Méthodes de Krylov : régularisation de la solution et accélération de la convergence

Frédéric Guyomarch

N° d'ordre: 2409

THÈSE

Présentée devant

devant l'Université de Rennes 1

pour obtenir

le grade de : Docteur de l'Université de Rennes 1
Mention Informatique

par

Frédéric Guyomarc'h

Équipe d'accueil : ALADIN
École Doctorale : MATISSE
Composante universitaire : IRISA

Titre de la thèse :

Méthodes de Krylov :
régularisation de la solution
et accélération de la convergence

soutenue le 10 novembre 2000 devant la commission d'examen

M. :	Yousef	Saad	Président
MM. :	Henk	Van Der Vorst	Rapporteurs
	Gérard	Meurant	
MM. :	Patrick	Perez	Examinateurs
	Pierre	Leca	
	Jocelyne	Erhel	

S'il n'y a pas de solution,
C'est qu'il n'y avait pas de problème

Marcel Duchamp

Remerciements

Je tiens en premier lieu à remercier Monsieur Yousef Saad de m'avoir fait l'honneur de présider ce jury. Je veux aussi le remercier pour toutes les idées et les conseils qu'il m'a apportés à chacune de nos rencontres.

Je remercie aussi Monsieur Gérard Meurant d'avoir accepté de juger ce travail ; je veux de même remercier Monsieur Henk Van Der Vorst d'avoir accepté la tâche de rapporteur, malgré l'obstacle de la langue.

Je remercie également Messieurs Pierre Leca et Patrick Pérez d'être présents dans le jury.

Je tiens aussi à exprimer ma très vive gratitude à Madame Jocelyne Erhel, qui m'a accueilli dans l'équipe Aladin et qui a fait preuve, en toutes circonstances, d'une patience exemplaire. Ce travail n'aurait jamais abouti sans ses conseils et lectures critiques.

Introduction

Les systèmes linéaires sont au cœur de nombreux problèmes du calcul scientifique. Parfois, leur présence est explicite ; d'autres fois, ils sont cachés derrière plusieurs interfaces de modélisation ou de calcul. Ce second cas de figure est le plus fréquent. L'exemple le plus classique est la résolution d'une équation non linéaire par la méthode de Newton [2] : on linéarise l'équation au voisinage de la solution approchée. La solution du système linéaire obtenu donne une nouvelle solution, meilleure si les hypothèses du théorème de convergence sont respectées. Bien souvent, le solveur linéaire est, comme dans le cas précédent, la boîte de calcul la plus sollicitée, bien qu'elle n'apparaisse pas explicitement lors de la mise en équations du modèle.

De plus, la taille des systèmes à résoudre, dépend directement du nombre d'inconnues. Celui-ci augmentant avec le nombre de degrés de liberté du modèle et la finesse de la discrétisation, les systèmes à résoudre deviennent vite énormes, ce qui nécessite des stockages spéciaux pour les matrices, généralement creuses. Parfois, ces matrices sont tellement grandes qu'il faut les conserver sur des supports auxiliaires car elles ne tiennent pas en mémoire principale. Quand on est dans ce cas, alors toutes les méthodes directes sont prohibées, car elles demandent des accès à la structure de la matrice, bien trop coûteux. C'est pourquoi, on se tourne vers les méthodes itératives.

Ces méthodes partent toutes d'une solution initiale x_0 que l'on va chercher à améliorer ; ce qui donnera une nouvelle solution approchée, que l'on affine à nouveau, jusqu'à ce que la solution courante soit suffisamment proche de la solution exacte. Pour mesurer l'erreur commise, on s'appuie principalement sur la norme du résidu $r_k = b - Ax_k$, car l'erreur exacte $e_k = A^{-1}b - x_k$ n'est pas calculable. Cela pose donc un premier problème : celui de la précision des calculs. En effet, on se rend compte que pour certaines matrices la convergence devient très lente à partir d'un certain point. Parfois ce comportement n'est que local, c'est ce que l'on appelle un palier. Il faut alors étudier les causes du palier [13, 40] et essayer de les supprimer. Enfin, le pire cas est quand l'accumulation des erreurs de calculs font diverger le système.

Un autre cas d'erreur arrive quand les données ne sont pas exactes. Le système que l'on résout, peut être dénaturé par du bruit. Dans ce cas, la solution désirée est la solution du système non perturbé et non celle du système bruité ; d'autant plus que ces deux solutions peuvent être très éloignées l'une de l'autre. Dans ce cas, on a l'habitude de modifier le système pour qu'il soit moins sensible aux erreurs et donc la solution de ce nouveau système est plus proche de la solution désirée que la solution du système original : c'est la régularisation [22, 32]. Les techniques les plus classiques sont

la régularisation de Tikhonov [42, 41] et les techniques de troncatures comme TSVD [21, 23].

Le second problème est la vitesse de convergence. On recherche, en effet, une vitesse de convergence maximale, car le solveur linéaire, on l'a dit, est souvent la clef de l'efficacité de l'application toute entière. Pour quantifier cette vitesse, on peut mesurer le temps CPU nécessaire à la résolution d'un ou plusieurs systèmes tests. On se rend cependant compte que lors d'une itération, il y a un petit nombre, indépendant des itérations, d'opérations avec la matrice. Ce sont des opérations de type BLAS2 si la matrice A est pleine. Sinon, cela peut être une multiplication creuse ou même l'utilisation d'une boîte noire qui calcule le produit de la matrice par un vecteur. En plus de ces opérations matricielles dont le coût est fixe pour une itération, il y a quelques opérations BLAS1 de mise à jour de vecteurs, dont le coût est la plupart du temps négligeable devant celui des premières. C'est pourquoi, on mesure l'efficacité des solveurs au nombre d'itérations nécessaires pour la convergence.

On cherchera donc tout au long de cette thèse à améliorer ces deux aspects des solveurs linéaires : la robustesse et l'efficacité, dans le cas où le système est symétrique défini positif. Cette propriété est très fréquente pour les matrices obtenues par la discrétisation d'EDP. On suppose aussi ces matrices suffisamment grandes pour rendre rédhibitoire le coût d'une méthode directe.

Le plan s'articule ainsi :

- dans le premier chapitre, on rappelle les propriétés des solveurs itératifs les plus classiques (relaxation, gradient conjugué), en détaillant plus particulièrement les méthodes de Krylov, à la base des chapitres suivants ;
- le second chapitre montre des tests effectués pour un problème classique de restauration d'images et confirme l'efficacité du gradient conjugué ;
- une technique de régularisation à base de filtres polynomiaux est proposée dans le troisième chapitre ;
- enfin des méthodes d'accélération de l'algorithme du gradient conjugué sont développées dans le quatrième chapitre.

La conclusion rappelle les résultats nouveaux et donne une vue des perspectives de ce travail.

Chapitre 1

Les solveurs itératifs

1.1 Méthodes de point fixe et méthodes de projection

Il y a deux grandes classes de méthodes itératives pour résoudre un système linéaire

$$Ax = b. \tag{1.1}$$

La première classe comprend les méthodes de point fixe : ce sont des méthodes qui utilisent une contraction pour, à chaque étape, se rapprocher de la solution exacte.

La seconde comprend les méthodes de projection qui calculent la projection de la solution exacte sur un certain espace. Si cet espace est de la forme $\langle v, Av \dots, A^{k-1}v \rangle = \mathcal{K}_k(A,v)$, alors la méthode est dite de Krylov, du nom de l'espace $\mathcal{K}_k(A,v)$, appelé espace de Krylov.

1.1.1 Méthodes de point fixe

Dans de telles méthodes, on calcule une solution approchée x_{k+1} à partir de la solution approchée de l'étape précédente :

$$x_{k+1} = Bx_k + c. \tag{1.2}$$

Généralement B provient d'une décomposition de A sous la forme : $A = M - N$. On écrit alors

$$Mx_{k+1} = Nx_k + b$$

d'où

$$\begin{aligned} B &= M^{-1}N \\ c &= M^{-1}b. \end{aligned}$$

La convergence de ces méthodes est régie par le théorème suivant :

Théorème 1.1 *On suppose qu'il existe un x unique tel que $x = Bx + c$. Alors la suite $\{x_k\}$ définie par l'itération (1.2) converge vers x quel que soit le vecteur initial x_0 si $\rho(B) < 1$ où $\rho(B) = \max\{|\lambda|, \lambda$ valeur propre de $B\}$ est le rayon spectral de B.*

Preuve voir [2, page 10]. □

On pose pour la suite

$$D = \begin{bmatrix} a_{1,1} & & \\ & \ddots & \\ & & a_{n,n} \end{bmatrix}, \quad -E = \begin{bmatrix} 0 & & \\ a_{2,1} & \ddots & \\ a_{n,1} & a_{n,n-1} & 0 \end{bmatrix}, \quad -F = \begin{bmatrix} 0 & a_{1,2} & a_{1,n} \\ & \ddots & a_{n-1,n} \\ & & 0 \end{bmatrix}.$$

La décomposition la plus simple est alors de prendre pour M la diagonale de A. On a donc

$$\begin{aligned} M &= D \\ N &= E + F \end{aligned}$$

C'est la méthode de Jacobi. Le calcul de x_{k+1} se fait de la façon suivante

$$[x_{k+1}]_i = \frac{1}{a_{i,i}} \left(b_i - \sum_{j=1}^{i-1} a_{i,j} [x_k]_j - \sum_{j=i+1}^{n} a_{i,j} [x_k]_j \right).$$

Si on utilise les composantes de x_{k+1} déjà calculées, pour la mise à jour du vecteur, on obtient la méthode de Gauss-Seidel :

$$[x_{k+1}]_i = \frac{1}{a_{i,i}} \left(b_i - \sum_{j=1}^{i-1} a_{i,j} [x_{k+1}]_j - \sum_{j=i+1}^{n} a_{i,j} [x_k]_j \right). \tag{1.3}$$

Cette méthode correspond à la décomposition

$$\begin{aligned} M &= D - E \\ N &= F. \end{aligned}$$

Pour A symétrique définie positive, cette méthode converge toujours. En effet

Théorème 1.2 *Soit $A \in \mathcal{M}_n(\mathbb{R})$ symétrique définie positive, alors la méthode de Gauss-Seidel définie par (1.3) converge pour tout x_0.*

Preuve voir [17, page 512]. □

Malheureusement cette méthode produit souvent une matrice B dont le rayon spectral est proche de 1. Pour y pallier, on définit une méthode dite de sur-relaxation qui fixe x_{k+1} comme un barycentre de x_k et de la solution produite par la méthode de Gauss-Seidel x_{k+1}^{GS}

$$x_{k+1} = (1 - \omega) x_k + \omega x_{k+1}^{\mathsf{GS}}.$$

Cela correspond à l'itération suivante

$$[x_{k+1}]_i = (1 - \omega) [x_k]_i + \frac{\omega}{a_{i,i}} \left(b_i - \sum_{j=1}^{i-1} a_{i,j} [x_{k+1}]_j - \sum_{j=i+1}^{n} a_{i,j} [x_k]_j \right), \tag{1.4}$$

et à la décomposition suivante

$$\begin{aligned} M_\omega &= \frac{1}{\omega}D - E \\ N_\omega &= \frac{1-\omega}{\omega}D + F. \end{aligned}$$

Pour certains problèmes, on connaît le ω optimal [44, 17], c'est-à-dire celui qui minimise le rayon spectral de $M^{-1}N$. Mais dans le cas général, il peut être nécessaire de faire une étude des valeurs propres pour le connaître.

1.1.2 Méthodes de projection

Une autre classe de méthode comprend les méthodes de projection [14, 1]. Pour définir une telle méthode, on utilise deux conditions. La première fixe «l'endroit» où l'on cherche la solution à la k^{e}itération : usuellement une sous-variété affine de $\mathbb{R}^n$. Pour les méthodes dites de Krylov, cette sous-variété est $x_0 + \mathcal{K}_k(A,r_0)$ où $\mathcal{K}_k(A,r_0)$ est l'espace de Krylov de dimension k engendré par A et r_0, r_0 étant le résidu initial $b - Ax_0$. C'est-à-dire que l'on a

$$\mathcal{K}_k(A,r_0) = \langle r_0, Ar_0, \ldots, A^{k-1}r_0 \rangle.$$

Cet espace contenant x_k est appelé espace des solutions. Donc pour les méthodes de Krylov, on a la condition suivante

$$x_k \in x_0 + \mathcal{K}_k(A,r_0). \tag{1.5}$$

Il faut alors fixer le x_k qui convient dans cet espace. On peut désirer que ce x_k minimise l'erreur ou le résidu dans une certaine norme, ou bien que l'un de ces deux vecteurs soit orthogonal (pour une certaine forme bilinéaire) à un espace de dimension k. Si A est symétrique définie positive, ces deux objectifs se rejoignent. Le théorème 1.3 le précise.

Théorème 1.3 *Soit W un sous-espace vectoriel de $\mathbb{R}^n$ et A une matrice symétrique définie positive de $\mathcal{M}_n(\mathbb{R})$. Alors le point z minimise la quantité suivante $\|e_y\|_A = \|A^{-1}b - y\|_A$ pour $y \in W$, si et seulement si le résidu associé $r_z = b - Az$ est orthogonal à W. C'est-à-dire $W^* r_z = 0$.*

Preuve Soit $y \in W$. Alors

$$e_y = A^{-1}b - y = e_z + z - y.$$

D'où

$$\begin{aligned} \|e_y\|_A^2 &= \|e_z + z - y\|_A^2, \\ &= \|e_z\|_A^2 + \|z - y\|_A^2 + 2\,(Ae_z, z - y)\,, \\ &= \|e_z\|_A^2 + \|z - y\|_A^2 + 2\,(r_z, z - y)\,. \end{aligned}$$

Soit $z - y = \lambda t$ avec $\lambda \in \mathbb{R}$ et $t \in W$, A-unitaire. On a donc :

$$\begin{aligned} \|e_y\|_A \geq \|e_z\|_A &\Rightarrow \lambda^2 + 2\lambda(r_z,t) \geq 0, \quad \forall \lambda \in \mathbb{R}, \\ &\Rightarrow (r_z,t)^2 \leq 0. \end{aligned}$$

Donc $(r_z,t) = 0$, d'où $r_z \perp W$. La réciproque est une conséquence immédiate du théorème de Pythagore. □

D'où, pour A symétrique définie positive, si on veut minimiser l'erreur e_k en norme A sur l'espace des solutions, il suffit de trouver le x_k qui rend le résidu orthogonal à cet espace. Cette condition est appelée condition de Petrov-Galerkin. Elle s'écrit ici

$$r_k \perp \mathcal{K}_k(A,r_0). \tag{1.6}$$

Ces deux conditions definissent un unique point x_k. La méthode de Lanczos et le gradient conjugué sont des algorithmes permettant de calculer ce x_k.

1.2 La méthode de Lanczos

C'est un procédé itératif qui comporte deux étapes : tout d'abord, on calcule une base orthonormée U_m de $\mathcal{K}_m$, ainsi que la matrice T_m de A dans cette base. On suppose A symétrique définie positive, ainsi T_m le sera aussi. Cette phase est appelée Procédé de Lanczos, dans le cas symétrique. Puis on produit la solution x_m vérifiant les conditions (1.5) et (1.6).

1.2.1 Le procédé de Lanczos

Il peut être vu comme une orthogonalisation de Gram-Schmidt sur une base de $\mathcal{K}_m$: la base des itérés $[r_0, \ldots, A^{m-1}r_0]$. On note U_m la matrice formée par les vecteurs $[u_1, \ldots, u_m]$, où u_k est l'orthonormalisé de $A^k r_0$ par rapport à U_{k-1}. C'est-à-dire

$$u_{k+1} \text{ est unitaire, parallèle à } Au_k - \sum_{j=0}^{k} (Au_k,u_j)u_j.$$

Mais pour tout $j \in \{0, \ldots, k-2\}$, $(Au_k,u_j) = 0$. En effet $(Au_k,u_j) = (u_k,Au_j)$. Or $Au_j \in \mathcal{K}_{j+1}$ et donc $u_k \perp Au_j$ dès que $k > j+1$, c'est-à-dire $j < k-1$. On en déduit que

$$u_{k+1} \text{ est unitaire, parallèle à } Au_k - (Au_k,u_k)u_k - (Au_k,u_{k-1})u_{k-1}.$$

On pose alors

$$\begin{aligned} \alpha_k &= (Au_k,u_k) \\ \beta_k &= (Au_k,u_{k-1}) \\ \gamma_{k+1} &= \|Au_k - \alpha_k u_k - \beta_k u_{k-1}\|_2 \end{aligned}$$

d'où $Au_k = \gamma_{k+1}u_{k+1} + \alpha_k u_k + \beta_k u_{k-1}$. Puis, en prenant le produit scalaire avec u_{k+1}, on a

$$\begin{aligned} \gamma_{k+1} &= (u_{k+1}, Au_k) \\ &= (Au_{k+1}, u_k) \\ &= \beta_{k+1}. \end{aligned}$$

Finalement, le procédé de Lanczos, pour construire une base de taille m, est résumé dans l'algorithme 1.1.

$\beta_1 = \|r_0\|$
$u_1 = \frac{1}{\beta_1} r_0$
$u_0 = 0$
for $k = 1, \ldots, m$ **do**
  $u_{k+1} = Au_k - \beta_k u_{k-1}$
  $\alpha_k = (u_{k+1}, u_k)$
  $u_{k+1} = x_{k+1} - \alpha_k u_k$
  $\beta_{k+1} = \|u_{k+1}\|_2$
  $u_{k+1} = \frac{1}{\beta_{k+1}} u_{k+1}$
end for

Algorithme 1.1: Procédé de Lanczos

U_m est une base orthonormée de $\mathcal{K}_m$, c'est-à-dire $U_m^* U_m = I_m$. De plus, on note T_m la matrice tridiagonale avec pour diagonale les $(\alpha_k)_{k=1\ldots m}$ et pour sur- et sous-diagonale les $(\beta_k)_{k=2\ldots m}$. On a alors

$$AU_m = U_m T_m + \beta_{m+1} u_{m+1} e_m^{(m)*},$$

avec $e_j^{(m)}$, j^{e} vecteur de la base canonique de $\mathbb{R}^m$. On en déduit, par multiplication avec U_m^* que:

$$T_m = U_m^* A U_m.$$

1.2.2 Production de la solution

On recherche la solution x_m vérifiant (1.5) et (1.6). Or

$$\begin{aligned} (1.5) &\Leftrightarrow x_m \in x_0 + \mathcal{K}_m \\ &\Leftrightarrow x_m = x_0 + U_m \Omega \end{aligned}$$

avec Ω un vecteur quelconque. On a alors

$$\begin{aligned} r_m &= b - Ax_m \\ &= r_0 - AU_m \Omega. \end{aligned}$$

Enfin

$$\begin{aligned}
(1.6) \quad &\Leftrightarrow \quad r_m \perp \mathcal{K}_m \\
&\Leftrightarrow \quad U_m^* r_m = 0 \\
&\Leftrightarrow \quad U_m^*(r_0 - AU_m\Omega) = 0 \\
&\Leftrightarrow \quad T_m\Omega = U_m^* r_0 \\
&\Leftrightarrow \quad T_m\Omega = U_m^*(\beta_1 u_1) \\
&\Leftrightarrow \quad \Omega = \beta_1 T_m^{-1} e_1^{(m)}.
\end{aligned}$$

C'est-à-dire que la solution est

$$x_m = x_0 + \beta_1 U_m T_m^{-1} e_1^{(m)},$$

que l'on calcule de la façon suivante

1: trouver f tel que $T_m f = \beta_1 e_1^{(m)}$
2: $x_m = x_0 + U_m f$

1.2.3 Calcul du résidu

La propriété intéressante à ce stade est la possibilité de calculer la norme du résidu (ou l'erreur en norme A^2) sans produire la solution [34]. En effet

$$\begin{aligned}
r_m &= b - Ax_m \\
&= b - A(x_0 + U_m f) \\
&= r_0 - AU_m f \\
&= \beta_1 u_1 - AU_m f \\
&= U_m \beta_1 e_1^{(m)} - AU_m f \\
&= U_m T_m f - AU_m f \\
&= -\beta_{m+1} u_{m+1} e_m^{(m)^*} f \\
&= -\beta_{m+1} f_m u_{m+1}.
\end{aligned}$$

D'où

$$\|r_m\|_2 = \beta_{m+1}|f_m| \tag{1.7}$$

avec

$$f_m = e_m^{(m)^*} f.$$

La mise à jour de $\|r_m\|_2$, à chaque itération, se résume donc au calcul de f_m, dernière composante du vecteur solution du système $T_m f = \beta_1 e_1^{(m)}$.

1.2.3.1 Grâce aux formules de Cramer

Les formules de Cramer donnent directement f_m sous forme d'un quotient de déterminant

$$f_m = \frac{\begin{vmatrix} \alpha_1 & \beta_2 & & & \beta_1 \\ \beta_2 & \ddots & \ddots & & 0 \\ & \ddots & \ddots & \beta_{m-1} & \vdots \\ & & \ddots & \alpha_{m-1} & \vdots \\ & & & \beta_m & 0 \end{vmatrix}}{\begin{vmatrix} \alpha_1 & \beta_2 & & \\ \beta_2 & \ddots & \ddots & \\ & \ddots & \ddots & \beta_m \\ & & \beta_m & \alpha_m \end{vmatrix}} = \frac{(-1)^{m-1}\prod_{k=1}^{m}\beta_k}{|T_m|}.$$

Le déterminant $|T_m|$ de T_m se calcule lui grâce à une récurrence d'ordre 2. En effet, en développant par rapport à la dernière colonne, on a

$$|T_m| = \alpha_m |T_{m-1}| - \beta_m \begin{vmatrix} \alpha_1 & \beta_2 & & & 0 \\ \beta_2 & \ddots & \ddots & & \vdots \\ & \ddots & \ddots & \beta_{m-2} & 0 \\ & & \beta_{m-2} & \alpha_{m-2} & \beta_{m-1} \\ 0 & \cdots & \cdots & 0 & \beta_m \end{vmatrix},$$

puis en développant ce dernier déterminant par rapport à la dernière ligne, on obtient

$$|T_m| = \alpha_m |T_{m-1}| - \beta_m^2 |T_{m-2}|.$$

La récurrence est initialisée par

$$\begin{cases} |T_0| = 1 \\ |T_1| = \alpha_1 \end{cases}$$

On a alors

$$f_{m+1} = -\beta_{m+1}\frac{|T_m|}{|T_{m+1}|}f_m.$$

D'où, en posant, $t_k = \frac{|T_k|}{|T_{k-1}|}$, on obtient les récurrences suivantes pour le calcul de f_m

$$\begin{cases} t_1 = \alpha_1 \\ f_1 = \frac{\beta_1}{t_1} \\ \forall k > 1, \quad t_k = \alpha_k - \frac{\beta_k^2}{t_{k-1}} \\ \qquad\qquad f_k = -\frac{\beta_k}{t_k} f_{k-1}. \end{cases}$$

On a donc

$$\|r_m\|_2 = \frac{\beta_{m+1}}{t_m}\|r_{m-1}\|_2.$$

1.2.3.2 Grâce à une factorisation LDL^*

Une autre méthode pour obtenir ce résultat, consiste à factoriser la matrice $T_m = LDL^*$, puis à résoudre les systèmes triangulaires bidiagonaux par récurrence. Supposons en effet que l'on connaisse une factorisation LDL^* de T_m, avec

$$L = \begin{bmatrix} 1 & & & \\ l_2 & \ddots & & \\ & \ddots & \ddots & \\ & & l_m & 1 \end{bmatrix} \quad \text{et} \quad D = \begin{bmatrix} d_1 & & & \\ & \ddots & & \\ & & \ddots & \\ & & & d_m \end{bmatrix}.$$

On résout alors successivement les systèmes $Lh = \beta_1 e_1$ puis $Dg = h$ et enfin $L^* f = g$. Le premier système s'écrit

$$\begin{bmatrix} 1 & & & \\ l_2 & \ddots & & \\ & \ddots & \ddots & \\ & & l_m & 1 \end{bmatrix} \begin{bmatrix} h_1 \\ \vdots \\ \vdots \\ h_m \end{bmatrix} = \begin{bmatrix} \beta_1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

Ce qui implique

$$h_i = (-1)^{i-1}\beta_1 \prod_{j=2}^{i} l_j.$$

Le deuxième système s'écrit

$$\begin{bmatrix} d_1 & & & \\ & \ddots & & \\ & & \ddots & \\ & & & d_m \end{bmatrix} \begin{bmatrix} g_1 \\ \vdots \\ \vdots \\ g_m \end{bmatrix} = \begin{bmatrix} h_1 \\ \vdots \\ \vdots \\ h_m \end{bmatrix}.$$

Ce qui implique

$$g_i = (-1)^{i-1}\beta_1 \frac{\prod_{j=2}^{i} l_j}{d_i}.$$

On remarque que les coefficients ne dépendent que de leur rang: pour avoir le vecteur g de taille $m+1$, il suffit de calculer $g_{m+1} = (-1)^m \beta_1 \frac{\prod_{j=2}^{m+1} l_j}{d_{m+1}}$ et de l'ajouter au vecteur g en dernière ligne. Le troisième système, enfin, s'écrit

$$\begin{bmatrix} 1 & l_2 & & \\ & \ddots & \ddots & \\ & & \ddots & l_m \\ & & & 1 \end{bmatrix} \begin{bmatrix} f_1 \\ \vdots \\ \vdots \\ f_m \end{bmatrix} = \begin{bmatrix} g_1 \\ \vdots \\ \vdots \\ g_m \end{bmatrix}.$$

On en déduit

$$\begin{aligned} f_m &= g_m \\ &= (-1)^{m-1}\beta_1 \frac{\prod_{j=2}^{m} l_j}{d_m} \end{aligned} \tag{1.8}$$

ainsi que

$$f_i = g_i - l_{i+1} f_{i+1}, \text{ pour } i = m-1, \ldots, 1. \tag{1.9}$$

Pour le calcul de la factorisation LDL^*, on procède par identification :

$$\begin{bmatrix} 1 & & & \\ l_2 & \ddots & & \\ & \ddots & \ddots & \\ & & l_m & 1 \end{bmatrix} \begin{bmatrix} d_1 & & & \\ & \ddots & & \\ & & \ddots & \\ & & & d_m \end{bmatrix} \begin{bmatrix} 1 & l_2 & & \\ & \ddots & \ddots & \\ & & \ddots & l_m \\ & & & 1 \end{bmatrix}$$

$$= \begin{bmatrix} d_1 & d_1 l_2 & & \\ d_1 l_2 & d_1 l_2^2 + d_2 & \ddots & \\ & \ddots & \ddots & d_{m-1} l_m \\ & & d_{m-1} l_m & d_{m-1} l_m^2 + d_m \end{bmatrix}$$

$$= \begin{bmatrix} \alpha_1 & \beta_2 & & \\ \beta_2 & \ddots & \ddots & \\ & \ddots & \ddots & \beta_m \\ & & \beta_m & \alpha_m \end{bmatrix}.$$

On a donc

$$d_1 = \alpha_1,$$

puis on utilise la récurrence suivante, pour $i \geq 2$

$$\begin{cases} l_i = \frac{\beta_i}{d_{i-1}} \\ d_i = \alpha_i - d_{i-1} l_i^2. \end{cases}$$

La seconde ligne peut aussi se calculer avec

$$d_i = \alpha_i - \frac{\beta_i^2}{d_{i-1}}.$$

On peut ici remarquer que $d_k = t_k = \frac{|T_k|}{|T_{k-1}|}$. En remplaçant alors, dans (1.8), l_i par sa valeur, on obtient

$$\begin{aligned} f_m &= (-1)^{m-1} \frac{\prod_{j=1}^{m} \beta_j}{\prod_{j=1}^{m} d_j} \\ &= -\frac{\beta_m}{d_m} f_{m-1}. \end{aligned} \tag{1.10}$$

On retrouve donc bien les formules obtenues grâce aux formules de Cramer, dans la partie 1.2.3.1 page 11.

Mais on peut aussi calculer f grâce à $L_{m-1}^{-*}[g]_{i=1\ldots m-1}$, c'est-à-dire le f que l'on obtient à l'itération précédente. On note $f^{(i)}$ le vecteur f correspondant à l'itération i ; et on note $\tilde{f}$ le vecteur f privé de sa dernière ligne. On peut alors remarquer que

$$\begin{aligned}
& L_{m+1}^* f^{(m+1)} = g^{(m+1)} \\
\Rightarrow \quad & \left[\begin{array}{c|c} L_m^* & l_{m+1} \\ \hline & 1 \end{array}\right] \left[\begin{array}{c} \tilde{f}^{(m+1)} \\ \hline g_{m+1} \end{array}\right] = \left[\begin{array}{c} \tilde{g}^{(m+1)} \\ \hline g_{m+1} \end{array}\right] \\
\Rightarrow \quad & L_m^* \tilde{f}^{(m+1)} + l_{m+1} g_{m+1} e_m^{(m)} = \tilde{g}^{(m+1)} \\
\Rightarrow \quad & \tilde{f}^{(m+1)} = L_m^{-*} g^{(m)} - g_{m+1} l_{l+1} L_m^{-*} e_m^{(m)} \\
\Rightarrow \quad & \tilde{f}^{(m+1)} = f^{(m)} - g_{m+1} l_{l+1} L_m^{-*} e_m^{(m)} \qquad (1.11)
\end{aligned}$$

1.2.3.3 Algorithme final

Si on utilise maintenant la relation (1.7), on obtient

$$\begin{aligned}
\|r_m\|_2 &= \beta_{m+1} |f_m| \\
&= \beta_{m+1} \frac{\beta_m}{d_m} |f_{m-1}| \\
&= \frac{\beta_{m+1}}{d_m} \|r_{m-1}\|_2 . \qquad (1.12)
\end{aligned}$$

On ajoute alors ce calcul dans la boucle de l'algorithme de Lanczos pour résoudre un système symétrique et on obtient l'algorithme 1.2 page ci-contre. Dans cet algorithme, n_k désigne la norme euclidienne de r_k et est l'indicateur de convergence. Il y a breakdown si $n_k = 0$, ce qui implique $\beta_k = 0$. Dans ce cas, on a

$$AU = UT$$

et x_k est la solution exacte du système. C'est un happy breakdown. De plus, $d_k \neq 0$, car d_k est le quotient du determinant de T_k par celui de T_{k-1}, deux matrices définies positives.

Pour la résolution du système tridiagonal, plusieurs implémentations sont possibles en fonction de la stratégie de calcul du résidu. Si on a choisi la factorisation LDL^*, alors le système se résout par la récurrence 1.9 à partir du vecteur g dont on calcule une composante à chaque itération du procédé de Lanczos. Si on ne connaît pas *a priori* la factorisation de T, on peut la calculer ou résoudre le système avec des rotations de Givens.

Au lieu de calculer la norme euclidienne du résidu, on peut calculer une borne supérieure de l'erreur en norme A grâce à la méthode des quadratures [18, 19], qui exprime $r_k^* A^{-1} r_k$ comme une intégrale avec une mesure constante par morceaux.

Require: x_0 solution initiale et tol seuil d'arrêt
1: $r_0 = b - Ax_0$
2: $\beta_1 = n_1 = \|r_0\|_2$
3: $u_1 = \frac{1}{\beta_1} r_0$
4: $u_0 = 0$, $k = 1$
5: **repeat**
6: $u_{k+1} = Au_k - \beta_k u_{k-1}$
7: $\alpha_k = u_{k+1}^* u_k$
8: $u_{k+1} = x_{k+1} - \alpha_k u_k$
9: $\beta_{k+1} = \|u_{k+1}\|_2$
10: **if** $\beta_{k+1} \neq 0$ **then**
11: $u_{k+1} = \frac{1}{\beta_{k+1}} u_{k+1}$
12: **end if**
13: **if** $k = 1$ **then**
14: $d_1 = \alpha_1$
15: **else**
16: $d_k = \alpha_k - \frac{\beta_k^2}{d_{k-1}}$
17: **end if**
18: $n_{k+1} = \frac{\beta_{k+1}}{d_k} n_k$
19: $k = k + 1$
20: **until** $n_k < tol$
21:
22: $m = k$
23: résoudre $T_m f = \beta_1 e_1^{(m)}$
24: $x_m = x_0 + U_m f$

Algorithme 1.2: Algorithme de Lanczos

1.3 Le gradient conjugué

1.3.1 Dérivation de l'algorithme de Lanczos en gradient conjugué

On a vu dans la partie 1.2.3.2 page 12, que l'on pouvait exprimer la dernière composante de f grâce à une récurrence. En poussant cette réflexion plus loin, on exprime par récurrence tout le vecteur f, et ainsi on calcule la nouvelle solution x_{k+1} grâce à des récurrences. Ce nouvel algorithme s'appelle algorithme du gradient conjugué [17, 36, 29]. Il a les mêmes propriétés que l'algorithme de Lanczos dont il est dérivé, c'est-à-dire qu'il permet de résoudre un système linéaire symétrique défini positif et la résolution se fait sans breakdown.

Pour la dérivation, on utilise un vecteur auxiliaire $p_k = -\beta_{k+1} f_k U_{k+1} L_{k+1}^{-*} e_{k+1}$. Or

$$L_{k+1}^{-*} e_{k+1} = \begin{bmatrix} -l_{k+1} L_k^{-*} e_k \\ \hline 1 \end{bmatrix}$$

donc

$$\begin{aligned} p_k &= -\beta_{k+1} f_k U_{k+1} \begin{bmatrix} -l_{k+1} L_k^{-*} e_k \\ \hline 1 \end{bmatrix} \\ &= -\beta_{k+1} f_k \left(u_{k+1} - l_{k+1} U_k L_k^{-*} e_k \right) \\ &= -\beta_{k+1} f_k u_{k+1} - \frac{\beta_{k+1}}{\beta_k} l_{k+1} \frac{f_k}{f_{k-1}} p_{k-1} \end{aligned}$$

or

$$-\beta_{k+1} f_k u_{k+1} = r_k$$

et

$$\begin{aligned} \frac{f_k}{f_{k-1}} &= -\frac{\beta_k}{d_k} \\ l_{k+1} &= \frac{\beta_{k+1}}{d_k} \end{aligned}$$

donc

$$p_k = r_k + \frac{\beta_{k+1}^2}{d_k^2} p_{k-1}.$$

On en déduit alors grâce à (1.11)

$$\begin{aligned} x_{k+1} &= x_k + f_{k+1} U_{k+1} L_{k+1}^{-*} e_{k+1} \\ &= x_k - \frac{f_{k+1}}{f_k \beta_{k+1}} p_k \\ &= x_k + \frac{1}{d_{k+1}} p_k \end{aligned}$$

et

$$r_{k+1} = r_k - \frac{1}{d_{k+1}} A p_k.$$

On pose alors

$$\begin{aligned} \alpha_k^{\mathsf{CG}} &= \frac{1}{d_{k+1}} \\ \beta_k^{\mathsf{CG}} &= \frac{\beta_{k+1}^2}{d_k^2}. \end{aligned}$$

On a alors les récurrences suivantes pour la mise à jour des vecteurs r_k et p_k

$$\begin{aligned} r_{k+1} &= r_k - \alpha_k^{\mathsf{CG}} A p_k && (1.13)\\ p_k &= r_k + \beta_k^{\mathsf{CG}} p_{k-1}. && (1.14) \end{aligned}$$

Les vecteurs p_k sont appelés directions de descente car c'est suivant la droite $x_k + \mathbb{R}p_k$ que l'on recherche x_{k+1}. Et comme $r_{k+1} \perp p_k$, x_{k+1} minimise $\|e_k\|_A$ sur la droite $x_k + \mathbb{R}p_k$, et donc l'erreur diminue.

1.3.2 L'algorithme du gradient conjugué

De plus, les α_k^{CG} et les β_k^{CG} peuvent se calculer sans les α_k et les β_k. En effet

$$\begin{aligned} p_k^* A p_k &= \left(-\beta_{k+1} f_k U_{k+1} L_{k+1}^{-*} e_{k+1}\right)^* A \left(-\beta_{k+1} f_k U_{k+1} L_{k+1}^{-*} e_{k+1}\right)\\ &= \beta_{k+1}^2 f_k^2 e_{k+1}^* L_{k+1}^{-1} T_{k+1} L_{k+1}^{-*} e_{k+1}\\ &= \beta_{k+1}^2 f_k^2 e_{k+1}^* D_{k+1} e_{k+1}\\ &= \beta_{k+1}^2 f_k^2 d_{k+1}\\ &= (r_k^* r_k)\, d_{k+1} \end{aligned}$$

et d'après (1.12)

$$\frac{\|r_k\|}{\|r_{k-1}\|} = \frac{\beta_{k+1}}{d_k}.$$

D'où

$$\begin{aligned} \alpha_k^{\mathsf{CG}} &= \frac{r_k^* r_k}{p_k^* A p_k},\\ \beta_{k+1}^{\mathsf{CG}} &= \frac{r_{k+1}^* r_{k+1}}{r_k^* r_k}. \end{aligned}$$

Théorème 1.4 *On appelle $R_k = [r_0, \ldots, r_{k-1}]$ et $P_k = [p_0, \ldots, p_{k-1}]$, les matrices dont les colonnes sont les vecteurs engendrés par les récurrences (1.13) et (1.14). On a alors l'égalité suivante*

$$\langle R_k \rangle = \langle P_k \rangle = \langle U_k \rangle = \mathcal{K}_k(A, r_0) \qquad (1.15)$$

Preuve On a

$$\begin{aligned} R_k &= [r_0, \ldots, r_{k-1}]\\ &= U_k \begin{bmatrix} -\beta_1 f_0 & & \\ & \ddots & \\ & & -\beta_k f_{k-1} \end{bmatrix}, \end{aligned}$$

donc $\langle R_k \rangle = \langle U_k \rangle = \mathcal{K}_k(A, r_0)$.

