Sup-Galilée – MACS 2

Analyse numérique avancée

Michel KERN [1]

2020–2021
Version de février 2021

1. Inria, Centre de Paris, 75012 Paris, Michel.Kern@inria.fr

Table des matières

1 La méthode du gradient conjugué **6**
- 1.1 Notations – généralités . . . 6
- 1.2 Définition de la méthode . . . 9
- 1.3 L'algorithme . . . 10
  - 1.3.1 Développement de l'algorithme . . . 10
  - 1.3.2 Mise en oeuvre de l'algorithme . . . 13
- 1.4 Convergence . . . 14

2 Systèmes linéaires non symétriques : la méthode GMRES **20**
- 2.1 Définition de la méthode GMRES, et premières propriétés . . . 20
- 2.2 L'algorithme . . . 21
  - 2.2.1 L'algorithme d'Arnoldi . . . 21
  - 2.2.2 Résolution du problème de moindres carrés . . . 24
  - 2.2.3 Conséquences de l'arithmétique flottante . . . 25
- 2.3 Remarques sur la convergence . . . 27

3 Préconditionnement **29**
- 3.1 Généralités . . . 29
  - 3.1.1 À gauche, à droite, ou des deux cotés ? . . . 30
- 3.2 Préconditionnement du gradient conjugué . . . 31
- 3.3 Exemples de préconditionnement . . . 33
  - 3.3.1 Préconditionnement diagonal . . . 33
  - 3.3.2 Préconditionnnment par des méthodes itératives classiques . . . 34
  - 3.3.3 Factorisations incomplètes . . . 35
  - 3.3.4 Préconditionnement polynomial . . . 39

4 Matrices creuses **42**
- 4.1 Introduction . . . 42
- 4.2 Stockage par diagonales . . . 44
  - 4.2.1 Description . . . 44
  - 4.2.2 Algorithmes . . . 45
- 4.3 Stockage CSR . . . 46
  - 4.3.1 Stockage « coordonnées » . . . 46
  - 4.3.2 Le stockage CSR . . . 47

A Rappels d'algèbre linéaire **50**
A.1 Matrices symétriques 50
A.2 Normes vectoriellles et matricielles 50
A.3 Conditionnement d'un système linéaire 52
A.4 Problèmes de moindres carrés 53
A.4.1 Propriétés mathématiques 53
A.4.2 Méthodes numériques 55
A.5 Méthodes itératives classiques 58

Avertissement

Cette édition du cours est encore un travail non–achevé. Il est possible, et malheureusement probable, qu'il reste des erreurs, des fautes de frappe, des explications peu claire, et même peut-être des fautes de mathématiques. Par conséquent, lisez ce cours avec attention et un esprit critique.

Dans l'espoir d'améliorer ce document pour les promotions à venir, je vous sera gré de me communiquer les erreurs que vous pourriez découvrir, par un message à `michel.kern@inria.fr`

Chapitre 1

La méthode du gradient conjugué

Nous présentons dans ce chapitre la méthode gradient conjugué. Cette méthode, associée à une méthode de préconditionnement (voir le Chapitre 3 pour une introduction à ce vaste sujet) s'est imposée depuis maintenant 40 ans comme la méthode itérative la plus utilisée pour résoudre des systèmes linéaires de tr1es grande taille, comme ceux provenant de la discrétisation d'équations aux dérivées partielles.

Ce chapitre suit essentiellement [14].

1.1 Notations – généralités

Dans tout ce chapitre $A \in \mathbf{R}^{n\times n}$ désigne une matrice définie positive. Nous allons étudier une *méthode itérative* pour approcher la solution d'un résolution du système linéaire $Ax = b$ (on notera x^* la solution exacte ($x^* = A^{-1}b$). La méthode, que nous étudierons aux paragraphes 1.2 et suivants, va générer une suite x_k (convergeant vers x^*, la démonstration en sera donnée au paragraphe 1.4). Étant donné un itéré x_k, nous pouvons lui associer :

le résidu $r_k = b - Ax_k$. Il s'agit d'une quantité calculable, elle jouera un rôle important dans l'algorithme ;

l'erreur $e_k = x^* - x_k$. Il s'agit d'une quantité qui n'est pas calculable sans connaissance préalable de la solution exacte. Son intérêt est essentiellement théorique.

Notons la relation suivante entre ces 2 quantités, que l'on obtient en multipliant la définition de l'erreur par A, et notant que $Ax^* = b$:

$$Ae_k = r_k. \tag{1.1}$$

Pour préciser le lien entre le résidu et l'erreur, il est commode d'introduire le conditionnement de la matrice A (on devrait plus précisément parler du conditionnement du système linéaire).

Définition 1.1. Le *conditionnement* de la matrice A (pour la norme matricielle $\|.\|$) est

$$\kappa(A) = \|A\| \left\|A^{-1}\right\|. \tag{1.2}$$

La définition et les principales propriétés des normes matricielles sont rappelées dans l'annexe A.

Notons les propriétés suivantes du conditionnement.

Proposition 1.1. *Soit $A \in \mathbf{R}^{n\times n}$ une matrice inversible.*

- *Pour $\lambda \in \mathbf{R}$, $\kappa(\lambda A) = \kappa(A)$;*
- *$\kappa(A) \geq 1$, et $\kappa_2(A) = 1$ si et seulement si A est une matrice orthogonale ;*
- *Si A est une matrice symétrique et définie positive, en notant $0 < \lambda_1 \leq \cdots \leq \lambda_n$ ses valeurs propres, on a*

$$\kappa_2(A) = \frac{\lambda_n}{\lambda_1}.$$

Démonstration. La démonstration se trouve dans de nombreux livres d'algèbre linéaire, voir les références données dans l'annexe A. ■

Nous pouvons maintenant donner le lien entre résidu et erreur

Proposition 1.2. *Soit $x_k \in \mathbf{R}^n$ et soit $r_k = b - Ax_k$ et $e_k = x^* - x_k$ respectivement le résidu et l'erreur associés. On a*

$$\frac{\|e_k\|}{\|e_0\|} \leq \kappa(A)\frac{\|r_k\|}{\|r_0\|}. \tag{1.3}$$

Démonstration. En inversant la relation entre résidu et erreur, il vient

$$\|e_k\| \leq \left\|A^{-1}\right\| \|r_k\| .$$

De même (mais sans inverser), les quantités initiales sont reliées par

$$\|r_0\| \leq \|A\| \|e_0\| .$$

L'inégalité cherchée découle des deux précédentes. ■

Dans le cas important où la matrice A est symétrique et définie positive, la résolution du système linéaire $Ax = b$ est équivalente à la minimisation d'une fonctionnelle

Proposition 1.3. *Un vecteur x^* est solution du système linéaire $Ax = b$ si et seulement si x^* minimise la fonctionnelle $J : \mathbf{R}^n \to \mathbf{R}$ définie par*

$$J(x) = \frac{1}{2}(Ax, x) - (b, x) \tag{1.4}$$

Démonstration. Une preuve rapide consiste à remarquer que la fonctionnelle J est différentiable, et que son gradient est donné par

$$\nabla J(x) = Ax - b,$$

et que la fonctionnelle J est strictement convexe (car A est définie positive)

On peut également donner une démonstration élémentaire du résultat. On calcule, pour tout $y \in \mathbf{R}^n$:

$$J(x^* + y) = J(x) + \frac{1}{2}(Ay, y) + (Ax^* - b, y)$$

où l'on a utilisé la symétrie de A pour obtenir le dernier terme.

Si x^* est solution de $Ax = b$, le dernier terme de cette somme est nul, et comme A est définie positive, on voit que x^* réalise le minimum (global, strict) de J ;

réciproquement, si x^* minimise J , la quantité $\frac{1}{2}(Ay,y)+(Ax^*-b,y)$ doit être positive pour tout $y \neq 0$. Mais si x^* n'est pas solution du système linéaire, le choix de $y = \varepsilon r^*$, avec $r^* = b - Ax^*$, qui n'est donc pas nul, donne

$$J(x^* + y) - J(x^*) = \frac{1}{2}\varepsilon^2(Ar^*, r^*) - \varepsilon \|r^*\|^2$$

qui est négatif pour $\varepsilon > 0$ assez petit.

■

Lorsque A est définie positive, elle définit une norme, appelée *norme de l'énergie* selon la définition

$$\|x\|_A^2 = (Ax, x), \forall x \in \mathbf{R}^n \tag{1.5}$$

(exercice : vérifier que cela définit bien une norme). La fonctionnelle J est liée à cette norme par la relation (noter que le dernier terme est constant)

$$\|x - x^*\|_A^2 = 2J(x) + (Ax^*, x^*). \tag{1.6}$$

qui exprime que minimiser J revient à minimiser l'erreur dans la norme de l'énergie.

Pour vérifier cette égalité, on développe le membre de gauche, en utilisant la symétrie de A et la définition de x^* :

$$\|x^* - x\|_A^2 = (Ax, x) - 2(Ax^*, x) + (Ax^*, x^*) = (Ax - 2b, x) + (Ax^*, x^*).$$

Remarque 1.1. Lorsque la matrice A provient de la discrétisation d'un problème coercif par la méthode des éléments finis, la norme de l'énergie est naturelle, et provient du produit scalaire associé à la forme bilinéaire a. Soit u_h et v_h deux fonctions de l'espace discret V_h associé à une triangulation d'un ouvert $\Omega \subset \mathbf{R}^d$ (supposé polygonal ou polyèdral). On note $(\phi_i)_{i\in N_h}$ les fonctions de la base nodale de V_h (avec $N_h = \dim V_h$), et on note respectivement x et y les vecteurs de coordonnées de u_h et v_h :

$$u_h = \sum_{i=1}^{N_h} x_i\phi_i, \quad v_h = \sum_{i=1}^{N_h} y_i\phi_i.$$

On note enfin A la matrice de masse $a_{ij} = a(\phi_j, \phi_i)$ et on a la relation fondamentale :

$$a(u_h, v_h) = (y,\, Ax) = (Ax, y),$$

qui implique entre autres que

$$\|x\|_A^2 = a(u_h, u_h).$$

On retrouve le lien bien connu entre le théorème de Lax–Milgram et la minimisation de la fonctionnelle

$$j(u_h) = \frac{1}{2}a(u_h, u_h) - L(u_h)$$

où L est la forme linéaire figurant au second membre de la formulation variationnelle.

Lorsque la forme bilinéaire a correspond à un opérateur de type « $-\operatorname{div} K \operatorname{grad} u$ » dans un domaine Ω borné (avec une fonction $K \in L^\infty(\Omega)$ vérifiant $K(x) \geq k_* > 0$ pour presque tout $x \in \Omega$), la forme bilinéaire s'écrit

$$a(u, u) = \int_\Omega K\,|\nabla u|^2$$

et correspond effectivement à une énergie (thermique).

1.2 Définition de la méthode

La méthode du gradient conjugué construit une suite d'itérés en cherchant à minimiser l'erreur dans la nome de l'énergie $\|[\|_A]e$ sur une suite de sous-espaces de dimensions croissantes. Par construction, la méthode trouvera forcément la solution exacte du problème (après n itération), mais sa réelle utilité vient de la qualité des approximations obtenues après un petit nombre d'itérations (quelques centaines pour des problèmes avec plusieurs millions d'inconnues). Commençons par définir les sous-espaces en questions.

Définition 1.2. Étant donné un vecteur $r_0 \in \mathbf{R}^n$, l'espace de Krylov de degré k relatif à A et r_0) est

$$K_k(A, r_0) = K_k(r_0) = \text{vect}\left\{r_0, Ar_0, \ldots, A^{k-1}r_0\right\} \tag{1.7}$$

On notera les propriétés suivantes, dont la démonstration est laissée au lecteur :

— $K_k(r_0) \subset K_{k+1}(r_0)$, et donc la dimension de $K_k(r_0)$ est une fonction (non strictement croissante) de k, avec $\dim K_k(r_0) \leq k$.

— on a $x \in K_k(r_0) \iff x = p(A)r_0$, pour un *polynôme* p de degré inférieur ou égal à $k-1$.

Notons également que si la suite $K_k(r_0)$ stagne, c'est-à-dire si $K_k(r_0) = K_{k+1}(r_0)$ alors la solution du système linéaire $Ax = b$ est contenue dans l'espace affine $x_0 + K_k(r_0)$ (avec $r_0 = b - Ax_0$). Nous supposerons que k est le plus petit entier pour lequel l'égalité se produit.

En effet, dans ce cas, $A^k r_0 \in K_k(r_0)$, et donc $A^k r_0 = \alpha r_0 + Aq(A)r_0$ pour un polynôme q de degré strictement inférieur à $k-1$, et un scalaire α, qui est nécessairement non-nul.

En effet, en supposant que ce ne soit pas le cas, on aurait alors $A^k r_0 = Aq(A)r_0$, soit $(A^{k-1} - q(A))r_0 = 0$, puisque A est inversible. Comme q est de degré strictement inférieur à $k-1$, l'égalité $A^{k-1}r_0 = q(A)r_0$ montre que r_0 est en fait dans $K_{k-1}(r_0)$, ce qui contredit la minimalité de k.

On peut donc écrire

$$b = Ax_0 + r_0 = Ax_0 + \frac{1}{\alpha}A\left(A^{k-1} - q(A)\right)r_0,$$

et nous venons de voir que la matrice $A^{k-1} - q(A)$ n'est pas nulle. La solution du système linéaire est donc $x_0 + \dfrac{1}{\alpha}\left(A^{k-1} - q(A)\right)r_0 \in x_0 + K_k(r_0)$.

Par conséquent, lors de l'étude d'un algorithme dont les itérées sont dans l'espace de Krylov $K_k(r_0)$, on pourra sans perte de généralité faire l'hypothèse que l'inclusion $K_k(r_0) \subset K_{k+1}(r_0)$ est *stricte* (sinon, l'algorithme a déjà trouvé la solution du système linéaire).

Nous pouvons maintenant donner la définition de la méthode du gradient conjugué.

Définition 1.3. Étant donné un itéré initial $x_0 \in \mathbf{R}^n$, l'itéré x_k de la méthode du gradient conjugué réalise le minimum de la fonctionnelle J définie en (1.4) sur l'espace de Krylov $K_k(r_0)$.

Notons tout de suite que x_k est bien défini de façon unique par cette condition. On peut le voir en appliquant le théorème de projection sur un convexe (ici un sous-espace affine) pour le produit scalaire associé à la norme de l'énergie (voir l'équation (1.6)). On peut préciser cette caractérisation. Dans toute la suite, nous noterons $r_k = b - Ax_k$.

Lemme 1.1. *A l'itération k, x_k est caractérisé par les conditions suivantes :*

i) $x_k \in x_0 + K_k(r_0)$,

ii) $r_k \perp K_k(r_0)$.

Démonstration. En utilisant (1.6), nous voyons que x_k est la solution du problème de minimisation

$$\min_{x \in x_0 + K_k(r_0)} \|x - x^*\|_A ,$$

autrement dit, x_k est la projection de la solution exacte x^* sur le sous-espace affine (convexe, fermé) $x_0 + K_k(r_0)$. Cette projection est caractérisée par les conditions

i) $x_k \in x_0 + K_k(r_0)$ (identique à **i)** ci-dessus,

ii) $x_k - x^* \perp_A K_k(r_0)$ (orthogonalité par rapport au produit scalaire de l'énergie).

Il reste à expliciter la seconde condition :

$$\begin{aligned} x_k - x^* \perp_A K_k(r_0) &\iff \forall y \in K_k(r_0),\ (y,\ A(x_k - x^*)) = 0 \\ &\iff \forall y \in K_k(r_0),\ (y,\ (Ax_k - b)) = 0 \iff r_k \perp K_k(r_0). \end{aligned}$$

■

Cette définition de l'algorithme du gradient conjugué reste abstraite. Nous allons maintenant en donner une version « réalisable ».

1.3 L'algorithme

Le développement de la version plus algorithmique du gradient conjugué utilise diverses propriétés des itérés que nous établirons dans une série de lemmes. Nous donnerons ensuite des éléments concernant la mise en oeuvre de la méthode.

1.3.1 Développement de l'algorithme

Lemme 1.2. *Pour $l < k$, on a $r_l \in K_k(r_0)$, et $(r_l,\ r_k) = 0$.*

Par conséquent, $K_k(r_0) = \text{vect}\{r_0, \cdots, r_{k-1}\}$.

Démonstration. Par définition, $r_l = b - Ax_l = r_0 + A(x_0 - x_l) \in K_l(r_0) + AK_l(r_0) \subset K_{l+1}(r_0) \subset K_k(r_0)$, puisque $l + 1 \leq k$.

D'après le lemme 1.1, r_k est orthogonal à $K_k(r_0)$, et d'après le premier point, ce sous espace contient tous les résidus précédents.

Le premier point montre l'inclusion $K_k(r_0) \supset \text{vect}\{r_0, \cdots, r_{k-1}\}$, et l'orthogonalité des résidus montre l'égalité des dimensions. ■

Ce résultat n'est évidemment valable que si $r_k \neq 0$, mais si $r_k = 0$, alors par définition x_k est solution du système.

Par ailleurs, l'algorithme ne peut pas stagner : en effet, si $x_k = x_{k+1}$, alors $r_k = r_{k+1}$, et le lemme précédent montre que $\|r_k\|_2^2 = (r_k,\ r_{k+1}) = 0$, et donc ce cas ne peut se produire que si x_k est solution du système.

Nous pouvons donc définir un vecteur p_k comme la direction du changement entre deux itérés successifs. Il est commode de poser

$$x_{k+1} = x_k + \alpha_k p_k, \qquad \text{et } p_0 = r_0 \tag{1.8}$$

où α_k est un scalaire qui reste à déterminer. Notons qu'à ce stade, p_k est défini à une constante près, et α_k pourrait être « absorbé » dans la définition de p_k. Nous verrons que cette indétermination sera résolue naturellement au cours de la détermination de l'algorithme.

Lemme 1.3. *La direction de descente p_k vérifie les deux propriétés suivantes :*

i) $p_k \in K_{k+1}(r_0)$ *;*

ii) $p_k \perp_A K_k(r_0)$ *(c'est-à-dire* $(y,\, Ap_k) = 0,\ \forall y \in K_k(r_0)$.

Ces deux propriétés determinent p_k à un scalaire multiplicatif près.

Démonstration. Comme les espaces de Krylov sont emboîtés, x_k et x_{k+1} sont dans le même espace affine, donc leur différence est dans l'espace vectoriel $K_{k+1}(r_0)$, et la remarque précédent le lemme montre que l'on peut supposer $\alpha_k \neq 0$.

