Manuel de Référence
Fascicule R6.01 : Méthodes itératives
Document : R6.01.02

Solveur linéaire de type gradient conjugué : étude théorique et implantation dans *Code_Aster*

Résumé :

Dans nombre de domaines, la simulation devient incontournable mais gourmande en capacité de traitement, et plus particulièrement, en résolution de 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 50 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'échappant pas à la règle. Puisqu'il propose deux solveurs directs (Gauss et multifrontal) et un itératif (GCPC pour Gradient Conjugué PréConditionné).
Dans cette note, on détaille d'un point de vue théorique, algorithmique et *Code_Aster*, les fondamentaux du GCPC, ses liens avec les solveurs directs, les méthodes de descente, celles d'optimisation continue et celles spectrales. On conclut par les difficultés d'implantation particulières du GCPC dans le code, son paramétrage et périmètre ainsi que quelques conseils d'utilisation.

Table des matières

1 Problématique

Avertissement :

Le lecteur pressé et/ou pas intéressé par les ressorts algorithmiques et théoriques du gradient conjugué peut d'emblée sauter au dernier paragraphe, [§5], qui récapitule les principaux « aspects *Code_Aster* » du GCPC.

Dans nombre de domaines, **la simulation** devient incontournable mais gourmande en capacité de calcul et plus particulièrement **en résolution de systèmes linéaires**. Ces inversions de systèmes sont en fait omniprésentes et souvent enfouies au plus profond d'autres algorithmes numériques : solveurs non-linéaires, intégration en temps, solveurs modaux…. On cherche le vecteur des inconnues nodales (ou de leurs incréments) $\mathbf{u}$ vérifiant un système linéaire du type

$$\mathbf{Ku} = \mathbf{f} \qquad \text{éq 1-1}$$

avec $\mathbf{K}$ la matrice de rigidité, creuse, mal conditionnée et souvent symétrique définie positive. Le vecteur $\mathbf{f}$ traduit l'application des forces généralisées au système mécanique.

Remarque :

En mécanique des structures le conditionnement $\eta(\mathbf{K})$ est connu pour être beaucoup plus mauvais que dans d'autres domaines, tels la mécanique des fluides par exemple. 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. [bib13])

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 50 ans deux types de solveurs se disputent la suprématie dans le domaine, **les solveurs directs et ceux itératifs** (la frontière entre les deux n'est d'ailleurs absolument pas étanche. Une méthode qualifiée de directe peut être considérée comme itérative par la suite (par exemple les méthodes de Krylov). D'autre part les méthodes itératives peuvent faire appel ponctuellement aux solveurs directs (préconditionneur de type Cholesky)) :

1) **Les premiers ont pour but de factoriser la matrice initiale sous une forme canonique** (LU, LDL^T…) permettant une résolution beaucoup plus aisée, par descente-remontée sur des systèmes triangulaires $\mathbf{L}$ ou $\mathbf{U}$ appropriés. En pratique, on effectue cette opération sur une permutée de la matrice initiale afin de limiter le remplissage du profil de la factorisée et de tenir compte du caractère creux de l'opérateur. Ces permutations/renumérotations/pivotages peuvent se déclencher à différents niveaux de l'algorithmes et sur des éléments de tailles variables (matrice, super-blocs, blocs…).
2) La théorie de ces méthodes est relativement bien ficelée (en arithmétique exacte comme en arithmétique finie), leur achèvement est quasi-assuré en un nombre fini d'opérations connu par avance et leur précision est aussi bonne que celle du problème initial. Leurs déclinaisons suivant moult types de matrices et d'architectures logicielles sont très complètes (packages PETSc, LAPACK, WSMP, MUMPS, UNFPACK… cf. [bib4] [§3.1]).
   Les solveurs itératifs démarrent eux à partir d'une estimation initiale de la solution et, à chaque itération, tentent de construire des vecteurs qui vont approcher la solution discrétisée (algorithmes basés sur le splitting d'opérateurs et le théorème du point fixe ou sur la minimisation d'une fonctionnelle quadratique). Ce processus est arrêté lorsqu'un critère de convergence est satisfait (en général un résidu relatif inférieur à une certaine valeur). En pratique, on effectue cette opération sur une matrice initiale préconditionnée (qui a de meilleures propriétés théoriques que l'opérateur initial vis-à-vis de l'algorithme), on tient

compte du caractère creux de l'opérateur et on cherche à optimiser en temps calcul l'outil élémentaire essentiel de ces algorithmes : le produit matrice-vecteur.
La théorie de ces méthodes comporte de nombreux « problèmes ouverts », surtout en arithmétique finie. En pratique, leur convergence n'est *a priori* pas connue, elle dépend de la structure de la matrice, du point de départ, du critère d'arrêt, du paramétrage des traitements numériques associés…
Contrairement à leurs homologues directs, il n'est pas possible de fournir LE solveur itératif qui va permettre de résoudre n'importe quel système linéaire. L'adéquation du type d'algorithme à une classe de problèmes se fait au cas par cas.
Même si, historiquement, les solveurs itératifs ont toujours eu droit de cité car pour certaines applications ils fonctionnent beaucoup mieux que les solveurs directs et ils requièrent théoriquement (à gestion mémoire équivalente) moins de mémoire.
Par exemple, dans la version de base du GC, on a juste besoin de connaître l'action de la matrice sur un vecteur. Il n'est donc pas nécessaire, *a priori*, de stocker entièrement ladite matrice pour résoudre le système ! D'autre part, contrairement aux solveurs directs, les itératifs ne sont pas soumis au « diktat » du phénomène de remplissage (« fill-in » en anglais) qui gangrène le profil des matrices creuses et aux avatars du pivotage qui nécessite de les stocker entièrement (on verra que dans *Code_Aster*, les propriétés de l'opérateur de travail et des astuces algorithmiques permettent néanmoins un stockage creux optimisé même pour les solveurs directs).

Finalement, 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 !
Dans la littérature en analyse numérique [bib26] [bib27] [bib14] [bib35] [bib40], on accorde souvent une place prépondérante aux solveurs itératifs en général, et, aux variantes du gradient conjugué en particulier. **Les auteurs les plus chevronnés** [bib16] [bib36] **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 !

D'où une **offre diversifiée des codes de mécanique dans le domaine** (cf. [bib4] [§8.3]): méthodes directes (Gauss, multifrontale…) ou itératives (Gauss-Seidel, SOR, GCPC, GMRES…). Ce choix des briques de base (l'algorithme de résolution) se décline ensuite suivant toute une kyrielle de traitements numériques qui interviennent à différents niveaux : stockage de la matrice, renuméroteur, préconditionneur, balancing…
***Code_Aster* n'échappe pas lui aussi à ce principe de précaution par la diversité de l'offre**. Ses résolutions de systèmes linéaires se structurent autour de trois solveurs (via le mot-clé SOLVEUR des opérateurs, cf. [§5.3], [§5.4]) :

1) Un **solveur direct de type Gauss** ([bib37], mot-clé 'LDLT'), avec ou sans la renumérotation Reverse Cuthill-Mac-kee, mais sans pivotage (à cause de son stockage SKYLINE). Ce stockage est complètement paginé (via le mot-clé TAILLE_BLOC de l'opérateur DEBUT) et donc permet le passage de gros cas avec une taille mémoire « aussi petite que l'on veut » et une robustesse aussi bonne que possible en arithmétique finie (aux erreurs d'arrondis près donc, cf. le mot-clé NPREC). Cependant, c'est au détriment de la CPU consommée (les accès disque sont coûteux), sachant que la complexité calcul intrinsèque de l'algorithme est déjà élevée : en $\vartheta\left(\frac{N^3}{3}\right)$ où N est la taille du système.
2) Un **solveur direct de type Multifrontal** ([bib33], mot-clé 'MULT_FRONT'), avec les renumérotations MD, MDA ou METIS et un stockage de la matrice MORSE. Il est partiellement paginé (seule la matrice initiale doit tenir en mémoire centrale avec quelques blocs de la matrice en cours de factorisation. Mais cette pagination n'est pas paramétrable par l'utilisateur, elle est intrinsèque aux choix opérés par le partitionneur et donc à la structure de

la matrice initiale) et parallélisé en mémoire partagé. Il allie robustesse, taille mémoire modulable, pour un coût CPU modéré (en raison de son organisation par blocs qui s'appuie, en plus, sur des bibliothèques mathématiques optimisées). **C'est donc tout naturellement LA METHODE PAR DEFAUT DU CODE**.

3) Un **solveur itératif de type Gradient Conjugué Préconditionné** ([bib22] [bib3], mot-clé 'GCPC'), avec ou sans renumérotation Cuthill-Mac-Kee, stockage des matrices (de la matrice initiale et de la matrice de préconditionnement) MORSE et préconditionnement ILU(p) (par une factorisée de Cholesky incomplète d'ordre p de la matrice initiale cf. [§4.2]). Comme tout processus itératif, sa convergence (en un nombre d'itérations raisonnable) n'est pas acquise en pratique. Compte-tenu de son processus algorithmique (chaque bloc matriciel est utilisé un grand nombre de fois, pour les produits matrice-vecteur, contrairement aux solveurs directs), le GCPC n'est pas paginé et la matrice doit donc tenir en un seul bloc de la mémoire centrale. Sa complexité calcul dépend du caractère creux de l'opérateur initial, de son conditionnement numérique, de l'efficacité du préconditionneur et de la précision requise.

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 : les **méthodes de descente** classiques et générales, ainsi que leurs liens avec le GC. Celui linéaire de la résolution de système SPD et celui non linéaire de l'optimisation continue.
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 et/ou problématiques 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 celle retenue par le GCPC de *Code_Aster*. On insiste notamment sur la notion de factorisation incomplète par niveaux, son principe et les heureux concours de circonstances qui la rendent licite.
On conclut par **les difficultés d'implantation particulières du GCPC dans le code** (prise en compte des conditions limites, encombrement mémoire, parallélisation), son **paramétrage** et **périmètre** ainsi que **quelques conseils d'utilisation**.

L'objet de ce document n'est pas de détailler, ni même d'aborder, tous les aspects potentiels du GC. Plusieurs notes HI n'y parviendraient pas tant il y a pléthore de travaux sur le sujet. La bibliographie proposée en fin de document en constitue un premier échantillon.
Nous avons néanmoins essayé d'aborder les aspects principaux en donnant un maximum de pistes, d'idées, de références, entremêlant les visions numériciennes, mécaniciennes et *Code_Aster*. Un effort particulier a été apporté pour mettre en perspective les choix conduits dans *Code_Aster* par rapport à la recherche passée et actuelle. On a de plus essayé de constamment lier les différents items abordés tout en s'interdisant toutes digressions mathématiques.
Cette note deviendra la nouvelle documentation de référence du code sur le GCPC (doc. [R6.01.02]).

Les figures [Figure 2.1-a] à [Figure 4.1-a] ont été empruntées au papier introductif de J.R. Shewchuk [bib38] avec son aimable autorisation : ©1994 by Jonathan Richard Shewchuk.

