Calcul matriciel et systèmes linéaires

Jocelyne Erhel Nabil Nassif Bernard Philippe

DEA Informatique et modélisation

Année 2004

Beyrouth, Liban

Contents

Chapter 1

Bases de l'algèbre linéaire

Ce chapitre introduit les notations et les opérations de base sur l'algèbre des matrices. Il se termine par des notions de complexité et de performances et par la description des bibliothèques BLAS et LAPACK.

1.1 Espaces vectoriels et vecteurs

On note $\mathbb{R}^n$ un espace vectoriel de dimension n.
La base canonique de $\mathbb{R}^n$ est $(e_1, \ldots, e_n)$.
Un vecteur $x \in \mathbb{R}^n$, de composantes $x_1, \ldots, x_n$, est noté $x = (x_i)$.
Les vecteurs sont notés verticalement.

1.2 Matrices et vecteurs

Une matrice $A \in \mathbb{R}^{m \times n}$ est notée $A = (a_{ij})$.
Le j^{eme} vecteur colonne de A est $a_j = Ae_j$.
Un système de n vecteurs $u_1, \ldots, u_n \in \mathbb{R}^m$ est noté sous forme matricielle $U = (u_1 \ldots u_n) \in \mathbb{R}^{m \times n}$, avec u_j le j^{eme} vecteur colonne de U.
On note $Vect(U)$ le sous-espace vectoriel (sev) engendré par les vecteurs colonnes de U.
Si U est un système libre de k vecteurs, $Vect(U)$ est un sev de dimension k.

1.2.1 Matrices carrées et matrices particulières

Une matrice $A \in \mathbb{R}^{m \times n}$ est carrée d'ordre n si $n = m$.

La trace d'une matrice carrée A d'ordre n est la somme de ses éléments diagonaux : $tr(A) = \sum_{i=1}^{n} a_{ii}$.

Le déterminant d'une matrice carrée A d'ordre n est noté $det(A)$.

La matrice identité d'ordre k, dans $\mathbb{R}^{k \times k}$, vaut $I_k = (e_1 \ldots e_k)$.

Une matrice carrée D est diagonale si les seuls éléments non nuls sont sur la diagonale ; elle est notée $D = diag(d_i)$, où $d = (d_i)$ est le vecteur formé par les éléments diagonaux.

Une matrice carrée L triangulaire inférieure si les seuls éléments non nuls sont dans le triangle inférieur ; on définit de même une matrice carrée U triangulaire supérieure.

Une matrice tridiagonale a trois diagonales non nulles, une matrice bidiagonale a deux diagonales non nulles.

1.2.2 Opérations sur les matrices

L'ensemble des matrices $\mathbb{R}^{m\times n}$ est un espace vectoriel de dimension mn.

Soit $A \in \mathbb{R}^{m\times n}$ et $B \in \mathbb{R}^{n\times p}$; le produit $C = AB \in \mathbb{R}^{m\times p}$ est défini par $c_{ij} = \sum_{k=1}^{n} a_{ik}b_{kj}$.

L'ensemble des matrices carrées $\mathbb{R}^{n\times n}$ est un anneau.

L'anneau n'est pas commutatif (il existe A et B tels que $AB \neq BA$).

L'anneau n'est pas intègre (il existe des diviseurs de zéro : il existe A et B tels que $AB = 0$).

Une matrice carrée A est inversible s'il existe une matrice B telle que $AB = I_n$. Si B existe, alors $BA = I_n$, B est unique, c'est l'inverse de A et on note $B = A^{-1}$.

L'anneau n'est pas un corps (il existe des matrices non inversibles, dites singulières).

Le produit de deux matrices diagonales est une matrice diagonale.

Le produit de deux matrices triangulaires est une matrice triangulaire.

1.2.3 Matrices symétriques

La transposée A^T d'une matrice carrée A est la matrice obtenue en interchangeant les lignes et les colonnes. Soit $A = (a_{ij})$ et $B = A^T = (b_{ij})$, on a donc $b_{ij} = a_{ji}$.

Une matrice carrée A est symétrique si $A = A^T$.

Les matrices A^TA et AA^T sont symétriques.

On a $(A^T)^T = A$ et $(AB)^T = B^TA^T$.

1.2.4 Partitions par blocs

Une matrice par blocs est définie par une partition où les éléments scalaires sont regroupés dans des sous-matrices ; on note $A = A_{ij}$.

On définit les mêmes règles d'opérations, en respectant les dimensions dans les produits. Attention à l'ordre des blocs dans les produits.

1.3 Produit scalaire et normes vectorielles

Le produit scalaire de deux vecteurs x et y est $x^Ty = \sum_{i=1}^{n} x_iy_i$.

Soit $x = (x_i)$, on a $x_i = e_i^Tx$.

Soit $A = (a_{ij})$, on a $a_{ij} = e_i^TAe_j$.

Dans le cas de vecteurs complexes, le produit scalaire hermitien est défini par

$$(x, y) \in \mathbb{C}^n \times \mathbb{C}^n \quad \rightarrow \quad x^*y = \sum_{i=1}^{n} \bar{x}_iy_i,$$

où le surlignage d'une grandeur indique qu'on en considère le conjugué.

Il est possible de définir plusieurs normes dans l'espace vectoriel $\mathbb{R}^n$.

Définition 1.3.1 *Une norme d'un espace vectoriel E est une application $\|.\|$ de E dans $\mathbb{R}_+$ qui vérifie les propriétés suivantes :*

$$\begin{aligned}
&\forall x \in E,\ \|x\| \geq 0,\\
&\forall \lambda \in \mathbb{R},\ x \in E,\ \|\lambda x\| = |\lambda|\|x\|,\\
&\forall x, y \in E,\ \|x+y\| \leq \|x\| + \|y\|.
\end{aligned}$$

Dans $\mathbb{R}^n$, les trois normes les plus courantes sont la norme infinie, la norme 1 et la norme euclidienne.

- *norme infinie :* $\|x\|_\infty = \max_{i=1,\cdots,n} |x_i|$,
- *norme 1 :* $\|x\|_1 = \sum_{i=1}^n |x_i|$,
- *norme 2 ou norme euclidienne :* $\|x\|_2 = \sqrt{x^T x} = \sum_{i=1}^n {x_i}^2$

où $x = (x_1, \cdots, x_n)^T$.

La norme euclidienne est donc définie par le produit scalaire $x^T y$.

Les vecteurs de la base canonique sont normés : $\|e_j\|_2 = 1$.

Proposition 1.3.1 *Inégalité de Cauchy-Schwarz :*

$$\forall (x, y) \in \mathbb{R}^n \times \mathbb{R}^n,\ |x^T y| \quad \leq \quad \|x\|_2 \|y\|_2. \tag{1.1}$$

l'égalité a lieu si et seulement si les vecteurs x et y sont liés.

Preuve. Laissée en exercice. ◇

Comme toutes les normes d'un espace de dimension finie, ces trois normes sont équivalentes. Les constantes qui les relient sont données dans la proposition suivante.

Proposition 1.3.2 *Pour tout vecteur $x \in \mathbb{R}^n$, on a les inégalités :*

$$\|x\|_2 \leq \|x\|_1 \leq \sqrt{n}\|x\|_2, \tag{1.2}$$
$$\|x\|_\infty \leq \|x\|_2 \leq \sqrt{n}\|x\|_\infty, \tag{1.3}$$
$$\|x\|_\infty \leq \|x\|_1 \leq n\|x\|_\infty. \tag{1.4}$$

Preuve. Laissée en exercice. ◇

1.4 Normes matricielles

Définition 1.4.1 *On suppose que l'on a choisi une norme dans chacun des deux espaces $\mathbb{R}^n$ et $\mathbb{R}^m$. On définit alors la norme matricielle subordonnée dans l'espace des matrices $\mathbb{R}^{m\times n}$ par*

$$\forall A \in \mathbb{R}^{m\times n},\ \|A\| = \max_{\|x\|=1} \|Ax\|.$$

Lorsque les normes 1, 2 ou infinie sont respectivement choisies pour les deux ensembles à la fois, on note les normes subordonnées correspondantes de la même manière.

Proposition 1.4.1 *Soit $A = (a_{ij}) \in \mathbb{R}^{m\times n}$. Alors :*

$$\|A\|_1 = \max_{j=1,\cdots,n} \sum_{i=1}^{m} |a_{ij}|, \tag{1.5}$$

$$\|A\|_\infty = \max_{i=1,\cdots,n} \sum_{j=1}^{m} |a_{ij}|. \tag{1.6}$$

Preuve. Laissée en exercice. ◇

Remarque 1.4.1 *La norme matricielle euclidienne n'est pas simple à calculer dans le cas général, contrairement aux autres normes. Voir le chapitre sur les valeurs singulières.*

Proposition 1.4.2 *Dans certains cas particuliers, la norme des matrices est connue.*

$$\|I\|_2 = 1$$
$$\|D\|_2 = max_i|d_i|$$

Définition 1.4.2 *Une norme matricielle de $\mathbb{R}^{n\times n}$ est une norme qui vérifie*

$$\forall A, B \in \mathbb{R}^{n\times n},\ \|AB\| \leq \|A\|\|B\|.$$

Proposition 1.4.3 *Les normes subordonnées sont des normes matricielles.*

La norme de Frobenius est la norme prise au sens de l'espace vectoriel de dimension mn.

Définition 1.4.3 *Pour toute matrice $A = (a_{ij}) \in \mathbb{R}^{n\times m}$, on définit sa norme de Frobenius par :*

$$\begin{aligned} \|A\|_F &= \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{m} {a_{ij}}^2}, \\ &= \sqrt{tr(A^T A)}, \end{aligned}$$

où la trace d'une matrice est égale à la somme de ses éléments diagonaux.

Proposition 1.4.4 *La norme de Frobenius est une norme matricielle. Elle vérifie les inégalités suivantes*

$$\begin{aligned} \|A\|_2 \leq \|A\|_F &\leq \sqrt{n}\|A\|_2, \\ \|AB\|_F &\leq \|A\|_F\|B\|_2, \\ \|AB\|_F &\leq \|A\|_2\|B\|_F, \\ \|AB\|_F &\leq \|A\|_F\|B\|_F, \end{aligned}$$

pour tout couple de matrices $(A, B) \in \mathbb{R}^{n\times m} \times \mathbb{R}^{m\times p}$.

Preuve. Laissée en exercice. ◇

1.5 Orthogonalité dans $\mathbb{R}^n$

Définition 1.5.1 $x \perp y \Leftrightarrow x^T y = 0$

Définition 1.5.2 $\cos(angle(x,y)) = \frac{x^T y}{\|x\|_2 \|y\|_2}$

Proposition 1.5.1 *Théorème de Pythagore :*
Si $x \perp y$, alors $\|x+y\|_2^2 = \|x\|_2^2 + \|y\|_2^2$.

Définition 1.5.3 *Soit S un sous-espace vectoriel de $\mathbb{R}^n$. L'orthogonal de S est défini par*

$$S^\perp = \{y / y^T x = 0, \quad \forall x \in S\}.$$

Proposition 1.5.2 *$S^\perp$ est un sous-espace vectoriel et les sous-espaces S et $S^\perp$ sont supplémentaires.*

Définition 1.5.4 *$U = (u_1 \cdots u_k) \in \mathbb{R}^{n\times k}$, $k \leq n$ est un système orthonormé ssi $u_i^T u_j = \delta_{ij}$ ssi $U^T U = I_k$.*

Remarque 1.5.1 *Attention, si $k < n$, on a $UU^T \neq I_n$.*

Proposition 1.5.3 *Un système orthonormé forme un système libre.*

Remarque 1.5.2 *$(e_1, \ldots, e_n)$ est une base orthonormée de $\mathbb{R}^n$.*

Proposition 1.5.4 *Théorème de la base incomplète. Soit U un système orthonormé de taille k. On peut compléter U par U_1 de taille $n-k$, pour former une base orthonormée de $\mathbb{R}^n$. Le système U_1 est une base orthonormée de $U^\perp$. Alors $U_1^T U_1 = I_{n-k}$ et $U^T U_1 = 0$. Tout vecteur x s'écrit*

$$x = UU^T x + U_1 U_1^T x.$$

Proposition 1.5.5 *$U \in \mathbb{R}^{n\times k}$, $k \leq n$ système orthonormé, alors*

$$\begin{array}{l} \|U\|_2 = 1. \\ \forall x \in \mathbb{R}^k, \quad \|Ux\|_2 = \|x\|_2, \\ \forall A \in \mathbb{R}^{k\times p}, \|UA\|_2 = \|A\|_2. \end{array}$$

Attention, si $k < n$, on a $\|AU\|_2 \neq \|A\|_2$.

Preuve. $\|Ux\|_2^2 = (Ux)^T(Ux) = x^T(U^T U)x = x^T x = \|x\|_2^2$
$\|UA\|_2 = \max_{\|x\|_2=1} \|UAx\|_2 = \max_{\|x\|_2=1} \|Ax\|_2 = \|A\|_2$. ◇

Définition 1.5.5 *Une matrice carrée $Q \in \mathbb{R}^{n\times n}$ est orthogonale ssi $Q^T Q = I_n$. Les colonnes de Q forment une base orthonormée de $\mathbb{R}^n$.*

Proposition 1.5.6 *$Q \in \mathbb{R}^{n\times n}$ matrice orthogonale, alors Q est inversible, Q^T est orthogonale et*

$$\begin{array}{l} Q^T Q = QQ^T = I_n, \ Q^{-1} = Q^T. \\ \|Q\|_2 = 1. \\ \forall A \in \mathbb{R}^{n\times p}, \|QA\|_2 = \|A\|_2, \\ \forall A \in \mathbb{R}^{m\times n}, \|AQ\|_2 = \|A\|_2. \end{array}$$

Preuve. $\|AQ\|_2 = \max_{\|x\|_2=1} \|AQx\|_2 = \max_{\|y\|_2=1} \|Ay\|_2 = \|A\|_2$ car Q est inversible ($\forall y, \exists x, Qx = y$) et $\|Qx\|_2 = \|x\|_2$. ◇

Proposition 1.5.7 *$Q \in \mathbb{R}^{n\times n}$ avec $\|Q\|_2 = 1$, alors Q est orthogonale. Autrement dit, une matrice carrée qui converve les normes est orthogonale.*

Preuve. $(Qx)^T(Qx) = x^T(Q^TQ)x = x^Tx$ donc $(Q(x+y))^TQ(x+y) = (x+y)^T(x+y)$ et $(Qx)^T(Qy) = x^Ty$.

D'où $e_i^T(Q^TQ)e_i = 1$ et $e_j^T(Q^TQ)e_i = 0,\ i \neq j$ donc $Q^TQ = I_n$ et Q est orthogonale. ◇

1.6 Image, noyau, rang d'une matrice

$A \in \mathbb{R}^{m\times n}$.

Définition 1.6.1 $Im(A) = \{y \in \mathbb{R}^m / y = Ax\} = Vect(a_1, a_2, \cdots, a_n)$
$ker(A) = \{x \in \mathbb{R}^n / Ax = 0\}$

Proposition 1.6.1 *$Im(A)$ est un sev de $\mathbb{R}^m$ et $ker(A)$ est un sev de $\mathbb{R}^n$.*

Définition 1.6.2 $rang(A) = dim(Im(A))$.

Proposition 1.6.2 $Im(A)^\perp = ker(A^T)$ *et* $Im(A^T) = ker(A)^\perp$.

Preuve. $y \in Im(A)^\perp \Leftrightarrow \forall x,\ \ (Ax)^Ty = 0 \Leftrightarrow \forall x,\ \ x^T(A^Ty) = 0 \Leftrightarrow A^Ty = 0 \Leftrightarrow y \in ker(A^T)$. ◇

Proposition 1.6.3 $rang(A) = rang(A^T)$
$dim(ker(A)) + rang(A) = n$
$dim(ker(A^T)) + rang(A^T) = m$
$rang(A) \leq min(m, n)$.

Preuve. Soit $r = rang(A^T)$, on va montrer que $rang(A) \geq r$. Par transposition, on en déduira que $r = rang(A)$.
Soit $X = \{x_1, \ldots, x_r\}$ une base de $Im(A^T)$ et $Y = AX$. Par construction, $Y \in Im(A)$. On va montrer que Y est un système libre, ce qui implique que $r \leq rang(A)$. Pour cela, on va montrer que $Yv = 0 \Rightarrow v = 0$.

$Yv = 0 \Rightarrow AXv = 0 \Rightarrow Xv \in ker(A)$. Or $ker(A) = Im(A^T)^\perp$ donc $Xv \in Im(A^T)^\perp$. Mais X est une base de $Im(A^T)$ donc $Xv \in Im(A^T)$. Donc $Xv = 0$ et puisque X est une base, $v = 0$.

D'après la proposition précédente, on a $dim(ker(A)) + rang(A^T) = n$, on en déduit l'égalité avec $rang(A)$. ◇

Proposition 1.6.4 $rang(A) = n \Leftrightarrow ker(A) = \{0\}$.

Preuve. Évident d'après ce qui précède.
◇

Définition 1.6.3 *Soit $A \in \mathbb{R}^{m\times n}$ une matrice rectangulaire avec $m \geq n$; A est dite de rang plein si $rang(A) = n$.*

Proposition 1.6.5 $rang(AB) \leq rang(A),\ rang(AB) \leq rang(B)$

Preuve. $ker(B) \subset ker(AB)$ donc $rang(AB) \leq rang(B)$.

$rang((AB)^T) = rang(B^TA^T) = rang(AB) \leq rang(A^T) = rang(A)$. ◇

1.6.1 Matrices de rang k

Proposition 1.6.6 *Soit $U = (u_1 \dots u_k)$ et $V = (v_1 \dots v_k)$ deux systèmes libres de $\mathbb{R}^m$ et de $\mathbb{R}^n$ respectivement, alors $UV^T \in \mathbb{R}^{m \times n}$ et*

$$ker(UV^T) = V^\perp,\ Im(UV^T) = Vect(U),\ rang(UV^T) = k.$$

Réciproquement, si A est une matrice de rang k, il existe $U = (u_1 \dots u_k)$ base de $Im(A)$ et $V = (v_1 \dots v_k)$ base de $ker(A)^\perp$ tels que $A = UV^T$.

Preuve. Soit $A = UV^T$, alors $x \in ker(A) \Leftrightarrow UV^T x = 0 \Leftrightarrow V^T x = 0 \Leftrightarrow x \in V^\perp$ donc $ker(A) = V^\perp$. On en déduit que $rang(A) = k$ car $dim(V^\perp) = n-k$. D'autre part, $Im(A) \subset Im(U)$ d'où, par égalité des dimensions, $Im(A) = Im(U)$.

Réciproquement, soit $V \in \mathbb{R}^{n \times k}$ une base orthonormée de $ker(A)^\perp$, complétée par V_1 dans $ker(A)$. Alors $\forall x,\ x = VV^T x + V_1 V_1^T x$ et $Ax = AVV^T x$. Soit $U = AV$, alors $Ax = UV^T x$ donc $A = UV^T$. De plus, U est un système libre car $Uy = 0 \Leftrightarrow AVy = 0 \Leftrightarrow Vy \in ker(A) \cap ker(A)^\perp \Leftrightarrow y = 0$, donc U est une base de $Im(A)$. ◇

1.6.2 Matrices de rang 1

En particulier, les matrices de rang 1 sont de la forme xy^T, où x et y sont deux vecteurs non nuls.

Proposition 1.6.7 *Si $v^T u \neq 0$, la transformation $P = \frac{1}{v^T u} uv^T$ est la projection sur la droite engendrée par le vecteur u orthogonalement au vecteur v. Si de plus $u = v$, la projection P est une projection orthogonale.*

Preuve. Il est évident que $Pu = u$ et $Pw = 0$ pour tout $w \perp v$. Puisque $\mathbb{R}^n = (u) \oplus (v)^\perp$, la projection P est caractérisée. Si $u = v$, la projection $P = \frac{1}{{\|u\|_2}^2} uu^T$ est la projection orthogonale sur (u). ◇

Remarque 1.6.1 *Si $u \perp v$, alors la transformation $N = \alpha uv^T$ est nilpotente, de noyau l'hyperplan orthogonal à v.*

Proposition 1.6.8 *Si $v^T u \neq 0$, la transformation $Q = I - \frac{1}{v^T u} uv^T$ est la projection sur l'hyperplan $(v)^\perp$ parallèlement à la droite engendrée par le vecteur u. Si de plus $u = v$, la projection Q est la projection orthogonale sur l'hyperplan $(u)^\perp$.*

Preuve. Évident. ◇

1.7 Notions de complexité et de performances

Les algorithmes de calcul sont caractérisés par leur complexité arithmétique, mesurée par le nombre d'opérations (additions, multiplications, etc) sur des réels, ainsi que par leur coût de stockage, mesuré par le nombre de variables réelles. Le stockage des nombres entiers et les calculs sur les nombres entiers sont d'un coût marginal, qui est ignoré. En pratique, les opérations arithmétiques et le stockage se font sur des nombres flottants (voir chapitre sur l'arithmétique flottante). Les opérations inutiles, telles qu'une multiplication par zéro ou par un, ne sont pas comptabilisées.

Les performances d'un algorithme se mesurent par sa vitesse de calcul. Celle-ci dépend de la complexité mais aussi de l'exploitation de l'architecture de l'ordinateur, notamment du parallélisme interne et de la hiérarchie des mémoires.

Pour la complexité, on donnera souvent le terme prédominant, du plus grand ordre. Pour une complexité polynomiale, ce terme est écrit sous la forme $O(n^k)$. Cela signifie que le nombre d'opérations N divisé par n^k tend vers une constante quand n tend vers l'infini.

La plupart des algorithmes d'algèbre linéaire ont une complexité polynomiale, par exemple $N = an^3 + bn^2 + cn + d$. On a pour cet exemple $N = O(n^3) = an^3 + O(n^2)$.

1.8 Bibliothèques BLAS et LAPACK

Les opérations de base d'algèbre linéaire sont regroupées dans une bibliothèque numérique appelée BLAS : Basic Linear Algebra Subroutines. Cette bibliothèque est souvent fournie par le constructeur et optimisée pour une architecture donnée. Les opérations sont divisées en trois niveaux, appelés BLAS1, BLAS2, BLAS3. L'optimisation concerne l'ordre des opérations et l'accès à la mémoire, pour exploiter au mieux la hiérarchie (mémoire principale, mémoire cache, etc). Les performances (vitesse de calcul) sont d'autant meilleures que le niveau est élevé.

Les opérations plus complexes sont regroupées dans la bibliothèque numérique appelée LAPACK : Linear Algebra Package. Les opérations de LAPACK utilisent au maximum les opérations BLAS, surtout BLAS3, qui est le plus performant en temps de calcul.

1.8.1 Opérations BLAS1

Ce niveau concerne les opérations entre vecteurs. Quelques exemples :

- combinaison linéaire de vecteurs $z = a * x + b * y$, $3n$ opérations.
- produit scalaire de vecteurs $a = a + x^T y$, $2n$ opérations.
- produit de matrices diagonales, n opérations.

Toutes les opérations BLAS1 ont une complexité en $O(n)$ opérations flottantes et un accès mémoire en $O(n)$ mots flottants.

Il n'y a qu'un niveau de boucle.

1.8.2 Opérations BLAS2

Ce niveau concerne les opérations entre matrices et vecteurs. Quelques exemples :

- produit matrice-vecteur $y = y + A * x$, $2mn$ opérations.
- produit extérieur de vecteurs $A = xy^T$, mn opérations.

Dans le cas de matrices carrées d'ordre n, toutes les opérations BLAS2 ont une complexité en $O(n^2)$ et un accès mémoire en $O(n^2)$.

Il y a deux niveaux de boucle imbriqués, ce qui permet de construire deux variantes suivant l'ordre des boucles. On peut ainsi choisir un calcul par lignes ou par colonnes. On peut aussi définir une partition de la matrice et effectuer les opérations par blocs, pour optimiser l'accès hiérarchique à la mémoire.

Soit $Q = I - \frac{1}{v^T u} uv^T$ la matrice de projection de rang 1 définie précédemment. Il est à noter que

$$Qx = x - \frac{1}{v^T u} uv^T x = x - \frac{1}{v^T u}(v^T x)u.$$

Il est inutile et coûteux de calculer la matrice Q. En effet, le produit Qx est une opération matrice-vecteur de type BLAS2 et de complexité $O(nm)$, alors que l'expression ci-dessus n'utilise que des opérations vectorielles de type BLAS1 et de complexité $O(n)$ ou $O(m)$.

1.8.3 Opérations BLAS3

Ce niveau concerne les opérations entre matrices. Quelques exemples :

- produit de matrices $C = C + A * B$, $2mnp$ opérations.
- produit $C = C + AA^T$, $2mn^2$ opérations.

Dans le cas de matrices carrées d'ordre n, toutes les opérations BLAS3 ont une complexité en $O(n^3)$, avec un accès mémoire en $O(n^2)$.

