text{PR}) \end{cases} \tag{2.3-5}$$

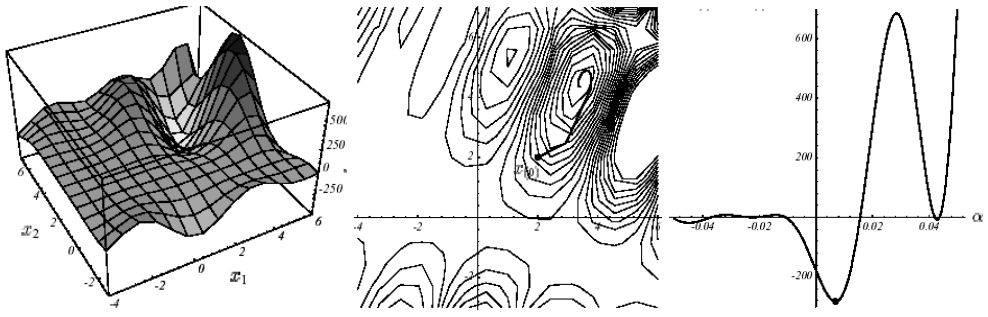


Figure 2.3-1. Exemple de J non convexe en $N=2$: graphe (a), convergence avec un GC non linéaire de type Fletcher-Reeves (b), plan de coupe de la première recherche unidimensionnelle (c).

21 Ces deux dernières méthodes sont utilisées par *Code_Aster* : la première dans les opérateurs non linéaires (STAT/DYNA/THER_NON_LINE), la seconde dans la macro de recalage (MACR_RECAL).

D'où l'algorithme, en nommant r^0 l'opposé du gradient et non plus le résidu (qui n'a plus lieu d'être ici),

Initialisation u^0 donné, $r^0=-\nabla J^0$, $d^0=r^0$
Boucle en i
(1) Trouver $\alpha^i=\underset{\alpha\in[\alpha_m,\alpha_M]}{Arg\min} J\left(u^i+\alpha\, d^i\right)$ (paramètre de descente)
(2) $u^{i+1}=u^i+\alpha^i d^i$ (nouvel itéré)
(3) $r^{i+1}=-\nabla J^{i+1}$ (nouveau gradient)
(4) Test d'arrêt via $J^{i+1}-J^i$, $\|d^i\|$ ou $\|r^{i+1}\|$ (par exemple)
(5) $\beta^{i+1}=\begin{cases}\dfrac{\|r^{i+1}\|^2}{\|r^i\|^2} & (\text{FR})\\ \dfrac{\langle r^{i+1}, r^{i+1}-r^i\rangle}{\|r^i\|^2} & (\text{PR})\end{cases}$ (paramètre de conjugaison)
(6) $d^{i+1}=r^{i+1}+\beta^{i+1}d^i$ (nouvelle dd)

Algorithme 3 : Méthodes du gradient conjugué non linéaire (FR et PR).

La désignation gradient conjugué non linéaire est plutôt ici synonyme de «non convexe» : il n'y a plus de dépendance entre le problème de minimisation (P_2) et un système linéaire (P_1). Au vue de l'algorithme 4, les grandes similitudes avec le GC paraissent dès lors tout à fait claires. Dans le cadre d'une fonction coût quadratique de type (2.1-2b), il suffit juste de substituer à l'étape (1) l'actualisation du résidu et le calcul du paramètre optimal de descente. Le GC n'est qu'une méthode de Fletcher-Reeves appliquée au cas d'une fonctionnelle quadratique convexe.

Maintenant que nous avons bien amorcé le lien entre les méthodes de descente, le gradient conjugué au sens solveur linéaire SPD et sa version, vue du bout de la lorgnette optimisation continue non linéaire, nous allons (enfin !) passer au vif du sujet et argumenter le choix d'une direction de descente du type (2.3-5) pour le GC standard. Pour plus d'informations sur les méthodes de type descente, le lecteur pourra se référer aux excellents ouvrages d'optimisation (en langue française) de M. Minoux[Min08], J.C. Culioli[Cu94] ou J.F.Bonnans et al[BGLS].

3 Le Gradient Conjugué (GC)

3.1 Description générale

3.1.1 Principe

Maintenant que les fondements sont mis en place nous allons pouvoir aborder l'algorithme du Gradient Conjugué (GC) proprement dit. Il appartient à un sous-ensemble de méthodes de descente qui regroupe les méthodes dites «de directions conjuguées»[22]. Celles-ci préconisent de construire progressivement des directions de descentes $d^0, d^1, d^2 \ldots$ linéairement indépendantes de manière à éviter les zigzags de la méthode de descente classique.

Quelle combinaison linéaire alors préconiser pour construire, à l'étape i, la nouvelle direction de descente ? Sachant bien sûr qu'elle doit tenir compte de deux informations cruciales : la valeur du gradient $\nabla J^i = -r^i$ et celles des précédentes directions $d^0, d^1 \ldots d^{i-1}$.

$$? \quad d^i = \alpha\, r^i + \sum_{j<i} \beta_j\, d^j \tag{3.1-1}$$

L'astuce consiste à choisir une indépendance vectorielle de type K-orthogonalité (comme l'opérateur de travail est SPD, il définit bien un produit scalaire *via* lequel deux vecteurs peuvent être orthogonaux, cf. figure 3.1-1)

$$\left(d^i\right)^T K\, d^j = 0 \quad i \neq j \tag{3.1-2}$$

appelée aussi conjugaison, d'où la désignation de l'algorithme. Elle permet de propager de proche en proche l'orthogonalité et donc de ne calculer qu'un coefficient de Gram-Schmidt à chaque itération. D'où un gain en complexité calcul et mémoire très appréciable.

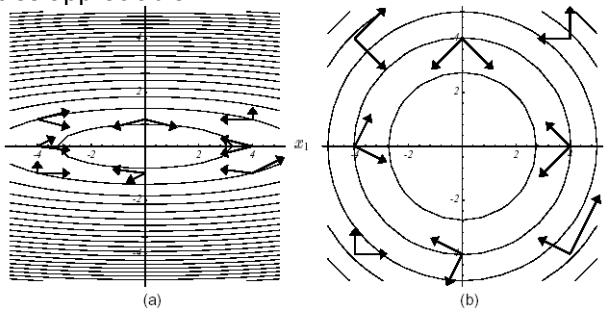


Figure 3.1-1. Exemple de paires de vecteurs K-orthogonaux en 2D: conditionnement de K quelconque (a), conditionnement parfait (i.e. égal à l'unité) = orthogonalité usuelle (b).

On peut donc se contenter d'une combinaison linéaire du type

$$d^i := r^i + \beta^i\, d^{i-1} \tag{3.1-3}$$

Et ce, d'autant plus qu'elle vérifie la condition suffisante (2.3-4) et que du fait de l'orthogonalité (2.3-3b), la recherche unidimensionnelle (2.3-2) qui suit s'opère dans un espace optimal : la plan formé par les deux directions orthogonales (r^i, d^{i-1}).

Il reste donc à déterminer la valeur optimale du coefficient de proportionnalité β^i. Dans le GC ce choix s'opère de manière à maximiser le facteur d'atténuation de (2.3-4), c'est-à-dire le terme

$$\frac{\langle \mathrm{r}^i, \mathrm{d}^i \rangle^2}{\langle \mathrm{K}^{-1} \mathrm{r}^i, \mathrm{r}^i \rangle \langle \mathrm{K} \mathrm{d}^i, \mathrm{d}^i \rangle} \tag{3.1-4}$$

Il conduit à l'expression

$$\beta^i := \frac{\|r^i\|^2}{\|r^{i-1}\|^2} \tag{3.1-5}$$

et induit la même propriété d'orthogonalité des résidus successifs que pour la Steepest Descent (mais sans les zigzags !)

$$\langle r^i, r^{i-1} \rangle = 0 \tag{3.1-6}$$

Se rajoute une condition «résidu-dd»

22 Les méthodes de Fletcher-Reeves et de Polak-Ribière (cf. §2.3) font aussi partie de cette famille de méthodes.

$$\langle r^i, d^i \rangle = \|r^i\|^2 \qquad (3.1\text{-}7)$$

qui impose d'initialiser le processus *via* $d^0 = r^0$.

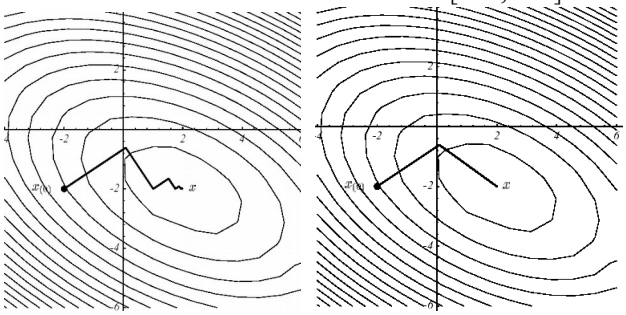
3.1.2 Algorithme

Bref, en récapitulant les relations (2.2-5), (2.3-1), (2.3-3) et (3.1-3), (3.1-5), (3.1-7) il advient l'algorithme classique

Initialisation u^0 donné, $r^0 = \mathbf{f\text{-}Ku^0}$, $d^0 = r^0$
Boucle en i
(1) $z^i = \mathbf{K d^i}$
(2) $\alpha^i = \dfrac{\|r^i\|^2}{\langle d^i, z^i \rangle}$ (paramètre optimal de descente)
(3) $u^{i+1} = u^i + \alpha^i d^i$ (nouvel itéré)
(4) $r^{i+1} = r^i - \alpha^i z^i$ (nouveau résidu)
(5) Test d'arrêt via $\langle r^{i+1}, r^{i+1} \rangle$ (par exemple)
(6) $\beta^{i+1} = \dfrac{\|r^{i+1}\|^2}{\|r^i\|^2}$ (paramètre de conjugaison optimal)
(7) $d^{i+1} = r^{i+1} + \beta^{i+1} d^i$ (nouvelle dd)

Algorithme 4 : Gradient conjugué standard (GC).

Sur l'exemple n°1, la «suprématie» du GC par rapport à la Steepest Descent est manifeste (cf. figure 3.1-2) et s'explique par l'écart entre le résultat de convergence (2.2-9) et celui de (3.2-9). Dans les deux cas, les mêmes points de départ et les mêmes critères d'arrêt ont été choisis : $u^0 = [-2, -2]^T$ et $\|r^i\|^2 < \varepsilon = 10^{-6}$.



a b

Figure 3.1-2. Convergences comparées, sur l'exemple n°1, de la Steepest Descent (a) et du GC (b).