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 très souvent 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 $\mathbf{f}$ vecteur traduisant l'application de forces généralisées au système thermo-mécanique

$$(P_1) \quad \mathbf{Ku} = \mathbf{f} \qquad \textbf{éq 2.1-1}$$

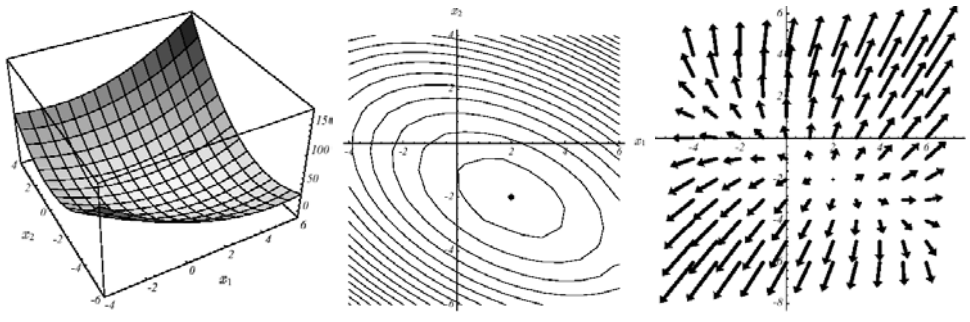
- Minimisation de la fonctionnelle quadratique représentant l'énergie du système, avec $<\ ,\ >$ le produit scalaire euclidien usuel,

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

Du fait de la « définie-positivité » de la matrice qui rend J strictement convexe, le vecteur annulant ∇J (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 ») correspond à l'unique (sans cette convexité, on est pas assuré de l'unicité. Il faut alors composer avec des minima locaux !) minimum global $\mathbf{u}$. Cela s'illustre par la relation suivante, valable quelle que soit $\mathbf{K}$ symétrique,

$$J(\mathbf{v}) = J(\mathbf{u}) + \frac{1}{2}(\mathbf{v}-\mathbf{u})^T\mathbf{K}(\mathbf{v}-\mathbf{u}) \qquad \textbf{éq 2.1-3}$$

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-a : Exemple de J quadratique en N=2 dimensions avec $\mathbf{K} := \begin{bmatrix} 3 & 2 \\ 2 & 6 \end{bmatrix}$ et $\mathbf{f} := \begin{bmatrix} 2 \\ -8 \end{bmatrix}$:
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 représentation limpide (cf. [Figure 2.1-a]) : 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 $\mathbf{e}(\mathbf{v}) = \mathbf{v} - \mathbf{u}$, plus parlante pour les mécaniciens,

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

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 $\mathbf{r}(\mathbf{v}) = \mathbf{f} - \mathbf{K}\mathbf{v}$

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

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*

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

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

$$(P_2)'' \quad \underbrace{\mathbf{K}^T\mathbf{K}}_{\widetilde{\mathbf{K}}}\mathbf{u} = \mathbf{K}^T\mathbf{f} \qquad \textbf{éq 2.1-7}$$

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-a], elle s'exprime trivialement sous forme d'intersections de droites :*

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

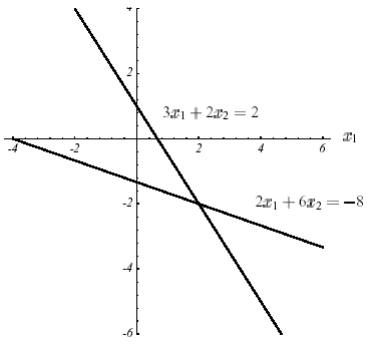


Figure 2.1-b : Résolution de l'exemple n°1 par intersection de droites.

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.

- *Lorsque la matrice n'est pas définie positive (cf. [Figure 2.1-a] (a)), trois cas de figures se présentent : elle est définie négative ((b), cela ne pose pas de problème, on travaille avec* $\widetilde{\mathbf{K}} = -\mathbf{K}$ *pour minimiser au lieu de maximiser), singulière ((c), l'ensemble des solutions, si il existe, est un hyperplan) ou quelconque ((d), il s'agit d'un problème de point selle sur lequel les méthodes de type descente ou gradient conjugué achoppent).*

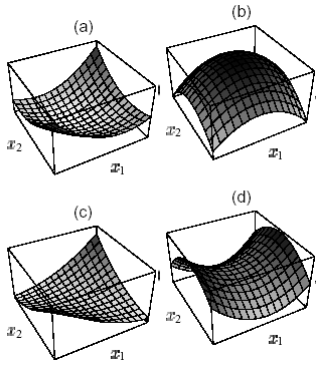


Figure 2.1-c : Forme de J suivant les propriétés de K.

- *L'erreur* $\mathbf{e(v)}$ *qui exprime la distance de la solution intermédiaire à la solution exacte et le résidu* $\mathbf{r(v)}$*, l'erreur que cette solution intermédiaire implique sur la résolution du système linéaire, sont liées par la relation*

$$\mathbf{r}(\mathbf{v}) = -\mathbf{K}\,\mathbf{e}(\mathbf{v}) \qquad \textbf{éq 2.1-9}$$

Plus important, ce résidu est l'opposé du gradient de la fonctionnelle

$$\mathbf{r}(\mathbf{v}) = -\nabla J(\mathbf{v}) \qquad \textbf{éq 2.1-10}$$

Il faut donc interpréter les résidus comme des directions de descente potentielles permettant de diminuer les valeurs de J *(elles sont orthogonales aux lignes d'isovaleurs cf. [Figure 2.1-a]). Ces rappels basiques s'avèreront utiles par la suite.*

2.2 Steepest Descent

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 $\mathbf{d}^i = -\nabla J^i = \mathbf{r}^i$ avec $J^i := J\left(\mathbf{u}^i\right)$ et $\mathbf{r}^i := \mathbf{f} - \mathbf{K}\mathbf{u}^i$. A la $i^{\text{ème}}$ itération, on va donc chercher à construire $\mathbf{u}^{i+1}$ tel que :

$$\mathbf{u}^{i+1} := \mathbf{u}^i + \alpha^i \mathbf{r}^i \qquad \textbf{éq 2.2-1}$$

et

$$J^{i+1} < J^i \qquad \textbf{éq 2.2-2}$$

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)

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

Les figures suivantes illustrent le fonctionnement de cette procédure sur l'exemple n°1 : partant du point $\mathbf{u}^0$=[-2,-2]^T (cf. (a)) on cherche le paramètre de descente optimal, α^0, suivant la ligne de plus grande pente $\mathbf{r}^0$; 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)

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

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 (le conditionnement de l'opérateur SPD $\mathbf{K}$ s'écrit comme le rapport de ses valeurs propres extrêmes $\eta(\mathbf{K}) := \frac{\lambda_{\max}}{\lambda_{\min}}$ 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), le nombre d'itérations requises peut être considérable (cf. [Figure 2.2-b](b)).

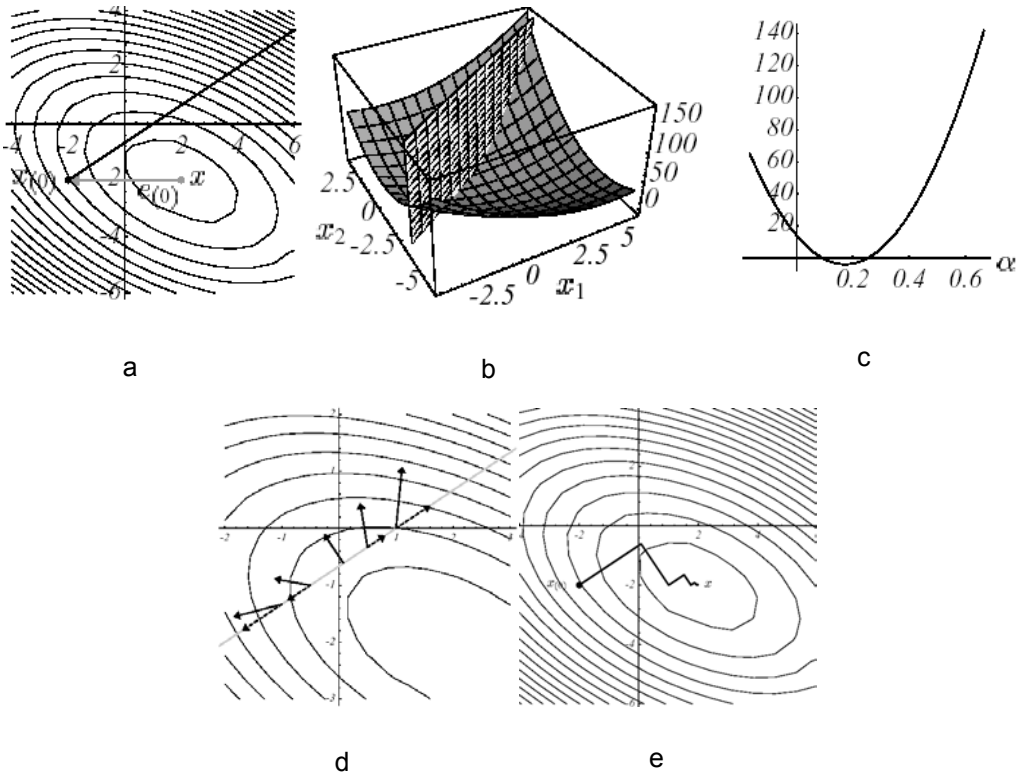


Figure 2.2-a Illustration de la Steepest Descent sur l'exemple n°1 : direction de descente initiale (a), intersection de surfaces (b), parabole correspondante (c), vecteurs gradient et leur projection le long de la direction de descente initiale (d) et processus global jusqu'à la convergence (e).