Les trois niveaux de boucle imbriqués permettent de définir six variantes suivant l'ordre des boucles. On peut choisir de parcourir chacune des matrices par lignes ou par colonnes. Comme dans le niveau BLAS2, on optimise l'accès à la mémoire en définissant une partition par blocs. De plus, comme l'algorithme fait plus d'opérations arithmétiques que d'accès aux données, on peut ré-utiliser des valeurs qui sont dans la mémoire cache. C'est pour cette raison que le niveau 3 est le plus performant et permet presque d'atteindre la performance maximale d'une machine.

1.8.4 Produit de matrices

Le produit de matrices est l'opération la plus utilisée dans BLAS3. L'analyse suivante montre comment organiser les calculs pour exploiter la hiérarchie de mémoires. On suppose que les calculs sont effectués sur un processeur qui possède une mémoire cache. Les matrices sont partitionnées par blocs et l'objectif est de déterminer la taille des blocs pour utiliser au mieux la mémoire cache. Cette étude est une adaptation au cas monoprocesseur de [17].

Soit M la taille du cache. Nous supposons que les matrices A, B et C sont respectivement de taille $n_1 \times n_2$, $n_2 \times n_3$ et $n_1 \times n_3$, et qu'elles ont été partitionnées en blocs de tailles respectives $m_1 \times m_2$, $m_2 \times m_3$ et $m_1 \times m_3$. On suppose $n_i = m_i * k_i$ pour tout $i = 1,\ 2,\ 3$. L'objet de l'étude est alors de trouver les valeurs m_i qui utilisent au mieux le cache, c'est-à-dire qui permettent le plus de réutilisation des données.

L'opération $C = A * B$ peut s'écrire par blocs

```
do i = 1, k1
  do k = 1, k2
    do j = 1, k3
      Cij := Cij + Aik * Bkj
    enddo
  enddo
enddo
```

Dans la boucle interne `j`, le bloc A_{ik} reste le même; on suppose donc qu'il réside dans le cache; sa taille est $m_1 m_2$. Cela entraîne la première contrainte :

$$m_1 m_2 \leq M$$

Il est évident que les blocs sont plus petits que les matrices, ce que l'on traduit par $1 \leq m_i \leq n_i$ pour $i = 1, 2, 3$. On a ainsi obtenu l'ensemble des contraintes sous lesquelles on doit minimiser les mouvements de données entre la mémoire et le cache.

Si on calcule le nombre de lectures nécessaires pour avoir en cache les variables nécessaires au produit, on trouve que la matrice A est lue une fois, la matrice B l'est k_1 fois et la matrice C l'est k_2 fois. Au total, le nombre des lectures est:

$$L = n_1 n_2 + n_1 n_2 n_3 (\frac{1}{m_1} + \frac{1}{m_2})$$

Il reste donc à trouver les valeurs m_1 et m_2 qui minimisent $\frac{1}{m_1} + \frac{1}{m_2}$ sous les contraintes précédentes. On en arrive à la politique suivante (le choix de m_3 est sans importance):

1. si $n_2 n_1 \leq M$ alors $m_1 = n_1$ et $m_2 = n_2$;
2. sinon si $n_2 \leq \sqrt{M}$ alors $m_1 = \frac{M}{n_2}$ et $m_2 = n_2$;
3. sinon si $n_1 \leq \sqrt{M}$ alors $m_1 = n_1$ et $m_2 = \frac{M}{n_1}$
4. sinon $m_1 = \sqrt{M}$ et $m_2 = \sqrt{M}$.

1.8.5 bibliothèque LAPACK

La bibliothèqe LAPACK regroupe la plupart des algorithmes d'algèbre linéaire. Elle contient toutes les fonctions pour résoudre les systèmes linéaires, les problèmes aux moindres carrés, la décomposition aux valeurs singulières, les problèmes de valeurs propres.

LAPACK est la meilleure référence pour l'algèbre linéaire sur matrices stockées sous format plein ou sous format bande. Elle est disponible sur le site NETLIB (http://www.netlib.org) et le manuel d'utilisation est édité [1].

LAPACK utilise une partition par blocs des matrices, de façon à exploiter les performances des opérations BLAS3. D'autre part, les algorithmes sont robustes vis-à-vis des erreurs d'arrondi et il est possible d'estimer la sensibilité aux variations des données (voir chapitre sur l'arithmétique flottante).

Les chapitres suivants décriront divers exemples d'algorithmes de la bibliothèque LAPACK.

Chapter 2

Précision

2.1 Erreurs de calcul

2.1.1 Sources d'erreur

Résoudre un problème signifie trouver une solution x qui dépend de données a et de formules. Ces formules sont issues en général d'une modélisation d'un problème scientifique (en physique, chimie, biologie, économie, etc). Nous ne discutons pas ici de l'erreur de modélisation qui est l'écart entre le phénomène réel et le phénomène simulé. Dans la suite, les variables x et a appartiennent à des espaces vectoriels normés et il existe une fonction F telle que le problème s'écrive

$$F(x, a) = 0.$$

En général, il n'est possible de résoudre qu'un problème approché. Ce problème discret, formulé en dimension finie, doit être résolu par un algorithme, qui peut être direct ou itératif. Enfin, l'algorithme est exécuté sur un ordinateur avec une précision finie. Dans le résultat d'un calcul scientifique, il apparaît ainsi plusieurs sources d'erreur :

- l'erreur due à l'approximation des données,
- l'erreur d'approximation ou de discrétisation,
- l'erreur due à l'algorithme s'il est itératif,
- l'erreur d'arrondi due à la précision finie.

Les différentes sources d'erreur interfèrent avec les erreurs d'arrondi lors d'un calcul. Par exemple, la résolution d'un problème très sensible aux variations sur les données donne lieu à des calculs avec de grandes erreurs d'arrondi.

2.1.2 Mesures de l'erreur

Tout calcul devrait s'accompagner d'une estimation des erreurs d'approximation commises. Pour mesurer celles-ci, nous définissons les erreurs absolues et relatives, ainsi que la notion de chiffres significatifs.

Définition 2.1.1 *Soit $x \in \mathbb{R}$ et $\tilde{x} \in \mathbb{R}$ une approximation de x. L'erreur absolue sur $\tilde{x}$ est $|x - \tilde{x}|$. Si $x \neq 0$, l'erreur relative est $|x - \tilde{x}|/|x|$.*

Définition 2.1.2 *Le nombre de chiffres significatifs de x est le nombre de chiffres à partir du premier chiffre non nul.*

Par exemple, le nombre de chiffres significatifs de $x = 0.0034560$ est 5. L'erreur relative sur $\tilde{x} = 0.00346$ vaut environ 1.16 10^{-3} et le nombre de chiffres significatifs exacts est ici 2.

Dans le cas de vecteurs, on définit les erreurs sur les normes ou les erreurs composante par composante.

Définition 2.1.3 *Soit $x \in \mathbb{R}^n$, $\tilde{x} \in \mathbb{R}^n$ une approximation de x et $\|.\|$ une norme définie sur $\mathbb{R}^n$. L'erreur absolue sur la norme est $\|x - \tilde{x}\|$.*
Si $x \neq 0$, l'erreur relative sur la norme est $\|x - \tilde{x}\|/\|x\|$.
Si $x_i \neq 0,\ i = 1, \ldots, n$, l'erreur relative composante par composante est $\max_{1 \leq i \leq n} |x_i - \tilde{x}_i|/|x_i|$.

2.2 Arithmétique flottante

Avant d'analyser l'impact des erreurs d'arrondi et des différentes sources d'erreur sur des calculs, nous allons détailler les caractéristiques arithmétiques d'un ordinateur. L'arithmétique flottante est définie en détails dans plusieurs ouvrages, citons par exemple [3, 8, 9, 10, 22, 25].

Un logiciel de calcul scientifique utilise plusieurs types de variables, qui correspondent à un codage spécifique. Les types arithmétiques les plus usuels sont le type *entier*, le type *réel* et le type *complexe*. Un complexe est formé de deux réels, la partie réelle et la partie imaginaire. Un réel est codé par un nombre flottant.

2.2.1 Format flottant

Définition 2.2.1 *Un système flottant est défini par une base b, un exposant minimal e_{min}, un exposant maximal e_{max}, un nombre de chiffres p. L'ensemble des nombres flottants normalisés est noté $\mathbb{F}$. Un nombre flottant non nul normalisé est*

$$x = (-1)^s m b^e$$

où $s \in \{0, 1\}$ est le bit de signe, l'exposant e est un entier tel que $e_{min} \leq e \leq e_{max}$, la mantisse m vérifie $1 \leq m < b$ et s'écrit

$$\left\{ \begin{array}{l} m = a_0 + a_1 * b^{-1} + a_2 * b^{-2} + \ldots + a_p * b^{-p} \\ avec \ \ 0 \leq a_i \leq b - 1 \ i = 0, ..p \quad et \ \ a_0 \neq 0 \end{array} \right.$$

Par convention, le nombre 0 a un exposant et une mantisse nuls.

Proposition 2.2.1 *L'ensemble $\mathbb{F}$ est symétrique par rapport à 0.*
Le plus grand nombre de $\mathbb{F}$ vaut

$$x_{max} = b^{e_{max}}(b - b^{-p})$$

et le plus petit nombre strictement positif vaut

$$u = b^{e_{min}}$$

de sorte que

$$\mathbb{F} \subset \mathbb{R}_{\mathbb{F}} = [-x_{max}, -u] \cup \{0\} \cup [u, x_{max}].$$

L'ensemble des nombres entiers de l'intervalle $[-x_{max}, x_{max}]$ est inclus dans $\mathbb{F}$.

Remarque 2.2.1 *Nous ne définissons pas ici les nombres dénormalisés qui sont des nombres flottants dans l'intervalle* $]-u, u[$.

L'écart minimal entre deux mantisses consécutives est b^{-p}. Ce nombre caractérise la précision du système flottant.

Définition 2.2.2 *Le nombre* $\epsilon = b^{-p}$ *est appelé la précision du système flottant.*

La proposition suivante détermine les deux nombres flottants qui encadrent un nombre réel.

Proposition 2.2.2 *Tout réel de* $\mathbb{R}_{\mathbb{F}}$ *est encadré par deux flottants consécutifs. Soit*

$$x^- = \max\{y \in \mathbb{F},\ y \leq x\} \text{ et } x^+ = \min\{y \in \mathbb{F},\ y \geq x\}.$$

Il n'existe pas de flottant entre x^- *et* x^+ *et* $0 \leq x^+ - x^- \leq |x|\epsilon$.

Preuve. Si $x \in \mathbb{F}, x^- = x^+ = x$.

Soit $x \in \mathbb{R}_{\mathbb{F}},\ x \notin \mathbb{F},\ x > u$, soit $x^- = b^e m$ alors $m \geq 1,\ x^- < x$ et $x^+ = b^e(m+\epsilon) = x^- + b^e\epsilon$. Donc $0 \leq x^+ - x^- = b^e\epsilon \leq b^e m\epsilon \leq x\epsilon$.

Le cas $x < u$ se traite de la même façon. ◇

On peut remarquer que le nombre flottant qui précède l'entier 1 vaut $1 - \epsilon/b$ et celui qui le suit vaut $1 + \epsilon$.

2.2.2 Arrondis

Définition 2.2.3 *Un arrondi est une application* fl *de* $\mathbb{R}_{\mathbb{F}}$ *dans* $\mathbb{F}$ *vérifiant les propriétés de monotonie et de projection suivantes.*

$$\begin{array}{l} \forall x, y \in \mathbb{R}_{\mathbb{F}},\quad x \leq y \Rightarrow fl(x) \leq fl(y), \\ \forall x \in \mathbb{F},\quad fl(x) = x. \end{array}$$

Tout arrondi est défini à l'aide des deux flottants x^+ et x^-.

Proposition 2.2.3 *Tout arrondi* fl *vérifie*

$$\begin{array}{l} \forall x \in \mathbb{R}_{\mathbb{F}},\quad fl(x) = x^+ \text{ ou } fl(x) = x^- \\ \forall x \in \mathbb{R}_{\mathbb{F}},\quad |fl(x) - x| \leq \epsilon|x| \end{array}$$

Preuve. Par définition, $x^- \leq x \leq x^+$, d'où par monotonie et par projection $x^- \leq fl(x) \leq x^+$. Or il n'existe aucun flottant strictement compris entre x^+ et x^- donc $fl(x)$ est égal à l'un des deux.

la deuxième partie se déduit de la proposition 2.2.2. ◇

2.2.3 Opérations arithmétiques

L'ensemble $\mathbb{F}$ est non clos pour les opérations arithmétiques de $\mathbb{R}$ et il faut donc définir le résultat flottant d'une opération flottante. Soit fl l'arrondi choisi.

Définition 2.2.4 *Soit $\cdot$ l'une des 4 opérations $\{+,-,\times,/\}$ dans $\mathbb{R}$. L'opération flottante correspondante $\odot$ est correcte pour l'arrondi fl si elle satisfait la propriété*

$$\forall x, y \in \mathbb{F} \text{ tels que } x \cdot y \in \mathbb{R}_{\mathbb{F}}, \quad x \odot y = fl(x \cdot y), \tag{2.1}$$

où fl désigne un mode d'arrondi. Autrement dit, le résultat flottant est l'arrondi du résultat exact, s'il n'y a pas de dépassement de capacité.

Proposition 2.2.4 *Si l'opération $\odot$ est correcte, alors*

$$\forall x, y \in \mathbb{F} \text{ tels que } x \cdot y \in \mathbb{R}_{\mathbb{F}}, \quad x \odot y = (x \cdot y)(1+\alpha) \text{ avec } |\alpha| \leq \epsilon.$$

Proposition 2.2.5 *Les propriétés des opérations sur les nombres réels ne sont pas toutes vérifiées avec les nombres flottants.*

- *Les opérations flottantes sont commutatives,*
- *elles ne sont pas associatives,*
- *elles ne sont pas distributives,*
- 0 *est élément neutre de l'addition flottante et* 1 *est élément neutre de la multiplication flottante,*
- *l'opposé de $x \in \mathbb{F}$ existe et vaut $-x$,*
- *par contre $x \in \mathbb{F}$ n'a pas toujours d'inverse dans $\mathbb{F}$.*

Par exemple, avec Matlab, $49 \times (1/49) = 1 - \epsilon/2$.

2.2.4 Exceptions

Il peut arriver que le résultat exact d'une opération ne soit pas dans $\mathbb{R}_{\mathbb{F}}$. Pour fermer le système arithmétique flottant, l'ensemble $\mathbb{F}$ est doté de nombres spéciaux, notés $+\infty, -\infty, NaN$ (Plus l'infini, Moins l'infini, Not a Number). Le nombre 0 a un signe. Les opérations sont définies avec ces nombres spéciaux. Une opération ayant pour résultat un des nombres spéciaux déclenche une exception.

Définition 2.2.5 *Soit $z \in \mathbb{R}$ le résultat exact d'une opération arithmétique entre deux flottants x et y ; il y a overflow si $|z| > x_{max}$ et underflow si $|z| < u$. Le résultat flottant est respectivement ∞ et 0 avec le signe adéquat. L'opération est invalide si le résultat ne peut être ni un flottant, ni l'infini. Le résultat flottant d'une opération invalide est NaN.*

Remarque 2.2.2 *Les nombres dénormalisés permettent de mettre en place un underflow graduel vers* 0.

Par exemple, les opérations $\frac{0}{0}$ et $(+\infty - \infty)$ sont invalides, avec comme résultat NaN. L'opération $\frac{1}{0}$ déclenche un overflow avec comme résultat $\pm\infty$ et l'opération $\frac{1}{\infty}$ déclenche un underflow avec comme résultat 0.

Type	Base b	Mantisse p	Exposant e_{min}	Exposant e_{max}
Simple	2	24	-126	+127
Double	2	53	-1022	+1023

Table 2.1: Formats simple et double précision de la norme IEEE-754

Type	précision ϵ	Seuil d'overflow x_{max}	Seuil d'underflow u
Simple	10^{-7}	10^{+38}	10^{-38}
Double	10^{-16}	10^{+308}	10^{-308}

Table 2.2: Valeurs caractéristiques approchées des formats simple et double précision de la norme IEEE-754

2.2.5 Norme IEEE-754

La norme IEEE-754 [39], publiée en 1985, spécifie un système flottant et l'arithmétique flottante associée. La plupart des constructeurs respectent aujourd'hui cette norme, ce qui facilite grandement le transfert de logiciels entre machines. Cette norme permet aussi d'établir des preuves sur le comportement des erreurs d'arrondi dans un algorithme.

La norme IEEE-754 spécifie deux formats, appelés simple et double précision, qui sont résumés dans les tables 2.2.5 et 2.2.5. La norme définit 4 arrondis.

Définition 2.2.6 *L'arrondi vers moins l'infini de x vaut x^-,*
l'arrondi vers plus l'infini de x vaut x^+,
l'arrondi vers zéro ou par troncature de x vaut x^+ si $x < 0$ et x^- si $x > 0$,
l'arrondi au plus près de x vaut x^- si $x - x^- < x^+ - x$, il vaut x^+ sinon. Dans le cas d'égalité, une règle de parité fixe le choix.

Remarque 2.2.3 *L'arrondi par défaut est l'arrondi au plus près, noté $\tilde{x}$ dans ce qui suit. Il vérifie*

$$|x - \tilde{x}| \leq |x|\epsilon/2.$$

Les 4 opérations arithmétiques $\{+, -, \times, /\}$ ainsi que la racine carrée et l'opération de congruence (remainder) sont correctes.

Les exceptions overflow, underflow, invalide, inexact ainsi que les nombres spéciaux et les opérations entre nombres spéciaux sont également spécifiés par la norme. Les opérations continuent après une exception. La norme utilise les nombres dénormalisés et l'underflow graduel.

2.2.6 Phénomène d'absorption

L'absorption se produit lors de l'addition de deux quantités avec des ordres de grandeur très différents.

Proposition 2.2.6 *Soit $x, y \in \mathbb{F}$, tels que $u < |x| < x_{max}$ et $x + y \in \mathbb{R}_{\mathbb{F}}$. Soit $x = b^e m$ avec $m \geq 1 + \epsilon$. Soit $\oplus$ l'addition flottante, avec arrondi au plus près. Alors*

$$x \oplus y \;=\; x \quad \Leftrightarrow \quad |y| \leq b^e \;\epsilon/2$$

Preuve. Le résultat est immédiat par définition de l'arrondi. ◇

Ce résultat conduit à la définition suivante.

Définition 2.2.7 *Soit deux réels x et y dans $\mathbb{R}_{\mathbb{F}}$. Il y a absorption de y par x si*

$$|y| \leq |x| \ \epsilon/2.$$

Le résultat d'une addition après absorption est en général le plus grand des deux opérandes.

Par exemple, en double précision, le résultat flottant de $1 + 10^{-16}$ est 1 donc le résultat flottant de $(1+10^{-16})-1$ vaut 0. Ici, le phénomène d'absorption est associé à un phénomène de cancellation, décrit ci-dessous.

Un exemple classique d'absorption se produit lors du calcul de la somme $S_n = \sum_{i=1}^{n} 1/i$ par l'algorithme avec indices croissants $S_{n+1} = S_n + 1/(n+1)$, si bien que cette série converge numériquement avec l'arithmétique flottante. Un changement d'algorithme avec un ordre de sommation différent permet de retrouver la divergence de la série. Pour plus de détails, voir par exemple [22].

2.2.7 Phénomène de cancellation

Ce phénomène se produit lors de la soustraction de deux nombres voisins. Il faut alors renormaliser le résultat car les chiffres de gauche s'éliminent (d'où le nom cancellation). On dit aussi élimination. Les chiffres de droite deviennent les premiers chiffres significatifs. Si les opérandes sont exacts, la cancellation est bénigne car la soustraction est en général exacte. Mais si les opérandes sont entachés d'erreur, l'ordre de grandeur du résultat exact est celui des incertitudes, donc le résultat flottant risque de n'avoir aucun sens ; la cancellation est dite catastrophique. Il faut noter que la cancellation est un révélateur des erreurs précédentes.

Définition 2.2.8 *Soit x et y deux réels dans $\mathbb{R}_{\mathbb{F}}$. Il y a élimination de x et y si*

$$0 < |x - y| \leq \epsilon/2 \ (|x| + |y|)$$

Soit $x(1+\alpha)$ et $y(1+\beta)$ deux approximations de x et y et soit E l'erreur relative commise sur la soustraction $x - y$. Alors

$$E = \frac{|x\alpha - y\beta|}{|x - y|} \leq F, \text{ avec } F = \max\{|\alpha|, |\beta|\}\frac{|x| + |y|}{|x - y|}.$$

Dans le cas de cancellation, $F \geq 2\max\{|\alpha|, |\beta|\}/\epsilon$ donc l'erreur relative E risque d'être supérieure à 1.

Par exemple, en base 10, $3.1451 - 3.1449 = 0.0002 = 210^{-4}$ mais, si l'on arrondit les opérandes à deux chiffres après la virgule, on obtient $3.15 - 3.14 = 0.01 = 110^{-2}$ soit un résultat avec deux ordres de grandeur en trop.

De manière générale, il y a cancellation catastrophique entre plusieurs nombres si la somme de ces nombres est beaucoup plus petite que la somme de leurs valeurs absolues, autrement dit si

$$0 < |\sum x_i| \leq \epsilon/2 \ (\sum |x_i|)$$

2.2.8 Extensions de la norme IEEE

Le calcul précis des fonctions mathématiques usuelles telles que l'exponentielle n'est toujours pas complètement résolu [29]. Il n'existe toujours pas de norme à ce sujet, qui reste un domaine de recherche.

La norme IEEE définit la simple et la double précision. En fait, de nombreux ordinateurs proposent aussi une précision étendue pour les calculs intermédiaires grâce à des registres de 80 bits. L'analyse d'une approche hybride utilisant différentes précisions lors des calcul reste aussi un sujet de recherche [25].

Parfois, il est nécessaire de garantir une très grande précision, c'est pourquoi il existe des bibliothèques d'arithmétique dite multi-précision, où la précision des calculs peut être arbitrairement grande [3]. Néanmoins, la précision reste finie et des erreurs d'arrondi sont inéluctables. La précision infinie est apportée par le calcul formel tant qu'il manipule des symboles et des formules. Mais l'évaluation d'une formule se fait de nouveau en précision finie. Il est important de comprendre les interactions entre calcul symbolique et calcul numérique [3].

Une solution sûre pour garantir la précision d'un résultat est de l'inclure dans un intervalle. L'arithmétique d'intervalles permet de réaliser ces encadrements. Afin de réduire la largeur de l'intervalle, il est en général nécessaire d'adapter les méthodes de calcul. Voir par exemple [3, 5].

2.3 Stabilité des problèmes

Nous allons maintenant étudier l'une après l'autre les différentes sources d'erreur. La théorie de la perturbation étudie l'impact de variations sur les données. De nombreux ouvrages traitent de cette sensibilité aux données, voir par exemple [22, 18, 40, 8, 3].

La définition d'un problème bien posé ci-dessous est celle de Hadamard.

Définition 2.3.1 *Un problème bien posé au voisinage de a possède une solution unique continue par rapport aux donnéees dans un voisinage de a. Il est singulier sinon.*

Définition 2.3.2 *Soit $F(x,a) = 0$ un problème bien posé. Soit x la solution associée à a et $x+\Delta x$ la solution associée à $a+\Delta a$. On suppose que $x \neq 0$. Le problème est stable si*

$$\kappa(a) = \overline{\lim_{\alpha \to 0}} \frac{1}{\alpha} \frac{\|\Delta x\|}{\|x\|} \text{ où } \|\Delta a\| \leq \alpha \|a\|$$

est fini. Dans ce cas, $\kappa(a)$ est le conditionnement relatif du problème au point a.

Au voisinage d'une donnée ou d'une solution nulle, on utilise un conditionnement absolu. Le conditionnement d'un problème singulier est infini. Un problème très mal conditionné, c'est-à-dire avec un grand conditionnement, est en général proche de la singularité.

La définition du conditionnement est liée au choix des métriques dans l'espace des données et dans l'espace des solutions.

Pour α suffisamment petit, on a

$$\|\Delta x\| \leq \kappa(a)\ \|x\|\ \alpha + O(\alpha^2). \tag{2.2}$$

2.3.1 Lien avec le résidu

Considérons le problème $F(x) = a$ de solution x et soit y une solution approchée. Le résidu $r = a - F(y)$ peut être considéré comme une perturbation de a puisque $F(y) = a - r$. Si le problème a un conditionnement $\kappa(a)$, il vient, pour $\|r\|$ assez petit,

$$\|y - x\| \leq \kappa(a)\ \|x\|\ \|r\|/\|a\| + O(\|r\|^2).$$

2.4 Stabilité des algorithmes directs

Nous nous plaçons ici dans le cadre des algorithmes directs et nous étudions l'impact des erreurs d'arrondi sur le résultat d'un calcul scientifique. Plus précisément, nous supposons que la solution exacte x du problème $F(x, a) = 0$ peut être calculée en un nombre fini d'opérations arithmétiques. Le cas de l'impact des arrondis sur les algorithmes itératifs est plus complexe. Voir par exemple [22, 8].

Nous voulons mesurer l'effet des erreurs d'arrondi. Pour cela, nous supposons que les opérations arithmétiques usuelles sont correctes, par exemple que l'arithmétique flottante satisfait la norme IEEE.