Remarques :

- *Cette méthode du GC a été développée indépendamment par M.R.Hestenes (1951) et E.Stiefel (1952) du 'National Bureau of Standard' de Washington D.C. (pépinière de numériciens avec aussi C.Lanczos). Les premiers résultats théoriques de convergence sont dus aux travaux de S.Kaniel (1966) et de H.A.Van der Vorst (1986) et elle a vraiment été popularisée pour la résolution de gros systèmes creux par J.K.Reid (1971). Le lecteur intéressé trouvera une histoire commentée et une bibliographie exhaustive sur le sujet dans les papiers de G.H.Golub, H.A Van der Vorst et Y.Saad [Go89][GV97][SV00].*
- *Pour la petite histoire, fort de sa diffusion très large dans le monde industriel et universitaire, et, de ses nombreuses variantes, le GC a été classé en troisième position du « Top10 » des meilleurs algorithmes du XX e siècle [Ci00]. Juste derrière les méthodes de Monte-Carlo et du simplexe mais devant l'algorithme **QR** , les transformées de Fourier et les solveurs multipôles !*

- *On trouve traces dans les codes d'EDF R&D du GC dès le début des années 80 avec les premiers travaux sur le sujet de J.P.Grégoire. Depuis il s'est répandu, avec un bonheur inégal, dans de nombreux domaines, _ N3S, Code_Saturne, COCCINELLE, Code_Aster, TRIFOU, ESTET, TELEMAC _, et il a été beaucoup optimisé, vectorisé, parallélisé [Greg]…*
- *Au test d'arrêt sur la norme du résidu, théoriquement licite mais en pratique parfois difficile à calibrer, on préfère souvent un critère d'arrêt adimensionnel, tel que le résidu relatif à la $i^{ème}$ itération :* $\delta^i := \frac{\|r^i\|}{\|f\|}$ *(cf. §5).*

3.2 Eléments de théorie

3.2.1 Espace de Krylov

En reprenant l'analyse de Petrov-Galerkin déjà évoquée pour la Steepest Descent, on peut synthétiser en une phrase l'action du GC. Elle réalise des projections orthogonales successives sur l'espace de Krylov $\kappa_i(K, r^0) := vect(r^0, Kr^0 \dots K^{i-1} r^0)$ où r^0 est le résidu initial : à la $i^{ème}$ itération $K_i = L_i = \kappa_i(K, r^0)$. On résout le système linéaire (P_1) en recherchant une solution approchée u^i dans le sous-espace affine (espace de recherche de dimension N)

$$A = r^0 + \kappa_i(K, r^0) \tag{3.2-1}$$

tout en imposant la contrainte d'orthogonalité (espace des contraintes de dimension N)

$$r^i := f - \mathbf{K}\mathbf{u}^i \perp \kappa_i(K, r^0) \tag{3.2-2}$$

Cet espace de Krylov a la bonne propriété de faciliter l'approximation de la solution, au bout de m itérations, sous la forme

$$K^{-1} f \approx u^m = r^0 + P_{m-1}(K) f \tag{3.2-3}$$

où P_{m-1} est un certain polynôme matriciel d'ordre $m-1$. En effet, on montre que les résidus et les directions de descente engendrent cet espace

$$\begin{aligned} vect(r^0, r^1 \dots r^{m-1}) &= \kappa_m(K, r^0) \\ vect(d^0, d^1 \dots d^{m-1}) &= \kappa_m(K, r^0) \end{aligned} \tag{3.2-4}$$

tout en permettant à la solution approchée, u^m , de minimiser la norme en énergie sur tout l'espace affine A

$$\|u^m\|_K \le \|u\|_K \quad \forall u \in A \tag{3.2-5}$$

Ce résultat joint à la propriété (3.2-4b) illustre toute l'optimalité du GC : contrairement aux méthodes de descente, le minimum d'énergie n'est pas réalisé successivement pour chaque direction de descente, mais conjointement pour toutes les directions de descente déjà obtenues.

Remarque :

- *On distingue une grande variété de méthodes de projection sur des espaces de type Krylov, appelées plus prosaïquement «méthodes de Krylov». Pour résoudre des systèmes linéaires[Saa03] (GC, GMRES, FOM/IOM/DOM, GCR, ORTHODIR/MIN…) et/ou des problèmes modaux [Boi01] (Lanczos, Arnoldi…). Elles diffèrent par le choix de leur espace de contrainte et par celui du préconditionnement appliqué à l'opérateur initial pour constituer celui de travail, sachant que des implantations différentes conduisent à des algorithmes radicalement distincts (version vectorielle ou par blocs, outils d'orthonormalisation…).*

3.2.2 Orthogonalité

Comme on l'a déjà signalé, les directions de descentes sont K -orthogonales entre elles. De plus, le choix du paramètre de descente optimal (cf. (2.2-4), (2.3-3a) ou étape (2)) impose, de proche en proche, les orthogonalités

$$\begin{aligned}&\langle d^i, r^m\rangle = 0 \quad \forall i < m\\ &\langle r^i, r^m\rangle = 0\end{aligned} \tag{3.2-6}$$

On constate donc une petite approximation dans l'appellation du GC, car les gradients ne sont pas conjugués (cf. (3.2-6b)) et les directions conjugués ne comportent pas que des gradients (cf. (3.1-3)). Mais ne «chipotons» pas, les ingrédients désignés sont quand même là !

A l'issue de N itérations, deux cas de figures se présentent :

- Soit le résidu est nul $r^N = 0$ donc convergence.
- Soit il est orthogonal aux N précédentes directions de descente qui constituent une base de l'espace fini d'approximation $\Re^N$ (comme elles sont linéairement indépendantes). D'où nécessairement $r^N = 0$ donc convergence.

Il semblerait donc que le GC soit une méthode directe qui converge en au plus N itérations, c'est du moins ce qu'on a cru avant de le tester sur des cas pratiques ! Car ce qui reste vrai en théorie, en arithmétique exacte, est mis à mal par l'arithmétique finie des calculateurs. Progressivement, notamment à cause des erreurs d'arrondi, les directions de descente perdent leurs belles propriétés de conjugaison et la minimisation sort de l'espace requis.

Dit autrement, on résout un problème approché qui n'est plus tout à fait la projection souhaitée du problème initial. La méthode (théoriquement) directe a révélée sa vraie nature ! Elle est itérative et donc soumise, en pratique, à de nombreux aléas (conditionnement, point de départ, tests d'arrêt, précision de l'orthogonalité…).

Pour y remédier, on peut imposer lors de la construction de la nouvelle direction de descente (cf. algorithme 4, étape (7)), une phase de réorthogonalisation. Cette pratique très répandue en analyse modale [Boi01] et annexe 2) et en décomposition de domaine[Boi08] peut se décliner sous différentes variantes : réorthogonalisation totale, partielle, sélective … *via* toute une panoplie de procédures d'orthogonalisation (GS, GSM, IGSM, Househölder, Givens).

Ces réorthogonalisations requièrent, par exemple, le stockage des N_{orth} premiers vecteurs $d^k \left(k = 1 \ldots N_{orth}\right)$ et de leur produit par l'opérateur de travail $z^k = Kd^k \left(k = 1 \ldots N_{orth}\right)$. Formellement, il s'agit donc de substituer aux deux dernières étapes de l'algorithme le calcul suivant

$$d^{i+1} = r^{i+1} - \sum_{k=0}^{\max\left(i, N_{orth}\right)} \frac{\langle r^{i+1}, \mathbf{Kd^k}\rangle}{\langle d^k, \mathbf{Kd^k}\rangle} d^k \quad (\text{nouvelle dd } K-\text{orthogonalisée}) \tag{3.2-7}$$

Au travers des différentes expériences numériques qui ont été menées (cf. notamment travaux de J.P.Grégoire et [Boi03]), il semble que cette réorthogonalisation ne soit pas toujours efficace. Son surcoût dû principalement aux nouveaux produits matrice-vecteur $K\,d^k$ (et à leur stockage) n'est pas toujours compensé par le gain en nombre d'itérations globales.

3.2.3 Convergence

Du fait de la structure particulière de l'espace d'approximation (3.2-3) et de la propriété de minimisation sur cet espace de la solution approchée u^m (cf. (3.2-5)), on obtient une estimation de la vitesse de convergence du GC

$$\begin{aligned}&\|e\left(u^i\right)\|_K^2 = \left(\omega^i\right)^2 \|e\left(u^0\right)\|_K^2\\ &\text{avec } \omega^i := \max_{1 \le i \le N} \left(1 - \lambda_i P_{m-1}\left(\lambda_i\right)\right)\end{aligned} \tag{3.2-8}$$

où l'on note $(\lambda_i ; v_i)$ les modes propres de la matrice K et P_{m-1} un polynôme quelconque de degré au plus $m-1$. Les fameux polynômes de Tchebycheff, *via* leurs bonnes propriétés de majoration dans l'espace des polynômes, permettent d'améliorer la lisibilité de ce facteur d'atténuation ω^i. Au terme de i itérations, au pire, la décroissance s'exprime alors sous la forme

$$\|e(u^i)\|_K \leq 2\left(\frac{\sqrt{\eta(K)}-1}{\sqrt{\eta(K)}+1}\right)^i \|e(u^0)\|_K \quad (3.2\text{-}9)$$

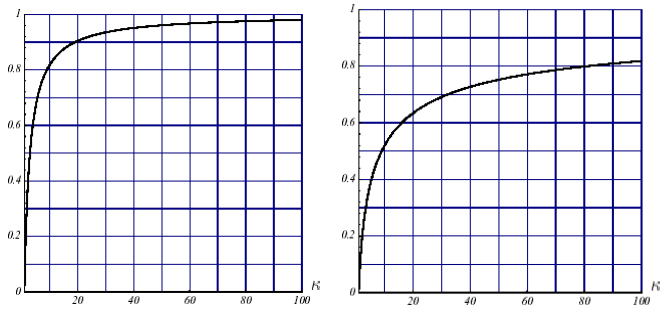
Elle assure la convergence superlinéaire (c'est-à-dire $\lim_{i\to\infty}\frac{J(u^{i+1})-J(u)}{J(u^i)-J(u)}=0$) du processus en un nombre d'itérations proportionnel à la racine carré du conditionnement de l'opérateur. Ainsi, pour obtenir

$$\frac{\|e(u^i)\|_K}{\|e(u^0)\|_K} \leq \varepsilon \ (\text{petit}) \quad (3.2\text{-}10)$$

il faut un nombre d'itérations de l'ordre de

$$i \approx \frac{\sqrt{\eta(K)}}{2}\ln\frac{2}{\varepsilon} \quad (3.2\text{-}11)$$

Evidemment comme nous l'avons maintes fois remarqué, un problème mal conditionné ralentira la convergence du processus. Mais ce ralentissement sera moins notable pour le GC que pour la Steepest Descent (cf. figures 3.2-1). Et de toute façon, la convergence globale du premier sera meilleure.



(a) (b)

Figure 3.2-1. Convergences comparées de la Steepest Descent (a) et du GC (b) (au facteur ½ près) en fonction du conditionnement κ .

Remarques :

- *En pratique, profitant de circonstances particulières, _ meilleur point de départ et/ou distribution spectrale avantageuse _, la convergence du GC peut être bien meilleure que ce que laisse espérer (3.2-9). Les méthodes de Krylov ayant tendance à débusquer prioritairement les valeurs propres extrêmes, le «conditionnement effectif» de l'opérateur de travail s'en trouve amélioré.*
- *Par contre, certaines itérations de la Steepest Descent peuvent procurer une meilleure décroissance du résidu que les mêmes itérations du GC. Ainsi, la première itération du GC est identique à celle de la Steepest-Descent et donc avec un taux de convergence égal.*

3.2.4 Complexité et occupation mémoire

Comme pour la Steepest Descent, la majeure partie du coût calcul de cet algorithme réside dans l'étape (1), le produit matrice-vecteur. Sa complexité est de l'ordre de $\Theta(kcN)$ où c est le nombre moyen de termes non nuls par ligne de K et k le nombre d'itérations requises à convergence. Pour être beaucoup plus efficace qu'un simple Cholesky (de complexité $\Theta\left(\frac{N^3}{3}\right)$) il faut donc :

- Bien prendre en compte le caractère creux des matrices issues des discrétisations éléments finis (stockage MORSE, produit matrice-vecteur optimisé *ad hoc*) : $c \ll N$.
- Préconditionner l'opérateur de travail : $k \ll N$.

On a déjà fait remarquer que pour un opérateur SPD sa convergence théorique se produit en, au plus, N itérations et proportionnellement à la racine du conditionnement (cf. (3.2-11)). En pratique, pour de gros systèmes mal conditionnés (comme c'est souvent le cas en mécanique des structures), elle peut être très lente à se manifester (cf. phénomène de Lanczos du paragraphe suivant).

Compte-tenu des conditionnements d'opérateurs généralement constatés en mécanique des structures et rappelés pour l'étude de complexité de la Steepest Descent, dont on reprend les notations, la complexité calcul du GC s'écrit $\Theta\left(cN^{\frac{1}{d}+1} \right)$ (resp. $\Theta\left(cN^{\frac{2}{d}+1} \right)$).

Pour ce qui est de l'occupation mémoire, comme pour la Steepest Descent, seul le stockage de la matrice de travail est éventuellement requis : $\Theta(cN)$. En pratique, la mise en place informatique du stockage creux impose la gestion de vecteurs d'entiers supplémentaires : par exemple pour le stockage MORSE utilisé dans *Code_Aster*, vecteurs des indices de fin de ligne et des indices de colonnes des éléments du profil. D'où une complexité mémoire effective de $\Theta(cN)$ réels et $\Theta(cN+N)$ entiers.