De plus

$$\begin{aligned} P_k &= [p_0, \ldots, p_{k-1}] \\ &= U_k L_k^{-*} \begin{bmatrix} -\beta_1 f_0 & & \\ & \ddots & \\ & & -\beta_k f_{k-1} \end{bmatrix}, \end{aligned}$$

d'où $\langle P_k \rangle = \langle U_k \rangle = \mathcal{K}_k(A, r_0)$. □

L'algorithme dérivé s'appelle algorithme du gradient conjugué, à cause de la relation de conjugaison des directions de descente (1.16). Cela est résumé dans l'algorithme 1.3 et le théorème 1.5.

Théorème 1.5 *Les p_k sont A-conjugués, c'est-à-dire*

$$p_i^* A p_j = 0 \quad \textit{pour } i \neq j. \tag{1.16}$$

Preuve

$$\begin{aligned} P_k^* A P_k &= \begin{bmatrix} -\beta_1 f_0 & & \\ & \ddots & \\ & & -\beta_k f_{k-1} \end{bmatrix} L_k^{-1} U_k^* A u_k L_k^{-*} \begin{bmatrix} -\beta_1 f_0 & & \\ & \ddots & \\ & & -\beta_k f_{k-1} \end{bmatrix} \\ &= \begin{bmatrix} -\beta_1 f_0 & & \\ & \ddots & \\ & & -\beta_k f_{k-1} \end{bmatrix} D_k \begin{bmatrix} -\beta_1 f_0 & & \\ & \ddots & \\ & & -\beta_k f_{k-1} \end{bmatrix} \end{aligned}$$

qui est une matrice diagonale. □

Require: x_0 vecteur initial
1: $r_0 = b - Ax_0$
2: $p_0 = r_0$
3: **for** $k = 0,1,\ldots$ **do**
4: $\quad \alpha_k = \frac{r_k^* r_k}{p_k^* A p_k}$
5: $\quad x_{k+1} = x_k + \alpha_k p_k$
6: $\quad r_{k+1} = r_k - \alpha_k A p_k$
7: $\quad \beta_{k+1} = \frac{r_{k+1}^* r_{k+1}}{r_k^* r_k}$
8: $\quad p_{k+1} = r_{k+1} + \beta_{k+1} p_k$
9: **end for**

Algorithme 1.3: Algorithme du Gradient Conjugué

1.3.3 Optimalité du Gradient Conjugué

Les théorèmes suivants montrent que la solution produite par l'algorithme du gradient conjugué est optimale pour la norme A, puis donnent une vitesse asymptotique de convergence, en fonction du conditionnement de la matrice A.

Théorème 1.6 *L'erreur $e_k = x - x_k$ vérifie les relations suivantes*

$$\begin{aligned}
\|e_k\|_A &= \min_{y \in x_0 + \mathcal{K}_k(A,r_0)} \|y - x\|_A \\
\|e_k\|_A &= \min_{q \in \mathbb{P}_{k-1}} \|(I - Aq(A))e_0\|_A \\
&= \min_{\substack{q \in \mathbb{P}_k \\ q(0)=1}} \|q(A)e_0\|_A
\end{aligned}$$

Preuve Tout d'abord $r_k \perp \mathcal{K}_k(A,r_0)$, car r_k est parallèle à u_{k+1}. Le théorème 1.3 permet alors de conclure pour la première égalité.

Puis $\forall y \in x_0 + \mathcal{K}_k(A,r_0), \exists! q \in \mathbb{P}_{k-1}$ tel que $y = x_0 + q(A)r_0$. On a donc

$$\begin{aligned}
A(x - y) &= b - Ax_0 - Aq(A)r_0 \\
&= (I - Aq(A))\, Ae_0,
\end{aligned}$$

d'où la 2$^{\text{e}}$ égalité.

Enfin, quand q décrit $\mathbb{P}_{k-1}$, $(I - Xq)$ décrit $\{p \in \mathbb{P}_k,\, p(0) = 1\}$. On en déduit la dernière égalité. □

Théorème 1.7 *Soit $\chi_2 = \frac{\lambda_n}{\lambda_1}$ le conditionnement de A, alors*

$$\|e_k\|_A \leq 2\|e_0\|_A \left(\frac{\sqrt{\chi_2} - 1}{\sqrt{\chi_2} + 1}\right)^k$$

Preuve On a

$$\begin{aligned}
\|e_k\|_A &= \min_{\substack{q \in \mathbb{P}_k \\ q(0)=1}} \|q(A)e_0\|_A \\
&\leq \min_{\substack{q \in \mathbb{P}_k \\ q(0)=1}} \max_{x \in [\lambda_1, \lambda_n]} |q(x)| \|e_0\|_A \\
&\leq 2\|e_0\|_A \left(\frac{\sqrt{\chi_2} - 1}{\sqrt{\chi_2} + 1}\right)^k.
\end{aligned}$$

Pour la démonstration de la dernière inégalité, on pourra consulter [12]. □

1.3.4 Préconditionnement

Le théorème 1.7 montre que l'efficacité de l'algorithme du gradient conjugué est fortement liée au conditionnement du système. Cela incite à modifier le système à résoudre et le remplacer par un système équivalent, dont le conditionnement est plus favorable au gradient conjugué. Cela s'appelle préconditionner le système [36, 29].

Si on appelle M une matrice symétrique définie positive, approchant A, on peut, par exemple, résoudre un des problèmes suivants

$$M^{-1}Ax = M^{-1}b, \tag{1.17}$$

ou

$$AM^{-1}u = b, \quad x = M^{-1}u. \tag{1.18}$$

Malheureusement dans ce cas, $M^{-1}A$ et AM^{-1} ne sont plus symétriques. On peut préserver cette symétrie, par exemple, si M est disponible sous forme factorisée $M = LL^*$. Alors on sépare la matrice de préconditionnement et on résout

$$L^{-1}AL^{-*}u = L^{-1}b, \quad x = L^{-*}u \tag{1.19}$$

qui est, lui, un système symétrique. Cela n'est pas cependant nécessaire car $M^{-1}A$ est auto-adjoint pour le produit scalaire M. En effet

$$(M^{-1}Ax,y)_M = (Ax,y) = (x,M^{-1}Ay)_M.$$

Et donc, il suffit de résoudre (1.17) avec le produit scalaire M dans l'algorithme du gradient conjugué. L'algorithme devient avec $z_k = M^{-1}r_k$

1: **for** $k = 0,1,\ldots$ **do**
2: $\quad \alpha_k = (z_k,z_k)_M/(M^{-1}Ap_k,p_k)_M$
3: $\quad x_{k+1} = x_k + \alpha_k p_k$
4: $\quad r_{k+1} = r_k - \alpha_k Ap_k$
5: $\quad \beta_{k+1} = (z_{k+1},z_{k+1})_M/(z_k,z_k)_M$
6: $\quad p_{k+1} = r_{k+1} + \beta_{k+1}p_k$
7: **end for**

Il n'est pas nécessaire de calculer les produits scalaires M car

$$\begin{aligned}(z_k,z_k)_M &= (z_k,r_k)\\ (M^{-1}Ap_k,p_k)_M &= (Ap_k,p_k).\end{aligned}$$

Si on injecte ces relations dans l'algorithme du gradient conjugué, on obtient l'algorithme 1.4. Il est appelé algorithme du gradient conjugué préconditionné. Son avantage sur la version non préconditionnée est que la vitesse de convergence dépend maintenant du conditionnement de la matrice $M^{-1}A$.

Sa mise en œuvre nécessite un vecteur intermédiaire supplémentaire et une résolution de système : $z_k = M^{-1}r_k$. Il faut donc choisir M de telle façon que ce surcoût ne soit pas trop important, et pour que le gain soit important, on choisit M telle que $M^{-1}A$ soit le plus proche possible de l'identité. Il existe de nombreuses manières pour choisir M. Dans la partie 1.3.5, on va voir quelques-unes de ces stratégies.

Require: x_0 solution initiale et M préconditionnement
1: $r_0 = b - Ax_0$
2: $z_0 = M^{-1}r_0$
3: $p_0 = z_0$
4: **for** $k = 0,1,\ldots$ **do**
5: $\quad \alpha_k = (z_k,r_k)/(p_k,Ap_k)$
6: $\quad x_{k+1} = x_k + \alpha_k p_k$
7: $\quad r_{k+1} = r_k - \alpha_k Ap_k$
8: $\quad z_k = M^{-1}r_{k+1}$
9: $\quad \beta_{k+1} = (z_{k+1},r_{k+1})/(z_k,r_k)$
10: $\quad p_{k+1} = z_{k+1} + \beta_{k+1}p_k$
11: **end for**

Algorithme 1.4: Algorithme du Gradient Conjugué Préconditionné

1.3.5 Différents préconditionnements

1.3.5.1 Factorisations LU incomplètes

Une première idée de préconditionnement est la factorisation LU. En effet, la résolution de systèmes triangulaires coûte peu. Malheureusement, généralement, cette factorisation mène à un remplissage qui est inacceptable pour de grandes matrices creuses. Pour éviter cela, on fait une factorisation incomplète de A : c'est-à-dire que l'on ne calcule pas tous les éléments de L et U, mais seulement une partie d'entre eux. On calcule alors L et U telles que

$$A = LU - R$$

avec $R = [r_{i,j}]_{i,j}$ une matrice qui a des zéros aux endroits où l'on veut que la factorisation soit exacte. On appelle patron de factorisation, l'ensemble $\mathcal{P}$ des indices (i,j) où $r_{i,j}$ est nul. Pour réaliser cette factorisation incomplète suivant un patron $\mathcal{P}$, on utilise l'algorithme 1.5 dérivé de l'algorithme de Gauss pour la factorisation LU.

Malheureusement, cet algorithme ne termine pas toujours. On peut par exemple tomber sur un pivot nul. Cependant sous certaines hypothèses [36], la factorisation est toujours possible.

On appelle une M-matrice, une matrice non-singulière telle que

- $a_{i,i} > 0$ pour $i = 1,\ldots,n$
- $a_{i,j} \leq 0$ pour $i = 1,\ldots,n$, $j = 1,\ldots,n$ et $i \neq j$
- $A^{-1} \geq 0$.

On a alors le théorème suivant.

Théorème 1.8 *Soit A une M-matrice et $\mathcal{P}$ un patron de factorisation, alors l'algorithme 1.5 est faisable et produit une factorisation incomplète de A régulière. C'est à dire que l'on a*

$$A = LU - R$$

Require: $\mathcal{P}$ patron de factorisation
1: **for** $k = 1, \ldots, n-1$ **do**
2: **for** $i = k+1, \ldots, n$ et $(i,k) \notin \mathcal{P}$ **do**
3: $a_{i,k} = \frac{a_{i,k}}{a_{k,k}}$
4: **for** $j = k+1, \ldots, n$ et $(i,k) \notin \mathcal{P}$ **do**
5: $a_{i,j} = a_{i,j} - a_{i,k}a_{k,j}$
6: **end for**
7: **end for**
8: **end for**

Algorithme 1.5: Factorisation LU incomplète

avec LU inversible.

Preuve Pour la preuve, se référer à [28]. □

Maintenant se pose le problème de la stratégie de remplissage, c'est-à-dire le choix de $\mathcal{P}$. Le choix le plus naturel est de prendre pour $\mathcal{P}$, le même schéma de remplissage que A, la diagonale en sus. Cette factorisation est appelée ILU(0).

Cependant la précision d'une telle factorisation peut être insuffisante. Pour l'améliorer, on autorise alors un certain remplissage. On peut choisir de prendre le même remplissage que celui du produit LU où L et U sont les matrices obtenues avec la factorisation ILU(0). C'est la factorisation ILU(1). Malheureusement cela ne se généralise pas directement. Pour avoir différent niveaux de remplissage, on définit un niveau initial pour chaque élément de A, par

$$lev_{i,j} = \begin{cases} 0 & \text{si } a_{i,j} \neq 0 \text{ ou si } i = j \\ \infty & \text{sinon.} \end{cases}$$

Puis on met à jour les niveaux pendant l'algorithme d'élimination de Gauss de la façon suivante

$$lev_{i,j} = \min\{lev_{i,j}, lev_{i,k} + lev_{k,j} + 1\}.$$

Les indices i, j et k sont les mêmes que lors de l'étape $a_{i,j} = a_{i,j} - a_{i,k}a_{k,j}$ de l'algorithme 1.5. En séparant la factorisation symbolique de la partie numérique, on obtient l'algorithme 1.3.5.1, pour calculer le niveau de chaque élément. Pour la factorisation ILU(p), on garde tous les éléments dont le niveau est inférieur à p. C'est-à-dire $\mathcal{P} = \{(i,j),\ lev_{i,j} > p\}$.

Cet algorithme de factorisation peut malheureusement être très coûteux. De plus il garde les éléments uniquement sur des critères structurels et non sur leur valeur. Il peut ainsi conduire à écarter des éléments de grande valeur de la factorisation. Pour éviter cela, on peut baser la stratégie de remplissage sur la valeur des éléments de L et U.

1: Pour tous les éléments non nuls de A, $lev_{i,j} = 0$
2: **for** $i = 2, \ldots, n$ **do**
3: **for** $k = 1, \ldots, i-1$ et $lev_{i,k} \leq p$ **do**
4: $lev_{i,j} = \min\{lev_{i,j}, lev_{i,k} + lev_{k,j} + 1\}$
5: **end for**
6: **end for**

Algorithme 1.6: Calcul des niveaux de remplissage pour ILU(p)

Pour cela, on reprend l'algorithme de Gauss, et pour chaque élément, au cours de la factorisation, on applique une règle de sélection : si l'élément ne satisfait pas une certaine règle, on ne le garde pas : c'est-à-dire, on le remplace par zéro. Généralement, on garde l'élément s'il est plus grand qu'un seuil de tolérance τ. Enfin pour chaque ligne calculée, on peut encore appliquer une règle de sélection. Cela permet, par exemple, de contrôler le nombre d'éléments en ne gardant que les λ plus grands, par exemple. Cela donne la factorisation ILUT(λ,τ).

1.3.5.2 Approche de l'inverse

Toutes ces stratégies essayent d'approcher A par LU au sens où l'erreur $-R = A - LU$ doit être «petite». Mais dans les algorithmes préconditionnés, c'est $L^{-1}AU^{-1} = I - L^{-1}RU^{-1}$ qui est utilisée, et donc il faut que $L^{-1}RU^{-1}$ soit petite, sinon le préconditionnement ne fera pas son office [10]. Si L et U sont bien conditionnées alors si R est petite, $L^{-1}RU^{-1}$ le sera aussi raisonnablement. C'est le cas si par exemple A est à diagonale dominante.

Si ce n'est pas le cas, on peut chercher à approcher A^{-1} directement pour ne pas souffrir du mauvais conditionnement possible de la factorisation ILU. C'est-à-dire trouver une matrice M telle que $\|I - AM\|_F$ soit petite. Cela est détaillé dans [11] par exemple.

1.3.5.3 Autres préconditionnements

On peut aussi utiliser les matrices définies dans les méthodes de point fixe, par exemple dans SOR. Malheureusement, cette matrice n'est pas symétrique. Pour la rendre symétrique, on peut faire la mise à jour dans l'algorithme une fois dans un sens (de 1 à n), une fois dans l'autre sens (de n à 1). Cela consiste à faire une itération de SOR avec les matrices M_ω et N_ω, puis une autre, immédiatement, avec leurs transposées M_ω^* et N_ω^*. On obtient

$$\begin{aligned} M &= M_\omega D^{-1} M_\omega^* \\ N &= N_\omega D^{-1} N_\omega^*. \end{aligned}$$

C'est la méthode SSOR (symmetric successive over-relaxation). Pour l'utiliser en préconditionnement, on réalise une étape, ou N étapes, de l'algorithme SSOR, lors de

la phase de préconditionnement, comme si on cherchait à résoudre $Az_k = r_k$.

On peut aussi utiliser un polynôme en A comme préconditionnement. En effet, il existe un polynôme Δ, tel que $\Delta(A) = A^{-1}$. En utilisant une approximation $\tilde{\Delta}$ de ce polynôme, on obtient une matrice de préconditionnement avec $\tilde{\Delta}(A)$. Ces préconditionnements font partie des préconditionnements polynômiaux [29, 5].

1.4 Conclusion

Ce chapitre présente une partie des algorithmes de résolution de systèmes linéaires. Ce sont ces algorithmes que l'on compare dans le chapitre deux. Par la suite, on modifie les algorithmes de Krylov. Dans le chapitre trois, le but est une régularisation de la solution ; dans le chapitre quatre, c'est l'accélération de la convergence du gradient conjugué, qui est visée.

Il existe d'autres algorithmes, mais ils ne sont pas traités ici. Ce sont par exemple, les méthodes multi-grilles [29] et les méthodes par blocs [33]. Il existe aussi de nombreux autres préconditionnements [5, 37, 39].

Chapitre 2

Comparaisons de solveurs linéaires, pour le problème de la restauration d'images

Alors que le premier chapitre présentait formellement les solveurs linéaires classiques, ce second montre leur comportement lors de tests numériques. Le problème choisi pour effectuer les comparaisons ente les solveurs est issu de l'analyse d'images : il s'agit de restaurer une image bruitée. Le modèle le plus simple, dit modèle quadratique, n'est pas satisfaisant d'un point de vue qualité de l'image restaurée. C'est pourquoi, pour les tests, on a choisi un modèle plus fin : celui des estimateurs robustes.

La première partie de ce chapitre présente ces deux modèles ; la seconde détaille la mise en œuvre des méthodes linéaires utilisées. Enfin les résultats de ces tests sont consignés dans la troisième partie.

2.1 Présentation des modèles

2.1.1 Le modèle quadratique

Le modèle de base de la restauration, dit modèle quadratique, peut s'exprimer ainsi : soit b une image (bruitée) de taille $n \times n$. On pose $N = n^2$, alors $b \in \mathbb{R}^N$ et soit α un paramètre réel positif, l'image restaurée est l'image minimisant la fonction d'énergie suivante :

$$H(x) = \|x - b\|_2^2 + \alpha \|Dx\|_2^2 \tag{2.1}$$

avec D un opérateur linéaire, généralement un opérateur de lissage. Pour les tests, D sera toujours l'opérateur de lissage du premier ordre : c'est-à-dire que Dx est le vecteur des écarts dans l'image x. C'est un vecteur de taille $2N(N-1)$.

On peut voir dans cette fonction d'énergie H, deux potentiels $\|x-b\|^2$ et $\|Dx\|^2$. Le premier signifie que l'image restaurée doit être proche de l'image bruitée. C'est normal car le bruit est considéré comme une perturbation de l'image originelle. Le second terme

impose à l'image restaurée une certaine régularité. En effet plus l'image est lisse, plus ce potentiel est bas ; il est minimal pour une image uniforme.

L'image x minimisant l'énergie H, est aussi solution de l'équation suivante :

$$(I + \alpha D^* D)x = b. \tag{2.2}$$

Donc pour restaurer l'image b, il suffit de résoudre le système 2.2, système linéaire à matrice symétrique définie positive.

2.1.2 La structure de $I + \alpha D^* D$

La matrice D^*D associe à tout point de l'image, la somme des écarts à ses voisins horizontaux et verticaux ; c'est-à-dire avec une notation à double indice et pour les points intérieurs de l'image :

$$\begin{aligned} x_{i,j} &\mapsto x_{i,j} - x_{i+1,j} \\ &+ x_{i,j} - x_{i-1,j} \\ &+ x_{i,j} - x_{i,j-1} \\ &+ x_{i,j} - x_{i,j+1} \\ &= 4x_{i,j} - x_{i+1,j} - x_{i-1,j} - x_{i,j-1} - x_{i,j+1}. \end{aligned}$$

Ou avec une notation à un seul indice :

$$x_k \mapsto 4x_k - x_{k-n} - x_{k-1} - x_{k+1} - x_{k+n}.$$

Pour les points du bord, on ne fait la somme que pour les voisins existants. C'est-à-dire pour les coins, la somme devient $2x_k - x_{v1} - x_{v2}$ et sur les autres points de la frontière $3x_k - x_{v1} - x_{v2} - x_{v3}$; $v1$, $v2$ et $v3$ étant les indices des voisins existants. La matrice $I + \alpha D^* D$ est donc de la forme suivante :

$$I + \alpha D^* D = \begin{pmatrix} A_0 & -\alpha I_n & & & \\ -\alpha I_n & A_1 & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & A_1 & -\alpha I_n \\ & & & -\alpha I_n & A_0 \end{pmatrix},$$

où A_0 et A_1 sont de dimension n et de la forme :

$$A_0 = \begin{pmatrix} 1+2\alpha & -\alpha & & & \\ -\alpha & 1+3\alpha & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & 1+3\alpha & -\alpha \\ & & & -\alpha & 1+2\alpha \end{pmatrix},$$

$$A_1 = \begin{pmatrix} 1+3\alpha & -\alpha & & & \\ -\alpha & 1+4\alpha & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & 1+4\alpha & -\alpha \\ & & & -\alpha & 1+3\alpha \end{pmatrix}.$$

On peut alors remarquer que le coefficient de la diagonale est 1 plus α fois le nombre de voisins du pixel x_k. Ce qui incite à introduire la fonction nv, égale au nombre de voisins du pixel :

$$nv(x_k) = \begin{cases} 2 & \text{si } x_k \text{ est dans un coin} \\ 3 & \text{si } x_k \text{ est sur un bord} \\ 4 & \text{sinon.} \end{cases} \tag{2.3}$$

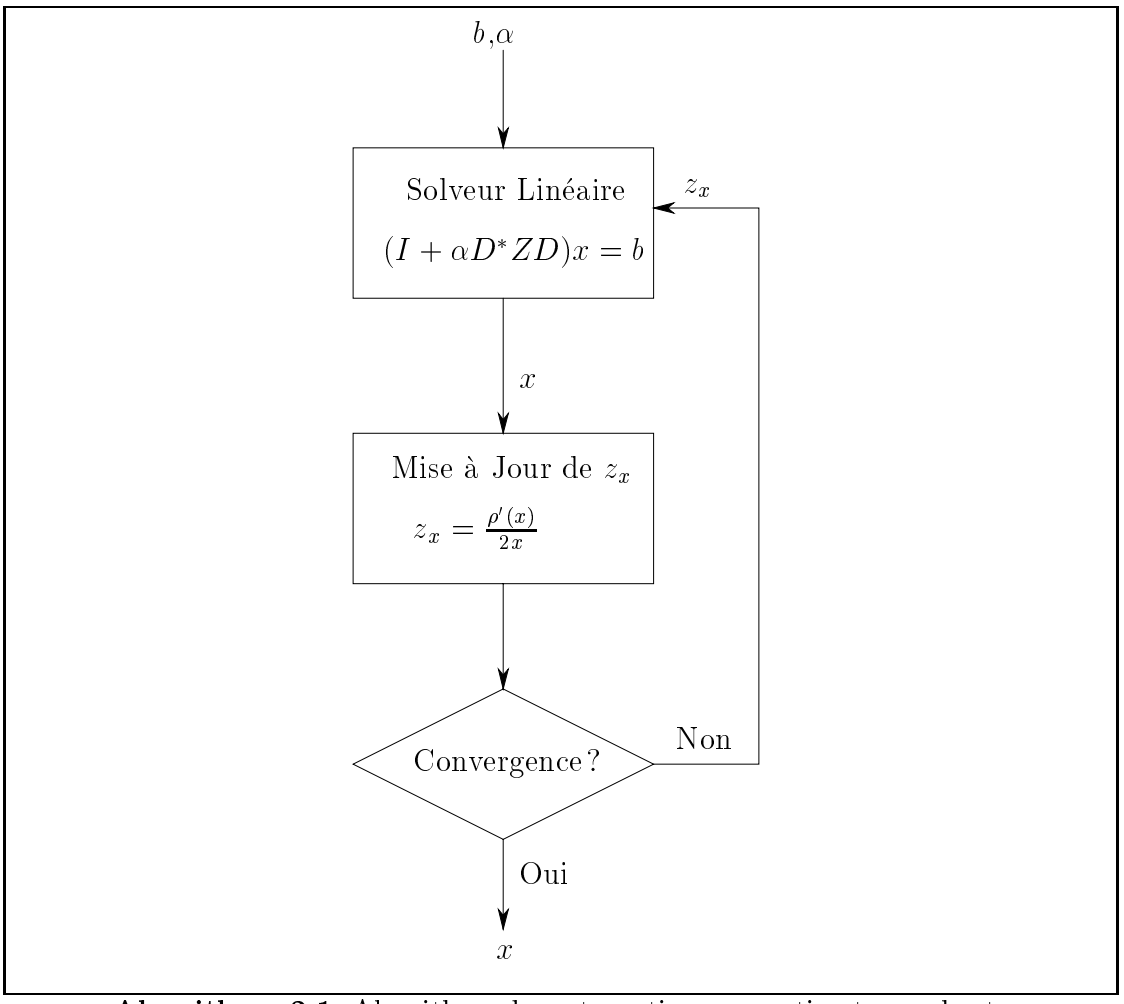
De même, on utilise la fonction sv qui vaut la somme des voisins du pixel :

$$sv(x_k) = \begin{cases} x_2 + x_{n+1} & \text{si } k = 1 \\ x_{n-1} + x_{2n} & \text{si } k = n \\ x_{n^2-2n+1} + x_{n^2-n+2} & \text{si } k = n^2 - n + 1 \\ x_{n^2-n} + x_{n^2-1} & \text{si } k = n^2 \\ x_{k-1} + x_{k+1} + x_{k+n} & \text{si } 2 \leq k \leq n-1 \\ x_{k-n} + x_{k-1} + x_{k+1} & \text{si } n^2 - n + 1 \leq k \leq n^2 - 1 \\ x_{k-n} + x_{k+1} + x_{k+n} & \text{si } k = n+1, 2n+1, \ldots, n^2 - 2n + 1 \\ x_{k-n} + x_{k-1} + x_{k+n} & \text{si } k = 2n, 3n, \ldots, n^2 - n \\ x_{k-n} + x_{k-1} + x_{k+1} + x_{k+n} & \text{sinon.} \end{cases} \tag{2.4}$$

Ces deux fonctions nv et sv sont très importantes car elles permettent d'exprimer toutes les opérations matricielles sans faire intervenir explicitement les matrices. De plus, elles cachent les effets de bord, car tous les pixels sont alors traités de la même façon. Enfin, les formules utilisant nv et sv, se généralisent très bien — elles ne changent pas — pour le modèle des estimateurs robustes, en utilisant des pondérations.

2.1.3 Les estimateurs robustes

La régularisation quadratique a le désavantage de lisser les discontinuités et les contours de l'image originelle. Pour prévenir cela, une régularisation respectant les contours a été récemment introduite en analyse d'image [16, 3, 9, 30]. Elle est caractérisée par la substitution de la norme quadratique classique de la régularisation de Tikhonov [42, 41] par une fonction d'erreur, robuste. Il est démontré dans [9] que le terme de régularisation robuste peut être écrit comme une somme pondérée de potentiels quadratiques. Ces poids sont alors de nouvelles variables reliées aux discontinuités spatiales du problème.



Algorithme 2.1: Algorithme de restauration avec estimateur robuste

Le problème robuste peut se formuler ainsi:

$$\text{argmin} H(x) \tag{2.5}$$

avec H une fonction d'énergie de la forme

$$H(x) = \|x - b\|^2 + \alpha \sum_{(k,j)\in\mathcal{C}} \rho(\|x_k - x_j\|) \tag{2.6}$$

où α est une constante positive et $\mathcal{C}$ un ensemble de cliques. Une clique est un ensemble de pixels voisins pour un système de voisinage. Ici ce sera toujours le système à 4 voisins, et on indexera toutes les paires de sommets par i. On note aussi $g_i(x) = \|x_k - x_j\|$ pour i représentant la paire de sommets(k,j). Ce qui permet de récrire l'énergie ainsi

$$H(x) = \|x - b\|^2 + \alpha \sum_{i\in\mathcal{C}} \rho(g_i(x)). \tag{2.7}$$

La fonction ρ est un M-estimateur [25], ce qui permet de traiter le problème des discontinuités de l'image. Cette fonction ρ est continue, paire et croissante sur $\mathbb{R}^+$ et croît moins vite qu'une fonctionnelle quadratique à l'infini pour moins pénaliser les discontinuités que celle-ci.

Pour résoudre le problème 2.5, on transforme celui-ci en une suite de problèmes quadratiques en introduisant des variables auxiliaires. Pour cela, on utilise le théorème suivant qui montre que le graphe de ρ est l'enveloppe inférieure d'une famille de paraboles, continuement indexées par $z \in]0,1]$.

Théorème 2.1 *Soit ρ une fonction paire de $\mathbb{R}$ dans $\mathbb{R}$, continuement dérivable, telle que :*

1. *ρ est croissante sur $\mathbb{R}^+$;*
2. *$\phi(v) = \rho(\sqrt{v})$ est strictement concave sur $\mathbb{R}^+$;*
3. *$\lim_{v\to\infty} \phi'(v) = 0$;*
4. *$\tau = \lim_{v\to 0^+} \phi'(v) < +\infty$.*

Alors, il existe une fonction ψ strictement convexe, continuement dérivable sur $]0,1]$ telle que

$$\forall u \in \mathbb{R}^+,\ \rho(u) = \min_{z\in]0,1]} \left(\tau z u^2 + \psi(z)\right). \tag{2.8}$$

Preuve voir [16, 3]. □

Le minimum dans (2.8) est alors donné par

$$\arg\min_{z\in]0,1]} \left(\tau z u^2 + \psi(z)\right) = \frac{\rho'(u)}{2\tau u}. \tag{2.9}$$

On peut alors remplacer la minimisation de $\|x-b\|^2 + \alpha \sum_{i\in\mathcal{C}} \rho(g_i(x))$, par la minimisation en $(x,\{z_i\})$ de

$$\|x-b\|^2 + \alpha \sum_{i\in\mathcal{C}} \left(\tau z_i g_i(x)^2 + \psi(z_i)\right), \tag{2.10}$$

car les deux sommes ont le même minimum global. Ces z_i sont de nouvelles variables, appelées poids adaptatifs. Il y a un coefficient par paire de pixels voisins. On appelle z le vecteur des z_i. On a alors :

$$H(x) = \min_z H^*(x,z) \tag{2.11}$$

avec

$$H^*(x,z) = \|x-b\|^2 + \alpha \sum_{i\in\mathcal{C}} \left(\tau z_i g_i(x)^2 + \psi(z_i)\right). \tag{2.12}$$

C'est-à-dire en notation matricielle

$$H^*(x,z) = x^*(I + \alpha\tau D^* Z D)x - 2x^*b + b^*b + \alpha \sum_{i\in\mathcal{C}} \psi(z_i) \tag{2.13}$$

avec Z la matrice diagonale contenant les z_i. On fait alors le changement d'échelle suivant $\alpha\tau \to \alpha$ et la fonction H^* devient

$$H^*(x,z) = x^*(I + \alpha D^* Z D)x - 2x^* b + c \tag{2.14}$$

où c est une constante. Le problème initial (2.5) est alors devenu le suivant :

$$\arg\min_{x,z} H^*(x,z). \tag{2.15}$$

La résolution de ce nouveau problème (2.15) se fait alors par minimisation alternée : on fixe une variable x ou z, et on minimise par rapport à l'autre. À z fixé, la minimisation est celle d'une fonctionnelle quadratique : elle est résolue par un solveur linéaire classique. À x fixé, la minimisation se fait grâce à la formule (2.9).

Cela s'appelle la méthode des moindres carrés pondérés itérés [24] ; en anglais *iterated reweighted least squares* (IRLS). Le shéma 2.1 résume cet algorithme.

2.2 La mise en œuvre

2.2.1 Opérations matricielles

Toutes les versions des algorithmes utilisés devront être «matrix-free». C'est-à-dire, qu'ils n'utilisent pas la matrice $A = I + \alpha D^* D$ ou $A = I + \alpha D^* Z D$, mais le résultat de son produit avec un vecteur. Pour les algorithmes de relaxation, c'est la matrice du solveur qui ne sera pas formée.

La structure de la matrice $I + \alpha D^* D$ est explicitée dans la partie 2.1.2. Celle de la matrice $A = I + \alpha D^* Z D$ utilisée pour la résolution avec IRLS, est exactement la même, et les valeurs sont simplement pondérées par un z_i. Ce qui veut dire que la fonction nv sera maintenant la somme des quatre coefficients pondérateurs situés entre x_k et ses voisins, et la fonction sv est la somme des valeurs des pixels voisins pondérée par les z_i correspondants.

Grâce à ces deux fonctions, on exprime la diagonale de A comme le vecteur

$$[1 + \alpha nv(x_k)]_{k=1\ldots n^2}$$

et le produit de A par x. On a en effet :

$$[Ax]_k = (1 + \alpha\ nv(x_k))x_k\ -\ \alpha\ sv(x_k). \tag{2.16}$$

Donc pour le gradient conjugué, la multiplication par la matrice $I + \alpha D^* D$ — ou par $I + \alpha D^* Z D$ dans le cas des estimateurs robustes — se fait grâce aux deux fonctions nv et sv.

2.2.2 Les méthodes de relaxation

De même, on exprime les matrices d'itération des méthodes de relaxation décrite dans la partie 1.1.1 page 5, de façon à ne plus faire intervenir ces matrices lors des

calculs. Pour la méthode de Jacobi, on a besoin de

$$\sum_{\substack{j=1 \\ j \neq i}}^{n} a_{i,j}[x]_j$$

où $a_{i,j}$ est le coefficient i,j de la matrice du système. Cette somme vaut α fois la somme des voisins du pixel, et donc la nouvelle valeur du pixel est

$$\frac{b_k + \alpha\ sv(x_k)}{1 + \alpha\ nv(x_k)}.$$

Pour l'algorithme de Gauss-Seidel, on utilise la même somme que pour l'algorithme de Jacobi. Cependant, les valeurs des pixels à gauche et au-dessus sont les valeurs déjà mises à jour. C'est-à-dire que, alors que pour l'algorithme de Jacobi, on devait faire la mise à jour dans une nouvelle variable, pour l'algorithme de Gauss-Seidel, on fait la mise à jour en place ; on met à jour les valeurs des pixels pour k allant de 1 à n^2. Dans ce cas, les valeurs x_{k-n}, valeur du pixel de gauche, et x_{k-1}, valeur du pixel d'au-dessus, ont déjà été modifiées, alors que les deux autres, pas encore. L'itération devient donc :

$$\text{pour } k = 1..n^2 \quad x_k = \frac{b_k + \alpha\ sv(x_k)}{1 + \alpha\ nv(x_k)}. \tag{2.17}$$

Pour la méthode SOR, on calcule la mise à jour grâce à la valeur de l'itération précédente $x^{(k)}$ et la valeur produite par la méthode de Gauss-Seidel $x_{\mathsf{GS}}^{(k+1)}$:

$$x^{(k+1)} = (1-\omega)x^{(k)} + \omega x_{\mathsf{GS}}^{(k+1)}. \tag{2.18}$$

Enfin, pour la méthode SSOR, on alterne le sens de la mise à jour : une fois de 1 à n^2, la suivante de n^2 à 1. Donc pour la première itération, les pixels de gauche et d'au-dessus ont déjà été modifiés, et pour la seconde, ce sont ceux de droite et d'en-dessous.

2.2.3 Les préconditionnements

Pour le préconditionnement diagonal, il s'agit de diviser par la matrice diagonale, dont les éléments sont ceux de la diagonale principale de A, c'est-à-dire $1+\alpha\ nv(x_k)$. Il suffit donc, pour la mise en œuvre du préconditionnement diagonal, de faire une passe pour diviser x_k par $1 + \alpha\ nv(x_k)$, pour tous les pixels de l'image.

Pour la factorisation de Cholesky incomplète IC(0), il s'agit de factoriser $A \approx LL^*$, sans remplissage : on ne calcule que des termes déjà présents dans A. La matrice L ne contient que la diagonale principale, la 1^{re} et la n^{e} sous-diagonale. Les termes de L sont les suivants :

– pour la $1^{\text{ère}}$ sous-diagonale :

$$L_{k,k-1} = -\frac{\alpha}{L_{k-1,k-1}},$$

– pour la $n^{\text{ème}}$ sous-diagonale:

$$L_{k,k-n} = -\frac{\alpha}{L_{k-n,k-n}},$$

– pour la diagonale principale:

$$\begin{aligned} L_{k,k} &= \sqrt{A_{k,k} - \frac{\alpha^2}{L_{k-1,k-1}^2} - \frac{\alpha^2}{L_{k-n,k-n}^2}} \\ &= \sqrt{(1+\alpha\ nv(x_k)) - \frac{\alpha^2}{L_{k-1,k-1}^2} - \frac{\alpha^2}{L_{k-n,k-n}^2}}. \end{aligned}$$

Le préconditionnement effectif se fait donc de la manière suivante: on ne stocke que la diagonale de L dans un vecteur l, c'est-à-dire que l'espace mémoire nécessaire est de la taille d'une image. Les valeurs des sous-diagonales étant obtenues immédiatement par calcul, lorsqu'on en a besoin. Puis on met à jour l'image en deux passes: de haut en bas puis de bas en haut. Ce qui donne:

– pour $k = 1..n^2$

$$l_k = \sqrt{(1+\alpha\ nv(x_k)) - \frac{\alpha^2}{l_{k-1}^2} - \frac{\alpha^2}{l_{k-n}^2}},$$

– pour $k = 1..n^2$

$$x_k = \frac{1}{l_k}\left(x_k + \frac{\alpha}{l_{k-n}}x_{k-n} + \frac{\alpha}{l_{k-1}}x_{k-1}\right),$$

– pour $k = n^2..1$

$$x_k = \frac{1}{l_k}\left(x_k + \frac{\alpha}{l_{k+n}}x_{k+n} + \frac{\alpha}{l_{k+1}}x_{k+1}\right).$$

2.2.4 Le test d'arrêt

Le test d'arrêt est le même pour tous les algorithmes: dès que moins de 3% de l'image est modifié lors d'une itération, l'algorithme s'arrête. C'est-à-dire, que pour chaque itération, on compte le nombre de pixels modifiés: un pixel est considéré comme modifié si la partie entière de sa valeur change. Si une itération change moins de 3% des pixels de l'image, alors on considère que la convergence s'est faite.