Soit $x(\epsilon)$ la solution calculée en arithmétique flottante avec une précision ϵ. Il existe deux façons de mesurer la qualité du résultat : une analyse directe et une analyse inverse.

Définition 2.4.1 *Un algorithme est stable au sens direct s'il existe $\psi(a)$ tel que*

$$\|x(\epsilon) - x\|/\|x\| \leq \psi(a)\ \epsilon.$$

Un algorithme est stable au sens inverse s'il existe $\phi(a)$ et $a(\epsilon)$ tels que

$$F(x(\epsilon), a(\epsilon)) = 0,\ \textit{avec}\ \|a(\epsilon) - a\|/\|a\| \leq \phi(a)\epsilon.$$

Proposition 2.4.1 *Si le problème est stable et si l'algorithme de résolution est stable au sens inverse avec $\phi(a)$ assez petit, alors l'algorithme est stable au sens direct et*

$$\|x(\epsilon) - x\|/\|x\| \leq \kappa(a)\ \phi(a)\ \epsilon + O(\epsilon^2). \tag{2.3}$$

Preuve. La formule se déduit immédiatement du résultat (2.2). ◇

La formule précédente montre qu'en général la précision relative sur un résultat de calcul est la précision machine multipliée par le conditionnement du problème à résoudre. Le résultat risque donc d'être d'autant plus imprécis que le problème est mal conditionné. Les erreurs d'arrondi sont amplifiées sous l'effet du conditionnement.

2.4.1 Exemple : produit scalaire

Soit $x \in \mathbb{R}^n$ et $y \in \mathbb{R}^n$ deux vecteurs non nuls et $s = x^T y$ leur produit scalaire.

Proposition 2.4.2 *Le conditionnement relatif composante par composante du produit scalaire vaut*

$$\kappa(x, y) = 2|x|^T|y|/|x^T y|.$$

Preuve. Soit $s + \Delta s = (x + \Delta x)^T(y + \Delta y)$, avec $|\Delta x| \leq \alpha|x|$ et $|\Delta y| \leq \alpha|y|$ alors $\Delta s = x^T\Delta y + y^T\Delta x + \Delta x^T\Delta y$ et $|\Delta s| \leq 2|x|^T|y|\alpha + |x|^T|y|\alpha^2$ d'où le résultat. $\diamond$

Nous considérons l'algorithme suivant, écrit en syntaxe Matlab, qui fixe l'ordre des opérations :

ALGORITHM : Produit scalaire

```
s=0 ;
for i=1:n,
    s = s+x(i) * y(i) ;
end ;
```

Pour démontrer la stabilité numérique de cet algorithme, nous avons besoin d'un lemme très souvent utilisé en analyse d'erreur.

Lemme 2.4.1 *Si $n\epsilon < 0.1$ et si $|\delta_i| \leq \epsilon$ alors*

$$\Pi_{i=1}^{n}(1 + \delta_i) = 1 + \theta \text{ avec } \theta \leq 1.06\ n\epsilon$$

Proposition 2.4.3 *Si $n\epsilon < 0.1$, l'algorithme est stable au sens inverse et on a*

$$fl(x^T y) = x^T(y + \Delta y)$$
$$|\Delta y| \leq 1.06\ n\epsilon|y|.$$

Preuve. Soit $s_i = s_{i-1} + x_i * y_i$ pour $i \geq 2$ et $s_1 = x_1 * y_1$. Alors

$fl(s_1) = x_1 * y_1(1 + \alpha_1)$,
$fl(s_i) = (fl(s_{i-1}) + x_i * y_i(1 + \alpha_i))(1 + \beta_i)$,
avec $|\alpha_i| \leq \epsilon$ et $|\beta_i| \leq \epsilon$,
d'où par récurrence
$fl(s_n) = x_1 * y_1(1 + \alpha_1)(1 + \beta_2)\ldots(1 + \beta_n)+$
$x_2 * y_2(1 + \alpha_2)(1 + \beta_2)\ldots(1 + \beta_n)+$
$x_n * y_n(1 + \alpha)_n)(1 + \beta_n)$
et par application du lemme,
$fl(s) = fl(s_n) = x_1 * y_1(1 + \theta_1) + \ldots + x_n * y_n(1 + \theta_n)$ avec $|\theta_i| \leq 1.06n\epsilon$.

$\diamond$

Une preuve similaire peut être faite pour tout ordre de sommation, avec une constante différente.

Les erreurs d'arrondi sont donc de l'ordre du conditionnement multiplié par la précision machine. Lorsque $|x^T y| << |x|^T|y|$, le conditionnement est élevé. Ce très grand conditionnement se manifeste par le biais d'un phénomène de cancellation globale dans le calcul du produit scalaire. Cette cancellation globale amplifie les erreurs d'arrondi.

2.4.2 Exemple : produit extérieur

Définition 2.4.2 *Soit $x \in \mathbb{R}^n$ et $y \in \mathbb{R}^n$ deux vecteurs non nuls. Leur produit extérieur est $A = xy^T$.*

Proposition 2.4.4 *Le conditionnement relatif composante par composante du produit extérieur vaut*

$$\kappa(x, y) = 2|x||y|^T/|xy^T|.$$

Proposition 2.4.5 *Le calcul du produit extérieur n'est pas stable au sens inverse. Il est stable au sens direct (composante par composante) et*

$$fl(xy^T) = xy^T + \Delta A \text{ avec } |\Delta A| \leq \epsilon |x||y|^T.$$

Preuve. Le résultat flottant $\tilde{A} = fl(xy^T)$ n'est pas en général une matrice de rang 1 donc ne peut être égal à $(x + \Delta x)(y + \Delta y)^T$. Par contre, $\tilde{A}_{ij} = x_i y_j(1 + \alpha_{ij})$ avec $|\alpha_{ij}| \leq \epsilon$ donc $\tilde{A} = A + \Delta A$ avec $|\Delta A|_{ij} \leq |x_i||y_j|\epsilon$. ◇

Chapter 3

Valeurs propres et valeurs singulières

3.1 Valeurs propres et vecteurs propres

Définition 3.1.1 *Le nombre complexe λ est valeur propre de la matrice carrée A si la matrice $A - \lambda I$ est singulière, autrement dit s'il existe un vecteur x non nul tel que $Ax = \lambda x$.*

Proposition 3.1.1 *Toute matrice carrée A d'ordre n a n valeurs propres complexes, qui sont les racines du polynôme caractéristique $det(A - \lambda I)$.*

On a $det(A) = \lambda_1 \dots \lambda_n$ et $tr(A) = \lambda_1 + \dots + \lambda_n$.

Définition 3.1.2 *La multiplicité algébrique d'une valeur propre est sa multiplicité dans le polynôme caractéristique. Une valeur propre est simple si sa multiplicité algébrique est 1. Sinon, c'est une valeur propre multiple.*

La multiplicité géométrique d'une valeur propre est la dimension du sous-espace vectoriel engendré par les vecteurs propres associés.

Si la multiplicité géométrique est strictement inférieure à la multiplicité algébrique, la valeur propre est défective.

Toute valeur propre simple est non défective.

Définition 3.1.3 *Une matrice est diagonalisable si et seulement si toutes ses valeurs propres sont non défectives.*

De manière équivalente, une matrice est diagonalisable si et seulement s'il existe une matrice inversible complexe V et une matrice diagonale complexe $D = diag(\lambda_i)$ telle que

$$AV = VD,\ A = VDV^{-1}.$$

V est la base des vecteurs propres et les éléments de D sont les valeurs propres.

Théorème 3.1.1 *Toute matrice symétrique a des valeurs propres réelles et des vecteurs propres réels.*

Elle est diagonalisable.

De plus, les vecteurs propres forment une base orthonormée.

3.2 Valeurs singulières et vecteurs singuliers

Le cas des matrices carrées symétriques est intéressant : les calculs restent réels, il existe un changement de base orthonormée tel que la matrice devienne diagonale.

Dans le cas carré non symétrique, il faut utiliser les nombres complexes, la matrice peut être non diagonalisable et de plus, la base des vecteurs propres n'est pas toujours orthonormée.

Dans le cas des matrices rectangulaires, il faut définir des bases pour l'espace de départ et pour l'espace d'arrivée. La Décomposition en Valeurs Singulières (SVD : Singular Value Decomposition) garantit l'existence de bases orthonormées dans chacun de ces espaces telles que la matrice devienne "diagonale".

Théorème 3.2.1 *Soit $A \in \mathbb{R}^{m\times n}$ et $p = \min(m,n)$.*
Il existe $U \in \mathbb{R}^{m\times m}$ orthogonale, il existe $V \in \mathbb{R}^{n\times n}$ orthogonale, il existe $\sigma_1 \geq \sigma_2 \geq \sigma_p \geq 0$, $\Sigma_1 = diag(\sigma_1, \sigma_2, \sigma_p) \in \mathbb{R}^{p\times p}$ matrice diagonale, tels que

$$\begin{array}{l} si\ n \geq m,\ \Sigma = \begin{pmatrix} \Sigma_1 & 0 \end{pmatrix} \\ si\ m \geq n, \Sigma = \begin{pmatrix} \Sigma_1 \\ 0 \end{pmatrix} \end{array}$$

et

$$A = U\Sigma V^T \tag{3.1}$$

Les valeurs singulières σ_i sont uniques.
Si les valeurs singulières sont toutes distinctes, les vecteurs singuliers à gauche u_i et à droite v_i sont uniques.
$\sigma_1 = \|A\|_2 = \max_{\|x\|_2=1} \|Ax\|_2$.

Preuve. Si $A = U\Sigma V^T$ alors $\|A\|_2 = \|\Sigma\|_2 = \sigma_1$. Donc si la décomposition existe, σ_1 est unique. On va montrer l'existence de la décomposition.

Soit $\sigma_1 = \|A\|_2$ et v_1 tel que $\|v_1\|_2 = 1, \|u_1\|_2 = 1, Av_1 = \sigma_1 u_1$.
Soit $U_1 = (u_1, X)$ et $V_1 = (v_1, Y)$ deux matrices orthogonales d'ordre m et n (théorème de la base incomplète).

$$U_1^T A V_1 = \begin{pmatrix} u_1^T \\ X^T \end{pmatrix} A \begin{pmatrix} v_1 & Y \end{pmatrix} = \begin{pmatrix} u_1^T A v_1 & u_1^T AY \\ X^T A v_1 & X^T AY \end{pmatrix} = \begin{pmatrix} \sigma_1 & w^T \\ 0 & B \end{pmatrix}$$

On va montrer que $w = 0$.
On a $\|U_1^T A V_1\|_2 = \|A\|_2 = \sigma_1$.
Soit $x = \begin{pmatrix} \sigma_1 \\ w \end{pmatrix}$ et $y = (U_1^T A V_1)x$. Alors

$$\|x\|_2^2 = \sigma_1^2 + w^T w \geq \sigma_1^2 \text{ et } \|y\|_2 \leq \|U_1^T A V_1\|_2 \|x\|_2 = \sigma_1 \|x\|_2$$

D'autre part,

$$y = \begin{pmatrix} \sigma_1 & w^T \\ 0 & B \end{pmatrix} \begin{pmatrix} \sigma_1 \\ w \end{pmatrix} = \begin{pmatrix} \sigma_1^2 + w^T w \\ Bw \end{pmatrix}$$

donc $\|y\|_2 \geq \sigma_1^2 + w^T w = \|x\|_2^2$ d'où $\|x\|_2 \leq \sigma_1$ et $w = 0$.
Donc

$$U_1^T A V_1 = \begin{pmatrix} \sigma_1 & 0 \\ 0 & B \end{pmatrix}$$

Nous allons maintenant faire une preuve par récurrence.
Si $A = u \in \mathbb{R}^{p\times 1}$ ou $A = v^T \in \mathbb{R}^{1\times p}$ alors la décomposition existe.
On a $B \in \mathbb{R}^{(m-1)\times(n-1)}$. Supposons que $B = U_2 \Sigma_2 V_2^T$. Soit

$$U = U_1 \begin{pmatrix} 1 & 0 \\ 0 & U_2 \end{pmatrix},\ V = V_1 \begin{pmatrix} 1 & 0 \\ 0 & V_2 \end{pmatrix},\ \Sigma = \begin{pmatrix} \sigma_1 & 0 \\ 0 & \Sigma_2 \end{pmatrix}$$

alors U et V sont des matrices orthogonales et

$$A = U \Sigma V^T$$

On a $\sigma_2 = \|B\|_2 \leq \sigma_1$.
Il reste à montrer l'unicité des vecteurs singuliers. On l'admet ici. ◇

On a donc les égalités suivantes :

$$\begin{array}{l} \text{Si } n \leq m,\ p = n : \\ Av_i = \sigma_i u_i,\ i = 1, \ldots, n, \\ A^T u_i = \sigma_i v_i,\ i = 1, \ldots, n, \\ A^T u_i = 0,\ i = n+1, \ldots, m. \\ \text{Si } m \leq n,\ p = m : \\ Av_i = \sigma_i u_i,\ i = 1, \ldots, m, \\ Av_i = 0,\ i = m+1, \ldots, n, \\ A^T u_i = \sigma_i v_i,\ i = 1, \ldots, m. \end{array}$$

Ces égalités permettent de faire le lien avec les valeurs propres des matrices symétriques $A^T A$ et AA^T.

Proposition 3.2.1 *Les valeurs singulières de A (complétées par 0 si $m < n$), sont les racines carrées des valeurs propres de $A^T A$ et les vecteurs singuliers à droite de A sont les vecteurs propres associés de $A^T A$.*

Les valeurs singulières de A (complétées par 0 si $n < m$), sont les racines carrées des valeurs propres de AA^T et les vecteurs singuliers à gauche de A sont les vecteurs propres associés de AA^T.

Preuve. Soit $m \geq n$, le cas $m \leq n$ est similaire.
$A = U\Sigma V^T$ d'où $A^T A = V \Sigma_1^2 V^T$ et $A^T A V = V \Sigma_1^2$ donc σ_i^2 est valeur propre de $A^T A$.
$A^T A v_i = A^T(\sigma_i u_i) = \sigma_i^2 v_i,\ i = 1, \ldots, n.$ ◇

Proposition 3.2.2 *Si A est carrée symétrique, les valeurs singulières de A sont les valeurs absolues des valeurs propres de A. On en déduit que les valeurs propres d'une matrice symétrique sont des nombres réels et que les vecteurs propres d'une matrice symétrique forment une base orthonormée.*

Preuve. Si A est symétrique, $A^T A = A^2$ et les valeurs propres de A^2 sont celles de A au carré. ◇

Les résultats suivants permettent de caractériser le rang, l'image et le noyau de A et A^T.

Proposition 3.2.3 *Le rang de la matrice A est le nombre de valeurs singulières non nulles.*
Autrement dit, soit $A = U\Sigma V^T$ la SVD de A avec $\sigma_1 \geq \sigma_r > 0 = \sigma_{r+1} = \sigma_p$. Alors $rang(A) = r$.

Proposition 3.2.4 *Soit $r = rang(A)$. Soit U_1 et V_1 les r premiers vecteurs singuliers $U_1 = (u_1, \ldots, u_r)$ et $V_1 = (v_1, \ldots, v_r)$. Soit D la matrice diagonale $D = diag(\sigma_1 \ldots \sigma_r)$. Alors*

$$A = U_1 D V_1^T = \sum_{j=1}^{r} \sigma_j u_j v_j^T.$$

Cette décomposition s'appelle la SVD réduite de A.

Proposition 3.2.5 *Soit $U = (U_1\ U_2)$ avec $U_2 = (u_{r+1}, \ldots, u_m)$ et $V = (V_1\ V_2)$ avec $V_2 = (v_{r+1}, \ldots, v_n)$. Alors*
$Im(A) = Vect(U_1)$ et $ker(A) = Vect(V_2) = Vect(V_1)^\perp$,
$Im(A^T) = Vect(V_1)$ et $ker(A^T) = Vect(U_2) = Vect(U_1)^\perp$.

Preuve.

$$A = U\Sigma V^T = (U_1\ U_2) \begin{pmatrix} D & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} V_1^T \\ V_2^T \end{pmatrix} = (U_1\ U_2) \begin{pmatrix} DV_1^T \\ 0 \end{pmatrix} = U_1 D V_1^T$$

$$A = U_1 D V_1^T = (u_1 \ldots u_r) diag(\sigma_1 \ldots \sigma_r)(v_1 \ldots v_r)^T = (u_1 \ldots u_r)(\sigma_1 v_1 \ldots \sigma_r v_r)^T = \sum_{j=1}^{r} \sigma_j u_j v_j^T.$$

$y \in Im(A) \Leftrightarrow y = U_1(DV_1^T)x \Rightarrow y \in Vect(U_1)$
$y \in Vect(U_1) \Rightarrow y = U_1 z$, soit $x = V_1 D^{-1} z$ alors $Ax = U_1 D V_1^T V_1 D^{-1} z = U_1 z = y$ donc $y \in Im(A)$. Donc $Im(A) = Vect(U_1)$.

$x \in ker(A) \Leftrightarrow U_1 D V_1^T x = 0 \Leftrightarrow DV_1^T x = 0 \Leftrightarrow V_1^T x = 0 \Leftrightarrow x \perp V_1 \Leftrightarrow x \in Vect(V_2)$. Donc $ker(A) = Vect(V_2)$.

$Im(A^T) = ker(A)^\perp = V_2^\perp = Vect(V_1)$,
$ker(A^T) = Im(A)^\perp = U_1^\perp = Vect(U_2)$.

$rang(A) = dim(Vect(U_1)) = r$. ◇

3.3 Approximation de rang k et rang numérique

La SVD permet aussi de déterminer des approximations d'une matrice afin de simplifier la résolution d'un problème. Elle permet aussi de calculer le rang numérique d'une matrice.

Théorème 3.3.1 *Soit $r = rang(A)$ et $\sigma_1 \geq \sigma_r > 0 = \sigma_{r+1} = \sigma_p$ les valeurs singulières de A. Soit $k < r$ et soit*

$$A_k = \sum_{j=1}^{k} \sigma_j u_j v_j^T.$$

Alors $rang(A_k) = k$ et

$$\|A - A_k\|_2 = \min_{B, rang(B)=k} \|A - B\|_2 = \sigma_{k+1}.$$

Autrement dit, A_k est la meilleure approximation de A avec un rang k.

Preuve. Les valeurs singulières de A_k sont $\sigma_i, i = 1, \ldots, k$ et 0 donc $rang(A_k) = k$. On a

$$A - A_k = \sum_{j=k+1}^{r} \sigma_j u_j v_j^T, \text{ donc } \|A - A_k\|_2 = \sigma_{k+1}.$$

Soit B une matrice de rang k, alors $dim(ker(B)) = n-k$. Puisque $dim(Vect(v_1, \ldots, v_{k+1})) = k+1$, il existe $z \neq 0$ tel que $z \in ker(B) \cap Vect(v_1, \ldots, v_{k+1})$ donc tel que $Bz = 0$ et $z = \sum_{j=1}^{k+1} (v_j^T z) v_j$, soit $v_j^T z = 0,\ j > k+1$. Alors $(A-B)z = Az = \sum_{j=1}^{r} \sigma_j u_j v_j^T z = \sum_{j=1}^{k+1} \sigma_j u_j v_j^T z$ et $\|Az\|_2^2 = \sum_{j=1}^{k+1} \sigma_j^2 (v_j^T z)^2 \geq \sigma_{k+1}^2 \sum_{j=1}^{k+1} (v_j^T z)^2 = \sigma_{k+1}^2 \|z\|_2^2$. Donc

$$\|A - B\|_2 \geq \frac{\|(A-B)z\|_2}{\|z\|_2} \geq \sigma_{k+1}.$$

◇

Le rang d'une matrice est le nombre de valeurs singulières non nulles. Numériquement, il est difficile de déterminer si une très petite valeur calculée est nulle. On introduit la notion de rang numérique.

Définition 3.3.1 *Le rang numérique associé à la tolérance δ est l'entier k tel que*

$$k = \min(rang(B),\ \|A - B\|_2 \leq \delta).$$

Par application du théorème précédent, on peut caractériser k :

Proposition 3.3.1 *Le rang numérique de A vaut k si et seulement si*

$$\sigma_1 \geq \ldots \sigma_k > \delta \geq \sigma_{k+1} \geq \ldots \geq \sigma_n.$$

Preuve. Il suffit d'appliquer le théorème et la définition ci-dessus. ◇

En pratique, il peut être difficile de choisir correctement la tolérance δ.

3.4 Calcul de la SVD

Nous montrons d'abord que tout algorithme de calcul doit être itératif.

Théorème 3.4.1 *Pour $n \geq 5$, le calcul des valeurs singulières d'une matrice de $\mathbb{R}^{m \times n}$ doit se faire par un algorithme itératif.*

Preuve. Les carrés des valeurs singulières sont les valeurs propres de $B = A^T A \in \mathbb{R}^{n \times n}$ donc sont les racines du polynôme caractéristique de B de degré n. Or, d'après la théorie de Galois, il est impossible de calculer par formules les racines d'un polynôme de degré supérieur ou égal à cinq. ◇

Le calcul des valeurs singulières est finalement le calcul des valeurs propres de $B = A^T A$. Le calcul des valeurs propres d'une matrice symétrique se fait en deux temps : on effectue une transformation (algorithme direct) pour obtenir une matrice tridiagonale qui a les mêmes valeurs propres puis on calcule les valeurs propres (algorithme itératif) de la matrice tridiagonale.

Mais si l'on calcule explicitement B, le calcul des petites valeurs singulières devient très imprécis et l'algorithme est instable numériquement. Pour éviter cela, on choisit une autre méthode : on effectue une transformation (algorithme direct) sur A pour obtenir une matrice bidiagonale C qui a les mêmes valeurs singulières puis on calcule les valeurs singulières de cette matrice bidiagonale.

Il est à noter que la matrice $C^T C$ est tridiagonale et peut être obtenue par tridiagonalisation de B donc la méthode suit implicitement les étapes d'une méthode de calcul de valeurs propres mais sans former explicitement les matrices B et $C^T C$, ce qui permet de garantir la stabilité numérique.

3.5 Bidiagonalisation

Ici, on suppose $m \geq n$, dans le cas contraire on peut utiliser A^T.

Théorème 3.5.1 *Soit $A \in \mathbb{R}^{m\times n}$ avec $m \geq n$. Il existe deux matrices orthogonales $Q \in \mathbb{R}^{m\times m}$ et $P \in \mathbb{R}^{n\times n}$ telles que*

$$Q^T A P = \begin{pmatrix} C \\ 0 \end{pmatrix},$$

où $C \in \mathbb{R}^{n\times n}$ est une matrice bidiagonale.

Preuve. admis. ◇

L'algorithme de bidiagonalisation est codé dans une procédure de LAPACK qui fait appel à BLAS3. La complexité est $4n^2(m - n/3)$.

Proposition 3.5.1 *Les valeurs singulières de C sont celles de A.*

Preuve. Soit $C = UDV^T$ la SVD de C, alors

$$A = \begin{pmatrix} Q_1 U & Q_2 \end{pmatrix} \begin{pmatrix} D \\ 0 \end{pmatrix} (PV)^T$$

est une SVD de A. ◇

Les valeurs singulières de C sont calculées par une méthode itérative, qui n'est pas détaillée ici.

La complexité globale est résumée dans le tableau ci-dessous, en supposant qu'il faut $2n$ itérations pour calculer toutes les valeurs singulières. Différents cas sont considérés, suivant qu'il faut calculer ou non les vecteurs singuliers.

calcul	nombre de multiplications flottantes
Σ, U_1, V	$O(7mn^2 + 11n^3/3)$
Σ, U_1	$O(7mn^2 - n^3)$
Σ, V	$O(2mn^2 + 4n^3)$
Σ	$O(2mn^2 - 2n^3/3)$

Chapter 4

Orthogonalisation

4.1 Algorithmes de Gram-Schmidt

Théorème 4.1.1 *Soit $A \in \mathbb{R}^{m\times n}$ de plein rang avec $m \geq n$. Alors il existe un unique couple de matrices (Q, R) tel que :*

$$Q \in \mathbb{R}^{m\times n} \text{ et } R \in \mathbb{R}^{n\times n},$$
$$Q^T Q = I_n,$$
$$R \text{ est triangulaire supérieure avec } diag(R) > 0$$
$$A = QR$$

Suivant les applications, on est intéressé par la matrice Q (orthogonalisation des colonnes de A) ou par la matrice R (problème aux moindres carrés avec la matrice A).

Il existe deux types d'approche pour obtenir cette décomposition :

- en utilisant des projections orthogonales,
- en utilisant des transformations orthogonales.

Dans le cas de l'orthogonalisation, on procède très souvent avec des projections; c'est ce que nous étudions dans ce paragraphe.

Pour tout $k \leq n$, on note $A_k = [a_1; a_2; \cdots; a_k]$ et $Q_k = [q_1; q_2; \cdots; q_k]$ les matrices constituées respectivement des k premières colonnes des matrices A et Q. On note R_k la matrice principale d'ordre k de R. De la relation $A_{k+1} = Q_{k+1}R_{k+1}$ on déduit que la colonne q_{k+1} s'obtient à partir de la projection de a_{k+1} sur l'orthogonal du sous-espace engendré par q_1, q_2, $\cdots$ q_k, que l'on normalise ensuite.