Remarque:

• *Ces considérations sur l'encombrement mémoire ne prennent pas en compte les problèmes de stockage d'un éventuel préconditionneur et de l'espace de travail que sa construction peut provisoirement mobiliser.*

3.3 Compléments

3.3.1 Equivalence avec la méthode de Lanczos

En «triturant» la propriété d'orthogonalité du résidu avec tout vecteur de l'espace de Krylov (cf. (3.2-2)), on montre que les m itérations du GC conduisent à la construction d'une factorisée de Lanczos du même ordre (cf. [Boi01] §5.2)

$$\mathrm{KQ}^m = \mathrm{Q}^m \mathrm{T}^m - \underbrace{\frac{\sqrt{\beta^m}}{\alpha^{m-1}} \mathrm{q}^m}_{\mathrm{R}^m} \mathrm{e}_m^{\mathrm{T}} \qquad (3.3\text{-}1)$$

en notant :

- R^m le « résidu » spécifique de cette factorisation,
- e_m le $m^{\text{ième}}$ vecteur de la base canonique,
- $q^i := \frac{r^i}{\|r^i\|}$ les vecteurs résidus normalisés à l'unité, appelés pour l'occasion vecteurs de Lanczos,
- $Q^m := \left[\frac{r^0}{\|r^0\|} \quad \dots \quad \frac{r^{m-1}}{\|r^{m-1}\|} \right]$ la matrice orthogonale qu'ils constituent.

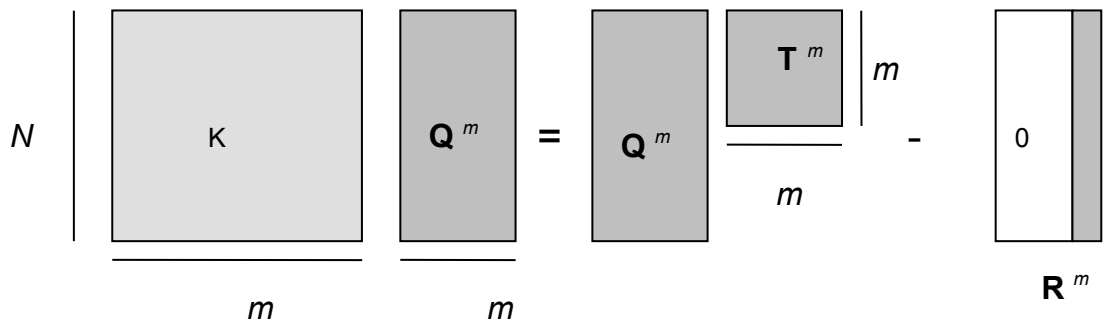


Figure 3.3-1. Factorisation de Lanczos induite par le GC.

La matrice de Rayleigh qui exprime la projection orthogonale de K sur l'espace de Krylov $\kappa_m\left(K, r^0\right)$ prend alors la forme canonique

$$T^m = \begin{bmatrix} \frac{1}{\alpha^0} & -\frac{\sqrt{\beta^1}}{\alpha^0} & 0 & 0 \\ -\frac{\sqrt{\beta^1}}{\alpha^0} & \frac{1}{\alpha^1}+\frac{\beta^1}{\alpha^0} & \cdots & 0 \\ 0 & \cdots & \cdots & -\frac{\sqrt{\beta^{m-1}}}{\alpha^{m-2}} \\ 0 & 0 & -\frac{\sqrt{\beta^{m-1}}}{\alpha^{m-2}} & \frac{1}{\alpha^{m-1}}+\frac{\beta^{m-1}}{\alpha^{m-2}} \end{bmatrix} \quad (3.3\text{-}2)$$

Presque sans calcul supplémentaire, le GC fournit ainsi l'approximation de Rayleigh de l'opérateur de travail sous une forme sympathique, _ matrice carrée symétrique tridiagonale de taille modulable _, dont il va être aisé de déduire le spectre (*via*, par exemple, un robuste **QR**, cf. [Boi01] annexe 2)).

A l'issue d'une inversion de système linéaire conduite par un simple GC on peut donc, à moindre frais, connaître, en plus de la solution recherchée, une approximation du spectre de la matrice et donc de son conditionnement[Greg]. Cette fonctionnalité pourrait être insérée dans *Code_Aster* de manière à mieux piloter le niveau de préconditionnement.

Remarques:

• *L'opérateur de travail étant SPD, sa matrice de Rayleigh hérite de cette propriété et on obtient directement sa décomposition* $T^m = L^m D^m \left(L^m\right)^T$ *avec*

$$L^m := \begin{bmatrix} 1 & 0 & 0 & 0 \\ -\sqrt{\beta^1} & 1 & \cdots & 0 \\ 0 & \cdots & \cdots & 0 \\ 0 & 0 & -\sqrt{\beta^{m-1}} & 1 \end{bmatrix} \text{ et } D^m := \begin{bmatrix} \frac{1}{\alpha^0} & 0 & 0 & 0 \\ 0 & \frac{1}{\alpha^1} & \cdots & 0 \\ 0 & \cdots & \cdots & 0 \\ 0 & 0 & 0 & \frac{1}{\alpha^{m-1}} \end{bmatrix} \quad (3.3\text{-}3)$$

La méthode de Lanczos ne calculant pas ces termes intermédiaires α^i *et* β^i*, mais directement les termes tridiagonaux, elle n'a pas accès à cette information. Par contre, elle peut prendre en compte des matrices symétriques non nécessairement définies positives.*

• *Donc, qu'il s'agisse d'inverser un système linéaire ou de déterminer une partie de son spectre, ces méthodes de Krylov fonctionnent sur le même principe : déterminer progressivement les vecteurs de bases engendrant l'espace de projection en même temps que le résultat de cette projection. Dans le GC comme dans Lanczos, on s'évertue à rendre cette projection la plus robuste (orthogonalité des vecteurs de base), la plus réduite (taille de projection = nombre d'itérations) et la plus simple possible (projection orthogonale). Pour le solveur modal, une autre contrainte se juxtapose : approximer le mieux possible le spectre de l'opérateur de travail par celui de la matrice de Rayleigh. En non symétrique, on parle de projection non plus orthogonale mais oblique et les rapprochements s'effectue alors entre GMRES et Arnoldi.*

• *En arithmétique exacte, au bout de* N *itérations, on a complètement déterminé tout le spectre de l'opérateur. En pratique, les problèmes d'arrondis et d'orthogonalisation nous en empêchent. Mais, si on*

est suffisamment patient ($m \gg N$), on est quand même capable de détecter tout le spectre : c'est ce qu'on appelle le phénomène de Lanczos (cf. [Boi01] §5.3.1/5.3.3).

- *Paradoxalement la perte d'orthogonalité est surtout imputable à la convergence d'un mode propre plus qu'aux erreurs d'arrondis ! Cette analyse due à CC.Paige (1980) atteste que dès qu'un mode propre est capturé il perturbe l'agencement orthogonal des vecteurs de Lanczos (dans le cas qui nous préoccupe, les résidus et les directions de descente). En modal, le traitement numérique de ce phénomène parasite a fait l'objet de nombreux développements palliatifs. Pour ce qui est du GC, il doit brider l'efficacité des méthodes de réorthogonalisation des directions de descente évoquées précédemment.*

3.3.2 Solveurs emboîtés

Comme on l'a déjà fait remarquer, les solveurs linéaires sont souvent enfouis au plus profond d'autres algorithmes, et en particulier, des solveurs non linéaires de type Newton. C'est pour le *Code_Aster* les cas d'application les plus fréquemment usités : opérateurs STAT_NON_LINE, DYNA_NON_LINE et THER_NON_LINE.
Dans une telle configuration, un solveur linéaire tel que le CG peut tirer son épingle du jeu (par rapport à un solveur direct). Son caractère itératif s'avère utile pour faire évoluer le test d'arrêt de l'algorithme (cf. étape (5) algorithme 4) en fonction de l'erreur relative du solveur non linéaire qui l'encapsule : c'est la problématique solveurs emboîtés (on parle aussi de relaxation) [CM01][GY97][BFG] dont le CERFACS est le «fer de lance» en France.

En relâchant au moment opportun la précision requise sur l'annulation du résidu (par exemple stratégie dans le cas méthode modale de type Krylov + GC) ou, au contraire en la durcissant (resp. méthode modale de type puissance + Gauss-Seidel + GC), on peut ainsi gagner en complexité calcul sur le GC sans modifier la convergence du processus global. Il faut bien sûr mettre au point des critères simples et peu coûteux pour que le gain soit substantiel. Cette fonctionnalité pourrait être insérée dans *Code_Aster* (pilotage interne et automatique de la valeur RESI_RELA).

Remarque :

- Certains auteurs se sont aussi posés la question de l'ordre d'enchaînement de ces algorithmes : «Newton- GC» ou «GC –Newton» ? Si d'un point de vue informatique et conceptuel, la question est vite tranchée : on préfère la première solution, plus lisible, plus souple et qui ne perturbe pas l'existant, d'un point de vue numérique et algorithmique, le partage est plus nuancé. Cela peut dépendre des «tripailles techniques» déployées par le solveur non-linéaire (matrice tangente, minimisation unidimensionnelle…) et du GC déployée (préconditionneur). Néanmoins, il semble que l'ordre naturel, «Newton- GC», soit le plus efficace et le plus évolutif.

3.3.3 Parallélisme

Le GC comme beaucoup de solveurs itératifs se prête bien au parallélisme. Il est assez scalable[23]. Les principales opérations élémentaires qui le constitue (produit matrice-vecteur, opération type[24] «daxpy et produit scalaire) se décomposent efficacement entre différents processeurs, seule la parallélisation du préconditionneur (souvent basé sur un solveur direct) peut s'avérer difficile (problèmes souvent dénommés : 'parallel preconditioning'). D'ailleurs les librairies d'algèbres linéaires (PETSc, HYPRE, TRILINOS…) ne proposent pas toujours une version parallèle de leurs préconditionneurs. Concernant les préconditionneurs, on a donc deux critères qui ont tendance à se contredire : l'efficacité parallèle et celle algorithmique.

23 Ou extensibilité en français. Une méthode est dite scalable d'un point de vue fort (c'est le critère par défaut, 'strong scalability'), si elle permet de résoudre d'autant plus vite un problème que l'on rajoute proportionnellement des processeurs. Moins communément, on parle de scalabilité faible ('weak scalability'), lorsqu'il s'agit de résoudre, en un temps comparable, un problème plus gros en augmentant d'autant le nombre de processeurs (cf. [Boi08] §2.4).

24 Pour reprendre une terminologie propre à la bibliothèque BLAS. Une opération de type DAXPY fait des combinaisons linéaires de vecteurs.

Les codes ont donc tendance à privilégier des préconditionneurs peu efficaces d'un point de vue algorithmique[25] (type Jacobi) mais peu gourmands en mémoire et très parallélisables. Le gain parallèle (appelé speed-up) est tout de suite très intéressant, mais le point de comparaison séquentiel peut ne pas être très glorieux[26] !

Une autre difficulté du parallélisme provient aussi de la distribution des données. Il faut veiller à ce que les différentes règles qui régissent la mise en données du calcul (attribution des zones de conditions limites, affectation des matériaux, choix des éléments finis…) ne soit pas affectées par le fait que chaque processeur ne les connaissant que partiellement. C'est ce qu'on appelle la «consistance» des données distribuées[Boi08b].

De même, il faut prévoir dans le code les communications *ad hoc* lors de chaque manipulation globale de données (critères d'arrêt, post-traitements…).

Remarques :

- *Ainsi sur machine CRAY, J.P.Grégoire[Greg] a adapté et parallélisé, en mémoire partagée et en mémoire distribuée, un algorithme de type gradient conjugué préconditionné. Quatre types d'opérations sont réalisés concurremment : les opérations vectorielles élémentaires, les produits scalaires, les produits matrice-vecteur et la construction du préconditionneur (diagonal). Des problèmes de 'parallel preconditioning' qui n'ont été résolus que pour le préconditionneur diagonal, n'ont malheureusement pas permis le portage de ces maquettes dans la version officielle de Code_Aster.*
- *Les solveurs directs et les autres solveurs itératifs (méthodes stationnaires Gauss-Seidel, SSOR..) sont réputées moins scalables que les méthodes de Krylov (GC, GMRES…).*

Nous allons d'ailleurs maintenant aborder l'épineuse question du préconditionnement pour le GC. Il deviendra alors le Gradient Conjugué PréConditionné : GCPC.