Pour montrer que $p_k \perp_A K_k(r_0)$, partons de

$$r_{k+1} = b - Ax_x - \alpha_k Ap_k = r_k - \alpha_k Ap_k. \tag{1.9}$$

Comme r_{k+1} est orthogonal à $K_{k+1}(r_0)$, et r_k est orthogonal à $K_k(r_0)$, leur différence est bien orthogonale à $K_k(r_0)$.

Comme p_k est orthogonal à un sous-espace de codimension 1 dans $K_{k+1}(r_0)$, sa direction est bien déterminé de manière unique. ■

On dit que p_k est *A-conjugué* à $K_k(r_0)$, ce qui explique l'origine du nom de la méthode.

Ce lemme montre que l'on peut déterminer p_k (à un facteur multiplicatif près) sans connaître x_{k+1}, et (1.8) permet alors de calculer x_{k+1}. On pourrait croire que la détermination de p_k est coûteuse. Il n'en est rien, et c'est ce qui constitue l'élément le plus étonnant de l'algorithme du gradient conjugué !

On montre, par un raisonnement similaire à celui du lemme 1.2, que

$$K_k(r_0) = \text{vect}\,(p_0, \ldots, p_{k-1}).$$

Lemme 1.4. *On a*

$$p_k = r_k + \beta_{k-1} p_{k-1}, \tag{1.10}$$

avec

$$\beta_{k-1} = -\frac{(r_{k-1},\, Ar_k)}{(r_{k-1},\, Ap_{k-1})}.$$

Démonstration. La première condition du lemme 1.3 et la remarque précédente montrent que, a priori, p_k s'écrit sous la forme

$$p_k = r_k + \sum_{j=0}^{k-1} \gamma_{kj} p_j$$

où les coefficients γ_{kj}, $j = 0, \ldots, k-1$ doivent être déterminés. Pour cela, nous utilisons la deuxième condition du lemme 1.3. Multiplions l'équation précédente par A, et prenons le produit scalaire avec r_l, pour $l = 0, \ldots, k-1$

$$(r_l,\, Ap_k) = (r_l,\, Ar_k) + \sum_{j=0}^{k-1} \gamma_{kj} (r_l,\, Ap_j), \quad l = 0, \ldots, k-1$$

Pour $l = 0, \ldots, k-1$, on a $(r_l,\, Ap_k) = 0$ d'après le lemme 1.3, puisque $r_l \in K_k(r_0)$. De même, mais pour $l = 0, \ldots, k-2$, $(r_l,\, Ar_k) = (r_k,\, Ar_l) = 0$ car $Ar_l \in K_k(r_o)$ et r_k est orthogonal à ce sous espace.

Pour les termes de la somme, pour $l = 0, \ldots, k-2$, utilisons la relation (1.9). Pour $l = 0$, le seul terme non-nul dans la somme est le premier (avec $j = 0$) qui vaut $\gamma_{k0}/\alpha_0((r_1 - r_0),\, r_0) = -\gamma_{k0}/\alpha_0 \, \|r_0\|_2^2$. On en déduit que $\gamma_{k0} = 0$. On procède de même pour les autres termes, pour montrer successivement que $\gamma_{kj} = 0$ pour $j = 0, \ldots, k-2$.

Il ne reste plus que le terme correspondant à $j = k-1$, qui conduit à la condition

$$0 = (r_{k-1},\, Ar_k) + \gamma_{k,k-1}(r_{k-1},\, Ap_{k-1})$$

et donne la valeur annoncée pour $\beta_{k-1} = \gamma_{k,k-1}$, à condition de montrer que $(r_{k-1},\, Ap_{k-1}) \neq 0$. Pour cela, notons que, comme $r_k = r_{k-1} - \alpha_{k-1} Ap_{k-1}$, on a

$$(r_k,\, r_{k-1}) = \|r_{k-1}\|_2^2 - \alpha_{k-1}(r_{k-1},\, Ap_{k-1}).$$

Le terme de gauche est nul par orthogonalité des résidus successifs. Comme le premier terme du second membre n'est pas nul (sauf si la méthode a convergé), il en est de même du second terme. ■

Il reste donc à calculer le coefficient α_k. Pour cela, nous pouvons utiliser le calcul itératif du résidu (1.9), et l'orthogonalité de r_{k+1} et p_k (conséquence de la propriété de minimisation vue au lemme 1.1 et de ce que $p_k \in K_{k+1}(r_o)$) :

$$(p_k,\, r_{k+1}) = (p_k,\, r_k) - \alpha_k(p_k,\, Ap_k)$$

qui permet de déterminer α_k (comme A est définie positive, le dénominateur est strictement positif).

L'algorithme du gradient conjugué consiste essentiellement à séquencer les équations (1.8), (1.9) et (1.10). Avant de détailler l'algorithme, notons que les formules pour α_k et β_k utilisées en pratique ne sont pas celles que nous avons obtenues ci-dessus, mais des formes équivalentes qui ont de meilleures propriétés de stabilité numérique.

Lemme 1.5. *Les quantités α_k et β_k peuvent se calculer par les formules suivantes*

$$\alpha_k = \frac{\|r_k\|_2^2}{(p_k,\, Ap_k)} \tag{1.11}$$

$$\beta_k = \frac{\|r_{k+1}\|_2^2}{\|r_k\|_2^2} \tag{1.12}$$

Démonstration. Pour α_k il suffit de noter que $(p_k,\, r_k) = \|r_k\|_2^2$. Pour le voir, il suffit d'utiliser (1.10), et l'orthogonalité de r_k avec $p_{k-1} \in K_k(r_0)$.

Pour β_k, la relation de récurrence (1.10) donne

$$(Ap_k,\, p_{k+1}) = (Ap_k,\, r_{k+1}) + \beta_k(Ap_k,\, p_k)$$

et le membre de gauche est nul puisque les directions successives sont A-conjuguées. On en déduit

$$\beta_k = -\frac{(Ap_k,\, r_{k+1})}{(Ap_k,\, p_k)}.$$

Calculons séparément le dénominateur et le numérateur.

— D'après l'expression de α_k obtenue dans la première partie du lemme,

$$(Ap_k,\, p_k) = \frac{\|r_k\|_2^2}{\alpha_k};$$

— En utilisant (1.9), il vient (par orthogonalité des résidus successifs)

$$\|r_{k+1}\|_2^2 = -\alpha_k (Ap_k,\, r_{k+1}).$$

On en déduit l'expression (1.12) pour β_k.

■

Nous pouvons maintenant donner une version de l'algorithme est proche de la version implémentable, mais qui reste idéale sur plusieurs points (critère d'arrêt, gestion de la mémoire).

Algorithme 1.1 Algorithme du gradient conjugué

Données: $b \in \mathbf{R}^n, x_0 \in \mathbf{R}^n, A \in \mathbf{R}^{n\times n}$
Résultats: k, x_k
$r_0 = b - Ax_0, p_0 = r_0, k = 0$
Tant que $k \leq$ itmax et pas de convergence **faire**
$\alpha_k = \dfrac{\|r_k\|_2^2}{(p_k,\, Ap_k)}$
$x_{k+1} = x_k + \alpha_k p_k$
$r_{k+1} = r_k - \alpha_k A p_k$
$\beta_k = \dfrac{\|r_{k+1}\|_2^2}{\|r_k\|_2^2}$
$p_{k+1} = r_{k+1} + \beta_k p_k$
fin Tant que

1.3.2 Mise en oeuvre de l'algorithme

L'algorithme précédent 1.1 est très simple à mettre en oeuvre : il n'est pas nécessaire de conserver les itérés successifs, et l'algorithme peut être implémenté avec seulement 4 vecteur. Le coût d'une itération est de 4 opérations vectorielles (ces opérations sont toutes du type $y = y+ax$, où x e t y sont des vecteurs et a est un scalaire), le calcul de 2 produits scalaires et une norme, et le produit de la matrice A par un vecteur. Dans beaucoup d'applications (notamment dans la résolution d'équations aux dérivées partielles), la matrice A possède une structure particulière permettant de calculer ce produit pour un coût très inférieur à n, ce qui serait le cas si A est stocké comme une matrice pleine générale. Comme nous le verrons au Chapitre 4, quand A est une matrice creuse, le produit matrice vecteur peut se calculer pour un coût proportionnel aux nombres d'éléments non-nuls de la matrice. Quand la matrice provient de la discrétisation d'une équation aux dérivées partielles par différences finies, éléments finis, ou volumes finis, le nombre d'éléments non-nuls sur une ligne est approximativement constant, ou au moins borné par une constante. Dans ce cas, le produit matrice – vecteur à un coût linéaire par rapport aux nombre d'inconnues. Cette propriété se transmet au coût total d'une itération.

Le coût total de l'algorithme dépend naturellement du nombre d'itérations. Cette question sera traitée au paragraphe 1.4.

Remarque 1.2 (Mis en oeuvre sans matrice). En poursuivant la discussion précédente, on voit qu'il n'est pas nécessaire de la matrice A sous la forme d'un adressage $(i, j) \to A_{ij}$. Il suffit de disposer d'une fonction (un sous-programme) qui prend en entrée un vecteur v et renvoie le résultat du produit Av. Cela veut dire que l'on revient à la vision de l'application linéaire sous-jacente à la matrice. En Matlab, et aussi avec la bibliothèque Scipy de Pyhon, il est possible d'utiliser le gradient conjugué soit avec une matrice stockée en mémoire, soit avec un opérateur linéaire (voir des exemples au paragraphe **??**.

Cette remarque donne lieu à ce que l'on appelle en anglais un mise en oeuvre *matrix free*.

Remarque 1.3 (Arrêt de l'algorithme). Nous n'avons pas réellement spécifié la condition d'arrêt de l'algorithme 3.2, en indiquant simplement « pas de convergence ». Bien entendu, ceci doit être précisé. Nous pouvons d'ores et déjà noter que l'algorithme 3.2 doit s'arrêter après au plus n itérations : en effet, le lemme 1.2 montre que les résidus successifs $(r_0, \ldots, r_{k-1}$ forment)une base orthonormale de $K_k(r_0)$, et le nombre de vecteurs orthogonaux ne peut excéder la dimension de l'espace. Cependant ce résultat n'est jamais utile en pratique. On utilise couramment la méthode du gradient conjugué pour résoudre des systèmes comportant plusieurs millions d'inconnues, pour lesquels on arrête l'algorithme après quelques centaines d'itérations, voire moins.

Ce fait remarquable justifie a posteriori le choix des espaces de Krylov pour chercher la solution : ils fournissent très rapidement des approximations précises de la solution. Ce point sera quantifié au paragraphe 1.4, et l'on parle (improprement) de la *convergence* de l'algorithme.

Nous pouvons indiquer dès maintenant le choix du critère d'arrêt le plus utilisé en pratique :

— Un critère d'arrêt ne doit utiliser que des quantités calculables. L'erreur n'entre évidemment pas dans ce cadre, et l'on doit utiliser le résidu, les deux quantités étant liées par l'équation 1.1. On arrête l'algorithme quand le résidu est suffisamment petit ;

— La notion de « petit » ne peut être absolue, et doit relative à la taille des données.

On recommande donc le critère d'arrêt suivant

$$\|r\| \leq \eta \|b\| , \tag{1.13}$$

où η (la tolérance) est une quantité spécifiée par l'utilisateur de la méthode, indiquant quand le résidu est suffisamment petit (on prend en général une valeur entre 10^{-12} et 10^{-6}. Nous n'avons pas précisé la norme utilisée, qui sera le plus souvent la norme euclidienne.

Une discussion détaillée des critères d'arrêt pour les méthodes itératives (ainsi que de nombreuses informations pratiques) se trouve dans le livre [4]. Ce livre explique entre autres pourquoi il ne faut pas remplacer, dans (1.13), la norme du second membre $\|b\|$ par celle du résidu initial $\|r_0\|$.

Nous pouvons maintenant donner une forme réalisable de l'algorithme du gradient conjugué, prenant en compte à la fois la gestion de la mémoire et le critère d'arrêt. Le seul point délicat est qu'il faut conserver les normes de deux résidus successifs.

1.4 Convergence

Commençons par une observation simple. Par sa définition, la méthode du gradient conjugué trouve la solution exacte du système $Ax = b$ en au plus n itérations, si n est la dimension du système. En effet, les sous-espaces de Krylov sont de dimension au plus n. Soit l'algorithme a trouvé une solution après $k < n$ itérations, soit à l'itération n la minimisation est posée sur tout l'espace, et l'itéré x_n est la solution.

Algorithme 1.2 Algorithme du gradient conjugué

Données: $b \in \mathbf{R}^n, x \in \mathbf{R}^n, A \in \mathbf{R}^{n\times n}, \eta > 0$
Résultats: k, x_k
$r = b - Ax, p = r, k = 0, r_{\text{old}} = \|r\|_2^2$
Tant que $k \leq \text{itmax}$ et $r_{\text{old}} \geq \eta \|b\|_2$ **faire**
$w = Ap$
$\alpha = \dfrac{r_{\text{old}}}{(p,\, w)}$
$x = x + \alpha p$
$r = r - \alpha w$
$r_{\text{new}} = \|r\|_2^2$, $\beta = \dfrac{r_{\text{new}}}{r_{\text{old}}}$
$p = r + \beta p$
$r_{\text{old}} = r_{\text{new}}$, $k = k + 1$
fin Tant que

Au moment où elle a été proposée, la méthode du gradient a d'abord été considérée comme une méthode directe, à cause de cette propriété. Mais ce point de vue n'est plus celui adopté à l'heure actuelle :

— tout d'abord la propriété concernant la solution exacte n'est plus vraie lorsque la méthode est implémentée en arithmétique en précision limitée. Dans ce cas, l'algorithme peut ne pas trouver la solution avant bien plus de n itérations ;

— Mais même si l'on considère la situation idéalisée, ce résultat n'a pas d'intérêt pratique. La méthode du gradient conjugué est couramment utilisée pour résoudre des systèmes linéaires à plusieurs centaines de milliers, voire des millions, d'inconnues. Or nous verrons dans la suie de ce paragraphe que la méthode peut produire des approximations précises de la solution en quelques centaines d'itérations, parfois encore moins.

Ainsi, le point de vue « moderne » sur la méthode du gradient conjugué est comme une méthode itérative, produisant une très bonne approximation de la solution exacte du système en un nombre limité d'itérations. C'est en ce sens que nous parlerons de « convergence », c'est à dire en majorant la différence entre l'itéré x_k et la solution exacte. Nous verrons que cela est possible en choisissant une norme liée au problème.

La première étape pour établir ces résultats est de réinterpréter la méthode du gradient conjugué en terme de polynômes. Nous noterons P_k l'espace vectoriel des polynômes de degré inférieur ou égal à k.

Proposition 1.4. *Soit x^* la solution du système $Ax = b$, et soi x_k l'itéré produit par l'algorithme du gradient conjugué après k itérations.*

i) *il existe un polynôme $p \in P_k$, vérifiant $p(0) = 1$, tel que*

$$x^* - x_k = p(A)(x^* - x_0); \tag{1.14}$$

ii) *L'itéré x_k est caractérisé par la condition*

$$\|x^* - x_k\|_A = \min_{\substack{p\in P_k \\ p(0)=1}} \|p(A)(x^* - x_0)\|_A \tag{1.15}$$

Preuve. **i)** Puisque $x_k \in x_0 + K_k(r_0)$, on a, pour certains coefficients γ_j, $j = 0, \ldots, k-1$

$$x_k - x^* = x_0 - x^* + \sum_{j=0}^{k-1} \gamma_j A^j r_0.$$

En utilisant l'égalité $r_0 = Ax^* - Ax_0$ on obtient (1.14) si l'on définit le polynôme $p(z) = 1 - \sum_{j=0}^{k-1} \gamma_j z^{j+1}$

ii) L'égalité (1.15) est une conséquence immédiate du point précédent et de la définition 1.3.

■

Pour exploiter ce résultat, nous utiliserons les valeurs propres et vecteurs propres de la matrice A. Notons qu'il s'agit là d'un outil uniquement théorique, la connaissance des valeurs propres de A n'est aucunement nécessaire à la mise en oeuvre de l'algorithme, comme on l'a vu au paragraphe 1.3.2. Comme A est symétrique, elle est diagonalisable avec une matrice de passage orthogonale, et ses valeurs propres sont réelles (et strictement positives, puisqu'elle est définie positive).

On a donc l'égalité

$$A = U \Lambda U^T, \tag{1.16}$$

où la matrice (carrée) U est orthogonale ($U^T U = I$), et la matrice Λ est diagonale, $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$, avec $0 < \lambda_1 \leq \ldots \leq \lambda_n$. On notera $\sigma(A) = \{\lambda_1, \ldots, \lambda_n\}$ le *spectre* de A (l'ensemble de ses valeurs propres).

Une conséquence immédiate est que, pour tout polynôme p, on également

$$p(A) = U p(\Lambda) U^T, \quad \text{avec } p(\Lambda) = \mathrm{diag}(p(\lambda_1), \ldots, p(\lambda_n)).$$

On notera également $A^{1/2}$ la matrice

$$A^{1/2} = U \Lambda^{1/2} U^T \quad \text{avec } p(\Lambda) = \mathrm{diag}(\lambda_1^{1/2}, \ldots, \lambda_n^{1/2}).$$

C'est l'unique matrice symétrique et définie positive vérifiant $A^{1/2} \cdot A^{1/2} = A$.

Le résultat suivant exploite ces notions.

Lemme 1.6. *Pour tout vecteur $x \in \mathbf{R}^m$, et pour tout polynôme p on a :*

i) $\|x\|_A = \|A^{1/2} x\|_2$,

ii) $\|p(A)x\|_A \leq \max_{\lambda \in \sigma(A)} |p(A)| \, \|x\|_A$.

Preuve. Il s'agit dans les deux cas d'un simple calcul.

i) $\|x\|_A^2 = x^T A x = X^T A^{1/2} A^{1/2} x = (A^{1/2}x)^T (A^{1/2}x) = \|A^{1/2}x\|_2$;

ii) D'après le point précédent, $\|p(A)x\|_A^2 = \|A^{1/2} p(A) x\|_2^2$, or $A^{1/2}$ et $p(A)$ commutent, d'où

$$\|x\|_A^2 \leq \|p(A)\|_2^2 \left\|A^{1/2} x\right\|_2^2,$$

où $\|p(A)\|_2$ est la norme matricielle, et la conclusion suit en remarquant que, pour $\|x\|_2 = 1$, $\|p(A)x\|_2 = \|U p(\Lambda) U^T x\|_2 = \|p(\Lambda) y\|_2$, avec $\|y\|_2 = \|U^T x\|_2 = 1$ (dans les 2 cas parce que U est orthogonale), et que la norme euclidienne d'une matrice diagonale est égale à son plus grand élément.

■

Ces résultats préliminaires nous permettent d'énoncer le résultat fondamental suivant.