Algorithme

D'où le déroulement

Initialisation $\mathbf{u}^0$ donné	
Boucle en i	
(1) $\mathbf{r}^i = \mathbf{f} - \mathbf{K}\mathbf{u}^i$	(nouveau résidu)
(2) $\alpha^i = \dfrac{\left\|\mathbf{r}^i\right\|^2}{\left\langle \mathbf{r}^i, \mathbf{K}\mathbf{r}^i \right\rangle}$	(paramètre optimal de descente)
(3) $\mathbf{u}^{i+1} = \mathbf{u}^i + \alpha^i \mathbf{r}^i$	(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

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

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

Eléments de théorie

On montre que cet algorithme converge, pour tout point de départ $\mathbf{u}^0$, à la vitesse (la définie positivité de l'opérateur nous assure de la « bonne nature » du paramètre ω^i. C'est bien un facteur d'atténuation car $0 \le \omega^i \le 1$)

$$\left\| e\left(\mathbf{u}^{i+1}\right) \right\|_{\mathbf{K}}^2 = \omega^i \left\| e\left(\mathbf{u}^{i}\right) \right\|_{\mathbf{K}}^2$$

$$\text{avec } \omega^i := 1 - \frac{\left\| \mathbf{r}^i \right\|^4}{\left\langle \mathbf{K}^{-1}\mathbf{r}^i, \mathbf{r}^i \right\rangle \left\langle \mathbf{K}\mathbf{r}^i, \mathbf{r}^i \right\rangle} \qquad \textbf{éq 2.2-6}$$

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

$$e\left(\mathbf{u}^i\right) = \sum_j \xi_j \mathbf{v}_j \qquad \textbf{éq 2.2-7}$$

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

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

Dans [éq 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. [bib2] [§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 (quelque soit $\mathbf{K}$ matrice SPD et $\mathbf{u}$ vecteur non nul : $1 \le \frac{\left\langle \mathbf{K}\mathbf{u}, \mathbf{u} \right\rangle \left\langle \mathbf{K}^{-1}\mathbf{u}, \mathbf{u} \right\rangle}{\left\| \mathbf{u} \right\|_2^4} \le \frac{\left(\eta(\mathbf{K})^{1/2} + \eta(\mathbf{K})^{-1/2} \right)^2}{4}$) on améliore grandement la lisibilité du facteur d'atténuation [éq 2.2-6]. Au terme de i itérations, au pire, la décroissance s'exprime sous la forme

$$\left\| e\left(\mathbf{u}^i\right) \right\|_{\mathbf{K}} \le \left(\frac{\eta(\mathbf{K}) - 1}{\eta(\mathbf{K}) + 1} \right)^i \left\| e\left(\mathbf{u}^0\right) \right\|_{\mathbf{K}} \qquad \textbf{éq 2.2-9}$$

Elle assure la convergence linéaire (c'est-à-dire $\lim_{i \to \infty} \frac{J\left(\mathbf{u}^{i+1}\right) - J(\mathbf{u})}{J\left(\mathbf{u}^i\right) - J(\mathbf{u})} \le \alpha := \left(\frac{\eta(\mathbf{K}) - 1}{\eta(\mathbf{K}) + 1} \right)^2 < 1$. Le taux de convergence asymptotique α est appelé rapport de Kantorovitch) du processus en un nombre d'itérations proportionnel au conditionnement de l'opérateur. Ainsi, pour obtenir

$$\frac{\left\| e\left(\mathbf{u}^i\right) \right\|_{\mathbf{K}}}{\left\| e\left(\mathbf{u}^0\right) \right\|_{\mathbf{K}}} \le \varepsilon \ \ (\text{petit}) \qquad \textbf{éq 2.2-10}$$

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

$$i \approx \frac{\eta(\mathbf{K})}{4} \ln \frac{1}{\varepsilon} \qquad \textbf{éq 2.2-11}$$

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 [éq 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-b])

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

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 ».

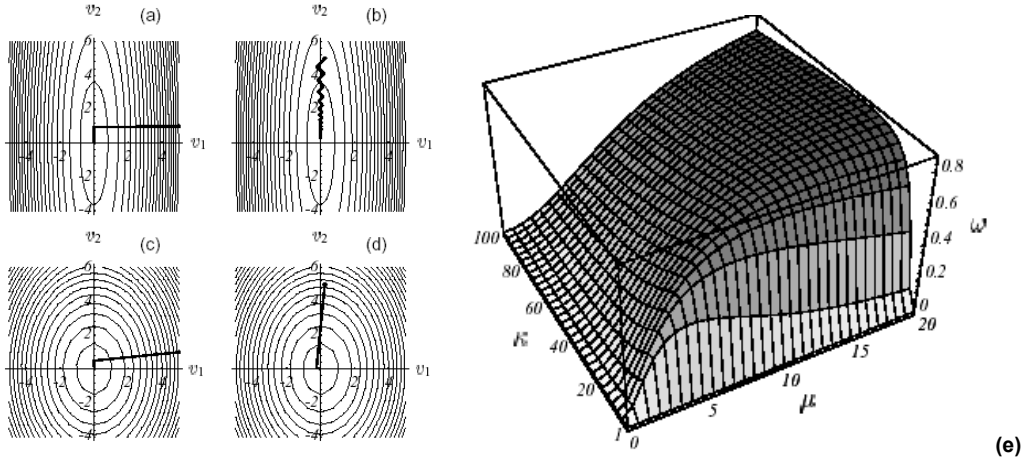


Figure 2.2-b : Convergence de la Steepest Descent sur l'exemple n°1 suivant les valeurs du conditionnement et du point de départ: κ grand et μ petit (a), κ et μ grand (b), κ et μ petit (c), κ petit et μ grand (d) et forme globale de ω=ω(κ,μ) (e).

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. [bib26] [§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. 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 [bib28] [§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. [bib35] [§5.3.2] [§5.3.3]).
- *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 $\Re^N$, ici $\mathcal{K}_i := vect(\mathbf{r}^i)$, perpendiculairement à un autre sous-espace, ici $\mathcal{L}_i := \mathcal{K}_i$. 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 $\boldsymbol{\delta}^i := \alpha^i \mathbf{r}^i$ tel que :

$$\begin{cases} \mathbf{u}^i := \mathbf{u}^0 + \boldsymbol{\delta}^i \quad \boldsymbol{\delta}^i := \alpha^i \mathbf{r}^i \in \mathcal{K}_i \\ \left\langle \mathbf{r}^0 - \mathbf{K}\boldsymbol{\delta}^i, \mathbf{w} \right\rangle = 0 \quad \forall \mathbf{w} \in \mathcal{L}_i = \mathcal{K}_i \end{cases} \qquad \textbf{éq 2.2-13}$$

  Ce cadre général constitue ce qu'on appelle les conditions de Petrov-Galerkin (cf. [bib35] [§5]).

Complexité et occupation mémoire

La majeure partie du coût calcul de l'algorithme 1 réside dans la mise à jour [éq 2.2-5] et, plus particulièrement, dans le produit matrice-vecteur $\mathbf{K}\mathbf{r}^i$. D'autre part, on a déjà mentionné que sa convergence était acquise et s'opérait en un nombre d'itérations proportionnel à la complexité de la matrice (cf. [éq 2.2-11]).
Sa complexité est donc, si on tient compte du caractère creux de l'opérateur, de l'ordre de $\mathcal{G}(\eta(\mathbf{K})cN)$ où c est le nombre moyen de termes non nuls par ligne de $\mathbf{K}$.
La discrétisation éléments finis des opérateurs elliptiques du second ordre (resp. du quatrième ordre) (cas le plus souvent rencontré en mécanique des structures) impliquant des conditionnements d'opérateurs en $\eta(\mathbf{K}) = \mathcal{G}(N^{2/d})$ (resp. $\eta(\mathbf{K}) = \mathcal{G}(N^{4/d})$), avec d la dimension d'espace, la complexité calcul de la Steepest Descent dans ce cadre s'écrit $\mathcal{G}\left(cN^{\frac{2}{d}+1}\right)$ (resp. $\mathcal{G}\left(cN^{\frac{4}{d}+1}\right)$).

Pour ce qui est de l'occupation mémoire, seul le stockage de la matrice de travail est éventuellement requis (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 DDLs (CFD, électromagnétisme…)) : $\mathcal{G}(cN)$. 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 $\mathcal{G}(cN)$ réels et $\mathcal{G}(cN+N)$ entiers.

2.3 Méthode de descente « générale »

Principe

Une étape cruciale de ces méthodes est le choix de leur direction de descente (par définition une direction de descente (dd) à la $i^{\text{ème}}$ étape, $\mathbf{d}^i$, vérifie : $\left(\nabla J^i\right)^T \mathbf{d}^i < 0$) (dd). Même si le gradient de la fonctionnelle en reste l'ingrédient principal, on peut tout à fait en choisir une autre version $\mathbf{d}^i$ que celle requise pour la Steepest Descent. A la $i^{\text{ème}}$ itération, on va donc chercher à construire $\mathbf{u}^{i+1}$ vérifiant

$$\mathbf{u}^{i+1} := \mathbf{u}^i + \alpha^i \mathbf{d}^i \qquad \textbf{éq 2.3-1}$$

avec

$$\alpha^i = \underset{\alpha\in[\alpha_m,\alpha_M]}{Arg\min} J\left(\mathbf{u}^i + \alpha\,\mathbf{d}^i\right) \qquad \textbf{éq 2.3-2}$$

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

$$\alpha^i := \frac{\left\langle \mathbf{r}^i, \mathbf{d}^i \right\rangle}{\left\langle \mathbf{d}^i, \mathbf{K}\mathbf{d}^i \right\rangle} \qquad \textbf{éq 2.3-3}$$

$$\left\langle \mathbf{d}^i, \mathbf{r}^{i+1} \right\rangle = 0$$

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

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

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 ($\mathbf{d}^i = \mathbf{r}^i$) 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 $\widetilde{\mathbf{K}}$ 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 $\mathbf{u}^0$, $\mathbf{d}^0$ donnés, $\mathbf{r}^0 = \mathbf{f} - \mathbf{K}\mathbf{u}^0$

Boucle en i

(1) $\mathbf{z}^i = \mathbf{K}\mathbf{d}^i$

(2) $\alpha^i = \dfrac{\langle \mathbf{r}^i, \mathbf{d}^i \rangle}{\langle \mathbf{d}^i, \mathbf{z}^i \rangle}$ (paramètre optimal de descente)

(3) $\mathbf{u}^{i+1} = \mathbf{u}^i + \alpha^i \mathbf{d}^i$ (nouvel itéré)

(4) $\mathbf{r}^{i+1} = \mathbf{r}^i - \alpha^i \mathbf{z}^i$ (nouvel résidu)

(5) Test d'arrêt via $J^{i+1} - J^i$, $\left\|\mathbf{d}^i\right\|$ ou $\left\|\mathbf{r}^{i+1}\right\|$ (par exemple)

(6) Calcul de la dd $\mathbf{d}^{i+1} = \mathbf{d}^{i+1}\left(\nabla J^k, \nabla^2 J^k, \mathbf{d}^k \ldots\right)$

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 ($\mathbf{d}^i = \mathbf{e}_i$), on obtient la méthode de Gauss-Seidel.*
- *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.*

Compléments

Avec une fonctionnelle J continue quelconque (cf. [Figure 2.3-a] (a)), 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 $\Rightarrow$ étape (4),
- Le calcul simplifié du paramètre de descente optimal $\Rightarrow$ étape (2).

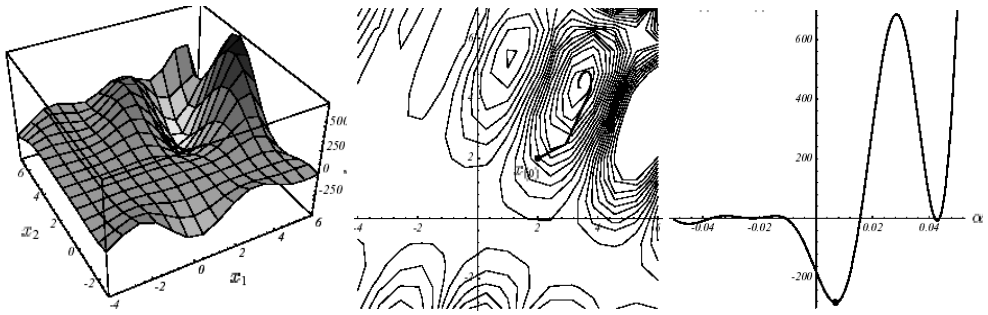
Leurs raisons d'être étant uniquement d'utiliser toutes les hypothèses du problème pour faciliter la minimisation unidimensionnelle [éq 2.3-2], on est alors contraint d'effectuer explicitement cette recherche linéaire sur une fonctionnelle plus chahutée avec cette fois de multiples minima locaux. Heureusement, 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…),
- $J, \nabla J$ (dichotomie sur les valeurs de la dérivée, règle de Wolfe, d'Armijo…),
- $J, \nabla J, \nabla^2 J$ (Newton-Raphson…)
- …

Pour ce qui est de la recherche de descente, là aussi de nombreuses solutions sont proposées dans la littérature (gradient conjugué non linéaire, quasi-Newton, Newton, Levenberg-Marquardt (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)) …). 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-a]).