25 Allant jusqu'à tolérer des convergence en milliers d'itérations.

26 En toute rigueur, pour exprimer un gain parallèle, on devrait prendre comme référence le meilleur calcul séquentiel. Ce dernier faisant appel à un préconditionneur plus efficace et éventuellement séquentiel. L'exercice soulève toutefois une difficulté : ce point de fonctionnement est souvent dur à exhiber. La taille des études, compte-tenu des caractéristiques hardware des machines, nécessitent d'emblée un calcul distribué sur des dizaines de processeurs !

4 Le Gradient Conjugué PréConditionné (GCPC)

4.1 Description générale

4.1.1 Principe

Comme on a pu le constater (et le marteler !) dans les paragraphes précédents, la rapidité de convergence des méthodes de descente, et notamment celle du gradient conjugué, dépend du conditionnement de la matrice $\eta(K)$. Plus il est proche de sa valeur plancher, 1, meilleure est la convergence.

Le principe de préconditionnement est alors «simple», il consiste à remplacer le système linéaire du problème (P_1) par un système équivalent du type (préconditionnement à gauche) :

$$\left(\tilde{P}_1\right) \quad \underbrace{\mathbf{M^{-1}K}}_{\tilde{\mathbf{K}}}\,\mathbf{u}=\underbrace{\mathbf{M^{-1}f}}_{\tilde{\mathbf{f}}} \tag{4.1-1}$$

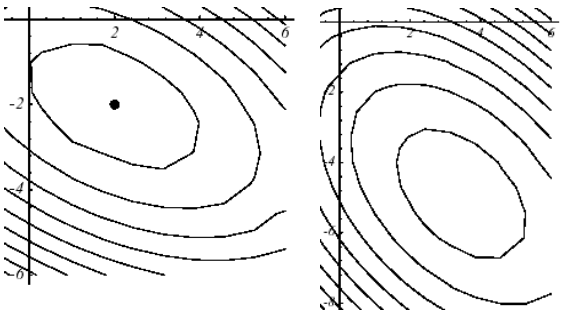
tel que, idéalement :

- Le conditionnement en soit évidemment amélioré (cette propriété théorique, tout comme la suivante, ne sont que très rarement démontrées. Elles ne sont souvent cautionnées que par des expériences numériques) : $\eta(\tilde{K}) \ll \eta(K)$.
- Tout comme la distribution spectrale : valeurs propres plus tassées.
- M^{-1} soit peu coûteux à évaluer (comme pour l'opérateur initial, on a souvent juste besoin de connaître l'action du préconditionneur sur un vecteur) : $\mathbf{Mv}=u$ facile à inverser.
- Voire facile à implanter et, éventuellement, efficace à paralléliser.
- Qu'il soit aussi creux que la matrice initiale car il s'agit de limiter l'encombrement mémoire supplémentaire.
- Qu'il conserve à la matrice de travail $\tilde{K}$ les mêmes propriétés que celle initiale : ici, le caractère SPD.

En théorie, le meilleur choix serait $M^{-1}=K^{-1}$ car alors $\eta\left(\tilde{K}=I_N\right)=1$, mais si il faut inverser complètement l'opérateur par une méthode directe pour construire ce préconditionneur, il n'est que de peu d'intérêt pratique ! Quoique, on verra par la suite que cette idée n'est pas aussi farfelue que cela.

Dit autrement, l'objectif d'un préconditionneur est de tasser le spectre de l'opérateur de travail ainsi, comme on l'a déjà mentionné, son « conditionnement effectif » sera amélioré de paire avec la convergence du GCPC.

Graphiquement, cela se traduit par une forme plus sphérique du graphe de la forme quadratique. Même sur un système à $N=2$ dimensions et avec un préconditionneur «fruste» (cf. figure 4.1-1), les effets sont notables.



(a) (b)

Figure 4.1-1. Effet du préconditionnement diagonal sur la paraboloïde de l'exemple n°1 : (a) sans, $\eta(K)=3.5$, (b) avec, $\eta(\tilde{K})=2.8$.

Dans l'absolu, on peut préconditionner un système linéaire par la gauche ('left preconditioning'), par la droite (resp. 'right') ou en faisant un mélange des deux (resp. 'split'). C'est cette dernière version qui va être retenue

pour notre opérateur SPD, car on ne peut pas directement appliquer le GC pour résoudre $\left(\tilde{P}_1\right)$: même si M^{-1} et K sont SPD, ce n'est pas forcément le cas de leur produit.

L'astuce consiste alors à utiliser une matrice de préconditionnement SPD, M, pour laquelle on va donc pouvoir définir une autre matrice (M étant symétrique réelle, elle est diagonalisable sous la forme $M=\mathrm{UDU}^T$ avec $D:=\mathrm{diag}\left(\lambda_i\right)\ \lambda_i>0$ et U matrice orthogonale). La matrice SPD recherchée provient alors de la décomposition associée $M^{1/2}=\mathrm{Udiag}\left(\sqrt{\lambda_i}\right)U^T$ avec $M^{1/2}$ définie telle que $(M^{1/2})^2=M$. D'où le nouveau problème de travail, cette fois SPD

$$\left(\hat{P}_1\right)\quad \underbrace{\mathrm{M}^{-1/2}\mathrm{KM}^{-1/2}}_{\hat{\mathrm{K}}}\underbrace{\mathrm{M}^{1/2}}_{\hat{\mathrm{u}}}\mathrm{u}=\underbrace{\mathrm{M}^{-1/2}}_{\hat{\mathrm{f}}}\mathrm{f} \tag{4.1-2}$$

sur lequel on va pouvoir appliquer l'algorithme standard du GC pour constituer ce qu'on appelle un Gradient Conjugue PréConditionné (GCPC).

4.1.2 Algorithme

Bref, en substituant dans l'algorithme 4, les expressions du problème précédent $\left(\hat{P}_1\right)$ et en travaillant un peu à la simplification du tout pour ne manipuler que des expressions en K, u et f, il advient le déroulement suivant.

Initialisation u^0 donné, $r^0=\mathbf{f-Ku^0}$, $d^0=M^{-1}r^0$, $g^0=d^0$
Boucle en i
(1) $z^i=\mathbf{Kd}^i$
(2) $\alpha^i=\dfrac{\langle r^i,g^i\rangle}{\langle d^i,z^i\rangle}$ (paramètre optimal de descente)
(3) $u^{i+1}=u^i+\alpha^i d^i$ (nouvel itéré)
(4) $r^{i+1}=r^i-\alpha^i z^i$ (nouveau résidu)
(5) Test d'arrêt via $\langle r^{i+1},r^{i+1}\rangle$ (par exemple)
(6) $g^{i+1}=M^{-1}r^{i+1}$ ($r$ésidu préconditionné)
(7) $\beta^{i+1}=\dfrac{\langle r^{i+1},g^{i+1}\rangle}{\langle r^i,g^i\rangle}$ (paramètre de conjugaison optimal)
(8) $d^{i+1}=g^{i+1}+\beta^{i+1}\ d^i$ (nouvelle dd)

Algorithme 5 : Gradient conjugué préconditionné (GCPC).

Mais en fait, le caractère symétrique du problème préconditionné initial $\left(\widetilde{P}_1\right)$ est tout relatif. Il est indissociable du produit scalaire sous-jacent. Si au lieu de prendre le produit scalaire euclidien usuel, on utilise un produit scalaire matriciel défini par rapport à K, M ou M^{-1}, il est possible de symétriser le problème préconditionné qui ne l'était pas initialement. Comme pour les méthodes de Krylov en modal, c'est le couple (opérateur de travail, produit scalaire) qu'il faut moduler pour s'adapter au problème !

Ainsi, $M^{-1}K$ étant symétrique par rapport au M-produit scalaire, on peut substituer ce nouvel opérateur de travail et ce nouveau produit scalaire dans l'algorithme du GC non préconditionné (algorithme 4)

$$K \Leftarrow M^{-1} K$$
$$\langle , \rangle \Leftarrow \langle , \rangle_M$$

Et (Ô surprise !) en travaillant un peu les expressions, on retrouve exactement l'algorithme du GCPC précédent (algorithme 5). On procède de même avec un préconditionnement à droite, $K M^{-1}$, via un M^{-1}-produit scalaire. Donc, préconditionnement à droite, à gauche ou «splitté à la mode SPD», conduisent tous rigoureusement au même algorithme. Cette constatation va nous être utile par la suite lorsqu'on s'apercevra que les matrices fournies par *Code_Aster* (et aussi les préconditionneurs qu'on leurs associera) ne sont pas toujours conformes au scénario idéal $\left(\hat{P}_1\right)$.

Remarques :

- *Cette variante du GCPC, qui est de loin la plus répandue, est parfois appelée dans la littérature : gradient conjugué préconditionné non transformé ('untransformed preconditioned conjugate gradient'). Par opposition à la version transformée qui manipule les entités propres de la nouvelle formulation.*
- *Les méthodes de descente générales et, particulièrement les GC non linéaires, se préconditionnent aussi (cf. [§2.3]). Cette fois avec un préconditionneur approximant l'inverse du Hessien au point considéré, de manière à rendre sphérique le graphe de la fonctionnelle au voisinage de ce point.*

4.1.3 «Survol» des principaux préconditionneurs

La solution souvent retenue pour sa simplicité de mise en œuvre, son rapport «efficacité numérique/surcoût calcul» pour des problèmes pas trop mal conditionnés, consiste à préconditionner par la diagonale de l'opérateur initial

$$M_J := \mathbf{diag}\left(K_{ii}\right) \tag{4.1-3}$$

C'est ce qu'on appelle le préconditionnement diagonal ou de Jacobi (JCG pour 'Jacobi Conjugate Gradient') par référence à la méthode stationnaire du même nom.

Remarques :

- *C'est une solution souvent retenue en CFD (pour EDF R&D : Code_Saturne, TELEMAC). Dans ces domaines, une grande attention portée au schéma de résolution non linéaire et à la construction du maillage produisent un problème souvent mieux conditionné. C'est ce qui a fait la renommée du JCG, transformé pour l'occasion en véritable «bête de course» dédiée aux gros calculateurs parallèles.*
- *C'est une solution proposée, en mécanique des structures, par bon nombre de codes commerciaux : ANSYS, Zébulon, NASTRAN. Elle a été présente dans Code_Aster mais son manque de fiabilité a conduit à sa résorption.*
- *Ce principe de préconditionneur «du pauvre» s'étend aux méthodes de descente en prenant cette fois la diagonale du Hessien.*

Un autre préconditionneur très répandu est le SSOR (pour Symetric Successive Over Relaxation). Comme le précédent, il est déduit d'une méthode itérative classique : la méthode de Gauss-Seidel relaxée. En décomposant l'opérateur initial sous la forme habituelle $K := D + L + L^T$ où D est sa diagonale et L sa partie triangulaire strictement inférieure, il s'écrit (en fonction du paramètre de relaxation ω)

$$M_{SSOR}(\omega) := \frac{1}{\omega(\omega-2)}(D+\omega L)D^{-1}(D+\omega L^T) \quad 0<\omega<2 \tag{4.1-4}$$

Il présente l'avantage de ne pas requérir de stockage mémoire et de surcoût calcul, puisqu'il est directement élaboré à partir de K, tout en étant très simple à inverser (une descente-remontée). Sur des problèmes modèles (le fameux «Laplacien sur le carré unité») des résultats théoriques ont été exhumés en éléments finis

$$\eta(K) := \Theta\left(\frac{1}{h^2}\right) \Rightarrow \eta\left(M^{-1}K\right) := \Theta\left(\frac{1}{h}\right) \tag{4.1-5}$$

D'autre part, il a la faculté, pour un opérateur SPD, de contingenter son spectre dans la bande $]0,1]$. Cependant, il s'est avéré dans la pratique industrielle, moins efficace que les préconditionneurs de type Cholesky incomplet que nous allons aborder dans le paragraphe suivant. Et il pose le problème délicat du choix du paramètre de relaxation optimal, par nature très «problème-dépendant».

Remarques :
- *Ce préconditionneur a été proposé par D.J.Evans (1967) et étudié par O.Axelsson (1974). Il se décline en moult versions : non symétrique, par blocs, avec paramètre de relaxation optimal…*
- *En posant* $\omega=1$ *on retrouve le cas particulier de Gauss-Seidel Symétrique*

$$M_{SGS}(\omega):=-(D+L)D^{-1}(D+L^T) \tag{4.1-6}$$

Une kyrielle d'autres préconditionneurs a ainsi vu le jour depuis une trentaine d'années dans la littérature : explicites (polynomiaux[BH89], SPAI, AINV…), implicites (Schwarz, IC…), multiniveaux (décomposition de domaine, multigrilles…) …. Certains sont spécifiques d'une application, d'autres plus généraux. Les «effets de modes» ont aussi fait leur oeuvre ! Pour plus d'informations on pourra consulter la somme monumentale commise par G.Meurant[Meu99] ou les livres de Y.Saad[Saa03] et H.A.Van der Vorst[Van01].