Théorème 1.1. *les itérés de la méthode du gradient conjugué vérifient l'inégalité suivante :*

$$\|x_k - x^*\|_A \leq \min_{\substack{p \in P_k \\ p(0)=1}} \max_{\lambda \in \sigma(A)} |p(\lambda)| \, \|x_0 - x^*\|_A \,. \tag{1.17}$$

Preuve. La preuve consiste simplement à expliciter la norme dans la définition 1.3 en utilisant le lemme 1.6. ■

La version simplifiée donnée au corollaire suivant, dont la preuve est immédiate, est souvent suffisante pour les applications.

Corollaire 1.1. *Pour tout polynôme $p \in P_k$, avec $p(0) = 1$, on a*

$$\|x_k - x^*\|_A \leq \max_{\lambda \in \sigma(A)} |p(\lambda)| \, \|x_0 - x^*\|_A \,.$$

Pour appliquer le Corollaire 1.1, il suffit de trouver un polynôme adapté à la situation. Une application particulièrement simple est de retrouver la terminaison en un nombre fini d'itérations de la méthode du gradient conjugué. On choisit pour cela le polynôme caractéristique de A, qui est explicite si l'on connaît les valeurs propres.

$$p(\lambda) = \prod_{i=1}^{n} \left(1 - \frac{\lambda}{\lambda_i}\right).$$

Ce polynôme s'annule en chaque valeur propre, le second membre de la borne du Corollaire 1.1 est donc nul, d'où $x_n = x^*$.

Dans le cas général, c'est-à-dire sans autre hypothèse sur la répartition des valeurs propres de la matrice A, on obtient le résultat de convergence suivant :

Proposition 1.5. *On note respectivement λ_1 et λ_n la plus petite et la plus grande valeur propre de A, et on note $\kappa(A) = \dfrac{\lambda_n}{\lambda_1}$ le* conditionnement *de A. Les itérés de la méthode du gradient conjugué vérifient :*

$$\|x_k - x^*\|_A \leq 2 \left(\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1}\right)^k \|x_k - x^*\|_A \,. \tag{1.18}$$

(partielle). Le maximum sur l'ensemble discret des valeurs propres est difficile à déterminer. Sans autre information que la valeur des valeurs propres extrêmes, il est naturel de chercher le minimum sur *l'intervalle* $[\lambda_1, \lambda_n]$, ce qui a naturellement pour effet d'augmenter le minimum du second membre de l'inégalité (1.17) en

$$\|x_k - x^*\|_A \leq \min_{\substack{p \in P_k \\ p(0)=1}} \max_{\lambda \in [\lambda_1, \lambda_n]} |p(\lambda)| \, \|x_0 - x^*\|_A \,.$$

La détermination du min-max est un problème d'approximation polynomiale, dont la solution est donnée par le polynôme de Tchebychev adapté à l'intervalle en question. Cette partie, technique, de la démonstration sera omise. Le lecteur pourra trouver les détails dans les références [1, para. 9.5.5.] ou [15]. ■

On retrouve ici le conditionnement, non pas comme indicateur de la sensibilité de la matrice aux perturbations, mais comme indicateur de la vitesse de convergence. Plus le conditionnement sera grand, plus le facteur gouvernant la convergence sera proche de 1, et plus cette convergence sera lente. On notera que c'est la racine carrée du conditionnement qui intervient, ce qui constitue un progrès par rapport à la méthode de relaxation (SSOR), où la vitesse de convergence dépend du conditionnement.

Exemple 1.1 (Le laplacien sur un carré)**.**
Ã titre d'illustration, prenons l'exemple de la discrétisation de l'équation de Poisson avec des conditions de Dirichlet homogène sur le carré unit par une méthode de différences finies. On résout l'équation aux dérivées partielles

$$-\Delta u = f \text{ dans } \Omega =]0,1[\times]0,1[, \quad u = 0 \text{ sur } \partial\Omega.$$

La discrétisation par une méthode de différences finies sur une grille régulière de pas h (on supposera que $h = 1/(N+1)$ où N est un entier) conduit à un système linéaire $Ax = b$ où la matrice $A \in \mathbf{R}^{N^2 \times N^2}$ est

$$A = \frac{1}{h^2} \left(\begin{array}{ccccc|ccccc|ccccc}
4 & -1 & & & & -1 & & & & & & & & & \\
-1 & 4 & -1 & & & & -1 & & & & & & & & \\
 & -1 & 4 & -1 & & & & -1 & & & & & & & \\
 & & -1 & 4 & -1 & & & & -1 & & & & & & \\
 & & & -1 & 4 & & & & & -1 & & & & & \\
\hline
\ddots & & & & & \ddots & \ddots & & & & \ddots & & & & \\
 & \ddots & & & & \ddots & \ddots & \ddots & & & & \ddots & & & \\
 & & -1 & & & & -1 & 4 & 1 & & & & -1 & & \\
 & & & \ddots & & & & \ddots & \ddots & \ddots & & & & \ddots & \\
 & & & & \ddots & & & & \ddots & \ddots & & & & & \ddots \\
\hline
 & & & & & -1 & & & & & 4 & -1 & & & \\
 & & & & & & -1 & & & & -1 & 4 & -1 & & \\
 & & & & & & & -1 & & & & -1 & 4 & -1 & \\
 & & & & & & & & -1 & & & & -1 & 4 & -1 \\
 & & & & & & & & & -1 & & & & -1 & 4
\end{array}\right)$$

et où le vecteur $b \in \mathbf{R}^{N^2}$ contient des approximations des valeurs de la fonction f aux sommets de la grille.

On sait que le conditionnement de la matrice A vaut

$$\kappa(A) = \frac{\sin^2\left(\dfrac{n\pi}{2(n+1)}\right)}{\sin^2\left(\dfrac{\pi}{2(n+1)}\right)} \approx \frac{4}{\pi^2 h^2},$$

ce qui conduit à un taux de convergence de

$$\rho = \frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1} \approx= 1 - \pi h + O(h^2).$$

On en déduit que le nombre d'itérations nécessaires pour réduire la norme du résidu initial d'un facteur donné est (approximativement) proportionnel au nombre de points de discrétisation sur un côté du carré.

Ce résultat sera confirmé numériquement au paragraphe **??**.

Cet exemple est représentatif de ce que l'on peut observer lorsqu'on résout une équation aux dérivées partielles (de type elliptique) par la méthode du gradient conjugué : pour atteindre un même niveau de réduction de la norme du résidu, le nombre d'itérations augmente lorsque l'on raffine le maillage. Comme le coût d'une itération augmente également (à peu près linéairement par rapport au nombre d'inconnues), le coût total de la méthode augmente plus vite que le nombre d'inconnues. On dit que la méthode du gradient conjugué n'est pas optimale. Il existe des méthodes optimales, la plus connue étant la méthode multigrille (voir la section **??**). Néanmoins, la méthode du gradient conjugué continue d'être utilisée, en partie pour sa simplicité de mise en oeuvre, mais également pour sa plus grande robustesse en comparaison avec la méthode multigrille. En réalité, les deux méthodes sont utilisées ensemble : on peut soit voir le gradient conjugué comme un *accélérateur* de la méthode multigrille, soit le multigrille comme un préconditioneur.

Remarque 1.4 (Influence de la répartition des valeurs propres).

Pour diminuer le nombre d'itérations de la méthode, on peut chercher à formuler, et à résoudre, un système équivalent au système original, mais dont le conditionnement est plus petit (ou plus généralement, dont les valeurs propres ont une répartition plus favorable pour la convergence). On parle de *préconditionnement* du système linéaire, un sujet que nous abordons au chapitre 3.

Chapitre 2

Systèmes linéaires non symétriques : la méthode GMRES

Dans ce chapitre nous considérons le cas d'un système linéaire

$$Ax = b, \quad A \in \mathbf{R}^{n\times n},\ b \in \mathbf{R}^n, \tag{2.1}$$

où la matrice A n'est pas nécessairement inversible. Notez que A sera toujours supposée inversible. Dans ce cas, la Proposition 1.3 n'est plus valable, la résolution du système n'est plus équivalente à la minimisation d'une fonctionnelle d'énergie.

Nous suivons également l'approche de Kelley [14, Chap. 3]

Il existe de nombreuses situations conduisant à la résolution d'un système non–symétrique. Citons :

— la discrétisation d'une équation d'advection–diffusion

$$-\Delta u + \operatorname{div}(\vec{q}u) = f \text{ dans } \Omega, \quad u = 0 \text{ sur } \partial\Omega,$$

où Ω est un ouvert de $\mathbf{R}^d$ ($d = 1, 2$ ou 3), et $\vec{q}$ est un champ de vecteurs donné ;

— la résolution des équations de Navier–Stokes, compressibles ou incompressibles ;

— certains problèmes d'élasticité non–linéaire ;

— de manière générale la résolution de problèmes non–linéaires par la méthode de Newton passe par la résolution, à chaque itération, d'un système linéaire dont la matrice est presque toujours non–symétrique.

2.1 Définition de la méthode GMRES, et premières propriétés

L'idée de la méthode GMRES (Résidu Minimum Généralisé, en anglais « Generalized Minimimum RESidual », inventée en 1986 par Y. Saad et M. H. Schultz [20]) est de minimiser la norme euclidienne du résidu sur les espaces de Krylov correspondant au résidu initial. De manière précise :

Définition 2.1. Étant donné un itéré initial $x_0 \in \mathbf{R}^n$, l'itéré x_k de la méthode GMRES réalise le minimum de

$$\|b - Ax\|_2^2 \tag{2.2}$$

sur l'espace affine de Krylov $x_0 + K_k(r_0)$.

L'espace de Krylov $K_k(r_0)$ a été défini au chapitre 1. Cette définition ne dépend pas de la symétrie de A, et nous pouvons donc toujours écrire

$$x_k = x_0 + \sum_{j=1}^{k-1} \gamma_j A^j r_0,$$

pour des coefficients $\gamma_1, \ldots, \gamma_{k-1}$. On en déduit

$$b - Ax_k = b - Ax_0 - A\sum_{j=1}^{k-1} \gamma_j A^j r_0 = r_0 - \sum_{j=1}^{k-1} \gamma_{j-1} A^j r_0.$$

Il existe donc un polynôme $p \in P_k$, avec $p(0) = 1$ tel que, si $x_k \in x_0 + K_k(r_0)$, $r_k = b - Ax_k$ s'écrit sous la forme

$$r_k = p(A)r_0. \tag{2.3}$$

Une conséquence de cette relation est que, comme le gradient conjugué, GMRES trouve toujours la solution du système après un nombre fini d'itérations.

Proposition 2.1. *Après au plus n itérations, l'itéré calculé par la méthode GMRES est égal à la solution exacte du système* (2.1).

Démonstration. On choisit pour p le polynôme caractéristique de A, qui est de degré n. Plus précisément, pour tenir compte de la normalisation $p(0) = 1$, on prend

$$p(\lambda) = \det(A - \lambda I)/\det(A).$$

Notons que $\det(A) \neq 0$ puisque nous avons supposé A inversible.

Le théorème de Cayley–Hamilton ([18, Thm 7.1.1], [11, par. 4.2, Thm 3] ou [22, Thm 3.3]) nous indique que $p(A) = 0$, et d'après les remarques précédant la proposition, cela implique $r_n = 0$. ■

2.2 L'algorithme

2.2.1 L'algorithme d'Arnoldi

Supposons connue une base de l'espace de Krylov $K_k(r_0)$, et notons $V_k \in \mathbf{R}^{k\times n} = [v_1^k, \ldots, v_k^k]$ la matrice dont les colonnes sont les vecteurs de cette base. Un choix possible (mais nous verrons que ce n'est pas le meilleur) est bien entendu de prendre $v_k = A^{k-1}r_0$. La solution approchée à l'étape k de la méthode GMRES s'écrit donc sous la forme

$$x_k = x_0 + \sum_{j=1}^{k} y_j v_j^k = x_0 + V_k y_k, \quad y_k = (y_1^k, \ldots, y_k^k)^T \in \mathbf{R}^k,$$

pour un vecteur y_k qui reste à déterminer (l'itéré x_k ne dépend pas du choix de la base V_k, mais sa représentation y_k en dépend). Le problème de minimisation qui définit x_k devient donc

$$\min_{y\in\mathbf{R}^k} \|r_0 - AV_k y\|_2^2 .$$

Cette expression sera valable pour tout choix de la matrice V_k. Mais le choix naturel de V_k indiqué précédemment pose des difficultés du point de vue de la mise en oeuvre numérique :

— L'espace de Krylov $K_k(r_0)$ est un bon espace pour approcher la solution, mais la base de puissances de $V_k = [r_0, Ar_0, \ldots, A^{k-1}r0]$ conduit à une matrice v_k mal conditionnée, et ceci d'autant plus que k augmente. Ce constat est lié à la méthode de la puissance pour calculer les valeurs propres (voir par exemple [16, Chap. 10]), qui montre que les vecteurs $A^k r_0$ deviennent parallèles au vecteur propre associé à la valeur propre de plus grand module (si celle-ci est simple, et moyennant une hypothèse raisonnable sur r_0) ;
— de manière générale, si la matrice V_k n'est pas orthogonale, le problème de moindres carrés précédent sera difficile et coûteux à résoudre.

Pour ces raisons, une première étape consiste à chercher une base orthonormée de l'espace de Krylov $K_k(r_0)$. Cela peut être réalisé par la *méthode d'Arnoldi* qui consiste à appliquer la méthode d'orthogonalisation de Gram–Schmidt à la suite des sous–espaces $K_j(r_0)$, pour j variant de 1 à k. Dans toute la suite, il sera commode de noter $\beta = \|r_0\|_2$, que l'on peut supposer non–nul.

À chaque étape, on suppose que l'on dispose d'une base orthonormée $(v_1, \ldots, v_k)$ de l'espace de Krylov $K_k(r_0)$, et on doit donc définir le vecteur v_{k+1} de sorte que $(v_1, \ldots, v_{k+1})$ soit une base orthonormée de $K_{k+1}(r_0)$. Pour cela, il suffit d'orthogonaliser le nouveau vecteur Av_k contre les éléments précédents de la base. Pour cela, on calcule

$$w_{k+1} = Av_k - \sum_{j=1}^{k} (Av_k, v_j) v_j,$$

et on définit $v_{k+1} = \frac{w_{k+1}}{\|w_{k+1}\|_2}$.

Dans sa version la plus simple, qui sera améliorée plus loin, cela conduit à l'algorithme 2.1. Par construction, les vecteurs produits par cet algorithme sont orthogonaux, tant que $h_{k+1,k} \neq 0$.

Algorithme 2.1 Algorithme d'Arnoldi, version « naïve »

Données: $k \in \mathbf{N}$, base orthonormée $(v_1, \ldots, v_k)$ de $K_k(r_0)$.
Résultats: Base orthonormée $(v_1, \ldots, v_{k+1})$ de $K_{k+1}(r_0)$

```
v_1 = r_0/β
pour j = 1, ..., k faire
    h_jk = (Av_k, v_j)
fin pour
w_{k+1} = Av_k
pour j = 1, ..., k faire
    w_{k+1} = w_{k+1} − h_jk v_j
fin pour
h_{k+1,k} = ‖w_{k+1}‖_2
si h_{k+1,k} = 0 alors stop
v_{k+1} = w_{k+1}/h_{k+1,k}
```

Mais si cette dernière éventualité se produit, alors l'algorithme a trouvé la solution du système linéaire.

Lemme 2.1. *Dans l'algorithme 2.1, si $h_{k+1,k} = 0$ alors $x^\star \in K_k(r_0)$.*

Démonstration. Par hypothèse, $Av_k = \sum_{j=1}^{k} h_{jk} v_j$, et donc $Av_k \in K_k(r_0)$, soit $AK_k(r_0) \subset K_k(r_0)$. Les colonnes de V_k forment une base orthonormée de $K_k(r_0)$, donc

$$AV_k = V_k \tilde{H}_k$$

où la matrice $\tilde{H}_k \in \mathbf{R}^{k\times k}$ est inversible (puisque A l'est). On en déduit

$$r_k = b - Ax_k = r_0 - A\underbrace{(x_k - x_0)}_{\in K_k(r_0)} = r_0 - V_k\tilde{H}_k y,$$

pour $y \in \mathbf{R}^k$.

Par défiintion, $r_0 = \beta v_1 = \|r_0\|_2 V_k e_1$ (où e_1 est le premier vecteur de la base canonique de $\mathbf{R}^k$), et par conséquent

$$\|r_k\|_2 = \left\|\beta V_k e_1 - V_k\tilde{H}y\right\|_2 = \|\beta e_1 - H_k y\|_2 ,$$

puisque V_k est orthogonale. Comme $\tilde{H}_k$ est inversible, on peut choisir y de manière à ce que $r_k = 0$, et $x_k \in x_0 + K_k(r_0)$ est alors la solution de (2.1). ■

Nous pouvons donc faire dans la suite l'hypothèse que l'algorithme d'Arnoldi ne s'arrête pas, et que nous disposons, à chaque itération k d'une base orthonormée $(v_1, \ldots, v_k)$ de l'espace de Krylov $K_k(r_0)$. Autrement dit la matrice $V_k = \Big(v_1, \ldots, v_k\Big)$ est orthogonale (on prendra garde que c'est une matrice rectangulaire)

$$V_k^T V_k = I.$$

L'algorithme 2.1 permet également de définir une matrice $H_k \in \mathbf{R}^{k+1\times k}$ par

$$(H_k)_{ji} = (Av_j, v_i),\ j = 1, \ldots i. \tag{2.4}$$

La matrice H_k est formée par colonnes dans l'algorithme. Elle a une structure particulière, appelée *forme de Hessenberg* : elle a une seule diagonale non nulle sous la diagonale principale. Nous verrons que cette structure est la clé pour une mise en oeuvre efficace de GMRES.