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

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



a b c

Figure 2.3-a : 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).

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

Initialisation $\mathbf{u}^0$ donné, $\mathbf{r}^0 = -\nabla J^0, \mathbf{d}^0 = \mathbf{r}^0$

Boucle en i

(1) Trouver $\alpha^i = \underset{\alpha\in[\alpha_m,\alpha_M]}{Arg\min} J\left(\mathbf{u}^i + \alpha\,\mathbf{d}^i\right)$ (paramètre de descente)

(2) $\mathbf{u}^{i+1} = \mathbf{u}^i + \alpha^i \mathbf{d}^i$ (nouvel itéré)

(3) $\mathbf{r}^{i+1} = -\nabla J^{i+1}$ (nouveau gradient)

(4) Test d'arrêt via $J^{i+1} - J^i$, $\left\|\mathbf{d}^i\right\|$ ou $\left\|\mathbf{r}^{i+1}\right\|$ (par exemple)

(5) $\beta^{i+1} = \begin{cases} \dfrac{\left\|\mathbf{r}^{i+1}\right\|^2}{\left\|\mathbf{r}^i\right\|^2} \quad (\text{FR}) \\ \dfrac{\left\langle \mathbf{r}^{i+1}, \mathbf{r}^{i+1} - \mathbf{r}^i \right\rangle}{\left\|\mathbf{r}^i\right\|^2} \quad (\text{PR}) \end{cases}$ (paramètre de conjugaison)

(6) $\mathbf{d}^{i+1} = \mathbf{r}^{i+1} + \beta^{i+1}\mathbf{d}^i$ (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 [éq 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 [éq 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 [bib28], J.C. Culioli [bib11] ou J.F. Bonnans et al [bib5].

3 Le gradient conjugué (GC)

3.1 Description générale

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 » (les méthodes de Fletcher-Reeves et de Polak-Ribière (cf. [§2.3]) font aussi partie de cette famille de méthodes). Celles-ci préconisent de construire progressivement des directions de descentes $\mathbf{d}^0$, $\mathbf{d}^1$, $\mathbf{d}^2$ … 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 $\nabla J^i = -\mathbf{r}^i$ et celles des précédentes directions $\mathbf{d}^0, \mathbf{d}^1 \ldots \mathbf{d}^{i-1}$.

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

L'astuce consiste à choisir une indépendance vectorielle de type $\mathbf{K}$-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-a])

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

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.

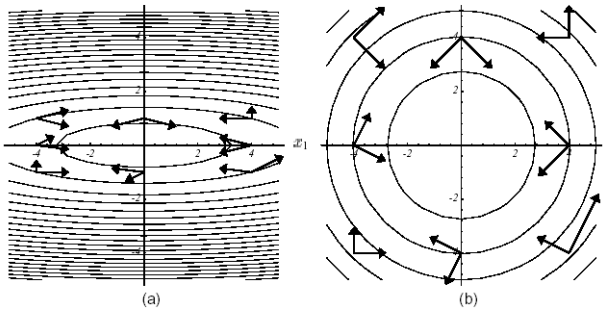


Figure 3.1-a : Exemple de paires de vecteurs K-orthogonaux en 2D: conditionnement de K quelconque (a), conditionnement parfait (i.e. égal à l'unité) = orthogonalité usuelle (b).

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

$$\mathbf{d}^i := \mathbf{r}^i + \beta^i \mathbf{d}^{i-1} \qquad \textbf{éq 3.1-3}$$

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

$$\frac{\left\langle \mathbf{r}^i, \mathbf{d}^i \right\rangle^2}{\left\langle \mathbf{K}^{-1}\mathbf{r}^i, \mathbf{r}^i \right\rangle \left\langle \mathbf{K}\mathbf{d}^i, \mathbf{d}^i \right\rangle} \qquad \textbf{éq 3.1-4}$$

Il conduit à l'expression

$$\beta^i := \frac{\left\|\mathbf{r}^i\right\|^2}{\left\|\mathbf{r}^{i-1}\right\|^2} \qquad \textbf{éq 3.1-5}$$

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

$$\left\langle \mathbf{r}^i, \mathbf{r}^{i-1} \right\rangle = 0 \qquad \textbf{éq 3.1-6}$$

Se rajoute une condition « résidu-dd »

$$\left\langle \mathbf{r}^i, \mathbf{d}^i \right\rangle = \left\|\mathbf{r}^i\right\|^2 \qquad \textbf{éq 3.1-7}$$

qui impose d'initialiser le processus via $\mathbf{d}^0 = \mathbf{r}^0$.

Algorithme

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

Initialisation $\mathbf{u}^0$ donné, $\mathbf{r}^0 = \mathbf{f} - \mathbf{K}\mathbf{u}^0, \mathbf{d}^0 = \mathbf{r}^0$

Boucle en i

(1) $\mathbf{z}^i = \mathbf{K}\mathbf{d}^i$

(2) $\alpha^i = \dfrac{\left\|\mathbf{r}^i\right\|^2}{\left\langle \mathbf{d}^i, \mathbf{z}^i \right\rangle}$ (paramètre optimal de descente)

(3) $\mathbf{u}^{i+1} = \mathbf{u}^i + \alpha^i \mathbf{d}^i$ (nouvel itéré)

(4) $\mathbf{r}^{i+1} = \mathbf{r}^i - \alpha^i \mathbf{z}^i$ (nouveau résidu)

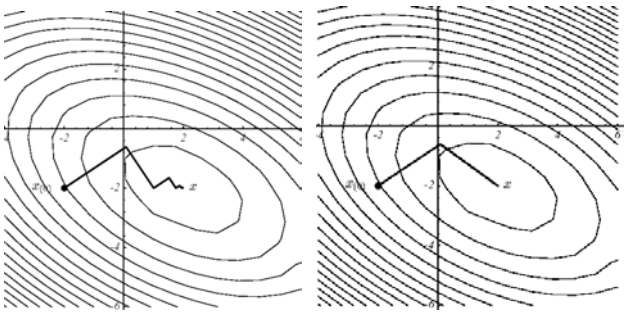
(5) Test d'arrêt via $\left\langle \mathbf{r}^{i+1}, \mathbf{r}^{i+1} \right\rangle$ (par exemple)

(6) $\beta^{i+1} = \dfrac{\left\|\mathbf{r}^{i+1}\right\|^2}{\left\|\mathbf{r}^i\right\|^2}$ (paramètre de conjugaison optimal)

(7) $\mathbf{d}^{i+1} = \mathbf{r}^{i+1} + \beta^{i+1} \mathbf{d}^i$ (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. [Figures 3.1-b]) et s'explique par les résultats de convergence [éq 2.2-9] et [éq 3.2-9]. Dans les deux cas, les mêmes points de départ et les mêmes critères d'arrêt ont été choisis : $\mathbf{u}^0$=[-2,-2]^T et $\left\|\mathbf{r}^i\right\|^2 < \varepsilon = 10^{-6}$.



a b

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

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 [bib15], [bib16], [bib36].*
- *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 [bib10]. 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 [bib18] [bib21]. 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 et parallélisé [bib19], [bib20], [bib22], [bib23].*

  Remarque :

  Etant très dépendant du conditionnement des matrices générées par ces domaines de la physique, il est généralement plus usité en mécanique des fluides ou en électromagnétisme qu'en thermo-mécanique des structures. D'où la suprématie, dans ce dernier champ d'activités, des solveurs directs… le Code_Aster avec sa multifrontale [bib33] n'échappant pas à la règle. On y cumule souvent forte hétérogénéités de matériau, de chargement et de taille de maille, non-linéarités et des jonctions éléments isoparamétriques/éléments structuraux qui gangrènent dramatiquement ce conditionnement [bib31], [bib13].

- *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 : $\delta^i := \dfrac{\left\|\mathbf{r}^i\right\|}{\left\|\mathbf{f}\right\|}$ (cf. [§5.2]).*

3.2 Eléments de théorie

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 $\kappa_i\left(\mathbf{K},\mathbf{r}^0\right):=vect\left(\mathbf{r}^0,\mathbf{K}\mathbf{r}^0\ldots\mathbf{K}^{i-1}\mathbf{r}^0\right)$ où $\mathbf{r}^0$ est le résidu initial : à la $i^{\text{ème}}$ itération $\mathcal{K}_i=\mathcal{L}_i=\kappa_i\left(\mathbf{K},\mathbf{r}^0\right)$. On résout le système linéaire (P_1) en recherchant une solution approchée $\mathbf{u}^i$ dans le sous-espace affine (espace de recherche de dimension N)

$$\mathcal{A}=\mathbf{r}^0+\kappa_i\left(\mathbf{K},\mathbf{r}^0\right) \qquad \textbf{éq 3.2-1}$$

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

$$\mathbf{r}^i:=\mathbf{f}-\mathbf{K}\mathbf{u}^i\perp\kappa_i\left(\mathbf{K},\mathbf{r}^0\right) \qquad \textbf{éq 3.2-2}$$

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

$$\mathbf{K}^{-1}\mathbf{f}\approx\mathbf{u}^m=\mathbf{r}^0+P_{m-1}\left(\mathbf{K}\right)\mathbf{f} \qquad \textbf{éq 3.2-3}$$

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

$$\begin{aligned} vect\left(\mathbf{r}^0,\mathbf{r}^1\ldots\mathbf{r}^{m-1}\right)&=\kappa_m\left(\mathbf{K},\mathbf{r}^0\right)\\ vect\left(\mathbf{d}^0,\mathbf{d}^1\ldots\mathbf{d}^{m-1}\right)&=\kappa_m\left(\mathbf{K},\mathbf{r}^0\right)\end{aligned} \qquad \textbf{éq 3.2-4}$$

tout en permettant à la solution approchée, $\mathbf{u}^m$, de minimiser la norme en énergie sur tout l'espace affine $\mathcal{A}$

$$\left\|\mathbf{u}^m\right\|_{\mathbf{K}}\leq\left\|\mathbf{u}\right\|_{\mathbf{K}}\quad\forall\mathbf{u}\in\mathcal{A} \qquad \textbf{éq 3.2-5}$$

Ce résultat joint à la propriété [éq 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[bib35], [bib27] (GC, GMRES, FOM/IOM/DOM, GCR, ORTHODIR/MIN…) et/ou des problèmes modaux [bib2], [bib39] (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…).