4.2 Factorisation incomplète de Cholesky

4.2.1 Principe

On vient de voir que les préconditionneurs s'inspire souvent de solveurs linéaires à part entière : Jacobi pour celui diagonal, Gauss-Seidel pour SSOR. Celui basé sur une factorisation Incomplète de Cholesky (IC) n'échappe pas à la règle ! Mais il s'appuie cette fois, non pas sur une autre méthode itérative, mais sur une méthode directe de type Cholesky. D'où la dénomination de ICCG ('Incomplete Cholesky Conjugate Gradient') donnée au couplage de ce préconditionneur avec le GCPC.

L'opérateur initial étant SPD, il admet une décomposition de Cholesky du type $K=\boldsymbol{C}\boldsymbol{C}^{\boldsymbol{T}}$ où $\boldsymbol{C}$ est une matrice triangulaire inférieure. On appelle factorisation incomplète de Cholesky, la recherche d'une matrice F triangulaire inférieure aussi creuse que possible et telle que FF^T soit proche de K dans un sens à définir. Par exemple, en posant $B=K-FF^T$ on va demander que l'erreur relative (exprimée dans une norme matricielle au choix)

$$\Delta:=\frac{\|B\|}{\|K\|} \tag{4.2-1}$$

soit le plus petite possible. A la lecture de cette définition «assez évasive» on entrevoit la profusion de scénarios possibles. Chacun y est allé de sa propre factorisation incomplète ! L'ouvrage de G.Meurant[Meu99] en montre la grande diversité : $IC(n)$, $MIC(n)$, relaxée, réordonnée, par blocs....

Toutefois, pour se simplifier la tâche, on impose souvent *a priori* la structure creuse de F, c'est-à-dire son graphe

$$\Theta(F):=\left\{(i,j),\ 1\le j\le i-1,\ 1\le i\le N,\ F_{ij}\neq 0\right\} \tag{4.2-2}$$

Il s'agit évidemment de trouver un compromis : plus ce graphe sera étendu et plus l'erreur (4.2-1) sera petite mais plus le calcul et le stockage de ce qui n'est (dans le cas qui nous intéresse) qu'un préconditionneur vont être coûteux. Généralement, les préconditionneurs sont récursifs et dans leur niveau de base, ils imposent à F la même structure que celle de C : $\Theta(F)=\Theta(C)$.

Remarques :
- *Initialement, ces factorisations incomplètes ont été développées pour résoudre itérativement un système linéaire de type* (P_1)

$$\mathrm{FF}^T u^{i+1}=f-\mathrm{Bu}^i \tag{4.2-3}$$

  Le «nerf de la guerre» étant alors le rayon spectral $\rho\left[\left(\mathrm{FF}^T\right)^{-1}B\right]$ *qu'un choix judicieux de* $\Theta(F)$ *peut contribuer notablement à faire chuter.*
- *Ce principe de factorisation incomplète se généralise sans peine au cas standard où l'opérateur s'écrit* $K=LU$ *avec cette fois* $B=K-LU$ *. On parle alors de factorisation Incomplète de type LU (ILU pour 'Incomplete LU').*

4.2.2 Stratégie retenue dans Code_Aster

Il s'agit d'un préconditionneur de type ILU (car nous verrons au paragraphe suivant que les matrices de travail de *Code_Aster* perdent souvent leur définie-positivité) inspiré des travaux de H. Van der Vorst[Van01]. Les matrices restant toutefois symétriques, on peut écrire $K = LDL^T$ et $B = K - LDL^T$.

Remarque :

- *La matrice n'étant plus SPD mais simplement symétrique régulière on n'est, a priori, pas assuré de l'existence d'une factorisée LDL^T sans avoir recours à des permutations de lignes et de colonnes ($PK = LDL^T$ avec P matrice de permutation). Scénario qui n'a été prévu dans les solveurs linéaires natifs de Code_Aster ('LDLT', 'MULT_FRONT', 'GCPC') et serait difficile à mettre en œuvre efficacement compte-tenu du stockage MORSE des matrices. Heureusement, on verra qu'un heureux concours de circonstance permet de débloquer la situation (cf. §5.1) !*

En toute rigueur on devrait parler de factorisation incomplète de type ILDLT mais dans la littérature et dans les documentations des codes, on amalgame déjà ILU et IC, voire leurs variantes, ce n'est donc pas la peine d'enrichir la liste des acronymes !

Cette factorisation incomplète, dans la droite ligne des rappels théoriques précédents, s'appuie sur deux constats que nous allons maintenant étayer :

- la notion de remplissage par niveaux,
- la faible magnitude des termes résultants de ce remplissage.

4.2.3 Remplissage par niveaux

La construction de la factorisée s'effectue, ligne par ligne, via la formule habituelle

$$L_{ij} = \frac{1}{D_j}\left(K_{ij} - \sum_{k=1}^{j-1} L_{ik} D_k L_{jk} \right) \tag{4.2-4}$$

D'où le phénomène de remplissage progressif du profil ('fill-in' en anglais) : initialement la matrice L à le même remplissage que le matrice K, mais au cours du processus, à un terme nul de K_{ij} peut correspondre un terme non nul de L_{ij}. Il suffit qu'il existe une colonne k ($< j$) comportant un terme non nul pour les lignes i et j (cf. figure 4.2-1).

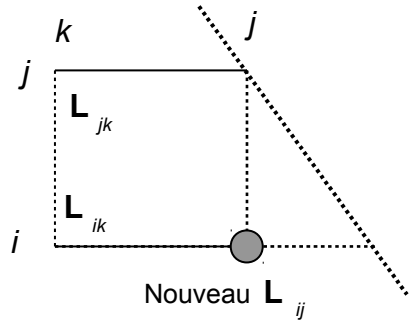


Figure 4.2-1. Phénomène de remplissage lors de la factorisation.

Ces termes non nuls pouvant d'ailleurs eux mêmes correspondre à des remplissages antérieurs, d'où une notion de niveau de récursivité pouvant s'interpréter comme autant de «niveaux» de remplissage. On parlera ainsi de factorisée incomplète de niveau 0 (stockée dans $L(0)$) si elle reproduit à l'identique la structure (mais bien sûr pas les valeurs qui sont différentes) de la partie diagonale inférieure stricte de K (i.e. le même graphe). La factorisée de niveau 1 (resp. $L(1)$) pourra elle inclure le remplissage induit par des termes non

nuls de K (termes notés r^1 dans la figure 4.2-2), celle de niveau 2 (resp. $L(2)$) pourra y mêler les nouveaux termes non nuls (r^1) précédents pour constituer d'éventuels nouveaux termes (notés r^2), et ainsi de suite récursivement…

Ceci est illustré sur le cas d'école d'une matrice creuse pentadiagonale résultant de la discrétisation différences finies du Laplacien sur une grille uniforme 2D (cf. figure 4.2-b). On n'en représente que la structure : d , termes diagonaux spectateurs, *, termes initialement non nuls, r^i , termes remplis à l'étape n°i.

La factorisée incomplète $L(3)$ conduit ici à un remplissage complet, l'erreur relative sera alors nulle $\Delta=0$ et le GCPC convergera en une itération. L'intérêt de l'exercice est bien sûr purement didactique !

$$K=\begin{bmatrix} * & & & & & & & \\ * & * & & & \text{sym} & & & \\ 0 & * & * & & & & & \\ 0 & 0 & * & * & & & & \\ * & 0 & 0 & * & * & & & \\ 0 & * & 0 & 0 & * & * & & \\ 0 & 0 & * & 0 & 0 & * & * & \end{bmatrix} \quad L(0)=\begin{bmatrix} d & & & & & & & \\ * & d & & & 0 & & & \\ 0 & * & d & & & & & \\ 0 & 0 & * & d & & & & \\ * & 0 & 0 & * & d & & & \\ 0 & * & 0 & 0 & * & d & & \\ 0 & 0 & * & 0 & 0 & * & d & \end{bmatrix}$$

$$L(1)=\begin{bmatrix} d & & & & & & & \\ * & d & & & 0 & & & \\ 0 & * & d & & & & & \\ 0 & 0 & * & d & & & & \\ * & r^1 & 0 & * & d & & & \\ 0 & * & r^1 & 0 & * & d & & \\ 0 & 0 & * & r^1 & 0 & * & d & \end{bmatrix} \quad L(2)=\begin{bmatrix} d & & & & & & & \\ * & d & & & 0 & & & \\ 0 & * & d & & & & & \\ 0 & 0 & * & d & & & & \\ * & r^1 & 0 & * & d & & & \\ 0 & * & r^1 & r^2 & * & d & & \\ 0 & 0 & * & r^1 & r^2 & * & d & \end{bmatrix}$$

Figure 4.2-2. Structure des différentes factorisées incomplètes ILU(p) *sur le cas d'école du Laplacien.*

4.2.4 Faible magnitude des termes résultants du remplissage

D'autre part, on remarque empiriquement une propriété très intéressante de ces nouveaux termes issus du remplissage: leur valeur absolue décroît en s'éloignant des zones déjà remplies par les termes non nuls de K. Soit, dans le cas précédent, de la sous-diagonale principale et de celle externe. Pour s'en convaincre il suffit de visualiser la fonction suivante (cf. figure 4.2-3)

$$y(k):=\log_{10}\left(\sqrt{\frac{\sum_{i=1}^{N-k} L_{k+i,i}^2}{N-k}}\right) \tag{4.2-5}$$

qui représente les ordres de grandeur des termes de la k ième sous-diagonale. Par exemple $y(0)$ correspond à la diagonale principale.

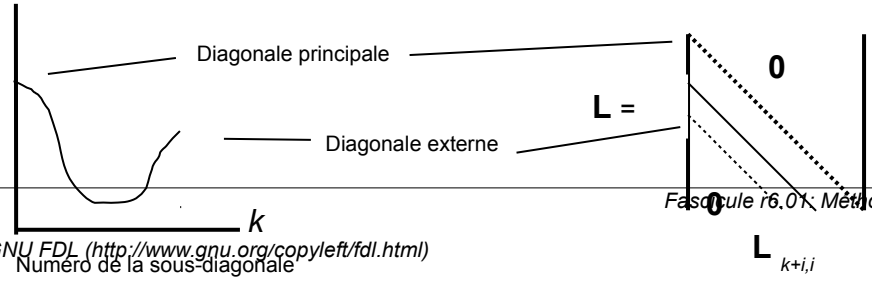


Figure 4.2-3. Importance relative des diagonales de L

Si on récapitule, on a donc :
- un niveau de remplissage récursif et paramétrable,
- une moindre importance des termes correspondant à des niveaux de remplissage élevés.

D'où un certain «blanc-seing» laissé aux factorisations incomplètes $ILU(p)$ négligeant ces termes médians.

Cette approximation «à bon compte» de K^{-1} servira alors de préconditionneur dans l'algorithme 5 du GCPC ($M^{-1}=K^{-1}$). Pour conserver un certain intérêt à la chose (sinon autant résoudre le problème directement !), on se limite aux premiers niveaux de remplissage : $p=0,1,2$ voire 3 . Tout dépend du nombre de systèmes linéaires que l'on aura à résoudre avec la même matrice.

Remarques :
- *Il existe peu de résultats théoriques sur ce type de préconditionneur. Ce qui ne l'empêche pas de se révéler souvent efficace [Greg].*
- *C'est une solution proposée par bon nombre de grands codes en mécanique des structures (ANSYS, Zébulon, NASTRAN…) et par toutes les librairies d'algèbres linéaires (PETSc, TRILINOS, HYPRE…). Par contre, cette phase de préconditionnement n'est pas toujours parallélisée. Seul le solveur de Krylov (GC, GMRES…) l'est. On retrouve alors les problèmes de 'parallel preconditioning' déjà évoqués.*

4.2.5 Complexité et occupation mémoire

Pour ce qui est du coût calcul supplémentaire avec une factorisée incomplète, il est très difficile à estimer et, en pratique, dépend grandement de la manière dont elle a été codée (à notre connaissance, il n'existe pas de résultats théoriques sur la chose). Avec un faible niveau de remplissage, on espère seulement qu'il est très inférieur au $\Theta\left(\frac{N^3}{3}\right)$ d'une factorisation complète.