L'algorithme d'Arnoldi peut s'écrire

$$h_{k+1,k}v_{k+1} = Av_k - \sum_{j=1}^{k} h_{jk}v_j.$$

Il est possible, et utile, de réécrire l'ensemble de ces relations sous forme matricielle :

$$AV_k = V_{k+1}H_k. \tag{2.5}$$

Illustrons la relation 2.5 la matrice H_k dans le cas $k = 4$

$$\begin{bmatrix} & & & \\ & & & \\ & A & & \\ & & & \\ & & & \end{bmatrix} \begin{bmatrix} v_1 & v_2 & v_3 & v_4 \end{bmatrix} = \begin{bmatrix} v_1 & v_2 & v_3 & v_4 & v_5 \end{bmatrix} \begin{bmatrix} h_{11} & h_{12} & h_{13} & h_{14} \\ h_{21} & h_{22} & h_{23} & h_{24} \\ & h_{32} & h_{33} & h_{34} \\ & & h_{43} & h_{44} \\ & & & h_{54} \end{bmatrix}$$

L'équation 2.5, l'orthogonalité de V_k (et l'orthogonalité de v_{k+1} à V_k) permettent d'écrire (rappelons que H_k est une matrice rectangulaire avec $k+1$ lignes et k colonnes) :

$$H_k = V_{k+1}^T A V_k. \tag{2.6}$$

Nous pouvons maintenant simplifier le problème de moindres carrés définissant x_k. Notons tout d'abord que nous pouvons écrire

$$r_0 = \beta V_{k+1} e_1,$$

où e_1 est le premier vecteur de la base canonique de $\mathbf{R}^{k+1}$, et $\beta = \|r_0\|_2$. Le résidu r_k peut donc se mettre sous la forme

$$r_k = b - Ax_k = r_0 - A(x_k - x_0) = r_0 - AV_k y_k,$$

pour un vecteur $y_k \in \mathbf{R}^k$, puisque les colonnes de V_k forment une base de $K_k(r_0)$. La relation 2.5 permet d'écrire

$$r_k = V_{k+1}(\beta e_1 - H_k y_k),$$

et l'orthogonalité de V_{k+1} montre alors que le problème de minimisation qui doit être résolu à l'itération k de GMRES est :

$$\min_{y \in \mathbf{R}^k} \|H_k y - \beta e_1\|_2 , \tag{2.7}$$

qui est un problème de moindres carrés posé en dimension k. Une fois ce problème résolu, en notant sa solution y_k, l'itéré x_k de GMRES s'en déduit par

$$x_k = x_0 + V_k y_k.$$

On voit ici une différence importante avec le cas du chapitre précédent : pour calculer x_k on a besoin de *tous* les vecteurs de la base de l'espace de Krylov qui ont déjà été calculés. Ces vecteurs sont aussi nécessaires pour l'algorithme d'Arnoldi : le coût d'une itération de GMRES augmente avec k.

2.2.2 Résolution du problème de moindres carrés

Nous donnerons dans ce paragraphe quelques indications sur la résolution de (2.7). Signalons tout de suite que, puisque la matrice H_k est rectangulaire, le « système » $H_h y = \beta e_1$ a une équation de plus que d'inconnues.

L'appendice A.4 contient des rappels sur la résolution numérique des problèmes de moindres carrés. En appliquant ces résultats au cas qui nous intéresse (le rôle de la matrice A dans l'appendice est ici joué par la matrice H_k, qui est de dimension $(k+1) \times k$, nous obtenons le résultat suivant :

Théorème 2.1. *Il existe une matrice orthogonale $Q_k \in \mathbf{R}^{(k+1)\times(k+1)}$ et une matrice triangulaire supérieure inversible $R_k \in \mathbf{R}^{k\times k}$ telles que*

$$H_k = Q_k \begin{pmatrix} R_k \\ 0 \end{pmatrix} \tag{2.8}$$

La solution du problème de moindres carrés (2.7) *est alors obtenue par :*

$$R_k y_k = \beta z_k, \tag{2.9}$$

avec $Q_k^T e_1 = \begin{pmatrix} z_k \\ \rho_k \end{pmatrix}$ *($z_k \in \mathbf{R}^k$, $\rho_k \in \mathbf{R}$), et le résidu est égal à*

$$\|\beta e_1 - H_k y_k\|_2 = \beta \, |\rho_k| .$$

Le coût de cette factorisation est (au premier ordre) $O(k^3)$. Par ailleurs, il est facile de voir, en utilisant l'orthogonalité de Q_k que R vérifie

$$H_k^T H_k = R_k^T R_K,$$

autrement dit R_k est le facteur de Cholesky de la matrice des équations normales, mais il a été obtenu sans former cette matrice. Le coût de la résolution du système triangulaire 2.9 est en $O(k^2)$, donc très inférieur à la factorisation.

En remontant les différentes définitions, nous voyons que

$$\|b - Ax_k\|_2 = \|\beta e_1 - H_k y_k\|_2 = \beta \, |\rho_k| \,,$$

ce qui veut dire que le résidu à l'itération k est disponible sans qu'il y ait besoin de calculer x_k !

2.2.3 Conséquences de l'arithmétique flottante

L'algorithme d'Arnoldi présenté en 2.1 suppose que les calculs sont réalisés exactement. Si l'on programme cet algorithme sur un ordinateur, en arithmétique flottante (celle utilisée par la majorité des langages, comme C, C++, Matlab ou Python), avec une précision relative d'environ 15 chiffres décimaux, on constate que les vecteurs produits ne sont pas orthogonaux. Ce phénomène bien connu est l'instabilité de la méthode Gram–Schmidt. Heureusement, il existe un remède simple, qui passe par une modification de l'algorithme : on orthogonalise immédiatement le vecteur Av_k contre chacun des vecteurs de la base $(v_1, \ldots, v_k)$. L'algorithme appelé « Gram–Schmidt modifié » est présenté dans l'algorithme 2.2. On peut montrer que les deux version

Algorithme 2.2 Algorithme d'Arnoldi avec Gram–Schmidt modifié, version numériquement stable

Données: $k \in \mathbf{N}$, base orthonormée $(v_1, \ldots, v_k)$ de $K_k(r_0)$.
Résultats: Base orthonormée $(v_1, \ldots, v_{k+1})$ de $K_{k+1}(r_0)$

```
v_1 = r_0/β
w_{k+1} = A v_k
pour j = 1, ..., k faire
    h_{jk} = (w_{k+1}, v_j)
    w_{k+1} = w_{k+1} − h_{jk} v_j
fin pour
h_{k+1,k} = ||w_{k+1}||_2
si h_{k+1,k} = 0 alors stop
v_{k+1} = w_{k+1}/h_{k+1,k}
```

sont mathématiquement équivalentes, mais ont des comportements différents en arithmétique flottante.

En incorporant un critère d'arrêt, on peut maintenant écrire une version complète de la méthode GMRES

Contrairement au gradient conjugué, le coût d'une itération de l'algorithme n'est pas constant, ni en calcul ni en stockage, puisque chaque itération doit enrichir l'espace de Krylov, et que la base orthonormée $(v_1, \ldots, v_k)$ doit être construite explicitement. Ainsi, à l'itération k de la méthode, on doit calculer

— 1 produit matrice–vecteur (le coût dépend de la matrice) ;

Algorithme 2.3 GMRES, avec Gram-Schmidt modifié, et résolution du problème de moindres carrés par QR

Données: Itéré initial $x_0 \in \mathbf{R}^n$, tolérance $\eta \in \mathbf{R}$, nombre maximum d'itérations $k_{\max}$.
Résultats: Solution approchée du système x_k

```
r_0 = b − Ax_0, β = ‖r_0‖_2, v_1 = r_0/β, ρ = β, k = 0
Tant que ρ > η ‖b‖_2 et k < k_max faire
  k = k + 1
    // Arnoldi (Gram–Schmidt modifié), calcul de v_{k+1}
    // Mise à jour de V_{k+1} et de H_k
  w_{k+1} = A v_k
  pour j = 1, ..., k faire
    h_{jk} = (w_{k+1}, v_j)
    w_{k+1} = w_{k+1} − h_{jk} v_j
  fin pour
  h_{k+1,k} = ‖w_{k+1}‖_2
  si h_{k+1,k} = 0 alors stop
  v_{k+1} = w_{k+1}/h_{k+1,k}
    // Résolution du problème de moindres carrés min_{y∈R^k} ‖βe_1 − H_k y‖_2
  Factoriser H_k = Q_k R_k
  e_1 = (1, 0, ..., 0)^T ∈ R^{k+1}, Q_k e_1 = (z_k^T, ρ_k)^T
  Résoudre R_k y_k = β z_k
  ρ = β ρ_k
fin Tant que
    // Calcul de la solution à convergence
x_k = x_0 + V_k y_k
```

— k produits scalaires de taille n
— k combinaisons linéaires de taille n
— la résolution du problème de moindres carrés, de coût $O(k^3)$.

Remarque 2.1. La dernière étape (résolution du problème de moindres carrés) peut être améliorée. On utilise la structure du problème (à chaque itération, on ajoute une colonne à la matrice H_k) pour mettre à jour itérativement la factorisation QR, ce qui ramène le coût de cette étape à $O(k)$. On utilise pour cela des rotations de Givens, voir les livres de Kelley [14, Par. 3.5] ou Saad [21, Par. 6.5.3] pour des détails sur cette étape technique.

En ce qui concerne le stockage, k itérations de GMRES demandent le stockage de k vecteurs de taille n (ainsi qu'une matrice de taille $k \times k$ dans notre implémentation simplifiée).

On voit donc que la principale limitation de GMRES est son coût, qui augmente plus que linéairement avec le nombre d'itérations. Une manière de contrôler le coût, en particulier du stockage, est de redémarrer la méthode après un nombre maximum d'itérations. Cette méthode, nommée GMRES(m), où m est le paramètre de redémarrage est également décrite dans les livres de Kelley [14, Alg. 3.5.2] ou Saad [21, Alg. 6.11].

2.3 Remarques sur la convergence

La convergence de GMRES est plus délicate à étudier que pour le gradient conjugué. La principale raison est qu'il n'est pas toujours possible de diagonaliser une matrice quelconque, et que même si c'est le cas, les valeurs propres ne jouent pas le même rôle que dans le cas défini positif. Nous nous contenterons donc des résultats les plus simples et d'indiquer des limitations connues.

Rappelons d'abord que, d'après la proposition 2.1, l'algorithme trouve toujours la solution exacte du système (2.1) en au plus n itérations (ce résultat est valable en arithmétique exacte). Comme pour le gradient conjugué, ce résultat n'a pas d'intérêt pratique, et ceci d'autant plus que les itérations de GMRES sont coûteuses. Remarquons également que la quantité $\|r_k\|_2$ décroît au cours de l'algorithme (on minimise cette quantité sur des espaces de dimension croissante), mais que cette décroissance n'est pas nécessairement stricte. Il existe des exemples pour lesquels le résidu stagne (reste constant) jusqu'à la dernière itération. C'est la cas pour la matrice et le second membre représentés ci–dessous pour $n = 4$ (mais l'exemple est valable pour n quelconque)

$$A = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

dont la solution exacte est $x = (0, 0, 0, 1)^T$. Si l'algorithme démarre avec $x_0 = 0$, le résidu reste nul jusqu'à la n-ème itération, où la méthode trouve la solution.

Comme pour le gradient conjugué, mais avec des conséquences qui seront moins spectaculaires, nous pouvons exploiter le lien entre l'espace de Krylov et les polynômes pour obtenir le résultat suivant, dont la démonstration vient simplement de (2.3) :

Théorème 2.2. *Soit $x_k \in K_k(r_o)$ l'approximation obtenue après k itérations de GMRES. Pour tout polynôme $p_k \in P_k$, avec $p_k(0) = 1$, on a*

$$\|r_k\|_2 \leq \|p_k(A) r_0\|_2$$

et on en déduit la majoration (plus faible, mais indépendante du résidu initial)

$$\frac{\|r_k\|_2}{\|r_0\|_2} \leq \|p_k(A)\|_2 . \tag{2.10}$$

Dans le cas où A est diagonalisable (rappelons que ce n'est pas vrai pour toute matrice), nous obtenons le résultat suivant :

Théorème 2.3. *Supposons que la matrice A soit diagonalisable*

$$A = V \Lambda V^{-1}$$

où $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$ est la matrice avec les valeurs propres de A sur la diagonale, et les colonnes de V sont formées par les vecteurs propres de A. Alors pour tout polynôme $p \in P_k$, avec $p(0) = 1$, on a la majoration

$$\frac{\|r_k\|_2}{\|r_0\|_2} \leq \kappa_2(V) \max_{z \in \{\lambda_1, \ldots, \lambda_n\}} |p(z)| . \tag{2.11}$$

Démonstration. Soit p un polynôme de degré k, avec $p(0) = 1$. On a

$$p(A) = Vp(\Lambda)V^{-1},$$

(rappelons que les valeurs propres $\lambda_1, \ldots, \lambda_n$, ainsi que les vecteurs propres, peuvent être complexes). On peut majorer la norme de $p(A)$ par

$$\|p(A)\|_2 \leq \|V\|_2 \left\|V^1\right\|_2 \|p(\Lambda)\|_2 ,$$

comme Λ est diagonale, on a encore $\|p(\Lambda)\|_2 = \max_{z\in\{\lambda_1,\ldots,\lambda_n\}} |p(z)|$. Par ailleurs on reconnaît $\|V\|_2 \left\|V^1\right\|_2 = \kappa_2(V)$, le conditionnement de la matrice des vecteurs propres. ■

Remarque 2.2. Ce résultat appelle les commentaires suivants, qui réduisent son application en pratique :

- Au moins pour les matrices diagonalisables, on peut donc se ramener à un problème de minimisation posé sur les polynômes. Ce problème est toutefois plus difficile à résoudre que celui rencontré pour les matrices symétriques et définies positives, parce qu'il est posé sur un sous-ensemble a priori quelconque du plan complexe. On peut se ramener à une partie convexe, en pratique des résultats existent si le spectre de A peut être inclus dans une ellipse *qui ne contient pas l'origine* (cette dernière hypothèse est restrictive), voir Saad[21, Par. 6.11.4] pour ces résultats.
- La quantité $\kappa_2(V)$ qui apparaît dans la borne (2.11) est difficile à estimer, et il n'existe pas de borne générale a priori (rappelons que dans le cas des matrices symétriques et définies positives, la matrice V peut être choisie orthogonale, ce qui conduit à $\kappa_2(V) = 1$. En général, même si l'on sait minimiser la quantité $\max_{z\in\{\lambda_1,\ldots,\lambda_n\}} |p(z)|$ (et le point précédent montre les limites de cette question), la borne obtenue est multipliée par le conditionnement des vecteurs propres, et ne donne donc qu'une indication très faible sur la convergence réelle de GMRES.

Pour ces raisons, on a cherché d'autres outils pour étudier la convergence de GMRES. On pourra consulter Greenbaum [12, Par. 3.2] pour une discussion des résultats connus.

Pour finir, signalons le résultat suivant, du à Greenbaum, Pták et Strakoš [13] : étant donné une suite (finie) décroissante $\eta_1 \geq \cdots \geq \eta_n$, il existe une matrice A et un second membre b tels que l'application de GMRES au système $Ax = b$ pour résidus les nombres η_n. De plus, les valeurs propres de la matrice peuvent être choisis à l'avance. Autrement dit, en toute généralité, la connaissance des valeurs propres ne donne aucune indication sur la convergence.

Chapitre 3

Préconditionnement

Ce chapitre donne quelques indications sur les les principales techniques de préconditionnement. Le terme « préconditionnement » vient de la réalisation que (au moins pour le gradient conjugué, c'est moins vrai pour GMRES), la convergence de la méthode est gouvernée par le conditionnement du système linéaire (voir en particulier la proposition 1.5). L'idée du préconditionnement est alors de remplacer le système à résoudre par un autre, qui possède bien entendu la même solution, mais dont la matrice soit « mieux conditionnée ». Ce terme est ici mis entre guillemets car, comme nous l'avions signalé au chapitre 1, la convergence du gradient conjugué dépend en réalité de la répartition de l'ensemble des valeurs propres, pas seulement du rapport des valeurs propres extrêmes.

Après quelques remarques générales sur ce qu'est une méthode de préconditionnement, nous verrons le cas particulier du gradient conjugué. Puis nous donnerons quelques exemples de méthodes générales. Ce chapitre n'est qu'une brève introduction à un vaste sujet. Une référence pour approfondir est le livre de Y. Saad [21], qui consacre deux chapitres au sujet, et deux autres pour étudier en détail des méthodes qui s'appliquent particulièrement bien aux systèmes issus de la discrétisation des équations aux dérivées partielles. Le livre (accessible en ligne) [4] donne des indications pratiques sur la mise en oeuvre des méthodes. Chen [7] est entièrement consacré à ce sujet, tandis que d'autres livres y consacrent une partie importante de leur contenu. On peut citer ceux de O. Axelsson [3], Axelsson et Barker [2], A. Greenbaum[12], G. Meurant[19] et H. van der Vorst [24]. Enfin, des articles de revue font régulièrement le point sur les progrès récents, citons par exemple celui, un peu plus ancien, de Benzi[5] et celui, récent, de Wathen[25]

3.1 Généralités

Étant donné un système linéaire

$$Ax = b, \quad A \in \mathbf{R}^n,\ b \in \mathbf{R}^n$$

le but d'une méthode de préconditionnement (on parle de manière quelque peu impropre de « préconditionneur ») est de remplacer ce système par un autre (qui a la même solution) est qui soit « plus facile » à résoudre. Il est difficile, et peu utile, de définir précisément ce que l'on entend par « plus facile ». En pratique, on cherche à diminuer le nombre d'itérations de la méthode originale, mais nous verrons que le préconditionnement a en général un sur–coût, et que le but sera finalement de réduire la durée totale de résolution.

Pour préciser quelque peu, une méthode de préconditionnement (à gauche, nous verrons qu'il y a d'autres possibilités) est donnée par une matrice M inversible, la *matrice de préconditionnement*, et le système préconditionné est

$$M^{-1}Ax = M^{-1}b. \tag{3.1}$$

Comme toujours, il est important de noter que l'écriture M^{-1} est symbolique. Il n'est jamais nécessaire de former explicitement l'inverse de M. Nous verrons qu'en général, la méthode se traduit par la nécessité de résoudre, à chaque itération de la méthode itérative, un système linéaire $Mz = r$. Le but est alors que

i) Le nombre d'itérations de la méthode avec préconditionnement soit plus faible que le nombre d'itérations de la méthode originale ;

ii) le sur–coût de chaque itération, induit par la résolution de l'étape de préconditionnement, reste faible.

Il faut ajouter ce que l'on appelle le coût de mise en place du préconditionnement, c'est-à-dire le coût du calcul de la matrice M à partir de A.

Il est habituel de dire que la matrice de préconditionnement est M, alors que on doit agir avec M^{-1}. La raison est que c'est M qui doit « ressembler » à A.

Deux exemples extrêmes aident à comprendre le compromis à réaliser :

— Choisir $M = A^{-1}$. Dans ce cas, le nombre d'itérations est réduit à 1, mais il est bien évident que l'on n'a rien gagné, puisque l'étape de préconditionnement est aussi difficile que le problème original ;

— Choisir $M = I$. On ne fait rien, ce qui ne coûte rien, mais ne change pas le nombre d'itérations.