Cette projection a pour matrice $L_k = I_m - Q_k {Q_k}^T$. Il existe deux façons équivalentes de l'écrire :

$$L_k = (I_m - q_1 {q_1}^T - \cdots - q_k {q_k}^T),$$
$$L_k = (I_m - q_k {q_k}^T) \cdots (I_m - q_1 {q_1}^T).$$

La première écriture conduit à l'algorithme de Gram-Schmidt classique (CGS), qui est très sensible aux erreurs d'arrondi. C'est pourquoi il est préférable d'utiliser la deuxième formulation, qui consiste à orthogonaliser successivement par rapport à q_1, à q_2 et ainsi de suite jusqu'à q_k. On obtient alors l'algorithme de Gram-Schmidt modifié (MGS).

ALGORITHM 1: MGS

$r_{11} := \|a_1\|$;
$q_1 := (1/r_{11})a_1$;
do $k = 1, m-1$,
  $w := a_{k+1}$;
  do $i = 1, k$
    $r_{i,k+1} := q_i^T w$;
    $w := w - r_{i,k+1} q_i$
  enddo ;
  $r_{k+1,k+1} := \|w\|$;
  $q_{k+1} := (1/r_{k+1,k+1})w$;
enddo

ALGORITHM 2: CGS

$r_{11} := \|a_1\|$;
$q_1 := (1/r_{11})a_1$;
do $k = 1, m-1$, {*orthogonalisation de* a_{k+1} *par rapport à* $q_1, \cdots q_k$ }
  $r'_{k+1} := {Q_k}^T a_{k+1}$;
  $w := a_{k+1} - Q_k r'_{k+1}$;
  $r_{k+1,k+1} := \|w\|$;
  $q_{k+1} := (1/r_{k+1,k+1})w$;
enddo

Les deux algorithmes sont mathématiquement équivalents mais leur comportement numérique est très différent. La stabilité numérique de (MGS) est établie par le théorème suivant [6] :

Théorème 4.1.2 *Avec une arithmétique flottante caractérisée par le paramètre de précision* ϵ*, l'algorithme MGS fournit un couple de matrices* $(\tilde{Q}, \tilde{R})$ *tel que :*

$$\|A - \tilde{Q}\tilde{R}\| = O(\epsilon\|A\|) \text{ et } \|\tilde{Q}^T\tilde{Q} - I_m\| = O(\epsilon\chi_2(A))$$

où le conditionnement $\chi_2(A)$ *est le rapport des valeurs singulières extrêmes de* A*.*

Par contre, l'algorithme CGS a un comportement nettement plus mauvais dans le cas d'orthogonalisation de systèmes presque liés (c.a.d. tels que $\chi_2(A)$ a une valeur élevée). Pour s'en convaincre on peut comparer les résultats obtenus par les applications des deux algorithmes à la matrice (exemple de

Lauchli) :

$$A = \begin{pmatrix} 1 & 1 & 1 \\ \epsilon & 0 & 0 \\ 0 & \epsilon & 0 \\ 0 & 0 & \epsilon \end{pmatrix}.$$

L'orthogonalité du système obtenu par MGS est fonction de ϵ tandis que les deuxième et troisième vecteurs obtenus par CGS ont un produit scalaire égal à 1/2.

4.2 Algorithme de Gram-Schmidt par blocs

On peut définir une version par blocs de l'algorithme (MGS) en remplaçant la manipulation des vecteurs par des opérations sur des blocs de vecteurs. L'orthogonalisation d'un vecteur par rapport à un autre qui était une correction de rang 1 devient l'orthogonalisation d'un bloc de vecteur par rapport à un bloc orthonormé de vecteurs, c'est-à-dire une correction de rang p où p est la largeur du bloc. La normalisation d'un vecteur est remplacée par l'application de MGS sur le bloc correspondant. On suppose que $m = lp$ et que les matrices A et Q sont partitionnées en l blocs de p colonnes: $A = [A_1; \cdots; A_l]$ et $Q = [Q_1; \cdots; Q_l]$.

On obtient alors l'algorithme suivant :

ALGORITHM 3: BGS

$(Q_1, R_{11}) := \text{MGS}(A_1)$;
do $k = 1, l-1$,
  $W := A_{k+1}$;
  do $i = 1, k$
    $R_{i,k+1} := {Q_i}^T W$;
    $W := W - Q_i R_{i,k+1}$
  enddo ;
  $(Q_{k+1}, R_{k+1,k+1}) := \text{MGS}(W)$
enddo

où les blocs R_{ik} sont des matrices carrées d'ordre p.

Il reste à voir la qualité numérique de ce nouvel algorithme. On montre en fait qu'elle peut être mauvaise, presqu'aussi mauvaise que CGS lorsque les colonnes de la matrice A sont presque dépendantes. Pour retrouver la qualité de l'algorithme MGS, il suffit de réorthogonaliser les blocs ; pour cela on double chaque appel de MGS [6, 23]. On appelle B2GS l'algorithme obtenu. Il est facile de montrer que l'algorithme BGS entraîne le même nombre $2nm^2$ d'opérations que l'algorithme MGS tandis que l'algorithme B2GS comporte $2n(m^2 + lp^2) = 2nm^2(1 + 1/l)$ opérations. Le surplus d'opérations sera donc relativement faible lorsque le nombre l de blocs sera petit. La comparaison des vitesses des trois algorithmes MGS, BGS et B2GS sur un processeur de CRAY 2 est illustrée par la figure 4.1 sur une matrice où $n = 1024$ et $m = 512$. On y constate que lorsque la taille du bloc est assez petite (ici $p = 32$ et donc $l = 16$) la perte provoquée par la réorthogonalisation est faible. Elle est de l'ordre de 13 % alors que la valeur de l laisserait supposer une perte deux fois

plus faible. Cela est dû au fait que les calculs qui sont répétés s'exécutent deux fois moins vite que le reste qui s'exprime par des multiplications matricielles.

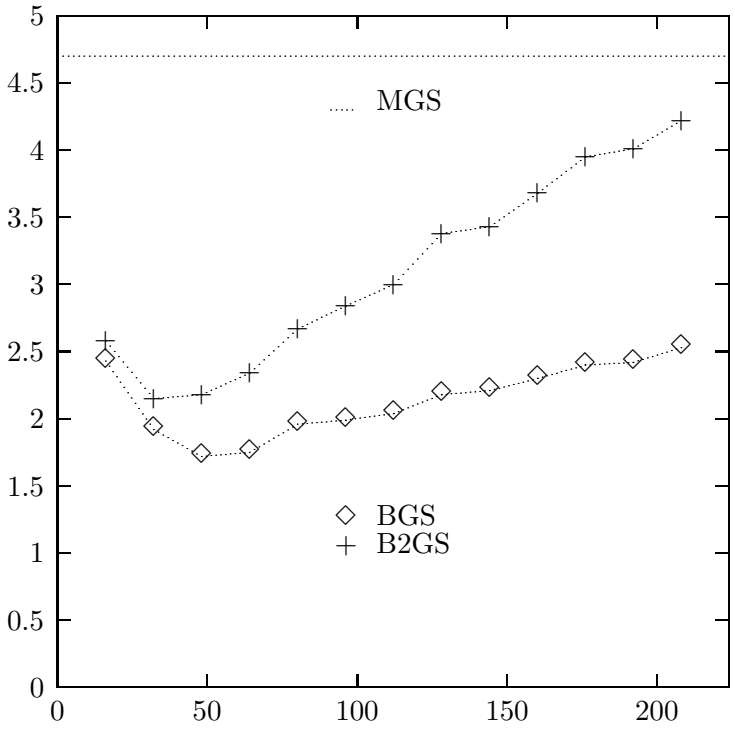

Figure 4.1: Temps de calcul (en s) en fonction de la taille des blocs.

4.3 Procédé d'Arnoldi

Dans beaucoup de méthodes itératives qui s'appliquent à une matrice carrée A d'ordre n, on a recours aux espaces de Krylov. Ces espaces sont définis par, outre la matrice, un vecteur initial v et un degré k : l'espace de Krylov

$$K_k(A, v) = Vect(v, Av, A^2v, \cdots, A^{k-1}v)$$

est le sous-espace engendré par les puissances successives de A appliquées au vecteur v. Ce sous-espace est donc celui des images des polynômes de A de degré inférieur ou égal à $k-1$ appliqués à v. On démontre facilement que si la dimension de ce sous-espace est strictement inférieure à k alors le sous-espace est invariant par A. Inversement, si k est supérieur au degré du polynôme minimal, alors le sous-espace est invariant par A. Nous nous plaçons maintenant dans le cas où $dim(K_k(A, v)) = k$. On exhibe un algorithme qui va construire une base orthonormée de $K_k(A, v)$. A cette fin, on applique l'algorithme MGS, non au système $(v, Av, A^2v, \cdots, A^{k-1}v)$ mais au système $(q_1, Aq_1, Aq_2, \cdots, Aq_{k-1})$ où les vecteurs $(q_i)_{i=1,k-1}$ sont les vecteurs du système orthonormé trouvé à l'étape $k-1$. Ces vecteurs sont donc calculés de proche en proche. Il est évident que si $(q_1, \cdots, q_{k-1})$ engendre toutes les images par des polynômes de degré inférieur ou égal à $k-2$, le système $(q_1, Aq_1, Aq_2, \cdots, Aq_{k-1})$ engendre toutes les images par des polynômes

de degré inférieur ou égal à $k-1$. On obtient donc bien une base orthonormée de $K_k(A,v)$. La factorisation QR réalisée par l'algorithme de Gram-Schmidt peut s'écrire matriciellement :

$$[q_1; Aq_1; \cdots; Aq_{k-1}] = [q_1; q_2; \cdots; q_k][e_1; \bar{H}_{k-1}]$$

où e_1 est le premier vecteur de la base canonique de $\mathbb{R}^k$ et $\bar{H}_{k-1}$ est une matrice de Hessenberg à k lignes et $k-1$ colonnes. En notant $Q_k = [q_1; q_2; \cdots; q_k]$ on obtient la relation fondamentale d'Arnoldi :

$$\begin{aligned} AQ_{k-1} &= Q_k \bar{H}_{k-1} \\ &= Q_{k-1} H_{k-1} + h_{k,k-1} q_k {e_{k-1}}^T \end{aligned}$$

où H_{k-1} est la matrice carrée constituée des $k-1$ premières lignes de $\bar{H}_{k-1}$. On en déduit aussi la relation

$${Q_{k-1}}^T A Q_{k-1} = H_{k-1} \tag{4.1}$$

L'algorithme d'Arnoldi qui construit le couple $(Q_m, \bar{H}_{m-1})$ s'écrit :

ALGORITHM 4: Arnoldi

$q_1 := (1/\|v\|)v$;
do $k = 1, m-1$,
  $w := Aq_k$;
  do $i = 1, k$
    $h_{i,k} := q_i^T w$;
    $w := w - h_{i,k} q_i$
  enddo ;
  $h_{k+1,k} := \|w\|$;
  $q_{k+1} := (1/h_{k+1,k})w$;
enddo

Il met en œuvre du calcul vectoriel (sur des vecteurs de longueur n) et des multiplications de la matrice par des vecteurs.

4.4 Algorithme de Lanczos symétrique

Lorsque la matrice A est symétrique, la relation 4.1 prouve qu'il en est de même de la matrice H_{k-1} qui est donc tridiagonale. L'algorithme d'Arnoldi dans ce cas se simplifie en l'algorithme de Lanczos :

ALGORITHM 5: Lanczos

$q_1 := (1/\|v\|)v$;
do $k = 1, m-1$,
  $w := Aq_k$;
  if $k > 1$, **then**
    $h_{k-1,k} := h_{k,k-1}$;
    $w := w - h_{k,k-1}q_{k-1}$;
  endif;
  $h_{k,k} := w^T q_k$;
  $w := w - h_{k,k}q_k$
  $h_{k+1,k} := \|w\|$;
  $q_{k+1} := (1/h_{k+1,k})w$;
enddo

Au contraire de l'Algorithme d'Arnoldi, les itérations de l'algorithme de Lanczos sont de complexité constante.

4.5 Algorithme de Lanczos non symétrique

Chapter 5

Résolution de systèmes linéaires par des méthodes directes

5.1 Inverse d'une matrice carrée et systèmes linéaires

Ce paragraphe a pour objet les matrices carrées et les systèmes linéaires. Il définit les notions de matrice inversible et de matrice singulière.

5.1.1 Inverse d'une matrice

Théorème 5.1.1 *$A \in \mathbb{R}^{n\times n}$, sont équivalents :*
A est inversible : il existe une matrice notée A^{-1}, telle que $AA^{-1} = I$ (et $A^{-1}A = I$),
$det(A) \neq 0$,
$rang(A) = n$,
le système linéaire $Ax = b$ a une solution pour tout b,
si le système linéaire $Ax = b$ a une solution, elle est unique,
$Im(A) = \mathbb{R}^n$,
$ker(A) = \{0\}$.

Preuve. $rang(A) = n$ équivaut à $Im(A) = \mathbb{R}^n$ et à $ker(A) = \{0\}$, ce qui équivaut aussi à $Ax = b$ a une solution pour tout b. D'après la proposition précédente, $rang(A) = n$ équivaut aussi à si le système a une solution, elle est unique.

Si $rang(A) = n$, soit Cb la solution unique du système $Ax = b$, alors $ACb = b, \forall b$ donc $AC = I$ et $CAx = Cb = x, \forall x$ donc $CA = I$ et A est inversible d'inverse C.

Réciproquement, si A est inversible alors $AA^{-1}b = b, \forall b$ donc le système $Ax = b$ a une solution pour tout b.

On admet que A inversible équivaut à $det(A) \neq 0$.

◇

Définition 5.1.1 *Une matrice carrée non inversible est dite singulière. Une matrice inversible est dite non singulière.*

Proposition 5.1.1 *Si A et B sont inversibles, alors AB est inversible et $(AB)^{-1} = B^{-1}A^{-1}$.*
Si A est inversible, alors A^T est inversible et $(A^T)^{-1} = (A^{-1})^T = A^{-T}$.

Preuve. $(AB)(B^{-1}A^{-1}) = A(BB^{-1})A^{-1} = AIA^{-1} = I$ donc AB est inversible d'inverse $B^{-1}A^{-1}$.

$A^T(A^{-1})^T = (A^{-1}A)^T = I$. ◇

5.1.2 Matrices particulières

Proposition 5.1.2 *Si Q est orthogonale, alors Q est inversible et $Q^{-1} = Q^T$.*

Une matrice diagonale $D = diag(d_i)$ est inversible si et seulement si $d_i \neq 0$ et l'inverse de D est la matrice diagonale $D^{-1} = diag(1/d_i)$.

Une matrice triangulaire L est inversible si et seulement si $L_{ii} \neq 0$ et l'inverse de L est triangulaire.

Preuve. Si Q est orthogonale, alors $Q^TQ = I$ donc $Q^{-1} = Q^T$.

Si D est diagonale, $Dx = b$ équivaut à $d_ix_i = b_i, i = 1, \ldots, n$, a une solution pour tout b si et seulement si $d_i \neq 0, i = 1, \ldots, n$; la solution vaut $x_i = b_i/d_i$ donc $D^{-1} = diag(1/d_i)$.

Si L est triangulaire inférieure, alors le système $Lx = b$ s'écrit $l_{11}x_1 = b_1, \sum_{j<i} l_{ij}x_j + l_{ii}x_i = b_i, i = 2, \ldots, n$ et a une solution pour tout b si et seulement si $l_{ii} \neq 0$; la solution vaut $x_1 = b_1/l_{11}, x_i = (b_i - \sum_{j<i} l_{ij}x_j)/l_{ii}, i = 2, \ldots, n$ donc x_i ne dépend que de $b_j, j \leq i$ et L^{-1} est triangulaire inférieure.

◇

Proposition 5.1.3 *Calculer l'inverse d'une matrice diagonale d'ordre n est de type BLAS1, avec n opérations.*

Résoudre un système triangulaire d'ordre n est de type BLAS2, avec $n^2/2$ opérations.

5.1.3 Résolution d'un système linéaire

On considère un système linéaire de n équations à n inconnues, qui a une solution unique, autrement dit le système $Ax = b$, avec A inversible.

La méthode de résolution de Cramer, basée sur les déterminants, est à proscrire, pour deux raisons :

- la complexité est élevée, en $O(n!)$, ce qui explose très vite.
- les phénomènes de cancellation dans les calculs de déterminants amplifient les erreurs d'arrondi et l'algorithme est numériquement instable. Même avec un système d'ordre 2, le résultat peut être complètement faux.

Si la matrice est diagonalisable, on peut en théorie exprimer le système dans la base des vecteurs propres et résoudre le système diagonal :

$$A = VDV^{-1}, Ax = b \Leftrightarrow D(V^{-1}x) = V^{-1}b.$$

Mais diagonaliser une matrice est très coûteux. Ensuite, il faudrait faire les calculs avec des nombres complexes, ce qui est nettement plus coûteux. De plus, toutes les matrices ne sont pas diagonalisables.

Les méthodes basées sur l'élimination d'inconnues, avec combinaisons linéaires des lignes, sont les méthodes directes de référence. Lorsqu'on élimine les inconnues, on aboutit finalement à un

système triangulaire supérieur, que l'on remonte pour calculer les coordonnées de la solution. Le processus d'élimination est en fait la résolution d'un système triangulaire inférieur. L'algorithme s'appelle la factorisation LU.

Pour garantir la stabilité numérique, il est nécessaire de modifier l'ordre d'élimination, en appliquant ce qu'on appelle une stratégie de pivot.

5.2 Factorisation LU

Théorème 5.2.1 *Soit $A \in \mathbb{R}^{n\times n}$ une matrice inversible. Alors il existe une matrice L triangulaire inférieure avec $L_{ii} = 1$ et une matrice triangulaire supérieure U inversible telles que*

$$A = LU \tag{5.1}$$

L'égalité (5.1) est appelée la factorisation de Gauss de A.

Preuve. admis. ◇

Pour résoudre un système linéaire, on procède par étapes :

- Factorisation de Gauss $A = LU$,
- Résolution du système triangulaire $Ly = b$ (descente),
- Résolution du système triangulaire $Ux = y$ (remontée).

Proposition 5.2.1 *L'algorithme de Gauss a une complexité en $n^3/3+O(n^2)$. C'est une procédure de LAPACK.*

5.2.1 Pivot partiel

5.2.2 Factorisation par blocs

La bibliothèque LAPACK utilise une partition par blocs pour la factorisation LU, afin d'utiliser BLAS3. Nous décrivons ici un algorithme sans pivot, mais LAPACK inclut des stratégies de pivot.

Supposons que la matrice A soit d'ordre n, qu'elle soit à diagonale dominante (donc l'algorithme de Gauss sans pivotage est licite) et qu'elle soit partagée en quatre blocs:

$$A \;=\; \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$$

où les blocs A_{11} et A_{22} sont carrés d'ordres respectifs m et $N-m$. La quantité $m \leq n$ représente la taille de bloc. On cherche à exprimer globalement l'étape qui traduirait m étapes de l'algorithme de Gauss classique. Il faut alors exprimer les blocs intervenant dans la décomposition LU où les blocs vérifient les égalités matricielles suivantes :

$$L \;=\; \begin{pmatrix} L_{11} & 0 \\ L_{21} & I \end{pmatrix} \text{ et } U \;=\; \begin{pmatrix} U_{11} & U_{12} \\ 0 & B \end{pmatrix}.$$

$$A_{11} = L_{11}U_{11} \quad (5.2)$$
$$A_{21} = L_{21}U_{11} \quad (5.3)$$
$$A_{12} = L_{11}U_{12} \quad (5.4)$$
$$A_{22} = L_{21}U_{12} + B \quad (5.5)$$

On en déduit le profil de la procédure (écrite sous forme récursive) :

ALGORITHM 6: BlockLU(A)
if $\dim(A) \leq m$ **then** {*On est arrivé au dernier bloc*}
factorisation LU de A ;
else {*On mène une étape de factorisation par bloc*}
définition des matrices $A_{11},\ A_{12},\ A_{21},\ A_{22}$;
factorisation LU de A_{11} pour obtenir L_{11} et U_{11} ;
résolution de $n-m$ systèmes triangulaires pour obtenir L_{21} (5.3) ;
résolution de $n-m$ systèmes triangulaires pour obtenir U_{12} (5.4) ;
mise à jour du bloc restant pour obtenir B (5.5) ;
LU(B)
endif

De manière évidente, on peut ensuite dérouler l'algorithme pour en supprimer la récursivité.

5.3 Factorisation de Cholesky

On considère la décomposition de matrices symétriques et définies positives c'est-à-dire telles que :

$$\forall x \neq 0,\ x^T A x > 0$$

5.3.1 Algorithme de factorisation

Théorème 5.3.1 Factorisation de Cholesky
Soit $A \in \mathbb{R}^{n\times n}$ une matrice symétrique et définie positive. Alors il existe une unique matrice triangulaire inférieure L qui vérifie :

$$A = LL^T \text{ et } diag(L) > 0$$

Preuve. On démontre le théorème par récurrence sur la dimension n de la matrice.

$n = 1$: le résultat est évident car alors le nombre qui représente la matrice est positif et le facteur de Cholesky est sa racine carrée.

On suppose la proposition vraie pour toute matrice d'ordre n. Soit A' une matrice d'ordre $n+1$ que l'on partitionne en blocs de la manière suivante où $A \in \mathbb{R}^{n\times n},\ a \in \mathbb{R}^n,$ et $\alpha \in \mathbb{R}$:

$$A' = \begin{pmatrix} A & a \\ a^T & \alpha \end{pmatrix}.$$

On recherche le facteur de Cholesky L' partitionné de la même manière :

$$L' = \begin{pmatrix} L & 0 \\ l^T & \lambda \end{pmatrix}$$

qui vérifie $L'{L'}^T = A'$ ou encore

$$LL^T = A \tag{5.6}$$
$$Ll = a \tag{5.7}$$
$$\|l\|^2 + \lambda^2 = \alpha. \tag{5.8}$$

La matrice A étant une matrice principale de la matrice A' est définie positive. On peut donc lui appliquer l'hypothèse de récurrence, ce qui détermine le facteur L vérifiant l'équation (5.6) et à diagonale positive. L'équation (5.7) permet alors de calculer $l = L^{-1}a$. En remplaçant dans la dernière équation les quantités déjà trouvées, il reste à résoudre :

$$\lambda^2 = \alpha - a^T A^{-1} a.$$

Il reste à prouver que le terme de droite de l'égalité est positif ce qui pemettra de conclure que $\lambda = \sqrt{\alpha - a^T A^{-1} a}$. On suppose que $a \neq 0$ car sinon le résultat est évident. Pour cela on étudie, pour tout vecteur $u \in \mathbb{R}^n$ non nul et tout réel ρ, la quantité :

$$\tau = (u^T \ \rho) \begin{pmatrix} A & a \\ a^T & \alpha \end{pmatrix} \begin{pmatrix} u \\ \rho \end{pmatrix} = \alpha\rho^2 + 2\rho u^T a + u^T A u$$

qui doit être strictement positive pour tout ρ. Le discriminant du trinôme est donc négatif : $(u^T a)^2 - \alpha(u^T A u) < 0$. Le résultat est obtenu en choisissant $u = A^{-1}a$ et en remarquant que $a^T A^{-1} a > 0$.

L'unicité et l'existence de la décomposition est donc prouvée. ◇

On peut noter que la décomposition en blocs introduite dans la démonstration aboutit à un premier algorithme qui récursivement construit le facteur de Cholesky de dimension $n+1$ à partir de celui de la matrice principale d'ordre n. Soit $C(n)$ le nombre d'opérations pour calculer le facteur de Cholesky d'une matrice d'ordre n. On obtient donc la relation de récurrence

$$C(n+1) = C(n) + n^2 + O(n)$$

car la résolution du système triangulaire qui calcule l est de complexité $n^2 + O(n)$. On en déduit que $C(n) = 1/3\ n^3 + O(n^2)$.