Orthogonalité

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

$$\left\langle \mathbf{d}^i, \mathbf{r}^m \right\rangle = 0 \quad \forall i < m$$
$$\left\langle \mathbf{r}^i, \mathbf{r}^m \right\rangle = 0 \qquad \textbf{éq 3.2-6}$$

On constate donc une petite approximation dans l'appellation du GC, car les gradients ne sont pas conjugués (cf. [éq 3.2-6b]) et les directions conjugués ne comportent pas que des gradients (cf. [éq 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 $\mathbf{r}^N=\mathbf{0} \Rightarrow$ 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 $\mathbf{r}^N=\mathbf{0} \Rightarrow$ 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 vrai 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 (cf. [bib2] [§5.3.1] et annexe 2) et en décomposition de domaine (cf. [bib4] [§5.2.6]) peut se décliner sous différentes variantes : réorthogonalisation totale, partielle, sélective … via toute une panoplie de procédures d'orthogonalisation : GS, GSM, IGSM, Househölder, Givens… Pour les méthodes de type Krylov, afin de ménager l'occupation mémoire et par soucis de robustesse, on préconise de ne réorthogonaliser systématiquement que par rapport aux premières directions qui engendrent le sous-espace invariant associé aux plus grandes valeurs propres (en traitement du signal, cet espace définirait ce qu'on appel la structure cohérente du champ de déplacement solution, i.e. sa substantifique moelle).
Ces réorthogonalisations requièrent le stockage des N_{orth} premiers vecteurs $\mathbf{d}^k \left(k=1\ldots N_{orth}\right)$ et de leur produit par l'opérateur de travail $\mathbf{z}^k = \mathbf{K}\mathbf{d}^k \left(k=1\ldots N_{orth}\right)$. Formellement, il s'agit donc de substituer aux deux dernières étapes de l'algorithme le calcul suivant

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

Au travers des différentes expériences numériques qui ont été menées (cf. notamment travaux de J.P. Gregoire et [bib3]), il semble que cette réorthogonalisation ne soit pas toujours efficace. Son surcoût dû principalement aux nouveaux produits matrice-vecteur $\mathbf{K}\mathbf{d}^k$ (et à leur stockage) n'est pas toujours compensé par le gain en nombre d'itérations globales. Contrairement à ce qui est généralement observé en décomposition de domaine (méthode itérative de Schur primal et FETI) où la mise en place de ce processus est préconisée. Il est vrai qu'il intervient alors sur un problème d'interface beaucoup plus petit et mieux conditionné que le problème global.

Convergence

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

$$\left\|e\left(\mathbf{u}^i\right)\right\|_{\mathbf{K}}^2 = \left(\omega^i\right)^2 \left\|e\left(\mathbf{u}^0\right)\right\|_{\mathbf{K}}^2$$
$$\text{avec } \omega^i := \max_{1\le i\le N}\left(1-\lambda_i P_{m-1}\left(\lambda_i\right)\right) \qquad \textbf{éq 3.2-8}$$

où l'on note $(\lambda_i ;\mathbf{v}_i)$ les modes propres de la matrice $\mathbf{K}$ 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

$$\left\|e\left(\mathbf{u}^i\right)\right\|_{\mathbf{K}} \le 2\left(\frac{\sqrt{\eta(\mathbf{K})}-1}{\sqrt{\eta(\mathbf{K})}+1}\right)^i \left\|e\left(\mathbf{u}^0\right)\right\|_{\mathbf{K}} \qquad \textbf{éq 3.2-9}$$

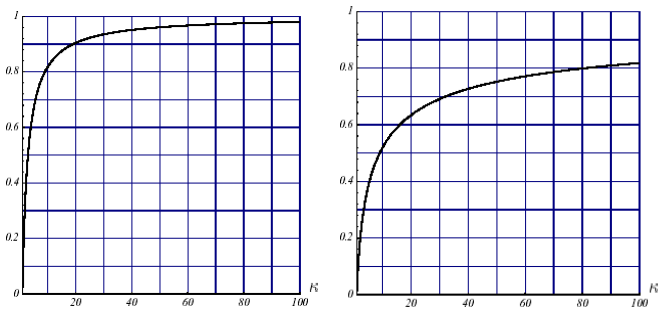
Elle assure la convergence superlinéaire (c'est-à-dire $\lim_{i\to\infty}\frac{J\left(\mathbf{u}^{i+1}\right)-J\left(\mathbf{u}\right)}{J\left(\mathbf{u}^{i}\right)-J\left(\mathbf{u}\right)}=0$) du processus en un nombre d'itérations proportionnel à la racine carré du conditionnement de l'opérateur.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

$$\frac{\left\|e\left(\mathbf{u}^i\right)\right\|_{\mathbf{K}}}{\left\|e\left(\mathbf{u}^0\right)\right\|_{\mathbf{K}}} \le \varepsilon \ \ (\text{petit}) \qquad \textbf{éq 3.2-10}$$

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

$$i \approx \frac{\sqrt{\eta(\mathbf{K})}}{2}\ln\frac{2}{\varepsilon} \qquad \textbf{éq 3.2-11}$$

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-a]). Et de toute façon, la convergence globale du premier sera meilleure.



(a) (b)

Figure 3.2-a : 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 [éq 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.*

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 $\vartheta(kcN)$ où c est le nombre moyen de termes non nuls par ligne de $\mathbf{K}$ et k le nombre d'itérations requises à convergence. Pour être beaucoup plus efficace qu'un simple Cholesky (de complexité $\vartheta\left(\frac{N^3}{3}\right)$) il faut donc :

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

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. [éq 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 $\vartheta\left(cN^{\frac{1}{d}+1}\right)$ (resp. $\vartheta\left(cN^{\frac{2}{d}+1}\right)$).

Pour ce qui est de l'occupation mémoire, comme pour la Steepest Descent, seul le stockage de la matrice de travail est éventuellement requis : $\vartheta(cN)$. 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 $\vartheta(cN)$ réels et $\vartheta(cN+N)$ 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

Equivalence avec la méthode de Lanczos

En « triturant » la propriété d'orthogonalité du résidu avec tout vecteur de l'espace de Krylov (cf. [éq 3.2-2]), on montre que les m itérations du GC conduisent à la construction d'une factorisée de Lanczos du même ordre (cf. [bib2] [§5.2])

$$\mathbf{KQ}^m = \mathbf{Q}^m\mathbf{T}^m - \underbrace{\frac{\sqrt{\beta^m}}{\alpha^{m-1}}\mathbf{q}^m\mathbf{e}_m^T}_{\mathbf{R}^m} \qquad \textbf{éq 3.3-1}$$

en notant :

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

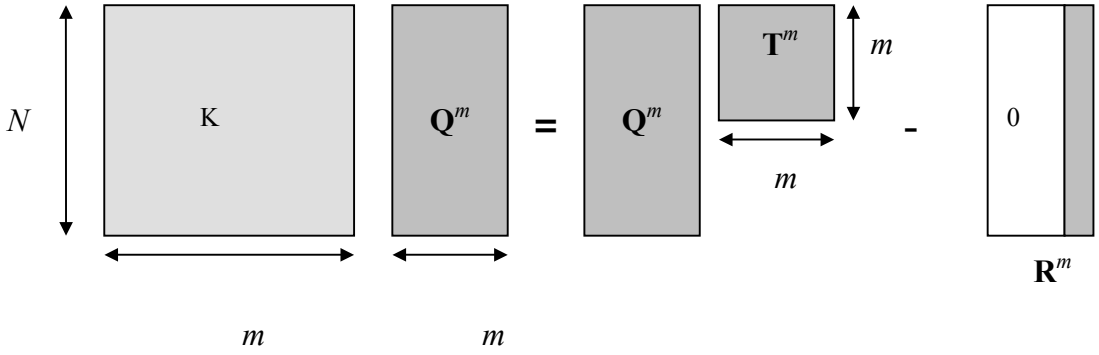


Figure 3.3-a : Factorisation de Lanczos induite par le GC.

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

$$\mathbf{T}^m = \begin{bmatrix} \frac{1}{\alpha^0} & -\frac{\sqrt{\beta^1}}{\alpha^0} & 0 & 0 \\ -\frac{\sqrt{\beta^1}}{\alpha^0} & \frac{1}{\alpha^1}+\frac{\beta^1}{\alpha^0} & \dots & 0 \\ 0 & \dots & \dots & -\frac{\sqrt{\beta^{m-1}}}{\alpha^{m-2}} \\ 0 & 0 & -\frac{\sqrt{\beta^{m-1}}}{\alpha^{m-2}} & \frac{1}{\alpha^{m-1}}+\frac{\beta^{m-1}}{\alpha^{m-2}} \end{bmatrix} \qquad \textbf{éq 3.3-2}$$

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 $\mathbf{QR}$, cf. [bib2] 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 [bib24]. Cette fonctionnalité devrait bientôt être insérée dans *Code_Aster* de manière à 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* $\mathbf{T}^m = \mathbf{L}^m \mathbf{D}^m \left(\mathbf{L}^m\right)^T$ *avec*

$$\mathbf{L}^m := \begin{bmatrix} 1 & 0 & 0 & 0 \\ -\sqrt{\beta^1} & 1 & \dots & 0 \\ 0 & \dots & \dots & 0 \\ 0 & 0 & -\sqrt{\beta^{m-1}} & 1 \end{bmatrix} \text{ et } \mathbf{D}^m := \begin{bmatrix} \frac{1}{\alpha^0} & 0 & 0 & 0 \\ 0 & \frac{1}{\alpha^1} & \dots & 0 \\ 0 & \dots & \dots & 0 \\ 0 & 0 & 0 & \frac{1}{\alpha^{m-1}} \end{bmatrix} \qquad \textbf{éq 3.3-3}$$

  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 [bib35].
- *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 (*$m>>N$ *!), on est quand même capable de détecter tout le spectre : c'est ce qu'on appelle le phénomène de Lanczos (cf. [bib2] [§5.3.1], [§5.3.3] ou [bib39]).*

  Remarque :

  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.

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) [bib12], [bib17], [bib8], [bib9] 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 [bib8]) ou, au contraire en la durcissant (resp. méthode modale de type puissance + Gauss-Seidel + GC [bib9]), 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 devrait bientôt 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.

Parallélisme

Le GC comme beaucoup de solveurs itératifs se prête bien au parallélisme. Il est scalable, c'est à dire que si un calcul à N ddls fonctionne sur P processeurs, son pendant à αN doit fonctionner sur αP. Les principales opérations élémentaires qui le constitue (produit matrice-vecteur et produit scalaire) se décomposent efficacement entre différents processeurs, seule la parallélisation du préconditionneur (souvent basé sur un solveur direct) peut s'avérer hasardeuse (problèmes souvent dénommés : 'parallel preconditioning').
Ainsi sur machine CRAY, J.P.Grégoire a adapté et parallélisé, en mémoire partagée [bib20] et en mémoire distribuée [bib23], 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*.