Si un compromis « optimal » existe, il doit se situer entre ces deux extrêmes. Mais notons que ce compromis ne peut être absolu, et dépendra de la nature du problème : difficulté de la résolution sans préconditionneur, coût d'une itération (pour le gradient conjugué, c'est essentiellement le coût de la multiplication de la matrice A par un vecteur, pour GMRES, le coût d'une itération n'est pas constant, il est encore plus important de minimiser la taille de l'espace de Krylov). Les méthodes de préconditionnement ont donc souvent une composante heuristique. Il est parfois possible de justifier certains choix, mais ce sera rarement le cas dans le cadre de ce cours.

Pour conclure ces généralités, notons que nous avons insisté sur la présentation algébrique, mais qu'une méthode de préconditionnement peut souvent provenir d'une méthode de résolution approchée d'un problème voisin, l'interprétation algébrique est alors simplement un cadre général.

3.1.1 À gauche, à droite, ou des deux cotés ?

Il y a plusieurs façons de préconditionner un système linéaire. Nous avons introduit précédemment le préconditionnement à gauche, qui consiste à remplacer le système original par

$$M^{-1}Ax = M^{-1}b. \tag{3.2}$$

Cela veut dire qu'à chaque étape d'une méthode de Krylov, on doit remplacer le produit $r = Av$ par la succession de

$$r = Av, \text{ suivi de la résolution } Mz = r.$$

Selon les méthodes et les situations, cette vision en deux étapes est plus ou moins explicite. Avant la résolution, on doit calculer le nouveau second membre $M^{-1}b$, mais souvent, cela revient simplement à préconditionner le résidu initial $M^{-1}(b - Ax_0)$.

On peut également considérer le système préconditionné par la droite :

$$AM^{-1}y = b, \text{ avec } Mx = y, \tag{3.3}$$

on doit donc cette fois résoudre une étape de préconditionnement à la fin des itérations pour retrouver la solution du problème original. Nous verrons que, au moins dans le cas de GMRES, la méthode avec préconditionnement peut être mise en oeuvre sans faire référence à la variable auxiliaire y.

Enfin, les deux idées précédents peuvent être combinées. On prend deux matrices M_1 et M_2 et on résout le système

$$M_1^{-1}AM_2^{-1}y = M_1^{-1}, \quad M_2x = y. \tag{3.4}$$

L'approche bilatérale est parfois utile quand la matrice de préconditionnement est obtenue sous la forme d'un produit :

$$M = LU,$$

où L est U peuvent être (mais ne sont pas obligatoirement) des matrices triangulaires, voir le paragraphe 3.3.3. Dans le cas du gradient conjugué, cette approche est nécessaire pour maintenir la symétrie du système préconditionné.

Quelles sont les similarités, et les différences, entre le préconditionnement à gauche et à droite ? Notons tout d'abord que les matrices $M^{-1}A$ et AM^{-1} sont semblables, et ont donc les même valeurs propres (dans les deux cas ce sont les valeurs propres du problème généralisé $Ax = \lambda Mx$). Si la convergence est gouvernée par les valeurs propres, les deux approches donneront une convergence similaire.

Dans le cas d'un système non–symétrique résolu par GMRES, on démontre (voir Saad [21, Prop. 9.1], que la méthode avec préconditionnement à droite minimise encore le résidu de la méthode originale, alors que la méthode avec préconditionnement à gauche minimise la norme du résidu préconditionné. Les deux quantités sont souvent proches, sauf dans le cas où la matrice de préconditionnement M est mal conditionnée.

L'algorithme GMRES avec préconditionnement est presque identique à l'algorithme sans préconditionnement. La seule différence est de remplacer dans l'algorithme 2.3, la ligne

$$w_{k+1} = Av_k$$

par

$$w_{k+1} = M^{-1}Av_k \text{ pour un préconditionnement à gauche,}$$

ou par

$$w_{k+1} = AM^{-1}v_k \text{ pour un préconditionnement à droite.}$$

3.2 Préconditionnement du gradient conjugué

Le cas du gradient conjugué demande une attention particulière, et nous verrons que la mise en oeuvre de la méthode est particulièrement simple.

Rappelons que le gradient conjugué ne peut fonctionner qu'avec une matrice symétrique et définie positive. A première vue, ni le préconditionnement à droite, ni à gauche ne sont possibles. Nous nous placerons donc dans le cas d'un préconditionnement bilatéral, où la matrice M est décomposée sous la forme $M = LL^T$ (la matrice L sera souvent obtenue par une factorisation incomplète, voir paragraphe 3.2, mais cette hypothèse n'est pas nécessaire ici). Si la matrice L est inversible (ce qui l'hypothèse minimale), alors M sera définie positive, et inversement, toute matrice définie positive peut se mettre sous la forme $M = LL^T$. Le système préconditionné est alors

$$L^{-1}AL^{-T}\hat{x} = L^{-1}b, \quad L^T x = \hat{x}, \tag{3.5}$$

et nous noterons avec des chapeaux toutes les quantités transformées :

$$\hat{A} = L^{-1}AL^{-T}, \quad \hat{b} = L^{-1}b.$$

La matrice $\hat{A}$ est symétrique et définie positive. Mais nous allons voir que l'introduction de ces quantités n'est utile qu'au niveau théorique, et que l'algorithme du gradient conjugué préconditionné par M peut se réécrire entièrement avec les données originales, et que seule la matrice M est nécessaire.

Pour voir cela, définissons les quantités liées au système préconditionné :

$$\hat{x}_k = L^T x_k, \quad \hat{p}_k = L^T p_k,$$

Pour le résidu, on a

$$\hat{r}_k = \hat{b} - \hat{A}\hat{x}_k = L^{-1}b - L^{-1}AL^{-T}L^T x_k = L^{-1}r_k,$$

et il sera utile d'introduire le vecteur z_k défini par

$$\hat{r}_k = L^T z_k, \text{ de sorte que } L^{-1}r_k = L^T z_k, \text{ soit } Mz_r = r_k.$$

Nous calculons maintenant le scalaire

$$\hat{\alpha}_k = \frac{(\hat{r}_k \hat{r}_k,)}{(\hat{A}\hat{p}_k, \hat{p}_k)} = \frac{(L^T z_k, L^T z_k)}{(L^{-1}AL^{-T}L^T p_k, L^T p_k)} = \frac{(z_k, r_k)}{(Ap_k, p_k)},$$

de sorte que l'on a

$$\begin{aligned} \hat{x}_{k+1} &= \hat{x}_k + \hat{\alpha}_k \hat{p}_k \quad &\Longrightarrow x_{k+1} &= x_k + \hat{\alpha}_k p_k \\ \hat{r}_{k+1} &= \hat{r}_k - \hat{\alpha}_k A\hat{p}_k \quad &\Longrightarrow r_{k+1} &= r_k - \hat{\alpha}_k A p_k. \end{aligned} \tag{3.6}$$

De même,

$$\hat{\beta}_k = \frac{(\hat{r}_{k+1}, \hat{r}_{k+1})}{(\hat{r}_k, \hat{r}_k)} = \frac{(z_{k+1}, r_{k+1})}{(z_k, r_k)},$$

et

$$\hat{p}_{k+1} = \hat{r}_k + \hat{\beta}_k \hat{p}_k \Longrightarrow p_{k+1} = L^{-T}\hat{r}_k + \hat{\beta}_k p_k = z_k + \hat{\beta}_k p_k. \tag{3.7}$$

La conclusion de ces calculs est bien que l'algorithme du gradient conjugué préconditionné peut être mis en oeuvre en ajoutant simplement une étape supplémentaire à chaque itération de l'algorithme, pour calculer le vecteur z_k, appelé le *résidu préconditionné*. L'algorithme obtenu

Algorithme 3.1 Algorithme du gradient conjugué

Données: $b \in \mathbf{R}^n, x_0 \in \mathbf{R}^n, A \in \mathbf{R}^{n\times n}$
Résultats: k, x_k
$r_0 = b - Ax_0, z_0 = M^{-1}r_0, p_0 = r_0, k = 0$
Tant que $k \leq$ itmax et pas de convergence **faire**
$\alpha_k = \dfrac{(r_k,\, z_k)}{(p_k,\, Ap_k)}$
$x_{k+1} = x_k + \alpha_k p_k$
$r_{k+1} = r_k - \alpha_k A p_k$
$z_{k+1} = M^{-1} r_{k+1}$
$\beta_k = \dfrac{(r_{k+1},\, z_{k+1})}{(r_k,\, z_k)}$
$p_{k+1} = z_{k+1} + \beta_k p_k$
fin Tant que

est le suivant, à comparer avec l'algorithme (nous avons noté sans chapeau les deux scalaires α_k et β_k) :

Il n'est pas inutile d'insister sur le fait que, comme toujours, la notation $z_{k+1} = M^{-1}r_{k+1}$ est un raccourci mathématique, et que l'implémentation revient à résoudre, à chaque itération un système

$$Mz_{k+1} = r_{k+1}$$

pour le résidu préconditionné. La matrice de préconditionnement aura été choisie de manière à ce que ce système soit « facile à résoudre ». Nous présentons plusieurs méthodes au paragraphe 3.3.

3.3 Exemples de préconditionnement

Nous donnons dans ce paragraphe quelques exemples de préconditionnement, parmi les méthodes les plus simples. Nous ne pourrons ici encore, et plus qu'ailleurs qu'effleurer la surface d'un domaine très vaste, et faisant encore l'objet de recherches actuelles.

Avant de décrire quelques méthodes, notons que l'on peut grossièrement classer les méthodes de préconditionnement en deux catégories (la réalité est comme toujours plus subtile) :

— des méthodes généralistes, faisant peu d'hypothèses sur la matrice, ou l'origine du problème ;
— des méthodes plus spécialisées, s'appliquant à une catégorie restreinte de problèmes.

Dans ce cours, nous nous bornerons à quelques exemples de la première catégorie (utilisation de méthodes itératives, factorisation incomplètes, préconditionnement polynomial). Mais il est assez clair que plus on utilise une information spécifique au problème à résoudre, plus on a de chances de concevoir un préconditionneur efficace. On consultera les références données au début de ce chapitre pour découvrir certaines des méthodes qui ont été proposées.

3.3.1 Préconditionnement diagonal

Cette méthode n'est certainement pas la plus efficace, nous la mentionnons en premier pour sa simplicité, et parce qu'elle est très généralement applicable.

Si A est une matrice inversible, notons D sa diagonale principale. Nous faisons l'hypothèse que D est inversible (ou de manière équivalente que $D_{ii} \neq 0,\ foralli$. Cette hypothèse est en

particulier vérifiée si A est définie positive (pourquoi ?), et elle le sera très souvent pour des matrices provenant de la discrétisation de modèles d'EDP. Un cas important où elle n'est *pas* vérifiée es celui des matrices de forme « point selle » :

$$A = \begin{pmatrix} P & Q \\ R & 0 \end{pmatrix}$$

qui interviennent dans les problèmes d'optimisation avec contraintes, et également dans les problèmes de Stokes ou Navier–Stokes (on a le plus souvent $P = R^T$), voir **??**.

Le préconditionnement diagonal consiste dans le choix $M = D$. L'étape de mise en place initiale est ici triviale, et l'étape de préconditionnement consiste simplement à effectuer une boucle sur toutes les inconnues : résoudre $Mz = r$ est équivalent à

$$z_i = \frac{r_i}{D_{ii}}, \ \forall i = 1, \ldots, n. \tag{3.8}$$

Si la matrice A est symétrique et définie positive, ses éléments diagonaux sont tous strictement positifs, ce qui fait que nous sommes dans la situation de la Section 3.2.

L'intérêt de ce préconditionneur est qu'il conduit à une mise à l'échelle des coefficients de la diagonale de A qui peut parfois améliorer le comportement des algorithmes. Mais cette remarque n'est pas toujours vérifiée (exemple du problème de Poisson, la diagonale est égale à 4 fois l'identité). On peut montrer que dans certains cas, la diagonale de A est la matrice diagonale qui réduit le plus le conditionnement parmi toutes les matrices diagonales. Mais cette réduction peut être très modeste (voir Meurant [19, Sec. 8.2] ou Greenbaum [12, Sec. 10.5] pour plus de détails).

3.3.2 Préconditionnment par des méthodes itératives classiques

Nous allons voir que les méthodes itératives dites « classiques », comme Jacobi, Gauss–Seidel ou SSOR (voir Section A.5) conduisent à des préconditionnement.

Nous décomposons la matrice A sous la forme

$$A = M - N$$

où M sera supposée inversible. On obtient une classe de méthodes itératives en résolvant le système équivalent $x = (M^{-1}N)x + M^{-1}b$ par une méthode de point fixe, qui conduit à l'itération

$$x^{k+1} = (M^{-1}N)x^k + M^{-1}b, \quad x^0 \text{ donné.} \tag{3.9}$$

Pour mettre en évidence le lien avec les méthodes de préconditionnement, nous éliminons la matrice N en remarquant que $N = M - A$, et donc l'itération (3.9) est équivalente à

$$x^{k+1} = x^k + M^{-1}(b - Ax^k), \tag{3.10}$$

et l'on reconnaît une itération de point fixe pour résoudre le système préconditionné (3.1). Plutôt que d'utiliser l'itération simple (dont la convergence est en général lente), on résout le système préconditionné par une méthode de Krylov (gradient conjugué ou GMRES), en utilisant la matrice M comme matrice de préconditionnement. À chaque itération de la méthode de Krylov, on effectue *une seule itération* de la méthode classique, en résolvant le système $Mz = r^k$.

Les méthodes classiques correspondent à des décompositions particulières. Suivant la tradition, nous notons $A = D - E - F$, avec D diagonale, E la sous–diagonales stricte de A, et F la sur–diagonale (stricte) de A. Si A est symétrique, on a $F = E^T$.

Jacobi On prend $M = D$, on retrouve le préconditionnement diagonal du paragraphe précédent ;

Gauss-Seidel On prend $M = D - E$. Le système $Mz = r$ est facile à résoudre, puisque la matrice M est ici triangulaire inférieure. Notons que si la matrice A est symétrique et définie positive, on doit utiliser la version symétrique de la méthode de Gauss–Seidel. Elle correspond au choix $\omega = 1$ du point suivant ;

SOR Dans le cas où la matrice A n'est pas symétrique, on prend $M = \dfrac{1}{\omega}(D - \omega E)$, M reste triangulaire inférieure. Dans le cas symétrique, on doit utiliser la version symétrique de SOR (SSOR), et on a alors :

$$M = (D - \omega E)D^{-1}(D - \omega E^T).$$

(nous avons supprimé le facteur $\omega(2 - \omega)$ puisque le préconditionneur n'est pas sensible à un changement de facteur d'échelle). On constate que la matrice M est un produit de matrices triangulaires (et diagonale), donc toujours *facile à inverser*. En détail, la résolution de $Mz = r$ pour le préconditionnement SSOR s'effectue de la manière suivante :

$$\begin{aligned} &\text{résoudre } (D - E) \;\; u = r \\ &\qquad\qquad\qquad\quad v = Du \\ &\text{résoudre } (D - E)^T z = v \end{aligned} \tag{3.11}$$

Il existe peu de résultats théoriques concernant ce type de préconditionnement. Dans le livre[12, Sec. 10.4], A. Greenbaum montre comment généraliser un résultat de comparaison des rayons spectraux pour comparer des conditionnements.

3.3.3 Factorisations incomplètes

Les méthodes de factorisation incomplète restent les plus utilisées parmi les méthodes généralistes, c'est-à-dire applicable uniquement avec la connaissance de la matrice.

Le point de départ de ces méthodes est la constatation que la factorisation LU (ou de Cholesky) d'une matrice creuse contient en général *beaucoup plus* d'éléments non–nuls que la matrice de départ. On peut alors chercher à calculer une factorisation *approchée*, dont les facteurs L et U contiennent moins d'éléments non–nuls que les « vrais » facteurs. Il existe plusieurs variantes de la méthode, avec de degrés de sophistication croissants. Nous nous contenterons comme d'habitude de présenter la variante la plus simple, appelée ILU(0).

Factorisation incomplète ILU(0)

Dans cette méthode, les matrices L et U (comme d'habitude L est triangulaire inférieure à diagonale unité, et U est triangulaire supérieure) remplissent les conditions suivantes :

$$\begin{cases} l_{ij} = u_{ij} = 0 & \text{si } a_{ij} = 0 \\ (LU)_{ij} = a_{ij} & \text{si } a_{ij} \neq 0. \end{cases} \tag{3.12}$$

Notons bien la différence avec une factorisation complète : la matrice A n'est *pas* égale au produit LU. Il existe une matrice R (avec $R_{ij} = 0$ si $a_{ij} \neq 0$) telle que

$$A = LU + R.$$

Par contre, on contrôle a priori la mémoire utilisée par la méthode.

On peut obtenir les facteurs incomplets de A par une modification simple de l'algorithme de factorisation de Gauss (sans pivotage). Dans l'algorithme 3.2, les seules modifications sont les deux tests « **si** $a_{ij} \neq 0$ ».

Algorithme 3.2 Factorisation incomplète ILU(0)

Données: $A \in \mathbf{R}^{n\times n}$
Résultats: Facteurs L et U de la factorisation incomplète, vérfiiant (3.12).

```
      // La factorisation est « sur place » : les matrices L et U écrasent A
  pour k = 1, ..., n − 1 faire
    pour i = k + 1, ..., n faire
      si a_ij ≠ 0 alors a_ik = a_ik/a_kk
      pour j = k + 1, ..., n faire
        si a_ij ≠ 0 alors a_ij = a_ij − a_ik a_kj
      fin pour
    fin pour
  fin pour
```

Il n'est pas évident que l'algorithme 3.2 permette toujours de calculer les matrices L et U, autrement dit que les conditions (3.12) déterminent ces matrices. Et en fait, ce n'est pas toujours possible, et l'algorithme 3.2 n'est pas un algorithme sens strict, il ne se termine pas toujours.

Nous présentons donc une condition suffisante relativement simple pour que l'algorithme 3.2 se termine. On pourra consulter [21, Sec. 10.3], [24, Chap. 13] pour plus de détails.

Définition 3.1. Une matrice (inversible A) est une M-matrice si et seulement si elle vérifie les conditions suivantes :

i) $a_{ij} \leq 0,\ \forall i \neq j$;

ii) tous les éléments de A^{-1} sont positifs.

Remarque 3.1. La condition **ii)** ci–dessus est équivalente à

$$\text{Si } x \text{ est la solution de } Ax = b, \text{ et si } b_i \geq 0, \forall i, \text{ alors } x_i \geq 0, \forall i.$$

Il s'agit d'une forme discrète du principe du maximum.