Car un compromis est à trouver entre occupation mémoire et complexité. Pour constituer ces factorisées incomplètes, il faut souvent allouer des espaces de travail temporaires. Ainsi, dans l'implantation du préconditionnement $ILU(p)$ de *Code_Aster*, il a été choisi de faciliter l'algorithmique (tri, recherche d'indices et de coefficients dans le profil, gestion de la récursivité et du remplissage…) en créant provisoirement des matrices représentant le stockage plein de la matrice initiale et de son remplissage.

A cet espace de travail temporaire, se rajoute bien sûr le stockage supplémentaire dû au préconditionneur final. Au niveau 0, il est théoriquement au moins du même ordre que celui de la matrice. D'où une complexité mémoire effective minimale du GCPC de $\Theta(2\,cN)$ réels et $\Theta(2\,cN+2\mathrm{N})$ entiers. Lorsqu'on monte en complétude, seule la pratique peut apporter un semblant de réponse.

En résumé, on montre empiriquement avec *Code_Aster*, qu'il faut prévoir un encombrement mémoire total de $\Theta(\alpha\,cN)$ réels et $\Theta(\alpha\,cN+2\mathrm{N})$ entiers avec
- $\alpha=2{,}5$ en $ILU(0)$ (niveau par défaut dans *Code_Aster*),
- $\alpha=4{,}5$ en $ILU(1)$,
- $\alpha=8{,}5$ en $ILU(2)$.

Pour plus de précisions sur l'implantation informatique du GCPC dans *Code_Aster* et son utilisation sur des cas-tests quasi-industriels, on pourra consulter les notes [Greg] ou [Boi03].

5 Le GCPC «natif» de *Code_Aster*

Deux types de gradient conjugué sont utilisables :

1. Le gradient conjugué «historique»[Greg] du code (mot-clé SOLVEUR/METHODE='GCPC'), complètement intégré à ses sources et adhérent à son progiciel de gestion mémoire.
2. Celui de PETSC[PET] (mot-clé SOLVEUR/METHODE='PETSC'+ALGORITHME='CG') appelé en tant que librairie externe.

Ce chapitre aborde les difficultés d'implantation dans *Code_Aster* du premier. Le chapitre suivant traite plus spécifiquement de PETSC.

5.1 Difficultés particulières

5.1.1 Prise en compte des conditions limites

Dans *Code_Aster*, il y a deux manières de prendre en compte les conditions aux limites et cette étape s'effectue lors de la construction effective de la matrice de rigidité :

- Par double dualisation[Pel01] (opérateurs AFFE_CHAR_ACOU/MECA/THER) en utilisant des ddls spécifiques, dits de Lagrange, qui englobent les groupes d'inconnues concernées et permettent de vérifier tous types de conditions limites linéaires (Dirichlet généralisés)
  - $\mathrm{T\,u} = 0$ (5.1-1)

  avec T matrice réelle de taille $p \times n$. Cette technique étoffe la matrice de rigidité en une nouvelle matrice, dite «dualisée», qui devient la nouvelle matrice de travail
  - $$\tilde{\mathrm{K}} = \begin{pmatrix} \mathrm{K} & \beta\,\mathrm{T}^{\mathrm{T}} & \beta\,\mathrm{T}^{\mathrm{T}} \\ \beta\,\mathrm{T} & -\alpha\,\mathrm{Id} & \alpha\,\mathrm{Id} \\ \beta\,\mathrm{T} & \alpha\,\mathrm{Id} & -\alpha\,\mathrm{Id} \end{pmatrix} \quad (5.1\text{-}2)$$

  où α et β sont deux réels strictement positifs.

- Par simple élimination des inconnues (opérateurs AFFE_CHAR_CINE) en substituant et en effectuant la mise a zéro des p lignes et colonnes concernées de la matrice de rigidité. Ceci n'est valable que pour des blocages de ddls, on ne peut donc pas prendre en compte de relation linéaire. La matrice de rigidité s'écrit alors
  $$\tilde{\mathrm{K}} = \begin{pmatrix} \bar{\mathrm{K}} & 0 \\ 0 & \mathrm{Id} \end{pmatrix} \quad (5.1\text{-}3)$$

  en notant $\bar{\mathrm{K}}$ sa partie inchangée.

Chacune des deux approches a ses avantages et ses inconvénients : généricité, modularité mais augmentation de la taille du problème, dégradation de son conditionnement et perte de sa définie positivité pour la première. Contrairement à la seconde qui en diminue la taille mais qui est circonscrite à certains types de conditions limites et est, informatiquement, plus délicate à mettre en œuvre.

Remarque :

D'autres approches étaient envisageables : simple dualisation, prise en compte des conditions limites dans la formulation variationnelle, gradient conjugué projeté…

5.1.2 Conséquence sur le GCPC

Avec AFFE_CHAR_CINE , l'opérateur de travail $\tilde{\mathrm{K}}$ restant SPD, toute la théorie rappelée dans les paragraphes précédents s'applique. Par contre, avec AFFE_CHAR_ACOU/MECA/THER (cas le plus fréquent en pratique), ce

n'est plus le cas, car il devient simplement symétrique et perd son caractère défini positif. Les conséquences sont alors de trois ordres :

- La convergence globale du GCPC (cf. (3.2-9)) n'est plus garantie,
- Lorsqu'elle se produit, elle est ralentie (cf. (3.2-11)) par un conditionnement dégradé , $\eta(K) \ll \eta(\tilde{K})$
- Le préconditionnement ne peut plus être effectué *via* une $IC(p)$, mais plutôt par une $ILU(p)$ (cf. §4.2). Faut il encore que la factorisation LDL^T soit toujours possible sans avoir à permuter ligne ou colonne !

Heureusement, une disposition adéquate des multiplicateurs de Lagrange par rapport aux groupes de ddls qu'ils concernent (ils doivent englober ces degrés de liberté cf. [Pel01] §4), permet d'obtenir sans coup férir cette factorisation incomplète (ouf !).

Remarques :

- *Dans un rapport EDF B.Nitrosso[Ni92] montre que le préconditionnement diagonal n'est pas envisageable (cf. §4.1) , car il conduit à l'annulation du produit scalaire au dénominateur de l'étape (7) de l'algorithme 5 : calcul du paramètre de conjugaison optimal. Donc, contrairement à Code_Saturne, Code_Aster ne peut pas proposer cette option peu fiable.*
- *Il n'en reste pas moins, qu'avec des conditions limites dualisées, le conditionnement de l'opérateur de travail est dégradé et donc, la convergence du GCPC ralentie. D'autres codes commerciaux, à l'instart d'ANSYS [PLM], ont déjà fait ce constat.*

5.1.3 Encombrement mémoire

Compte tenu des éléments du §4.2, on a noté empiriquement une complexité mémoire effective du GCPC avec le préconditionneur de type Cholesky Incomplet ('LDLT_INC'), en α fois la taille de K avec :

- $\alpha = 2,5$ en $ILU(0)$ (niveau par défaut dans *Code_Aster*),
- $\alpha = 4,5$ en $ILU(1)$,
- $\alpha = 8,5$ en $ILU(2)$.

Avec le préconditionneur utilisant une factorisation simple précision issue de MUMPS ('LDLT_SP'), la consommation mémoire est plus importante (au moins $\alpha > 10$). Mais en non linéaire, cette factorisation peut être conservée pendant plusieurs résolutions de systèmes (mot-clé REAC_PRECOND). Le surcoût calcul dû à sa construction est donc finalement moindre.

Remarques :

- *Ces chiffres sont à comparer à la taille de la factorisée complète qui est de l'ordre typiquement de $\alpha = 30$ à 100 (cela dépend notamment du renuméroteur utilisé en pré-traitement). La disproportion de ces chiffres illustre bien l'avantage concurrentiel principal du GCPC par rapport aux solveurs directs : l'occupation mémoire requise.*
- *Cependant, contrairement aux solveurs directs de Code_Aster , le GCPC n'a pu bénéficier d'une pagination mémoire. Toute la matrice doit être contenue en RAM car elle est utilisée un grand nombre de fois via le produit matrice-vecteur. Alors que les méthodes directes manipulent des objets beaucoup plus gros mais ceux-ci sont déchargés automatiquement et partiellement sur disque (progiciel JEVEUX et/ou facultés Out-Of-Core de MUMPS).*

5.1.4 Parallélisme

Contrairement à d'autres solveurs du code (MULT_FRONT, MUMPS, PETSC) et pour des problèmes de 'parallel preconditioning', le GCPC natif n'est utilisable qu'en séquentiel (cf. § 3.3).
Par exemple, il ne bénéficie pas de la distribution des tâches et des données que la procure le parallélisme MPI mis en place autour du solveur direct MUMPS. Ainsi, un surcoût mémoire de $\alpha = 30$ avec MUMPS In-Core séquentiel peut très vite descendre au niveau d'un GCPC préconditionné par *ILU(0)* dès qu'on utilise une douzaine de processeurs et/ou qu'on active ses facultés Out-Of-Core.

5.2 Périmètre d'utilisation

Tous les opérateurs générant des systèmes symétriques réels sauf ceux requérant obligatoirement une détection de singularité (calcul modal, stabilité et flambement ...). En non linéaire, si le problème est réel non symétrique, on peut utiliser ce solveur pourvu qu'on force la symétrisation (cf. §5.3).
Liste des opérateurs *Code_Aster* pouvant utiliser GCPC :

- CALC_FORC_AJOU,
- CALC_MATR_AJOU,
- CALC_PRECONT → Approximation symétrisée possible (cf. §5.3),
- DYNA_LINE_TRAN,
- DYNA_NON_LINE → Approximation symétrisée possible,
- MACRO_MATR_AJOU,
- ASSEMBLAGE,
- MECA_STATIQUE,
- NUME_DDL/FACTORISER/RESOUDRE,
- MODE_STATIQUE,
- STAT_NON_LINE → Approximation symétrisée possible,
- THER_LINEAIRE,
- THER_NON_LINE → Approximation symétrisée possible,
- THER_NON_LINE_MO → Approximation symétrisée possible.

5.3 Caractère symétrique de l'opérateur de travail

Si la matrice n'est pas symétrique deux cas de figures se présentent. Soit le solveur linéaire est inséré dans un processus non-linéaire (opérateurs mentionnés cf. liste §5.2), soit celui-ci est linéaire (les autres opérateurs).

Dans le premier cas, on transforme le problème initial $\mathrm{Ku}=f$ en un nouveau problème symétrisé $\frac{1}{2}\left(K+K^T\right)u=f$. On suppose par là que le solveur non linéaire englobant (algorithme de Newton) va compenser l'approximation de l'opérateur initial par $\tilde{K}:=\frac{1}{2}\left(K+K^T\right)$. Ce n'est d'ailleurs pas la seule approximation de ce processus non linéaire… le choix de la matrice tangente en est, par exemple, une autre.

Cette approximation symétrisée ne nuit pas à la robustesse et à la cohérence de l'ensemble et évite une résolution non symétrique plus dispendieuse. Cette opération est réalisée en activant le mot-clé SYME='OUI' (par défaut 'NON') du mot-clé facteur SOLVEUR et elle est licite pour tous les solveurs non linéaires du code.

Lorsque le problème est purement linéaire, on ne peut attendre aucune compensation d'un quelconque processus englobant et cette approximation devient impossible. Le recourt au GCPC natif est illicite et il faut résoudre ce système non symétrique *via* les autres solveurs linéaires. La nature de l'opérateur de travail est détectée automatiquement, nul n'est besoin de la notifier *via* la mot-clé SYME.

5.4 Paramétrage et affichage

Pour activer cette fonctionnalité, il faut initialiser le mot-clé METHODE à 'GCPC' dans le mot-clé facteur SOLVEUR [U4.50.01]. Par ailleurs, deux types de préconditionneurs sont disponibles (mot-clé PRE_COND):

- PRE_COND='LDLT_INC': Un classique Cholesky Incomplet (cf. §4.2) dont le niveau de remplissage est piloté par le mot-clé NIVE_REMPLISSAGE=k (k=0 ne signifie pas «sans préconditionnement» mais préconditionnement de type ILU(0)).
- PRE_COND='LDLT_SP': Une factorisation simple précision effectuée par le solveur direct externe MUMPS[R6.02.03]. Cette solution est, *a priori* , plus coûteuse que la première en termes de

mémoire/temps mais celle-ci est mutualisable, en non linéaire, entre différentes résolutions de systèmes linéaires (mot-clé REAC_ITER). Par contre, la robustesse et la qualité de ce préconditionneur peut accélérer grandement la convergence du GCPC.