Pour l'algorithme non linéaire, le test est le même: si la résolution du système linéaire modifie moins de 3% des pixels, alors l'algorithme s'arrête.

2.3 Les résultats numériques

2.3.1 Déroulement des tests

Les premiers tests comparent les méthodes de relaxation et aussi plusieurs ω pour SOR. Puis on compare les différents préconditionnements pour le gradient conjugué.

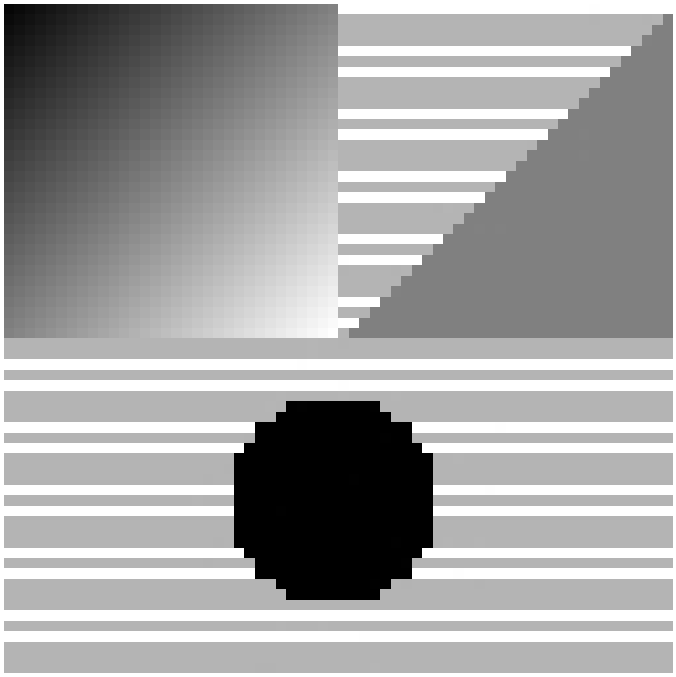


figure 2.1 – *Mire de test*

Enfin, on compare SOR au gradient conjugué préconditionné par une factorisation de Cholesky incomplète, car dans leur catégorie, ces deux algorithmes ont donné les meilleurs résultats.

Pour tous les tests, on a créé un ensemble de 20 images de tests, qui sont des quadrillages plus ou moins fins et des mires circulaires. Ces images sont en 256 niveaux de gris et de taille 256 par 256. La figure 2.1 représente un exemple de mire typique. À toutes ces images, on a ajouté un bruit blanc ayant une amplitude de 20 niveaux de gris. Ce sont ces images bruitées que l'on essaie de restaurer grâce aux modèles présentés.

L'estimateur robuste utilisé pour les tests est celui de Geman-Mac Clure [16]. La fonction ρ vaut :

$$\rho(x) = \frac{x^2}{\sigma + x^2},$$

et on a dans ce cas :

$$z_x = \frac{\sigma}{(\sigma + x^2)^2}.$$

2.3.2 Test des méthodes de relaxation

Le premier test (figures 2.2, 2.3 et 2.4) est l'étude de la convergence pour le premier système linéaire : tous les z_x sont fixés à 1 et il n'y a pas de remise à jour. On peut tout

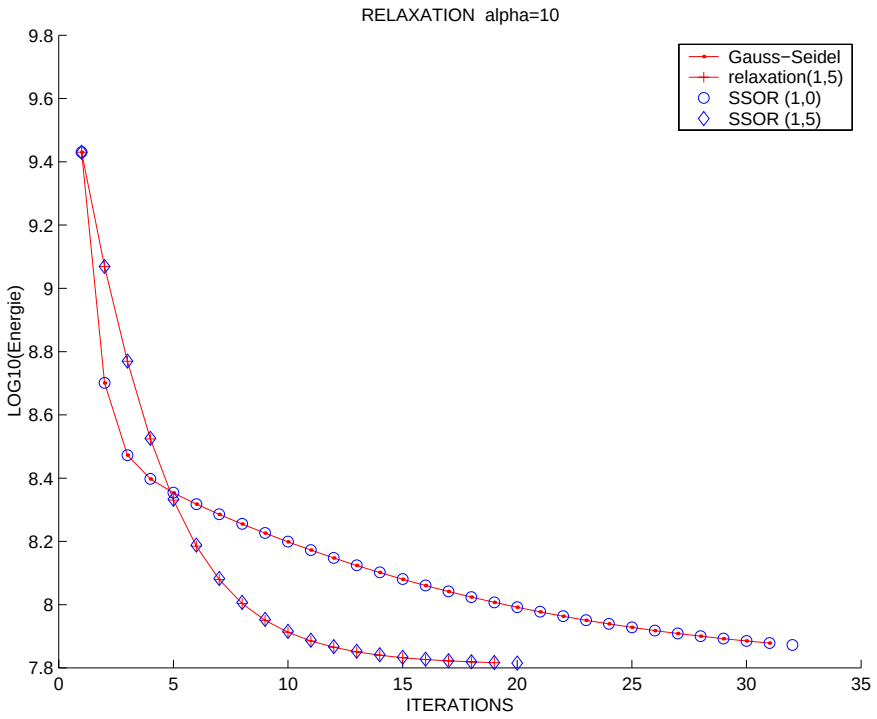


FIG. 2.2 – *Convergence des méthodes de relaxation pour* $\alpha = 10$

d'abord remarquer la concordance entre SOR et SSOR, ce qui fait que, par la suite on n'utilisera plus que SOR pour les tests.

De plus, on peut voir que le taux de convergence très bon au début, chute vers la fin. C'est pourquoi pour la résolution complète non-linéaire, on utilise un critère d'arrêt du solveur linéaire sur le nombre d'itérations : après 3 itérations on stoppe le solveur et on met à jour tous les z_x. La perte de précision du solveur linéaire est contrebalancée par le gain en temps, ce qui permet de faire beaucoup plus d'itérations non linéaires.

Pour les tests complets, ce sont ces itérations non linéaires que l'on comptabilise, car elles ont un coût fixe : 3 itérations du solveur linéaire. Le tableau 2.5 donne le nombre moyen d'itérations sur l'ensemble des images de test. On voit que SOR, avec le bon ω, est toujours plus performante que ses concurrents et que plus le système est mal-conditionné, c'est-à-dire plus α est grand, plus le gain est important. C'est pourquoi, par la suite, pour les tests ente les méthodes de relaxation et celles de projection, on utilise SOR pour représenter sa classe.

2.3.3 Comparaison avec le gradient conjugué

De même que pour la relaxation, on teste tout d'abord le comportement des algorithmes de préconditionnement sur une seule itération non linéaire (figures 2.6, 2.7 et

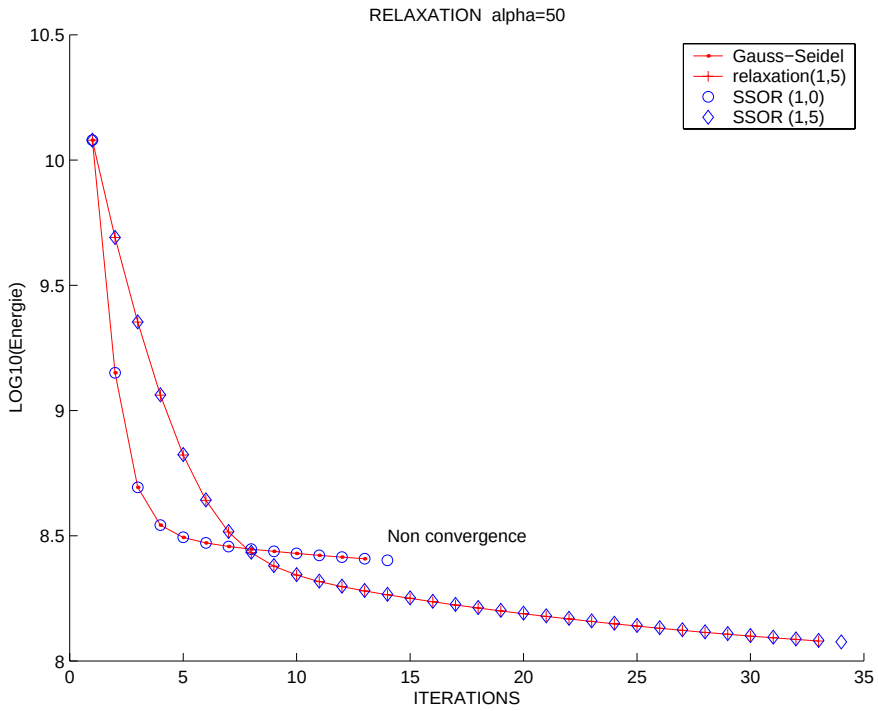


FIG. 2.3 – *Convergence des méthodes de relaxation pour* $\alpha = 50$

2.8). Trois préconditionnements sont testés pour les mêmes images que ci-dessus : pas de préconditionnement, préconditionnement diagonal et préconditionnement par factorisation de Cholesky incomplète sans remplissage. Le solveur linéaire étant toujours le gradient conjugué.On y voit la supériorité du préconditionnement de Cholesky.

2.3.4 Comparaison entre SOR et le gradient conjugué

On compare maintenant les deux algorithmes qui sont les plus performants pour les tests précédents, *ie* SOR et le gradient conjugué préconditionné par la factorisation incomplète.

Le premier test (figure 2.9) montre le nombre d'itérations nécessaires pour atteindre la convergence du premier système, pour les différentes méthodes, en fonction du paramètre α ; c'est-à-dire que pour ce test le modèle de restauration est quadratique. Les images de tests sont ici de petites images de taille 64 par 64. On voit que le gradient conjugué a besoin de deux fois moins d'itérations pour converger. Cependant, en temps de calculs, une itération du gradient conjugué préconditionné, coûte deux fois plus cher qu'une de SOR, dans le cas présent ; ce qui fait que, au coût total, en temps CPU, les deux algorithmes sont équivalents, les images produites étant de facture similaire.

Le second test (figure 2.10) correspond à la résolution pour une grande image de

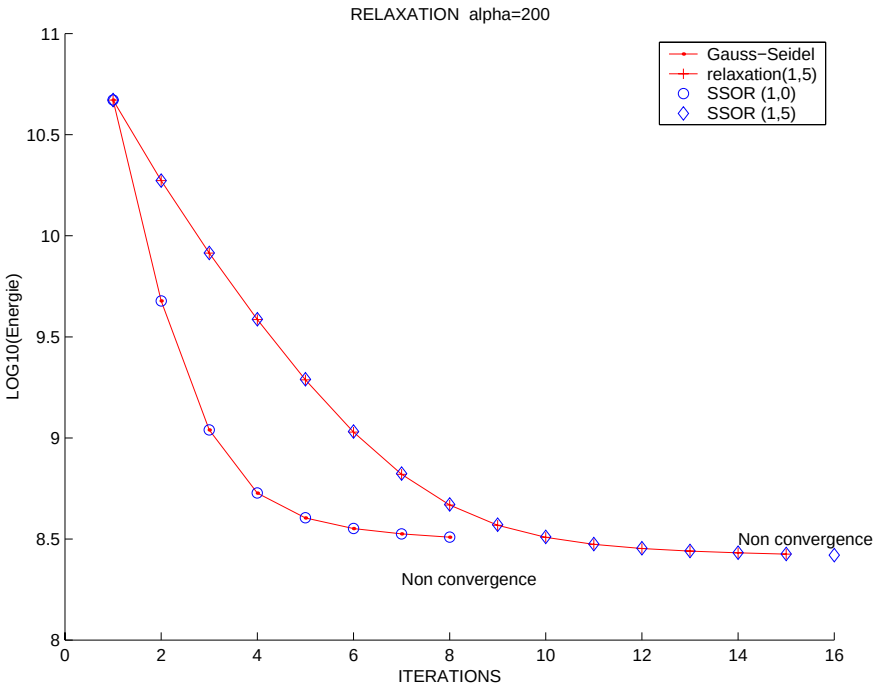


FIG. 2.4 – *Convergence des méthodes de relaxation pour* $\alpha = 200$

taille 1024 par 1024. On calcule, en secondes, le temps CPU nécessaire à la restauration de l'image avec différents α. On voit que dès que le problème augmente en conditionnement, le gradient conjugué est bien meilleur que SOR, dont la convergence se dégrade vite pour les grandes valeurs de α. À partir de $\alpha = 50$, SOR s'arrête sur le critère du nombre d'itérations maximales mais n'a pas convergé. C'est la valeur indiquée par le symbole ‡ dans le tableau 2.10.

2.4 Conclusion

Pour être complets, ces tests devraient inclure d'autres préconditionnements pour le gradient conjugué, comme le préconditionnement SOR ou multigrilles et ils devraient aussi tester une autre catégorie de solveurs : les multigrilles [29]. Les conclusions ne peuvent donc être que partielles, mais montrent assez clairement les comportements du gradient conjugué et de SSOR : la stabilité du premier dont les performances restent similaires pour différentes tailles et différents conditionnements de problèmes, alors que le second n'arrive plus à résoudre convenablement le problème dès que celui-ci devient trop grand ou trop mal conditionné.

Méthode	Nb d'itérations moyen $\alpha = 1$	Nb d'itérations moyen $\alpha = 10$	Nb d'itérations moyen $\alpha = 100$
Jacobi	12.6	58.0	112.8
Gauss-Seidel	11.4	44.8	108.6
Relaxation $\omega = 1.2$	10.4	42.7	109.2
Relaxation $\omega = 1.4$	**9.9**	40.0	99.4
Relaxation $\omega = 1.6$	12.1	**37.7**	92.9
Relaxation $\omega = 1.8$	17.4	42.1	**70.2**

FIG. 2.5 – *Convergence globale pour les méthodes de relaxation*

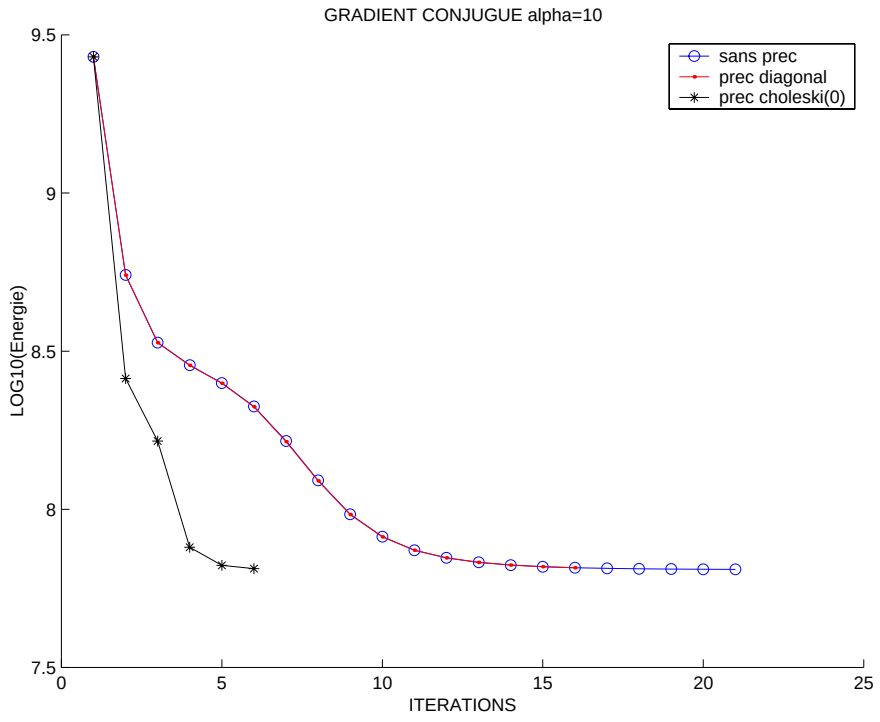


FIG. 2.6 – *Convergence du gradient conjugué préconditionné pour* $\alpha = 10$

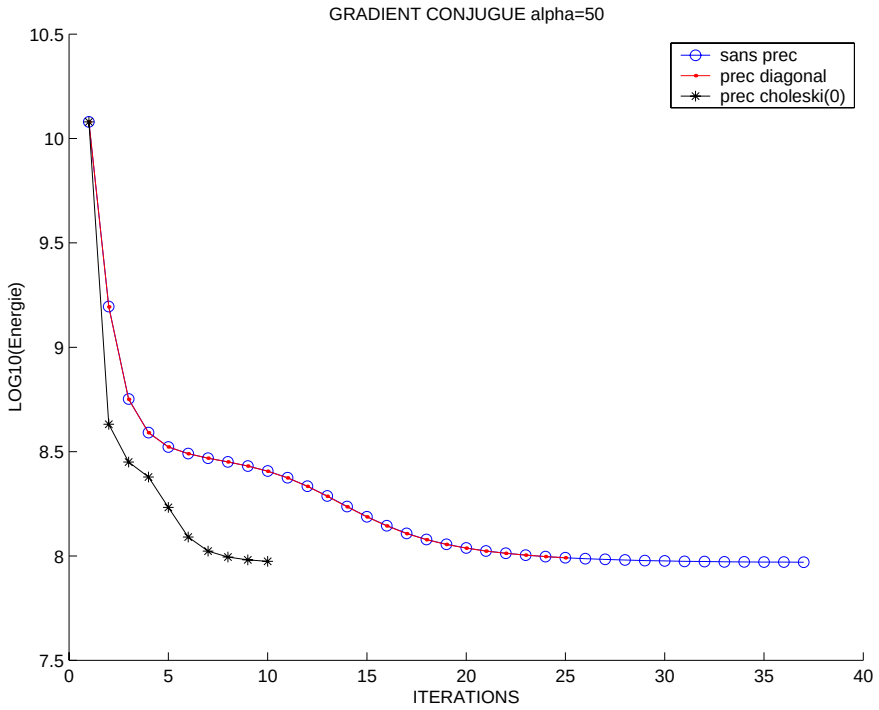


FIG. 2.7 – *Convergence du gradient conjugué préconditionné pour* $\alpha = 50$

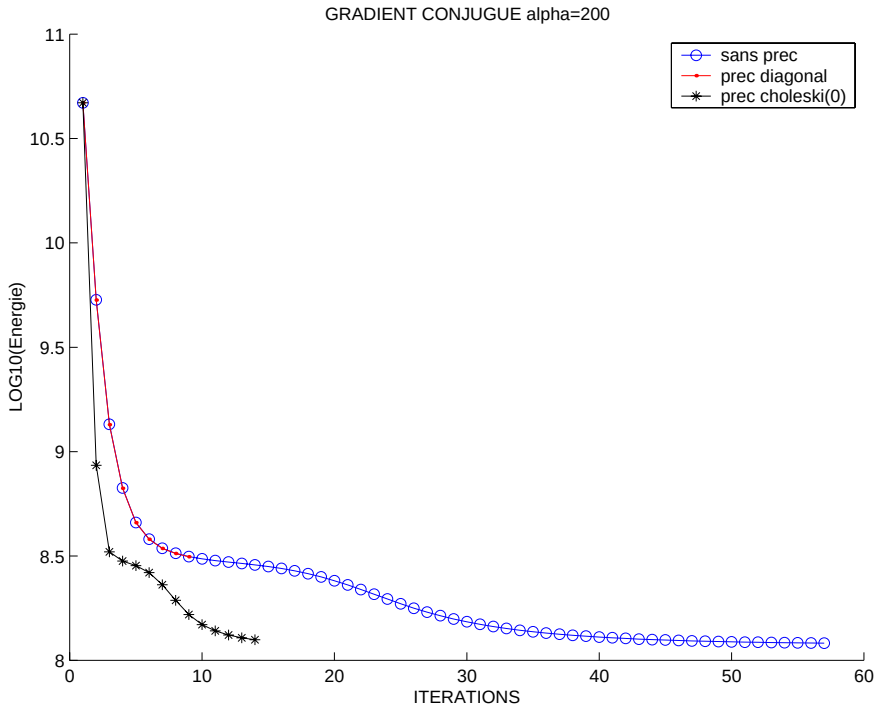


FIG. 2.8 – *Convergence du gradient conjugué préconditionné pour* $\alpha = 200$

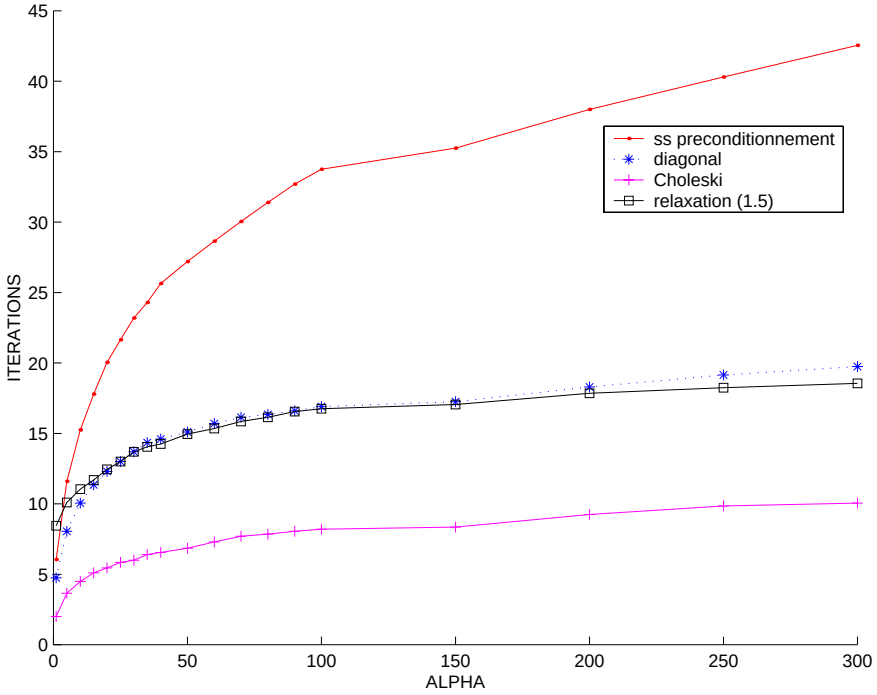


FIG. 2.9 – *Convergence du premier système pour de petites images* 64×64

α	Gradient conjugué (Choleski)	Relaxation $\omega = 1{,}5$
5	40.4	49.8
10	45.2	73.9
20	59.0	110.5
50	71.6	165.0 ‡

FIG. 2.10 – *Temps en secondes, pour la restauration d'images* 1024×1024

Chapitre 3

Filtres polynomiaux

3.1 Introduction

Le chapitre précédent présente des tests d'analyse d'image avec un modèle basé sur la régularisation de Tikhonov [42, 41]. Pour ce modèle, le filtre de régularisation est fixé. Quand la forme de cette fonction ne convient pas, il existe d'autres techniques, basées par exemple sur la fonction exponentielle [6]. On va présenter ici une nouvelle méthode basée sur des filtres polynomiaux, dont la fonction filtre n'est pas *a priori* fixée. Pour la développer, on s'appuie tout d'abord sur l'étude de la régularisation de Tikhonov.

3.1.1 Régularisation de Tikhonov

3.1.1.1 Présentation

Nous allons maintenant étudier plus en détail la régularisation de Tikhonov : le but est d'expliciter en termes de valeurs propres la solution régularisée x_ϕ. Le problème initial est la résolution de $Ax = b$, avec b bruité et A symétrique mal conditionnée, voire singulière. Alors ce trop mauvais conditionnement de A, amplifie le bruit présent dans b, si bien que la solution du système $x_* = A^\dagger b$ n'a plus aucun sens (Figure 3.10). Le système régularisé est

$$\min_x \left(\|Ax - b\|^2 + \rho \|x\|^2 \right),$$

équivalent à

$$\left(A^2 + \rho I \right) x = Ab,$$

dont la solution [1] est

1. On utilisera la même notation pour la fonction d'une variable réelle et celle définie sur l'espace des matrices, en posant $\phi(A) = U\phi(\Lambda)U^*$ avec $\phi(\Lambda) = \mathrm{diag}\left(\phi(\lambda_1) \dots \phi(\lambda_n)\right)$. Si ϕ est analytique et que toutes les valeurs propres de A sont à l'intérieur du disque de convergence de la série $\phi(x) = \sum_{i \geq 0} a_i x^i$, alors cette définition coïncide avec la définition usuelle :

$$\phi(A) = \sum_{i \geq 0} a_i A^i.$$

$$\begin{aligned} x_\rho &= \left(A^2 + \rho I\right)^{-1} Ab, \\ &= \phi_\rho^T(A) A^\dagger b, \end{aligned} \tag{3.1}$$

avec

$$\begin{cases} A^\dagger \text{ le pseudo-inverse de } A, \\ \phi_\rho^T(t) = \frac{t^2}{t^2+\rho} \text{ la fonction-filtre de Tikhonov.} \end{cases}$$

On peut remarquer que $\phi_\rho^T(0) = \frac{d\phi_\rho^T}{dt}(0) = 0$.

3.1.1.2 Étude spectrale de la régularisation

Pour bien percevoir l'action de la régularisation, il faut se placer dans la base des vecteurs propres [2] de A. Soit $A = U\Lambda U^*$ la diagonalisation de A. U est une base orthonormale de vecteurs propres et Λ est la matrice diagonale, contenant les valeurs propres : $\Lambda = \text{diag}\,(\lambda_1 \dots \lambda_n)$ avec $\lambda_1 \leq \dots \leq \lambda_n$. On pose

$$\begin{aligned} J_0 &= \{j | \lambda_j = 0\}, \\ J_1 &= \{j | \lambda_j \neq 0\}, \\ J &= \{1 \dots n\}, \\ \lambda_j^\dagger &= \begin{cases} \frac{1}{\lambda_j} \text{ si } \lambda_j \neq 0 \\ 0 \text{ si } \lambda_j = 0, \end{cases} \\ \Lambda^\dagger &= \text{diag}(\lambda_1^\dagger \dots \lambda_n^\dagger). \end{aligned}$$

Dans ce cas on a :

$$A^\dagger = U\Lambda^\dagger U^*,$$

et

$$\begin{aligned} x_\rho &= U\phi_\rho^T(\Lambda)\Lambda^\dagger U^* b, \\ &= \sum_{j \in J_1} \phi_\rho^T(\lambda_j) \frac{1}{\lambda_j} \left(u_j^* b\right) u_j, \qquad (3.2) \\ &= \sum_{j \in J_1} f^T(\lambda_j) \left(u_j^* b\right) u_j, \\ &= f^T(A) b^1, \qquad (3.3) \end{aligned}$$

avec $f^T(t) = \frac{\phi_\rho^T(t)}{t}$ et b^1 la projection orthogonale de b sur $\ker(A)^\perp$. La solution du système initial, non-régularisée est

$$x_* = \sum_{j \in J_1} \frac{1}{\lambda_j} \left(u_j^* b\right) u_j.$$

2. Dans le cas non-symétrique, ce sont les vecteurs singuliers de A qui interviennent à la place des vecteurs propres ; et on a A^*A au lieu de A^2.

Cela signifie que l'on atténue chaque composante de la solution, en fonction de la valeur propre correspondante. Typiquement on filtre pour les petites valeurs propres car une erreur ε_j sur b, suivant la composante u_j se traduirait par une erreur $\frac{1}{\lambda_j}\varepsilon_j$ dans la solution du système non-régularisé. Donc pour de petites valeurs propres ($\lambda_j \ll 1$), l'erreur est extrêmement amplifiée, au point de gâter complètement la solution x_*.

Dans la solution x_ρ, l'erreur est de la forme : $\frac{\phi_\rho^T(\lambda_j)}{\lambda_j}\varepsilon_j$. C'est-à-dire que l'amplification de l'erreur est bornée, et tend même vers 0 pour les petites valeurs propres. Bien sûr, la solution est différente ; on a :

$$x_* - x_\phi = \sum_{j\in J_1} \left[\frac{1-\phi(\lambda_j)}{\lambda_j}\right] \left(u_j^* b\right) u_j.$$

C'est-à-dire que plus le système est mal conditionné, plus la solution régularisée s'éloigne de la solution du système initial.

3.2 Filtres polynomiaux

3.2.1 Présentation

Imaginons maintenant que l'on connaisse un filtre ϕ, adapté à la régularisation du système initial. Ce filtre peut dépendre de A, de b, de notre connaissance du problème modélisé. On peut voir d'après 3.3, que connaître ϕ, c'est presque connaître la solution régularisée du problème. En effet on connaît alors $f = \frac{\phi}{t}$, la "fonction solution", puis $x_\phi = f(A)b$.

Cette fonction solution f doit être définie sur $[0,\lambda_n]$. Ce qui impose que ϕ soit dérivable et de valeur nulle en 0. On a alors $f(0) = \phi'(0)$. De plus

$$\begin{aligned} f(A)b &= \sum_{j\in J_1} f(\lambda_j)(u_j^* b)u_j + \sum_{j\in J_0} f(0)(u_j^* b)u_j, \\ &= f(A)b^1 + f(0)b^0. \end{aligned}$$

D'où si $f(0) = 0$, on a alors

$$\begin{aligned} x_\phi &= f(A)b^1, \\ &= f(A)b. \end{aligned}$$

Or $f(0) = \phi'(0)$; c'est pourquoi on impose que $\phi \in \mathcal{C}^1([0,\lambda_n])$ et que

$$\phi(0) = 0, \tag{3.4}$$
$$\phi'(0) = 0. \tag{3.5}$$

On définit alors la solution filtrée par

$$\begin{aligned} x_\phi &= f(A)b, \\ &= \phi(A)A^\dagger b, \\ &= U\phi(\Lambda)\Lambda^\dagger U^* b, \\ &= \sum_{j\in J_1} \phi(\lambda_j)\frac{1}{\lambda_j}\left(u_j^* b\right) u_j. \end{aligned}$$

On a alors

$$\begin{aligned} Ax_\phi &= \sum_{j\in J_1} \phi(\lambda_j)\left(u_j^* b\right) u_j, \\ &= \sum_{j\in J} \phi(\lambda_j)\left(u_j^* b\right) u_j, \\ &= \phi(A)b. \end{aligned}$$

On va construire une méthode itérative définissant une solution x_k à chaque pas de l'algorithme et qui converge vers x_ϕ. Le résidu de la méthode est donc

$$\begin{aligned} r_k &= A(x_\phi - x_k), \\ &= \phi(A)b - Ax_k. \end{aligned}$$

3.2.2 L'espace des solutions

On recherche les solutions dans un espace de Krylov, c'est-à-dire que les solutions x_k pourront s'exprimer sous forme polynomiale:

$$\exists Q_k \in \mathbb{R}_{k,1}[X] \text{ tel que } x_k = Q_k(A)b. \tag{3.6}$$

On impose $Q_k(0) = 0$ pour satisfaire la condition (3.5) et ainsi éliminer immédiatement de la solution x_k tous les vecteurs du noyau. On a alors

$$\begin{aligned} x_k &= \sum_{j\in J_1} Q_k(\lambda_j)(u_j^* b)u_j, \\ &= \sum_{j\in J_1} \lambda_j Q_k(\lambda_j)\frac{1}{\lambda_j}(u_j^* b)u_j, \\ &= \phi_k(A)A^\dagger b, \end{aligned}$$

en posant $\phi_k(X) = XQ_k(X) \in \mathbb{R}_{k+1}[X]$. On a alors

$$\begin{aligned} x_k &= \phi_k(A)A^\dagger b, \\ r_k &= \phi(A)b - \phi_k(A)b, \end{aligned}$$

et

$$\begin{aligned} \phi_k(0) &= 0, \\ \phi_k'(0) &= Q_k(0) = 0. \end{aligned}$$

C'est-à-dire que l'on a

$$\phi_k \in \mathbb{R}_{k+1,2}[X].$$

On alors pose $\mathcal{L}_k(A,b) = \mathcal{K}_k(A,Ab) = \langle Ab \dots A^k b \rangle$, l'espace des solutions et $\mathcal{R}_k(A,b) = \mathcal{K}_k(A,A^2b) = \langle A^2b, \dots, A^{k+1}b \rangle$. Ce dernier espace n'est pas exactement l'espace des résidus, mais plutôt celui des seconds membres filtrés, car $\phi(A)b \notin \mathcal{R}_k(A,b)$. On a bien sûr :

$$x_k \in \mathcal{L}_k(A,b), \tag{3.7}$$
$$\phi_k(A)b \in \mathcal{R}_k(A,b). \tag{3.8}$$

On note m la dimension maximale de cet espace pour $k \geq 1$. m est aussi le cardinal de

$$J_2 = \left\{ j \in J_1 | u_j^* b \neq 0 \right\}.$$

3.2.3 Définition des ϕ_k

On pose aussi :

$$\begin{aligned} \mathcal{R}_k^X &= \left\langle X^2 \dots X^{k+1} \right\rangle \\ &= \mathbb{R}_{k+1,2}[X] \end{aligned}$$

et

$$\begin{aligned} \mathcal{L}_k^X &= \mathbb{R}_{k,1}[X] \\ &= \left\langle X \dots X^k \right\rangle \\ &= \langle X \rangle \oplus \left\langle X^2 \dots X^k \right\rangle \\ &= \langle X \rangle \oplus \mathcal{R}_{k-1}^X. \end{aligned}$$

Les équations (3.7) et (3.8) se récrivent donc ainsi, en termes de polynômes :

$$Q_k \in \mathcal{L}_k^X \tag{3.9}$$
$$\phi_k \in \mathcal{R}_k^X. \tag{3.10}$$

On veut, de plus, que ϕ_k soit proche de ϕ et converge vers ϕ pour $k \to \infty$. Pour cela, on définit un produit scalaire sur $\mathcal{C}^1([0,g])$, noté $\langle \ , \ \rangle$, puis ϕ_k comme la projection orthogonale de ϕ sur $\mathcal{R}_k^X$.

3.2.4 Propriétés de l'espace

3.2.4.1 Le choix du filtre

Pour filtrer les plus petites valeurs propres de A, la fonction filtre doit être proche de 0 pour ces petites valeurs propres, et proche de 1 pour les plus grandes. Pour cela, on utilise une fonction filtre définie par morceaux. C'est à dire qu'on partitionne l'intervalle

des valeurs propres $[\lambda_1,\lambda_n] \subset [a_0,a_L]$ en L intervalles tels que $[a_0,a_L] = \amalg_{l=1}^{L}[a_{l-1},a_l[$ avec $a_L > \lambda_n$. En pratique, on utilise seulement 2 intervalles.

Le premier intervalle commence à 0 dans le cas des matrices positives. C'est le seul cas traité ci-dessous. On appelle a la borne supérieure du premier intervalle et g la borne supérieure du second ; g doit être un majorant des valeurs propres. La fonction filtre passe de 0 à 1 sur $[0,a]$. Puis sur le second intervalle, il n'y a pas de filtrage : ϕ vaut 1 sur $[a,g]$. On peut calculer un g convenable, c'est-à-dire supérieur à toutes les valeurs propres de A, grâce à la méthode des disques de Gershgorin ; ce qui est assez peu coûteux.

De plus, sur chaque intervalle, on définit ϕ par des polynômes. On veut aussi que $\phi \in \mathcal{C}^1([0,g])$, et que $\phi(0) = \phi'(0) = 0$. Pour assurer ces conditions, il faut des polynômes de degré au moins 3. C'est donc le degré que l'on choisit. Cela donne la définition suivante pour le filtre :

$$\phi_a^C(t) = \begin{cases} -\frac{2}{a^3}t^3 + \frac{3}{a^2}t^2 & \forall t \in [0,a], \\ 1 & \forall t \in [a,g]. \end{cases} \tag{3.11}$$

3.2.4.2 Le produit scalaire

Le choix du produit scalaire est lié aux intervalles de la définition de la fonction filtre ; on va en effet utiliser la même subdivision que pour le filtre. On définit un produit scalaire $\langle \psi_1,\psi_2 \rangle_{a_{l-1},a_l}$ sur chaque intervalle $[a_{l-1},a_l]$ de la subdivision, par

$$\langle \psi_1,\psi_2 \rangle_{a_{l-1},a_l} = \int_{a_{l-1}}^{a_l} \frac{\psi_1(t)\psi_2(t)}{\sqrt{(t-a_{l-1})(a_l-t)}}\,dt.$$

Le produit scalaire sur l'intervalle $[0,g]$ est la somme pondérée des produits scalaires sur tous les petits intervalles.

$$\langle \psi_1,\psi_2 \rangle = \sum_{l=1}^{L} \rho_l \int_{a_{l-1}}^{a_l} \frac{\psi_1(t)\psi_2(t)}{\sqrt{(t-a_{l-1})(a_l-t)}}\,dt.$$

Pour le filtre défini ci-dessus (3.11), le produit scalaire devient :

$$\langle \psi_1,\psi_2 \rangle = \int_0^a \frac{\psi_1(t)\psi_2(t)}{\sqrt{t(a-t)}}\,dt + \rho \int_a^g \frac{\psi_1(t)\psi_2(t)}{\sqrt{(t-a)(g-t)}}\,dt.$$

3.2.4.3 Polynômes de Chebyshev

Soit $\varsigma^{(l)}$ la similitude directe définie par

$$\begin{array}{rcl} \varsigma^{(l)} & : & [a_{l-1},a_l] \longrightarrow [-1,1], \\ & & \left\{ \begin{array}{rcl} \varsigma^{(l)}(t) & = & \frac{2}{a_l-a_{l-1}}t - \frac{a_l+a_{l-1}}{a_l-a_{l-1}} \\ \left(\varsigma^{(l)}\right)^{-1}(t) & = & \frac{a_l-a_{l-1}}{2}t + \frac{a_l+a_{l-1}}{2}. \end{array} \right. \end{array}$$

$\varsigma^{(l)}$ est affine croissante et transforme $[a_{l-1},a_l]$ en $[-1,1]$.