Exemple 3.1.
— Si A est une matrice symétrique et définie positive, dont tous les éléments hors–diagonaux sont négatifs, alors A est une M-matrice.
— Si A est une matrice avec $a_{ii} > 0,\ \forall i$ et $a_{ij} \leq 0,\ \forall i \neq j$, telle que

$$|a_i| > \sum_{i\neq j} |a_{ij}|\,, \tag{3.13}$$

alors A est une M-matrice.
La condition (3.13) exprime que A est *à diagonale strictement dominante*

On démontre alors le résultat suivant (voir [21, Thm. 10.2] ou[24, Thm. 13.3] pour la démonstration) :

Théorème 3.1. *Soit $A \in \mathbf{R}^{n\times n}$ une M-matrice (inversible). L'algorithme 3.2 calcule des matrices L (triangulaire inférieure, à diagonale unité) et U (triangulaire supérieure) vérifiant les conditions* (3.12).

Dans le cas où A est symétrique et définie positive, il est naturel de chercher une matrice L (triangulaire inférieure, avec des éléments diagonaux positifs) telle que

$$\begin{cases} l_{ij} = 0 & \text{si } a_{ij} = 0 \\ (LL^T)_{ij} = a_{ij} & \text{si } a_{ij} \neq 0. \end{cases} \tag{3.14}$$

La factorisation obtenue s'appelle une *factorisation de Cholesky incomplète*, et existe si A est une M-matrice. L'algorithme 3.2 peut être adapté à cette situation.

Nous verrons que, dans certaines situations, il peut être plus commode de modifier la factorisation de sorte que l'on ait

$$A = (L + D)D^{-1}(L + D)^T + R$$

où les éléments diagonaux de L sont cette fois égaux à 1 (D est toujours la diagonale de A).

Un exemple : factorisation incomplète pour le problème modèle

Nous présentons dans ce paragraphe un exemple de calcul de factorisation incomplète. La motivation vient du problème modèle, la discrétisation par différences finies de l'équation de Poisson sur un carré, avec un maillage régulier de pas h. Pour simplifier, nous supposons que le schéma utilisé conduit à une matrice pentadiagonale, comme à l'exemple 1.1, ou la figure 3.1.

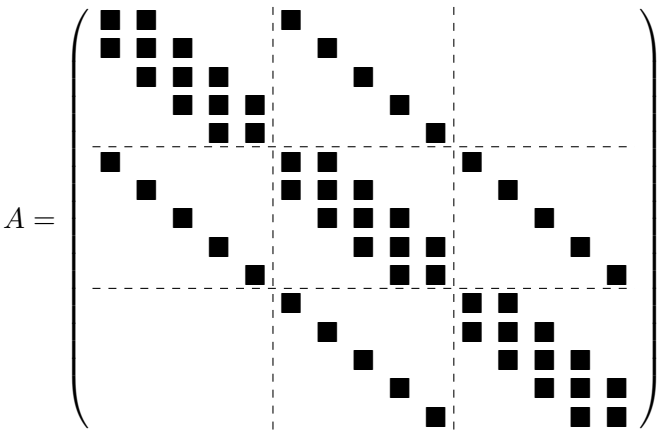


FIGURE 3.1 – Un exemple de matrice penta–diagonale

Nous décomposons la matrice A sous la forme

$$A = L + D + L^T \tag{3.15}$$

où D est la diagonale de A et L sa partie strictement triangulaire inférieure. Pour le problème modèle, tout les éléments diagonaux sont égaux à 4, et tous les éléments hors–diagonaux sont

0. L n'est donc *pas* la même matrice que celle de (3.12)

égaux à -1. Cependant, la valeur de ces éléments ne joue aucun rôle, seule la *structure* de la matrice A est importante, c'est-à-dire que nous considérons une matrice A décomposée comme en (3.15), où la matrice L n'a que deux diagonales non–nulles. Ces diagonales sont situées à une distance de 1 et de n de la diagonale principale (n est le nombre de points de grilles, sans compter les bords, la matrice A est de taille $n^2 \times n^2$. Nous ferons l'hypothèse que A est symétrique et définie positive (c'est le cas pour le problème modèle), et nous chercherons donc une factorisation de Cholesky incomplète, notée IC(0).

Pour cet exemple, il est commode de modifier légèrement la forme de la factorisation incomplète. Nous cherchons la matrice de préconditionnement M sous la forme factorisée

$$M = (L_M + \Delta)\Delta^{-1}(L_M + \Delta)^T,$$

ou Δ est diagonale et L_M strictement triangulaire inférieure.

La condition que M n'aie d'éléments non–nuls qu'aux mêmes places que A force L_M a n'avoir que deux diagonales non nulles, également aux distances 1 et n de la diagonale principale. Donc sur une ligne $1 \leq i \leq n^2$ donnée, les seuls éléments non nuls de L_M sont ceux situés en $(i, i-1)$ et $(i, i-n)$. Nous adoptons pur la suite de cet exemple la convention qu'un indice nul ou négatif n'est pas pris en compte.

En effectuant le produit des matrices définissant M nous obtenons

$$M = L_M + L_M^T + \Delta + L_M\Delta^{-1}L_M^T.$$

Les calculs se simplifient si les choses sont prises dans le bon ordre. Nous cherchons tout d'abord la *structure* des différents termes, sans nous soucier de calculer les valeurs.

— C'est simple pour les 3 premiers termes : le premier et le second n'ont, par construction, que deux diagonales non nulles, aux mêmes places que L. Et le troisième est diagonal ;

— L'élément générique du dernier terme (appelons H cette matrice) est :

$$h_{ij} = \sum_{k=1}^{n^2} (L_M)_{ik}(L_M)_{jk}\Delta_{kk}^{-1}. \tag{3.16}$$

Mais du fait de la structure de L_M, k ne peut prendre qu'un nombre très restreint de valeurs : on doit avoir $k = i-1$ ou $k = i-n$, et symétriquement, $k = j-1$ ou $k = j-n$. Pour que ces conditions soient compatibles, il faut que, soit $j = i$, soit $j = i-n+1$ (avec $j \leq i$). Par conséquent, les termes non nuls de H sont, soit sur la diagonale principale, soit sur les diagonales situés à une distance de $n-1$ de la diagonale principale.

Nous voyons maintenant comment les différents termes du produit définissant M contribuent. Les deux diagonales à la distance de $n-1$ de la diagonale principale n'existent pas dans A, et ne vont donc pas contribuer au préconditionneur. C'est la deuxième condition (3.12) qui l'impose. Ces deux termes constituent les éléments de la matrice R définie par $A = M + R$.

Ensuite, les seuls éléments venant se placer sur les diagonales 1 et n sont ceux provenant de L_M. Par conséquent (toujours à cause de la deuxième condition (3.12), nous devons avoir

$$L_M = L,$$

et ceci sans aucun calcul. C'est ce qui explique la forme choisie pour la factorisation.

Enfin, il nous reste à déterminer la matrice diagonale Δ. Pour cela nous exploiterons l'égalité des éléments diagonaux : $M_{ii} = A_{ii}$, pour $1 \leq i \leq n^2$. D'après (3.16), les éléments diagonaux de H sont donnés par

$$h_{ii} = l_{i,i-1}^2/\Delta_{i-1,i-1} + l_{i,i-n}^2/\Delta_{i-n,i-n},$$

et on doit donc avoir (en notant simplement Δ_i l'élément diagonal de la ligne i.

$$a_{i,i} = \Delta_i + l^2_{i,i-1}/\Delta_{i-1} + l^2_{i,i-n}/\Delta_{i-n,}, \tag{3.17}$$

ce qui donne une relation de récurrence pour $\Delta_{i,i}$ (l'initialisation se fait en prenant en compte que les éléments d'indice négatifs ou nuls sont absents). En raisonnant sur la grille (avec une numérotation par ligne de gauche à droite et de bas en haut), on voit que l'élément de Δ correspondant à un sommet se calcule en fonction de quantités associées aux points directement en bas, et à gauche.

On pourra trouver dans [21, Sec. 10.3.4] une approche plus générale, qui se base directement sur la grille de différences finies.

Généralisations

La méthode de base que nous avons exposée peut être généralisée dans différentes directions :

- On peut autoriser plus d'éléments non–nuls dans L et U que dans A. Pour cela, on se fixe un sous–ensemble $S \subset \{(i,j), i \neq j, 1 \leq i,j \leq n\}$ de positions où l'on impose $l_{ij} = u_{ij} = 0$ (on exclut toujours la diagonale). Le cas particulier vu précédemment est $S = \{(i,j)\ a_{ij} = 0\}$. Dans l'algorithme 3.2 il suffit alors de remplacer la condition $a_{ij} \neq 0$ par $(i,j) \notin S$. Le théorème 3.1 s'étend également, sous les mêmes hypothèses ;
- On peut également décider au cours du calcul quels éléments seront conservés, par exemple sur un critère de taille (factorisation avec seuil) ;
- on peut sommer tous les éléments que l'on ne conserve pas, et ajouter la somme à la diagonale de L ou de U. On parle de factorisation modifiée ;
- si la factorisation échoue (lorsque la matrice A n'est pas une M–matrice), on peut ajouter à A un multiple de l'identité, en espérant que la factorisation aboutisse. La justification est que de toutes façons la méthode repose sur des heuristiques.

Pour ces différents aspects, on pourra consulter des références plus spécialisées, par ordre de difficulté approximativement croissante : [21, Chap. 10], [24, Chap. 13], [19, Chap. 8], [3, Chap. 7].

3.3.4 Préconditionnement polynomial

Une autre classe assez générale de préconditionneurs est le préconditionnement polynomial. On cherche ici une matrice de préconditionnement M de la forme

$$M = p(A),$$

où p est un polynôme. Le système préconditionné est donc

$$p(A)Ax = p(A)b$$

et l'on remarque que comme $p(A)$ et A commutent, il est indifférent que le préconditionnement soit appliqué à droite ou à gauche.

Une première idée est d'utiliser la série de Neumann. Rappelons le résultat suivant :

Lemme 3.1. *Soit G une matrice vérifiant $\|G\| < 1$ pour une norme subordonnée (cf. Lemme A.1). On a l'égalité*

$$(I-G)^{-1} = \sum_{k=0}^{\infty} G^k = I + G + G^2 + \ldots + G^k + \ldots \tag{3.18}$$

où la série est absolument convergente dans $\mathcal{L}(\mathbf{R}^n)$.

Il est alors naturel de tronquer la série, et de proposer la somme $G_k = I + G + \ldots + G^k$ comme approximation de l'inverse de G.

Pour appliquer cette idée à une matrice quelconque A, il est nécessaire d'introduire un facteur de mise à l'échelle ω. La remarque simple que (on désigne toujours par D la diagonale de A)

$$\omega A = D - (D - \omega A)$$

montre que (prendre l'inverse des deux membres pour vérifier cette égalité) :

$$(\omega A)^{-1} = \left[I - (I - \omega D^{-1} A)\right]^{-1} D^{-1}$$

En appliquant le Lemme 3.1 à la matrice

$$G = I - \omega D^{-1} A,$$

on définit donc M^{-1} par

$$M^{-1} = (I + G + \ldots + G^k) D^{-1}. \tag{3.19}$$

Puisque par définition $D^{-1}A = 1/\omega(I - G)$, on peut vérifier que

$$M^{-1} A = \frac{1}{\omega}(I - G^{k+1}).$$

L'évaluation de G^{k+1} peut être numériquement délicate, la formule du binôme est à éviter !

Nous supposons maintenant que A est symétrique et définie positive, et nous noterons $0 < \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$ ses valeurs propres. Les valeurs propres de la matrice $p(A)A$ sont donc les nombres $\lambda_i p(\lambda_i)$ mais l'ordre n'est pas nécessairement préservé. L'utilisation de la série de Neumann revient à privilégier l'information spectrale autour de 0. Il peut être préférable de considérer le spectre de $p(A)A$ dans son ensemble, et de chercher un polynôme p qui rende le spectre aussi proche de 1 que possible. Notons $q(\lambda) = \lambda p(\lambda)$, et remarquons qu'on doit avoir $q(0) = 0$. Comme lorsque nous avons étudié la convergence du gradient conjugué (Section 1.4), il n'est pas possible de travailler sur le spectre discret, nous ferons donc l'hypothèse que nous connaissons un intervalle $[a, b]$, avec $a > 0$, contenant toutes les valeurs propres de A (cette hypothèse n'est pas particulièrement réaliste). On cherche donc à minimiser l'erreur $\|1 - q\|$, sur l'ensemble des polynômes de degré fixé (mais pas connu a priori), tels que $q(0) = 0$. Les deux choix les plus courants sont :

le polynôme du minimax on prend

$$\|1 - q\|_\infty = \max_{[a,b]} |1 - q(\lambda)| .$$

On montre que la solution est donnée encore une fois par un polynôme de Tchebychev (voir [21, Sec. 12.3] ou [19, Sec. 8.18] pour les détails ;

le polynôme des moindres carrés on prend

$$\|1 - q\|_2^2 = \int_a^b |1 - q(\lambda)|^2 \, d\lambda.$$

Il existe une description explicite du polynôme optimal, mais il peut aussi être intéressant de résoudre le problème de moindres carrés numériquement. Nous renvoyons également à [21, Sec. 12.3] ou [19, Sec. 8.18] pour les détails.

L'avantage du préconditionnement polynômial est qu'il se prête facilement à une mise en oeuvre parallèle, puisque tout ce qui est demandé est de multiplier la matrice A par un vecteur. Ceci est en contraste avec les méthodes de factorisation incomplètes, qui demandent la résolution de systèmes triangulaires, une étape qui est intrinsèquement séquentielle. Le principal inconvénient de ces méthodes est qu'elles reposent sur une estimation des valeurs propres de A, et qu'une telle estimation est difficile à obtenir. Des méthodes adaptatives (où l'estimation des valeurs propres est améliorée au cours des itérations) existent, mais leur implémentation est complexe.

Pour conclure, rappelons comment on peut évaluer le résidu préconditionné. Pour calculer $z = p(A)r$, où $p(\lambda) = \sum i = 0^k \alpha_i \lambda^i$, il ne faut surtout pas calculer les puissances de A, il est recommandé d'utiliser la méthode Horner, comme dans l'algorithme 3.3 (cf. [19, 8.18.4]).

Algorithme 3.3 Algorithme de Horner pour le calcul de $z = \sum_{i=0}^{k} \alpha_i A^i r$

$b_k = \alpha_k r$
pour $i = k - 1, \ldots, 0$ **faire**
  $b_i = \alpha_i r + A b_{i+1}$
fin pour
$z = b_0$

Chapitre 4

Matrices creuses

Nous donnons dans ce chapitre quelques indications sur les structures de données permettant de stocker efficacement des matrices creuses, et les algorithmes pour exploiter ces structures de données.

4.1 Introduction

Il n'existe pas de définition mathématique d'une matrice creuse, mais on peut donner une « définition » opérationnelle :

Définition 4.1. Une matrice creuse est une matrice dont un très grande partie des éléments sont nuls, et pour laquelle une représentation informatique qui ne stocke que les éléments non–nuls conduit à un gain en performance, en mémoire ou en temps de calcul.

Cette définition reste imprécise sur « la très grande partie », et sur la manière d'exploiter l'information. Il est clair que le stockage seul ne suffit pas, et que l'exploitation pour utiliser la matrice dans un code de calcul est critique.

De manière pragmatique, on sait que certaines applications conduisent à des matrices creuses, et on sait stocker ces matrices de manière à pouvoir réaliser les opérations les plus communes. On doit donc se poser non–seulement la question de comment stocker de manière économique ces matrices, mais également celle des algorithmes permettant de réaliser les opérations nécessaires.

L'une des applications rencontrée le plus fréquemment est la discrétisation d'équations aux dérivées partielles par des méthodes comme les différences finies, les éléments finis ou les volumes finis. Dans les trois cas, on obtient une matrice creuse car la discrétisation est locale. Les inconnues en un point ne sont reliées qu'à un petit nombre d'inconnues, en général géométriquement proches. Dans le cas des différences finies, ces inconnues correspondent aux voisins immédiats dans les directions d'espace (parfois à des voisins à deux niveaux). Dans le cas des éléments finis, les inconnues en un point géométrique ne sont reliées qu'à un petit nombre de celles coreepondant à des points partageant un élément avec le point considéré.

Notons toutefois que cette proximité géométrique ne se traduit pas obligatoirement par une proximité des inconnues discrètes, voir par exemple la matrice obtenue par la dsicrétisation du Laplacien par le schéma de différences finies « à 5 points ». Avec la numérotation habituelle (par ligne), si les voisins horizontaux sont numérotés consécutivement, les voisins verticaux sont à une distance qui augmente avec le nombre de points de discrétisation. Ce qui reste vrai est que

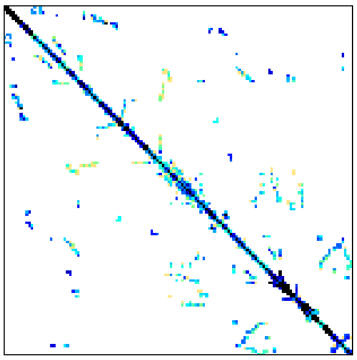
le nombre d'inconnues par ligne de la matrice est borné indépendamment du nombre de points de discrétisation.

Enfin, notons que c'est la combinaison de l'application et de la méthode de discrétisation qui conduit à une matrice creuse. Les méthodes intégrales ou les méthodes spectrales conduisent à des matrices pleines, mais de dimension plus faible.

Une autre application, avec laquelle l'auteur est moins familier, provient des matrices représentant des graphes, que l'on rencontre maintenant de plus en plus souventdans les applications étudiant différentes formes de réseaux (internet, réseaux sociaux, ...).

Nous montrons sur la figure 4.1 deux exemples de matrices creuses. Elles proviennent toutes deux de la collection de matrices creuses « SuiteSparse Matrix Collection [1] » [9]. L'Image de gauche est la matrice `Boeing/bcsstk36`. Elle provient de la discrétisation par éléments finis d'un absorbeur de chocs automobile. La matrice est de taille 23052, et a 1143140 éléments non nuls, ce qui représente 0,2%, une économie d'un facteur 500.

L'image de droite est la matrice `SNAP/soc-sign-bitcoin-alpha`. Elle provient de la description du réseau de confiance sur une plateforme d'échange de bitcoinsSNAP/soc-sign-bitcoin-alpha . Elle est de taille 3783, avec 24186 éléments non–nuls, soit seulement 0,17%. Elle est proportionnellement plus creuse que la précédente, ce qui ne saute pas aux yeux !