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

4 Le gradient conjugué préconditionné (GCPC)

4.1 Description générale

Principe

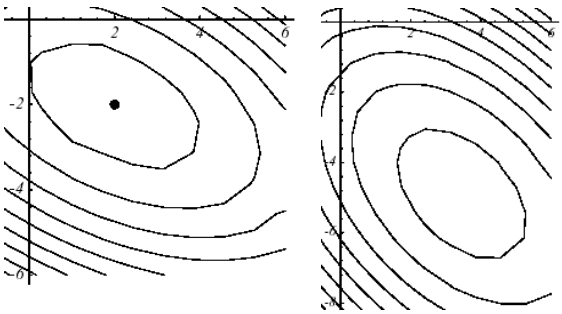
Comme on a pu le constater (et le marteler !) dans les paragraphes précédents, la rapidité de convergence des méthodes de descente, et notamment celle du gradient conjugué, dépend du conditionnement de la matrice $\eta(\mathbf{K})$. Plus il est proche de sa valeur plancher, 1, meilleure est la convergence.
Le principe de préconditionnement est alors « simple », il consiste à remplacer le système linéaire du problème (P_1) par un système équivalent du type

$$\left(\widetilde{P}_1\right) \quad \underbrace{\mathbf{M}^{-1}\mathbf{K}}_{\widetilde{\mathbf{K}}}\mathbf{u} = \underbrace{\mathbf{M}^{-1}\mathbf{f}}_{\widetilde{\mathbf{f}}} \qquad \textbf{éq 4.1-1}$$

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) : $\eta(\widetilde{\mathbf{K}}) << \eta(\mathbf{K})$.
- Tout comme la distribution spectrale : valeurs propres plus tassées.
- $\mathbf{M}^{-1}$ soit peu coûteux à évaluer (comme pour l'opérateur initial, on a souvent juste besoin de connaître l'action du préconditionneur sur un vecteur) : $\mathbf{Mv} = \mathbf{u}$ 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 $\widetilde{\mathbf{K}}$ les mêmes propriétés que celle initiale : ici, le caractère SPD.

En théorie, le meilleur choix serait $\mathbf{M}^{-1}=\mathbf{K}^{-1}$ car alors $\eta(\widetilde{\mathbf{K}} = \mathbf{I}_N) = 1$, 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-a]), les effets sont notables.



(a) (b)

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

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 $\left(\widetilde{P}_1\right)$: même si $\mathbf{M}^{-1}$ et $\mathbf{K}$ sont SPD, ce n'est pas forcément le cas de leur produit.

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

$$\left(\hat{P}_1\right) \quad \underbrace{\mathbf{M}^{-1/2}\mathbf{K}\mathbf{M}^{-1/2}}_{\hat{\mathbf{K}}}\underbrace{\mathbf{M}^{1/2}\mathbf{u}}_{\hat{\mathbf{u}}}=\underbrace{\mathbf{M}^{-1/2}\mathbf{f}}_{\hat{\mathbf{f}}} \qquad \textbf{éq 4.1-2}$$

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

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 $\mathbf{K}$, $\mathbf{u}$ et $\mathbf{f}$, il advient le déroulement suivant.

Initialisation $\mathbf{u}^0$ donné, $\mathbf{r}^0=\mathbf{f}-\mathbf{K}\mathbf{u}^0$, $\mathbf{d}^0=\mathbf{M}^{-1}\mathbf{r}^0$, $\mathbf{g}^0=\mathbf{d}^0$

Boucle en i

(1) $\mathbf{z}^i=\mathbf{K}\mathbf{d}^i$

(2) $\alpha^i=\dfrac{\left\langle \mathbf{r}^i,\mathbf{g}^i\right\rangle}{\left\langle \mathbf{d}^i,\mathbf{z}^i\right\rangle}$ (paramètre optimal de descente)

(3) $\mathbf{u}^{i+1}=\mathbf{u}^i+\alpha^i\mathbf{d}^i$ (nouvel itéré)

(4) $\mathbf{r}^{i+1}=\mathbf{r}^i-\alpha^i\mathbf{z}^i$ (nouveau résidu)

(5) Test d'arrêt via $\left\langle \mathbf{r}^{i+1},\mathbf{r}^{i+1}\right\rangle$ (par exemple)

(6) $\mathbf{g}^{i+1}=\mathbf{M}^{-1}\mathbf{r}^{i+1}$ (résidu préconditionné)

(7) $\beta^{i+1}=\dfrac{\left\langle \mathbf{r}^{i+1},\mathbf{g}^{i+1}\right\rangle}{\left\langle \mathbf{r}^i,\mathbf{g}^i\right\rangle}$ (paramètre de conjugaison optimal)

(8) $\mathbf{d}^{i+1}=\mathbf{g}^{i+1}+\beta^{i+1}\mathbf{d}^i$ (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 $\mathbf{K}$, $\mathbf{M}$ ou $\mathbf{M}^{-1}$, 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, $\mathbf{M}^{-1}\mathbf{K}$ étant symétrique par rapport au $\mathbf{M}$-produit scalaire, on peut substituer ce nouvel opérateur de travail et ce nouveau produit scalaire dans l'algorithme du GC non préconditionné (algorithme 4)

$$\mathbf{K} \Leftarrow \mathbf{M}^{-1}\mathbf{K}$$

$$\langle , \rangle \Leftarrow \langle , \rangle_{\mathbf{M}}$$

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, $\mathbf{K}\mathbf{M}^{-1}$, via un $\mathbf{M}^{-1}$-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.*

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

$$\mathbf{M}_J := \mathbf{diag}\left(\mathbf{K}_{ii}\right) \qquad \textbf{éq 4.1-3}$$

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

Remarques :

- *C'est la solution souvent retenue en mécanique des fluides[Gre97] (N3S, Code_Saturne), où le processus d'Uzawa produit naturellement un problème bien conditionné, et qui a fait la renommée du JCG, transformé pour l'occasion en véritable « bête de course » dédié à de gros calculateurs : vectorisation, parallélisation, gestion des caches et des mémoires…*
- *C'est une solution proposée, en mécanique des structures, par bon nombres de codes commerciaux : ANSYS, Zébulon, NASTRAN (cf. [bib4] [§8.3]). 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 Succesive Over Relaxation). Comme le précédent, il est déduit d'une méthode itérative « ancestrale » : la méthode de Gauss-Seidel relaxée. En décomposant l'opérateur initial sous la forme habituelle $\mathbf{K} := \mathbf{D} + \mathbf{L} + \mathbf{L}^T$ où $\mathbf{D}$ est sa diagonale et $\mathbf{L}$ sa partie triangulaire strictement inférieure, il s'écrit (en fonction du paramètre de relaxation ω)

$$\mathbf{M}_{SSOR}(\omega) := \frac{1}{\omega(\omega-2)}(\mathbf{D}+\omega\mathbf{L})\mathbf{D}^{-1}(\mathbf{D}+\omega\mathbf{L}^T) \quad 0<\omega<2 \qquad \textbf{éq 4.1-4}$$

Il présente l'avantage de ne pas requérir de stockage mémoire et de surcoût calcul, puisqu'il est directement élaboré à partir de $\mathbf{K}$, 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

$$\eta(\mathbf{K}) := \vartheta\left(\frac{1}{h^2}\right) \Rightarrow \eta(\mathbf{M}^{-1}\mathbf{K}) := \vartheta\left(\frac{1}{h}\right) \qquad \textbf{éq 4.1-5}$$

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 ω=1 on retrouve le cas particulier de Gauss-Seidel Symétrique*

$$\mathbf{M}_{SGS}(\omega) := -(\mathbf{D}+\mathbf{L})\mathbf{D}^{-1}(\mathbf{D}+\mathbf{L}^T) \qquad \textbf{éq 4.1-6}$$

Une kyrielle d'autres préconditionneurs a ainsi vu le jour depuis une trentaine d'années dans la littérature : surtout polynomiaux [bib7], [bib34] mais aussi ACP, ADI, ADDKR, par éléments…. 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 [bib27] ou les livres de Y.Saad [bib35] et H.A.Van der Vorst [bib40].

4.2 Factorisation incomplète de Cholesky

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 son « frère ennemi », 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 $\mathbf{K} = \mathbf{C}\mathbf{C}^T$ où $\mathbf{C}$ est une matrice triangulaire inférieure. On appelle factorisation incomplète de Cholesky, la recherche d'une matrice $\mathbf{F}$ triangulaire inférieure aussi creuse que possible et telle que $\mathbf{F}\mathbf{F}^T$ soit proche de $\mathbf{K}$ dans un sens à définir. Par exemple, en posant $\mathbf{B} = \mathbf{K} - \mathbf{F}\mathbf{F}^T$ on va demander que l'erreur relative (exprimée dans une norme matricielle au choix)

$$\Delta := \frac{\|\mathbf{B}\|}{\|\mathbf{K}\|} \qquad \textbf{éq 4.2-1}$$

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 [bib27] 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 $\mathbf{F}$, c'est-à-dire son graphe

$$\Xi(\mathbf{F}) := \{(i,j),\ 1 \le j \le i-1,\ 1 \le i \le N,\quad \mathbf{F}_{ij} \neq 0\} \qquad \textbf{éq 4.2-2}$$

Il s'agit évidemment de trouver un compromis : plus ce graphe sera étendu et plus l'erreur [éq 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 $\mathbf{F}$ la même structure que celle de $\mathbf{C}$: $\Xi(\mathbf{F}) = \Xi(\mathbf{C})$.

Remarques :

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

$$\mathbf{F}\mathbf{F}^T\mathbf{u}^{i+1} = \mathbf{f} - \mathbf{B}\mathbf{u}^i \qquad \textbf{éq 4.2-3}$$

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

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 [bib40]. Les matrices restant toutefois symétriques, on peut écrire $\mathbf{K} = \mathbf{L}\mathbf{D}\mathbf{L}^T$ et $\mathbf{B} = \mathbf{K} - \mathbf{L}\mathbf{D}\mathbf{L}^T$.

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 aucun solveurs linéaires du Code_Aster et que, de toute façon, le stockage MORSE des matrices du GCPC interdit. 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.

Remplissage par niveaux

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

$$\mathbf{L}_{ij} = \frac{1}{\mathbf{D}_j}\left(\mathbf{K}_{ij} - \sum_{k=1}^{j-1} \mathbf{L}_{ik}\mathbf{D}_k\mathbf{L}_{jk} \right) \qquad \textbf{éq 4.2-4}$$

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

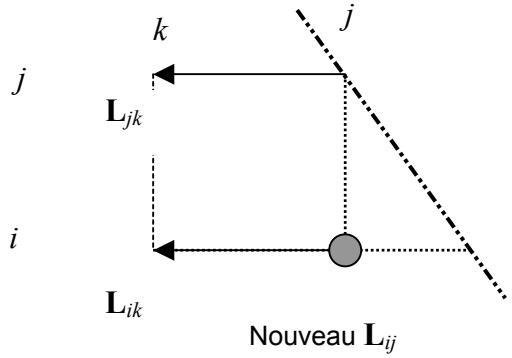


Figure 4.2-a : Phénomène de remplissage lors de la factorisation.

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 $\mathbf{L}(0)$) si elle reproduit à l'identique la structure (mais bien sûr pas les valeurs qui sont différentes) de la partie diagonale inférieure stricte de $\mathbf{K}$ (i.e. le même graphe). La factorisée de niveau 1 (resp. $\mathbf{L}(1)$) pourra elle inclure le remplissage conduit par des termes non nuls de $\mathbf{K}$ (termes notés r^1 dans la [Figure 4.2-b]), celle de niveau 2 (resp. $\mathbf{L}(2)$) 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 $\mathbf{L}(3)$ conduit ici à un remplissage complet, l'erreur relative sera alors nulle $\Delta = 0$ et le GCPC converge en une itération. L'intérêt de l'exercice est bien sûr purement didactique !