Remarques :

- *Pour minimiser la taille du profil de la matrice de travail et de son préconditionneur (avec le Cholesky Incomplet), un algorithme de renumérotation est disponible par défaut : 'RCMK' pour 'Reverse Cuthill Mac-Kee' (cf. [LT98] §5.4.2). Il est néanmoins désactivable.*
- *Le «bras de levier» principal pour modifier le compromis «temps calcul/occupation mémoire» du GCPC, reste le préconditionneur et ses différents paramètres numériques.*

Contrairement aux méthodes directes, il a fallu fixer un nombre d'itérations maximum discriminant les calculs itératifs convergés des non convergés. Ce seuil (paramétrable) est arbitrairement fixé à la moitié des ddls du problème de travail $\hat{\mathbf{K}}$. Si au bout de *i*=NMAX_ITER étapes, le résidu relatif $\delta^i := \frac{||r^i||}{||f||}$ n'est pas inférieur à RESI_RELA, le calcul s'arrête en ERREUR_FATALE.

Mot-clé facteur	Mot-clé	Valeur par défaut	Références
SOLVEUR	METHODE='GCPC'	'MULT_FRONT'	[§1]
	PRE_COND='LDLT_INC'/ 'LDLT_SP'	'LDLT_INC'	[§4.2][§5.4]
	NIVE_REMPLISSAGE=0,1,2…	0	[§4.2]
	REAC_PRECOND	5	[§5.4]
	RENUM='RCMK' ou 'SANS'	'RCMK'	[§5.4]
	NMAX_ITER= i_{max} , nombre d'itérations maximum admissible. Si $i_{max}=0$, fixé automatiquement à $N/2$	0	[Algorithme 5]
	Test d'arrêt en résidu relatif. A l'itération *i* $\frac{||r^i||}{||f||} < \text{RESI\_RELA}$.	10^{-6}	[Algorithme 5 et §3.3]
	SYME='OUI' ou 'NON' Approximation symétrisée de l'opérateur de travail par $\tilde{K} := \frac{1}{2}\left(K + K^T\right)$ Licite uniquement en non linéaire	'NON'	[§5.2/3]

Tableau 5.4-1. Récapitulatif du paramétrage du GCPC natif.

Pour être complet sur le paramétrage et l'affichage dans le fichier message (.mess), mentionnons :

- La valeur INFO =2 trace l'évolution des normes des résidus absolu $||r^i||$ et relatif $\frac{||r^i||}{||f||}$, ainsi que le numéro d'itération correpondant, *i* , dès qu'il décroît d'au moins 10% (non paramétrable).
- A convergence, c'est-à-dire dès que $\frac{||r^i||}{||f||} < \text{RESI\_RELA}$, on rappelle en plus la valeur de la norme du résidu absolu initial $||r^0||$.

5.5 Conseils d'utilisation

Globalement, le meilleur compromis robustesse/encombrement mémoire/coût CPU semble revenir [Anf03] [Boi03] aux deux solveurs directs : la multifrontale et MUMPS. Surtout lorsqu'on traite des problèmes de type «multiples seconds membres» (ex. opérateur DYNA/STAT/THER_NON_LINE sans réactualisation de la matrice tangente).
Cependant, pour des problèmes plutôt bien conditionnés (thermique, géométrie simple avec un maillage et des caractéristiques matériaux relativement homogènes…) ou très gourmands en mémoire (plusieurs millions de ddls), le GCPC peut s'avérer une alternative intéressante. Surtout avec le préconditionneur LDLT_SP, qui est plus couteux mais aussi plus fiable..

Du fait de sa modularité naturelle et de la simplicité de ses constituants (produit matrice-vecteur et produit scalaire), le GCPC reste beaucoup plus simple à maintenir et à faire évoluer que les autres solveurs directs. C'est le solveur «passe partout» facile à implanter (du moins dans sa version de base) et très pédagogique. Il est souvent branché dans des processus plus généraux (solveurs emboîtés, solveurs d'interface de la décomposition de domaines…) ou adapté, au cas par cas, pour des structures de matrices particulières.

6 Les solveurs itératifs de Krylov *via* PETSc

6.1 La librairie PETSc

La librairie PETSc[PET] (pour 'Portable Extensive Toolkit for Scientific Computation') du laboratoire Argonne (DOE et NSF) propose, outre une grande variété de solveurs (directs, itératifs, modaux, ODE…), des utilitaires de manipulation de données distribuées et de profiling des calculs parallèles. C'est un produit du domaine public, interfacé avec différents langages (F77/90, C/C++, Python), parallélisé en MPI et qui a atteint une certaine maturité[27].
Certains codes, tel MEF++[Tar04] n'ont pas hésité à s'adosser complètement à PETSc pour tirer pleinement parti de ses aspects toolkits et de ses facultés de parallélisme. Cet aspect performance[28] est la priorité du produit qui revendique de longue date des records en la matière (résolution de systèmes linéaires de 500 millions d'inconnues en 2004).

6.2 Implantation dans *Code_Aster*

Le chantier logiciel de l'implantation de PETSc dans Code_Aster est détaillé dans la note de T.De Soza[Des07] et tracée dans l'outil de Rex *Aster*. Nous en synthétisons ici les grandes lignes :

1) Une exécution PETSc *via Code_Aster* se déroule en trois étapes : pré-allocation de l'espace mémoire et remplissage de la matrice, paramétrage du solveur de Krylov choisi et de son préconditionneur, fourniture du second membre et résolution effective.
2) La conversion du format de matrice *Aster* (format CSC 'Compressed Sparse Column') au format propre à PETSc (CSR 'Compressed Sparse Row'). En symétrique, on ne stocke dans *Aster* que la partie triangulaire supérieure alors que PETSc requiert toute la matrice. D'autre part, PETSc a besoin pour être efficace d'indicateurs particuliers : en séquentiel, le nombre moyen de termes non nuls par ligne, en parallèle, celui des blocs diagonaux et extra-diagonaux du bloc de lignes associé au processeur. Cette phase de préallocation des objets PETSc est fastidieuse mais heureusement peu coûteuse par rapport aux autres étapes[29].
3) En mode parallèle, la distribution des données est entièrement pilotée par PETSc. Ce dernier attribue un bloc de lignes contiguës à chaque processeur. L'utilisateur *Aster* ne peut pas, pour l'instant, intervenir véritablement sur ce choix (à part changer le nombre de processeurs !). C'est la même philosophie que pour le parallélisme OpenMP de la multifrontale *Aster*. Avec MUMPS, le choix est plus large : l'utilisateur peut distribuer ses données directement ou *via* des outils automatiques (éventuellement pilotables).
4) Le caractère indéfini de la plupart des matrices Aster (Lagrange, éléments finis mixtes…) empêchent l'utilisation des renuméroteurs et des préconditionneurs propres à PETSc. Pour pouvoir fonctionner efficacement, on «bluffe» l'outil en lui fournissant des matrices renumérotées[30] par le RCMK[31] d'*Aster* (qui gère spécifiquement les Lagranges).

6.3 Périmètre d'utilisation

Concernant les opérateurs *Aster*, c'est le même que pour le GCPC natif du code (cf. §5.2).

27 Le projet a débuté en 1991 et parmi ses caractéristiques citons : une vingtaine de développeurs (core-team : 9), une version par an, des dizaines de milliers d'utilisateurs.
28 Un des pères fondateurs de PETSc, William Gropp[WG], travaille aussi à la normalisation du langage MPI (MPICH). Il est très actif sur les aspects informatiques et algorithmiques du calcul parallèle.
29 Des outils de préparation des données (cf. MURGE[MUR]) ou des macro-librairies (Numerical Platon, Arcane…) pourraient à terme dégager les codes hôtes de ces contingences fastidieuses que sont le remplissage des structures de données propres à chaque produit externe.
30 Pour les préconditionneurs avec factorisation incomplète (ILU(*k*) cf. §4.2).
31 RCMK pour algorithme de renumérotation de type Reverse-Cuthill MacKee.

Pour ce qui est des fonctionnalités de la librairie PETSc, nous nous sommes limités pour l'instant à quelques solveurs linéaires itératifs de type Krylov :

- CG ('CG') : gradient conjugué standard (le même que le GCPC natif d'*Aster; M.R.Hestenes et E.Stiefel 1952*),
- CR ('CR') : résidu conjugué (*M.R.Hestenes et E.Stiefel 1952*),
- GMRES ('GMRES') : Generalised Minimal RESidual (*Saad et Schultz 1986*),
- FGMRES ('FGMRES') : Flexible Generalised Minimal RESidual
- GCR ('GCR'): Generalised Conjugate Residual (*S. C. Eisenstat, H. C. Elman, and H. C. Schultz 1983*).

et aux préconditionneurs :

- ILU(k) ('LDLT_INC'): factorisation incomplète par niveau (la même que le GCPC natif d'*Aster*; cf. §4.2),
- JACOBI ('JACOBI') : préconditionneur diagonal standard (cf. §4.1),
- SOR ('SOR') : Successive Over Relaxation (cf. §4.2).
- Boomeramg ('BOOMER') : multigrille algébrique fourni par la bibliothèque *Hypre*.
- Multilevel ('ML') : multigrille algébrique fourni par la bibliothèque *ML*.
- GAMG ('GAMG'): multigrille algébrique de la bibliothèque PETSc

Par ailleurs comme pour 'GCPC' un autre pré-conditionneur (interne à *Code_Aster*) est disponible : il s'agit de 'LDLT_SP' qui s'appuie sur une factorisation simple précision reconduite pendant plusieurs résolutions itératives.
Le préconditionneur 'BLOC_LAGR' est un préconditionneur par blocs de type Lagrangien augmenté qui s'applique aux systèmes avec multiplicateurs de Lagrange. C'est un préconditionneur interne à code_aster développé sur base PETSc. Un système avec multiplicateurs de Lagrange présente la structure de point-selle suivante :

$$\begin{pmatrix} K & B^T \\ B & 0 \end{pmatrix} \begin{pmatrix} u \\ \lambda \end{pmatrix} = \begin{pmatrix} f \\ g \end{pmatrix} \quad (1)$$

Le préconditionneur de type Lagrangien augmenté (voir [Mer15]) est de la forme :

$$M = \begin{pmatrix} K + \gamma B^T B & 2B^T \\ 0 & -\frac{1}{\gamma} I \end{pmatrix} \quad (2)$$

Enfin, il est possible de combiner le solveur itératif GMRES avec le préconditionneur LDLT_SP (dit préconditionneur de premier niveau) et un préconditionneur à mémoire limitée « Limited Memory Preconditioner » , dit préconditionneur de second niveau. Cette combinaison permet d'accélérer la résolution d'un problème non-linéaire. En effet, la solution du problème non-linéaire est obtenue en résolvant une succession de problèmes linéarisés. Le préconditionneur de second-niveau est construit à partir d'informations spectrales (paires de Ritz) issues des résolutions linéaires précédentes (voir [Mer15]).

Remarques :

- *Les méthodes CG et CR sont à réserver aux modélisations d'Aster conduisant à des matrices SPD (cas le plus répandu nonobstant les problèmes de doubles Lagranges cf. §5.1/ 6.2). En non symétrique, il faut faire appel aux autres méthodes (GMRES et GCR).*
- *Tous les préconditionneurs de PETSc proposent une kyrielle de paramètres. Pour l'instant, seul le facteur de remplissage du ILU est accessible à l'utilisateur.*
- *En mode parallèle, seul JACOBI offre un préconditionneur strictement équivalent au mode séquentiel. Les autres, au premier chef ILU et SOR, préconditionnent en utilisant le bloc diagonal local au processeur.*

6.4 Caractère symétrique de l'opérateur de travail

On peut faire les mêmes remarques que pour le GCPC natif (§ 5.3). Sauf, qu'avec PETSc, on peut utiliser un solveur de Krylov compatible avec les systèmes non symétriques (GMRES, GCR). Le paramètre 'SYME' perd donc beaucoup de son intérêt.