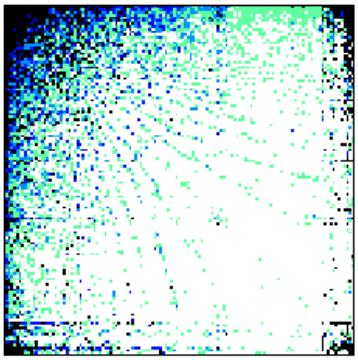


FIGURE 4.1 – Deux exemples de matrice creuses

Ce chapitre sera une brève introduction au sujet. Il existe un grand nombre de modes de stockage différents, chacun possédant des variantes. Nous ne chercherons pas à être exhaustif, et ne présenterons que deux modes de stockage, avec les algorithmes associés :

le stockage par diagonales qui est bien adapté aux matrices provenant de discrétisation d'EDP par différences finies sur des grilles régulières. Si ce mode est moins utilisé, il reste toujours utile, et permet une introduction simple des algorithmes ;

le stockage dit « CSR » ou Compressed Sparse Rows, qui signifie stockage par lignes creuses compressées. Ce mode permet de stocker des matrices creuses générales.

On trouvera une descrption plus détaillée des modes de stockage et des algorithmes associés dans le livre de Y. Saad[21, Chap. 3]

1. `https://sparse.tamu.edu/`

4.2 Stockage par diagonales

Ce mode de stockage est adapté aux matrices dont les éléments sont disposés le long de diagonales parallèles à la diagonale principale. Comme signalé plus haut, c'est en particulier le cas des matrices obtenues par la discrétisation par différences finies d'une équation aux dérivées partielles sur une grille régulière. Dans le cas le plus simple, en dimension 1, on trouve l'exemple bien connu de la matrice tridiagonale (h est le pas de discrétisation).

$$A = \frac{1}{h^2} \begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix} \tag{4.1}$$

En dimension 2, la figure 3.1 montre une exemple d'une matrice de ce type.

4.2.1 Description

Pour décrire le stockage par diagonales, on considère une matrice $A \in \mathbf{R}^{n\times n}$, dont les éléments sont disposés le long de N_d dagonales, dont les positions sont connues. Nous suivrons la convention de Matlab qui consiste à repérer les diagonales par leur distance à la diagonale principale. Par convention, celle–ci est la diagonale 0, les diagonales supérieures ont des distances positives et les diagonales inférieures des distances négatives. Par exemple, la matrice de l'équation (4.1) a trois diagonales, en position -1, 0 et 1.

Le stocakge diagonal représente ce type de matrice par une structure consistant en

— un tableau à deux dimensions `ADIAG`, de taille $n \times N_d$, dont chaque colonne correspond à une diagonale ;
— un tableau `IOFF` (pour offsets, décalages) qui indique les numéros des diagonales.

Plus précisément, pour $1 \leq i \leq n$ et $1 \leq k \leq N_d$, `ADIAG`(i, k). contient l'élément $A_{i,i+\texttt{IOFF}(k)}$.

Remarque 4.1. Il est commode d'utiliser un tableau rectangulaire `ADIAG`, mais on voit que seule la diagonale principale est de taille n. Les autres diagonales sont incomplètes, soit « au début » pour les sous–diagonales, soit « à la fin » pour les sur–diagonales. Les positions correspondantes dans `ADIAG`, qui correspondraient soit à des indices négatifs, soit à des indices plus grands que n ne seont pas utilisés (et on ne doit pas y accéder).

Exemple 4.1.
Nous prenons l'exemple de la matrice du Laplacien sur une grille3×3 (dont le seul intéret est de pouvoir représenter toute la matrice !), qui donc une matrice 9×9. Au facteur 1/16 près, la

matrice est

$$A = \begin{pmatrix} 4 & -1 & & -1 & & & & & \\ -1 & 4 & -1 & & -1 & & & & \\ & -1 & 4 & & & -1 & & & \\ -1 & & & 4 & -1 & & -1 & & \\ & -1 & & -1 & 4 & -1 & & -1 & \\ & & -1 & & -1 & 4 & & & -1 \\ & & & -1 & & & 4 & -1 & \\ & & & & -1 & & -1 & 4 & -1 \\ & & & & & -1 & & -1 & 4 \end{pmatrix}$$

Les diagonales sont en positions $\texttt{Ioffs} = (-3, -1, 0, 1, 3)$, et le tableau `Adiag` contient

$$\texttt{Adiag} = \begin{pmatrix} * & * & 4 & -1 & -1 \\ * & -1 & 4 & -1 & -1 \\ * & -1 & 4 & 0 & -1 \\ -1 & 0 & 4 & -1 & -1 \\ -1 & -1 & 4 & -1 & -1 \\ -1 & -1 & 4 & 0 & -1 \\ -1 & 0 & 4 & -1 & * \\ -1 & -1 & 4 & -1 & * \\ -1 & -1 & 4 & * & * \end{pmatrix}$$

4.2.2 Algorithmes

L'algorithme le plus important pour exploiter le stockage creux est celui qui réalise le produit de la matrice par un vecteur donné. Nous avons vu aux chapitres 1 et 2 que cette opération permet d'implémenter les algorithmes du gradient conjugué et de GMRES. On note communément cette opération *matvec*. Nous noterons x le vecteur à multplier, et $y = Ax$ le résultat

Dans le cas du stockage par diagonales, il est naturel de boucler sur les diagonales. Comme on l'a vu plus haut, $\texttt{ADIAG}(i, k)$ contient l'élément $A_{i,i+d}$ avec $d = \texttt{IOFF}(k)$, et cet élément doit être multiplié par x_{i+d} pour s'ajouter au résultat y_i.

Pour écrire l'algorithme, nous devons prendre en compte la remarque 4.1, qui dit que i et $i + d$ doivent être compris 1 et n.

On voit que, pour les matrices qui se prêtent à ce mode de stockage, l'algorithme 4.1 est efficace. Le nombre d'opérations (additions et multiplications) est (presque) égal au nombre d'éléments non–nuls : chaque passage dans la boucle « pour i » multiplie le vecteur x par une diagonale (en ne tenant pas compte des éléments auxquels on n'accède pas). De plus, le sur–coût du au mode de stockage est négligeable, puisqu'il suffit de calculer d pour chaque diagonal, et que N_d est faible en pratique.

Les algorithmes pour résoudre des systèmes triangulaires dans le stockage par diagonales, qui sont nécéssaires dans beaucoup de préconditionnement, par exemple pour utiliser une fac-

Algorithme 4.1 Produit matrice–vecteur pour le stockage par diagonales

$y = 0$
pour $k = 1 : N_d$ **faire**
  $d = \texttt{IOFF}(k)$
  $i1 = \max(1, 1 - d); i2 = \min(n, n - d)$
  pour $i = i1 : i2$ **faire**
    $y(i) = y(i) + \texttt{ADIAG}(i, k)x(i + d)$
  fin pour
fin pour

torisation LU incomplète, sont plus difficiles à mettre en oeuvre. Il n'est pas facile, dans ce cas, d'exploiter la structure par diagonales, et il est nécessaire de procéder par ligne ou par colonne.

4.3 Stockage CSR

Ce format de stockage est celui qui est le plus utilisé pour des matrices creuses générales, telles que celles produites dans une méthode d'éĺéments finis. Il est utilisé, avec sa proche vaiante CSC, qui propose un stockage par colonnes, dans un grand nombre de logiciels ou de bibliothèques (c'est le stockage creux de Matlab, et Python utilise CSC).

Nous prendrons comme exemple la matrice suivante qui est naturellement trop petite pour que le stockage creux soit utile, mais qui permet une description explicite des structures de données.

$$A = \begin{pmatrix} A_{11} & 0 & 0 & A_{14} & 0 \\ A_{21} & A_{22} & 0 & A_{24} & 0 \\ A_{31} & 0 & A_{33} & A_{34} & A_{35} \\ 0 & 0 & A_{43} & A_{44} & 0 \\ 0 & 0 & 0 & 0 & A_{55} \end{pmatrix} \tag{4.2}$$

4.3.1 Stockage « coordonnées »

Avant de décrire le stockage CSR, nous discutons rapidement du stockage *coordonnées*, qui est le plus naturel, mais ne permet pas d'implémenter facilement les algorithmes comme le produit matrice vecteur.

Son principe est simple : l'élément A_{ij} de la matrice est représenté par ses « coordonnées » i et j, aisni que la valeur A_{ij}.

Nous noterons NNZ le nombre d'éléments non–nuls de la matrice. Cette quantité importante est, dans les cas que nous considérons, très inférieure au nombre d'éléments total de A, qui est n^2. Pour le stockage coordonnées, nous aurons besoin de trois tableaux de taille NNZ, qui seront notés IA (pour les indices de ligne), JA (pour les indices de colonne), et AA (pour les valeurs).

Pour $1 \leq k \leq \texttt{NNZ}$, $\texttt{AA}(k)$ contient la valeur de l'élément placé à la ligne $i = \texttt{IA}(k)$ et la colonne $j = \texttt{JA}(k)$. La figure 4.2 illustre cela pour notre matrice exemple.

Comme nous le voyons sur la figure 4.2, les éléments peuvent être dans n'importe quel ordre. Mais il est souvent commode de les trier, d'abord par ligne, puis par colonne croissante, comme

AA	A_{55}	A_{35}	A_{33}	A_{24}	A_{11}	A_{14}	A_{44}	A_{21}	A_{31}	A_{22}	A_{34}	A_{43}
IA	5	3	3	2	1	1	4	2	3	2	3	4
JA	5	5	3	4	1	4	4	1	1	2	4	3

FIGURE 4.2 – Stockage « ccordonnées » de la matrice (4.2), éléments non triés

illustré sur la figure 4.3.

AA	A_{11}	A_{14}	A_{21}	A_{22}	A_{24}	A_{31}	A_{33}	A_{34}	A_{35}	A_{43}	A_{44}	A_{55}
IA	1	1	2	2	2	3	3	3	3	4	4	5
JA	1	4	1	2	4	1	3	4	5	3	4	5

FIGURE 4.3 – Stockage « ccordonnées » de la matrice (4.2), éléments triés

Comme nous l'avons déjà noté, ce format n'est pas très maniable. Sa simplicité fait qu'il est souvent utilisé comme format d'entrée pour un logiciel avsnt d'être converti en interne en un format plus maniable, comme le format CSR. C'est le cas de Matlab : la commande `sparse` prend une description de la matrice au format coordonnées.

4.3.2 Le stockage CSR

Le stockage CSR se déduit par une variation simple du stockage coordonnée. Il conserve les deux tableaux `AA` et `JA` mais remplace le tableau des indices de ligne `IA` par un tableau de pointeurs vers le premier élément de chaque ligne. Comme nous le verrons plus loin, il permet de mettre en oeuvre de manière efficace la plupart des algorithmes utiliśe par les méhodes itératives.

Pour l'exemple de la matrice A de refeq :matfull, les trois tableaux sont

AA	A_{11}	A_{14}	A_{21}	A_{22}	A_{24}	A_{31}	A_{33}	A_{34}	A_{35}	A_{43}	A_{44}	A_{55}
JA	1	4	1	2	4	1	3	4	5	3	4	5
IA	1	3	6	10	12	13						

FIGURE 4.4 – Stockage CSR de la matrice (4.2)

Comme nous l'avons vu plus haut, les tableaux `AA` et `JA` sont les mêmes que sur la figure 4.3. Pour interpreter le tableau `IA`, notons que

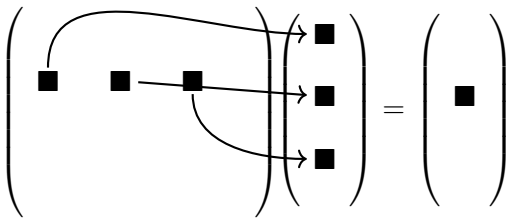
- la première ligne va de l'élément stocké en `IA(1)` $= 1$ dans `AA` à l'élément stocké en `IA`$(2) - 1 = 2$. La ligne 1 de A contient bien deux éléments, stockés dans les colonnes 1 et 4 ;
- la ligne 3 de A est entre les éléments `IA`$(3) = 3$ et `IA`$(4) - 1 = 9$ de IA, soit 4 éléments, rangés aux places 6, 7, 8 et 9. Les colonnes correspondantes sont 1,3, 4 et 5.

De manière générale : *les éléments de la ligne i sont stockés dans les tableaux* `JA` *et* `AA` *aux positions situées entre* `IA(i)` *et* $\texttt{IA}(i+1)-1$. *De plus, si* $j = \texttt{JA}(k)$, *pour* $\texttt{IA}(i) \leq k \leq \texttt{IA}(i+1)-1$, *alors* $AA(k)$ *contient* A_{ij}.

Comme son nom l'indique, le stockage CSR est orienté par lignes. Cela aura bien entendu une inluence sur la manière d'implémenter les différents algorithmes. Nous verrons qu'ils ont pour la plupart une structure commune, formée de deux boucles imbriquées. La première parcourt les lignes de la matrice, la seconde parcourt les éléments non–nuls ds la ligne en question.

Nous commencerons par l'algorithme qui est à la fois le plus simple et sans doute le plus important : le produit d'une matrice creuse A stockée en format CSR par un vecteur x, et nous noterons $y = Ax$ le vecteur résultat. Comme l'algorithme naturel procède en prenant le produit scalaire d'une ligne de A par x, la mise en oeuvre suit de près la description mathématique.

Algorithme 4.2 Produit matrice–vecteur pour le stockage CSR



$y = 0$
pour $i = 1, \ldots, n$ **faire**
  $k1 = \texttt{IA}(i),\ k2 = \texttt{IA}(i+1) - 1$
  pour $k = k1, \ldots, k2$ **faire**
    $j = \texttt{JA}(k); \quad y(i) = y(i) + \texttt{AA}(k)x(j)$
  fin pour
fin pour

Nous pouvons également écrire cet algorithme à un plus haut niveau, en remplaçant la boucle interne par une somme. Ces deux versions sont mathématiquement équivalentes, et il n'y a pas de raison objective de préféreer l'une à l'autre. Le niveau de détail supérieur dans la première version peut être vu comme un avantage ou un inconvénient.

Algorithme 4.3 Produit matrice vecteur en stockage CSR — version alternative

pour $i = 1, \ldots, n$ **faire**
  $k1 = \texttt{IA}(i),\ k2 = \texttt{IA}(i+1) - 1$
  $$y(i) = \sum_{k=k1}^{k2} \texttt{AA}(k)\, x(\texttt{JA}(k))$$
fin pour

Dans certains cas, il peut être nécessaire de calculer le produit $A^T x$ de la transposée de A par un vecteur, naturellement sans stocker explicitement A^T. Puisque A est stocké par lignes, il faut trouver un algorithme qui accede aux *colonnes* de A^T. Pour cela, nous devons interprérter le produit d'une matrice par un vecteur d'une manièere différente.

Soit dinc $B \in \mathbf{R}^n$ une matrice, $x \in \mathbf{R}^n$ un vecteur. Plus tard, B sera A^T, mais il est plus clair de généraliser la situation. En revenant à la définition du produit Bx comme l'application (dans une base fixée) de l'application linéaire représentée par la matrice B au vecteur x, nous voyons que le produit $y = Bx$ est une combinaison linéaire des colonnes de B par les coefficients de x. En notant $B = [b_1, \ldots, b_n]$ les colonnes de B, nous avons donc :

$$y = \sum_{j=1}^{n} x_j b_j,$$

ce qui conduit à un algorithme *par colonnes* pour le produit

pour $j = 1, \ldots, n$ **faire**
  pour $i = 1, \ldots, n$ **faire**
    $y_i = y_i + B_{ij} x_j$
  fin pour
fin pour

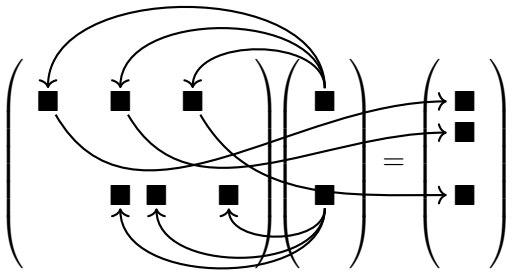
Dans le cas où la matrice B est A^T, on obtient, en échageant le rôle des indices i et j :

pour $i = 1, \ldots, n$ **faire**
  pour $j = 1, \ldots, n$ **faire**
    $y_j = y_j + A_{ij} x_i$
  fin pour
fin pour

et cette fois, on accède aux éléments de A par ligne.

En adaptant cet algorithme au stockage CSR, nous obtenons finalement

Algorithme 4.4 Produit de la transpoée d'une matrice par un vecteur en stockage CSR



$y = 0$
pour $i = 1, \ldots, n$ **faire**
  $k1 = \texttt{IA}(i),\ k2 = \texttt{IA}(i+1) - 1$
  pour $k = k1, \ldots, k2$ **faire**
    $j = \texttt{JA}(k); \quad y(j) = y(j) + \texttt{AA}(k) x(i)$
  fin pour
fin pour

Annexe A

Rappels d'algèbre linéaire

Ce chapitre contient des résultats utiles d'algèbre linéaire, rappelés sans démonstration. Pour plus de détails, on consultera les références suivantes [1, 8, 10, 15].

A.1 Matrices symétriques

Définition A.1. Une matrice $A \in \mathbf{R}^{n\times n}$ est symétrique si et seulement si :

$$A = A^T, \quad \text{c'est à dire : } a_{ij} = a_{ji}, \quad \forall (i,j) \in [1,n]$$

ou de manière équivalente :

$$(Ax, y) = (x, Ay), \quad \forall (x,y) \in \mathbf{R}^n,$$

où le *produit scalaire* est défini par

$$(x, y) = y^T x, \quad \forall (x,y) \in \mathbf{R}^n.$$

Rappelons que toutes les valeurs propres d'une matrice symétrique sont réelles, que les vecteurs propres correspondant à des valeurs propres distinctes sont orthogonaux, et qu'une matrice symétrique est diagonalisable dans une base orthonormée.

Définition A.2. Une matrice symétrique $A \in \mathbf{R}^{n\times n}$ est définie positive si et seulement si elle vérifie l'une des conditions (équivalentes) suivantes :

- **i)** $(Ax, x) > 0, \quad \forall x \in \mathbf{R}^n,\ x \neq 0$;
- **ii)** toutes les valeurs propres de A sont strictement positives.
- **iii)** Il existe une matrice $L \in \mathbf{R}^n$ *inversible et triangulaire inférieure* telle que $A = LL^T$ (factorisation de Cholesky).

A.2 Normes vectoriellles et matricielles

Nous rappelons dans ce paragraphe les principales propriétés des normes matricielles, essentiellement sans définition (voir les références générales citées précédemment).

Les normes permettent de « mesurer » la taille des éléments d'un espace vectoriel. Nous verrons au paragraphe A.3 qu'elles ont un rôle essentiel à jouer pour quantifier la sensibilité de la solution d'un système linéaire à des perturbations sur les données (second membre ou matrice).

Définition A.3. Une norme sur un espace vectoriel E est une application de E dans $\mathbf{R}^+$, notée $\|.\|$ telle que :

— $\forall (x,y) \in E^2, \quad \|x+y\| \leq \|x\| + \|y\|$;
— $\forall x \in E,\ \forall \lambda \geq 0, \quad \|\lambda x\| = |\lambda|\,\|x\|$;
— $\|x\| = 0 \Rightarrow x = 0$.