En fait, l'algorithme le plus courant est celui obtenu à partir de la décomposition suivante (avec les mêmes notations que pour le théorème) :

$$A' = \begin{pmatrix} \alpha & a^T \\ a & A \end{pmatrix}.$$

On recherche le facteur de Cholesky L' partitionné de la même manière :

$$L' = \begin{pmatrix} \lambda & 0 \\ l & L \end{pmatrix}$$

qui vérifie $L'L'^T = A'$ ou encore

$$\begin{aligned} \lambda^2 &= \alpha & (5.9)\\ \lambda l &= a & (5.10)\\ l\ l^T + LL^T &= A. & (5.11) \end{aligned}$$

Ces équations se résolvent en

$$\begin{cases} \lambda = \sqrt{\alpha} \\ l = (1/\lambda)a \\ \text{Factorisation de Cholesky de } (A - l\ l^T) \end{cases} \quad (5.12)$$

On peut donc dérouler la récurrence suivant l'algorithme suivant :

ALGORITHM 7: Cholesky (Right Looking)

$L :=$ triangle_inférieur(A) ;
$L(1,1) := \sqrt{L(1,1)}$;
$L(2:n,1) := (1/L(1,1)) * L(2:n,1)$;
do $j = 1, n-1,$
  do $k = j+1, n$
    $L(k:n,k) := L(k:n,k) - L(k,j) * L(k:n,j)$;
  enddo ;
  $L(j+1,j+1) := \sqrt{L(j+1,j+1)}$;
  $L(j+2:n,j+1) := (1/L(j+1,j+1)) * L(j+2:n,j+1)$;
enddo

Cet algorithme est appelé Right Looking (ou Fan Out), car dès qu'une colonne de L est prête, elle corrige toutes les colonnes suivantes. On peut aussi imaginer d'organiser autrement les calculs : on retarde chaque mise à jour d'une colonne au plus tard possible; ainsi une colonne de la matrice A n'est considérée que lorsque toutes les précédentes de L sont construites. Cela revient à inverser dans l'algorithme 7 les boucles indicées par `j` et `k`. On obtient alors l'algorithme 8 appelé Left Looking (ou Fan In).

ALGORITHM 8: Cholesky (Left Looking)

$L(1,1) := \sqrt{A(1,1)}$;
$L(2:n,1) := (1/L(1,1)) * A(2:n,1)$;
do $k = 2, n$,
  $L(k:n,k) := A(k:n,k)$;
  do $j = 1, k-1$
    $L(k:n,k) := L(k:n,k) - L(k,j) * L(k:n,j)$;
  enddo ;
  $L(k,k) := \sqrt{L(k,k)}$;
  $L(k+1:n,k) := (1/L(k,k)) * L(k+1:n,k)$;
enddo

En fait, si l'on considère que l'algorithme est composé de trois boucles emboîtées (aux deux boucles indiquées, il faut rajouter la boucle de l'opération vectorielle interne), il existe six manières de les ordonner. Pour plus de précision, on peut consulter [20, 21].

Proposition 5.3.1 *L'algorithme de Cholesky a une complexité en* $n^3/6+O(n^2)$. *C'est une procédure de LAPACK, codée par blocs.*

5.3.2 Stabilité numérique de Cholesky

5.4 Conditionnement d'un système linéaire

Nous étudions la sensibilité d'un système linéaire aux variations sur les données (conditionnement). Beaucoup d'ouvrages traitent de ce sujet, notamment [18, 40, 26, 22, 8, 10].

Nous introduisons d'emblée une définition qui va être justifiée par les résultats associés.

Définition 5.4.1 *Le conditionnement d'une matrice* A *pour les systèmes linéaires est donné par*

$$\kappa(A) = \|A\|\|A^{-1}\|.$$

La bibliothèqe LAPACK fournit des procédures pour estimer le conditionnement d'une matrice.

Théorème 5.4.1 *Soit* A *une matrice inversible et* x *la solution de* $Ax = b$. *Soit* $\Delta A, \Delta b$ *tels que*

$$\|\Delta A\| \leq \alpha\|A\|, \|\Delta b\| \leq \beta\|b\|, \alpha\ \kappa(A) < 1.$$

Alors $A + \Delta A$ *est inversible. Soit* $x + \Delta x$ *la solution du système*

$$(A + \Delta A)(x + \Delta x) = b + \Delta b,$$

alors

$$\frac{\|\Delta x\|}{\|x\|} \leq \frac{\kappa(A)}{1 - \alpha\kappa(A)}(\alpha + \beta).$$

Ce théorème correspond à la définition générale, où les données sont soit A et b (si seulement A est la donnée, alors $\beta = 0$).

La preuve du théorème nécessite le lemme suivant :

Lemme 5.4.1 *Si $\|F\| < 1$ alors $(I + F)$ est inversible et $\|(I + F)^{-1}\| \leq \frac{1}{1-\|F\|}$.*

Preuve.

$$A + \Delta A = A(I + A^{-1}\Delta A),$$
$$\|A^{-1}\Delta A\| \leq \kappa(A)\alpha < 1,$$

donc d'après le lemme, $I + A^{-1}\Delta A$ est inversible et

$$\|(I + A^{-1}\Delta A)^{-1}\| \leq \frac{1}{1 - \alpha\kappa(A)}.$$

Soit $y = x + \Delta x$, on a $(A + \Delta A)y = b + \Delta b, \quad Ax = b$ donc $A(y - x) = \Delta b - \Delta A y$ et

$$\|y - x\| \leq \|A^{-1}\|(\|\Delta b\| + \|\Delta A\|\|y\|).$$
$$\text{ or } \|\Delta b\| \leq \beta\|b\| \leq \beta\|A\|\|x\|,$$
$$\text{d'où } \|\Delta x\| \leq \kappa(A)(\beta\|x\| + \alpha\|y\|).$$

Il reste à majorer $\|y\|$. On a

$$A(I + A^{-1}\Delta A)y = A(x + A^{-1}\Delta b),$$
$$\|y\| \leq \|(I + A^{-1}\Delta A)^{-1}\|(\|x\| + \|A^{-1}\|\Delta b\|),$$
$$\|y\| \leq \tfrac{1}{1-\alpha\kappa(A)}(1 + \beta\kappa(A))\|x\|.$$

Par conséquent, $\beta\|x\| + \alpha\|y\| \leq \frac{1}{1-\alpha\kappa(A)}(\alpha + \beta)\|x\|$.

On en déduit que

$$\|\Delta x\| \leq \frac{\kappa(A)}{1 - \alpha\kappa(A)}(\alpha + \beta)\|x\|.$$

◇

5.4.1 Lien avec le résidu

Comme pour le cas général, il est possible de faire le lien avec le résidu.

Soit x la solution du système linéaire $Ax = b$. Nous supposons qu'un algorithme de résolution calcule le vecteur y, avec un résidu $r = b - Ay$.

Corollaire 5.4.1

$$\frac{\|x - y\|}{\|x\|} \leq \kappa(A)\frac{\|b - Ay\|}{\|b\|}. \tag{5.13}$$

Il est possible d'avoir un résultat plus fin en introduisant une perturbation de la matrice. Si $\|b - Ay\|$ est assez petit, alors y est solution d'un système linéaire perturbé et nous déterminons la plus petite perturbation possible.

Théorème 5.4.2 *Soit $Ax = b$ un système linéaire d'ordre n et soit $y \in \mathbb{R}^n$. Soit*

$$S = \{\alpha,\ \textit{il existe}\ \Delta A,\ \Delta b,\ \|\Delta A\| \leq \alpha\ \|A\|,\ \|\Delta b\| \leq \alpha\|b\|,\ (A + \Delta A)y = b + \Delta b\}$$

Soit

$$\eta = \frac{\|b - Ay\|}{\|A\|\|y\| + \|b\|}. \tag{5.14}$$

Si $\eta\kappa(A) < 1$ alors $\min_{\alpha \in S} = \eta$.

Preuve. Soit $r = b - Ay$ et soit $\alpha \in S$. Alors

$$\begin{array}{l}(A + \Delta A)y = b + \Delta b,\\ r = \Delta A y - \Delta b,\\ \|r\| \leq \alpha(\|A\|\|y\| + \|b\|),\\ \text{donc } \eta \leq \alpha.\end{array}$$

Réciproquement, soit

$$\begin{array}{l}\Delta A = \eta\ \frac{\|A\|}{\|r\|\ \|y\|}\ ry^T, \Delta b = -\eta\frac{\|b\|}{\|r\|}r.\\ \text{Alors } \|\Delta A\| = \eta\|A\|,\ \|\Delta b\| = \eta\|b\|.\\ \text{On a } (A + \Delta A)y = b - r + \Delta A y = b - r + \eta\ \frac{\|A\|\|y\|}{\|r\|}r,\\ (A + \Delta A)y = b - \frac{\|b\|}{\|A\|\|y\|+\|b\|}r = b + \Delta b,\\ \text{donc } \eta \in S.\end{array}$$

Donc η est le minimum de S. ◇

En combinant les deux théorèmes sur le conditionnement et l'analyse inverse, on obtient une estimation d'erreur pour la solution calculée y.

Corollaire 5.4.2 *Soit x la solution du système linéaire $Ax = b$ et soit y une solution calculée telle que $\eta\kappa(A) < 1$, où η est défini par (5.14). Alors*

$$\frac{\|x - y\|}{\|x\|} \leq 2\frac{\kappa(A)}{1 - \eta\kappa(A)}\eta \tag{5.15}$$

Le calcul de $\|r\|$ puis η et une estimation du conditionnement $\kappa(A)$ permettent donc d'estimer une borne de l'erreur sur la solution.

Chapter 6

Résolution itérative de systèmes linéaires

Dans tout ce chapitre, on cherche à résoudre le système

$$Ax = b, \tag{6.1}$$

où $A \in \mathbb{R}^{n\times n}$ est une matrice inversible et $b \in \mathbb{R}^n$ le second membre.

6.1 Méthodes itératives linéaires

Les méthodes itératives basiques sont les méthodes appelées ici linéaires. Elles ne sont plus beaucoup utilisées en soi car leur convergence n'est pas toujours assurée et est en général lente, surtout pour les systèmes de grande taille. Elles servent par contre à accélérer la convergence d'une classe importante de méthodes itératives, appelées méthodes polynomiales. Pour une description détaillée de ces méthodes, on pourra consulter [4, 7, 28, 35].

Dans ces méthodes itératives, la matrice n'est pas transformée, seule l'opération produit matrice-vecteur $y = Ax$ est requise. Il est donc possible d'utiliser des versions dites "matrix-free" où la matrice n'est pas stockée. Les méthodes itératives nécessitent en général moins d'espace mémoire que les méthodes directes.

Dans tout ce qui suit, on note x_k l'approximation de la solution à l'itération k, on note $r_k = b - Ax_k$ le résidu et $e_k = x^* - x_k$ l'erreur, de sorte que $r_k = Ae_k$.

Les méthodes linéaires sont basées sur une décomposition

$$A = M - N, \tag{6.2}$$

où M est une matrice inversible. L'itération est un schéma de point fixe, défini, à partir de x_0 donné, par

$$Mx_{k+1} = Nx_k + b. \tag{6.3}$$

On obtient donc $Mx_{k+1} = (M - A)x_k + b = Mx_k + r_k$ soit

$$x_{k+1} = x_k + M^{-1}r_k.$$

On a aussi $Mx_{k+1} = Nx_k + (M - N)x^*$ d'où $M(x^* - x_{k+1}) = N(x^* - x_k)$ donc

$$e_{k+1} = (M^{-1}N)e_k.$$

Théorème 6.1.1 *Soit ρ le rayon spectral de la matrice $M^{-1}N$, correspondant à la décomposition $A = M - N$ de la matrice inversible A. La méthode (6.3) est convergente pour tout vecteur initial x_0 si et seulement si $\rho < 1$.*

Les exemples les plus classiques sont les méthodes de Jacobi, Gauss-Seidel, SOR et SSOR.

Soit $A = D - E - F$ la décomposition de A en parties diagonale, triangulaire inférieure et triangulaire supérieure.

La méthode de Jacobi est définie par $M = D$, celle de Gauss-Seidel par $M = D - E$. La méthode SOR (Successive Over Relaxation) résout $\omega Ax = \omega b$ avec la décomposition $M = D - \omega E$, où ω est un paramètre choisi pour accélérer la convergence. On a les itérations

$$\begin{aligned} Jacobi &: Dx_{k+1} = (E+F)x_k + b \\ Gauss-Seidel &: (D-E)x_{k+1} = Fx_k + b \\ SOR &: (D-\omega E)x_{k+1} = (\omega F + (1-\omega)D)x_k + \omega b \end{aligned}$$

La méthode SSOR (Symmetric SOR) effectue une itération SOR suivie d'une itération SOR en sens inverse. On obtient

$$M = \frac{1}{\omega(2-\omega)}(D-\omega E)D^{-1}(D-\omega F)$$

Il est possible de prouver la convergence pour certaines classes de matrices. Pour la démonstration de ces théorèmes et pour d'autres résultats, voir par exemple [18, 43].

Définition 6.1.1 *Une matrice $A = (a_{ij})$ d'ordre n est à diagonale strictement dominante si*

$$|a_{jj}| > \sum_{i=1, i\neq j}^{n} |a_{ij}|, \ j = 1, \ldots, n$$

Théorème 6.1.2 *Si A est à diagonale strictement dominante, les méthodes de Jacobi et de Gauss-Seidel convergent pour tout x_0.*

Théorème 6.1.3 *Soit A une matrice symétrique avec des éléments diagonaux positifs et soit $0 < \omega < 2$. La méthode SOR converge pour tout x_0 si et seulement si A est définie positive.*

6.2 Méthodes de projection

6.2.1 Définition d'une méthode polynomiale

Commençons par donner un résultat qui donner des raisons de rechercher des approximations polynomiales pour la résolution d'un système.

Proposition 6.2.1 *Il existe un polynôme p de degré au plus $n-1$ tel que $A^{-1} = p(A)$.*

Preuve. D'après le théorème de Cayley-Hamilton, le polynôme caractéristique q de A est de degré n et vérifie $q(A) = 0$, $q(0) = det(A)$, où $det(A)$ est le déterminant de A. Soit $q(X) = det(A) + Xq_1(X)$ alors $Aq_1(A) = -det(A)I$. Puisque A est inversible, $det(A) \neq 0$ et $-\frac{1}{det(A)}Aq_1(A) = I$. Soit p le polynôme défini par $p(X) = -\frac{1}{det(A)}q_1(X)$, alors $A^{-1} = p(A)$. $\diamond$

Soit x_0 une approximation initiale et $r_0 = b - Ax_0$. Alors la solution x^* vérifie $x^* = x_0 + A^{-1}r_0 = x_0 + p(A)r_0$.

L'objectif des méthodes polynomiales est d'approcher ce polynôme $p(X)$.

Définition 6.2.1 *Les méthodes polynomiales sont définies par des itérations*

$$x_k = x_0 + s_{k-1}(A)r_0, \tag{6.4}$$

où s_{k-1} est un polynôme de degré au plus $k-1$.

Le sous-espace $\mathcal{K}_k(A, r_0) = eng\{r_0, Ar_0, \ldots, A^{k-1}r_0\}$ est appelé espace de Krylov associé à A et r_0. Une méthode itérative est polynomiale si et seulement si elle vérifie la condition de sous-espace

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

ou, de façon équivalente,

$$x_{k+1} \in x_k + \mathcal{K}_{k+1}(A, r_0). \tag{6.5}$$

C'est une méthode de sous-espace liée aux espaces de Krylov.

Proposition 6.2.2 *Dans une méthode polynomiale, le résidu r_k et l'erreur e_k vérifient*

$$r_k = q_k(A)r_0, \quad e_k = q_k(A)e_0, \tag{6.6}$$

où $q_k(X) = 1 - Xs_{k-1}(X)$ est un polynôme de degré au plus k qui vérifie $q_k(0) = 1$.

Preuve. $r_k = b - A(x_0 + s_{k-1}(A)r_0) = r_0 - As_{k-1}(A)r_0 = q_k(A)r_0$ où $q_k(X) = 1 - Xs_{k-1}(X)$.
$e_k = A^{-1}r_k = A^{-1}q_k(A)r_0 = q_k(A)A^{-1}r_0 = q_k(A)e_0$. ◇

Définition 6.2.2 *L'ensemble des polynômes résiduels de degré k est défini par*

$$\mathcal{P}_k^0 = \{q \text{ est un polynôme de degré } k \text{ et } q(0) = 1\}.$$

On aboutit donc à deux directions pour définir une méthode polynomiale :

- La première, la moins courante, consiste à définir explicitement le polynôme. C'est le cas de la méthode CHEBY qui s'applique au cas où la matrice est symétrique définie positive. Comme nous ne l'étudions pas ici, on peut se reporter à [4] pour obtenir l'algorithme.
- La deuxième manière de considérer une méthode polynomiale est de travailler avec les espaces de Krylov. Dans ce cas, le polynôme résiduel n'est pas explicitement connu. C'est cette approche que suivent toutes les méthodes actuelles.

Par la proposition suivante, on montre que les méthodes polynomiales, définies à partir des espaces de Krylov, doivent engendrer des suites finies qui aboutissent à la solution exacte.

Proposition 6.2.3 *La suite des sous-espaces de Krylov $\mathcal{K}_k(A, r_0)$ est une suite croissante de sous-espaces, et donc stationnaire à partir d'un rang p ; pour $k \leq p$, la dimension de $\mathcal{K}_k(A, r_0)$ est égale à k, et le système $\{r_0, Ar_0, \ldots, A^{k-1}r_0\}$ est une base de $\mathcal{K}_k(A, r_0)$.*

Le sous-espace $\mathcal{K}_p(A, r_0)$ est un sous-espace invariant de A; l'équation $Ay = r_0$ a une solution dans cet espace.

Preuve. La première partie de la proposition est évidente puisque tous les sous-espaces sont au plus de dimension n, dimension de A. Soit p le premier rang où la dimension de $\mathcal{K}_{k+1}(A, r_0)$ est égale à k. Alors $\mathcal{K}_{p+1}(A, r_0) = \mathcal{K}_p(A, r_0)$, et la suite est stationnaire à partir de ce rang. On a donc :

$$A\mathcal{K}_p(A, r_0) \subset \mathcal{K}_{p+1}(A, r_0) = \mathcal{K}_p(A, r_0),$$

ce qui pouve l'invariance de l'espace. L'opérateur A définit donc une bijection de l'espace dans lui-même. Le vecteur r_0 appartenant à $\mathcal{K}_p(A, r_0)$, il a un antécédent dans cet espace. ◇

En fait, pour les grands systèmes, on cherche à arrêter les itérations avant d'aboutir à la solution exacte. A l'étape k, pour déterminer l'itéré x_k, on souhaiterait satisfaire une condition de minimisation de l'erreur pour certain produit scalaire. Cela n'est pas toujours possible, et la recherche dans le domaine a abouti à une condition plus générale que nous abordons maintenant.

6.2.2 Méthodes polynomiales de projection

Définition 6.2.3 *Une méthode de projection de Krylov est une méthode polynomiale définie par une matrice B et deux conditions : la condition de sous-espace (6.5) et la condition dite de Petrov-Galerkin :*

$$(Be_k)^T u = 0, \ \forall u \in \mathcal{K}_k(A, r_0). \tag{6.7}$$

Le choix de B dicte la construction de la méthode de projection. Lorsque la matrice B définit un produit scalaire, on obtient la propriété de minimisation souhaitée :

Proposition 6.2.4 *Si B est une matrice symétrique définie positive alors, pour x_k satisfaisant la condition de sous-espace (6.5), la condition de Petrov-Galerkin (6.7) est équivalente à minimiser l'erreur :*

$$\|e_k\|_B = \min_{y \in \mathcal{K}_k(A, r_0)} \|e_0 - y\|_B, \tag{6.8}$$

où $e_k = x - x_k = x - (x_0 + y) = e_0 - y$. Dans ce cas x_k est déterminé de manière unique.

Preuve. Soit $x_k = x_0 + y$ avec $y \in \mathcal{K}_k(A, r_0)$. La condition (6.7) exprime que l'erreur $e_k = e_0 - y$ doit appartenir au B-orthogonal de l'espace $\mathcal{K}_k(A, r_0)$. Or $e_k \in e_0 + \mathcal{K}_k(A, r_0)$. Il s'ensuit que e_k est la projection B-orthogonale de e_0 sur $(\mathcal{K}_k(A, r_0))^{\perp_B}$. Le vecteur y solution du problème aux moindres carrés (6.8) est bien la B-projection de e_0 sur $\mathcal{K}_k(A, r_0)$. ◇

6.2.3 Préconditionnement d'une méthode polynomiale

La vitesse de convergence des méthodes itératives polynomiales dépend de certaines propriétés de la matrice A. Une façon d'accélérer la convergence est de transformer le problème (6.1) en le préconditionnant à l'aide d'une matrice connue. Le nouveau système à résoudre est équivalent mais l'objectif est que la méthode itérative polynomiale converge plus rapidement.

Définition 6.2.4 *Un préconditionnement est défini par une matrice inversible $C \in \mathbb{R}^{n,n}$. Un préconditionnement à gauche résout le système équivalent*

$$CAx = Cb,$$

tandis qu'un préconditionnement à droite résout le système équivalent

$$AC(C^{-1}x) = b.$$

Dans un système préconditionné à gauche, la matrice est donc CA, de sorte que le produit $v = Au$ est remplacé par la séquence des deux produits $w = Au, v = Cw$.

Si $C = A^{-1}$, alors $CA = I$ de sorte que le système préconditionné est trivial. Le préconditionnement C est souvent choisi pour approcher A^{-1}.

Proposition 6.2.5 *Une méthode polynomiale préconditionnée à gauche est caractérisée par*

$$x_k \in x_0 + \mathcal{K}_k(CA, Cr_0),$$

et une méthode polynomiale préconditionnée à droite est caractérisée par

$$x_k \in x_0 + C\mathcal{K}_k(AC, r_0).$$

Preuve. Considérons le système $CAx = Cb$. Soit x_0 le choix initial de la méthode polynomiale associée. Alors le résidu s_0 vérifie $s_0 = Cb - CAx_0 = C(b - Ax_0) = Cr_0$ et le polynôme p_k est appliqué à CA.

La preuve est similaire pour le système préconditionné à droite. ◇

Les méthodes linéaires vues précédemment sont en fait des méthodes polynomiales préconditionnées.

Proposition 6.2.6 *Une méthode linéaire est une méthode polynomiale préconditionnée à droite. Le préconditionnement est $C = M^{-1}$ et le polynôme résiduel est $q(X) = (1 - X)^k$.*

Preuve. $r_{k+1} = r_k - AM^{-1}r_k = (I - AM^{-1})r_k = (I - AM^{-1})^k r_0$. ◇

6.3 Cas où A est symétrique définie positive

Nous avons vu que la méthode SOR converge si A est symétrique définie positive. Toutefois, la vitesse de convergence est en général très lente. Il est préférable d'utiliser la méthode de Gradient Conjugué.

6.3.1 Méthode du Gradient Conjugué

La méthode du Gradient Conjugué (GC) est la méthode polynomiale de choix dans le cas où A est symétrique définie positive. Elle est basée sur la minimisation de la fonctionnelle ϕ. De nombreux ouvrages décrivent cette méthode. Nous décrivons la méthode GC dans le cadre des méthodes de projections de Krylov.

Définition 6.3.1 *La méthode du gradient conjugué (GC) est la méthode de projection de Krylov dans laquelle $B = A$ et A est symétrique définie positive.*

La proposition 6.2.4 se traduit immédiatement par

Proposition 6.3.1 *Les itérés x_k du gradient conjugué sont bien définis pour tout $k \geq 0$,à partir des conditions d'espace et de Petrov-Galerkin. Ils sont tels qu'ils minimisent l'erreur $\|x - x_k\|_A$ en norme A, quantité égale au résidu $\|b - Ax_k\|_{A^{-1}}$ en norme A^{-1}. En au plus n étapes, la solution exacte est trouvée (c.a.d. $x = x_k$).*

Définition 6.3.2 *D'après la condition d'espace (6.5), il existe $\alpha_k \in \mathbb{R}$ et $p_k \in \mathcal{K}_{k+1}(A, r_0)$ tels que*

$$x_{k+1} = x_k + \alpha_k p_k. \tag{6.9}$$

Le vecteur p_k est appelé direction de descente.

Proposition 6.3.2 *La condition de Galerkin (6.7) est équivalente à à*

$$r_k \perp \mathcal{K}_k(A, r_0). \tag{6.10}$$

Preuve. Evidente. ◇

Théorème 6.3.1 *Tant que les résidus r_k sont non nuls, la méthode GC vérifie*

$$\begin{array}{l} r_0 = b - Ax_0, \\ x_{k+1} = x_k + \alpha_k p_k, \\ r_{k+1} = r_k - \alpha_k A p_k, \\ eng\{r_0, r_1, \ldots, r_k\} = eng\{p_0, p_1, \ldots, p_k\} = \mathcal{K}_{k+1}(A, r_0), \\ r_k^T r_j = 0,\ j \neq k, \\ p_k^T A p_j = 0,\ j \neq k, \end{array}$$

et les directions de descente peuvent être choisies pour vérifier la récurrence (à une constante près)

$$p_0 = r_0,\ p_{k+1} = r_{k+1} + \beta_k p_k. \tag{6.11}$$

Preuve. La condition (6.9) implique $r_{k+1} = r_k - \alpha_k A p_k$. On a $r_k \in \mathcal{K}_{k+1}(A, r_0)$ et

$$\begin{array}{l} eng\{r_0, r_1, \ldots, r_k\} \subseteq \mathcal{K}_{k+1}(A, r_0), \\ eng\{p_0, p_1, \ldots, p_k\} \subseteq \mathcal{K}_{k+1}(A, r_0). \end{array}$$

La condition de Galerkin (6.10) et le fait que les résidus sont dans l'espace de Krylov impliquent immédiatement l'orthogonalité des résidus.

Si $r_j \neq 0,\ j = 0, 1, \ldots, k$, alors le sous-espace $eng\{r_0, r_1, \ldots, r_k\}$ est de dimension $k+1$ (base orthogonale) et on obtient l'égalité des sous-espaces. Le sous-espace de Krylov $\mathcal{K}_k(A, r_0)$ est donc de dimension k.