On appelle $\mathcal{T}_i$ le i^{e} polynôme de Chebyshev sur $[-1,1]$ et $\mathcal{T}_i^{(l)}(t) = \mathcal{T}_i\left(\varsigma^{(l)}(t)\right)$. Alors

$$\left\{\begin{array}{lrcl} & \mathcal{T}_0^{(l)}(t) & = & 1, \\ & \mathcal{T}_1^{(l)}(t) & = & \varsigma^{(l)}(t) \\ \forall i \geq 1 & \mathcal{T}_{i+1}^{(l)}(t) & = & 2\varsigma^{(l)}(t)\mathcal{T}_i\left(\varsigma^{(l)}(t)\right) - \mathcal{T}_{i-1}\left(\varsigma^{(l)}(t)\right), \\ & & = & \frac{4}{a_l-a_{l-1}} t\mathcal{T}_i^{(l)}(t) - 2\frac{a_l+a_{l-1}}{a_l-a_{l-1}}\mathcal{T}_i^{(l)}(t) - \mathcal{T}_{i-1}^{(l)}(t), \end{array}\right.$$

et

$$X\mathcal{T}_i^{(l)} = \frac{a_l - a_{l-1}}{4}\mathcal{T}_{i+1}^{(l)} + \frac{a_l + a_{l-1}}{2}\mathcal{T}_i^{(l)} + \frac{a_l - a_{l-1}}{4}\mathcal{T}_{i-1}^{(l)}, \quad i \geq 0, \tag{3.12}$$

avec $\mathcal{T}_{-1}^{(l)}(X) = 0$.

Ces polynômes de Chebyshev $\left(\mathcal{T}_k^{(l)}\right)_{k\in\mathbb{N}}$ forment une base orthogonale de l'espace $\left(\mathbb{R}[X],\langle\quad\rangle_{a_{l-1},a_l}\right)$; on a en effet

$$\begin{array}{rcl} \langle\mathcal{T}_i^{(l)},\mathcal{T}_j^{(l)}\rangle_{a_{l-1},a_l} & = & 0 \text{ si } i \neq j, \\ & = & \pi \text{ si } i = j = 0, \\ & = & \frac{\pi}{2} \text{ si } i = j \neq 0. \end{array}$$

3.2.4.4 Bases orthonormales de $(\mathbb{R}[X],\langle\quad\rangle)$

Le calcul d'une base orthonormée $(\mathcal{P}_j)_{j\in\mathbb{N}}$ de $(\mathbb{R}[X],\langle\quad\rangle)$ peut se faire à faible coût par récurrence.

On part de la base $\left(X^i\right)_{i\in\mathbb{N}}$, et on l'orthonormalise grâce au procédé de Gram-Schmidt. Cela mène à la très classique récurrence à 3 termes : on orthogonalise par rapport aux 2 dernières directions. Cela donne le schéma suivant pour $i > 2$:

$$\begin{array}{rcl} \alpha_i & = & \langle X\mathcal{P}_i,\mathcal{P}_i\rangle, \\ \mathcal{S}_i & = & X\mathcal{P}_i - \alpha_i\mathcal{P}_i - \beta_i\mathcal{P}_{i-1}, \\ \beta_{i+1} & = & \|\mathcal{S}_i\|_{\langle\quad\rangle}, \\ \mathcal{P}_{i+1} & = & \dfrac{1}{\beta_{i+1}}\mathcal{S}_i. \end{array}$$

Cela calcule une base $(\mathcal{P}_j)_{j\in\mathbb{N}}$ orthonormée pour le produit scalaire $\langle\quad\rangle$, à partir d'un vecteur initial $\mathcal{S}_0$. On pose $\beta_1 = \|\mathcal{S}_0\|_{\langle\quad\rangle}$ et $\mathcal{P}_1 = \frac{1}{\beta_1}\mathcal{S}_0$. Le choix de $\mathcal{S}_0$ dépend de l'espace dont on veut calculer une base. Si c'est $\mathbb{R}[X]$ tout entier, on commence à 1 : on orthonormalise alors la base canonique $(1,X,\dots,X^n,\dots)$. On a vu d'après 3.8 que l'on veut des filtres de valuation 2. Donc la base à orthogonaliser est $(X^2,\dots,X^n,\dots)$; d'où le vecteur initial $\mathcal{S}_0 = X^2$. On a donc $\beta_1 = \|X^2\|_{\langle\rangle}$ et $\mathcal{P}_1 = \frac{1}{\beta_1}X^2$.

3.2.4.5 Calcul efficace des produits scalaires α_j et β_{j+1}

Pour le calcul des produits scalaires de la récurrence, on va utiliser les expansions des $\mathcal{P}_i$ dans les bases de Chebyshev ; on pose pour $1 \leq l \leq L$:

$$\mathcal{P}_j = \sum_{i=0}^{j+1} \mu_{i,j}^{(l)} \mathcal{T}_i^{(l)}$$

avec $\mu_{j+1,j}^{(l)} = 0$. On pose aussi

$$X\mathcal{P}_j = \sum_{i=0}^{j+1} \sigma_i^{(l)} \mathcal{T}_i^{(l)}.$$

Les σ_i se déduisent des $\mu_{i,j}$ grâce à la récurrence 3.12 page précédente. En effet

$$\begin{aligned}
X\mathcal{P}_j &= X \sum_{i=0}^{j} \mu_{i,j}^{(l)} \mathcal{T}_i^{(l)} \\
&= \sum_{i=0}^{j} X\mu_{i,j}^{(l)} \mathcal{T}_i^{(l)} \\
&= \sum_{i=0}^{j} \mu_{i,j}^{(l)} \left(\frac{a_l - a_{l-1}}{4} \mathcal{T}_{i+1}^{(l)} + \frac{a_l + a_{l-1}}{2} \mathcal{T}_i^{(l)} + \frac{a_l - a_{l-1}}{4} \mathcal{T}_{i-1}^{(l)} \right) \\
&= \sum_{i=0}^{j+1} \left(\frac{a_l - a_{l-1}}{4} (\mu_{i+1,j}^{(l)} + \mu_{i-1,j}^{(l)}) + \frac{a_l + a_{l-1}}{2} \mu_{i,j}^{(l)} \right) \mathcal{T}_i^{(l)}.
\end{aligned}$$

D'où

$$\sigma_i^{(l)} = \frac{a_l - a_{l-1}}{4} \left(\mu_{i+1,j}^{(l)} + \mu_{i-1,j}^{(l)} \right) + \frac{a_l + a_{l-1}}{2} \mu_{i,j}^{(l)}.$$

Connaissant les composantes de $\mathcal{P}_j$ et $X\mathcal{P}_i$ dans toutes les bases $\left(\mathcal{T}_j^{(l)}\right)$, on peut calculer les produits scalaires partiels :

$$\begin{aligned}
\langle X\mathcal{P}_j, \mathcal{P}_j \rangle_{a_{l-1}, a_l} &= \left\langle \sum_i \sigma_i \mathcal{T}_i, \sum_i \mu_i \mathcal{T}_i \right\rangle_{a_{l-1}, a_l}, \\
&= \pi \sigma_0^{(l)} \mu_0^{(l)} + \frac{\pi}{2} \sum_{i=1}^{j+1} \sigma_i^{(l)} \mu_i^{(l)}.
\end{aligned}$$

Ce qui donne la valeur de α_j :

$$\begin{aligned}
\alpha_j &= \langle X\mathcal{P}_j,\mathcal{P}_j\rangle, \\
&= \sum_{l=1}^{L} \rho_l \langle X\mathcal{P}_j,\mathcal{P}_j\rangle_{a_{l-1},a_l}, \\
&= \sum_{l=1}^{L} \left(\pi\rho_l\sigma_0^{(l)}\mu_0^{(l)} + \frac{\pi}{2}\rho_l \sum_{i=1}^{j+1} \sigma_i^{(l)}\mu_i^{(l)} \right), \\
&= \pi \sum_{l=1}^{L} \rho_l\sigma_0^{(l)}\mu_0^{(l)} + \frac{\pi}{2} \sum_{l=1}^{L}\sum_{i=0}^{j} \rho_l\sigma_i^{(l)}\mu_i^{(l)}.
\end{aligned}$$

On pose maintenant

$$\mathcal{S}_j = \sum_{i=0}^{j+1} \upsilon_i^{(l)} \mathcal{T}_i^{(l)}.$$

On a alors

$$\upsilon_i^{(l)} = \sigma_i^{(l)} - \alpha_j\mu_{i,j}^{(l)} - \beta_j\mu_{i,j-1}^{(l)},$$

car

$$\mathcal{S}_j = X\mathcal{P}_j - \alpha_j\mathcal{P}_j - \beta_j\mathcal{P}_j.$$

Ce qui permet le calcul de $\beta_{j+1} = \langle \mathcal{S}_j,\mathcal{S}_j\rangle^{\frac{1}{2}}$. On a alors

$$\beta_{j+1} = \left(\sum_{l=1}^{L} \rho_l \left(\pi\upsilon_0^2 + \frac{\pi}{2} \sum_{i=1}^{j+1} \left(\upsilon_i^{(l)}\right)^2 \right) \right)^{\frac{1}{2}}.$$

On en déduit

$$\mu_{i,j+1}^{(l)} = \frac{1}{\beta_{j+1}} \upsilon_i^{(l)}.$$

On a ainsi, pour $1 \leq l \leq L$ et $1 \leq i \leq j+1$,

$$\begin{aligned}
\sigma_i^{(l)} &= \frac{a_l - a_{l-1}}{4} \left(\mu_{i+1,j}^{(l)} + \mu_{i-1,j}^{(l)} \right) + \frac{a_l + a_{l-1}}{2} \mu_{i,j}^{(l)}, \\
\alpha_j &= \pi \sum_{l=1}^{L} \rho_l\sigma_0^{(l)}\mu_0^{(l)} + \frac{\pi}{2} \sum_{l=1}^{L}\sum_{i=0}^{j} \rho_l\sigma_i^{(l)}\mu_i^{(l)}, \\
\upsilon_i^{(l)} &= \sigma_i^{(l)} - \alpha_j\mu_{i,j}^{(l)} - \beta_j\mu_{i,j-1}^{(l)}, \\
\beta_{j+1} &= \left(\sum_{l=1}^{L} \rho_l \left(\pi\upsilon_0^2 + \frac{\pi}{2} \sum_{i=1}^{j+1} \left(\upsilon_i^{(l)}\right)^2 \right) \right)^{\frac{1}{2}}, \\
\mu_{i,j+1}^{(l)} &= \frac{1}{\beta_{j+1}} \upsilon_i^{(l)}.
\end{aligned}$$

3.2.4.6 Propriété fondamentale de la base $(\mathcal{P}_j)$

On appelle $\mathbf{P}_k$, le vecteur-ligne dont les éléments sont les polynômes $\mathcal{P}_j$ pour $1 \leq j \leq k$.

$$\mathbf{P}_k = [\mathcal{P}_1 \dots \mathcal{P}_k].$$

Les polynômes vérifient la récurrence suivante

$$\begin{aligned} \mathcal{P}_1 &= \frac{X^2}{\beta_1}, \\ \mathcal{P}_{j+1} &= \frac{1}{\beta_{j+1}} (X\mathcal{P}_j - \alpha_j \mathcal{P}_j - \beta_j \mathcal{P}_{j-1}) \quad \forall j \geq 2, \\ \Leftrightarrow X\mathcal{P}_j &= \beta_{j+1}\mathcal{P}_{j+1} + \alpha_j \mathcal{P}_j + \beta_j \mathcal{P}_{j-1}. \end{aligned}$$

On appelle alors T_k la matrice tridiagonale formée avec les $(\alpha_j)_{1 \leq j \leq k}$ et les $(\beta_j)_{2 \leq j \leq k}$:

$$T_k = \begin{bmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_{k-1} & \alpha_{k-1} & \beta_k \\ & & & \beta_k & \alpha_k \end{bmatrix}.$$

Et on appelle $\bar{T}_k$ la matrice T_k complétée, c'est-à-dire, qu'on lui a ajouté une dernière ligne $(0 \dots 0 \ \beta_{k+1})$.

$$\bar{T}_{k+1} = \begin{bmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_{k-1} & \alpha_{k-1} & \beta_k \\ & & & \beta_k & \alpha_k \\ & & & & \beta_{k+1} \end{bmatrix}.$$

On a alors

$$\begin{aligned} X\mathbf{P}_k &= \mathbf{P}_k T_k + (0 \dots 0 \ \beta_{k+1}\mathcal{P}_{k+1}) \\ &= \mathbf{P}_{k+1}\bar{T}_{k+1}. \end{aligned} \tag{3.13}$$

3.2.5 Calcul du filtre ϕ_k

ϕ_k est la projection de ϕ sur $\mathcal{R}_k^X$. Or $\{\mathcal{P}_k\}_k$ est une base orthonormée de $\mathcal{R}_k^X$, pour le produit scalaire associé à cette projection. Donc

$$\phi_k = \sum_{i=1}^{k} \langle \phi, \mathcal{P}_i \rangle \mathcal{P}_i \tag{3.14}$$

$$= \sum_{i=1}^{k} \gamma_i \mathcal{P}_i \tag{3.15}$$

avec $\gamma_i = \langle \phi, \mathcal{P}_i \rangle$. On pose alors $\gamma_i^{(l)} = \langle \phi, \mathcal{T}_i^{(l)} \rangle_{a_{l-1},a_l}$. Or ϕ est polynomial par morceaux et les intervalles de définition de ϕ sont les $[a_{l-1},a_l]$. On appelle $\phi^{(l)}$ le polynôme correspondant à ϕ sur $[a_{l-1},a_l]$ et $\phi_i^{(l)}$ les composantes de $\phi^{(l)}$ dans $\left(\mathcal{T}_i^{(l)}\right)_{i\in\mathbb{N}}$. Soit d le degré maximal de $\phi^{(l)}$. Alors

$$\phi^{(l)} = \sum_{i=0}^{d} \phi_i^{(l)} \mathcal{T}_i^{(l)}.$$

Ces $\phi_i^{(l)}$ sont calculés dès le choix du filtre. On a alors

$$\begin{aligned}
\gamma_k^{(l)} &= \langle \phi^{(l)}, \mathcal{T}_k \rangle_{[a_{l-1},a_l]}, \\
&= \left\langle \sum_{i=0}^{d} \phi_i^{(l)} \mathcal{T}_i^{(l)}, \mathcal{T}_k^{(l)} \right\rangle_{[a_{l-1},a_l]}, \\
&= \begin{cases} \pi \phi_0^{(l)} & \text{si } k = 0, \\ \frac{\pi}{2} \phi_k^{(l)} & \text{si } 1 \leq k \leq d. \end{cases}
\end{aligned}$$

D'où

$$\begin{aligned}
\gamma_k &= \langle \phi, \mathcal{P}_k \rangle, \\
&= \sum_{l=1}^{L} \langle \phi^{(l)}, \mathcal{P}_k \rangle_{[a_{l-1},a_l]}, \\
&= \sum_{l=1}^{L} \left\langle \sum_{i=0}^{d} \phi_i^{(l)} \mathcal{T}_i^{(l)}, \sum_{i=0}^{k+1} \mu_{i,k}^{(l)} \mathcal{T}_i^{(l)} \right\rangle_{[a_{l-1},a_l]}, \\
&= \sum_{l=1}^{L} \left(\pi \phi_0^{(l)} \mu_{0,k}^{(l)} + \frac{\pi}{2} \sum_{i=1}^{\min(d,k+1)} \phi_i^{(l)} \mu_{i,k}^{(l)} \right), \\
&= \pi \sum_{l=1}^{L} \phi_0^{(l)} \mu_{0,k}^{(l)} + \frac{\pi}{2} \sum_{l=1}^{L} \sum_{i=1}^{\min(d,k+1)} \phi_i^{(l)} \mu_{i,k}^{(l)}.
\end{aligned}$$

La complexité de ce produit scalaire est en $O(Lk)$. C'est indépendant de la taille N de la matrice, mais augmente avec le nombre d'itérations. Ainsi, pour un petit nombre d'itérations, c'est très peu coûteux. L'algorithme demande le stockage des $\mu_{i,k}^{(l)}$ et $\mu_{i,k-1}^{(l)}$, composantes des polynômes $\mathcal{P}_k$ et $\mathcal{P}_{k-1}$ dans les bases de $(\mathcal{T}_i)_{i\in\mathbb{N}}$, ainsi que les composantes $\phi_i^{(l)}$ de la fonction ϕ.

3.2.6 Calcul du polynôme solution Q_k

On avait $\phi_k \in \langle X^2 \dots X^{k+1} \rangle = \mathcal{R}_k^X$. Soit Γ_k le vecteur des composantes de ϕ_k dans la base des $(\mathcal{P}_k)$. On a alors

$$\phi_k = \mathbf{P}_k \Gamma_k \text{ et } \Gamma_k = \begin{bmatrix} \gamma_1 \\ \vdots \\ \gamma_k \end{bmatrix}.$$

On a aussi $Q_k = \frac{\phi_k}{X} \in \langle X \dots X^k \rangle = \mathcal{L}_k^X$. Soit $\omega_{0,k}, \omega_{1,k} \dots \omega_{k-1,k}$, les composantes de Q_k dans la base $(X, \mathcal{P}_1 \dots \mathcal{P}_{k-1})$. On a alors

$$\begin{aligned} Q_k &= \omega_{0,k} X + \sum_{j=1}^{k-1} \omega_{j,k} \mathcal{P}_j, \\ &= \omega_{0,k} X + \mathbf{P}_{k-1} \Omega_{k-1}. \end{aligned}$$

avec $\Omega_k = \begin{bmatrix} \omega_{1,k} \\ \vdots \\ \omega_{k-1,k} \end{bmatrix}$. On pose aussi $\tilde{\Omega}_k \begin{bmatrix} \omega_{0,k} \\ \Omega_k \end{bmatrix}$. On a alors

$$\begin{aligned} Q_k &= [X, \mathcal{P}_1 \dots \mathcal{P}_{k-1}] \, \tilde{\Omega}_k, \\ \text{et } \phi_k &= X Q_k, \\ &= \omega_{0,k} X^2 + X \mathbf{P}_{k-1} \Omega_k, \\ &= \omega_{0,k} \beta_1 \mathcal{P}_1 + \mathbf{P}_k \bar{T}_k \Omega_k, \\ &= \mathbf{P}_k (\omega_{0,k} \beta_1 e_1 + \bar{T}_k \Omega_k). \end{aligned}$$

D'où $\mathbf{P}_k \Gamma_k = \mathbf{P}_k (\omega_{0,k} \beta_1 e_1 + \bar{T}_k \Omega_k)$. Or les $(\mathcal{P}_j)$ forment une famille libre, donc

$$\Gamma_k = \omega_{0,k} \beta_1 e_1 + \bar{T}_k \Omega_k. \tag{3.16}$$

On appelle alors $\tilde{T}_k$ la matrice $\bar{T}_k$ avec la colonne $\beta_1 e_1$ ajoutée à gauche :

$$\tilde{T}_k = \begin{bmatrix} \beta_1 & \alpha_1 & \beta_2 & & & \\ & \beta_2 & \alpha_2 & \beta_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & \beta_{k-2} & \alpha_{k-2} & \beta_{k-1} \\ & & & & \beta_{k-1} & \alpha_{k-1} \\ & & & & & \beta_k \end{bmatrix}.$$

On déduit alors de 3.16

$$\Gamma_k = \tilde{T}_k \tilde{\Omega}_k. \tag{3.17}$$

C'est-à-dire que pour connaître les composantes $\tilde{\Omega}_k$ de Q_k dans la base $(X, \mathcal{P}_1 \dots \mathcal{P}_{k-1})$, on peut résoudre le système triangulaire (3.17).

3.2.7 Calcul de la solution x_k

Il faut alors calculer $x_k = Q_k(A)b$. On pose pour cela $p_k = \mathcal{P}_k(A)b$. Les p_k vérifient les mêmes récurrences que les $\mathcal{P}_k$, c'est-à-dire

$$p_{k+1} = \frac{1}{\beta_{k+1}}(Ap_k - \alpha_k p_k - \beta_k p_{k-1}) \text{ pour } k \geq 1,$$

et avec $p_1 = \frac{1}{\beta_1}A^2 b$ et $p_0 = 0$. On a alors

$$\begin{aligned} x_k &= Q_k(A)b, \\ &= (Ab, \mathcal{P}_1(A)b \ldots, \mathcal{P}_{k-1}(A)b)\,\tilde{\Omega}_k, \\ &= (Ab, p_1 \ldots, p_{k-1})\,\tilde{\Omega}_k, \\ &= (Ab, P_{k-1})\,\tilde{\Omega}_k, \end{aligned}$$

en posant $P_k = [p_1, \ldots, p_k]$. On pose aussi $\tilde{P}_k = [Ab, P_{k-1}]$, donc $x_k = \tilde{P}_k \tilde{\Omega}_k$.

On cherche alors à exprimer cette relation sous la forme d'une récurrence. Pour cela on pose

$$\tilde{\omega_k} = \begin{bmatrix} \omega_{0,k} \\ \vdots \\ \omega_{k-2,k} \end{bmatrix},$$

si bien que

$$\tilde{\Omega}_k = \left[\begin{array}{c} \tilde{\omega}_k \\ \hline \omega_{k-1,k} \end{array}\right].$$

On pose aussi

$$\bar{\delta}_k = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \beta_k \\ \alpha_k \end{bmatrix},$$

et

$$u_k = \tilde{T}_k^{-1} \bar{\delta}_k.$$

On a alors

$$\begin{aligned}
&\tilde{T}_{k+1}\tilde{\Omega}_{k+1} = \Gamma_{k+1},\\
\Rightarrow\quad &\left[\begin{array}{c|c} \tilde{T}_k & \bar{\delta}_k \\ \hline 0 & \beta_{k+1} \end{array}\right]\left[\begin{array}{c} \tilde{\omega}_{k+1} \\ \hline \omega_{k,k+1} \end{array}\right] = \left[\begin{array}{c} \Gamma_k \\ \hline \gamma_{k+1} \end{array}\right],\\
\Rightarrow\quad &\left\{\begin{array}{l} \tilde{T}_k\tilde{\omega}_{k,k+1} + \bar{\delta}_k\omega_{k,k+1} = \Gamma_k,\\ \beta_{k+1}\omega_{k,k+1} = \gamma_{k+1}, \end{array}\right.\\
\Rightarrow\quad &\left\{\begin{array}{l} \omega_{k,k+1} = \frac{\gamma_{k+1}}{\beta_{k+1}},\\ \tilde{T}_k\tilde{\omega}_{k+1} = \Gamma_k - \bar{\delta}_k\omega_{k,k+1}. \end{array}\right.
\end{aligned}$$

On déduit alors de cette dernière ligne que

$$\begin{aligned}
\tilde{\omega}_{k+1} &= \tilde{T}_k^{-1}\Gamma_k - \tilde{T}_k^{-1}\bar{\delta}_k\omega_{k,k+1},\\
&= \tilde{\Omega}_k - \frac{\gamma_{k+1}}{\beta_{k+1}}u_k.
\end{aligned}$$

De plus,

$$\begin{aligned}
x_{k+1} &= \tilde{P}_{k+1}\tilde{\Omega}_{k+1},\\
&= \tilde{P}_k\tilde{\omega}_k + \frac{\gamma_{k+1}}{\beta_{k+1}}p_k,\\
&= \tilde{P}_k\left(\tilde{\Omega}_k - \frac{\gamma_{k+1}}{\beta_{k+1}}u_k\right) + \frac{\gamma_{k+1}}{\beta_{k+1}}p_k.
\end{aligned}$$

On pose alors

$$q_{k+1} = p_k - \tilde{P}_k u_k,$$

et on obtient

$$x_{k+1} = x_k + \frac{\gamma_{k+1}}{\beta_{k+1}}q_{k+1}.$$

Il faut alors trouver une récurrence pour le calcul des $q_{k+1} = p_k - \tilde{P}_k\tilde{T}_k^{-1}\bar{\delta}_k$. On part de $\tilde{T}_k u_k = \bar{\delta}_k$, qui s'écrit par blocs de la manière suivante :

$$\left[\begin{array}{c|c} \tilde{T}_{k-1} & \bar{\delta}_{k-1} \\ \hline 0 & \beta_k \end{array}\right]\left[\begin{array}{c} v_{k-1} \\ \hline \frac{\alpha_k}{\beta_k} \end{array}\right] = \left[\begin{array}{c} 0 \\ \vdots \\ \beta_k \\ \hline \alpha_k \end{array}\right] = \left[\begin{array}{c} \delta_{k-1} \\ \hline \alpha_k \end{array}\right]$$

avec

$$u_k = \left[\begin{array}{c} v_{k-1} \\ \hline \frac{\alpha_k}{\beta_k} \end{array}\right] \text{ et } \delta_{k-1} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \beta_k \end{bmatrix}.$$

On a alors

$$\begin{aligned}
&\tilde{T}_{k-1}v_{k-1} + \tfrac{\alpha_k}{\beta_k}\bar{\delta}_{k-1} = \delta_{k-1},\\
\Rightarrow v_{k-1} &= \tilde{T}_{k-1}^{-1}\delta_{k-1} - \tfrac{\alpha_k}{\beta_k}\tilde{T}_{k-1}^{-1}\bar{\delta}_{k-1} \quad .\\
&= \tilde{T}_{k-1}^{-1}\delta_{k-1} - \tfrac{\alpha_k}{\beta_k}u_{k-1}.
\end{aligned}$$

Il faut alors calculer $\tilde{T}_{k-1}^{-1}\delta_{k-1}$. Soit alors w_{k-2} tel que

$$\left[\begin{array}{c|c} \tilde{T}_{k-2} & \bar{\delta}_{k-2} \\ \hline 0 & \beta_{k-1} \end{array}\right] \left[\begin{array}{c} w_{k-2} \\ \hline \frac{\beta_k}{\beta_{k-1}} \end{array}\right] = \left[\begin{array}{c} 0 \\ \hline \beta_k \end{array}\right].$$

Cela implique

$$\begin{aligned}
&\tilde{T}_{k-2}w_{k-2} + \tfrac{\beta_k}{\beta_{k-1}}\bar{\delta}_{k-2} = 0,\\
\Rightarrow w_{k-2} &= -\tfrac{\beta_k}{\beta_{k-1}}\tilde{T}_{k-2}^{-1}\bar{\delta}_{k-2},\\
&= -\tfrac{\beta_k}{\beta_{k-1}}u_{k-2}.
\end{aligned}$$

D'où

$$\tilde{T}_{k-1}^{-1}\delta_{k-1} = \left[\begin{array}{c} -\frac{\beta_k}{\beta_{k-1}}u_{k-2} \\ \frac{\beta_k}{\beta_{k-1}} \end{array}\right],$$

et donc

$$\begin{aligned}
\tilde{P}_{k-1}\tilde{T}_{k-1}^{-1}\delta_{k-1} &= \frac{\beta_k}{\beta_{k-1}}\left[\begin{array}{c|c} \tilde{P}_{k-2} & p_{k-2} \end{array}\right] \left[\begin{array}{c} -u_{k-2} \\ \hline 1 \end{array}\right],\\
&= \frac{\beta_k}{\beta_{k-1}}\left(p_{k-2} - \tilde{P}_{k-2}u_{k-2}\right),\\
&= \frac{\beta_k}{\beta_{k-1}}q_{k-1},
\end{aligned}$$

c'est-à-dire

$$\begin{aligned}
\tilde{P}_k u_k &= \left[\begin{array}{c|c} \tilde{P}_{k-1} & p_{k-1} \end{array}\right] \left[\begin{array}{c} v_{k-1} \\ \hline \frac{\alpha_k}{\beta_k} \end{array}\right],\\
&= \tilde{P}_{k-1}v_{k-1} + \frac{\alpha_k}{\beta_k}p_{k-1},\\
&= \tilde{P}_{k-1}\tilde{T}_{k-1}^{-1}\delta_{k-1} - \frac{\alpha_k}{\beta_k}\tilde{P}_{k-1}u_{k-1} + \frac{\alpha_k}{\beta_k}p_{k-1},\\
&= P_{k-1}\tilde{T}_{k-1}^{-1}\delta_{k-1} + \frac{\alpha_k}{\beta_k}q_k,\\
&= \frac{\beta_k}{\beta_{k-1}}q_{k-1} + \frac{\alpha_k}{\beta_k}q_k.
\end{aligned}$$

D'où finalement

$$q_{k+1} = p_k - \frac{\alpha_k}{\beta_k} q_k - \frac{\beta_k}{\beta_{k-1}} q_{k-1}.$$

3.2.8 Identification des q_k

En fait, ces vecteurs auxiliaires ne sont rien d'autre que les p_k "divisés" par A :

Théorème 3.1 *Les q_k vérifient en effet la relation suivante :*

$$\forall k \in \{1 \dots m\} \quad A q_k = \beta_k p_k.$$

Preuve La preuve se fait par récurrence. En effet, la relation est vraie pour $k = 0$ et $k = 1$; on a pour ces indices

- $q_0 = 0$ et $p_0 = 0$;
- $q_1 = Ab$ et $p_1 = \frac{A^2 b}{\beta_1}$.

Supposons maintenant que la relation $Aq_k = \beta_k p_k$ soit vraie au rang k, alors

$$\begin{aligned} A q_{k+1} &= A p_k - \frac{\alpha_k}{\beta_k} A q_k - \frac{\beta_k}{\beta_{k-1}} A q_{k-1}, \\ &= A p_k - \alpha_k p_k - \beta_k p_{k-1}, \\ &= \beta_{k+1} p_{k+1}. \end{aligned}$$

□

3.2.9 L'algorithme

L'initialisation se fait de la manière suivante

$$\begin{aligned} \beta_1 &= \|X^2\|_{\langle ,\rangle} \\ \mathcal{P}_1 &= \frac{X^2}{\beta_1} \\ &= \sum_{i=0}^{2} \mu_{i,1}^{(l)} \mathcal{T}_i^{(l)} \\ \gamma_1 &= \langle \phi, \mathcal{P}_1 \rangle \\ p_1 &= \frac{A^2 b}{\beta_1} \\ q_1 &= Ab \\ x_1 &= Q_1(A) b \\ &= \frac{\gamma_1}{\beta_1} Ab \end{aligned}$$

ce qui donne, au final, l'algorithme 3.1 page suivante.

Require: ϕ polynomial par morceaux sur $[a_{l-1},a_l]$ pour $l = 1 \dots L$

$\beta_1 = \langle X,X \rangle^{\frac{1}{2}}$
$\mathcal{P}_1 = \frac{X^2}{\beta_1}$
$\gamma_1 = \langle \phi,\mathcal{P}_1 \rangle$
$p_1 = \frac{A^2 b}{\beta_1}$, $p_0 = 0$
$q_1 = Ab$, $q_0 = 0$
$x_1 = \frac{\gamma_1}{\beta_1} Ab$, $x_0 = 0$
for $k = 1, \dots$ **do**
  calcul de $X\mathcal{P}_k$
  $\alpha_k = \langle X\mathcal{P}_k,\mathcal{P}_k \rangle$
  $\mathcal{S}_k = X\mathcal{P}_k - \alpha_k \mathcal{P}_k - \beta_k \mathcal{P}_{k-1}$
  $s_k = Ap_k - \alpha_k p_k - \beta_k p_{k-1}$
  $\beta_{k+1} = \langle \mathcal{S}_k,\mathcal{S}_k \rangle^{\frac{1}{2}}$
  $\mathcal{P}_{k+1} = \frac{1}{\beta_{k+1}} \mathcal{S}_k$
  $p_{k+1} = \frac{1}{\beta_{k+1}} s_k$
  $\gamma_{k+1} = \langle \mathcal{P}_{k+1},\phi \rangle$
  $q_{k+1} = p_k - \frac{\alpha_k}{\beta_k} q_k - \frac{\beta_k}{\beta_{k-1}} q_{k-1}$
  $x_{k+1} = x_k + \frac{\gamma_{k+1}}{\beta_{k+1}} q_{k+1}$
end for

Algorithme 3.1: Régularisation par filtre polynomial

3.3 Étude de la convergence

Nous allons montrer tout d'abord que l'algorithme 3.1, présente comme un redémarrage au bout de m étapes. Puis on montre que les filtres ϕ_k convergent vers ϕ pour la norme issue du produit scalaire.

3.3.1 Comportement après m itérations

L'application

$$\begin{aligned} \Upsilon : \mathbb{R}_{m,2}[X] &\rightarrow \mathcal{R}_m(A,b) \\ P &\mapsto P(A)b \end{aligned}$$

est un isomorphisme d'espace euclidien. Le produit scalaire défini sur $\mathbb{R}_{m,2}[X]$, induit donc un produit scalaire sur $\mathcal{R}_m(A,b)$, en posant :

$$\langle P(A)b,Q(A)b \rangle_{\mathcal{R}_m(A,b)} = \langle P,Q \rangle_{\mathbb{R}_{m,2}[X]},$$

c'est-à-dire :

$$\langle x,y \rangle_{\mathcal{R}_m(A,b)} = \langle \Upsilon^{-1}(x),\Upsilon^{-1}(y) \rangle_{\mathbb{R}_{m,2}[X]}.$$

Donc tant que $k \leq m$, toutes les propriétés des polynômes $\mathcal{P}_k$ se transposent sur les vecteurs p_k. En particulier, $(p_k)_{k\in\{1,\dots,m\}}$ forme une base orthonormée de $\mathcal{L}_m(A,b)$. On a alors

$$s_m = Ap_m - \alpha_m p_m - \beta_m p_{m-1}.$$

Or $\forall i \in \{1,\dots,m\}$, on a

$$\langle s_m,p_i\rangle = \langle Ap_m,p_i\rangle - \alpha_m\langle p_m,p_i\rangle - \beta_m\langle p_m,p_{i-1}\rangle,$$

et donc

– si $i = m$

$$\begin{aligned}\langle s_m,p_m\rangle &= \langle Ap_m,p_m\rangle - \alpha_m\langle p_m,p_m\rangle - \beta_m\langle p_{m-1},p_m\rangle,\\ &= \langle X\mathcal{P}_m,\mathcal{P}_m\rangle - \langle X\mathcal{P}_m,\mathcal{P}_m\rangle 1 - 0,\\ &= 0\ ;\end{aligned}$$

– si $i = m-1$

$$\begin{aligned}\langle s_m,p_{m-1}\rangle &= \langle Ap_m,p_{m-1}\rangle - \alpha_m\langle p_m,p_{m-1}\rangle - \beta_m\langle p_{m-1},p_{m-1}\rangle,\\ &= \langle X\mathcal{P}_m,\mathcal{P}_{m-1}\rangle - 0 - \langle X\mathcal{P}_m,\mathcal{P}_{m-1}\rangle 1,\\ &= 0\ ;\end{aligned}$$

– si $i < m-1$

$$\begin{aligned}\langle s_m,p_i\rangle &= \langle Ap_m,p_i\rangle,\\ &= \langle X\mathcal{P}_m,\mathcal{P}_i\rangle,\\ &= \langle \mathcal{P}_m,X\mathcal{P}_i\rangle,\\ &= \langle \mathcal{P}_m,\beta_{i+1}\mathcal{P}_{i+1} + \alpha_i\mathcal{P}_i + \beta_i\mathcal{P}_{i-1},\\ &= \beta_{i+1}\langle \mathcal{P}_m,\mathcal{P}_{i+1}\rangle + \alpha_i\langle \mathcal{P}_m,\mathcal{P}_i\rangle + \beta_i\langle \mathcal{P}_m,\mathcal{P}_{i-1}\rangle,\\ &= 0, \text{ car } \mathcal{P}_m \perp_{\langle,\rangle} (\mathcal{P}_i)_{i\in\{1..m-1\}}\,.\end{aligned}$$

D'ou $s_m \perp_{\langle,\rangle} \mathcal{R}_m$. Or $s_m \in \mathcal{R}_m$, donc $s_m = 0$, d'où $p_{m+1} = 0$. Il y a pseudo-breakdown à la m^{e} étape. On a alors, d'après (3.13)

$$\begin{aligned}X\mathbf{P}_m &= \mathbf{P}_{m+1}\bar{T}_{m+1},\\ \Rightarrow AP_m &= P_{m+1}\bar{T}_{m+1},\\ \Rightarrow AP_m &= P_mT_m, \text{ car } p_{m+1} = 0,\\ \text{d'où } P_m^*AP_m &= T_m.\end{aligned}$$

Et, on a aussi $x_{m+1} = x_m$.