6.5 Paramétrage et affichage

Pour activer cette fonctionnalité, il faut initialiser le mot-clé METHODE à 'PETSC' dans le mot-clé facteur SOLVEUR [U4.50.01]. Globalement, on peut faire les mêmes commentaires qu'au § 5.4.

Mot-clé facteur	Mot-clé	Valeur par défaut	Références
SOLVEUR	METHODE='PETSC'	'MULT_FRONT'	[§1]
	ALGORITHME='CG'/'CR'/ 'GMRES'/'GCR'/'FGMRES'/'GMRES_LMP'	'FGMRES'	[§6.3]
	PRE_COND='LDLT_INC' /'JACOBI'/'SOR'/'BOOMER'/'ML'/'SANS'	'LDLT_SP'	[§6.3]
	NIVE_REMPLISSAGE=0,1,2…	0 (si LDLT_INC)	[§4.2]
	REMPLISSAGE= α taille estimée du préconditionneur	1.0 (si LDLT_INC)	[§5.1]
	RENUM='RCMK' ou 'SANS'	'RCMK'	
	NMAX_ITER= i_{max} , nombre d'itérations maximum admissible. Si $i_{max} \leq 0$, fixé automatiquement par PETSc (paramétré par PETSC_DEFAULT à maxits=10^5)	-1	[Algorithme 5] pour GC ; équivalent pour les autres solveurs de Krylov.
	Test de convergence : $\|r^i\| < \max(\text{RESI\_RELA}.\|f\|, 10^{-50})$ Test de divergence : $\|r^i\| > 10^5.\|f\|$	10^{-6}	[Algorithme 5] pour GC ; équivalent pour les autres solveurs de Krylov.
	SYME='OUI' ou 'NON' Approximation symétrisée de l'opérateur de travail. Licite uniquement en non linéaire. N'a d'intérêt que pour 'CG' et 'CR'.	'NON'	[§6.4]

Tableau 6.5-1. Récapitulatif du paramétrage de l'appel à PETSc via Code_Aster.

6.6 Conseils d'utilisation

On peut prodiguer les mêmes conseils d'utilisation que pour le GCPC natif (cf. § 5.5). Nonosbstant le fait que, contrairement au GCPC natif, certains solveurs de PETSc sont adaptés aux systèmes non symétriques.
En règle générale, l'utilisation du gradient conjugué «simple» (GCPC natif ou CG de PETSc) est peu robuste sur les modélisations Aster. On lui préfère des algorithmes plus sophistiqués : GMRES… Sur des gros calculs ($>5.10^5$ degrés de liberté) très consommateurs en résolution de systèmes linéaires (opérateurs STAT_NON_LINE, MECA_STATIQUE…), on a tout intérêt à activer le parallélisme induit par PETSc.

7 Traitement des échecs

7.1 Échecs rencontrés

Contrairement à un solveur direct, un solveur itératif aura en général tendance à échouer lors de la phase de résolution du système linéaire : par exemple si le critère de convergence n'est pas vérifié au bout du nombre maximum d'itérations préalablement fixé. Cela peut être causé par un pré-conditionneur pas assez robuste ou une matrice quasi singulière.
Des échecs sont aussi possibles lors de la phase de construction du pré-conditionneur mais sont plus rares : par exemple si MUMPS manque de mémoire lors de la factorisation simple précision pour construire le pré-conditionneur LDLT_SP. Dans ce dernier cas la solution est de relancer le calcul avec plus de mémoire ou en augmentant la valeur de PCENT_PIVOT.

7.2 Récupération en cas d'échec

Comme les solveurs directs, les solveurs itératifs de *Code_Aster* sont capables de signaler l'échec ou le succès de la résolution linéaire. Cette fonctionnalité est exploitée dans le solveur non-linéaire [R5.03.01] afin de permettre :
– la découpe du pas de temps en cas d'échec lors de la phase de résolution quel que soit le pré-conditionneur
– la réactualisation du pré-conditionneur lorsque LDLT_SP est utilisé

Lorsque LDLT_SP est utilisé, on procède d'abord à la réactualisation, puis on réalise à nouveau le pas de temps courant. Si un nouvel échec se produit, le pas de temps est alors découpé.

Il est à noter qu'aucune récupération n'est prévue en cas d'échec lors de la phase de construction du pré-conditionneur.

7.3 Mise en œuvre

L'opérateur DEFI_LIST_INST permet d'activer le traitement des échecs. En présence d'une action de découpe (ACTION='DECOUPE') ou de réactualisation du pré-conditionneur (ACTION='REAC_PRECOND'), on active dans STAT_NON_LINE ou DYNA_NON_LINE la capture des erreurs dans les solveurs itératifs.

7.4 Interception de l'exception

En cas d'échec de la stratégie de découpage ou de réactualisation du pré-conditionneur, une exception est levée. Elle peut être interceptée dans le fichier de commandes en utilisant son nom : ResolutionError.

8 Bibliographie

8.1 Livres/articles/proceedings/thèses…

[BFG] A.Bouras, V.Fraysse & L.Giraud. *A relaxation strategy for inner-outer linear solvers in DD methods* . Rapport CERFACS TR/PA/00/17 (2000).
[BGLS] J.F.Bonnans, J.C.Gilbert, C.Lemarechal & C.Sagastizabal. *Optimisation numérique: aspects théoriques et pratiques* . Ed. Springer collection mathématiques & applications, 27 (1997).
[Ci00] B.A. Cipra. *The best of the 20th century: editors name top10 algorithms.* SIAM News, 33-4 (2000).
[Che05] K.Chen. *Matrix preconditioning techniques and applications* . Ed. Cambridge University Press (2005).
[CM01] F.Chaitin-Chatelin & T.Meskausas. *Inner-outer iterations for mode solvers in structural mechanics : application to Code_Aster* . Rapport CERFACS (2001).
[Cu94] J.C. Culioli. *Introduction à l'optimisation* . Ed. Ellipse (1994).
[Dav06] T.A.Davis. *Direct methods for sparse linear systems* . Ed. SIAM (2006).
[Duf06] I.S.Duff et al. *Direct methods for sparse matrices* . Ed. Clarendon Press (2006).
[Go89] G.H.Golub & D.P.O'learly. *Some history of the conjugate gradient and Lanczos algorithms : 1948-1976* . SIAM review, 31-1 (1989), pp50-102.
[Gol96] G.Golub & C.Van Loan. *Matrix computations* . Ed. Johns Hopkins University Press (1996).
[GV97] G.H.Golub & H.A.Van der Vorst. *Closer to the solution: Iterative linear solvers. The state of the art in numerical analysis* . Ed. Clarendon press (1997), pp93-92.
[GY97] G.H.Golub & Q.Ye. *Inexact preconditioned conjugate gradient method with inner-outer iteration* . Rapport Stanford University, SCCM-97-04 (1997).
[Jo84] P.Joly. *Méthodes de gradient conjugué* . Rapport n°84016 du lab. d'analyse numérique de PARIS VI (1984).
[LT98] P.Lascaux & R.Théodor. *Analyse numérique matricielle appliquée à l'art de l'ingénieur* . Ed. Masson (1998).
[Liu89] J.W.H.Liu. *Computer solution of large sparse positive definite systems* . Prentice Hall (1981).
[Mer15] Sylvain Mercier *Solveurs non linéaires rapides en mécanique des solides*, thèse de l'Université de Toulouse
[Meu99] G.Meurant. *Computer solution of large linear systems* . Ed. Elsevier (1999).
[Min08] M.Minoux. *Programmation mathématique, théorie et algorithmes* . Ed. Lavoisier (2008).
[Saa03] Y.Saad. *Iterative methods for sparse matrices* . Ed. PWS (2003).
[Sh94] J.R.Shewchuk. *An introduction to the conjugate gradient method without the agonizing pain.* Rapport interne de l'Université de Carnegie Mellon (1994).
[SV00] Y.Saad & H.A.Van Der Vorst. *Iterative solution of linear system in the 20th-century* . J. Comp. Appl. Math., 123 (2000), pp35-65.
[Van01] H.Van Der Vorst. *Iterative Krylov methods for large linear systems* . Ed. Cambridge University Press (2001).

8.2 Rapports/compte-rendus EDF

[Anf03] N.Anfaoui. *Une étude des performances de Code_Aster : proposition d'optimisation* . Stage de DESS de mathématiques appliquées de PARIS VI (2003).
[BH89] T. Buffard & J.M.Herard. *Une méthode de préconditionnement polynomial pour des systèmes SPD* . Note EDF R&D HE-41/89.16 (1989).
[Boi01] O.Boiteau. *Algorithme de résolution pour le problème généralisé* . Documentation de Référence *Code_Aster* R5.01.01 (2001).
[Boi03] O.Boiteau. *Analyse de l'implantation du gradient conjugué dans le Code_Aster et tests d'autres variantes via la bibliothèque CXML* . Compte-rendu EDF R&D CR-I23/03/014 (2003).
[Boi08] O.Boiteau. *Décomposition de domaines et parallélisme* . Documentation de Référence *Code_Aster* . R6.01.03 (2008).
[Boi08b] O.Boiteau. *Amélioration de la robustesse du code TELEMAC en mode parallèle* . Compte-rendu EDF R&D I23/08/008 (2008).
[Boi09] O.Boiteau. *Généralités sur les solveurs linéaires directes et utilisation du produit MUMPS* . Documentation de Référence *Code_Aster* . R6.02.03 (2009).
[Des07] T.DeSoza. *Evaluation et développement du parallélisme dans Code_Aster* . Stage de Master ENPC (2007) et note EDF R&D HT-62/08/01464 (2008).
[Greg] J.P.Gregoire. *Algorithme du GCPC pour la résolution d'un système linéaire symétrique : présentation générale et mode d'emploi* . Note EDF R&D HI/4333-07 (1982).
Vectorisation efficace du GC dans le cas de matrices symétriques creuses . Note EDF R&D HI-72/6710 (1990).
Parallélisation du GC pour des matrices creuses sur CRAY C98 . Note EDF R&D HI-76/95/019 (1996).
Accélération de la convergence du gradient conjugué préconditionné en utilisant la factorisation incomplète par niveau. Note EDF R&D HI-76/00/005 (2000).
Parallélisation en mémoire distribuée du GC et du code utilisateur. Note EDF R&D HI-76/01/007 (2001).
GCPC et calcul rapide des valeurs propres. Note EDF R&D HI-76/01/001 (2001).
[Ni92] B.Nitrosso. *Méthodes directes pour matrices creuses SPD. Des techniques de base à l'état de l'art* . Note EDF R&D HE-41/92/27 (1992).
[Pel01] J.Pellet. *Dualisation des conditions limites* . Documentation de Référence *Code_Aster* R3.03.01 (2001).
[Tar04] N.Tardieu. *Parallélisation de MEF++, principes et performances* . Compte-rendu EDF R&D (2004).
[Tar09] N.Tardieu. *Solveur multigrille pour la mécanique des structures* . Note EDF R&D HT-62/08/01615 (2009).

8.3 Ressources internet

[MUR] *Site internet officiel du produit MURGE* : http://murge.gforge.inria.fr/.
[PET] *Site internet officiel de la librairie PETSc* : http://www.mcs.anl.gov/petsc/petsc-as/.
[PLM] G.Poole, Y.C.Liu & J.Mandel : Advancing analysis capabilities in ANSYS though solver technology. (2001) disponible sur le site ANSYS.
[WG] Site professionnel de William Gropp : http://www.cs.uiuc.edu/homes/wgropp .

9 Description des versions du document

Version Aster	Auteur(s) Organisme(s)	Description des modifications
3.0	J.P.GREGOIRE EDF/R&D/SINETICS	Texte initial
6.4	O.BOITEAU, J.P.GREGOIRE EDF/R&D/SINETICS J.PELLET EDF/R&D/AMA	
9.4	O.BOITEAU EDF/R&D/SINETICS	Corrections et ajout de PETSC
v10.4	O.BOITEAU EDF/R&D/SINETICS	Corrections formelles dues aux portages .doc/.odt; Corrections sur le parallélisme; Rajout de LDLT_SP.
V13.4	N.Béreux EDF/R&D/ERMES	Ajout du préconditionneur BLOC_LAGR et de l'algorithme GMRES_LMP