La condition de Galerkin et la récurrence $r_{k+1} = r_k - \alpha_k A p_k$ induisent $p_j^T A p_k = 0,\ j \leq k-1$. Puisque A est symétrique, on obtient $p_k^T A p_j = 0,\ j \leq k-1$.

Puisque $p_0 \in eng\{r_0\}$, on peut choisir $p_0 = r_0$.

Puisque $r_{k+1} \in eng\{p_0, p_1, \ldots, p_{k+1}\}$, on peut écrire

$$r_{k+1} = \sum_{i=0}^{k+1} \gamma_i p_i.$$

Pour $j \leq k-1,\ Ap_j \in \mathcal{K}_{j+2} \subseteq \mathcal{K}_{k+1}$ donc $r_{k+1}^T A p_j = 0$. D'autre part, $p_i^T A p_j = 0, i \neq j$ d'où $\gamma_j = 0, j \leq k-1$. On en déduit que

$$r_{k+1} = \gamma_k p_k + \gamma_{k+1} p_{k+1}.$$

Puisque $dim(\mathcal{K}_k) = k$, on a $\gamma_{k+1} \neq 0$ et on choisit $\gamma_{k+1} = 1$. On en déduit la récurrence (6.11). ◇

Remarque 6.3.1 *L'hypothèse que A est symétrique permet de montrer la récurrence courte (6.11) sur les directions de descente, qui relie p_{k+1} à seulement p_k et non aux p_j.*

Il reste à caractériser α_k et β_k pour définir complètement la méthode.

Théorème 6.3.2 *On considère le système $Ax = b$, où A une matrice spd. La méthode du Gradient Conjugué est définie par l'algorithme 9.*

ALGORITHM 9: CG

* Initialisation ;
choisir x_0 ;
$r_0 = b - Ax_0$;
$p_0 = r_0$;
* Iterations ;
for $k = 0, 1, \ldots$ **until** convergence **do**
$\alpha_k = \frac{\|r_k\|_2^2}{p_k^T A p_k}$;
$x_{k+1} = x_k + \alpha_k p_k$;
$r_{k+1} = r_k - \alpha_k A p_k$;
$\beta_k = \frac{\|r_{k+1}\|_2^2}{\|r_k\|_2^2}$;
$p_{k+1} = r_{k+1} + \beta_k p_k$;
end do

Preuve. Les récurrences sont déjà prouvées.

Les résidus sont orthogonaux donc, en particulier, $r_{k+1}^T r_k = 0$, d'où $r_k^T r_k - \alpha_k r_k^T A p_k = 0$. Mais $r_k = p_k - \beta_{k-1} p_{k-1}$ et les directions de descente sont A-conjuguées donc $r_k^T A p_k = p_k^T A p_k$ et $\alpha_k = r_k^T r_k / p_k^T A p_k$.

Les directions de descente sont conjuguées donc, en particulier, $p_{k+1}^T A p_k = 0$. D'autre part,

$$r_{k+1}^T A p_k = \frac{1}{\alpha_k} r_{k+1}^T (r_k - r_{k+1}) = -\frac{1}{\alpha_k} r_{k+1}^T r_{k+1}$$

d'où

$$\beta_k = \frac{r_{k+1}^T r_{k+1}}{\alpha_k p_k^T A p_k} = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}.$$

Réciproquement, si les suites sont définies par les récurrences ci-dessus, il est aisé de montrer par récurrence que la méthode vérifie les conditions d'espace (6.5) et de Petrov-Galerkin (6.7). ◇

6.3.2 Lien avec la méthode de Lanczos

La méthode de Lanczos symétrique construit itérativement une base orthonormée de l'espace de Krylov ayant un vecteur v_1 comme départ. La base orthonormée V_k vérifie $V_k^T A V_k = T_k$, où T_k est une matrice tridiagonale. L'objet de ce paragraphe est de montrer que la méthode de Gradient Conjugué est une variante de la méthode de Lanczos.

Lemme 6.3.1 *Soit* $v_{j+1} = \frac{r_j}{\|r_j\|}$, $j = 0, 1, \ldots$ *et* $V_k = (v_1, \ldots, v_k)$. *Alors le système* V_k *est une base orthonormée de l'espace de Krylov* $\mathcal{K}_k(A, r_0)$ *et*

$$Av_{k+1} = \delta_k v_k + \gamma_{k+1} v_{k+1} + \delta_{k+1} v_{k+2}, \tag{6.12}$$

où

$$\delta_k = -\frac{\sqrt{\beta_{k-1}}}{\alpha_{k-1}}, \quad \gamma_{k+1} = \frac{1}{\alpha_k} + \frac{\beta_{k-1}}{\alpha_{k-1}}.$$

Preuve. les vecteurs v_k sont orthonormés et engendrent l'espace de Krylov.

Les relations de récurrence s'écrivent

$$\begin{array}{l} r_k = p_k - \beta_{k-1} p_{k-1} \text{ et } Ap_k = \frac{1}{\alpha_k}(r_k - r_{k+1}), \\ \text{d'où} \\ Ar_k = \frac{1}{\alpha_k}(r_k - r_{k+1}) - \frac{\beta_{k-1}}{\alpha_{k-1}}(r_{k-1} - r_k), \\ Ar_k = -\frac{\beta_{k-1}}{\alpha_{k-1}} r_{k-1} + (\frac{1}{\alpha_k} + \frac{\beta_{k-1}}{\alpha_{k-1}}) r_k - \frac{1}{\alpha_k} r_{k+1}, \end{array}$$

d'où la relation (6.12). ◇

De ce lemme, nous déduisons l'équivalence entre le Gradient Conjugué et une méthode de Lanczos.

Théorème 6.3.3 *La méthode GC construit une base orthonormée* $V_k = (v_1, \ldots, v_k)$ *de l'espace de Krylov* $\mathcal{K}_k(A, r_0)$ *qui vérifie*

$$AV_k = V_k T_k + \delta_k v_{k+1} u_k^T \tag{6.13}$$

où $T_k \in \mathbb{R}^{k \times k}$ *est une matrice tridiagonale symétrique et où* $u_k^T = (0 \ldots 0\ 1) \in \mathbb{R}^k$.

Autrement dit, la méthode GC applique la méthode de Lanczos au vecteur de départ $v_1 = r_0/\|r_0\|$.

La méthode GC calcule $x_k = x_0 + V_k y$ *où* $y \in \mathbb{R}^k$ *est solution du système linéaire*

$$T_k y = \|r_0\|_2 u_1, \tag{6.14}$$

où $u_1^T = (1\ 0 \ldots 0) \in \mathbb{R}^k$.

Preuve. Posons

$$T_k = \begin{pmatrix} \gamma_1 & \delta_1 & & \\ \delta_1 & \gamma_2 & \delta_2 & \\ & \cdot & \cdot & \cdot \\ & & \delta_{k-1} & \gamma_k \end{pmatrix},$$

alors, d'après (6.12), $AV_k = V_k T_k + \delta_k v_{k+1} u_k^T$.

Cette propriété (6.13) est caractéristique de la méthode de Lanczos.

Soit $x_k = x_0 + V_k y$ alors $r_k = r_0 - AV_k y$ et $V_k^T r_k = 0$ s'écrit $V_k^T A V_k y = V_k^T r_0$. Or $V_k^T A V_k = T_k$ et $V_k^T r_0 = \|r_0\|_2 u_1$. ◇

Remarque 6.3.2 *La matrice* T_k *est symétrique définie positive, comme la matrice* A.

6.3.3 Convergence de GC

Nous avons vu la construction de la méthode du Gradient Conjugué. Nous allons maintenant étudier les propriétés de convergence.

Proposition 6.3.3 *Les coefficients β_k et α_k existent et sont uniques tant que $r_k \neq 0$.*

Preuve. Tant que $r_k \neq 0$, le coefficient β_k est bien défini et est unique. Tant que $r_k \neq 0$, on a aussi $p_k \neq 0$. La matrice A est définie positive donc $p_k^T A p_k \neq 0$ et le coefficient α_k existe et est unique. ◇

Remarque 6.3.3 *L'hypothèse que A est définie positive garantit l'existence et l'unicité de α_k.*

Il peut paraître paradoxal d'étudier la vitesse de convergence d'une méthode qui se termine en un nombre fini d'itérations. Néanmoins, en pratique, l'ordre n est très grand et la méthode du Gradient Conjugué fournit une approximation assez précise en beaucoup moins d'itérations. Nous établissons un premier résultat sur la décroissance des normes d'erreur. Il est à noter que c'est la norme A de l'erreur, donc la norme A^{-1} du résidu, qui décroît strictement.

Théorème 6.3.4 *Soit*

$$\kappa(A) = \|A\|_2 \|A^{-1}\|_2$$

le conditionnement spectral de A.

La méthode GC a une convergence strictement monotone, plus précisément

$$\|e_{k+1}\|_A \leq (1 - \frac{1}{\kappa(A)})^{1/2} \|e_k\|_A. \tag{6.15}$$

Preuve.

$r_{k+1} = r_k - \alpha_k A p_k$,
$\|e_{k+1}\|_A^2 = e_{k+1}^T A e_{k+1} = r_{k+1}^T A^{-1} r_{k+1} = r_k^T A^{-1} r_k - 2\alpha_k r_k^T p_k + \alpha_k^2 p_k^T A p_k$
or $p_k^T A p_k = \frac{r_k^T r_k}{\alpha_k}$
et $r_k^T p_k = r_k^T r_k$
d'où $\|e_{k+1}\|_A^2 = \|e_k\|_A^2 - \alpha_k \|r_k\|_2^2$.

Nous allons minorer successivement $\|r_k\|_2^2$ et α_k. Nous avons

$\|e_k\|_A^2 = r_k^T A^{-1} r_k \leq \|A^{-1}\|_2 \|r_k\|_2^2$.
D'autre part,
$p_k^T A p_k = (r_k + \beta_{k-1} p_{k-1})^T A p_k = r_k^T A p_k = r_k^T A r_k + \beta_{k-1} r_k^T A p_{k-1}$
or $r_k^T A p_{k-1} = (p_k - \beta_{k-1} p_{k-1})^T A p_{k-1} = -\beta_{k-1} p_{k-1}^T A p_{k-1} \leq 0$
donc $p_k^T A p_k \leq r_k^T A r_k \leq \|A\|_2 \|r_k\|_2^2$ et $\alpha_k = \frac{\|r_k\|_2^2}{p_k^T A p_k} \geq \frac{1}{\|A\|_2}$.

On en conclut que

$\alpha_k \|r_k\|_2^2 \geq \frac{1}{\|A\|_2 \|A^{-1}\|_2} \|e_k\|_A^2$
donc $\|e_{k+1}\|_A^2 \leq (1 - \frac{1}{\kappa(A)}) \|e_k\|_A^2$.

◇

Le facteur de décroissance dépend du conditionnement. Si celui-ci est grand, le coefficient devient proche de 1 et la convergence risque d'être lente.

Nous allons maintenant établir une majoration de l'erreur améliorant le résultat ci-dessus. Pour cela, nous allons exploiter la propriété de minimisation de la norme de l'erreur et le caractère polynomial de la méthode.

Théorème 6.3.5 *Les itérations de GC vérifient*

$$\|e_k\|_A = \min_{q \in \mathcal{P}_k^0} \|q(A)e_0\|_A.$$

Preuve. L'ensemble $\{\|q(A)e_0\|_A,\ q \in \mathcal{P}_k^0\}$ n'est rien d'autre que l'ensemble $\{\|b - Ay\|_{A^{-1}},\ y \in x_0 + \mathcal{K}_k(A, r_0)\}$. La propriété de minimisation est donc équivalente sur l'espace de Krylov et sur l'espace des polynômes résiduels. ◇

Nous pouvons maintenant traduire cette propriété sous la forme dite "min-max", en utilisant la décomposition de A en valeurs propres.

Corollaire 6.3.1 *Les itérations du GC vérifient*

$$\|e_k\|_A \leq \|e_0\|_A \min_{q \in \mathcal{P}_k^0} \max_{z \in \sigma(A)} |q(z)|, \tag{6.16}$$

où $\sigma(A)$ est l'ensemble des valeurs propres $0 < \lambda_1 \leq \ldots \leq \lambda_n$ de A.

Preuve. Soit $A = U\Sigma U^{-1}$ la décomposition spectrale de A, où

- Σ est la matrice diagonale $diag(\lambda_1, \ldots, \lambda_n)$, avec $\{\lambda_i,\ i = 1, \ldots, n\}$ les valeurs propres strictement positives de A,
- U est la matrice orthogonale de colonnes les vecteurs propres de A.

On a $A^m = U\Sigma^m U^{-1}$ et $q(A) = Uq(\Sigma)U^{-1}$.

Soit $e_0 = \sum_{i=1}^n \mu_i u_i$ alors $\|e_0\|_A^2 = \sum_{i=1}^n \mu_i^2 \lambda_i$.

En outre, $q(A)e_0 = \sum_{i=1}^n q(\lambda_i)\mu_i u_i$ d'où $\|q(A)e_0\|_A \leq \max_i |q(\lambda_i)| \|e_0\|_A$; en utilisant le théorème 6.3.5, on en déduit la relation (6.16). ◇

La propriété "min-max" permet d'utiliser la théorie de l'approximation sur les polynômes et de calculer une majoration de l'erreur.

Corollaire 6.3.2 *Les itérations de GC vérifient*

$$\|e_k\|_A \leq 2\|e_0\|_A \left(\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1}\right)^k, \tag{6.17}$$

où $\kappa(A) = \frac{\lambda_n}{\lambda_1}$ est le conditionnement de A.

Preuve. Nous utilisons, sans le démontrer, le fait que

$$\min_{q \in \mathcal{P}_k^0} \max_{z \in [\lambda_1, \lambda_n]} |q(z)| = \frac{1}{|C_k(\frac{\lambda_n + \lambda_1}{\lambda_n - \lambda_1})|},$$

où C_k est le polynôme de Chebyshev de première espèce de degré k. Voir par exemple [35]. D'où le résultat du théorème. ◇

Remarque 6.3.4 *La borne (6.17) est meilleure que la borne (6.15). En effet,*

$$\frac{\sqrt{\kappa(A)}-1}{\sqrt{\kappa(A)}+1} \leq \frac{\sqrt{\kappa(A)-1}}{\sqrt{\kappa(A)}}$$

car $\sqrt{\kappa(A)}-1 \leq \sqrt{\kappa(A)-1}$.

6.3.4 Convergence superlinéaire

Nous avons vu que la convergence du Gradient Conjugué est liée aux valeurs propres. D'autre part, les valeurs propres de la matrice T_k sont les valeurs de Ritz et convergent vers les valeurs propres de A (voir méthode de Lanczos). Lorsqu'une valeur extrémale du spectre a convergé, les itérations du Gradient Conjugué se poursuivent comme s'il y avait eu une déflation dans le spectre de A. Tout se passe comme si le conditionnement était plus petit, par exemple λ_{n-1}/λ_1 et la convergence s'accélère. Puis une autre valeur de Ritz converge vers λ_{n-1} et la convergence s'accélère de nouveau. Ce phénomène s'appelle convergence superlinéaire. Il est étudié en détails dans [37, 41].

6.3.5 Gradient Conjugué Préconditionné

Soit A une matrice symétrique définie positive et C une matrice inversible. Les systèmes préconditionnés définis dans 6.2.4 ne sont pas symétriques.

Soit maintenant C une matrice symétrique définie positive et $C = LL^T$ sa factorisation de Cholesky (qui existe). Un moyen de préserver la symétrie est de considérer le système préconditionné

$$L^T AL(L^{-1}x) = L^T b,$$

qui s'écrit

$$By = c,$$

avec $B = L^T AL$ et $y = L^{-1}x,\ c = L^T b$. La matrice B est symétrique définie positive donc il est possible d'appliquer la méthode du Gradient Conjugué au système $By = c$.

Maintenant, par un changement de variable, il est facile d'écrire l'algorithme du Gradient Conjugué appliqué à $By = c$ sous la forme suivante :

ALGORITHM 10: PCG

* Initialisation ;
choisir x_0 ;
$r_0 = b - Ax_0$;
$z_0 = Cr_0$;
$p_0 = z_0$;
* Iterations ;
for $k = 0, 1, \ldots$ **until** convergence **do**
$\alpha_k = \frac{r_k^T z_k}{p_k^T A p_k}$;
$x_{k+1} = x_k + \alpha_k p_k$;
$r_{k+1} = r_k - \alpha_k A p_k$;
$z_{k+1} = C r_k$;
$\beta_k = \frac{r_{k+1}^T z_{k+1}}{r_k^T z_k}$;
$p_{k+1} = z_{k+1} + \beta_k p_k$;
end do

La méthode du Gradient Conjugué est définie par la condition de Galerkin liée au produit scalaire euclidien. Mais il est également possible de définir la même condition en utilisant un autre produit scalaire.

Puisque la matrice C est symétrique définie positive, elle définit le produit scalaire

$$< x, y >_C = x^T C y \tag{6.18}$$

Maintenant, la matrice AC est auto-adjointe définie positive pour le produit scalaire (6.18). Donc il est possible d'appliquer l'algorithme du Gradient Conjugué avec le produit scalaire (6.18) au système préconditionné $AC(C^{-1}x) = b$.

Il est facile de voir que l'algorithme obtenu est identique à l'algorithme 10.

L'algorithme 10 est appelé l'algorithme du Gradient Conjugué Préconditionné par C.

Des hypothèses plus générales sur le préconditionnement C sont étudiées dans [2, 7].

6.4 Propriétés des méthodes de projection

Nous allons maintenant étudier les propriétés des méthodes de projection de Krylov et retrouver celles du Gradient Conjugué. Cette partie nous permettra de définir et de caractériser les méthodes de projection de Krylov dans le cas symétrique indéfini et dans le cas non symétrique.

Définition 6.4.1 *Pour x_0 donné, une méthode de projection de Krylov échoue à l'étape k si la condition de Galerkin (6.7) n'a pas une solution unique.*

Théorème 6.4.1 *Soit V_k une base orthonormée de l'espace de Krylov $\mathcal{K}_k(A, r_0)$ et soit $C_k = V_k^T B V_k$. La méthode de projection de Krylov n'échoue pas à l'étape k pour tout x_0 si et seulement si C_k est non singulière.*

Si la méthode n'échoue pas à l'étape k, alors

$$e_k = P_k e_0 \text{ où } P_k = I - V_k C_k^{-1} V_k^T B$$

est la matrice de la projection sur $(B^T \mathcal{K}_k(A, r_0))^\perp$ parallèlement à $\mathcal{K}_k(A, r_0)$.

L'itérée x_k est définie par $x_k = x_0 + V_k y$ où y est solution du système linéaire

$$C_k y = V_k^T B e_0. \tag{6.19}$$

Si la méthode n'a pas échoué jusqu'à l'étape k et si $\mathcal{K}_{k+1}(A, r_0) = \mathcal{K}_k(A, r_0)$, alors la méthode a convergé, donc $r_k = e_k = 0$.

Tant que la méthode n'échoue pas et ne converge pas, alors $dim(\mathcal{K}_k(A, r_0) = k$.

Si la méthode n'échoue pas, alors elle converge en au plus n itérations.

Preuve. La condition d'espace s'écrit $x_k = x_0 + V_k y$ et la condition de Galerkin s'écrit $V_k^T B e_k = 0$, soit $V_k^T B e_0 - C_k y = 0$. Ce système linéaire a une solution unique pour tout e_0 si et seulement si C_k est non singulière.

Si C_k est inversible alors

$$\begin{aligned} y &= C_k^{-1} V_k^T B e_0, \\ e_k &= e_0 - V_k y = (I - V_k C_k^{-1} V_k^T B) e_0 = P_k e_0. \end{aligned}$$

Si $\mathcal{K}_{k+1}(A, r_0) = \mathcal{K}_k(A, r_0)$, alors $A^k r_0 \in \mathcal{K}_k(A, r_0)$ et $A^{-1} r_0 \in \mathcal{K}_k(A, r_0)$, donc $x^* = x_0 + A^{-1} r_0$ est solution de la condition de Galerkin. Si la méthode n'a pas échoué, cette solution est unique donc $x_k = x^*$.

La suite des espaces de Krylov est croissante, donc elle est stationnaire à partir d'un certain rang $k \leq n$ et et dans ce cas la méthode a convergé. ◇

Théorème 6.4.2 *Si la matrice B est définie, la méthode de projection de Krylov associée n'échoue pas pour tout x_0.*

Si B n'est pas définie, il existe x_0 tel que la méthode de projection de Krylov associée échoue.

Si B est symétrique définie positive, alors la convergence est monotone : $\|e_k\|_B \leq \|e_{k-1}\|_B$.

Si de plus, BA^{-1} est définie, alors la convergence est strictement monotone : il existe $\epsilon > 0$, tel que pour tout k,

$$\|e_k\|_B \leq (1 - \epsilon) \|e_{k-1}\|_B.$$

Preuve. Si B est définie alors $\forall z \neq 0,\ V_k z \neq 0,\ (V_k z)^T B (V_k z) \neq 0$ càd $z^T C_k z \neq 0$ donc $C_k z \neq 0$ et C_k est inversible.

Si B n'est pas définie, soit $z \neq 0$ tel que $z^T B z = 0$ et soit $r_0 = z$ et $V_1 = \{r_0 / \|r_0\|\}$. Alors $C_1 = 0$ et le système (6.19) n'a pas une solution unique (il peut avoir une infinité de solutions si $r_0^T B A r_0 = 0$).

Si B est symétrique définie positive, la condition de Galerkin est aussi une condition de minimisation :

$$\|e_k\|_B = \min_{x \in x_0 + \mathcal{K}_k(A, r_0)} \|x - x^*\|_B.$$

Il suffit d'appliquer cette condition à x_{k-1} pour obtenir la convergence monotone.

La preuve de la convergence strictement monotone est faite dans [24]. ◇

Remarque 6.4.1 *Pour la méthode GC, $B = A$ donc B est symétrique définie positive et $BA^{-1} = I$ est définie. On retrouve que la méthode n'échoue pas et a une convergence strictement monotone.*

Dans la méthode GC, il est possible de construire une base $A-$orthonormée de l'espace de Krylov à l'aide d'une récurrence courte. Les théorèmes prouvés dans [12, 13] donnent une caractérisation des méthodes pour lesquelles une telle récurrence existe. Nous donnons ici une condition suffisante.

Théorème 6.4.3 *Si B et BA sont symétriques, alors il existe une récurrence courte permettant de calculer une base "$B-$orthogonale" de l'espace de Krylov.*

Preuve. Nous faisons la preuve par récurrence. Pour $k = 0$, le choix $p_0 = r_0$ convient. Supposons qu'il existe une base $B-$orthonormée $(p_0, \ldots, p_{k-1})$ de $\mathcal{K}_k(A, r_0)$. Nous cherchons p_k sous la forme

$$p_k = r_k + \sum_{i=1}^{k-1} \beta_i p_i.$$

Puisque B est symétrique, $(Bp_k)^T p_j = p_k^T (Bp_j)$. Alors

$p_k^T B p_j = r_k^T B p_j + \sum_{i=1}^{k-1} p_i^T B p_j$,
or $p_i^T B p_j = 0,\ j \leq k-2, j \neq i,$ par récurrence,
et $r_k^T B p_j = (Ae_k)^T B p_j = e_k^T (BA)^T p_j = e_k^T B(Ap_j),$ car B et BA sont symétriques,
or $Ap_j \in \mathcal{K}_k(A, r_0),\ j \leq k-2,$
donc $r_k^T B p_j = 0,\ j \leq k-2,$
d'où $\beta_j = 0,\ j \leq k-2,$
et $p_k = r_k + \beta_{k-1} p_{k-1}$.

◇

6.5 Cas où A est symétrique indéfinie

Lorsque A est symétrique mais indéfinie, on peut choisir $B = A$ pour définir une méthode symétrique mais on ne garantit pas l'absence d'échec ou choisir $B = A^2$ qui est symétrique définie positive (donc pas d'échec et convergence monotone due à une propriété de minimisation). Dans les deux cas, il est possible d'utiliser la méthode de Lanczos puisque A est symétrique. Il existe plusieurs méthodes, détaillées par exemple dans [4] ; les méthodes SYMMLQ et MINRES sont dues à [30].

6.5.1 Méthode de Lanczos

Le procédé de Lanczos construit une base orthonormée V_k de l'espace de Krylov $\mathcal{K}_k(A, r_0)$. Soit $v_1 = r_0/\|r_0\|_2$, on a

$$AV_k = V_k T_k + \delta_k v_{k+1} u_k^T, \tag{6.20}$$

où $T_k \in \mathbb{R}^{k,k}$ est une matrice tridiagonale symétrique. On peut aussi l'écrire

$$AV_k = V_{k+1} \overline{T}_k \tag{6.21}$$

où

$$\overline{T}_k = \begin{pmatrix} T_k \\ \delta_k u_k^T \end{pmatrix}.$$

Remarque 6.5.1 *La méthode de Lanczos construit la même suite (x_k) que le gradient conjugué, mais elle le fait à partir de la relation :*

$$x_k \quad = \quad x_0 + \|r_0\| V_k {T_k}^{-1} e_1. \tag{6.22}$$

L'itéré n'est pas défini à l'étape k lorsque T_k est singulière (voir le théorème 6.4.1, ce qui peut arriver puisque la matrice A n'est pas supposée être définie. Le Gradient Conjugué, qui revient à factoriser au fur et à mesure la matrice T_k, s'arrête donc à la même étape. En fait, le procédé de Lanczos peut malgré tout être poursuivi et donc l'itéré pourra peut-être être construit à une étape ultérieure. Ce dépassement de l'obstacle n'est pas possible avec l'algorithme du Gradient Conjugué.