3.3.2 Vitesse de convergence

On va maintenant chercher à quantifier la vitesse asymptotique de convergence. Les deux premiers lemmes, 3.1 et 3.2, et la proposition 3.1 permettent de majorer l'écart en norme $\|\ \|_{\langle\ ,\ \rangle}$, par l'écart en norme uniforme.

Lemme 3.1 $\|\phi - \phi_k\| = \mathrm{d}\,(\phi,\mathbb{R}_{k+1,2}[X])_{\langle\ ,\ \rangle}$.

Preuve ϕ_k est, en effet, la projection $\langle\ ,\ \rangle$-orthogonale de ϕ sur $\mathbb{R}_{k+1,2}[X]$. □

Lemme 3.2 *Pour un produit scalaire de la forme*

$$\|\psi\|^2_{\langle\ ,\ \rangle} = \int_0^a \frac{\psi^2(t)}{\sqrt{t(a-t)}}\,dt + \rho\int_a^b \frac{\psi^2(t)}{\sqrt{(t-a)(b-t)}}\,dt$$

on a la majoration suivante

$$\|\psi\|^2_{\langle\ ,\ \rangle} \leq (1+\rho)\pi\|\psi\|^2_\infty. \tag{3.18}$$

Preuve On a en effet :

$$\begin{aligned}
\|\psi\|^2_{\langle\ ,\ \rangle} &= \int_0^a \frac{\psi^2(t)}{\sqrt{t(a-t)}}\,dt + \rho\int_a^b \frac{\psi^2(t)}{\sqrt{(t-a)(b-t)}}\,dt \\
&\leq \|\psi\|^2_\infty \left(\int_0^a \frac{1}{\sqrt{t(a-t)}}\,dt + \rho\int_a^b \frac{1}{\sqrt{(t-a)(b-t)}}\,dt\right) \\
&\leq (1+\rho)\pi\|\psi\|^2_\infty.
\end{aligned}$$

□

Proposition 3.1 $\|\phi-\phi_k\|_{\langle\ ,\ \rangle} \leq \sqrt{(1+\rho)\pi}\ \mathrm{d}(\phi,\mathbb{R}_{k+1,2}[X])_\infty$.

Preuve On a en effet :

$$\begin{aligned}
\|\phi-\phi_k\|_{\langle\ ,\ \rangle} &= \min_{\mathcal{P}\in\mathbb{R}_{k+1,2}} \|\phi-\mathcal{P}\|_{\langle\ ,\ \rangle} \\
&\leq \sqrt{(1+\rho)\pi}\min_{\mathcal{P}\in\mathbb{R}_{k+1,2}} \|\phi-\mathcal{P}\|_\infty, \quad \text{d'après le lemme 3.2} \\
&= \sqrt{(1+\rho)\pi}\ \mathrm{d}(\phi,\mathbb{R}_{k+1,2}[X])_\infty.
\end{aligned}$$

□

Pour estimer plus précisément cette vitesse de convergence, on va introduire les polynômes de Berstein associés à une fonction f, qui forment une suite convergeant uniformément vers f. On aura alors une majoration de $\mathrm{d}(\phi,\mathbb{R}_{k+1,2}[X])_\infty$.

Pour f définie sur [0,1], le n^{e} polynôme de Berstein est :

$$B_n(f)(X) = \sum_{k=0}^{n} f\left(\frac{k}{n}\right)\begin{pmatrix} n \\ k \end{pmatrix} X^k(1-X)^{n-k}.$$

On a $B_n(f)(0) = f(0) = 0$ et $B'_n(f)(0) = nf\left(\frac{1}{n}\right)$ qui converge vers $f'(0) = 0$.

Théorème 3.2 *Soit f ν-lipschitzienne sur* [0,1] *et soit M un majorant de f, alors*

$$\|f - B_n(f)\|_\infty \leq \frac{3}{2}M^{\frac{1}{3}}\nu^{\frac{2}{3}}\ \frac{1}{\sqrt[3]{n}}.$$

Preuve On a

$$\begin{aligned} f(x) - B_n(f,x) &= \sum_{k=0}^{n} \left(f(x) - f\left(\frac{k}{n}\right)\right) \left(\begin{array}{c} n \\ k \end{array} \right) x^k (1-x)^{n-k} \\ &= \sum_{\left|\frac{k}{n}-x\right|<\delta} \left(f(x) - f\left(\frac{k}{n}\right)\right) \left(\begin{array}{c} n \\ k \end{array} \right) x^k (1-x)^{n-k} \\ &\quad + \sum_{\left|\frac{k}{n}-x\right|>\delta} \left(f(x) - f\left(\frac{k}{n}\right)\right) \left(\begin{array}{c} n \\ k \end{array} \right) x^k (1-x)^{n-k} \end{aligned}$$

Or f étant continue sur [0,1] compact, elle est bornée par M. De plus, on a, pour $\delta > 0$ et $\forall x,y$ tels que $|y-x| < \delta$, $|f(y) - f(x)| < \nu\delta$. Donc

$$\begin{aligned} |f(x) - B_n(f,x)| &\leq \sum_{\left|\frac{k}{n}-x\right|<\delta} \left| f(x) - f\left(\frac{k}{n}\right)\right| \left(\begin{array}{c} n \\ k \end{array} \right) x^k (1-x)^{n-k} \\ &\quad + \sum_{\left|\frac{k}{n}-x\right|>\delta} \left| f(x) - f\left(\frac{k}{n}\right)\right| \left(\begin{array}{c} n \\ k \end{array} \right) x^k (1-x)^{n-k} \\ &\leq \nu\delta \sum_{\left|\frac{k}{n}-x\right|<\delta} \left(\begin{array}{c} n \\ k \end{array} \right) x^k (1-x)^{n-k} + 2M \sum_{\left|\frac{k}{n}-x\right|>\delta} \left(\begin{array}{c} n \\ k \end{array} \right) x^k (1-x)^{n-k} \\ &\leq \nu\delta + \frac{M}{2n\delta^2}, \end{aligned}$$

car $\sum_{\left|\frac{k}{n}-x\right|>\delta} \left(\begin{array}{c} n \\ k \end{array} \right) x^k (1-x)^{n-k} \leq \frac{1}{4n\delta^2}$.

Ceci étant vrai pour tout $\delta > 0$, c'est vrai en particulier pour le minimum de la fonction $\delta \mapsto \nu\delta + \frac{M}{2n\delta^2}$, qui vaut $\frac{3}{2}\nu^{\frac{2}{3}} M^{\frac{1}{3}} \frac{1}{\sqrt[3]{n}}$. □

Corollaire 3.1 *Soit $\phi \in \mathcal{C}^1([0,g])$ alors $\|\phi - B_k(\phi)\|_\infty \leq \frac{3}{2} g^{\frac{2}{3}} \|\phi\|^{\frac{1}{3}} \|\phi'\|^{\frac{2}{3}} . \frac{1}{\sqrt[3]{k}}$.*

Preuve On pose $f(y) = \phi(gy)$; f est définie sur [0,1] et est $g\|\phi'\|$-lipschitzienne. □

On pose pour la suite $\bar{M} = \frac{3}{2} g^{\frac{2}{3}} \|\phi\|^{\frac{1}{3}} \|\phi'\|^{\frac{2}{3}}$, pour avoir $\|\phi - \mathrm{B}_k(\phi)\|_\infty \leq \bar{M} \frac{1}{\sqrt[3]{k}}$. Le théorème 3.2 ne permet pas de conclure directement car $B_k(\phi) \notin \mathbb{R}_{k+1,2}[X]$. En effet, $B'_k(\phi)(0) \neq 0$. Il faut donc appliquer le théorème à $\phi - B_k(\phi) - XB'_k(\phi)(0)$, qui est dans $\mathbb{R}_{k+1,2}[X]$. Le lemme 3.3 donne une majoration de $\|XB'_k(\phi)(0)\|_\infty$, puis le théorème 3.3 exprime la vitesse de convergence du filtre.

Lemme 3.3 *Soit $\phi \in \mathcal{C}^1([0,g])$. On suppose qu'il existe $h > 0$, tel que ϕ soit deux fois dérivable sur [0,h] et que ϕ'' soit bornée sur [0,h]. Alors $k\phi\left(\frac{1}{k}\right) \leq \frac{1}{k} \max_{t\in[0,h]} |\phi''(t)|$.*

Preuve D'après l'inégalité de Taylor-Lagrange, on a

$$\left|\phi\left(\frac{1}{k}\right) - \phi(0) - \frac{1}{k}\phi'(0)\right| \leq \frac{1}{k^2} \max_{t\in[0,h]} |\phi''(t)|.$$

D'où $k\phi\left(\frac{1}{k}\right) \leq \frac{1}{k}\max_{t\in[0,h]} |\phi''(t)|$. □

Théorème 3.3 *Soit $\phi \in \mathcal{C}^1([0,g])$. On suppose qu'il existe $h > 0$, tel que ϕ soit deux fois dérivable sur $[0,h]$ et que ϕ'' soit bornée sur $[0,h]$. Alors $\|\phi - \phi_k\|_{\langle,\rangle} \in O\left(\frac{1}{\sqrt[3]{k}}\right)$.*

Preuve

$$\begin{aligned}
\|\phi - \phi_k\|_{\langle,\rangle} &\leq \|\phi - \left(B_k(\phi) - XB_k'(\phi)(0)\right)\|_{\langle,\rangle} \\
&\leq \sqrt{(1+\rho)\pi}\|\phi - \left(B_k(\phi) - XB_k'(\phi)(0)\right)\|_\infty \\
&\leq \sqrt{(1+\rho)\pi}\left(\|\phi - B_k(\phi)\|_\infty + \|XB_k'(\phi)(0)\|_\infty\right) \\
&\leq \sqrt{(1+\rho)\pi}\left(\frac{\bar{M}}{\sqrt[3]{k}} + gk\phi\left(\frac{1}{k}\right)\right) \\
&\leq \sqrt{(1+\rho)\pi}\left(\frac{\bar{M}}{\sqrt[3]{k}} + \frac{g\|\phi''\|}{k}\right).
\end{aligned}$$

Ceci implique alors que $\|\phi - \phi_k\|_{\langle,\rangle} \in O\left(\frac{1}{\sqrt[3]{k}}\right)$. □

Les hypothèses du théorème 3.3 sont bien évidemment toujours satisfaites pour une définition de ϕ comme un polynôme par morceaux.

3.4 Essais numériques

3.4.1 Le problème de test

3.4.2 L'implémentation

L'algorithme 3.1 a été implémenté en Matlab. Il y a deux modules principaux :

- le premier calcule le polynôme solution à partir d'un filtre donné. Ici le filtre est de la forme 3.11. Il n'y a donc que deux paramètres a et g à fixer. Le paramètre g, supérieur à la plus grande valeur propre, est estimé en fonction de la matrice A ; on ne peut donc jouer que sur a pour trouver la meilleure régularisation. Pendant le calcul, on garde en mémoire les composantes de tous les polynômes dans les deux bases $\mathcal{T}_j^{(0,a)}$ et $\mathcal{T}_j^{(a,b)}$, pour un calcul aisé des produits scalaires. Le coût de ce calcul est négligeable.
- le second module applique le polynôme solution au vecteur b. On calcule $Q(A)b$ en réappliquant la récurrence qui a permis de calculer Q. Le coût est essentiellement dû au produit matrice-vecteur ; il y en a $m = \deg(Q)$. Le coût final est donc en $\mathrm{O}(mN^2)$ où N est la taille de A.

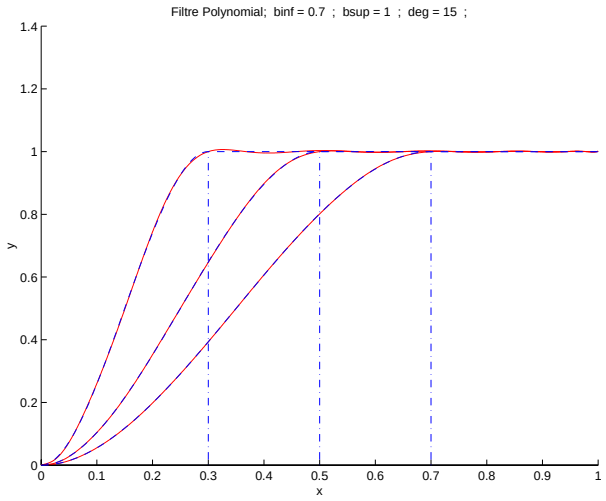


FIG. 3.1 – *Polynômes filtres*

Pour les essais numériques, on génère un problème en appliquant un flou à une image, puis en la bruitant. Ici la matrice de flou est la matrice générée par la fonction *blur* du package *Regularization Tools [22]*. Les paramètres sont les suivants :

- N vaut la taille de l'image ;
- *band* est fixé à 8 ;
- *sigma* est fixé à 1.7.

La matrice produite est symétrique et toutes ses valeurs propres sont entre 0 et 1. On utilise donc 1 comme borne supérieure de l'intervalle de définition du filtre, a variant entre 0 et 1.

3.4.3 Les polynômes calculés

Pour la figure 3.1, on a choisi d'aller jusqu'au degré 15. On peut y voir le filtre polynomial par morceaux ϕ_a^C en pointillés et le filtre polynomial $\phi_{15,a}$ en trait plein. Les traits pointillés verticaux désignent les différentes valeurs de a. Pour l'exemple, on a choisi trois valeurs : 0.3, 0.5 et 0.7.

Sur la figure 3.2, on voit les polynômes Q_{15} utilisés pour produire la solution. Ils sont très proche de la fonction $\frac{1}{x}$ sur l'intervalle $[a,1]$ et sur $[0,a]$, ils reviennent vers 0 pour aténuer l'effet des petites valeurs propres.

3.4.4 Les images

L'image de départ est une photographie d'Albert Einstein. L'image est de taille 256 par 256 et chaque pixel est codé sur 256 niveaux de gris. La matrice A est donc de taille 65536 par 65536. Sur le premier test (figures 3.3 et 3.4), il n'y a pas de bruit. On

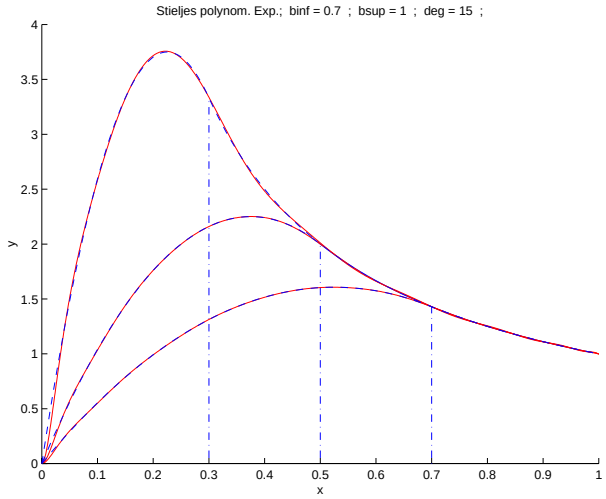


FIG. 3.2 – *Polynômes solution*

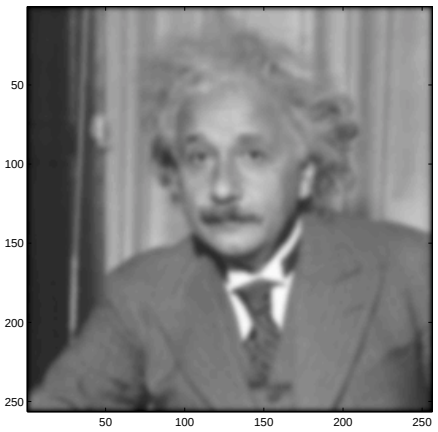


FIG. 3.3 – *Image floue*

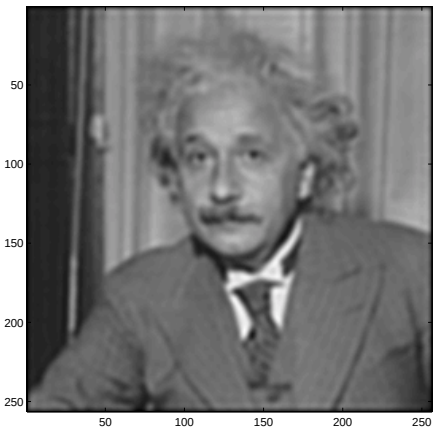


FIG. 3.4 – *Image restaurée avec* $a = 0.5$

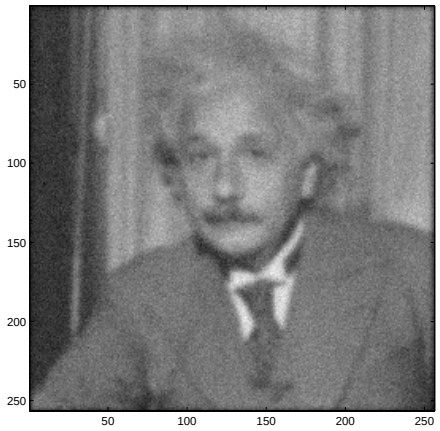


FIG. 3.5 – *Image floue bruitée*

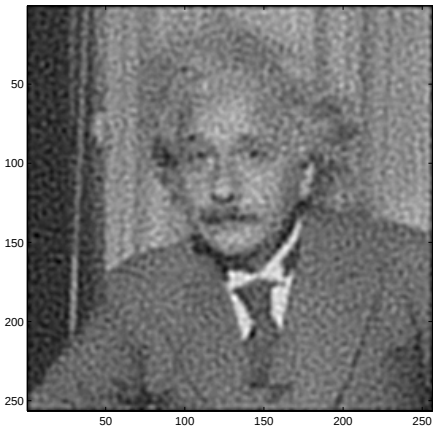


FIG. 3.6 – *Image restaurée avec* $a = 0.3$

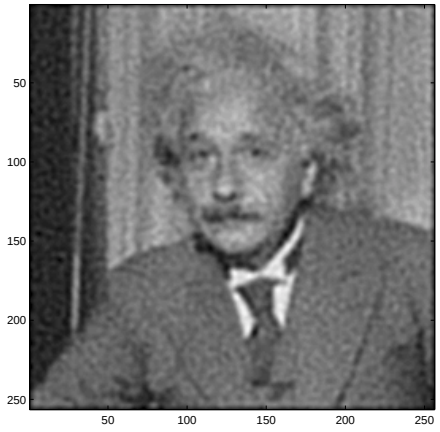


FIG. 3.7 – *Image restaurée avec* $a = 0.5$

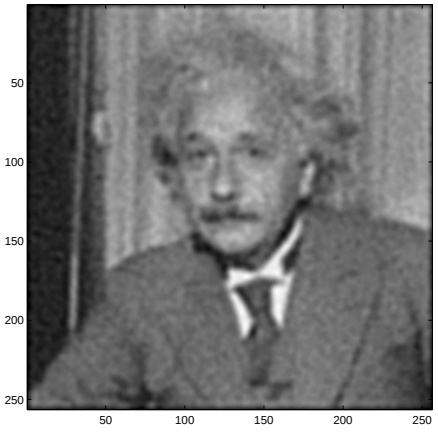


FIG. 3.8 – *Image restaurée avec* $a = 0.7$

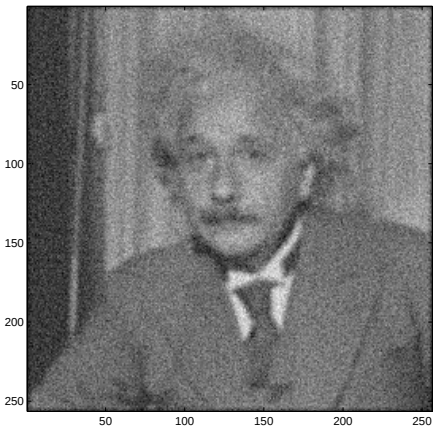


FIG. 3.9 – *Image restaurée par Tikhonov*

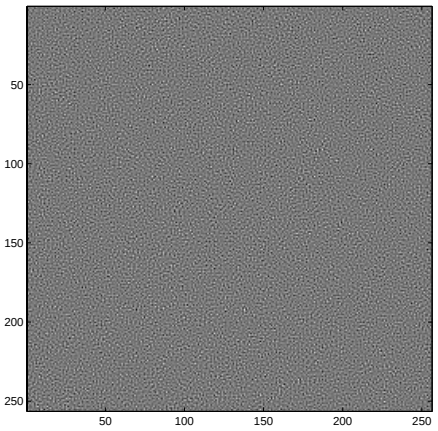


FIG. 3.10 – *Image restaurée par GC*

se contente d'enlever le flou. Ce test est là pour valider le solveur dans un cas «facile», c'est-à-dire sans bruit.

Pour la suite, c'est-à-dire les images 3.5 à 3.10, on ajoute après le flou et pour chaque pixel, un bruit gaussien de moyenne nulle et d'écart type 10. Les images 3.6, 3.7 et 3.8, montrent la restauration de l'image floue bruitée par la méthode des filtres polynomiaux, pour un paramètre a valant respectivement 0.3, 0.5 et 0.7. On peut voir que plus ce paramètre est grand plus l'image est lissée, et donc on enlève moins de flou (image 3.8). De l'autre coté (image 3.6), pour une trop petite valeur de a, l'image restaurée est dégradée par le bruit initial, qui est amplifié par l'inversion de A. Pour la valeur $a = 0.5$ (image 3.7), la restauration est bonne. Malheureusement, le choix automatique de ce paramètre optimal est encore un problème ouvert.

Les images 3.9 et 3.10 montrent, pour comparaison, les résultats obtenus avec, respectivement la régularisation de Tikhonov ($\rho = 1$) et le gradient conjugué sans régularisation. On voit que la régularisation de Tikhonov, dont la fonction filtre est plus plate, donne une image plus lisse, plus floue ; mais elle est moins sensible au bruit. Pour le gradient conjugué, le bruit est devenu dominant, et l'image restaurée ne vaut plus rien. Ce résultat ne dépend pas du nombre d'itérations. Ici, le bruit ajouté est trop important pour qu'une résolution avec une régularisation jouant sur le nombre d'itérations du gradient conjugué puisse se faire.

3.5 Conclusion

Dans ce chapitre est présentée une nouvelle méthode pour la régularisation de systèmes mal-conditionnés. Cette méthode n'impose pas de filtre particulier, mais calcule des approximations polynomiales d'un filtre fixé par l'utilisateur. La seule contrainte sur ce filtre est qu'il doit être défini comme un polynôme par morceaux.

Le coût d'une itération de cet algorithme est similaire à celui d'une itération du gradient conjugué et la vitesse de convergence des filtres est en $O(\frac{1}{\sqrt[3]{k}})$. Malheureusement le problème de la convergence de x_k est toujours ouvert.

Les tests numériques montrent que la méthode se comporte bien, et on peut voir, sur les images restaurées, que les contours ressortent plus nettement qu'avec la régularisation de Tikhonov. Il faut cependant bien choisir le paramètre a. Le choix automatique de ce paramètre doit pouvoir se faire par des techniques telles que les courbes en L ou la validation croisée [20]. Mais cela n'est pas encore confirmé par l'expérimentation.

Chapitre 4

Accélération de la résolution des systèmes linéaires

4.1 Cadre

On va dans ce chapitre étudier comment accélérer la résolution de systèmes linéaires lorsqu'on a déjà résolu au moins un système avec la matrice $A \in \mathcal{M}_n(\mathbb{R})$, symétrique définie positive. C'est-à-dire comment diminuer le nombre d'itérations nécessaires pour converger. Pour cela on décompose l'espace en deux : un premier espace, dont une base forme la matrice W, et qui contiendra les directions qui nuisent à la convergence. Typiquement cet espace contient les directions propres associées aux plus petites valeurs propres, car, comme c'est montré dans [13], ce sont les directions qui gènent le plus le gradient conjugué dans sa convergence, car elles y introduisent des paliers. On pose alors

$$D = W^* A W.$$

Si on choisit de représenter W dans une base A-orthogonale, alors D est diagonale. C'est ce qui arrive si, par exemple, on choisit les vecteurs de descente du gradient conjugué [15], ou des vecteurs propres exacts de A. Cela n'est plus vrai si les vecteurs de W ne sont que des approximations de vecteurs propres [38].

Le second espace est, lui, un supplémentaire de W, pour que leur somme (directe) soit l'espace tout entier. En fonction des besoins, on peut prendre le supplémentaire orthogonal ou le supplémentaire A-orthogonal. Pour cela, on introduit les deux projections suivantes :

$$H = I - W\left(W^* A W\right)^{-1} W^* A,$$

qui est la projection A-orthogonale sur $W^{\perp_A}$ le long de W ; et

$$H^* = I - AW\left(W^* A W\right)^{-1} W^*,$$

qui est la projection A^{-1}-orthogonale sur $W^{\perp}$ le long de AW.

Ces projections rendent le diagramme suivant commutatif.

$$\begin{array}{ccc} \mathbb{R}^n & \xrightarrow{A} & \mathbb{R}^n \\ {\scriptstyle H}\downarrow & & \downarrow{\scriptstyle H^*} \\ \mathbb{R}^n & \xrightarrow[A]{} & \mathbb{R}^n \end{array} \tag{4.1}$$

C'est à dire que les matrices vérifient les égalités suivantes

$$\begin{cases} AH = H^*A \\ H^2 = H \\ (H^*)^2 = H^*. \end{cases}$$

On a donc finalement

$$AH = H^*A = H^*AH. \tag{4.2}$$

4.2 Amélioration du vecteur de départ

4.2.1 Le choix de x_0

Supposons maintenant, que l'on ait résolu un premier système $Ay = c$, par la méthode du gradient conjugué. On appelle $w_1, \ldots$ les directions de descente successivement engendrées et $s_0, \ldots$ les résidus. Puis, on cherche à résoudre une seconde équation

$$Ax = b. \tag{4.3}$$

On peut alors, tout d'abord améliorer le point de départ du second système (4.3). On garde pour cela m vecteurs de l'espace engendré lors de la résolution du premier système (m paramètre fixé), et on appelle alors $W = [w_1, \ldots, w_m]$.

Une première idée, introduite dans [34, 35, 43] pour la méthode de Lanczos, est de choisir une solution initiale x_0 telle que le résidu initial $r_0 = b - Ax_0$ soit orthogonal à l'espace de Krylov $\mathcal{K}_m(A, s_0)$, car grâce au théorème 1.3, on peut affirmer que x_0 minimise sur W l'erreur initiale en norme A. On veut donc garantir la propriété suivante :

$$W^*r_0 = 0. \tag{4.4}$$

Soit x_{-1} une première approximation de la solution et $r_{-1} = b - Ax_{-1}$, le résidu correspondant. Pour garantir que la solution initiale x_0 satisfait la condition d'orthogonalité $W^*r_0 = 0$, on peut choisir x_0 de la forme

$$x_0 = x_{-1} + W\left(W^*AW\right)^{-1} W^*r_{-1} \tag{4.5}$$

$$r_0 = b - Ax_0 = H^*r_{-1} \tag{4.6}$$

où x_{-1} est arbitraire, par exemple une solution initiale qui ne satisferait pas la condition d'orthogonalité, et $r_{-1} = b - Ax_{-1}$ est le résidu associé à x_{-1}. Ce x_0 est même la seule solution, si on impose $x_0 \in x_{-1} + W$. En effet, on peut alors écrire $x_0 = x_{-1} + Wy$ pour un certain vecteur $y \in \mathbb{R}^m$, et donc $r_0 = r_{-1} - AWy$. On a alors

$$\begin{aligned} & W^* r_0 = 0 \\ \Rightarrow \quad & W^* r_{-1} - W^* AWy = 0 \\ \Rightarrow \quad & y = (W^* AW)^{-1} W^* r_{-1} \\ \Rightarrow \quad & x_0 = x_{-1} + W (W^* AW)^{-1} W^* r_{-1}. \end{aligned}$$

Cette formule (4.5) est aussi utilisée dans [15, 34, 35, 43].

Si on ne sait rien de plus sur x_{-1}, on peut toujours prendre $x_{-1} = y_0$, par exemple. Alors $r_{-1} = b - Ax_{-1} = b - c + s_0$ et $r_0 = H^* r_{-1} = H^*(b - c) + s_m$. On peut aussi choisir $x_{-1} = y_j$ avec $j > m$, le nombre d'itérations du premier système. Alors $r_{-1} = b - Ax_{-1} = b - c + s_j$ et $r_0 = H^* r_{-1} = H^*(b - c) + s_j$. En général, cette seconde solution sera meilleure car $\|s_j\|_{A^{-1}} \leq \|s_m\|_{A^{-1}}$. Mais de toutes façons, la qualité de la solution initiale dépendra de $\|H^*(b - c)\|_{A^{-1}}$.

On appelle gradient conjugué avec projection initiale (Init-CG) cette variante du gradient conjugué, qui est résumée dans l'algorithme 4.1.

Require: x_{-1}
1: $r_{-1} = b - Ax_{-1}$
2: $x_0 = x_{-1} + WD^{-1}W^* r_{-1}$
3: $r_0 = b - Ax_0$
4: $p_0 = r_0$
5: **for** $k = 0,1,\ldots$ **do**
6: $\quad \alpha_k = \frac{r_k^* r_k}{p_k^* A p_k}$
7: $\quad x_{k+1} = x_k + \alpha_k p_k$
8: $\quad r_{k+1} = r_k - \alpha_k A p_k$
9: $\quad \beta_{k+1} = \frac{r_{k+1}^* r_{k+1}}{r_k^* r_k}$
10: $\quad p_{k+1} = r_{k+1} + \beta_{k+1} p_k$
11: **end for**

Algorithme 4.1: Algorithme du gradient conjugué avec projection initiale

4.2.2 Lien avec le gradient conjugué par blocs

InitCG est toujours une méthode de type gradient conjugué. Mais pendant les $\frac{m}{2}$ premières itérations, cet algorithme est équivalent à une méthode de gradient conjugué par blocs.

Théorème 4.1 *Soit k le nombre d'itérations pendant la seconde résolution. Tant que $k < m/2$, l'algorithme du gradient conjugué appliqué au système linéaire $Ax = b$,*

démarré avec une solution initiale x_0 donnée par (4.5) et un résidu initial donnés par (4.6), est mathématiquement équivalent à l'algorithme du gradient conjugué par blocs avec des blocs de taille 2, démarrant avec le bloc initial $[s_0,r_0]$.

Bien sûr, ce résultat n'a pas d'intérêt pratique parce que r_0 n'est pas connu avant que W soit calculé, si bien qu'on ne peut commencer le gradient conjugué par blocs sans avoir déjà résolu le premier système.

La preuve de ce théorème suit celle de [35], donnée pour la méthode Lanczos.

Preuve Soit $S_j = (s_0,s_1,\ldots,s_j)$ et $R_k = (r_0,r_1,\ldots,r_k)$. On montre tout d'abord que S_j est orthogonal à R_k pour tout (j,k) tel que $j+k \leq m$.

Soient s et r deux vecteurs de S_j et R_k respectivement. D'après (1.15) page 17, il existe deux polynômes $Q \in \mathbb{R}_j[X]$ et $P \in \mathbb{R}_k[X]$ tels que $s = Q(A)s_0$ et $r = P(A)r_0$. On a alors

$$s^*r = (Q(A)s_0)^*(P(A)r_0)$$

et il suffit alors de prouver l'orthogonalité pour les monômes comme A^is_0 et A^lr_0 avec $i \leq j$ et $l \leq k$. Cette propriété vient alors directement du choix du résidu initial r_0. En effet, comme A est symétrique,

$$(A^is_0)^*(A^lr_0) = (A^{i+l}s_0)^*r_0.$$

Et maintenant, comme $i+l \leq j+k \leq m$, $A^{i+l}s_0 \in \mathcal{K}_m(A,s_0) = W$ et par construction $W^*r_0 = 0$. On en conclut que $(A^is_0)^*(A^ir_0) = 0$ et ainsi $s^*r = 0$.

Maintenant, on considère l'algorithme du gradient conjugué par blocs, démarré avec le bloc $[s_0,r_0]$, r_0 défini par (4.6). On appelle $[s_k,r_k]$ le bloc de taille 2 des résidus et $[w_k,p_k]$ le bloc de taille 2 des directions de descente. L'algorithme sans préconditionnement, donne

$$[s_{k+1},r_{k+1}] = [s_k,r_k] - A[w_k,p_k]\eta_k$$

avec

$$\eta_k = ([w_k,p_k]^*A[w_k,p_k])^{-1}[s_k,r_k]^*[s_k,r_k].$$

Tant que $2k+1 \leq m$, on a $S_k \perp R_k$, donc $p_k^*Aw_k = 0$ car $Aw_k \in S_{k+1}$ et $p_k \in R_k$. D'où, pour $k < \frac{m}{2}$, on a $w_k^*Ap_k = p_k^*Aw_k = 0$. On en déduit alors :

$$[w_k,p_k]^*A[w_k,p_k] = \begin{pmatrix} w_k^*Aw_k & 0 \\ 0 & p_k^*Ap_k \end{pmatrix}.$$

Aussi $s_k^*r_k = 0$ car $s_k \in S_k$ et $r_k \in R_k$, donc

$$[s_k,r_k]^*[s_k,r_k] = \begin{pmatrix} s_k^*s_k & 0 \\ 0 & r_k^*r_k \end{pmatrix}.$$

D'où on déduit

$$\eta_k = \begin{pmatrix} s_k^*s_k/w_k^*Aw_k & 0 \\ 0 & r_k^*r_k/p_k^*Ap_k \end{pmatrix} = \begin{pmatrix} \gamma_k & 0 \\ 0 & \alpha_k \end{pmatrix}$$

et

$$\begin{cases} s_{k+1} = s_k - \gamma_k A w_k \\ r_{k+1} = r_k - \alpha_k A p_k \end{cases}$$

qui sont exactement les formules des algorithmes 1.3 et 4.1. On procède de même pour le calcul de $[w_k,p_k]$. □

4.3 Amélioration du comportement asymptotique

4.3.1 La procédure de Lanczos avec projection

Soit $A \in \mathcal{M}_n(\mathbb{R})$, symétrique définie positive, k vecteurs linéairement indépendants $w_1,w_2,\ldots,w_k$ et v_1 vecteur unitaire orthogonal à w_i pour $i = 1,2,\ldots,k$. On pose $W = [w_1,w_2,\ldots,w_k]$. Comme A est définie positive, la matrice W^*AW est non-singulière.

L'algorithme 4.2, que l'on appelera algorithme de Lanczos avec projection, construit une suite de vecteurs $\{v_j\}_{j=1,2,\ldots}$ telle que

$$v_{j+1} \perp [W,v_1,v_2,\ldots,v_j] \tag{4.7}$$

et

$$\|v_{j+1}\|_2 = 1. \tag{4.8}$$

Pour cela, on applique la procédure de Lanczos standard 1.2, [36, p.174] à la matrice auxiliaire

$$B = AH = A - AW\left(W^*AW\right)^{-1}W^*A \tag{4.9}$$

avec pour vecteur initial v_1. La matrice B est symétrique positive mais pas définie positive. On a en effet $\ker B = W$. La singularité de cette matrice pourrait causer un breakdown. Cependant le fait de produire des vecteurs dans $W^\perp$ assure la bonne terminaison de l'algorithme. C'est le résultat du théorème 4.5 page 79.

La suite $\{v_j\}_{j=1,2,\ldots}$ vérifie alors

$$BV_j = V_jT_j + \sigma_{j+1}v_{j+1}e_j^*, \tag{4.10}$$

où

$$T_j = \begin{pmatrix} \rho_1 & \sigma_2 & & & \\ \sigma_2 & \rho_2 & \sigma_3 & & \\ & \ddots & \ddots & \ddots & \\ & & \sigma_{j-1} & \rho_{j-1} & \sigma_j \\ & & & \sigma_j & \rho_j \end{pmatrix}$$

et où $V_j = [v_1,v_2,\ldots,v_j]$ et e_j est le dernier élément de la base canonique de $\mathbb{R}^j$.

La procédure de Lanczos garantit que les vecteurs v_j sont orthogonaux entre eux. De (4.10), on tire

$$\sigma_{j+1}v_{j+1} = Bv_j - \sigma_jv_{j-1} - \rho_jv_j.$$

Une récurrence immédiate donne alors pour $j = 1,2,\ldots,$ $v_{j+1}^* W = 0$: on a bien les propriétés (4.7) et (4.8).

En substituant B dans la procédure de Lanczos, on obtient l'algorithme 4.2.

Require: $W = [w_1, w_2, \ldots, w_k]$
Require: v_1 vecteur initial tel que $v_1^* W = 0$ et $\|v_1\|_2 = 1$
1: **for** $j = 1,2,\ldots,m$ **do**
2: Solve $W^* A W \hat{\nu}_j = W^* A v_j$ for $\hat{\nu}$
3: $z_j = A v_j - A W \hat{\nu}_j = B v_j$
4: $\rho_j = v_j^* z_j$
5: $\hat{v}_{j+1} = z_j - \sigma_j v_{j-1} - \rho_j v_j$
6: $\sigma_{j+1} = \|\hat{v}_{j+1}\|_2$
7: **if** $\sigma_{j+1} = 0$ **then break**
8: $v_{j+1} = \frac{\hat{v}_{j+1}}{\sigma_{j+1}}$
9: **end for**

Algorithme 4.2: Algorithme de Lanczos avec Projection

4.3.2 L'algorithme du gradient conjugué avec projection

Tout comme on peut dériver la méthode du gradient conjugué de la procédure de Lanczos, on peut alors dériver l'algorithme du gradient conjugué avec projection de l'algorithme 4.2. Pour cela, il faut tout d'abord revenir au système à résoudre:

$$Ax = b, \tag{4.11}$$

avec A symétrique définie positive. On recherche une méthode de projection, dérivée de 4.2, en s'inspirant d'une technique standard décrite dans [36, p.179].