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

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

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

Faible magnitude des termes résultant 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 $\mathbf{K}$. 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-c])

$$y(k) := \log_{10}\left(\sqrt{\frac{\sum_{i=1}^{N-k} \mathbf{L}_{k+i,i}^2}{N-k}}\right) \qquad \textbf{éq 4.2-5}$$

qui représente les ordres de grandeurs des termes de la $k^{\text{ième}}$ sous-diagonales. Par exemple $y(0)$ correspond à la diagonale principale.

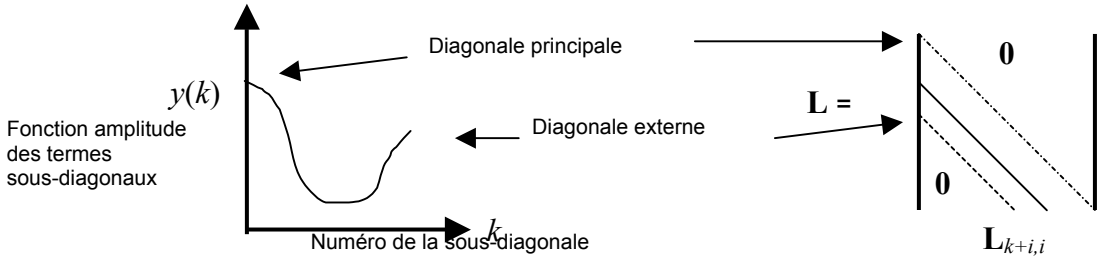


Figure 4.2-c : Importance relative des diagonales de L

Si on récapitule, on a donc :

- un niveau de remplissage récursif et paramétrable,
- une moindre importance des termes correspondant à des niveaux de remplissage élévés.

D'où un certain « blanc-seing » laissé aux factorisations incomplètes ILU(p) négligeant ces termes médians. Cette approximation « à bon compte » de $\mathbf{K}^{-1}$ servira alors de préconditionneur dans l'algorithme 5 du GCPC ($\mathbf{M}^{-1}=\mathbf{K}^{-1}$). 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 [bib22].*
- *C'est une solution proposée, en mécanique des structures, par bon nombre de grands codes : ANSYS, Zébulon, NASTRAN, Code_Aster (cf. [bib4] [§8.3]).*

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 $\vartheta\left(\frac{N^3}{3}\right)$ 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 $\vartheta(2cN)$ réels et $\vartheta(2cN+2N)$ 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 $\vartheta(\alpha cN)$ réels et $\vartheta(\alpha cN+2N)$ entiers avec

- α=2,5 en ILU(0) (niveau par défaut dans *Code_Aster*),
- α=4,5 en ILU(1),
- α=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 [bib22] ou [bib3].

Remarque :

Dans Code_Aster, l'encombrement mémoire d'un entier est identique à celui d'un réel et, pour le GCPC, aucune réservation de place ou pagination mémoire n'ont pu être mises en place. Ce qui n'arrange rien !

5 Implantation dans *Code_Aster*

5.1 Difficultés particulières

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 [bib30] (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)

$$\mathbf{Tu} = \mathbf{0} \qquad \textbf{éq 5.1-1}$$

avec $\mathbf{T}$ matrice réelle de taille p x n. Cette technique étoffe la matrice de rigidité en une nouvelle matrice, dite « dualisée », qui devient la nouvelle matrice de travail

$$\widehat{\mathbf{K}} = \begin{pmatrix} \mathbf{K} & \beta\mathbf{T}^T & \beta\mathbf{T}^T \\ \beta\mathbf{T} & -\alpha\mathbf{Id} & \alpha\mathbf{Id} \\ \beta\mathbf{T} & \alpha\mathbf{Id} & -\alpha\mathbf{Id} \end{pmatrix} \qquad \textbf{éq 5.1-2}$$

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

$$\widehat{\mathbf{K}} = \begin{pmatrix} \overline{\mathbf{K}} & 0 \\ 0 & \mathbf{Id} \end{pmatrix} \qquad \textbf{éq 5.1-3}$$

en notant $\overline{\mathbf{K}}$ 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.
Dans *Code_Aster*, la primauté étant donnée à la facilité d'utilisation ainsi qu'aux contingences numérico-informatiques de robustesse, d'implantation, de maintenance et d'évolutivité, c'est clairement la première approche qui a été privilégiée.

Remarque :

D'autres approches étaient envisageables : simple dualisation, prise en compte des conditions limites dans la formulation variationnelle, gradient conjugué projeté (GCP cf. [bib4] [§6.2.3])…

Conséquence sur le GCPC

Avec AFFE_CHAR_CINE, l'opérateur de travail $\widetilde{\mathbf{K}}$ 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. [éq 3.2-9]) n'est plus garantie,
- Lorsqu'elle se produit, elle est ralentie (cf. [éq 3.2-11]) par un conditionnement dégradé $\eta(\mathbf{K}) << \eta(\widehat{\mathbf{K}})$,
- 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 $\mathbf{LDL}^T$ 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 ddls cf. [bib30] [§4]), permet d'obtenir sans coup férir cette factorisation incomplète. B.Nitrosso [bib29] a ainsi montré qu'elle est stable et que, d'autre part, utilisée en tant que préconditionneur, elle rétablit la convergence globale du GCPC (ouf !).

Remarques :

- *Dans ce même rapport, B.Nitrosso 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 N3S et 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 [bib31], ont déjà fait ce constat.*

Encombrement mémoire

Compte tenu des éléments du [§4.2] et du fait que dans *Code_Aster*, un entier ait le même encombrement mémoire qu'un réel (8 octets), la complexité mémoire effective du GCPC est d'au moins α la taille de $\mathbf{K}$ avec

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

D'autre part, contrairement aux solveurs directs, le GCPC n'a pu bénéficier d'une pagination car, sur ce point, son algorithmique le dessert : tous les blocs matriciels sont utilisés un grand nombre de fois, en fait à chaque itération, via le produit matrice-vecteur.
De plus, contrairement a ce qui a été fait dans N3S, à la définition du cas de calcul (maillage, matériaux, chargements et conditions limites…), on ne réserve pas, par avance, une partie de la mémoire pour y stocker les vecteurs requis par le GCPC. Donc, au fur et à mesure de l'exécution des opérateurs du fichier de commande, la mémoire devient un « vrai gruyère » où il devient difficile d'insérer de gros objets.
La pagination des solveurs directs leur permet de franchir sans trop de difficultés cet écueil… il en est bien sûr autrement du GCPC. D'où, suivant l'enchaînement des commandes, une inflation des tailles mémoires requises pour faire fonctionner son « job » avec METHODE='GCPC'. Il est donc conseillé de travailler en mode 'POURSUITE' et de contingenter la résolution du système linéaire en tête des fichiers de commande : la mémoire sera ainsi au maximum préservée.
Si le dépassement mémoire est modeste, on peut aussi tenter de contingenter ce phénomène d'émiettement en modifiant le paramétrage de la gestion mémoire via le paramétrage (mots-clé facteur MEMOIRE, mots-clés GESTION, TYPE_ALLOCATION ou PARTITION) de la commande DEBUT [U4.11.01].

Remarquons tout de même que le stockage MORSE (Il est aussi plébiscité par la multifrontale. Le solveur de Gauss utilise, quant à lui, un stockage plus « fruste », SKYLINE (« ligne de ciel » en bon français !), qui permet une gestion plus aisée du remplissage mais amplifie l'occupation mémoire) des matrices et des vecteurs choisi pour le GCPC permet de gérer au mieux le caractère creux des opérateurs et donc, limite l'encombrement mémoire.

Parallélisation

Contrairement à la multifrontale, le GCPC n'a pas été parallélisé. Ce travail a pourtant déjà été réalisé [bib20], [bib23], mais uniquement pour un préconditionnement diagonal (cf. [§4.1-3]) car il pose des problèmes de construction parallèle efficace du préconditionneur afin de ne pas perturber les autres étapes.
A ces problèmes de 'parallel preconditioning' se rajoutent des contingences informatiques fortes en mémoires disrtribuées. Pour plus d'informations et des références sur ce parallélisme dit « numérique » et ses réminicences dans *Code_Aster*, on pourra consulter [bib4] [§3].

5.2 Périmètre d'utilisation

Liste des commandes *Code_Aster* pouvant utiliser GCPC :

- CALC_FORC_AJOU,
- CALC_MATR_AJOU,
- CALC_PRECONT,
- DYNA_NON_LINE ▸ Approximation symétrisée possible (cf. [§5.3]),
- DYNA_TRAN_EXPLI ▸ Approximation symétrisée possible,
- MACR_ASCOUF_CALC ▸ Approximation symétrisée possible,
- MACR_ASPIC_CALC ▸ Approximation symétrisée possible,
- MACRO_MATR_AJOU,
- MACRO_MATR_ASSE,
- MECA_STATIQUE,
- NUME_DDL,
- 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 $\mathbf{Ku}=\mathbf{f}$ en un nouveau problème symétrisé $\frac{1}{2}\left(\mathbf{K}+\mathbf{K}^T\right)\mathbf{u}=\mathbf{f}$. On suppose par là que le solveur non linéaire englobant (algorithme de Newton) va compenser l'approximation de l'opérateur initial par $\widetilde{\mathbf{K}}:=\frac{1}{2}\left(\mathbf{K}+\mathbf{K}^T\right)$. 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 les trois solveurs linéaires du code : 'LDLT', 'MULT_FRONT' et 'GCPC'.

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 actuellement dans *Code_Aster* est illicite (il faudrait faire appel à certaines de ses variantes : GMRES, Orthomin, double GC, équation normale…) et il faut inverser ce système non symétrique via LDLT' ou 'MULT_FRONT'. 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].
Seul le préconditionnement de type Cholesky incomplet est disponible et il n'est pas possible de s'en dispenser : NIVE_REMPLISSAGE=0 ne signifie donc pas « sans préconditionnement » mais préconditionnement de type ILU(0).
Pour minimiser la taille du profil de la matrice de travail et de son préconditionneur, un algorithme de renumérotation est disponible par défaut : 'RCMK' pour 'Reverse Cuthill Mac-Kee' (cf. [bib26] [§5.4.2]). Il est néanmoins désactivable.
En « jouant » sur le niveau de complétude du préconditionnement, on peut modifier le compromis « temps calcul/occupation mémoire ».
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 $\widetilde{\mathbf{K}}$. Si au bout de i=NMAX_ITER étapes, le résidu relatif $\delta^i := \frac{\|\mathbf{r}^i\|}{\|\mathbf{f}\|}$ 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_INC'	[§4.2]
	NIVE_REMPLISSAGE=0,1,2…	0	[§4.2]
	RENUM='RCMK' ou 'SANS'	'RCMK'	
	NMAX_ITER= i_{max}, nombre d'itérations maximum admissible. Si i_{max}=0, fixé automatiquement à $N/2$	0	[Algorithme 5]
	RESI_RELA= $\frac{\|\mathbf{r}^i\|}{\|\mathbf{f}\|}$, valeur maximale du résidu relatif à convergence (à la $i^{ème}$ itération).	10^{-6}	[Algorithme 5 et [§3.3]
	SYME='OUI' ou 'NON' Approximation symétrisée de l'opérateur de travail par $\widetilde{\mathbf{K}} := \frac{1}{2}\left(\mathbf{K}+\mathbf{K}^T\right)$ Licite uniquement en non linéaire	'NON'	[§5.2/3]

Tableau 5.4-1 : Récapitulatif du paramétrage du GCPC.