6.5.2 Méthode MINRES

On a $e_k^T A^2 y = r_k^T A y$, donc pour $B = A^2$, la condition de Galerkin s'écrit, d'après la proposition 6.2.4,

$$\min \|r_k\|_2.$$

La condition d'espace s'écrit $x_k = x_0 + V_k y$ d'où

$$r_k = r_0 - AV_k y = \|r_0\|_2 v_1 - V_{k+1}\overline{T}_k y = V_{k+1}(\|r_0\|_2 u_1 - \overline{T}_k y)$$

et la condition de Galerkin devient

$$\min_{y \in \mathbb{R}^k} \|\|r_0\|_2 u_1 - \overline{T}_k y\| \tag{6.23}$$

La méthode MINRES résout (6.23) en factorisant $\overline{T}_k$ à l'aide de rotations de Givens. Cette factorisation peut s'effectuer à l'aide de récurrences courtes, car T_k est symétrique. La généralisation au cas non symétrique est la méthode GMRES décrite plus loin.

6.5.3 Méthode CR

La méthode MINRES utilise la propriété de minimisation. La méthode CR, qui est aussi basée sur $B = A^2$, utilise quant à elle la propriété d'orthogonalité. Les deux méthodes construisent la même suite d'itérés (x_k). La condition de Galerkin s'écrit $e_{k+1}^T A^T A r_j = 0,\ j \leq k$ soit

$$r_{k+1}^T A r_j = 0,\ j \leq k, \tag{6.24}$$

autrement dit, les résidus sont $A-$conjugués. Les vecteurs de descente p_k sont A^2-orthonormés donc les vecteurs Ap_k sont orthogonaux au sens classique.

La méthode CR (Conjugate Residuals) utilise les récurrences courtes comme dans le Gradient Conjugué pour garantir $r_{k+1}^T A r_k = 0$ et $(Ap_{k+1})^T A p_k = 0$. Elle applique la méthode de Lanczos pour construire la base orthonormée $(Ap_0/\|Ap_0\|_2, \ldots, Ap_{k-1}/\|Ap_{k-1}\|_2)$ de l'espace de Krylov $A\mathcal{K}_k(A, r_0)$.

6.5.4 Méthode SYMMLQ

Ici, on choisit $B = A$ comme dans le Gradient Conjugué. La condition de Galerkin s'écrit donc comme dans le Gradient Conjugué

$$x_k = x_0 + V_k y,\ T_k y = \|r_0\|_2 u_1. \tag{6.25}$$

La méthode SYMMLQ résout le système linéaire (6.25) comme dans la méthode de Lanczos, et donc comme dans le Gradient Conjugué, sans propriété de minimisation, puisque la matrice A n'est pas supposée être définie.

Tant qu'il n'y a pas d'échec, si le procédé de Lanczos s'arrête, la méthode SYMMLQ a convergé. En effet, les espaces de Krylov deviennent stationnaires et on peut appliquer le théorème 6.4.1. C'est un cas d'échec "heureux".

Il peut y avoir échec "sérieux" si la matrice T_k est singulière. En effet, dans le théorème 6.4.1, la matrice $C_k = V_k^T A V_k$ vaut T_k. La méthode SYMMLQ utilise une factorisation LQ de T_k pour éviter les situations d'échec.

6.6 Cas où A est non symétrique - méthodes de type gradient conjugué

Si A est non symétrique, on peut préconditionner le système (6.1) pour se ramener à un système symétrique défini positif. C'est l'idée des méthodes dites des équations normales. Voir par exemple [35, 2, 4].

6.6.1 Méthode CGNR

En préconditionnant à gauche par A^T, on obtient

$$A^T A x = A^T b, \tag{6.26}$$

sur lequel on peut appliquer l'algorithme du Gradient Conjugué. La méthode calcule alors

$$x_k \in x_0 + \mathcal{K}_k(A^T A, A^T r_0)$$

tel que

$$\|r_k\|_2 = \min_{y \in x_0 + \mathcal{K}_k(A^T A, A^T r_0)} \|b - Ay\|_2.$$

On obtient ainsi la méthode dite CGNR, Gradient Conjugué appliqué aux équations normales qui minimise la norme euclidienne du résidu.

Une variante de CGNR, appelée LSQR, qui est très souvent utilisée pour résoudre les problèmes aux moindres carrés, est décrite dans [31].

6.6.2 Méthode CGNE

En préconditionnant à droite par A^T, on obtient

$$A A^T (A^{-T} x) = b, \tag{6.27}$$

sur lequel on peut appliquer l'algorithme du Gradient Conjugué. La méthode calcule alors

$$x_k \in x_0 + A^T \mathcal{K}_k(AA^T, r_0)$$

tel que

$$\|e_k\|_2 = \min_{y \in x_0 + A^T \mathcal{K}_k(AA^T, r_0)} \|x^* - y\|_2.$$

On obtient ainsi la méthode dite CGNE, Gradient Conjugué appliqué aux équations normales qui minimise la norme euclidienne de l'erreur.

6.6.3 Convergence de CGNR et CGNE

Les méthodes CGNR et CGNE ont les avantages de la méthode du Gradient Conjugué : récurrence courte, minimisation, convergence strictement monotone.

Par contre, dans la borne d'erreur (6.17), le conditionnement est celui de la matrice préconditionnée c'est-à-dire

$$\kappa(A^T A) = \kappa(AA^T) = \kappa(A)^2 = (\frac{\sigma_n}{\sigma_1})^2,$$

où $0 < \sigma_1 < \ldots < \sigma_n$ sont les valeurs singulières de A.

6.7 Cas où A est non symétrique - méthode GMRES

Une autre solution pour le cas non symétrique, comme pour le cas symétrique indéfini, est de choisir $B = A^T A$ qui est symétrique définie positive. La méthode de Krylov associée est la méthode GMRES [36].

Théorème 6.7.1 *La méthode GMRES est caractérisée par*

$$\begin{aligned} &x_k \in x_0 + \mathcal{K}_k(A, r_0), \\ &r_k \perp A\mathcal{K}_k(A, r_0), \end{aligned}$$

et cette condition est équivalente à

$$\|r_k\|_2 = \min_{x \in x_0 + \mathcal{K}_k(A, r_0)} \|b - Ax\|_2. \tag{6.28}$$

La méthode GMRES n'échoue pas (le problème (6.28) a une solution unique). De plus, la convergence est monotone et la solution est atteinte en au plus n itérations.

Preuve. Il suffit d'appliquer le théorème 6.4.2. ◇

Remarque 6.7.1 *Par contre, la matrice $BA = A^T A^2$ n'est pas symétrique et il n'est pas possible d'appliquer le théorème 6.4.3 pour définir une récurrence courte.*

Puisque la matrice $BA^{-1} = A^T$ n'est pas définie en général, la convergence peut ne pas être strictement monotone. Dans l'exemple ci-dessous, le résidu stagne jusqu'à l'itération n où il s'annule.

Exemple 6.7.1 *Soit $(u_1, \ldots, u_n)$ la base canonique de $\mathbb{R}^n$ et soit la matrice A définie par*

$$\begin{cases} Au_i = u_{i+1}, \ 1 \leq i \leq n-1, \\ Au_n = u_1. \end{cases}$$

Soit $b = u_1$. Le système linéaire $Ax = b$ a pour solution $x^* = u_n$.

Si $x_0 = 0$, les espaces de Krylov sont

$$\mathcal{K}(A, r_0) = eng\{u_1, \ldots, u_k\}, \ 1 \leq k \leq n-1.$$

Les itérés valent

$$\begin{cases} x_k = 0, \ r_k = u_1, \ 1 \leq k \leq n-1, \\ x_n = u_n. \end{cases}$$

Il y a stagnation durant $n-1$ itérations et convergence à la n^{ieme} itération.

La proposition suivante caractérise les situations de stagnation dans GMRES.

Proposition 6.7.1 *Si $\|r_{k+1}\|_2 = \|r_k\|_2$ alors $r_k^* A r_k = 0$.*

Réciproquement, si $r_k^* A r_k = 0$, alors $r_{k+1} = r_k$ ou $r_k = r_{k-1}$.

Preuve. Si $\|r_{k+1}\|_2 = \|r_k\|_2$, alors r_k est l'unique solution du problème (6.28) pour l'indice $k+1$ donc $r_k = r_{k+1}$ et $r_k \perp A\mathcal{K}_{k+1}(A, r_0)$; or $r_k \in \mathcal{K}_{k+1}(A, r_0)$ d'où $r_k \perp Ar_k$.

Réciproquement, si $r_k = 0$, alors $r_{k+1} = r_k = 0$. On peut donc supposer que $r_k \neq 0$. Alors l'espace de Krylov $\mathcal{K}_{k+1}(A, r_0)$ est de dimension $k+1$.

Supposons que $r_k^* A r_k = 0$.

On sait que $r_k \in \mathcal{K}_{k+1}(A, r_0)$, alors analysons deux situations possibles.

Si r_k n'est pas dans $\mathcal{K}_k(A, r_0)$ alors $eng\{\mathcal{K}_k(A, r_0), r_k\} = \mathcal{K}_{k+1}(A, r_0)$. Comme $r_k \perp A\mathcal{K}_k$ et $r_k \perp Ar_k$, on en déduit que $r_k \perp A\mathcal{K}_{k+1}$ donc que r_k est solution du problème (6.28) pour l'indice $k+1$ et par unicité, $r_{k+1} = r_k$.

Sinon, $r_k \in \mathcal{K}_k$ donc $x_k - x_0 \in \mathcal{K}_k, A(x_k - x_0) \in \mathcal{K}_k$ d'où, grâce à la proposition (6.2.3), $x_k - x_0 \in \mathcal{K}_{k-1}$. Par conséquent, r_k est solution du problème (6.28) à l'indice $k-1$ et par unicité, $r_k = r_{k-1}$. ◇

6.7.1 Lien avec la méthode d'Arnoldi

Dans le cas symétrique, les méthodes GC et MINRES sont liées au procédé de Lanczos. Ici, la méthode GMRES est liée au procédé d'Arnoldi.

Définition 6.7.1 *Tant que la dimension de l'espace de Krylov est maximale, le procédé d'Arnoldi construit une base orthonormée $V_{k+1} = (v_1, \ldots, v_{k+1})$ de l'espace de Krylov $\mathcal{K}_{k+1}(A, v_1)$. Cette base vérifie*

$$AV_k = V_k H_k + h_{k+1,k} v_{k+1} u_k^T = V_{k+1} \overline{H}_k \tag{6.29}$$

où $H_k \in \mathbb{R}^{k \times k}$ est une matrice de Hessenberg, $\overline{H}_k \in \mathbb{R}^{k+1 \times k}$ est définie par

$$\overline{H}_k = \begin{pmatrix} H_k \\ h_{k+1,k} u_k^T \end{pmatrix}$$

et où $u_k^T = (0 \ldots 0 \ 1) \in \mathbb{R}^k$.

La méthode GMRES peut maintenant être construite grâce à la méthode d'Arnoldi.

Théorème 6.7.2 *Si la méthode GMRES n'a pas convergé, le problème (6.28) est équivalent au problème*

$$\min_{y\in\mathbb{R}^k} (\|r_0\|_2 u_1 - \overline{H}_k y) \tag{6.30}$$

où V_{k+1} et $\overline{H}_k$ sont la base et la matrice construites par le procédé d'Arnoldi appliqué à $v_1 = \frac{r_0}{\|r_0\|_2}$.

Preuve. Le système (6.19) est équivalent au problème (6.28). La matrice $C_k = V_k^T B V_k$ vaut ici $(AV_k)^T(AV_k) = \overline{H}_k^T \overline{H}_k$ car V_{k+1} est un système orthonormé.

Le second membre du système (6.19) vaut $V_k^T B e_0 = \overline{H}_k^T V_{k+1}^T r_0 = \|r_0\|_2 \overline{H}_k^T u_1$ car v_1 est orthogonal à $v_i, i \geq 2$.

Résoudre le système (6.19) équivaut donc à résoudre le système

$$\overline{H}_k^T \overline{H}_k y = \|r_0\|_2 \overline{H}_k^T u_1,$$

qui est le système des équations normales associé au problème aux moindres carrés (6.30). ◇

Il reste à choisir une méthode de résolution de (6.30). La méthode GMRES, telle qu'elle est définie dans [36] et couramment utilisée, utilise une factorisation QR de la matrice $\overline{H}_k$ par des rotations de Givens. Il est alors possible de calculer $\|r_k\|_2$ sans calculer x_k et d'avoir un critère d'arrêt simple.

6.7.2 Convergence de GMRES

Comme pour GC, la propriété de minimisation de GMRES peut se traduire sous forme polynomiale.

Théorème 6.7.3 *Si A est diagonalisable, soit $A = U\Sigma U^{-1}$, où les colonnes de U forment une base de vecteurs propres et où $\Sigma = diag(\lambda_1, \ldots, \lambda_n)$ et soit $\kappa(U) = \|U\|_2\|U^{-1}\|_2$ le conditionnement de U.*

Les itérations de GMRES vérifient

$$\|r_k\|_2 \leq \|r_0\|_2 \ \kappa(U) \ \min_{q\in\mathcal{P}_k^0} \max_{z\in\sigma(A)} |q(z)|. \tag{6.31}$$

Preuve. La propriété de minimisation (6.28) est équivalente à la propriété

$$\|r_k\|_2 = \min_{q\in\mathcal{P}_k^0} \|q(A)r_0\|_2.$$

Soit $q \in \mathcal{P}_k^0$, alors $q(A) = Uq(\Sigma)U^{-1}$; soit $r_0 = U\mu$, alors $q(A)r_0 = Uq(\Sigma)\mu$.

Donc $\|q(A)r_0\|_2 \leq \|U\|_2\|q(\Sigma)\|_2\|\mu\|_2$.

Or $\mu = U^{-1}r_0$ d'où $\|\mu\|_2 \leq \|U^{-1}\|_2\|r_0\|_2$.

En outre $\|q(\Sigma)\|_2 = \max_{\lambda_i} |q(\lambda_i)|$, ce qui donne l'inégalité (6.31).

◇

6.7.3 Redémarrage de GMRES

L'inconvénient principal de GMRES est l'absence de récurrence. Le procédé d'Arnoldi requiert le stockage des vecteurs v_k et l'orthogonalisation du vecteur Av_{k+1} par le procédé de Gram-Schmidt modifié. Le nombre d'opérations, outre le produit par A, est donc en $O(nk^2)$. Pour limiter le coût, à la fois en mémoire et en temps de calcul, on utilise en pratique un redémarrage.

La méthode GMRES(m) effectue des cycles de m itérations de GMRES, en redémarrant avec la dernière approximation x_m. Cela permet de limiter le stockage à $O(m)$ vecteurs et de réduire le temps de calcul. Toutefois, le choix de m est délicat.

Remarque 6.7.2 *Les normes euclidiennes des résidus dans GMRES(m) décroissent mais il peut y avoir stagnation. Dans l'exemple 6.7.1, la méthode GMRES(m) stagne sans converger pour tout* $m < n$.

Voici un squelette de l'algorithme GMRES(m), où n'est pas détaillée la factorisation QR de la matrice $\overline{H}$ avec des rotations de Givens.

ALGORITHM 11: GMRES(m)

* Initialisation ;
choisir x_0 ;
$r_0 = b - Ax_0$;
* Iterations ;
until convergence **do**
  $v_1 = \frac{r_0}{\|r_0\|_2}$;
  * procédé d'Arnoldi ;
  for $j = 1, m$
    $w = Av_j$;
    for $i = 1, j$
      $h_{ij} = v_i^T w$;
      $w = w - h_{ij} v_i$;
    end for ;
    $h_{j+1,j} = \|w\|_2$;
    $v_{j+1} = w / h_{j+1,j}$;
    * problème aux moindres carrés
    $\overline{H}_j = Q_j R_j$;
    calculer $\|r_j\|_2$;
    test de convergence
  end for ;
  calculer y_m solution de $\min_y (\|r_0\|_2 e_1 - \overline{H}_m y)$;
  $x_m = x_0 + V_m y_m$;
  $r_m = b - Ax_m$;
  test de convergence
  $x_0 = x_m$; $r_0 = r_m$;
end do

6.8 Cas où A est non symétrique - méthodes de gradient bi-conjugué

Nous venons de voir que la méthode GMRES a de bonnes propriétés de minimisation mais nécessite un redémarrage car elle ne possède pas de récurrence courte. A l'opposé, les méthodes de type gradient bi-conjugué utilisent une récurrence courte mais n'ont pas de propriété de minimisation et sont susceptibles d'échouer. La méthode BICG est une méthode de projection de Krylov, les variantes CGS, BICGSTAB et QMR ne sont plus des méthodes de projection. Alors que la méthode GMRES est liée au procédé d'Arnoldi, les méthodes bi-conjuguées sont connectées à la méthode de Lanczos non symétrique. Pour plus de détails sur ces méthodes, voir par exemple [35, 4, 7, 28, 19].

6.8.1 Construction de BICG

La méthode du gradient bi-conjugué (BICG) résout le système augmenté

$$\begin{pmatrix} A & 0 \\ 0 & A^T \end{pmatrix} \begin{pmatrix} x \\ \tilde{x} \end{pmatrix} = \begin{pmatrix} b \\ \tilde{b} \end{pmatrix}$$

par la méthode de projection de Krylov où la matrice B est choisie égale à

$$\begin{pmatrix} 0 & A^T \\ A & 0 \end{pmatrix}$$

Proposition 6.8.1 *La méthode BICG est définie par les choix de x_0 et de $\tilde{x}_0$ et par les conditions d'espace et de Galerkin suivantes :*

$$\begin{aligned} &x_k \in x_0 + \mathcal{K}_k(A, r_0), \\ &\tilde{x}_k \in \tilde{x}_0 + \mathcal{K}_k(A^T, \tilde{r}_0), \\ &r_k \perp \mathcal{K}_k(A^T, \tilde{r}_0), \\ &\tilde{r}_k \perp \mathcal{K}_k(A, r_0). \end{aligned}$$

La méthode risque d'échouer.

La méthode BICG possède une récurrence courte.

Preuve. Soit

$$\tilde{A} = \begin{pmatrix} A & 0 \\ 0 & A^T \end{pmatrix}.$$

L'espace de Krylov associé est

$$\begin{pmatrix} \mathcal{K}_k(A, r_0) \\ \mathcal{K}_k(A^T, \tilde{r}_0) \end{pmatrix},$$

ce qui donne bien la condition d'espace. De plus,

$$B\tilde{A}^{-1} = \begin{pmatrix} 0 & I \\ I & 0 \end{pmatrix},$$

ce qui donne bien la condition de Galerkin.

La matrice B n'est pas définie donc la méthode risque d'échouer, d'après le théorème 6.4.2.

Les matrices B et $B\tilde{A}$ sont symétriques, la méthode possède donc une récurrence courte, d'après le théorème 6.4.3. ◇

L'algorithme BICG est construit de la façon suivante.

ALGORITHM 12: BICG

* Initialisation ;
choisir x_0 et $\tilde{b}$ et $\tilde{x}_0$;
$r_0 = b - Ax_0$; $\tilde{r}_0 = \tilde{b} - A\tilde{x}_0$;
$p_0 = r_0$; $\tilde{p}_0 = \tilde{r}_0$;
* Iterations ;
for $k = 0, 1, \ldots$ **until** convergence **do**
$\alpha_k = \frac{\tilde{r}_k^T r_k}{\tilde{p}_k^T A p_k}$;
$x_{k+1} = x_k + \alpha_k p_k$;
$r_{k+1} = r_k - \alpha_k A p_k$;
$\tilde{r}_{k+1} = \tilde{r}_k - \alpha_k A^T \tilde{p}_k$;
$\beta_{k+1} = \frac{\tilde{r}_{k+1}^T r_{k+1}}{\tilde{r}_k^T r_k}$;
$p_{k+1} = r_{k+1} + \beta_{k+1} p_k$;
$\tilde{p}_{k+1} = \tilde{r}_{k+1} + \beta_{k+1} \tilde{p}_k$;
end do

Les résidus et les directions de descente vérifient les conditions d'orthogonalité suivantes :

$$r_{k+1}^T \tilde{r}_k = 0, \; A p_{k+1}^T \tilde{p}_k = p_{k+1}^T A^T \tilde{p}_k = 0.$$

6.8.2 Lien avec Lanczos non symétrique

Théorème 6.8.1 *La méthode BICG construit deux bases $V_k = (v_1, \ldots, v_k)$ et $W_k = (w_1, \ldots, w_k)$ des espaces de Krylov $\mathcal{K}_k(A, r_0)$ et $\mathcal{K}_k(A^T, \tilde{r}_0)$ qui vérifient*

$$\begin{aligned} AV_k &= V_k T_k + \delta_k v_{k+1} u_k^T, \\ A^T W_k &= W_k T_k^T + \gamma_k w_{k+1} u_k^T, \\ V_k^T W_k &= I, \end{aligned} \tag{6.32}$$

où $T_k \in \mathbb{R}^{k \times k}$ est une matrice tridiagonale et où $u_k^T = (0 \ldots 0 \; 1) \in \mathbb{R}^k$.

Autrement dit, la méthode BIGC applique la méthode de Lanczos non symétrique, appelée aussi dans la suite méthode Bi-Lanczos, aux vecteurs de départ $v_1 = r_0/\|r_0\|_2$ et $w_1 = \mu \tilde{r}_0$ tel que $v_1^T w_1 = 1$.

La méthode BICG résout le système linéaire

$$T_k y = \|r_0\|_2 u_1.$$

Elle échoue si T_k est singulière.

Preuve. La preuve est la même que pour CG, où V_k et W_k sont les bases $(r_0, \ldots, r_{k-1})$ et $(\tilde{r}_0, \ldots, \tilde{r}_{k-1})$ normalisées de manière à avoir $V_k^T W_k = I$. Il y a échec si la matrice

$$C_k = \begin{pmatrix} 0 & T_k^T \\ T_k & 0 \end{pmatrix},$$

du théorème 6.4.2 est singulière, donc si la matrice T_k est singulière.

◇

Il existe des situations d'échec "heureux" dans la méthode Bi-Lanczos, comme pour les méthodes de Lanczos et d'Arnoldi.

Proposition 6.8.2 *Si la méthode Bi-Lanczos s'est poursuivie jusqu'à l'indice $k-1$ et s'arrête parce que $Av_k \in eng\{v_k, v_{k-1}\}$ (dans ce cas, $Aw_k \in eng\{w_k, w_{k-1}\}$), alors $\mathcal{K}_k(A, v_1) = \mathcal{K}_{k+1}(A, v_1)$ et $\mathcal{K}_k(A^T, w_1) = \mathcal{K}_{k+1}(A^T, w_1)$.*

Preuve. Par hypothèse, il n'y pas eu d'échec auparavant et on peut appliquer le théorème 6.4.1. ◇

Cet arrêt est un échec "heureux" pour la méthode BICG puisqu'alors la méthode a convergé.

6.8.3 Convergence de BICG

La méthode BICG présente l'avantage de posséder une récurrence courte. Par contre, elle requiert à chaque itération deux produits matrice-vecteur, par A et A^T. De plus, elle peut échouer (échec "sérieux") si la matrice T_k est singulière. Enfin, le résidu ou l'erreur ne vérifient pas de propriété de minimisation, on ne peut pas prouver une convergence monotone. On observe effectivement dans de nombreux cas une convergence très irrégulière.

Pour éviter les échecs dans la méthode Bi-Lanczos, il est possible d'utiliser une version dite "look-ahead" [32, 15]. L'idée est de construire plusieurs vecteurs à la fois qui sont bi-orthogonaux par blocs. Cette version est efficace sauf dans les cas d'échecs dits "incurables".

6.8.4 Variantes CGS et BICGSTAB

Pour éviter les produits par A^T, il est possible de modifier le polynôme sous-jacent dans BICG. C'est l'idée des méthodes CGS [38] et BICGSTAB [42]. Il est à noter que ces variantes de méthodes de Krylov ne sont plus des méthodes de projection.

6.9 Cas où A est non symétrique - méthode QMR

La méthode QMR permet d'éviter une convergence irrégulière. Ce n'est plus à proprement parler une méthode de projection de Krylov. Elle est basée aussi sur la méthode Bi-Lanczos et impose une condition de quasi-minimisation [16, 14]. Plus précisément, on a

$$\begin{aligned} x_k &= x_0 + V_k y, \\ r_k &= r_0 - AV_k y = V_{k+1}(\beta u_1 - \overline{T}_k y). \end{aligned}$$

Puisque V_{k+1} n'est pas un système orthogonal, minimiser $\|r_k\|$ serait trop coûteux. On définit le problème de quasi-minimisation

$$\min_y \|\Omega_{k+1}^{-1}(\gamma u_1 - \Omega_{k+1}\overline{T}_k y)\|_2 \tag{6.33}$$

où Ω_{k+1} est une matrice diagonale.