Les w_i et les v_j sont toujours comme définis dans l'algorithme de Lanczos projeté 4.2 et W et V_j désignent l'espace engendré par ces vecteurs. On suppose aussi que le vecteur initial x_0 est tel que $r_0 = b - Ax_0 \perp W$. On pose $v_1 = r_0/\|r_0\|_2$ et

$$\mathcal{K}_{k,j}(A,W,r_0) = [W,V_j].$$

À la j^{e} étape de la méthode, on recherche une solution approchée x_j telle que

$$x_j \in x_0 + \mathcal{K}_{k,j}(A,W,r_0) \tag{4.12}$$

et

$$r_j = b - Ax_j \perp \mathcal{K}_{k,j}(A,W,r_0). \tag{4.13}$$

Lemme 4.1 *Si x_j et r_j satisfont (4.12) et (4.13), alors $\exists c_j \in \mathbb{R}$ tel que*

$$r_j = c_j v_{j+1}. \tag{4.14}$$

On a aussi $\mathcal{K}_{k,j}(A,W,r_0) = [W,r_0,r_1,\ldots,r_{j-1}]$ et les résidus r_j sont deux-à-deux orthogonaux.

Preuve D'après (4.12), la solution x_j peut s'écrire

$$x_j \quad = \quad x_0 + W\hat{\xi}_j + V_j\hat{\eta}_j \tag{4.15}$$

pour un certain $\hat{\xi}_j$ et $\hat{\eta}_j$. De plus, d'après (4.9) et (4.10) on a

$$AV_j = AW\Delta_j + V_jT_j + \sigma_{j+1}v_{j+1}e_j^*,$$

où $\Delta_j = (W^*AW)^{-1}\,W^*AV_j$. D'où

$$\begin{aligned} r_j &= r_0 - AW\hat{\xi}_j - AV_j\hat{\eta}_j \\ &= r_0 - AW\hat{\xi}_j - \left(AW\Delta_j + V_jT_j + \sigma_{j+1}v_{j+1}e_j^*\right)\hat{\eta}_j. \end{aligned} \tag{4.16}$$

En multipliant (4.16) par W^*, et en utilisant les conditions d'orthogonalité (4.7) et (4.13), on obtient immédiatement le système suivant pour $\hat{\xi}_j$ et $\hat{\eta}_j$,

$$W^*AW\hat{\xi}_j + W^*AW\Delta_j\hat{\eta}_j = 0.$$

Or W^*AW est non singulière, donc $\hat{\xi}_j = -\Delta_j\hat{\eta}_j$. Donc (4.15) et (4.16) deviennent

$$x_j = x_0 - W\Delta_j\hat{\eta}_j + V_j\hat{\eta}_j$$

et

$$r_j = r_0 - V_jT_j\hat{\eta}_j - \sigma_{j+1}v_{j+1}e_j^*\hat{\eta}_j. \tag{4.17}$$

De plus, parce que $r_0 = \|r_0\|_2 v_1$ par définition et que $V_j^*r_j = 0$ d'après (4.13), on a

$$V_j^*r_j = V_j^*r_0 - T_j\hat{\eta}_j = \|r_0\|_2 e_1 - T_j\hat{\eta}_j = 0,$$

avec e_1 le premier vecteur de la base canonique de $\mathbb{R}^j$. D'où

$$\hat{\eta}_j = \|r_0\|_2 T_j^{-1}e_1. \tag{4.18}$$

C'est-à-dire, en substituant (4.18) dans (4.17), que $\exists c_j = -\sigma_{j+1}\|r_0\|_2\left(e_j^*T_j^{-1}e_1\right)$ tel que

$$r_j = c_jv_{j+1}.$$

□

Soit $T_j = L_jD_jL_j^*$ la décomposition LDL^* de la matrice symétrique T_j. On pose

$$P_j = [p_0,p_1,\ldots,p_{j-1}] = (-W\Delta_j + V_j)\,L_j^{-*}\Lambda_j \tag{4.19}$$

avec $\Lambda_j = \begin{bmatrix} c_0 & & & \\ & c_1 & & \\ & & \ddots & \\ & & & c_{j-1} \end{bmatrix}$.

Proposition 4.1 *La solution x_j, le résidu r_j, et la direction de descente p_j satisfont les récurrences suivantes*

$$\begin{aligned} x_j &= x_{j-1} + \alpha_{j-1} p_{j-1}, \\ r_j &= r_{j-1} - \alpha_{j-1} A p_{j-1}, \\ p_j &= r_j + \beta_{j-1} p_{j-1} - W \hat{\mu}_j \end{aligned} \tag{4.20}$$

avec α_{j-1},β_{j-1} réels et $\hat{\mu}_j \in \mathbb{R}^m$. On a aussi $\mathcal{K}_{k,j}(A,W,r_0) = [W,p_0,p_1,\ldots,p_{j-1}]$

Preuve Soit

$$\hat{\zeta}_j = \|r_0\|_2 \, (L_j D_j \Lambda_j)^{-1} \, e_1.$$

La solution approchée est donnée par

$$\begin{aligned} x_j &= x_0 + \|r_0\|_2 \, (-W\Delta_j + V_j) \, T_j^{-1} e_1 \\ &= x_0 + P_j \hat{\zeta}_j. \end{aligned}$$

De plus, comme $L_j D_j \Lambda_j$ est triangulaire inférieure, on a la relation suivante

$$\hat{\zeta}_j = \begin{bmatrix} \hat{\zeta}_{j-1} \\ \alpha_{j-1} \end{bmatrix}$$

pour un certain scalaire α_{j-1}. On en déduit immédiatement la relation de récurrence pour x_j.

On réécrit maintenant (4.19) de la façon suivante

$$P_j \Lambda_j^{-1} L_j^* \Lambda_j = -W\Delta_j \Lambda_j + V_j \Lambda_j.$$

Or $\Lambda_j^{-1} L_j^* \Lambda_j$ est bidiagonale, triangulaire supérieure; d'où, en comparant les dernières colonnes des deux côtés de l'équation précédente, on a

$$p_{j-1} - \beta_{j-2} p_{j-2} = -W\hat{\mu}_{j-1} + c_{j-1} v_j,$$

avec $\beta_{j-2} = -c_{j-1} u_{j-1,j} / c_{j-2}$ et $\hat{\mu}_{j-1} = c_{j-1} \hat{\nu}_j$. On a alors la récurrence pour p_j. □

Proposition 4.2 *Les vecteurs p_j sont deux-à-deux A-orthogonaux , c'est à dire que $P_j^* A P_j$ est diagonale. De plus, ils sont aussi A-orthogonaux à tous les w_i, c'est-à-dire, $W^* A P_j = 0$.*

Preuve En effet

$$\begin{aligned} P_j^* A P_j &= P_j^* \left(-AW\Delta_j + AV_j\right) L_j^{-*}\Lambda_j \\ &= P_j^* \left(V_j T_j + \sigma_{j+1} v_{j+1} e_j^*\right) L_j^{-*}\Lambda_j \\ &= \Lambda_j^* L_j^{-1} \left(-W\Delta_j + V_j\right)^* \left(V_j T_j + \sigma_{j+1} v_{j+1} e_j^*\right) L_j^{-*}\Lambda_j \\ &= \Lambda_j^* L_j^{-1} T_j L_j^{-*}\Lambda_j \\ &= \Lambda_j^* D_j \Lambda_j \end{aligned}$$

est diagonale et

$$\begin{aligned} W^* A P_j &= W^* \left(-AW\Delta_j + AV_j\right) L_j^{-*}\Lambda_j \\ &= W^* \left(V_j T_j + \sigma_{j+1} v_{j+1} e_j^*\right) L_j^{-*}\Lambda_j \\ &= 0. \end{aligned}$$

□

En utilisant l'orthogonalité des r_j et l'A-orthogonalité des p_j, on peut exprimer les coefficients de (4.20) grâce aux vecteurs W,r_j et p_j.

Proposition 4.3 *Les coefficients du gradient conjugué avec projection satisfont les relations*

$$\begin{aligned} \alpha_j &= \frac{r_j^* r_j}{p_j^* A p_j}, \\ \hat{\mu}_j &= \left(W^* A W\right)^{-1} W^* A r_j, \\ \beta_j &= \frac{r_{j+1}^* r_{j+1}}{r_j^* r_j}. \end{aligned} \tag{4.21}$$

Preuve En multipliant (4.20) par r_{j-1}^* on a

$$\alpha_{j-1} = \frac{r_{j-1}^* r_{j-1}}{r_{j-1}^* A p_{j-1}} = \frac{r_{j-1}^* r_{j-1}}{\left(\beta_{j-2} p_{j-2} + r_{j-1} - W\hat{\mu}_{j-1}\right)^* A p_{j-1}} = \frac{r_{j-1}^* r_{j-1}}{p_{j-1}^* A p_{j-1}}.$$

De même, les expressions pour $\hat{\mu}_j$ et β_{j-1} sont obtenues en multipliant (4.20)par $(AW)^*$ et $(Ap_{j-1})^*$ respectivement. On a alors

$$\begin{aligned} \hat{\mu}_j &= \left(W^* A W\right)^{-1} W^* A r_j, \\ \beta_{j-1} &= -\frac{p_{j-1}^* A r_j}{p_{j-1}^* A p_{j-1}} = -\frac{1}{\alpha_{j-1}} \frac{\left(r_{j-1} - r_j\right)^* r_j}{p_{j-1}^* A p_{j-1}} \\ &= \frac{1}{\alpha_{j-1}} \frac{r_j^* r_j}{p_{j-1}^* A p_{j-1}} = \frac{r_j^* r_j}{r_{j-1}^* r_{j-1}}. \end{aligned}$$

□

Les relations (4.20) et (4.21) donnent alors l'algorithme 4.3 page suivante.

Require: k vecteurs linéairement indépendants w_1,w_2,... ,w_k. $W = [w_1,w_2,\ldots,w_k]$
1: x_0 solution initiale telle que $W^* r_0 = 0$, avec $r_0 = b - Ax_0$
2: Résoudre $W^*AW\hat{\mu}_0 = W^*Ar_0$
3: $p_0 = r_0 - W\hat{\mu}_0$
4: **for** $j = 1,2,\ldots,m$ **do**
5: $\quad \alpha_{j-1} = \frac{r_{j-1}^* r_{j-1}}{p_{j-1}^* Ap_{j-1}}$
6: $\quad x_j = x_{j-1} + \alpha_{j-1} p_{j-1}$
7: $\quad r_j = r_{j-1} - \alpha_{j-1} Ap_{j-1}$
8: $\quad \beta_{j-1} = \frac{r_j^* r_j}{r_{j-1}^* r_{j-1}}$
9: $\quad$ Résoudre $W^*AW\hat{\mu}_j = W^*Ar_j$
10: $\quad p_j = \beta_{j-1} p_{j-1} + r_j - W\hat{\mu}_j$
11: **end for**

Algorithme 4.3: Algorithme du Gradient Conjugué avec Projection

4.3.2.1 Choix du vecteur initial

De même que pour InitCG, on choisit (4.5) page 68, pour le calcul du vecteur initial :

$$x_0 = x_{-1} + WD^{-1}W^* r_{-1}, \qquad r_0 = b - Ax_0 = H^* r_{-1}. \tag{4.22}$$

On satisfait ainsi la condition d'orthogonalité initiale : $W^* r_0$.

4.3.2.2 Préconditionnement de l'algorithme

La version préconditionnée du gradient conjugué avec projection peut être obtenue de la manière suivante. On suppose que l'on veut résoudre le système

$$L^{-1}AL^{-*}y = L^{-1}b, \qquad x = L^{-*}y, \tag{4.23}$$

et on applique l'algorithme 4.3 au système 4.23. En redéfinissant les variables de la façon suivante

$$\begin{array}{lcl} L^{-*}W & \rightarrow & W, \\ L^{-*}y_j & \rightarrow & x_j, \\ Lr_j & \rightarrow & r_j, \\ L^{-*}p_j & \rightarrow & p_j, \end{array} \tag{4.24}$$

et en posant $M = LL^*$, on obtient l'algorithme 4.4 page suivante.

Quand W est la matrice nulle, l'algorithme 4.4 est exactement un gradient conjugué préconditionné standard 1.4. En plus des matrices A et M, cinq vecteurs et trois matrices de stockage sont requises : p, Ap, r, x, z, W, AW et W^*AW.

Require: k vecteurs linéairement indépendants $W = [w_1, w_2, \ldots, w_k]$
Require: x_0 tel que $W^* r_0 = 0$, avec $r_0 = b - Ax_0$
1: $z_0 = M^{-1} r_0$
2: Résoudre $W^* A W \hat{\mu}_0 = W^* A z_0$
3: $p_0 = -W\hat{\mu}_0 + z_0$
4: **for** $j = 1,2,\ldots,m$ **do**
5: $\quad \alpha_{j-1} = \frac{r_{j-1}^* z_{j-1}}{p_{j-1}^* A p_{j-1}}$
6: $\quad x_j = x_{j-1} + \alpha_{j-1} p_{j-1}$
7: $\quad r_j = r_{j-1} - \alpha_{j-1} A p_{j-1}$
8: $\quad z_j = M^{-1} r_j$
9: $\quad \beta_{j-1} = \frac{r_j^* z_j}{r_{j-1}^* z_{j-1}}$
10: $\quad$ Résoudre $W^* A W \hat{\mu}_j = W^* A z_j$
11: $\quad p_j = \beta_{j-1} p_{j-1} + z_j - W\hat{\mu}_j$
12: **end for**

Algorithme 4.4: Algorithme du Gradient Conjugué avec Projection, Préconditionné

4.3.2.3 Formalisme polynomial

La version polynomiale de l'algorithme 4.4 permet de dégager plus facilement les propriétés de l'espace des solutions $\mathcal{K}_{m,k}$. Pour cela il faut tout d'abord exprimer r_k et p_k grâce à des polynômes à deux variables, en A et H.

Théorème 4.2 *Pour $k \geq 0$, il existe deux polynômes à deux variables $\Phi_k(X,Y)$ et $\Psi_k(X,Y)$ in $\mathbb{R}\langle X,Y\rangle$ tels que $r_k = \Phi_k(A,H) r_0$ et $p_k = \Psi_k(A,H) r_0$.*

De plus, $\Phi_0(X,Y) = 1$, $\Psi_0(X,Y) = Y$ et on a les relations de récurrence suivantes dans l'espace des polynômes $\mathbb{R}\langle X,Y\rangle$:

$$\begin{aligned} \Phi_{k+1}(X,Y) &= \Phi_k(X,Y) - \alpha_k X \Psi_k(X,Y), \\ \Psi_{k+1}(X,Y) &= Y\Phi_{k+1}(X,Y) + \beta_{k+1}\Psi_k(X,Y). \end{aligned}$$

Preuve Comme $p_0 = Hr_0$, on définit immédiatement $\Phi_0(X,Y) = 1$ et $\Psi_0(X,Y) = Y$. Les récurrences sont déduites de l'algorithme 4.3 : il est en effet immédiat de réécrire la définition de r_{k+1} ainsi: $\Phi_{k+1}(A,H)r_0 = \Phi_k(A,H)r_0 - \alpha_k A\Psi_k(A,H)r_0$. Ce qui donne la relation de récurrence pour Φ_k. Grâce à la relation $Hr_{k+1} = r_{k+1} - W\hat{\mu}_{k+1}$, on réécrit la définition de p_{k+1} ainsi

$$\Psi_{k+1}(A,H)r_0 = H\Phi_{k+1}(A,H)r_0 + \beta_{k+1}\Psi_k(A,H)r_0,$$

ce qui donne la récurrence pour Ψ_k. □

On peut maintenant simplifier ces expressions polynomiales en introduisant des polynômes à une seule variable, en XY. On a, en effet, le résultat suivant

Théorème 4.3 *Pour $k \geq 0$, il existe des polynômes de degré k à une variable $\Xi_k(Z)$ et $\Omega_k(Z)$ de $\mathbb{R}_k[Z]$ tels que*

$$\Phi_k(X,Y) = \Xi_k(XY) \text{ et } \Psi_k(X,Y) = Y\Omega_k(XY). \tag{4.25}$$

Preuve La démonstration se fait par récurrence: le théorème est vrai pour $k = 0$, car $\Phi_0(X,Y) = 1$ et $\Psi_0(X,Y) = Y$. Supposons maintenant que ce soit vrai au rang k, alors

$$\begin{aligned}
\Phi_{k+1}(X,Y) &= \Phi_k(X,Y) - \alpha_k X \Psi_k(X,Y) \\
&= \Xi_k(XY) - \alpha_k XY\Omega_k(XY) \\
&= \Xi_{k+1}(XY). \\
\Psi_{k+1}(X,Y) &= Y\Phi_{k+1}(X,Y) + \beta_{k+1}\Psi_k(X,Y) \\
&= Y\Xi_{k+1}(XY) + \beta_{k+1}Y\Omega_k(XY) \\
&= Y(\Xi_{k+1}(XY) + \beta_{k+1}\Omega_k(XY)) \\
&= Y\Omega_{k+1}(XY).
\end{aligned}$$

Ces polynômes sont clairement de degré $k+1$. Cela démontre donc le tréorème 4.3. □

Cette formulation polynomiale conduit à une caractérisation de l'espace $R_k = \langle r_0, \ldots, r_{k-1}\rangle$ comme un espace de Krylov. Ce qui permet par la suite de donner une borne asymptotique de convergence grâce aux théorèmes existants pour l'algorithme du Gradient Conjugué.

Théorème 4.4 *Pour $k \geq 0, \langle r_0, \ldots, r_k\rangle = \mathcal{K}_k(H^*AH, r_0)$ et*

$$\mathcal{K}_{m,k}(A, s_0, r_0) = \mathcal{K}_m(A, s_0) \overset{\perp_A}{\oplus} \mathcal{K}_k(H^*AH, r_0).$$

Preuve Le polynôme $\Xi_k(Z)$ étant de degré k, $(\Xi_i(Z))_{i=0\ldots k}$ est une famille indépendante et forme une base de $\mathbb{R}_k[Z]$. D'où, parce que $r_k = \Xi_k(AH)r_0$,

$$\langle r_0, \ldots, r_k\rangle = \mathcal{K}_k(AH, r_0).$$

De plus, d'après (4.2) page 68, on a $H^*AH = AH$, d'où par récurrence $(H^*AH)^i r_0 = (AH)^i r_0$ et

$$\langle r_0, \ldots, r_{k-1}\rangle = \mathcal{K}_k(H^*AH, r_0).$$

La caractérisation de $\mathcal{K}_{m,k}$ s'en déduit alors tout de suite. □

4.3.2.4 Étude de la convergence

La première proposition définit le gradient conjugué avec projection comme une Balanced Projection Method (BPM). On peut alors en tirer les deux théorèmes 4.5 et 4.28,

qui, respectivement, prouvent que l'algorithme du gradient conjugué avec projection, converge et donne une borne asymptotique de la vitesse de convergence.

Proposition 4.4 *L'algorithme 4.3 est équivalent à une Balanced Projection Method [27], définie par l'espace des solutions*

$$x_{j+1} - x_j \in \mathcal{K}_{k,j}(A,W,r_0) \tag{4.26}$$

et la condition de Petrov-Galerkin

$$r_0 \perp W \quad et \quad r_j \perp \mathcal{K}_{k,j}(A,W,r_0). \tag{4.27}$$

Preuve On voit tout de suite par récurrence que $p_j \in \mathcal{K}_{m,j}(A,W,r_0)$ car la condition de l'espace des solutions est satisfaite par l'algorithme 4.3. Par construction, il satisfait la condition de Petrov-Galerkin. Réciproquement, la BPM définie par (4.26) et (4.27) satisfait toutes les relations de récurrence de l'algorithme du condient conjugué projeté 4.3. □

Le théorème suivant est alors une conséquence directe de cette théorie.

Théorème 4.5 *Soit A une matrice symétrique définie positive et W une famille de vecteurs linéairement indépendants. Soit x^* la solution exacte du système linéaire $Ax = b$. L'algorithme du gradient conjugué projeté appliqué au système linéaire $Ax = b$ ne connaît pas d'échec et la solution approchée x_j est le minimum unique de la norme de l'erreur $\|x_j - x^*\|_A$ sur tout l'espace affine $x_0 + \mathcal{K}_{k,j}(A,W,r_0)$. De plus il existe $\epsilon > 0$, indépendant de x_0, tel que pour tout k*

$$\|x_j - x^*\|_A \leq (1 - \epsilon) \|x_{j-1} - x^*\|_A.$$

Preuve Voir théorèmes 2.4, 2.6 et 2.7 dans [27]. □

On a aussi grâce à la caractérisation de l'espace des solutions le théorème suivant.

Théorème 4.6 *Soit κ le conditionnement de H^*AH. Alors*

$$\|x_* - x_j\|_A \leq 2 \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^j \|x_* - x_0\|_A. \tag{4.28}$$

Preuve Le théorème 4.3 montre que $r_j = P_j(AH)r_0$ où P_j est un polynôme de degré j. Ainsi, d'après (4.2), on a $r_j = P_j(H^*AH)r_0$. Puis la propriété de minimisation conduit au résultat souhaité. □

4.3.2.5 Formalisme de préconditionnement

Une autre façon d'arriver au même résultat est de voir la projection comme un préconditionnement. On peut effectivement montrer que l'algorithme du gradient conjugué avec projection est équivalent à l'algorithme du gradient conjugué préconditionné par une matrice symétrique semi-définie positive HH^*.

Par rapport à l'algorithme 4.3, on a introduit la matrice H en utilisant le fait que $H^*r_k = r_k$, $p_0 = Hr_0$ et que $r_{k+1} - \mu_{k+1}w_m = Hr_{k+1}$. On utilise aussi l'égalité suivante $r_k^*z_k = r_k^*r_k$ qui vient de $r_k^*z_k = r_k^*Hr_k = (H^*r_k)^*r_k = r_k^*r_k$.

Finalement, on voit que l'algorithme du gradient conjugué projeté est une forme de gradient conjugué préconditionné, avec une matrice symétrique semi-définie positive HH^*. Ce n'est pas classique car la matrice de préconditionnement est usuellement prise non-singulière. On peut alors rattacher l'algorithme 4.3 à la classification définie dans [1], et l'on peut alors voir le gradient conjugué projeté comme une version $Omin(A,C,A)$ de PCG, avec

$$C = HH^*.$$

Lemme 4.2 *L'algorithme du gradient conjugué projeté est équivalent à la version $Omin(A,C,A)$ du gradient conjugué préconditionné appliqué à A avec le préconditionnement $C = HH^*$ et avec un résidu initial r_0 tel que $r_0 \perp W$.*

Preuve Les relations $p_0 = -W\hat{\mu}_0 + r_0$ et $p_j = -W\hat{\mu}_j + \beta_{j-1}p_{j-1} + r_j$ peuvent se réécrire respectivement $p_0 = Hr_0$ et $p_j = Hr_j + \beta_{j-1}p_{j-1}$. Mais, comme $r_j \perp W$, on a $r_j = H^*r_j$, donc

$$p_0 = Cr_0 \text{ et } p_j = Cr_j + \beta_{j-1}p_{j-1}$$

Ce sont exactement les relations définissant $Omin(A,C,A)$. Il suffit alors de démontrer que les coefficients α_j et β_j sont les mêmes. Il suffit pour cela de voir si $r_j^*Cr_j = r_j^*r_j$. En effet on a $r_j^*Cr_j = r_j^*HH^*r_j = (H^*r_j)^*H^*r_j = r_j^*r_j$. □

On déduit de ce lemme l'algorithme 4.5, une réécriture de l'algorithme ProjCG, avec un formalisme de préconditionnement.

Ici le préconditionnement C est singulier et donc la convergence n'est pas garantie. Cependant, comme le résidu initial est orthogonal à W, le résultat suivant peut être démontré.

Théorème 4.7 *Le gradient conjugué projeté est équivalent à la version $Odir(A,C,A)$ de PCG appliquée à A et C et démarrée avec $r_0 \perp W$. Ainsi le gradient conjugué projeté converge.*

Preuve Comme $\alpha_j = r_j^*r_j$, on a $\alpha_j \neq 0$ sauf si x_j est la solution exacte. D'où le gradient conjugué projeté ne connaît pas d'échec et, comme démontré dans [1], les deux versions *Omin* et *Odir* sont équivalentes. Maintenant grâce au théorème 3.1 de [1], on prouve que le gradient conjugué projeté converge. □

Require: x_{-1} solution approchée
1: $r_{-1} = b - Ax_{-1}$
2: $x_0 = x_{-1} + WD^{-1}W^* r_{-1}$
3: $r_0 = b - Ax_0 = H^* r_{-1}$
4: $z_0 = Hr_0 = HH^* r_0$
5: $p_0 = z_0$
6: **for** $k = 0,1,\ldots$ **do**
7: $\quad \alpha_k = \frac{r_k^* z_k}{p_k^* A p_k}$
8: $\quad x_{k+1} = x_k + \alpha_k p_k$
9: $\quad r_{k+1} = r_k - \alpha_k A p_k$
10: $\quad z_{k+1} = Hr_{k+1} = HH^* r_{k+1}$
11: $\quad \beta_{k+1} = \frac{r_{k+1}^* z_{k+1}}{r_k^* z_k}$
12: $\quad p_{k+1} = z_{k+1} + \beta_{k+1} p_k$
13: **end for**

Algorithme 4.5: Algorithme du Gradient Conjugué avec Projection, formalisme de préconditionnement

On peut alors démontrer d'une autre manière le théorème 4.6 page 79.

Théorème 4.8 *L'algorithme du gradient conjugué projeté est équivalent au gradient conjugué, version Omin(A,I,A), appliqué au système linéaire $H^*AH\tilde{x} = H^*b$. Et donc, donc le taux asymptotique de convergence est gourverné par le conditionnement κ de H^*AH et il est donné par la formule (4.28).*

Preuve Le gradient conjugué projeté commence avec $x_0 = H\tilde{x}_0 + Wy_0$ donné par la formule (4.5). D'où

$$r_0 = H^* r_0 = H^*(b - Ax_0) = H^*b - H^*AH\tilde{x}_0 - H^*AWy_0 = H^*b - H^*AH\tilde{x}_0.$$

L'algorithme Omin(A,I,A) appliqué à $H^*AH\tilde{x} = H^*b$ donne cela :

Require: $\tilde{x}_0$ solution initiale
1: $\tilde{r}_0 = H^*b - H^*AH\tilde{x}_0$ et $\tilde{p}_0 = \tilde{r}_0$.
2: **for** $j = 1,2,\ldots$ **do**
3: $\quad\tilde{\alpha}_{j-1} = \frac{\tilde{r}_{j-1}^*\tilde{r}_{j-1}}{\tilde{p}_{j-1}^*H^*AH\tilde{p}_{j-1}}$
4: $\quad\tilde{x}_j = \tilde{x}_{j-1} + \tilde{\alpha}_{j-1}\tilde{p}_{j-1}$
5: $\quad\tilde{r}_j = \tilde{r}_{j-1} - \tilde{\alpha}_{j-1}H^*AH\tilde{p}_{j-1}$
6: $\quad\tilde{\beta}_{j-1} = \frac{\tilde{r}_j^*\tilde{r}_j}{\tilde{r}_{j-1}^*\tilde{r}_{j-1}}$
7: $\quad\tilde{p}_j = \tilde{r}_j + \beta_{j-1}\tilde{p}_{j-1}$
8: **end for**

Maintenant, on pose $x_j = H\tilde{x}_j + Wy_0$ et $p_j = H\tilde{p}_j$. Grâce à $r_j = H^*r_j$ et $H^*AWy_0 = 0$, on a

$$r_j = H^*(b - Ax_j) = H^*b - H^*AH\tilde{x}_j = \tilde{r_j}.$$

Maintenant

$$\tilde{p}_{j-1}^*H^*AH\tilde{p}_{j-1} = (H\tilde{p}_{j-1})^*A(H\tilde{p}_{j-1}) = p_j^*Ap_j.$$

En rassemblant ces égalités, on obtient $\tilde{\alpha}_j = \frac{r_j^*r_j}{p_j^*Ap_j}$ et $\tilde{\beta}_{j-1} = \frac{r_j^*r_j}{r_{j-1}^*r_{j-1}}$. Puis en réécrivant, l'algorithme ci-dessus avec x_j,r_j,p_j,on obtient exactement l'algorithme du gradient conjugué projeté. On peut alors appliquer le résultat classique sur le taux de convergence du gradient conjugué. □

4.3.3 L'algorithme du gradient conjugué avec espace augmenté

4.3.3.1 Simplification de la récurrence

On va alors voir comment on peut sous certaines conditions, simplifier l'orthogonalisation et obtenir une récurrence à quatre termes ; c'est-à-dire que les opérations avec la matrice W peuvent être, sous ces conditions, remplacées par des opérations de rang un. On a en effet le théorème suivant :

Théorème 4.9 *Soit $W = [w_1,\ldots,w_m]$ vérifiant les propriétés suivantes : il existe une matrice U, une matrice diagonale D et un vecteur w tels que*

$$AW = WU + we_m^*$$
$$W^*AW = D$$

alors pour tout $r \in W^\perp$,

$$(W^*AW)^{-1}W^*Ar = \mu e_m$$

et

$$Hr = r - \mu w_m$$

avec $\mu = \frac{r^*w}{w_m^*Aw_m}$.

Preuve On pose $\hat{\mu} = (W^*AW)^{-1}W^*Ar$. On a alors

$$\begin{array}{rrcl} & W^*AW\hat{\mu} & = & W^*Ar \\ \Rightarrow & D\hat{\mu} & = & U^*W^*r + (w^*r)e_m \\ \Rightarrow & \hat{\mu} & = & D^{-1}(w^*r)e_m \quad \text{car } W^*r = 0 \\ & = & \frac{(w^*r)}{D_{m,m}}e_m & \\ & = & \mu e_m. & \end{array}$$

On a aussi $Hr = r - W\hat{\mu} = r - \mu w_m$. □

Si les conditions du théorème 4.9 sont remplies, on peut remplacer, dans l'algorithme 4.3, les étapes :

1: Résoudre $W^*AW\hat{\mu}_j = W^*Ar_j$

2: $p_j = \beta_{j-1}p_{j-1} + r_j - W\hat{\mu}_j$

par

1: $\mu_{k+1} = \frac{r_{k+1}^* w}{w_m^* A w_m}$

2: $p_{k+1} = r_{k+1} + \beta_{k+1}p_k - \mu_{k+1}w_m$.

Dans le cas où W est l'espace de Krylov calculé par l'algorithme du gradient conjugué, on a alors $W = [w_1, \ldots, w_m]$ avec $w_{i+1} = p_i$ pour $i = 0, \ldots, m-1$ si on décide de garder m directions du premier espace de Krylov. Cet espace vérifie bien les hypothèses du théorème 4.9 page précédente. En effet

$$\begin{aligned} Aw_i &= \frac{1}{\alpha_{i-1}}(r_{i-1} - r_i) \\ &= \frac{1}{\alpha_{i-1}}(w_i - \beta_{i-1}w_{i-1} - w_{i+1} + \beta_i w_i) \\ &= -\frac{1}{\alpha_{i-1}}w_{i+1} + \frac{1+\beta_i}{\alpha_{i-1}}w_i - \frac{\beta_{i-1}}{\alpha_{i-1}}w_{i-1}. \end{aligned}$$

C'est-à-dire

$$AW = WT + \frac{1}{\alpha_{m-1}}w_{m+1}e_m^*,$$

avec

$$T = \begin{bmatrix} \frac{1+\beta_1}{\alpha_0} & -\frac{\beta_1}{\alpha_1} & & & \\ -\frac{1}{\alpha_0} & \frac{1+\beta_2}{\alpha_1} & -\frac{\beta_2}{\alpha_2} & & \\ & -\frac{1}{\alpha_1} & \ddots & \ddots & \\ & & \ddots & \ddots & -\frac{\beta_{m-1}}{\alpha_{m-1}} \\ & & & -\frac{1}{\alpha_{m-2}} & -\frac{1+\beta_m}{\alpha_{m-1}} \end{bmatrix}.$$

Cela veut dire que l'on peut calculer p_k grâce à une récurrence à quatre termes :

$$p_{k+1} = r_{k+1} + \beta_{k+1}p_k - \mu_{k+1}w_m.$$

Or d'après la proposition 4.2 page 74, on a $\forall k, W^*Ap_k = 0$, c'est-à-dire en particulier $w_m^*Ap_k = 0$. D'où

$$\begin{array}{rrcl} & w_m^*Ap_{k+1} & = & w_m^*Ar_{k+1} + \beta_{k+1}w_m^*Ap_k - \mu_{k+1}w_m^*Aw_m \\ \Rightarrow & 0 & = & w_m^*Ar_{k+1} + 0 - \mu_{k+1}w_m^*Aw_m \\ \Rightarrow & \mu_{k+1} & = & \frac{w_m^*Ar_{k+1}}{w_m^*Aw_m}. \end{array}$$

Cela permet de déduire l'algorithme 4.6, du gradient conjugué avec espace augmenté (AugCG).

Require: x_{-1} et W espace de Krylov
1: $r_{-1} = b - Ax_{-1}$
2: $x_0 = x_{-1} + WD^{-1}W^*r_{-1}$
3: $r_0 = b - Ax_0$
4: $p_0 = (I - WD^{-1}(AW)^*)r_0$
5: **for** $k = 0,1,\ldots$ **do**
6: $\quad \alpha_k = \frac{r_k^*r_k}{p_k^*Ap_k}$
7: $\quad x_{k+1} = x_k + \alpha_k p_k$
8: $\quad r_{k+1} = r_k - \alpha_k Ap_k$
9: $\quad \beta_{k+1} = \frac{r_{k+1}^*r_{k+1}}{r_k^*r_k}$
10: $\quad \mu_{k+1} = \frac{r_{k+1}^*Aw_m}{w_m^*Aw_m}$
11: $\quad p_{k+1} = r_{k+1} + \beta_{k+1}p_k - \mu_{k+1}w_m$
12: **end for**

Algorithme 4.6: Algorithme du Gradient Conjugué avec Espace Augmenté

4.3.3.2 Considérations pratiques

Tout d'abord d'un point de vue opérations élémentaires, le surcoût le plus important est à l'initialisation. En effet, le calcul des x_0 et p_0 convenables est le plus gourmand en temps de calcul ; ces opérations sont de type BLAS2 (projection sur un sous-espace de dimension m). L'autre surcoût est pendant les itérations, et correspond à l'ajout d'un produit scalaire (`ddot`) et d'une mise à jour de vecteur (`daxpy`), qui sont des calculs très peu chers. AugCG n'ajoutant pas de nouveaux produits matrice-vecteur, le surcoût global devrait être aisément contrebalancé par la réduction du nombre d'itérations. De plus toutes les opérations étant de type BLAS1 ou BLAS2, elles peuvent être vectorisées ou parallélisées.

Pour ce qui est de la mémoire utilisée, là encore, le surcoût le plus important est à l'initialisation. Il faut en effet stocker les m vecteurs de W, alors que, pendant les itérations, on n'a besoin que de deux vecteurs supplémentaires w_m et Aw_m. Si la mémoire centrale n'est pas suffisante, on peut envisager de garder W dans la mémoire

de masse. Si l'on n'y a pas recours, la question du choix de m se pose ; cela sera examiné dans la partie 4.4, lors des essais numériques.

Il faut aussi analyser les effets des imprécisions de calcul. Souvent dans les procédures de type Lanczos, les erreurs d'arrondis mènent à des pertes d'orthogonalité qui ralentissent (voire empêchent) la convergence. Cette situation est analysée pour la méthode de Lanczos dans [34] et [43].

Dans l'algorithme du gradient conjugué avec espace augmenté, on impose une condition initiale d'orthogonalité (4.4) et deux conditions d'orthogonalité lors des itérations. Pour la relation (4.4), on suit le schéma proposé dans [43] pour le résidu initial r_0, ce qui correspond à un procédé de Gram-Schmidt modifié, puis on étend ce schéma à la direction de descente initiale p_0. Cela garantit une plus grande stabilité numérique pour r_0 et p_0. Ce qui veut dire que, au lieu d'utiliser la formulation suivante

1: $x_0 = x_{-1} + WD^{-1}W^* r_{-1}$
2: $r_0 = b - Ax_0$,

on utilise plutôt, celle-ci, mathématiquement équivalente, mais plus stable numériquement :

1: $x_0 = x_{-1}$
2: $r_0 = b - Ax_0$
3: **for** $j = 1, \ldots, m$ **do**
4: $\quad x_0 = x_0 + \frac{r_0^* w_j}{w_j^* A w_j} w_j$
5: $\quad r_0 = r_0 - \frac{r_0^* w_j}{w_j^* A w_j} A w_j$
6: **end for**.

On étend alors ce schéma aux itérations : en effet, la nouvelle direction de descente p_{k+1} est A-orthogonale à w_m, dernier vecteur de l'espace W correspondant à la précédente résolution du système, et A-orthogonale à p_k, direction de descente de l'itération précédente. Là encore on applique le schéma de Gram-Schmidt modifié : tout-d'abord on A-orthogonalise par rapport à w_m puis par rapport à p_k. Tout cela est résumé dans l'algorithme 4.7 page suivante.

La perte d'orthogonalité peut apparaître pendant les itérations, comme cela est observé dans [38]. Une réorthogonalisation complète, comme dans [38] et [4], peut alors remédier à ce problème dans la plupart des cas.