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 $\left\|\mathbf{r}^i\right\|$ et relatif $\frac{\left\|\mathbf{r}^i\right\|}{\left\|\mathbf{f}\right\|}$, ainsi que le numéro d'itération correpondant, i, dès qu'il décroît d'au moins 10% (non paramétrable).
- La valeur `INFO`=3 trace ces évolutions à chaque itération.
- A convergence, c'est-à-dire dès que $\frac{\left\|\mathbf{r}^i\right\|}{\left\|\mathbf{f}\right\|} < \texttt{RESI\_RELA}$, on rappelle en plus la valeur de la norme du résidu absolu initial $\left\|\mathbf{r}^0\right\|$.

5.5 Conseils d'utilisation

Comme on l'a déjà maintes fois rappelé, dans *Code_Aster* toute une série de facteurs remet en cause les principaux avantages concurentiels « historiques » (cf. [§1]) du GCPC par rapport à ses alter-egos directs :

- Les efforts deployés pour paginer au mieux les ressources mémoires requises par les solveurs directs (pagination complète pour `'LDLT'` et partielle pour '`MULT_FRONT'`).
- La non nécessité d'une stratégie de pivotage qui permet un stockage creux de la matrice de travail (`MORSE` pour `'MULT_FRONT'` et `SKYLINE` pour `'LDLT'`).
- Une gestion optimisée du remplissage pour '`MULT_FRONT'` (matrices frontales pleines).
- Parallélisation en mémoire partagée pour '`MULT_FRONT'` (cf. [bib33] ou [bib4] [§3.2]).
- La matrice de travail est préalablement assemblée et stockée avant le recourt au solveur.
- Mauvais conditionnement généralement constatés en mécanique des structures agravés par la prise en compte des Dirichlet généralisés via des Lagranges (cf. [§5.1]).

Globalement, le meilleur compromis robustesse/encombrement mémoire/coût CPU semble revenir [bib3] à la méthode par défaut du mot-clé `SOLVEUR` : la multifrontale. Cependant, pour des problèmes plutôt bien conditionnés (thermique [bib1], géométrie simple avec un maillage et des caractéristiques matériaux relativement homogène…) ou très gourmands en mémoire (plusieurs millions de ddls), le GCPC peut s'avérer une alternative intéressante.

Du fait de sa modularité naturelle et de la simplicité de ses constituants (produit matrice-vecteur et produit scalaire), il reste néanmoins 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.

D'autre part, une propriété pas encore exploitée dans la version officielle de *Code_Aster*, sa scalabilité intrinsèque, accrédite une parallélisation sur un grand nombre de processeurs. Alors que pour les mêmes nombres de processeurs, les speed-ups des méthodes directes se détériorent grandement. C'est une des raisons qui a conduit à souvent retenir le GCPC comme solveur d'interface pour les méthodes de décomposition de domaines [bib4].

Un dernier petit avantage du GCPC, qui a eu historiquement son importance, peut résider dans le fait qu'il accumule moins les erreurs d'arrondi que ses concurrents directs. Elles sont circonscrites à la dernière itération alors que les méthodes directes les accumulent au cours de la factorisation. Ceci dit on les retrouve indirectement lorsque la convergence tarde à venir et que les résidus générés ne sont plus tout à fait orthogonaux !

En bref, le GCPC implanté dans le *Code_Aster* offre un compromis robustesse/complexité calcul/mémoire moins performant que la multifrontale. Néanmoins, ce constat général est sans doute à pondérer, tant il est vrai qu'il rend d'énormes services dans d'autres domaines de la physique : N3S, TRIFOU, *Code_Saturne*, COCCINELLE, ESTET….
On peut essayer de synthétiser, d'un point de vue « utilisateur du *Code_Aster* » l'état des lieux précédent, dans le tableau ci-dessous.

Solveurs						**Taille de problème**		
	Robustesse	**Mémoire**	**CPU**	**Paramétrage**	**Maintenabilité**	**Petit cas (<10^2 DDL)**	**Cas standards (<10^9 DDL)**	**Très gros cas (>10^9 DDL)**
MULT_FRONT DEFAUT	Excellente	Du fait de la pagination : faible	Bon	Rien à faire	Standard	non	oui	A voir
LDLT	Excellente	Du fait de la pagination : aussi faible que l'on veut	Coûteux	Rien à faire	Assez facile	oui	non	interdit
GCPC	Très variable	Très variable	Très variable	A adapter au cas par cas	Facile	oui	A voir Oui en thermique	Oui ?

Tableau 5.5-1 : Récapitulatif des solveurs linéaires dans le *Code_Aster*.

6 Bibliographie

[1] N. ANFAOUI : Une étude des performances de *Code_Aster* : proposition d'optimisation. Stage du DESS de mathématiques appliquées de PARIS VI (2003).

[2] O. BOITEAU : Algorithme de résolution pour le problème généralisé. Doc. du *Code_Aster* disponible sur le site (http://www.code-aster.org), [R5.01.01] (2000).

[3] O. BOITEAU : Analyse de l'implantation du gradient conjugué dans le *Code_Aster* et tests d'autres variantes via la bibliothèque CXML. CR-I23/03/014 (2003).

[4] O. BOITEAU : Décomposition de domaine et parallélisme en mécanique des structures : Etat de l'art et benchmark pour une implantation raisonnée dans *Code_Aster*. HI-23/03/009 (2003).

[5] J.F. BONNANS, J.C. GILBERT, C. LEMARECHAL & C. SAGASTIZABAL : Optimisation numérique: aspects théoriques et pratiques. Ed. Springer collection mathématiques & applications, 27 (1997).

[6] A. BOURAS, V. FRAYSSE & L. GIRAUD : A relaxation strategy for inner-outer linear solvers in DD methods. Rapport CERFACS TR/PA/00/17 (2000).

[7] T. BUFFARD & J.M. HERARD : Une méthode de préconditionnement polynomial pour des systèmes SPD. HE-41/89.16 (1989).

[8] F. CHAITIN-CHATELIN & T. MESKAUSAS : Inner-outer iterations for mode solvers in structural mechanics : application to *Code_Aster*. (2001).

[9] F.CHAITIN-CHATELIN & T.MESKAUSAS. Inner-outer iterations for power method with Chebyschev acceleration in neutronics. (2002).

[10] B.A. CIPRA : The best of the 20th century: editors name top10 algorithms. SIAM News, 33-4 (2000).

[11] J.C. CULIOLI : Introduction à l'optimisation. Ed. Ellipse (1994).

[12] V. FRAYSSE : The power of backward error analysis. HDR de l'Institut National Polytechnique de Toulouse (200).

[13] I. FRIED : Condition of finite element matrices generated from non uniform meshes. AIAA J., 10 (1972), pp219-231.

[14] G.H. GOLUB & V. LOAN : Matrix computations. Ed. J.Hopkins University Press (1984).

[15] G.H. GOLUB & D.P. O'LEARY : Some history of the conjugate gradient and Lanczos algorithms : 1948-1976. SIAM review, 31-1 (1989), pp50-102.

[16] 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.

[17] G.H. GOLUB & Q. YE : Inexact preconditioned conjugate gradient method with inner-outer iteration. Rapport Stanford University, SCCM-97-04 (1997).

[18] J.P. GREGOIRE : Algorithme du GCPC pour la résolution d'un système linéaire symétrique : présentation générale et mode d'emploi. HI/4333-07 (1982).

[19] J.P. GREGOIRE : Vectorisation efficace du GC dans le cas de matrices symétriques creuses. HI-72/6710 (1990).

[20] J.P. GREGOIRE : Parallélisation du GC pour des matrices creuses sur CRAY C98. HI-76/95/019 (1996).

[21] J.P. GREGOIRE, C. ROSE & B. THOMAS : Direct and iterative solvers for finite-element problems. Numerical algorithms, 16 (1997), pp39-53.

[22] J.P. GREGOIRE : Accélération de la convergence du gradient conjugué préconditionné en utilisant la factorisation incomplète par niveau. HI-76/00/005 (2000).

[23] J.P. GREGOIRE : Parallélisation en mémoire distribuée du GC et du code utilisateur. HI-76/01/007 (2001).

[24] J.P. GREGOIRE : GCPC et calcul rapide des valeurs propres. HI-76/01/001 (2001).

[25] P. JOLY : Méthodes de gradient conjugué. Rapport n°84016 du lab. d'analyse numérique de PARIS VI (1984).

[26] P. LASCAUX & R. THEODOR : Analyse numérique matricielle appliquée à l'art de l'ingénieur. Ed. Masson (1994).

[27] G. MEURANT : Computer solution of large linear systems. Studies in mathematics and its applications, Ed. Elsevier, 1999.

[28] M. MINOUX : Programmation mathématique, théorie et algorithmes. Ed. Dunod (1983).

[29] B. NITROSSO : Méthodes directes pour matrices creuses SPD. Des techniques de base à l'état de l'art. HE-41/92.27 (1992).

[30] J. PELLET : Dualisation des conditions aux limites. Doc. du *Code_Aster* [R3.03.01] (2001) disponible sur le site.

[31] G. POOLE, Y. C.LIU & J. MANDEL : Advancing analysis capabilities in ANSYS though solver technology. (2001) dispo. sur le site ANSYS.

[32] C. ROSE : Une méthode multifrontale parallèle pour la résolution directe des systèmes linéaires. HI-76/95/021 (1995).

[33] C. ROSE : Méthode multifrontale. Doc. du *Code_Aster* [R6.02.02] (2001) dispo. sur le site.

[34] Y. SAAD : Practical use of polynomial preconditionings for the conjugate gradient method. Rapport de recherche YALEU/DCS/RR-282, Université de Yale (1983).

[35] Y. SAAD : Iterative methods for sparse linear systems. Ed. PWS, 1996.

[36] Y. SAAD & H.A. VAN DER VORST : Iterative solution of linear system in the 20th-century. J. Comp. Appl. Math., 123 (2000), pp35-65.

[37] D. SELIGMANN : A propos des méthodes de décomposition de type Gauss. Doc. du *Code_Aster* [R6.02.01] (1993) disponible sur le site.

[38] J.R. SHEWCHUK : An introduction to the conjugate gradient method without the agonizing pain. Rapport interne de l'Université de Carnegie Mellon (1994).

[39] J.L. VAUDESCAL : Introduction aux méthodes de résolution de problèmes aux valeurs propres de grande tailles. HI-72/00/01 (2000).

[40] H. VAN DER VORST : Iterative Krylov methods for large linear systems. Ed. Cambridge University Press (2001).

Page laissée intentionnellement blanche.