Ce problème est résolu par une factorisation QR de la matrice $\Omega_{k+1}\overline{T}_k$.

La méthode de Lanczos avec "look-ahead" permet d'éviter les situations d'échec. Une version "Transpose-Free" (TFQMR) ne requiert pas le produit par A^T. La convergence n'est pas monotone mais est plus régulière que pour les variantes de BICG. En outre, il existe un résultat de convergence.

6.10 Problèmes numériques dans les méthodes de Krylov

6.10.1 Perte d'orthogonalité et dérive du résidu

Les méthodes de Krylov reposent sur des conditions d'orthogonalité, qui sont souvent vérifiées par récurrence. De même, l'itéré x_k et le résidu r_k sont souvent calculés par deux récurrences, qui vérifient implicitement $r_k = b - Ax_k$. Ces conditions sont démontrées en arithmétique exacte, mais ne sont pas vérifiées en arithmétique flottante, à cause des erreurs d'arrondi générées. Cela se traduit par deux phénomènes [] :

- une perte d'orthogonalité ; par exemple, dans le Gradient Conjugué, les directions de descente ne sont plus conjuguées et les résidus ne sont plus orthogonaux. Cette perte d'orthogonalité engendre une irrégularité et un ralentissement de la convergence. Dans les versions matrix-free notamment, le produit par A engendre des erreurs d'arrondi assez grandes pour provoquer une perte d'orthogonalité. Celle-ci peut être corrigée par une réorthogonalisation totale ou partielle.
- un résidu calculé qui n'est plus égal à $b - Ax$; par exemple, dans le Gradient Bi-Conjugué, on observe une dérive entre r_k calculé et $b - Ax_k$. Par conséquent, le critère d'arrêt basé sur r_k n'est plus valide et les itérations risquent de s'arrêter avant réelle convergence. Ce problème peut être corrigé en recalculant régulièrement le résidu.

6.10.2 Breakdowns et near-breakdowns

Certaines méthodes de Krylov peuvent échouer, par exemple SYMMLQ ou BICG, lorsque la matrice tridiagonale du procédé de Lanczos ou de Bi-Lanczos devient singulière. De même, le procédé de Lanczos peut s'arrêter si l'espace de Krylov devient invariant (échec "heureux"). Numériquement, la situation se dégrade dès que la matrice de Lanczos devient proche de la singularité ou dès que l'espace de Krylov devient presque invariant. On parle alors de "near-breakdown".

Il faut en fait appliquer la méthode de Lanczos avec "look-ahead" dès qu'un problème numérique est détecté [].

6.11 Préconditionnement

En général, les méthodes de Krylov convergent trop lentement et il faut préconditionner le système. L'idée est de trouver une matrice C telle que CA soit mieux conditionnée que A et telle que le produit par C soit peu coûteux. L'idéal serait de choisir $C = A^{-1}$, aussi cherche-t-on à approcher l'inverse de A. Pour le Gradient Conjugué, un préconditionnement symétrique défini positif garantit de conserver les propriétés de la méthode. Il existe différentes façons de construire un préconditionnement. Nous donnons ci-dessous un bref aperçu, plus de détails peuvent se trouver dans [35, 28].

6.11.1 Décomposition de A

Les préconditionnements les plus simples sont basés sur les méthodes linéaires, donc sur une décomposition de A.

Le préconditionnement diagonal, dit aussi de Jacobi, consiste à choisir $C = D^{-1}$ où $D = diag(A)$.

Le préconditionnement SSOR consiste à choisir $C = (D+\omega L)D^{-1}(D+\omega U)$ où $A = D+L+U$, avec D diagonale, L triangulaire inférieure et U triangulaire supérieure. En général, on choisit $\omega = 1$.

Ces deux préconditionnements sont symétriques définis positifs dès que A l'est.

6.11.2 Factorisation incomplète

Les méthodes itératives sont souvent appliquées à des matrices creuses, dont une grande partie des coefficients sont nuls. Un des inconvénients des méthodes directes appliquées aux matrices creuses est le coût de stockage induit par le remplissage lors de la factorisation de Gauss ou de Cholesky, voir par exemple [33]. L'idée des factorisations incomplètes est de limiter le remplissage. On obtient alors

$$A = LU + R$$

et le préconditionnement est défini par $C = U^{-1}L^{-1}$.

Il existe diverses stratégies pour limiter le remplissage, mais la factorisation incomplète peut ne pas exister. Toutefois, elle existe pour toute stratégie si A est une $M-$matrice [27]. Le cas le plus simple est la méthode ILU(0) où aucun remplissage n'est autorisé.

Ces préconditionnements sont très souvent utilisés pour leur efficacité, bien qu'ils requièrent un stockage supplémentaire conséquent et un temps de calcul assez important.

6.11.3 Préconditionnement polynomial

Dans la mesure où les méthodes de Krylov sont des méthodes polynomiales, il paraît naturel de les préconditionner par des polynômes. L'objectif est de choisir un polynôme qui approche l'inverse de A. Plus précisément, on cherche $q \in \mathcal{P}_m$ tel que $\|1 - Xq(X)\|$ soit minimal pour une norme définie sur l'espace des polynômes. Il existe principalement deux choix, une norme uniforme et une norme aux moindres carrés, toutes deux définies sur un compact contenant le spectre de A [34].

L'avantage est un coût mémoire négligeable, mais l'inconvénient est une efficacité parfois réduite.

6.11.4 Inverse approché

Ici, il s'agit de trouver une matrice C qui minimise $\|I - CA\|$ ou $\|I - AC\|$ pour une norme à préciser. Ce type de préconditionnement peut s'avérer très efficace, mais coûteux à calculer. En outre, les conditions d'existence sont mal définies dans le cas non symétrique.

6.11.5 Multigrille et multiniveaux

Les méthodes multigrilles et multiniveaux sont des méthodes itératives qui peuvent être utilisées en soi. Mais, comme pour les méthodes linéaires de type Gauss-Seidel, il est possible de définir un préconditionnement à partir de ces méthodes. Par exemple, un $V-$cycle d'une méthode multigrille est un préconditionnement symétrique défini positif si A l'est.

L'intérêt de ces méthodes est de réduire notablement le conditionnement de A, lorsque A est issue d'une discrétisation d'EDP. Si h est le pas de discrétisation, le conditionnement passe souvent de $O(h^{-1})$ à $O(1)$.

6.11.6 Problèmes approchés

Les préconditionnements décrits jusqu'ici sont basés sur des concepts algébriques. Lorsque la matrice A est issue d'une discrétisation d'EDP, il peut être très efficace de construire un préconditionnement à partir d'une décomposition de l'EDP ou d'une EDP plus simple.

6.11.7 Déflation et systèmes augmentés

Le phénomène de convergence superlinéaire est à la base des méthodes de projection et de déflation pour accélérer la convergence d'une séquence de systèmes linéaires ou à chaque redémarrage de GMRES(m). L'idée est de calculer les valeurs de Ritz et les vecteurs de Ritz associés aux plus petites valeurs propres. Ces vecteurs engendrent un espace qui approche un espace invariant de A. Les méthodes de déflation et de systèmes augmentés définissent une projection associée à ce sous-espace [11].

Chapter 7

Cas des grands systèmes

7.1 Stockage des matrices creuses

Dans de nombreuses simulations numériques, la discrétisation du problème aboutit à manipuler une matrice de très grande taille (d'ordre pouvant aller jusqu'à 10^8) mais dont la plupart des coefficients sont nuls. Dans ce cas, on considère un stockage creux (par abus, on parle d'une matrice creuse) qui permet de ne pas stocker une grande partie des coefficients nuls et de réduire considérablement la complexité des résolutions de systèmes.

Les stockages courants sont :

le stockage bande : on stocke toutes les diagonales situées entre les deux diagonales contenant les éléments non nuls les plus éloignés de la diagonale principale comme colonnes d'une matrice rectangulaire dont le nombre de lignes est égal à l'ordre de la matrice creuse tandis que le nombre de colonnes de la matrice est égal à la largeur de la bande.

le stockage profil : Il peut arriver que les éléments non nuls extrêmes de chaque ligne soient à des distances de la diagonale très différentes suivant les lignes. Un stockage bande oblige alors à stocker un grand nombre d'éléments non nuls. Pour diminuer cet effet, on peut stocker les portions de chaque ligne situées entre ces éléments extrêmes.

le stockage par coordonnées (COO): dans un tableau à une dimension, on range tous les éléments non nuls de la matrice. Les indices de ligne et de colonne de chacun de ces éléments sont stockés dans le même ordre dans deux tableaux d'entiers à une dimension. C'est le stockage de référence dans MATLAB.

le stockage compact par lignes (CSR) : dans un tableau à une dimension on range tous les éléments non nuls par ligne, une ligne après l'autre. Les indices de colonnes et les limites de lignes sont retenus dans deux tableaux d'entiers.

le stockage compact par colonnes (CSC) : ce format est le même que le précédent, mais en remplaçant le rôle des lignes par celui des colonnes.

Il existe encore d'autres stockages compacts, par exemple par diagonales creuses ou en définissant des blocs.

La bibliothèque LAPACK permet le stockage bande. Beaucoup de solveurs directs ou itératifs utilisent un stockage compact, par lignes ou par colonnes.

7.2 Produit matrice-vecteur

Dans les méthodes itératives de résolution, les deux opérations les plus coûteuses sont le produit matrice-vecteur et le préconditionnement. Les stockages creux permettent de réduire la complexité du produit matrice-vecteur. Dans ce qui suit, on considère un stockage compact par lignes.

On suppose que la matrice A d'ordre n a $nz(A)$ éléments non nuls que l'on a rangés dans le tableau `a(1:nz)` en les énumérant par lignes. Dans le tableau `ja(1:nz)` on range les indices de colonnes de ces éléments dans le même ordre et dans le tableau `ia(1:n+1)` on indique la liste des indices des démarrages de lignes de la matrice A stockée dans `a` avec par convention la valeur `nz+1` dans `ia(n+1)`. Etant donné deux vecteurs $x,\ y\ \in \mathbb{R}^n$, l'opération

$$y := y + A\ x$$

s'exprime par le code FORTRAN suivant :

```
do i = 1, n
  do k = ia(i),ia(i+1)-1
    y(i) = y(i) + a(k) * x(ja(k))
  end
end
```

La boucle interne est un produit scalaire creux; il met en jeu un *gather* (lecture indirecte). Si on avait considéré le stockage compact par colonnes, on serait arrivé à une boucle comportant un *scatter* (écriture indirecte). Ces opérations sont optimisées sur les calculateurs d'aujourd'hui.

La complexité passe de $O(n^2)$ pour l'opération BLAS2 standard à $O(nz(A))$ pour le produit en mode creux.

7.3 Factorisation de Cholesky

Soit A une matrice symétrique définie positive d'ordre n, et $A = LL^T$. Du fait que la factorisation de Cholesky ne nécessite pas de pivot, donc pas de permutation des lignes ou colonnes en cours de calcul, il est possible de prévoir à l'avance la structure creuse du facteur de Cholesy L.

7.3.1 Stockages bande et profil

Le facteur de Cholesky d'une matrice bande est une matrice bande de même largeur de bande.

La complexité passe de $O(n^3)$ pour l'opération LAPACK en mode plein à $O(nl^2)$ en mode bande, où l est la demi-largeur de bande.

De même, le facteur de Cholesky d'une matrice profil a le même profil.

Il est donc intéressant de minimiser la largeur de bande d'une matrice ou de minimiser le profil. Des techniques de renumérotation, avec permutation symétrique des lignes et des colonnes, sont utilisées dans ce but. Soit P la matrice de permutation associée. On effectue alors la factorisation de Cholesky sur la matrice P^TAP, qui est toujours symétrique définie positive.

7.3.2 Remplissage dans le stockage compact

La structure creuse de L n'est pas celle de A : des éléments nuls dans A deviennent non nuls dans L. Considérons par exemple la matrice de droite dans la figure 7.1. Le facteur L est dans ce cas

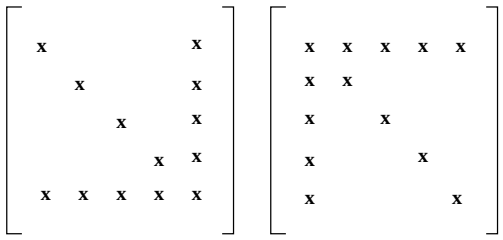

Figure 7.1: Deux types de matrices provoquant des remplissages différents.

une matrice pleine. Par contre, pour la matrice de gauche, la structure de L est la même. Entre ces deux cas extrêmes, il peut y avoir plus ou moins de remplissage.

Pour prédire la structure creuse en général, on utilise le graphe de la matrice et la notion de graphe d'élimination.

Definition 7.3.1 *Soit $A \in \mathbb{R}^{n\times n}$ une matrice symétrique creuse. On définit alors son graphe par :*

- *un ensemble de sommets : $X = \{1, \cdots, n\}$,*
- *un ensemble d'arcs : $G = \{(i,j) \in X^2 \mid i > j \text{ et } \alpha_{ij} \neq 0\}$*

Le degré d'un sommet $x \in X$ est égal au nombre de ses voisins dans le graphe, c'est-à-dire au nombre de sommets $y \in X$ tels que $(y,x) \in G$. Cela représente le nombre de coefficients non nuls dans la colonne d'indice x sous la diagonale de la matrice A.

Chaque étape de l'algorithme de Cholesky correspond à un processus d'élimination des inconnues. La figure 7.2 montre l'influence de cette élimination sur les inconnues suivantes dont les indices $i1,\ i2,\ i3$ correspondent aux indices de ligne des coefficients non nuls de la colonne j : cela crée les trois éléments indiqués sur la matrice par un gros point noir.

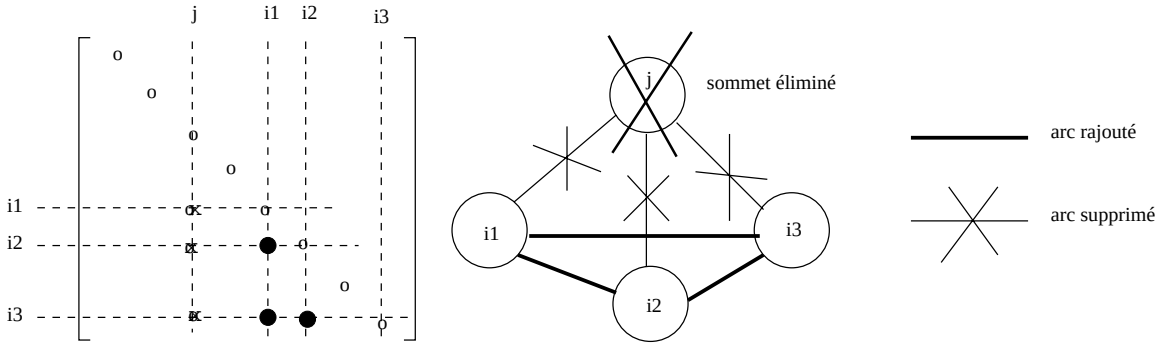

Figure 7.2: Remplissage après élimination d'un nœud.

Sur le graphe, cela correspond à supprimer les arcs entre j et ses voisins et à créer tous les arcs qui n'existent pas encore entre les voisins. On constate donc que dès que les voisins d'un sommet sont nombreux, son élimination entraîne un grand remplissage de la partie restante de la matrice.

7.3.3 Factorisation symbolique

Le processus complet conduit à la construction d'un arbre d'élimination. Avant d'effectuer les calculs, une phase préliminaire, basée sur cet arbre d'élimination, définit la structure creuse de L et prépare les tableaux pour le stockage compact.

7.3.4 Renumérotation

Les exemples précédents montrent que l'ordre d'élimination influe fortement le niveau de remplissage. Des techniques de renumérotation, basées sur une matrice de permutation P, visent à minimiser le remplissage.

Un algorithme efficace et beaucoup utilisé est l'algorithme du degré minimal. Le principe en est de construire par récurrence une suite de graphes selon la procédure suivante :

1. $(X_0, G_0) = (X, G)$
2. Construction de (X_{i+1}, G_{i+1}) à partir de (X_i, G_i) par :
   (a) trouver $x \in X_i$ de degré minimal dans G_i (solution non nécessairement unique)
   (b) permuter x et $i+1$;
   (c) construire G_{i+1}) à partir de G_i en éliminant $i+1$.

Il existe un autre algorithme qui a aussi de bons effets sur le remplissage : la dissection emboîtée. De plus, il a l'avantage d'introduire du parallélisme dans la factorisation et dans les résolutions des systèmes triangulaires qui en découlent.

L'algorithme est du type *diviser pour régner*. On suppose que la matrice A est irréductible, c'est-à-dire qu'il n'existe pas de permutation des indices qui décompose la matrice en deux ou plusieurs blocs diagonaux, ou encore dit autrement, que le graphe de la matrice A est connexe (il existe toujours un chemin entre deux sommets différents). Si A n'est pas irréductible, on applique l'algorithme à chacune des sous-matrices irréductibles qui la composent.

Le principe de l'algorithme est de partitionner récursivement le graphe. A chaque graphe (X, G) que l'on considère (le graphe entier à la première étape, des sous-graphes ensuite), on recherche une partie X_1 de X, appelée séparateur, telle que si on la supprime de (X, G) avec les arcs correspondants, on aboutisse à un graphe à deux composantes connexes et donc à deux sous-graphes indépendants. En numérotant les sommets des deux sous-graphes avant ceux du séparateur, on obtient une matrice de type flèche (par blocs) du type

$$\begin{pmatrix} A_1 & & C_1 \\ & A_2 & C_2 \\ {C_1}^t & {C_2}^t & B \end{pmatrix}.$$

Les étapes suivantes consistent à donner la même structure aux matrices A_1 et A_2 qui sont aussi des matrices symétriques définies positives si A l'est. Nous avons vu dans l'exemple de la figure 7.1 l'intérêt de telles structures.

7.4 Factorisation LU

Dans le cas non symétrique, il est nécessaire d'appliquer une permutation des colonnes ou des lignes pour garantir la stabilité numérique de la factorisation LU. En général, la stratégie est appliquée en cours de calcul. Le critère de choix du pivot est basé à la fois sur l'algorithme de degré minimal, pour minimiser le remplissage, et sur le pivot partiel, pour la stabilité numérique. Dans le code SuperLU, la stratégie est appliquée a priori, avant de lancer les calculs, ce qui permet d'effectuer une factorisation symbolique comme pour Cholesky.

Bibliography

[1] E. Anderson, z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide, Third Edition*. SIAM, 1999.

[2] S.F. Ashby, T.A. Manteuffel, and P.E. Saylor. A taxonomy for Conjugate Gradient Methods. *SIAM Journal on Numerical Analysis*, 26:1542–1568, 1990.

[3] J-C. Bajard, O. Beaumont, J-M. Chesneaux, M. Daumas, J. Erhel, D. Michelucci, J-M. Muller, B. Philippe, N. Revol, J-L. Roch, and J. Vignes. *Qualité des Calculs sur Ordinateurs. Vers des arithmétiques plus fiables?* Masson, 1997.

[4] R. Barret, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst. *Templates for the solution of linear systems: building blocks for iterative methods - 2nd edition*. SIAM / netlib, Philadelphia, PA, 1994.

[5] O. Beaumont. *Algorithmique pour les intervalles : Comment obtenir un résultat sûr quand les données sont incertaines*. thèse de doctorat, université de Rennes 1, January 1999.

[6] A. Björck. Numerics of gram-schmidt orthogonalization. *Linear Algebra Appl.*, 197/198:297–316, 1994.

[7] A.M. Bruaset. *A survey of preconditioned iterative methods*. Pitman Research Notes in Mathematics Series. Longman Scientific and Technical, 1995.

[8] F. Chatelin and V. Frayssé. *Lectures on Finite Precision Computations*. SIAM, 1995.

[9] Jean-Marie Chesnaux. *L'Arithmétique Stochastique et le Logiciel CADNA*. Habilitation à diriger des recherche, Université Paris VI, 1995.

[10] J. Erhel. *Vitesse et précision en calcul scientifique*. habilitation à diriger des recherches, Université de Rennes 1, Juin 1992.

[11] J. Erhel and F. Guyomarc'h. An augmented conjugate gradient method for solving consecutive symmetric positive definite systems. *SIAM Journal on Matrix Analysis and Applications*, 21(4):1279–1299, 2000.

[12] V. Faber and T. Manteuffel. Necessary and sufficient conditions for the existence of a conjugate gradient methods. *SIAM Journal on numerical analysis*, 21:352–362, 1984.

[13] V. Faber and T. Manteuffel. Orthogonal error methods. *SIAM journal on numerical analysis*, 24(1):170–187, 1987.

[14] R. Freund. A transpose-free quasi-minimal residual algorithm for non-hermitian linear systems. *SIAM journal on scientific computing*, 14:470–482, 1993.

[15] R. Freund, M. Gutknecht, and N. Nachtigal. An implementation of the look-ahead Lanczos algorithm for non-hermitian matrices. *SIAM journal on scientific computing*, 14:137–158, 1993.

[16] R. Freund and N. Nachtigal. QMR : a quasi-minimal residual method for non-Hermitian linear systems. *Numerische mathematik*, 60:315–339, 1991.

[17] K. Gallivan, W. Jalby, and U. Meier. The use of blas3 in linear algebra on a parallel processor with hierachical memory. *SIAM J. Sci. Stat. Comput.*, 1986.

[18] G.H Golub and C.F Van Loan. *Matrix Computations. third edition.* John Hopkins, 1996.

[19] M. Gutknecht. Lanczos-type solvers for nonsymmetric linear systems of equations. *Acta numerica*, 6:271–397, 1997.

[20] M. Hahad, J. Erhel, and T. Priol. Factorisation parallèle de Cholesky pour matrices creuses sur une mémoire virtuelle partagée. Rapport de recherche 1988, INRIA, 1993.

[21] M. Hahad, J. Erhel, and T. Priol. A new approach to parallel sparse Cholesky on DMPCs. In *International Conference on Parallel Processing, USA,*, Aout 1994.

[22] N.J. Higham. *Accuracy and Stability of Numerical Algorithms.* SIAM, 1995.

[23] W. Jalby and B. Philippe. Stability analysis and improvement of the block gram-schmidt algorithm. *SIAM J. Scient. and Stat. Comput.*, 12(5), 1991.

[24] W. D. Joubert and T. A. Manteuffel. *Iterative Methods for Nonsymmetric Linear Systems*, chapter 10, pages 149–171. Academic Press, 1990.

[25] Philippe Langlois. *Précision finie et méthodes automatiques.* Habilitation à diriger des recherche, Université de Lyon 1, 2001.

[26] P. Lascaux and R. Theodor. *Analyse numérique matricielle appliquée à l'art de l'ingénieur, Tomes 1 et 2.* Masson, 1986.

[27] 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.

[28] G. Meurant. *Computer solution of large linear systems.* North Holland, Amsterdam, 1999.

[29] J-M. Muller. *Elementary Functions, Algorithms and Implementation.* Birkhauser Boston,, 1997.

[30] C. Paige and M. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM journal on numerical analysis*, 12:617–629, 1975.

[31] C. Paige and M. Saunders. LSQR : an algorithm for sparse linear equations and sparse least squares. *ACM transactions on mathematical software*, 8:43–71, 1982.

[32] B. Parlett, D. Taylor, and Z. Liu. A look-ahead Lanczos algorithm for unsymmetric matrices. *Mathematics of computation*, 44:105–124, 1985.

[33] J Reid, I. Duff, and A. Erisman. *Direct methods for sparse matrices*. Oxford University Press, London, 1986.

[34] Y. Saad. Practical use of polynomial preconditioning for the conjugate gradient method. *SIAM J. Sci. Stat. Comput.*, 6(4):865–881, 1985.

[35] Y. Saad. *Iterative methods for sparse linear systems*. PWS Publishing Company, 1996.

[36] Y Saad and H Schultz. GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 7:856–869, 1986.

[37] 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.

[38] P. Sonnefeld. CGS, a fast Lanczos-type solver for nonsymmetric linear systems. *SIAM journal on scientific and statistical computing*, 10(36-52), 1989.

[39] American National Standard. Ieee standard for binary floating-point arithmetic. ANSI/IEEE Std. 754, IEEE, 1985.

[40] G.W. Stewart and Ji guang Sun. *Matrix Perturbation Theory*. Academic Press, 1990.

[41] A. van der Sluis and H. van der Vorst. the rate of convergence of conjugate gradients. *Numerische Mathematik*, 48:543–560, 1986.

[42] H. van der Vorst. Bi-CGSTAB: a fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM journal on scientific and statistical computing*, 13:631–644, 1992.

[43] R. S. Varga. *Matrix Iterative Analysis*. Prentice-Hall, Englewood Cliffs, 1962.