On peut aussi dériver une version préconditionnée de AugCG, car, en pratique, les algorithmes sont utilisés en version préconditionnée. On suppose que le premier système a été résolu avec le gradient conjugué préconditionné par la matrice symétrique définie positive M^{-1}. Donc

$$\langle W \rangle = \langle M^{-1} S \rangle = M^{-1} \mathcal{K}_m(AM^{-1}, s_0).$$

La méthode AugCG est définie par le sous-espace des solutions

$$\mathcal{K}_{m,k}(A, s_0, r_0) = \langle M^{-1}(S + R_k) \rangle = \langle W \rangle + \langle P_k \rangle$$

et par la condition de Galerkin

$$r_{k+1} \perp \mathcal{K}_{m,k}(A, s_0, r_0).$$

Les résultats prouvés ci-dessus tiennent toujours et donc :

$$\langle R_k \rangle = \mathcal{K}_k(AHM^{-1}, r_0).$$

On peut alors voir le gradient conjugué préconditionné avec espace augmenté comme un gradient conjugué deux fois préconditionné, par HH^* et M^{-1}. Ceci donne l'implantation 4.7.

Require: w_j et Aw_j pour $j = 1, \ldots, m$ et x_{-1}, solution initiale
1: $x_0 = x_{-1}$
2: $r_0 = b - Ax_0$
3: **for** $j = 1 \ldots m$ **do**
4: $\quad x_0 = x_0 + \frac{r_0^* w_j}{w_j^* A w_j} w_j$
5: $\quad r_0 = r_0 - \frac{r_0^* w_j}{w_j^* A w_j} A w_j$
6: **end for**
7: $z_0 = M^{-1} r_0$
8: **for** $j = 1 \ldots m$ **do**
9: $\quad z_0 = z_0 - \frac{z_0^* A w_j}{w_j^* A w_j} w_j$
10: **end for**
11: $p_0 = z_0$
12: **for** $k = 0,1, \ldots$ **do**
13: $\quad \alpha_k = \frac{r_k^* z_k}{p_k^* A p_k}$
14: $\quad x_{k+1} = x_k + \alpha_k p_k$
15: $\quad r_{k+1} = r_k - \alpha_k A p_k$
16: $\quad z_{k+1} = M^{-1} r_{k+1}$
17: $\quad \mu_{k+1} = \frac{z_{k+1}^* A w_m}{w_m^* A w_m}$
18: $\quad z_{k+1} = z_{k+1} - \mu_{k+1} w_m$
19: $\quad \beta_{k+1} = \frac{r_{k+1}^* z_{k+1}}{r_k^* z_k}$
20: $\quad p_{k+1} = z_{k+1} + \beta_{k+1} p_k$
21: **end for**

Algorithme 4.7: Gradient Conjugué avec Espace Augmenté

4.3.4 L'algorithme du gradient conjugué déflaté

4.3.4.1 Introduction

On va maintenant voir comment utiliser l'algorithme du gradient conjugué projeté pour résoudre plusieurs systèmes linéaires symétriques de la forme

$$Ax^{(s)} = b^{(s)}, \quad s = 1,2, \ldots, \nu, \tag{4.29}$$

où $A \in \mathbb{R}^{n\times n}$ est symétrique définie positive et où les différents seconds membres dépendent des solutions des systèmes précédents. Comme dans le cas du gradient conjugué avec espace augmenté, on réutilise l'espace de Krylov précédemment calculé. Mais alors que pour AugCG on l'utilisait "brut", l'idée est maintenant d'en extraire des approximations des vecteurs propres. Cette idée est déjà présente dans l'algorithme de GMRES déflaté, comme on peut le voir par exemple dans [8, 26, 31]. On va donc injecter quelques vecteurs approchés dans l'espace des solutions, généralement les vecteurs correspondants aux plus petites valeurs propres (c'est-à-dire les plus proches de zéro), lorsque l'on résout les systèmes de 4.29, sauf le premier. À partir du 3^{e} système, on raffine les vecteurs propres déjà calculés grâce à l'espace de Krylov engendré lors de la résolution précédente. On est alors en droit d'espérer que la convergence sera améliorée de résolution en résolution.

En effet, si on appelle λ_i les valeurs propres de A avec

$$\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n.$$

Alors, si les colonnes $w_1, \ldots, w_k$ de W, sont les vecteurs propres exacts de A, associés aux plus petites valeurs propres $\lambda_1, \ldots, \lambda_k$, on a immédiatement

$$\kappa(H^*AH) = \lambda_n/\lambda_{k+1}.$$

Dans ce cas, le gain sur le conditionnement du système à résoudre, est connu explicitement. Si, maintenant, les colonnes de W ne sont pas exactement les vecteurs propres, mais seulement des approximations des vecteurs propres associés à $\lambda_1, \ldots, \lambda_k$, on peut s'attendre quand même à ce que

$$\kappa(H^*AH) \approx \lambda_n/\lambda_{k+1}.$$

4.3.4.2 Calcul des approximations de vecteurs propres

Il y a plusieurs manières de calculer des vecteurs propres approchés puis de les raffiner au cours des différentes résolutions par gradient conjugué, appliquées au système 4.29.

Soit

$$W^{(s)} = [w_1^{(s)}, w_2^{(s)}, \ldots, w_k^{(s)}]$$

les vecteurs propres que l'on veut utiliser pour résoudre le s^{e} système. Au départ $W^{(1)} = \emptyset$. Par la suite, les vecteurs et les scalaires générés par l'algorithme du gradient conjugué déflaté (DefCG) appliqué au s^{e} système de 4.29 sont notés avec l'exposant (s). Idéalement, $W^{(s)}$ correspond aux vecteurs propres associés aux k valeurs propres $\lambda_1, \lambda_2, \ldots, \lambda_k$ de A. On utilise alors ProjCG pour résoudre chacun des systèmes de 4.29. Le premier est avec un $W^{(s)}$ vide ce qui correspond à une résolution par le gradient conjugué standard.

Après la résolution du s^{e} système de 4.29, on met à jour l'ensemble $W^{(s)}$ des approximations des vecteurs propres pour la résolution suivante, ce qui donne le nouvel espace $W^{(s+1)}$. Dans [8], trois techniques de projections sont décrites pour obtenir de

telles approximations. Ici on ne retiendra que l'une d'entre elles, suggérée par Morgan [31] et appelée *projection harmonique*. Cette approche donne les meilleurs résultats pour trouver les valeurs propres proches de zéro.

Soient l vecteurs linéairement indépendants $z_1, z_2, \ldots, z_l$, cette méthode calcule les k vecteurs propres désirés en résolvant le problème aux valeurs propres généralisées

$$Gy - \theta F y = 0,$$

avec $Z = [z_1, z_2, \ldots, z_l]$, $G = (AZ)^* AZ$ et $F = Z^* AZ$. Pour chaque nouveau système, la matrice Z est définie ainsi

$$Z^{(s)} = [W^{(s)}, P_l^{(s)}], \tag{4.30}$$

avec [1]

$$P_l^{(s)} = \left[p_0^{(s)}, p_1^{(s)}, \ldots, p_{l-1}^{(s)} \right].$$

On résout alors le problème généralisé

$$G^{(s)} y_i - \theta_i F^{(s)} y_i = 0, \quad i = 1, \ldots, k \tag{4.31}$$

où

$$F^{(s)} = \left(Z^{(s)} \right)^* AZ^{(s)}, \quad G^{(s)} = \left(AZ^{(s)} \right)^* AZ^{(s)}. \tag{4.32}$$

et où $\theta_1, \ldots, \theta_k$ sont les k plus petites valeurs de Ritz. Puis on définit les nouvelles approximations des vecteurs propres pour le système suivant par

$$W^{(s+1)} = Z^{(s)} Y^{(s)}, \tag{4.33}$$

avec

$$Y^{(s)} = [y_1^{(s)}, \ldots, y_k^{(s)}].$$

Cela décrit la méthode mathématiquement. Du point de vue de l'implémentation, il est possible d'éviter la multiplication matrice-matrice $AZ^{(s)}$ pour $G^{(s)}$ et $F^{(s)}$ dans (4.32).

Lemme 4.3 *Soit*

$$R_l^{(s)} = \left[r_0^{(s)}, r_1^{(s)}, \ldots, r_{l-1}^{(s)} \right], \qquad \tilde{\Delta}_{l+1}^{(s)} = \left[\hat{\mu}_0^{(s)}, \hat{\mu}_1^{(s)}, \ldots, \hat{\mu}_l^{(s)} \right],$$

et

$$\tilde{L}_l^{(s)} = \begin{bmatrix} \frac{1}{\alpha_0^{(s)}} & & & \\ -\frac{1}{\alpha_0^{(s)}} & \frac{1}{\alpha_1^{(s)}} & & \\ & -\frac{1}{\alpha_1^{(s)}} & \ddots & \\ & & \ddots & \frac{1}{\alpha_{l-1}^{(s)}} \\ & & & -\frac{1}{\alpha_{l-1}^{(s)}} \end{bmatrix}, \quad \tilde{U}_l^{(s)} = \begin{bmatrix} 1 & -\beta_0^{(s)} & & & \\ & 1 & -\beta_1^{(s)} & & \\ & & \ddots & \ddots & \\ & & & 1 & -\beta_{l-1}^{(s)} \\ & & & & 1 \end{bmatrix}.$$

1. On pourrait garder les résidus $r_i^{(s)}$ plutôt que les directions de descentes $p_i^{(s)}$, mais les formules (4.35 - 4.37) deviennent plus complexes.

Alors la matrice $AP_l^{(s)}$ peut être calculée ainsi :

$$AP_l^{(s)} = R_{l+1}^{(s)} \tilde{L}_l^{(s)} = \left(W^{(s)} \tilde{\Delta}_{l+1}^{(s)} + P_{l+1}^{(s)} \tilde{U}_l^{(s)} \right) \tilde{L}_l^{(s)} \tag{4.34}$$

Preuve Ce résultat découle directement des deux relations suivantes, de l'algorithme 4.3 :

$$\begin{aligned} r_j^{(s)} &= p_j^{(s)} - \beta_{j-1}^{(s)} p_{j-1}^{(s)} + W^{(s)} \hat{\mu}_j^{(s)} \\ Ap_j^{(s)} &= \frac{1}{\alpha_j^{(s)}} r_j^{(s)} - \frac{1}{\alpha_j^{(s)}} r_{j+1}^{(s)} \end{aligned}$$

□

Puis, cette relation établie, on peut alors résoudre le problème harmonique (4.31) sans autre produit supplémentaire avec la matrice A.

Théorème 4.10 *Soit*

$$\tilde{D}_l^{(s)} = \begin{bmatrix} d_0^{(s)} & & & \\ & d_1^{(s)} & & \\ & & \ddots & \\ & & & d_{l-1}^{(s)} \end{bmatrix} \text{ avec } d_j^{(s)} = \left(p_j^{(s)} \right)^* A p_j^{(s)},$$

et

$$\tilde{G}^{(s)} = \begin{bmatrix} \frac{d_0^{(s)}}{\alpha_0^{(s)}} \left(1 + \beta_0^{(s)} \right) & -\frac{d_1^{(s)}}{\alpha_0^{(s)}} & & & \\ -\frac{d_1^{(s)}}{\alpha_0^{(s)}} & \frac{d_1^{(s)}}{\alpha_1^{(s)}} \left(1 + \beta_1^{(s)} \right) & -\frac{d_2^{(s)}}{\alpha_1^{(s)}} & & \\ & -\frac{d_2^{(s)}}{\alpha_1^{(s)}} & \ddots & \ddots & \\ & & \ddots & \ddots & -\frac{d_{l-1}^{(s)}}{\alpha_{l-2}^{(s)}} \\ & & & -\frac{d_{l-1}^{(s)}}{\alpha_{l-2}^{(s)}} & \frac{d_{l-1}^{(s)}}{\alpha_{l-1}^{(s)}} \left(1 + \beta_{l-1}^{(s)} \right) \end{bmatrix}.$$

Alors les matrices $G^{(s)}$, $F^{(s)}$ et $AW^{(s)}$ sont données par

$$G^{(s)} = \begin{bmatrix} \left(AW^{(s)} \right)^* AW^{(s)} & \left(W^{(s)} \right)^* AW^{(s)} \tilde{\Delta}_{l+1}^{(s)} \tilde{L}_l^{(s)} \\ \left(\tilde{\Delta}_{l+1}^{(s)} \tilde{L}_l^{(s)} \right)^* \left(W^{(s)} \right)^* AW^{(s)} & \tilde{G}^{(s)} \end{bmatrix} \tag{4.35}$$

$$F^{(s)} = \begin{pmatrix} \left(W^{(s)} \right)^* AW^{(s)} & 0 \\ 0 & \tilde{D}_l^{(s)} \end{pmatrix} \tag{4.36}$$

$$AW^{(s+1)} = [AW^{(s)}, AP_l^{(s)}] Y^{(s)} \tag{4.37}$$

où $AP_l^{(s)}$ est donné par (4.34).

Preuve Les résultats désirés sont obtenus grâce à (4.33), (4.32), (4.34) et à l'A-othorgonalité de P et W. □

Pour cette approche, il faut garder les $d_i^{(s)}$, les $\alpha_i^{(s)}$, les $\beta_i^{(s)}$, les $\hat{\mu}_i^{(s)}$ et les $p_i^{(s)}$ des l premières étapes de l'algorithme ainsi que les matrices $W^{(s)}$, $AW^{(s)}$ et $\left(W^{(s)}\right)^* AW^{(s)}$.

4.3.4.3 Le gradient conjugué déflaté pour les systèmes avec multiples seconds membres

En résumé, l'algorithme DefCG pour des équations successives, fonctionne de la mamière suivante. On suppose que l'on veut déflater avec k vecteurs propres. Dans un premier temps, on exécute le gradient conjugué standard avec un nombre l d'itérations supérieur à k. Les données $d_i^{(1)}$, $\alpha_i^{(1)}$, $\beta_i^{(1)}$ pour $i = 0,1,\ldots,l-1$, et $P_l^{(1)} = \left[p_0^{(1)},\ldots,p_{l-1}^{(1)}\right]$ sont sauvegardées. Puis $G^{(1)}$ et $F^{(1)}$ sont calculées d'après (4.35) et (4.36) avec $s = 1$. Le problème aux valeurs propres (4.31) est alors résolu pour k vecteurs propres qui constituent les colonnes de $Y^{(1)}$; puis $W^{(2)}$ est calculé par $W^{(2)} = \left[P_l^{(1)}\right] Y^{(1)}$ car $W^{(1)} = \emptyset$. Dans les étapes suivantes, l'algorithme déflaté est utilisé au lieu du Gradient Conjugué, avec pour espace de déflation $W = W^{(s)}$. Les matrices $AW^{(s)}$ et $(W^{(s)})^* AW^{(s)}$ sont calculées grâce à (4.37). On procède alors comme avant, sauf que $W^{(s+1)}$ est défini par $W^{(s+1)} = [W^{(s)}, P_l^{(s)}] Y^{(s)}$. Les matrices $G^{(s)}$ et $F^{(s)}$ servant au calcul des vecteurs propres $y_i^{(s)}$ sont obtenues grâce aux formules (4.35 - 4.37).

La version préconditionnée de cette méthode est obtenue en considérant le système préconditionné suivant

$$L^{-1}AL^{-*}y^{(s)} = L^{-1}b^{(s)}, \quad x^{(s)} = L^{-*}y^{(s)}, \quad s = 1,2,\ldots,\nu.$$

Comme dans le cas du gradient conjugué avec projection, préconditionné, on applique la méthode aux systèmes $L^{-1}AL^{-*}y^{(s)} = L^{-1}b^{(s)}$ et on redéfinit alors les variables d'origine comme en (4.24). Tout est alors comme en (4.31) sauf $G^{(s)}$ qui devient maintenant

$$G^{(s)} = \begin{bmatrix} \left(AW^{(s)}\right)^* M^{-1} AW^{(s)} & \left(W^{(s)}\right)^* AW^{(s)} \tilde{\Delta}_{l+1}^{(s)} \tilde{L}_l^{(s)} \\ \left(\tilde{\Delta}_{l+1}^{(s)} \tilde{L}_l^{(s)}\right)^* \left(W^{(s)}\right)^* AW^{(s)} & \tilde{G}^{(s)} \end{bmatrix}, \tag{4.38}$$

avec $M = LL^*$. Enfin le calcul de $AP_l^{(s)}$ dans (4.34) devient

$$M^{-1}AP_l^{(s)} = M^{-1}R_{l+1}^{(s)} \tilde{L}_l^{(s)} = \left(W^{(s)} \tilde{\Delta}_{l+1}^{(s)} + P_{l+1}^{(s)} \tilde{U}_l^{(s)}\right) \tilde{L}_l^{(s)}. \tag{4.39}$$

Si on rassemble toutes ces relations, on obtient l'algorithme 4.8 page ci-contre.

En plus de la mémoire requise par l'algorithme du DefCG préconditionné, $l+1$ vecteurs $\hat{\mu}_0^{(s)}, \hat{\mu}_1^{(s)}, \ldots, \hat{\mu}_l^{(s)}$ doivent être stockés pendant les itérations, ainsi que trois matrices $\tilde{L}_l^{(s)}$, $U_l^{(s)}$ et $\tilde{D}_l^{(s)}$.

Require: l et k avec $l \geq k$
1: Résoudre le premier système de 4.29 avec le gradient conjugué préconditionné
2: Calculer $G^{(1)}$ et $F^{(1)}$ d'après 4.38 et 4.36.
3: $W^{(1)} = \emptyset$
4: **for** $s = 2,3,\ldots,\nu$ **do**
5: Résoudre 4.31 pour k vecteurs propres $y_1^{(s-1)}, y_2^{(s-1)}, \ldots, y_k^{(s-1)}$
6: $W^{(s)} = \left[W^{(s-1)}, P_l^{(s-1)}\right] Y^{(s-1)}$
7: Résoudre le s^{e} système de 4.29 avec DefCD préconditionné avec $W = W^{(s)}$
8: Calculer $G^{(s)}$ et $F^{(s)}$ d'après 4.38, 4.35, 4.36 et 4.39
9: **end for**

Algorithme 4.8: Algorithme du Gradient Conjugué Déflaté Préconditionné

4.3.4.4 Considérations pratiques

Pour l'examen des coûts de calculs et de mémoire de l'algorithme DefCG, on suppose que $k \ll n$ et donc on négligera tous les termes ne contenant pas n.

Pour DefCG, il faut conserver W et AW en plus des vecteurs usuels du gradient conjugué. Cela veut dire un stockage supplémentaire de $2k$ vecteurs de longueur n. De plus, DefCG demande le calcul de Hr_j à chaque étape. Cela peut être effectué grâce à des opérations de type BLAS2 pour $z_j = (AW)^* r_j$ et $r_j - W(W^*AW)^{-1} z_j$. Le coût de ces opérations est $O(kn)$: le surcoût en temps CPU n'est donc pas trop élevé. Il reste alors le coût du calcul de W

On a vu que pour AugCG, qui utilise, pour W, l'espace engendré par les directions successives de descente, seule la dernière colonne w_m de W avait besoin d'être gardée car la projection H se résumait à l'orthogonalisation contre ce seul vecteur w_m. Malheureusement, si s est plus grand que 2, il n'y a pas de solution simple pour améliorer W. La seule solution, en effet, est de conserver tous les vecteurs des directions de descente $W^{(s)}$, ce qui devient très vite prohibitif.

Pour économiser quelques calculs, il est intéressant que les colonnes de W soient A-orthogonales, car, dans ce cas, la matrice W^*AW devient diagonale. Ce n'est cependant pas essentiel et le surcoût est peu important.

Les approximations de vecteurs propres sont calculées grâce au problème propre généralisé (4.31). Et donc, il faut , en plus de W et AW stocker les l vecteurs $P_l^{(s)}$, ce qui fait en tout un surcoût en mémoire de $2k + l$ vecteurs de taille n.

Le calcul de W grâce à (4.33) et de AW grâce à (4.37) requiert essentiellemnent des opérations de type BLAS3 dont la complexité est en $O(n(l+1)k)$. Le calcul de W^*AW est une opération BLAS3 de coût $O(nk^2)$. Et donc, si k et l restent petits, le surcoût total reste modeste.

4.4 Applications numériques

4.4.1 Comparaisons InitCg et AugCG

4.4.1.1 Conditions des tests

Voici quelques résultats numériques qui montrent l'efficacité des méthodes AugCG et InitCG. Les exemples sont tirés de problèmes tests, et les algorithmes sont implémentés en Matlab. On ne donne ici que des résultats de convergence car les temps de calculs ne seraient pas significatifs.

Un premier système $Ay = c$ est d'abord résolu, puis un second $Ax = b$. On donne la convergence du second système pour différentes matrices A et différents choix de seconds membres b. Dans tous les exemples, on prend $c = Ae$ avec $e = [1,1,\ldots,1]$ et $y_0 = 0$ (donc $s_0 = c$). De plus, on prend $x_{-1} = y_j$ où j est le nombre total d'itérations faites pour le premier système $Ay = c$. On a donc

$$r_0 = H^*(b - c) + s_j.$$

Pour le second membre b, on étudie deux cas de figures différents. Tout d'abord, un $b^{(1)}$ "proche" de c:

$$b^{(1)} = \nu c + \epsilon e + AWe$$

ainsi $H^*b^{(1)} = \nu H^*c + \epsilon H^*e = \nu s_m + \epsilon H^*e$ et

$$r_0^{(1)} = (\nu - 1)s_m + s_j + \epsilon H^*e.$$

Dans le second cas, $b^{(2)}$ est "loin" de c:

$$b^{(2)} = e$$

et donc

$$r_0^{(2)} = H^*(e - Ae) + s_j.$$

Le critère d'arrêt pour le second système est $\|r_k\| \leq tol\|b\|$ où tol est spécifié à chaque exemple.

Toutes les figures, tracent la norme relative du résidu $\|r_k\|/\|b\|$, en fonction du nombre d'itérations (qui est égal au nombre de produits matrice-vecteur). Pour les exemples de 2 à 6, la ligne en pointillés représente le gradient conjugué démarré avec x_{-1}, la ligne avec des tirets représente InitCG et la ligne pleine AugCG, tous deux démarrés aussi avec x_{-1}.

4.4.1.2 Exemple 1

On choisit d'abord la matrice du Laplacien avec des conditions aux limites de Dirichlet, discrétisé par un schéma aux différences finies à 5 points, sur une grille carrée de taille 32×32. La matrice est donc de taille 900.

On prend ici $b = b^{(1)}$ pour montrer la stabilité numérique de InitCG. tol vaut 10^{-12} et on prend $\epsilon = 10^{-2}$ et $\nu = 10$. On compare les deux formulations, stable et instable,

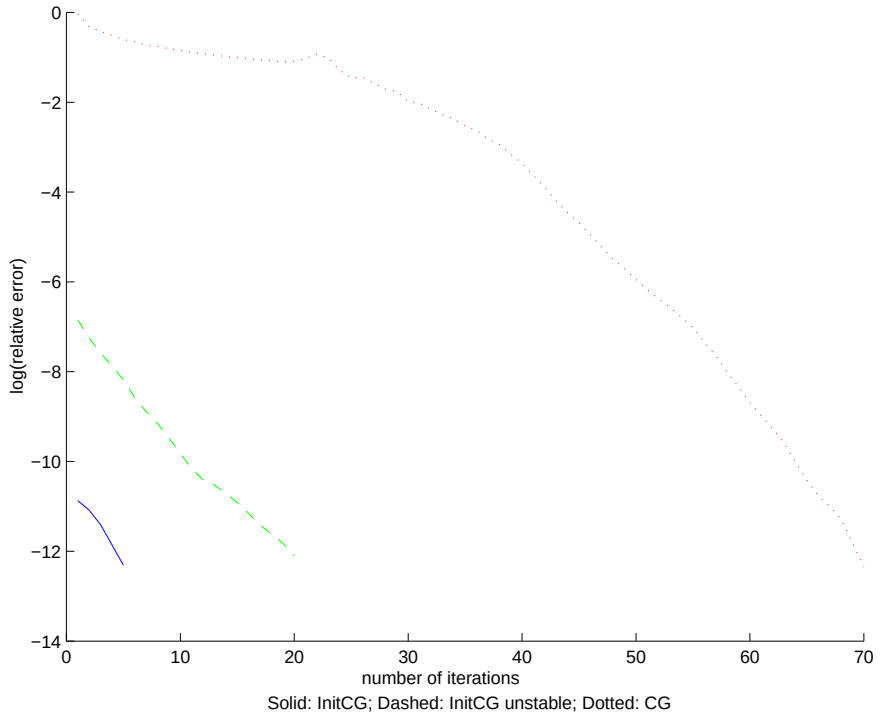


FIG. 4.1 – *Résultats pour le Laplacien d'ordre 900 avec le second membre* $b^{(1)}$

de la page 85. La figure 4.1 montre la courbe de convergence pour la résolution de $Ax = b$ pour le gradient conjugué (pointillés), pour InitCg avec une orthogonalisation non modifiée (tirets) et pour InitCG avec une orthogonalisation modifiée pour une plus grande stabilité numérique (courbe pleine).

Ici $m = 65$ et $j = 68$, qui correspond à $\|s_j\| \leq 10^{-12}\|c\|$. Clairement, dans ce cas, la formulation stable est, comme attendue, meilleure et, à partir de maintenant, on n'utilisera plus que celle-là. On remarque aussi que InitCG converge plus vite que le gradient conjugué.

4.4.1.3 Exemple 2

Comme on ne s'intéresse qu'à la convergence, on peut utiliser des matrices diagonales pour lesquelles on choisit facilement les valeurs propres. C'est déjà fait ainsi dans [43, 13]. On prend $A = diag([1 : 500])$.

Pour les deux seconds membres, $m = 30$ et $tol = 10^{-9}$. On choisit $b = b^{(1)}$ avec $\nu = 1$ et $\epsilon = 10^{-2}$. La figure 4.2 montre le résultat pour $b^{(1)}$ et $b^{(2)}$.

InitCG et AugCG sont semblables pour des seconds membres proches, mais pour des seconds membres plus quelconques, InitCg n'est plus aussi efficace que AugCG, car $\|H^*(b-c)\|$ est trop grand. InitCG et AugCG sont cependant comparables pendant les premières itérations, comme prédit par la théorie, mais après la vitesse de convergence asymptotique de AugCG est meilleure. On note encore une fois que les deux algorithmes

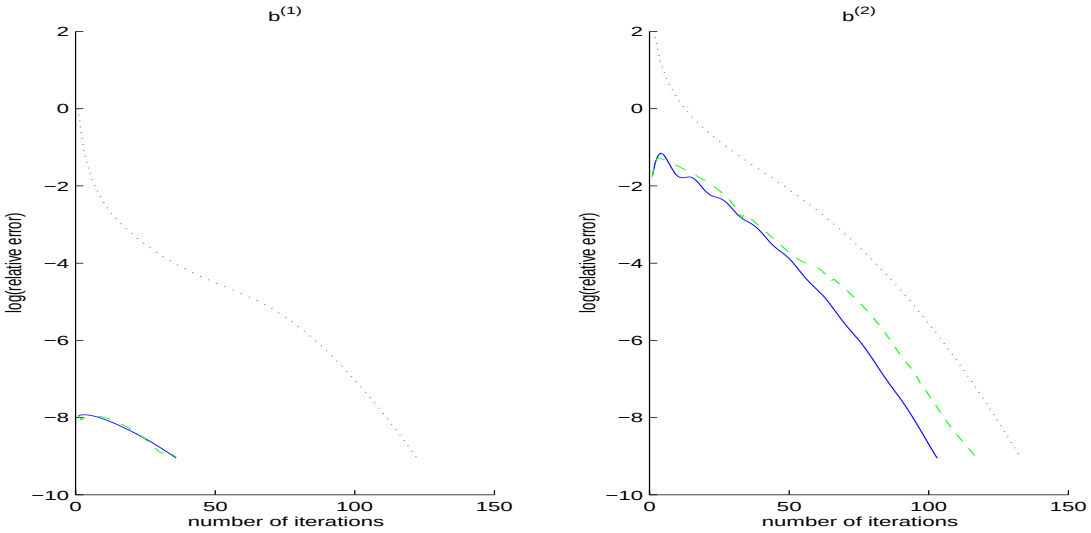


FIG. 4.2 – *Résultats pour la matrice diag*([1 : 500])

convergent plus vite que CG.

4.4.1.4 Exemple 3

On étudie maintenant l'impact de la taille m de l'espace de Krylov, sur l'efficacité de AugCG. La matrice A est la matrice S1RMQ4M1, du groupe Cylshell [2]. La matrice est de taille $N = 5489$ et a 133950 coefficients non nuls. Son conditionnement estimé est 1.8110^6. On préconditionne le système par une factorisation de Cholesky incomplète IC(1).

On prend $b = b^{(2)}$ et $tol = 10^{-9}$. La figure 4.3 montre les courbes de convergence de CG et AugCG pour m variant de 10 à 50. Jusqu'à $m = 40$, AugCG économise à peu près m itérations puis un peu moins.

4.4.1.5 Exemple 4

Dans cet exemple et le suivant, on étudie les effets du préconditionnement et du second membre. On utilise la matrice S2RMQ4M1 de la même collection. La taille de la matrice est $N = 5489$ et 134420 coefficients sont non nuls. Le conditionnement est estimé à 1.7710^8.

On prend $b = b^{(1)}$ avec $\nu = 1$ et $\epsilon = 10^{-3}$. On garde $m = 20$ vecteurs en mémoire et la tolérance est fixée à $tol = 10^{-6}$. La figure 4.4 montre les résultats pour différents préconditionnement de type Cholesky incomplets, avec 4 niveaux de remplissages: IC(1), IC(2), IC(3) et IC(4).

Ici InitCG est plutôt efficace car les deux seconds membres sont proches. Le résidu relatif $\|r_0\|/\|b\|$ est bien plus faible avec InitCG et AugCG qu'avec CG, presque égal à

2. http://math.nist.gov/MatrixMarket/data

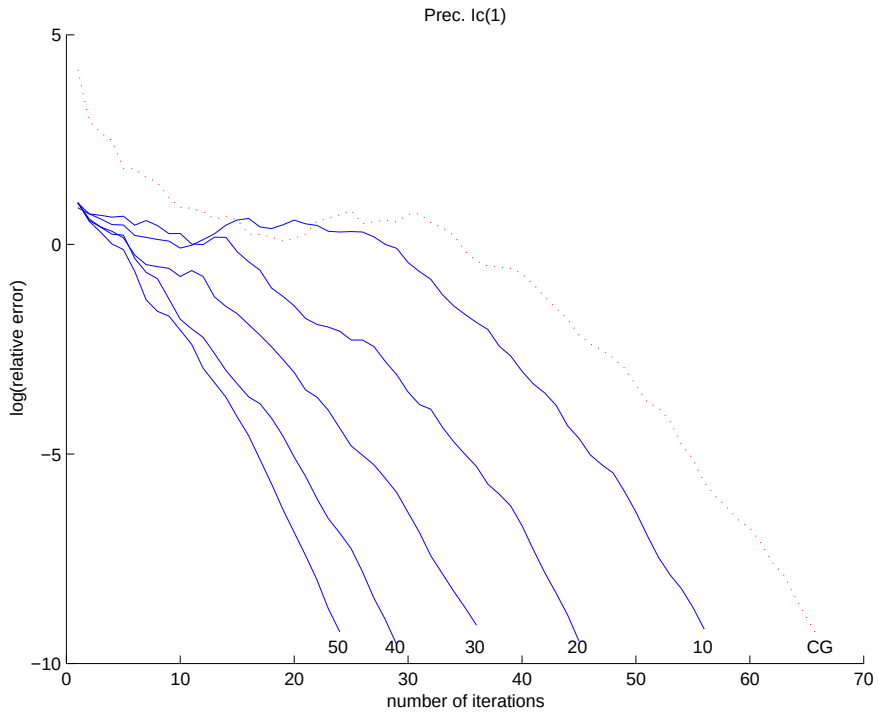


FIG. 4.3 – *Résultats pour la matrice S1RMQ4M1 avec le second membre* $b^{(2)}$

la tolérance 10^{-6}.

AugCG économise 36 itérations avec le préconditionnement IC(1) et à peu près 20 avec les trois autres préconditionnements. Pour IC(1), le gain initial est relativement important car CG converge lentement. Malheureusement dans tous les cas, le gain initial est en partie perdu car le résidu augmente. On peut reconnaître des formes similaires dans les courbes de convergences de AugCG et de CG, décalées d'environ m itérations.

4.4.1.6 Exemple 5

On choisit la même matrice S2RMQ4M1, et les paramètres sont $b = b^{(2)}$, $m = 20$ et $tol = 10^{-6}$. la figure 4.5 montre les résultats pour les préconditionnements IC(1), IC(2), IC(3) et IC(4).

InitCG n'est pas réellement efficace car le second membre b est trop éloigné de c. Avec tous les préconditionnements, AugCG économise autour de 15 itérations : AugCG reste efficace même si on améliore le conditionnement.

4.4.1.7 Exemple 6

On étudie ici un exemple où le gradient conjugué converge lentement pour montrer l'impact du préconditionnement et de m. La matrice est S3RMT3M1 toujours de la même collection. La matrice est de taille $N = 5489$ avec 111579 coefficients non-nuls.

Le conditionnement estimé est 2.4810^{10}. On prend $b = b^{(2)}$ et $tol = 10^{-6}$. La figure 4.6 montre les résultats pour $m = 20$ avec IC(1) comme préconditionnement, $m = 20$ avec IC(2) comme préconditionnement, $m = 50$ et IC(1), $m = 100$ et IC(1).

Dans tous les cas InitCG est inefficace, car le second membre est trop éloigné. Avec IC(1) et $m = 20$, AugCG n'est pas très efficace car la convergence du gradient conjugué est très mauvaise lors des premières itérations du premier système. Il y a alors deux moyens pour améliorer la convergence. Tout d'abord, améliorer le préconditionnement : avec IC(2) et $m = 20$, AugCG économise 15 itérations par rapport à CG. Ou bien, on peut augmenter le nombre de vecteurs conservés du premier système, en conservant le préconditionnement IC(1). Et là, alors que $m = 20$ était inefficace, pour $m = 50$ et $m = 100$, la convergence est alors bien améliorée.

Si la convergence du système initial est bonne, alors AugCG est efficace et m peut être petit. Mais si la convergence est mauvaise, alors pour l'améliorer, il faut un m assez grand. Malheureusement, pour des questions de mémoire, m ne peut être pris aussi grand que l'on veut.

En effet, le calcul des vecteurs initiaux x_0, r_0 et p_0 demande le stockage de m vecteurs de dimension N. Si on utilise un espace de stockage secondaire, le gain en itérations devrait contrebalancer les surcoûts d'entrées-sorties. Le coût en calculs augmente aussi avec m pour le calcul de ces vecteurs initiaux.

Le choix d'un m optimal, minimisant le temps de calcul est toujours un problème ouvert. Il vaut sûrement mieux utiliser un bon préconditionnement avec un m modéré, mais il faut aussi que m soit suffisamment grand pour capturer les plus petites valeurs propres de A dans W, car ce sont elles qui ralentissent la convergence[13, 40].

4.4.2 Comparaisons entre AugCG et DefCG

A présent, on va illustrer les vitesses de convergence asymptotiques des algorithmes du gradient conjugué avec espace augmenté et du gradient conjugué déflaté. Les algorithmes sont toujours implémentés en Matlab, et le critère d'arrêt pour les exemples 7 à 9 est sur le résidu relatif:

$$\|b - Ax_j\|_2 / \|b\|_2 \leq 10^{-7},$$

et toutes les figures, sauf 4.10, montrent le résidu relatif en fonction du nombre d'itérations.

4.4.2.1 Exemple 7

Tout d'abord, on teste l'algorithme 4.8 page 91 pour le Laplacien. La matrice est de taille 400, et est calculée comme pour l'exemple 4.4.1.2 page 92. Le second membre b est généré aléatoirement, toutes les composantes étant indépendantes et suivent une distribution normale de moyenne 0 et de variance 1 ($\mathcal{N}(0,1)$).

La plus grande valeur propre de A est 7.9553 et les quatre plus petites sont 0.0447, 0.1112, 0.1112, 0.1777. On calcule les vecteurs propres exacts w_1,w_2,w_3 associés aux trois plus petites valeurs propres 0.0447, 0.1112, 0.1112. On résout alors le système grâce au gradient conjugué standard avec $x_0 = 0$, puis le gradient conjugué déflaté.

On prend $x_{-1} = 0$ et $W = [w_1], [w_1,w_2], [w_1,w_2,w_3]$. les courbes de convergence sont tracées figure 4.7 respectivement pleine, mixte, des croix et des tirets

On peut voir sur la figure 4.7, que la convergence du gradient conjugué déflaté avec juste $W = [w_1]$ est meilleure que celle du gradient conjugué. En effet, le premier résout un système dont le conditionnement est $\kappa = 7.9553/0.1112$. Avec deux vecteurs propres, la convergence n'est pas meilleure, ce qui est logique car le conditionnement est le même $\kappa = 7.9553/0.1112$. Enfin, avec trois vecteurs $W = [w_1,w_2,w_3]$, le conditionnement passe à $\kappa = 7.9553/1.777$, et la convergence est la meilleure.

4.4.2.2 Exemples 8 et 9

Les exemples 8 et 9 montrent l'efficacité de l'algorithme du gradient conjugué déflaté pour résoudre le problème 4.29 page 86. La solution initiale est toujours $x_0 = 0$ pour résoudre avec le gradient conjugué et $x_{-1} = 0$ pour la résolution avec des algorithmes projetés. Le nombre de systèmes pour 4.29 est 10 et les $b^{(s)}$ sont des vecteurs aléatoires calculés comme pour l'exemple 4.4.2.1. On garde les vecteurs de l'espace des solutions pendant $l-20$ itérations et pour la déflation on calcule $k = 5$ vecteurs propres approchés, associés aux plus petites valeurs propres. Ce calcul se fait grâce à la fonction QZ de Matlab, entre deux résolutions consécutives. Les matrices sont tirées de la collection Harwell-Boeing [3].

La courbe en pointillés correspond à la résolution du système avec le gradient conjugué préconditionné. La courbe avec des tirets est pour l'algorithme 4.8 du gradient conjugué déflaté pour les systèmes $s = 2, \ldots, s = 10$. La courbe pleine est pour l'algorithme avec espace augmenté.

Pour l'exemple 8, la matrice est BCSSTK15 du groupe BCSSTRUC2 de la collection Harwell-Boeing. L'ordre de la matrice est 3948 et le système est préconditionné par une factorisation de Cholesky incomplète IC(1). Les résultats sont sur la figure 4.8 page 103.

Pour l'exemple 9, la matrice est 1138BUS du groupe PSADMIT. L'ordre de la matrice est 1138 et le système est préconditionné par une factorisation de Cholesky incomplète IC(0). Les résultats sont aussi sur la figure 4.8.

Pour ces deux exemples, on observe que le nombre d'itérations décroît significativement après les premières résolutions et tend rapidement vers une limite inférieure. Ceci tient à ce que les vecteurs propres calculés sont alors, après quelques raffinements, exacts et le conditionnement du système ne bouge alors plus, et donc à partir de ce moment les convergences sont semblables.

En pratique, on ne raffine plus les vecteurs propres, quand on se rend compte que les vecteurs propres ne varient plus avec le raffinement. On peut aussi augmenter k et l. Pour détecter qu'une paire de Ritz (y_i,μ_i) a convergé, on peut tester la norme de $Ay_i - \mu_i y_i$. Cela est fait dans [7].

L'algorithme ne se comporte cependant pas toujours aussi bien, comme le montre l'exemple suivant.

3. WWW: http://math.nist.gov/MatrixMarket/data/

4.4.2.3 Exemple 10

La matrice de test est S3RMT3M3 du groupe CYLSHELL. Pour cette expérience, on choisit $l = k = 10$ pour l'algorithme 4.8 et $m = 30$ pour l'algorithme 4.7. Le préconditionnement est IC(2) et le test d'arrêt est

$$\|b - Ax_j\|_2 / \|b\|_2 < 10^{-8}.$$

On résout pour 5 seconds membres différents; sinon tout est comme les exemples précédents. Les résultats sont tracés sur la figure 4.9.

Les systèmes 2, 3 et 5 résolus par le gradient conjugué déflaté, et le système 2 résolu par le gradient conjugué avec espace augmenté ne convergent pas. A partir de 10^{-4}, la convergence devient très instable et pour certains systèmes les courbes remontent fortement : l'algorithme diverge.

Théoriquement, les résidus r_j pour les deux algorithmes 4.8 et 4.7, sont orthogonaux à W. Cependant, en pratique, cette orthogonalité est perdue petit à petit au cours des itérations. Cela est illustré par la figure 4.10 qui montre la fonction suivante

$$othor(j) = \min_{i=1\ldots k} \left(\frac{w_i^* r_j}{\|w_i\|_2 \|r_j\|_2} \right).$$

On la trace pour le second système de AugCG et le 5$^{\text{e}}$ de DefCG. Les deux courbes de la fonction $othor(j)$ montrent alors une grosse perte d'orthogonalité, telle qu'elle détruit la convergence.

Un remède pour recupérer l'orthogonalité est d'ajouter une étape de réorthogonalisation

$$r_j = r_j - W\,(W^*W)^{-1}\,W^* r_j \tag{4.40}$$

Tout de suite après avoir calculé r_j dans les algorithmes. Le coût de cette opération est $O(kn)$. On exécute à nouveau les deux algorithmes avec cette nouvelle correction, et on trace les résultats dans la figure 4.9 à droite. Cette fois, seuls les systèmes 3 et5 résolus avec DefCG et le second système résolu avec AugCG ne convergent pas, mais toutes les courbes de convergence décroissent. On trace aussi la fonction $othor(j)$ après correction du résidu r_j avec 4.40 dans la figure 4.10. On remarque que les courbes sont bien plus basses que sans correction.

4.5 Conclusion

Dans ce chapitre sont présentées différentes méthodes pour accélérer la convergence du gradient conjugué préconditionné. Le premier point concerne la modification de la solution initiale. Cela améliore les premières itérations car pendant $\frac{m}{2}$ itérations, on obtient la vitesse de convergence du gradient conjugué par blocs, démarré avec le bloc $[s_0,r_0]$.

Le second point cherche à améliorer le comportement asymptotique. Cette accélération s'obtient par la modification de l'espace des solutions, en lui ajoutant des directions pertinentes, issue de la connaissance que l'on a de la matrice ou des calculs faits lors

de la résolution antérieure d'un système avec la même matrice. On montre alors que la vitesse de convergence est régie par le conditionnement de la matrice «projetée» H^*AH.

Deux algorithmes sont alors proposés. Le premier, AugCG, réutilise l'espace de Krylov W du premier système et a un surcoût pendant les itérations très faible car la projection sur W se réduit à l'orthogonalisation par rapport à un seul vecteur. Cependant, le gain, évident sur les exemples numériques, est peu quantifiable.

Le second algorithme, DefCG, utilise les valeurs propres de A. Le gain est, là, quantifiable car la vitesse de convergence est maintenant gouvernée par le conditionnementde la matrice déflatée. Cependant, le surcoût pendant les itérations est légèrement supérieur, car il faut orthogonaliser par rapport à tous ces vecteurs propres. Un gros avantage de cette méthode est que les approximations des vecteurs propres utilisés pour la déflation peuvent être affinés entre les résolutions de systèmes jusqu'à obtenir une très bonne acuité et donc un gain maximal. Les expériences numériques confirment cela très bien.

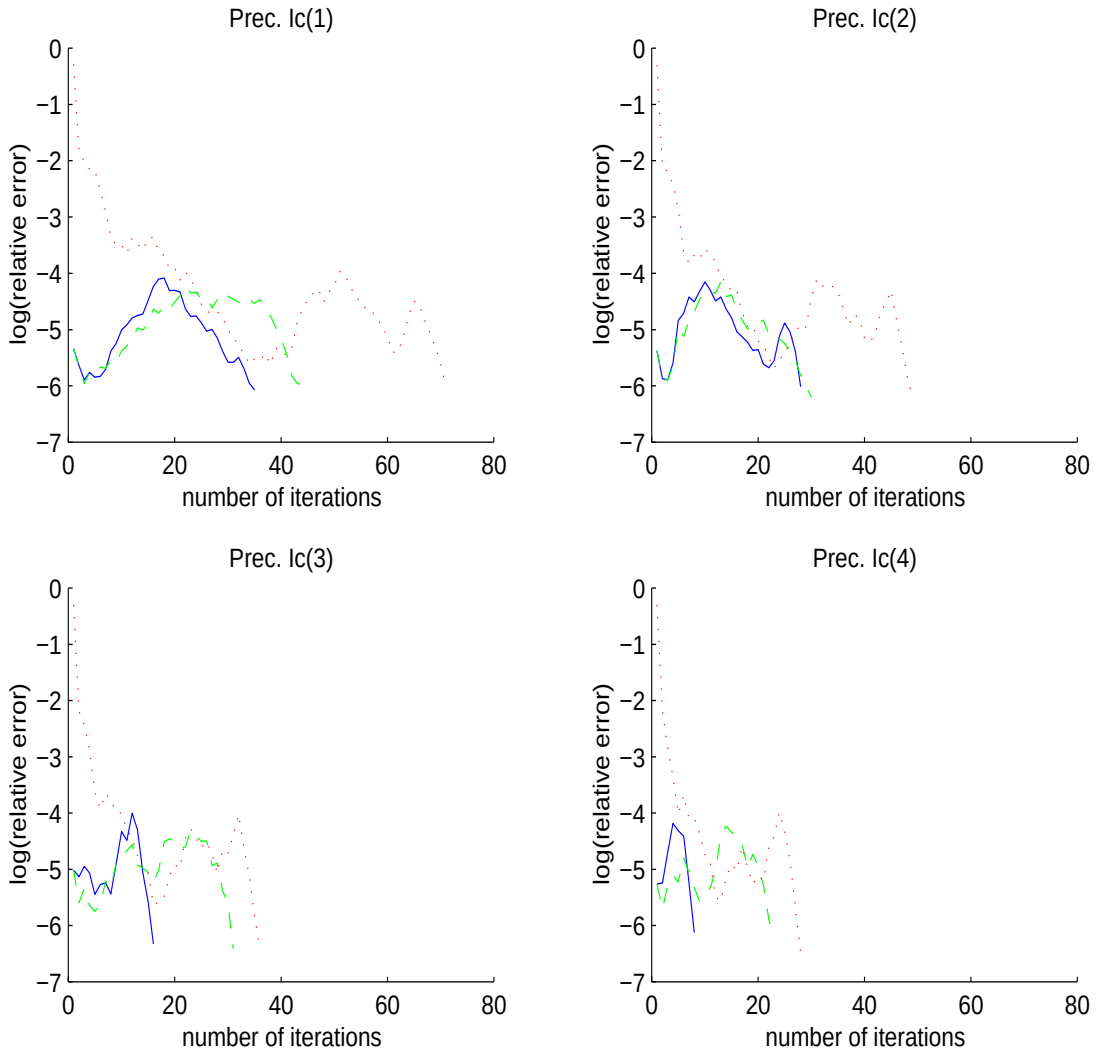


FIG. 4.4 – *Résultats pour la matrice S2RMQ4M1 avec le second membre* $b^{(1)}$

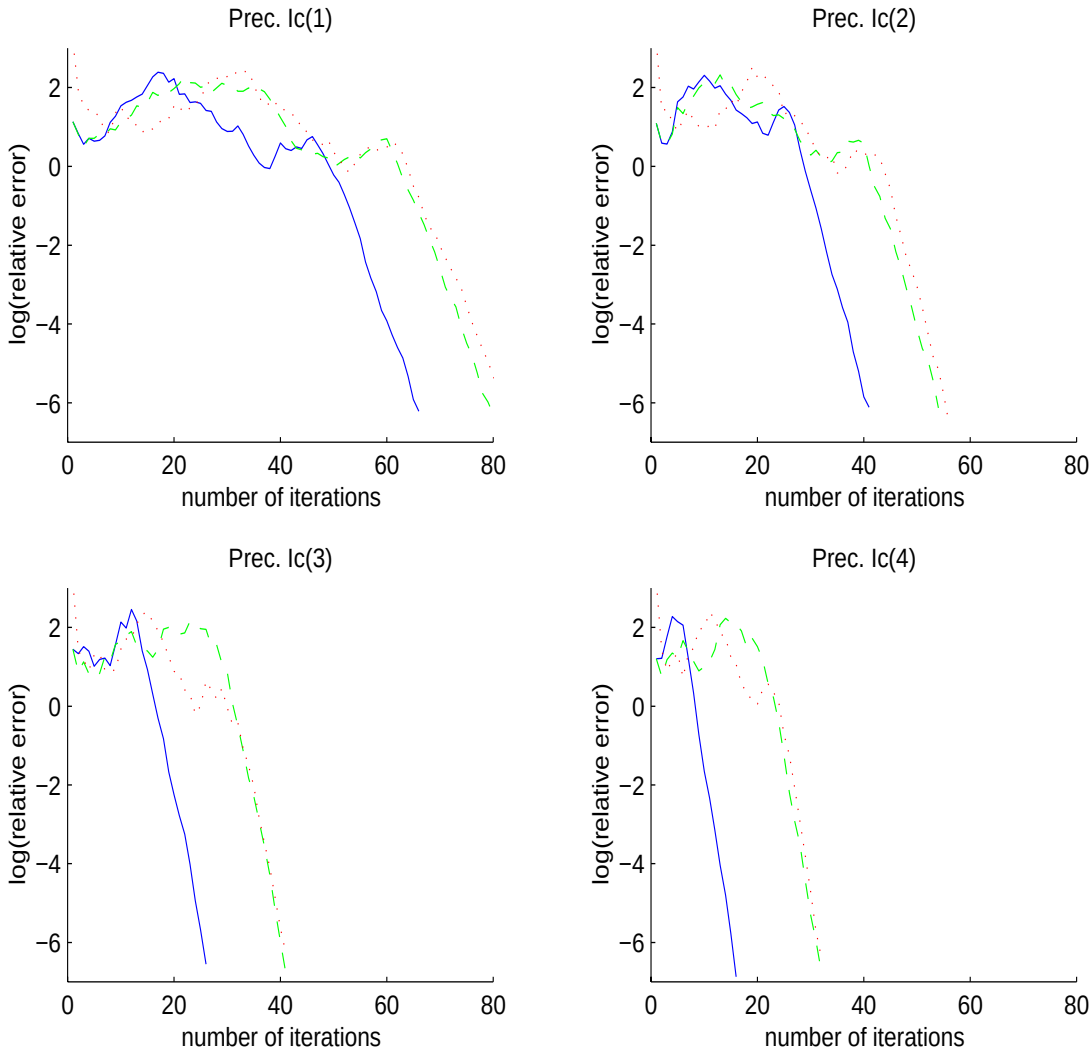


FIG. 4.5 – *Résultats pour la matrice S2RMQ4M1 avec le second membre* $b^{(2)}$

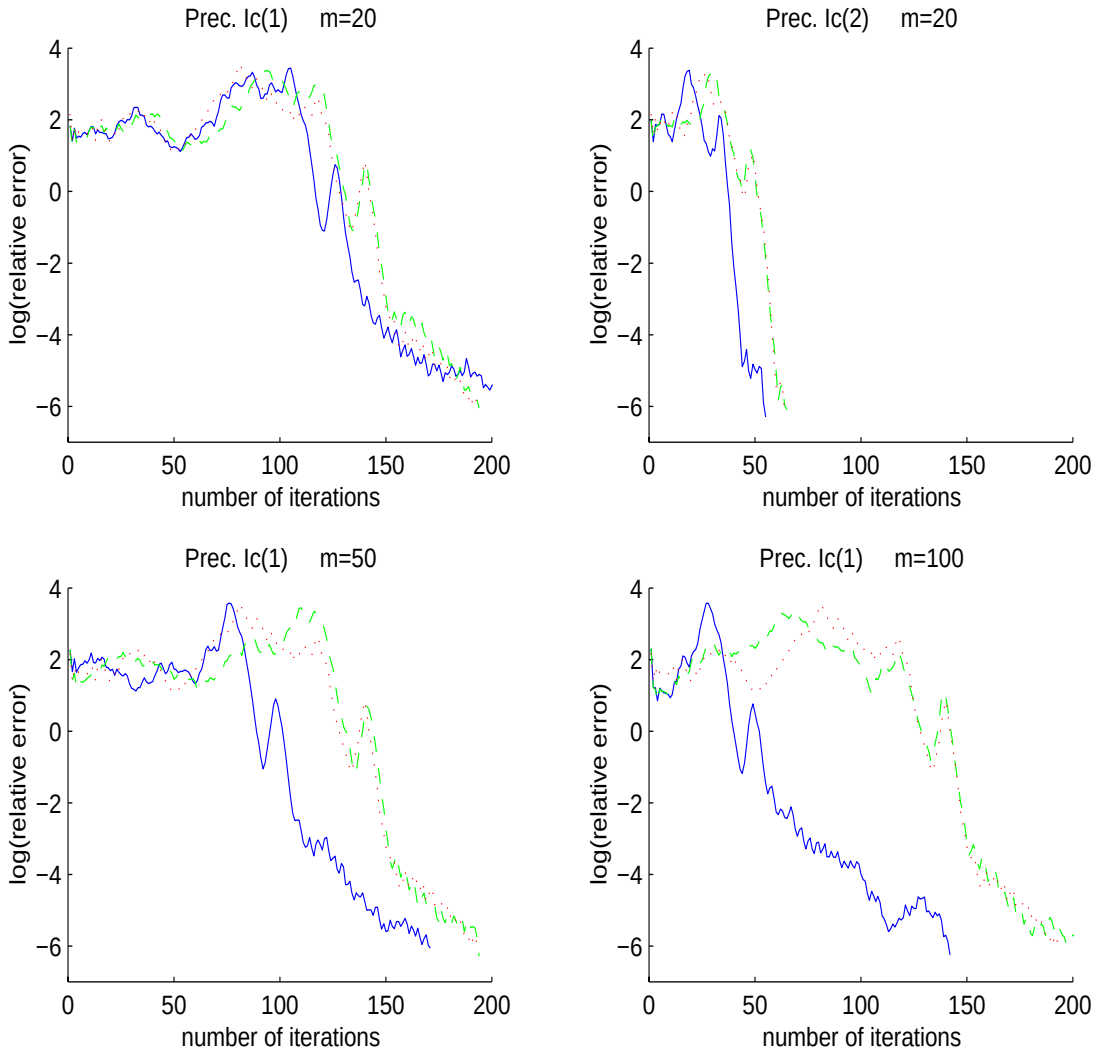


FIG. 4.6 – *Résultats pour la matrice S3RMT3M1 avec le second membre* $b^{(2)}$

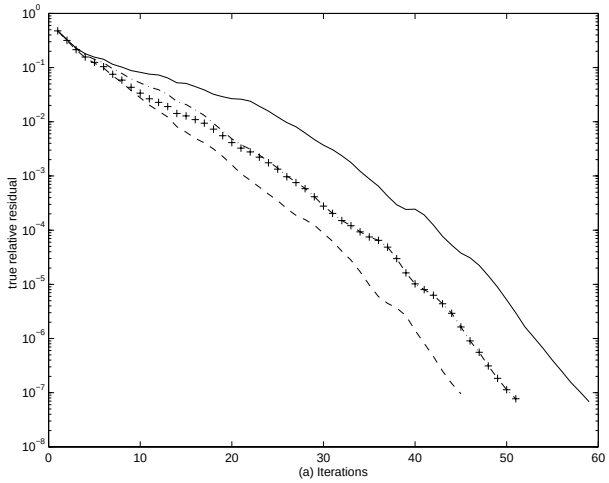


FIG. 4.7 – *Exemple 7*

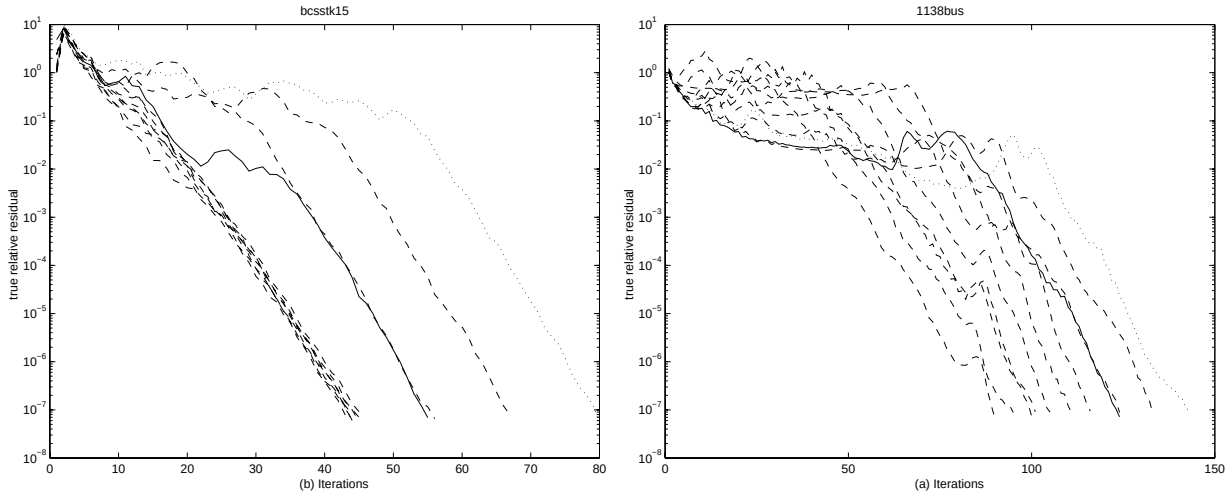


FIG. 4.8 – *Courbes de convergence pour les matrices BCSSTK15 et 1138bus*

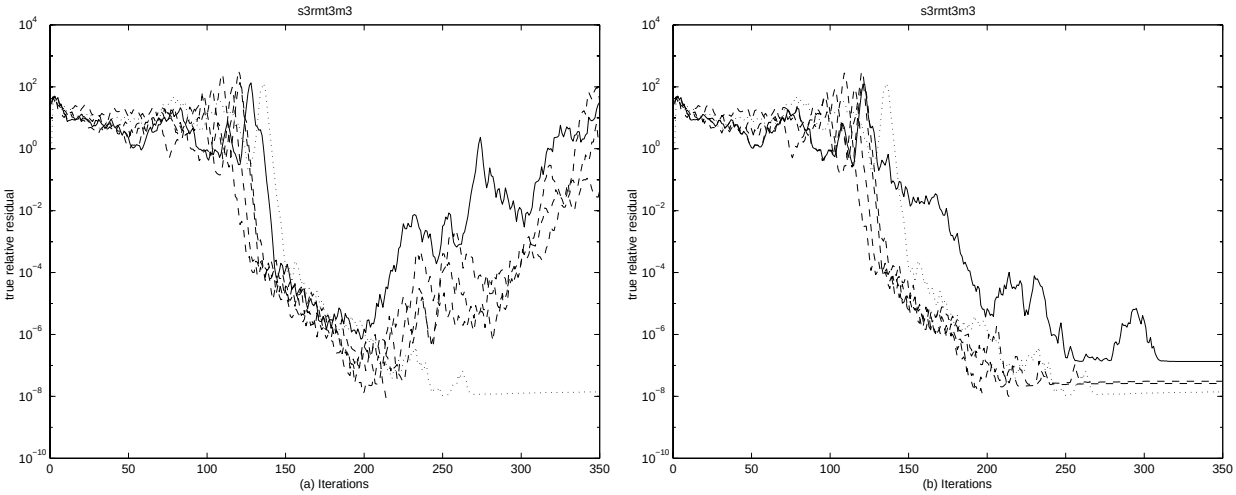


FIG. 4.9 – *Courbes de convergence pour les matrices S3RMT3M3 sans, puis avec correction de* r_j

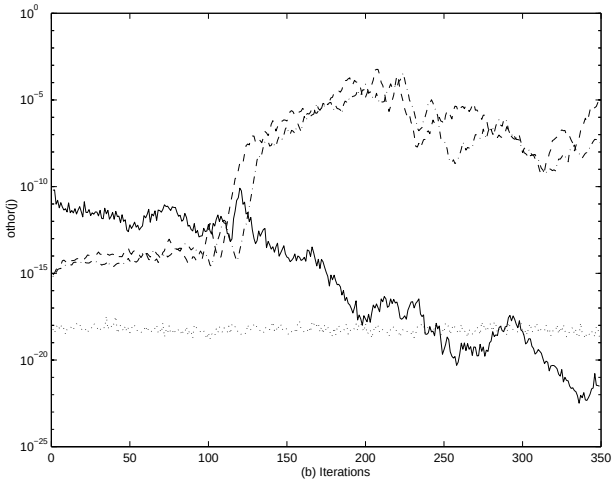


FIG. 4.10 – *Orthogonalité de* r_j *et de* W

Conclusion

On s'intéresse dans cette thèse à la résolution de systèmes linéaires, pierre angulaire du calcul scientifique. Après un rappel des principaux algorithmes pour résoudre les systèmes linéaires et un test comparant ces algorithmes sur un problème de restauration d'images avec des estimateurs robustes, on y présente deux nouveaux volets.

Le premier volet porte sur la régularisation de systèmes linéaires mal-conditionnés dont le second membre est bruité. Pour résoudre ce problème, on présente une nouvelle méthode itérative basée sur des filtres polynomiaux et dont le filtre de régularisation n'est pas *a priori* fixé ; ce qui permet à l'utilisateur de choisir la régularisation en fonction du problème. Cette méthode, de formulation «à la Lanczos», est alors dérivée pour approcher la solution régularisée à chaque itération. Puis on démontre que les filtres utilisés convergent vers la fonction filtre fixée par l'utilisateur. Mais la convergence des approximations x_k vers la solution filtrée x_ϕ n'est pas encore démontrée.

Pour ce qui est du comportement numérique de l'algorithme, on choisit un filtre en deux parties : les composantes de la solution qui sont suivant les directions propres associées à des valeurs propres comprises entre 0 et un paramètre a, sont atténuées pour diminuer la contamination de la solution par le bruit. Les autres composantes ne sont pas modifiées. Les solutions des systèmes sont bien proches de la solution attendue, sans être trop contaminées par le bruit présent dans le second membre. On y voit cependant la grande sensibilité de l'algorithme par rapport au paramètre a, dont dépend le filtre choisi, et le choix du a optimal paraît délicat. Il faudrait pour un choix automatique, coupler cet algorithme avec une méthode de recherche de paramètres optimaux. Il serait aussi intéressant de tester cet algorithme sur un problème où le filtre idéal n'est pas simplement passe-bas ou passe-haut. On pourrait alors couper l'intervalle en plus de deux segments pour la définition de ce filtre.

Le second volet concerne l'augmentation des espaces de Krylov pour accélérer la convergence de l'algorithme du gradient conjugué. Il s'agit de résoudre une séquence de systèmes dont seuls les seconds membres évoluent. On présente tout d'abord des résultats théoriques sur l'adjonction d'un espace W quelconque à l'espace des solutions. On caractérise alors le nouvel espace des solutions comme une somme d'espaces de Krylov et on montre que la vitesse de convergence asymptotique est régie par le conditionnement de H^*AH, où H est la projection A-orthogonale sur $W^{\perp_A}$.

Puis on présente deux nouveaux algorithmes : AugCG et DefCG. Le premier utilise

la structure particulière des espaces de Krylov pour réduire le coût de la projection : en utilisant l'espace de Krylov généré par une résolution antérieure, il suffit d'orthogonaliser par rapport à la dernière direction pour avoir l'orthogonalité toute entière. On ne sait pas estimer précisément le conditionnement du nouveau système, mais on peut voir sur les tests numériques que l'on gagne généralement autant d'itérations que l'on a conservé de vecteurs dans W. Si on résout plus de deux systèmes, on peut conserver plusieurs espaces de Krylov, en orthogonalisant à chaque fois et pour chaque espace, par rapport à la dernière direction. C'est ce qui est fait, par exemple, pour des problèmes de sous-domaines, où la taille du système, qui est celle de l'interface, est petite par rapport à la taille du problème global et donc le stockage est peu important.

Le second algorithme, DefCG, utilise des approximations de Ritz des petites valeurs propres de A, pour enrichir l'espace des solutions. Dans ce cas, on connaît une bonne estimation du nouveau conditionnement et donc le taux asymptotique de convergence. Mais on est obligé d'orthogonaliser par rapport à toutes les directions de W. Un avantage de cet algorithme est qu'il peut être raffiné au fur et à mesure des résolutions : en effet, avec le nouvel espace de Krylov produit, on calcule de nouvelles approximations des vecteurs propres de meilleure qualité, et ce, jusqu'à convergence des paires de Ritz.

Enfin, ces deux algorithmes sont testés sur des exemples tirés de la bibliothèque Harwell-Boeing. Ces tests confirment tous les résultats théoriques. Ces tests mettent aussi en avant la difficulté de régler les paramètres des méthodes : c'est-à-dire le nombre de vecteurs à conserver et le nombre de valeurs propres à calculer.

Il reste donc à généraliser ces algorithmes au cas non symétrique et à les implémenter de manière efficace, pour une intégration dans les bibliothèques scientifiques.

Bibliographie

[1] S. F. Ashby, T. A. Manteuffel, and P. E. Saylor. A taxonomy for conjugate gradient methods. *SIAM J. Numer. Anal.*, 27:1542–1568, 1990.

[2] J. Baranger. *Analyse Numérique*. Hermann, Paris, 1991.

[3] M. Black and A. Rangarajan. On the unification of line processes, outlier rejection, and robust statistics with applications in early vision. *Int. J. Computer Vision*, 19(1):75–104, 1996.

[4] M. Brieu. *Homogénéisation et endommagement de composites élastomères par techniques de calcul parallèle*. thèse de doctorat, Ecole Normale Supérieure de Cachan, France, jan 1999.

[5] A. M. Bruaset. *A survey of preconditioned iterative methods*. Longman Scientific & Technical, Harlow, 1995.

[6] D. Calvetti, L. Reichel, and Q. Zhang. New iterative solution methods for large and very ill-conditioned linear systems of equations. *Numer. Math.*, to appear.

[7] C. Le Calvez and B. Molina. Implicitly restarted and deflated FOM and GMRES. Technical report, LIFL USTL, 1998.

[8] A. Chapman and Y. Saad. Deflated and augmented Krylov subspace techniques. *Numer. Linear Algebra Appl.*, 4:43–66, 1997.

[9] P. Charbonnier, L. Blanc-Féraud, G. Aubert, and M. Barlaud. Deterministic edge-preserving regularization in computed imaging. *IEEE Trans Image Processing*, 6(2):298–311, 1997.

[10] E. Chow and Y. Saad. Experimental study of ILU preconditioners for indefinite matrices. *Journal of Computational and Applied Mathematics*, 87:387–414, 1997.

[11] E. Chow and Y. Saad. Approxmate inverse preconditioners via sparse-sparse iterations. *SIAM J. Sci. Comput.*, 19(3):995–1023, May 1998.

[12] P. Concus, G. H. Golub, and D. P. O'Leary. A generalized conjugate gradient method for the numerical solution of elliptic partial differential equations. Technical Report STAN-CS-76-533, Stanford University, 1976.

[13] A. Van der Sluis and H. Van der Vorst. The rate of convergence of conjugate gradients. *Numerische Mathematik*, 48:543–560, 1986.

[14] H. A. Van der Vorst and T. F. Chan. Linear system solvers : sparse iterative methods. *Parallel Numerical Algorithms, ICASE/LaRC Interdisciplinary Series in Science and Engineering*, 4:91–118, 1997.

[15] J. Erhel and F. Guyomarc'h. An Augmented Conjugate Gradient Method for Solving Consecutive Symmetric Positive Definite Linear Systems. *Siam J. Matrix Anal. Appl.*, 21(4):1279–1299, 2000.

[16] D. Geman and G. Reynolds. Constrained restoration and the recovery of discontinuities. *IEEE Trans. Pattern Anal. Machine Intell.*, 14(3):367–383, 1992.

[17] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins, Baltimore and London, 1996. Third Edition.

[18] G. H. Golub and G. Meurant. Matrices, moments and quadrature. *Numerical Analysis*, 303:105–156, 1994.

[19] G. H. Golub and G. Meurant. Matrices, moments and quadrature II or how to compute the norm of the error in itérative methods. *BIT*, 37(3):687–705, 1997.

[20] M. Hanke. *Conjugate gradient type methods for ill-posed problems*. Longman Scientific & Technical, Harlow, 1995.

[21] P. C. Hansen. Truncated svd solutions to discrete ill-posed problems with ill-determined numerical rank. *Siam J. Sci. Statist. Comput.*, 11:503–518, 1990.

[22] P. C. Hansen. Regularization tools. A matlab Package for Analysis and Solution of Discrete Ill-Posed Problems, June 1992.

[23] P. C. Hansen, T. Sekii, and H. Shibahashi. The modified truncated SVD method for regularization in general form. *Siam J. Sci. Statist. Comput.*, 13(5):1142–1150, September 1992.

[24] P. Holland and R. Welsh. Robust regression using iteratively reweighted least-squares. *Commun. Statis.-Theor. Meth.*, A6(9):813–827, 1977.

[25] P. J. Hubert. *Robust Statistics*. John Wiley & Sons, New York, 1981.

[26] K. Burrage J. Erhel and B. Pohl. Restarted GMRES preconditioned by deflation. *Journal of Computational and Applied Mathematics*, 69:303–318, 1996.

[27] W. D. Joubert and T. A. Manteuffel. *Iterative methods for Nonsymmetric Linear Systems*, chapter 10, pages 149–171. Academic Press, 1990.

[28] J. A. Meijerink and H. A. Van Der Vorst. An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrices. *Math. Comp.*, 31(137):148–162, 1977.

[29] G. Meurant. *Computer solution of large linear systems*. North Holland, Amsterdam, 1999.

[30] É. Mémin and T. Risset. Hardware driven considerations for VLSI derivation of discontinuity preserving energy based applications. *Real Time Imaging*, to appear.

[31] R. B. Morgan. A restarted GMRES method augmented with eigenvectors. *SIAM J. Matrix Analysis and Applications*, 16(4):1154–1171, 1995.

[32] A. Neumaier. Solving ill-conditioned and singular linear systems: a tutorial on regularization. *Siam Rev.*, 40(3):636–666, September 1998.

[33] D. P. O'Leary. The block conjugate gradient algorithm and related methods. *Linear algebra and its applications*, 29:293–322, 1980.

[34] B. Parlett. A new look at the Lanczos algorithm for solving symmetric systems of linear equations. *Linear algebra and its applications*, 29:323–346, 1980.

[35] Y. Saad. On the Lanczos method for solving symmetric linear systems with several right-hand sides. *Mathematics of computation*, 178:651–662, 1987.

[36] Y. Saad. *Iterative Methods for Sparse Linear Systems*. PWS Publishing Compagny, Boston, 1996.

[37] Y. Saad and B. Suchomel. ARMS: An algebraic recursive multilevel solver for general sparse linear systems. Technical Report umsi-99-107, MSI, 1999.

[38] Y. Saad, M. Yeung, J. Erhel, and F. Guyomarc'h. A Deflated Version of the Conjugate Gradient Algorithm. *Siam J. Sci. Comput.*, 21(5):1909–1926, 2000.

[39] Y. Saad and J. Zhang. BILUTM: A domain-based multi-level block ILUT preconditioner for general sparse matrices. Technical Report umsi-98-118, MSI, 1998.

[40] G. L. Sleijpen and A. Van der Sluis. Further results on the convergence behavior of conjugate-gradients and Ritz values. *Linear algebra and its applications*, 246:233–278, 1996.

[41] A. N. Tikhonov. Regularisation of incorrectly posed problems. *Soviet. Math. Dokl.*, 4:1624–1627, 1963.

[42] A. N. Tikhonov. Solution of incorrectly formulated problems and the regularisation method. *Soviet. Math. Dokl.*, 4:1036–1038, 1963.

[43] H. van der Vorst. An iterative method for solving $f(A)x = b$ using Krylov subspace information obtained for the symmetric positive definite matrix A. *Journal of Computational and Applied Mathematics*, 18:249–263, 1987.

[44] R. S. Varga. *Matrix Iterative Analysis*. Prentice-Hall, Englewood Cliffs, 1962.

Table des matières

Table des figures

Table des algorithmes

Résumé

De nombreux problèmes de calcul scientifique réclament la résolution de systèmes linéaires. Des algorithmes récents et performants pour résoudre ces systèmes sont basés sur les méthodes de Krylov. L'espace des solutions de celles-ci est un espace de Krylov et la solution est alors définie par une condition d'orthogonalité dite de Galerkin.

Dans une première partie, on modifie la définition de la solution pour la résolution de systèmes mal-conditionnés, en introduisant une nouvelle technique de régularisation basée sur des filtres polynomiaux. Le point fort de cette méthode est que la forme des filtres n'est pas fixée par la méthode mais peut être quelconque, et donc dictée par les spécificités du problème.

Dans la seconde partie, on modifie l'espace des solutions pour accélérer la convergence. Deux techniques sont explorées. La première permet de recycler un espace de Krylov utilisé pour résoudre une première équation. La seconde, basée sur des techniques de déflation, cherche à atténuer l'effet néfaste des plus petites valeurs propres. Cette dernière peut, de plus, s'affiner lors de la résolution de plusieurs systèmes, jusqu'à éliminer complètement l'impact de ces petites valeurs propres.

Tous ces algorithmes sont implémentés et testés sur des problèmes issus de l'analyse d'images et de la mécanique. Cette validation numérique confirme les résultats théoriques.
Mots clés : solveur linéaire, espace de Krylov, gradient conjugué, régularisation, filtres polynomiaux, déflation.

Abstract

Many problems in scientific computation require to solve linear systems. Recent efficient solvers are based on Krylov methods. Their solution space is a Krylov subspace and the solution is then defined by an orthogonality condition, called Galerkin's condition.

In the first part, the definition of the solution is modified for the resolution of ill-conditionned systems and a new regularization technique, based on polynomial filters, is introduced. The main interest in the method is that the shape of the filters is independant of the method. It can be arbitrary, and thus directed by the specificities of the problem.

In the second part, the solution space is modified to accelerate the convergence. Two techniques are proposed. The first allows a Krylov subspace to be reused in the solution of a succeeding equation. The second, based on deflation techniques, tries to dampen the effect of the smallest eigenvalues. Moreover, it can be refined when solving multiple systems, to progressively eliminate the impact of these small eigenvalues.

These algorithms are implemented and tested on problems from image analysis and mecanic. This numerical validation confirms the theoretical results.
Key words : linear solver, Krylov subspace, conjugate gradient, regularization, polynomial filters, deflation.