Exemple A.1.
Quand l'espace vectoriel E est de dimension finie ($E = \mathbf{R}^n$), les exemples les plus courants sont

— La norme euclidienne : $\|x\|_2 = \left(\sum_{i=1}^n |x_i|^2\right)^{1/2}$;
— La norme du max : $\|x\|_\infty = \max_{i=1,\ldots,n} |x_i|$;
— Plus généralement, la norme-p est définie pour $p \in [1,\ +\infty[$ par $\|x\|_p = (\sum_{i=1}^n |x_i|^p)^{1/p}$. La norme 2 en est un cas particulier.

Définition A.4. Étant donné une norme vectorielle, la quantité

$$\|A\| = \sup_{x \neq 0} \frac{\|Ax\|}{\|x\|} = \sup_{\|x\|=1} \|Ax\| \tag{A.1}$$

définit une norme sur l'espace vectoriel des matrices, dite *norme subordonnée* à la norme vectorielle.

Une norme vectorielle subordonnée vérifie l'inégalité

$$\|AB\| \leq \|A\|\,\|B\|\,, \tag{A.2}$$

pour deux matrices pour lesquelles le produit est défini, et en particulier

$$\|Ax\| \leq \|A\|\,\|x\|\,.$$

Dans le cas des normes p pour $p = 1$, $p = 2$ et $p = \infty$, on dispose d'expressions explicites pour les normes matricielles correspondantes :

Pour $p = 1$, $\|A\|_1 = \max_{1\leq j\leq n} \sum_{i=1}^m |a_{ij}|$;

Pour $p = 2$, $\|A\|_2 = \max_{1\leq i\leq n} \sqrt{\lambda_i}$, où λ_i est une valeur propre de $A^T A$;

Pour $p = \infty$, $\|A\|_\infty = \max_{1\leq i\leq m} \sum_{j=1}^n |a_{ij}|$.

Terminons en signalant qu'il existe des normes sur l'espace des matrices qui ne sont pas des normes subordonnées. L'exemple le plus courant est la *norme de Frobenius* :

$$\|A\|_F = \left(\sum_{i=1}^m \sum_{j=1}^n |a_{ij}|^2\right)^{1/2}.$$

Comme une norme subordonnée vérifie $\|I\| = 1$ (pourquoi ?), et que $\|I\|_F = n$, la norme de Frobenius ne peut-être une norme subordonnée. Par contre, elle vérifie l'inégalité (A.2).

Le Lemme suivant sera utilisé au paragrpahe A.3 pour déterminer la taille d'une perturbation pouvant laisser une matrice inversible.

Lemme A.1. *Soit $\|.\|$ une norme subordonnée, et A une matrice vérifiant $\|A\| < 1$. La matrice $I + A$ est inversible. De plus,*

$$\left\|(I+A)^{-1}\right\| \leq \frac{1}{1-\|A\|}. \tag{A.3}$$

Preuve. Raisonnons par l'absurde : si $I + A$ est singulière, soit x un vecteur non nul dans le noyau de cette matrice. On a alors

$$(I+A)x = 0 \Rightarrow Ax = -x$$

soit en passant aux normes

$$\|x\| = \|Ax\| \leq \|A\| \, \|x\| < \|x\|,$$

ce qui est évidemment une contradiction.

Pour prouver (A.3), notons $B = (I + A)^{-1}$. Partant de $B(I - A) = I$, il vient $B = I + BA$, puis en prenant les normes $\|B\| \leq 1 + \|B\| \, \|A\|$, ce qui donne (A.3). ■

A.3 Conditionnement d'un système linéaire

Le conditionnement d'un système linéaire (ou plus précisément d'une matrice) est un nombre qui mesure comment la solution de ce système varie si les données (le second membre, ou les coefficients de la matrice) sont perturbés. Cette mesure est une propriété intrinsèque du système, et ne dépend pas de la méthode de résolution.

Il sera commode de séparer l'effet des perturbations sur le secopnd membre et sur la matrice.

Définition A.5. Étant donné une matrice inversible $A \in \mathbf{R}^{n\times n}$, et une norme matricielle subordonnée $\|.\|$, le *conditionnement* de A, noté $\kappa(A)$ est le nombre défini par (pour la norme considérée) :

$$\kappa(A) = \|A\| \left\|A^{-1}\right\| \tag{A.4}$$

Théorème A.1. *Soit $A \in \mathbf{R}^{n\times n}$ une matrice inversible, et $b \in \mathbf{R}^n$ supposé non-nul. Nous considérons le système linéaire $Ax = b$, et des perturbations $\delta b \in \mathbf{R}^n$ et $\delta A \in \mathbf{R}^{n\times n}$.*

— *Pour le système perturbé $A\hat{x} = \hat{b}$, avec $\hat{b} = b + \delta b$, et $\hat{x} = x + \delta x$, l'erreur relative sur x vérifie :*

$$\frac{\|\delta x\|}{\|x\|} \leq \kappa(A)\frac{\|\delta b\|}{\|b\|}. \tag{A.5}$$

— *Pour le système perturbé $\hat{A}\hat{x} = b$, avec $\hat{A} = A + \delta A$ et $\hat{x} = x + \delta x$, où δA vérifie $\left\|A^{-1}\delta A\right\| < 1$, l'erreur relative sur x vérifie*

$$\frac{\|\delta x\|}{\|x\|} \leq \frac{\kappa(A)}{1-\kappa(A)\dfrac{\|\delta A\|}{\|A\|}}\frac{\|\delta A\|}{\|A\|}. \tag{A.6}$$

Dans le cas où la norme est une norme p, nous noterons κ_p le conditionnement.

Proposition A.1. *Propriétés du conditionnement. Soit $A \in \mathbf{R}^n$ une matrice inversible.*

— *Pour toute norme subordonnée $\kappa(A) \geq 1$;*

— *$\kappa(\lambda A) = \kappa(A)$, $\forall \lambda \in \mathbf{R}^*$;*

— *Pour la norme euclidienne, $\kappa_2(A) = 1$ si A est (proportionnelle à) une matrice orthogonale.*
— *Si A est une matrice symétrique et définie positive, $\kappa_2(A) = \dfrac{\lambda_n}{\lambda_1}$ où λ_n (resp. λ_1) est la plus grande (resp. petite) valeur propre de A.*

On dit qu'une matrice est *mal conditionnée* si son conditionnement est « très grand ». Dans ce cas, la solution d'un système linéaire est très sensible à des perturbations sur ses données.

A.4 Problèmes de moindres carrés

Nous rappelons dans ce paragraphe les principaux résultats concernant les problèmes de moindres carrés linéaires. Ces problèmes interviennent dans des domaines très divers chaque fois que l'on cherche à estimer des paramètres à partir de données expérimentales, dans le cas où le modèle est linéaire. L'utilisation d'un nombre de mesures plus grand que le nombre de paramètres permet de réduire l'influence des erreurs expérimentales.

Nous considérons donc une matrice $A \in \mathbf{R}^{m\times n}$, avec $m \geq n$ (et en général $m \neq n$), et un vecteur $\hat{b} \in \mathbf{R}^m$. Nous cherchons un vecteur $x \in \mathbf{R}^n$ solution du système linéaire

$$Ax = b.$$

Il est connu que, dans le cas ou $m > n$, ce système n'a en général pas de solution, et que si une solution existe (si $b \in \operatorname{Im} A$), celle-ci ne sera en général pas unique, mais définie à l'addition d'un élément de $\operatorname{Ker} A$ près. On va alors chercher un vecteur x tel que l'équation soit vérifiée « au mieux », en un sens qu'il faut naturellement préciser. Le choix le plus commun, dont l'une des justifications est son lien avec le minimum de vraisemblance est de minimiser la norme euclidienne du résidu. On résout donc le problème

$$\min_{x\in\mathbf{R}^n} \|Ax - b\|_2^2 \tag{A.7}$$

Comme nous ne ferons qu'effleurer le sujet, donnons quelques références qui permettent de l'approfondir . En français, citons les livre [15], ou [1]. En anglais, le classique est [10], une référence encyclopédique sur les problèmes de moindres carrés est [6]. Un autre livre, moins complets, mais plus abordable, est [23].

A.4.1 Propriétés mathématiques

Nous allons voir que le problème (A.7) est équivalent à une équation linéaire, mais pour un opérateur différent de A.

Théorème A.2. *Soit $A \in \mathbf{R}^{m\times n}$ et soit $b \in \mathbf{R}^m$. Un élément $\hat{x} \in \mathbf{R}^n$ est une solution de* (A.7) *si et seulement si*

$$A^T A\hat{x} = A^T b \tag{A.8}$$

Démonstration. On peut obtenir facilement (A.8) en calculant le gradient de la fonctionnelle $x \to \dfrac{1}{2}\|Ax - z\|^2$. Nous laissons cette méthode en exercice au lecteur. Nous présenterons plutôt l'argument élémentaire suivant, emprunté à Björck [6].

Soit x vérifiant (A.8). On a pour tout $y \in \mathbf{R}^n$:

$$b - Ay = b - Ax + A(x - y).$$

L'équation normale (A.8) implique que les deux termes de la somme sont orthogonaux (le résidu $b - Ax$ est orthogonal a l'image de A). Le théorème de Pythagore implique :

$$\|b - Ay\|_2^2 = \|b - Ax\|_2^2 + \|A(x - y)\|_2^2 \geq \|b - Ax\|_2^2 .$$

x est donc bien solution de (A.7)

Réciproquement, soit x tel que $A^T(b - Ax) = w \neq 0$. Choisissons $y = x + \varepsilon w$, avec $\varepsilon > 0$. On a alors :

$$\|b - Ay\|_2^2 = (b - Ay,\, b - Ay) = \|b - Ax\|_2^2 - 2\varepsilon(b - Ax,\, w) + \|Aw\|_2^2 < \|b - Ax\|_2^2$$

si ε est suffisamment petit. x n'est donc pas solution de (A.7). ■

Remarque A.1. L'équation normale (A.8) se réécrit :

$$A^T (A\hat{x} - b) = 0,$$

ce qui exprime simplement que le résidu $b - A\hat{x}$ est dans le noyau de A^T, c'est-à-dire orthogonal à l'image de A (voir le théorème A.3). Ceci conduit à l'illustration géométrique bien connue :

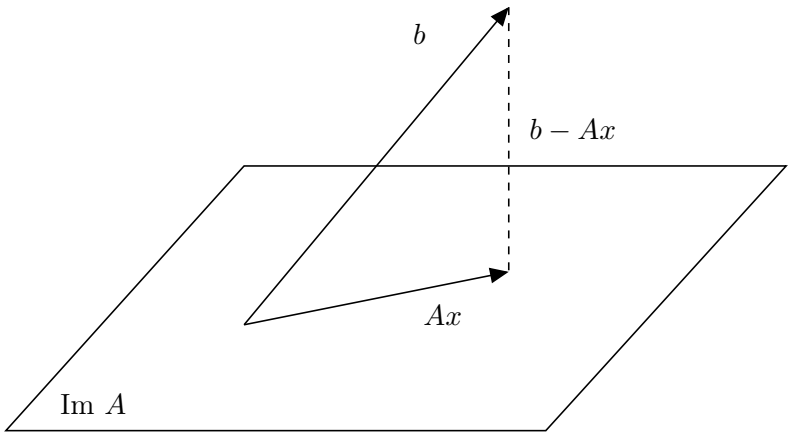


FIGURE A.1 – Illustration géométrique des moindres carrés

La solution du problème de moindres carrés est telle que Ax est la projection de b sur l'image de A.

Notons que nous n'avons pour l'instant évoqué ni l'existence, ni l'unicité, pour les solutions de (A.7) (ou de (A.8)). Nous avons simplement montré l'équivalence des deux problèmes. Nous allons préciser ces points dans le théorème suivant.

Théorème A.3. *Le problème* (A.7) *admet toujours une solution. De plus, cette solution est unique si, et seulement si, la matrice A est injective (ou ce qui est équivalent de rang maximal)*

Démonstration. Rappelons les résultats généraux d'algèbre linéaire :

$$\begin{aligned} \operatorname{Ker} A^T A &= \operatorname{Ker} A \\ \operatorname{Im} A^T A &= \operatorname{Im} A^T \end{aligned} \tag{A.9}$$

Un sens est évident à chaque fois. Pour l'autre sens, nous avons pour la première égalité :

$$A^T Ax = 0 \Rightarrow (A^T Ax,\, x) = 0 \Rightarrow (Ax,\, Ax)_{\mathbf{R}^m} = \|Ax\|_2 = 0 \Rightarrow Ax = 0.$$

La deuxième s'en déduit sans autre calcul car :

$$(\operatorname{Im} A^T) = (\operatorname{Ker} A)^\perp = (\operatorname{Ker} A^T A)^\perp = \operatorname{Im} A^T A.$$

Nous obtenons donc l'existence d'une solution, puisque le second membre de (A.8) est élément de $\operatorname{Im} A^T = \operatorname{Im} A^T A$.

Enfin, A et $A^T A$ sont injectifs en même temps, ce qui donne le résultat, puisque $A^T A$ est carrée donc inversible si elle est injective. ■

Sous l'hypothèse que A est de rang maximal, nous pouvons donner un résultat plus précis :

Proposition A.2. *Sous l'hypothèse que A est de rang n, la matrice des équations normales $A^T A$ est* définie positive.

Démonstration. Il est facile de voir que $A^T A$ est semi-définie positive (c'est en fait vrai indépendamment du rang de A) :

$$(A^T Ax,\, x) = (Ax,\, Ax) = \|x\|_2^2 \geq 0.$$

De plus, quand A est de rang n, nous avons vu (au lemme A.3) que la matrice (carrée) $A^T A$ est injective, donc inversible. Elle est donc définie positive. ■

A.4.2 Méthodes numériques

Équations normales et factorisation de Cholesky

Comme nous l'avons vu au paragraphe A.4.1 (équation (A.8)), la solution d'un problème de moindres carrés se ramène, en théorie du moins, à la résolution d'un système linéaire pour la matrice $A^T A$ (dite matrice des équation normales) :

$$A^T Ax = A^T z. \tag{A.10}$$

Nous avons vu également (proposition A.2) que, sous l'hypothèse que A est rang n, la matrice $A^T A$ est définie positive. Cette matrice est de taille n par n, et les équations normales représentent une « compression » d'information, puisque $n \leq m$. Le système (A.10) peut donc (toujours en théorie) être résolu par la factorisation de Cholesky. Nous rappelons cette méthode bien connue :

Proposition A.3. *Soit C une matrice symétrique et définie positive. Il existe une unique matrice R triangulaire supérieure, à éléments diagonaux strictement positifs, telle que*

$$C = R^T R \tag{A.11}$$

Preuve. Elle se trouve dans les références citées ci-dessus. Notons que la preuve est constructive : elle fournit un algorithme pour calculer la matrice R à partir de C. ■

Il suffit ensuite de résoudre les deux systèmes linéaires

$$\text{(A.12)} \qquad \begin{cases} R^T y &= A^T z \\ Rx &= y \end{cases}$$

pour obtenir la solution. Le premier a une matrice triangulaire inférieure, le second une matrice triangulaire supérieure.

Le nombre d'opérations flottantes nécessaires à la formation de la matrice des équations normales est $n(n+1)m + mn$ (en tirant parti de la symétrie). Le nombre d'opérations de la factorisation de Choleski est $n^3/3$ opérations flottantes (additions et multiplications), plus n divisions et n extractions de racines carrées. La solution des équations triangulaires (A.12) prend n^2 opérations, et est donc négligeable.

Le coût de la méthode des équations normales est donc, en ne gardant que les termes principaux : $mn^2 + \frac{1}{3}n^3$.

En dépit de sa simplicité, et de son coût raisonnable (nous verrons que les autres méthodes de résolution des problèmes de moindres carrés sont plus chères), la méthodes des équations normales n'est pas recommandée, pour des raisons de stabilité. Elle cumule deux inconvénients :

i) Le simple fait de former la matrice $A^T A$ peut faire perdre de l'information sur les petits coefficients de la matrice A, ce qui peut avoir des conséquences désastreuses, comme le montre l'exemple A.2 ci-dessous.

ii) De plus, le conditionnement de la matrice $A^T A$ est le carré de celui de A (puisque les valeurs propres de $A^T A$ sont les carrés des valeurs singulières de A).

Exemple A.2 (d'après Björck [6])**.**
Soit le problème $\min_{x \in \mathbf{R}^3} \|Ax - z\|$, avec

$$A = \begin{pmatrix} 1 & 1 & 1 \\ \varepsilon & 0 & 0 \\ 0 & \varepsilon & 0 \\ 0 & 0 & \varepsilon \end{pmatrix}, \quad z = \begin{pmatrix} 1, & 0, & 0, & 0 \end{pmatrix}^T.$$

Cet exemple peut correspondre à un problème où la somme $x_1 + x_2 + x_3$ est connue avec beaucoup plus de précision de que les composants de x individuellement. La calcul exact donne

$$A = \begin{pmatrix} 1+\varepsilon^2 & 1 & 1 \\ 1 & 1+\varepsilon^2 & 1 \\ 1 & 1 & 1+\varepsilon^2 \end{pmatrix}, \quad A^T z = \begin{pmatrix} 1, & 1, & 1 \end{pmatrix}^T, \quad x = \frac{1}{3+\varepsilon^2} \begin{pmatrix} 1, & 1, & 1 \end{pmatrix}^T$$

Supposons que $\varepsilon = 10^{-4}$ et que le calcul soit effectué avec 8 chiffres significatifs. Alors $1 + \varepsilon^2 = 1.000000001$ sera arrondi à 1, et la matrice $A^T A$ calculée sera singulière. Tout se passe comme si les trois dernières lignes de A étaient nulles, et n'avaient contribué aucune information.

Les difficultés illustrées par cet exemple peuvent être évitées, ou tout du moins les limites repoussées, en passant en double précision. Dans ces conditions, les équations normales deviennent une méthode acceptable. Toutefois, la méthode présentée au paragraphe suivant lui est préférable comme méthode de résolution générale.

La factorisation QR

La méthode « moderne » pour résoudre les problèmes de moindres carrés est basée sur une factorisation dite QR de la matrice A, où Q est une matrice orthogonale, et R est triangulaire supérieure. La réduction de A à une forme triangulaire s'effectue par une suite de multiplications par des matrices orthogonales élémentaires, qui peuvent être soit des symétries (méthode de Householder), soit des rotations (méthode Givens). Nous ne détaillerons pas cette étape, renvoyant encore une fois aux références citées plus haut, et nous nous contenterons de donner le résultat de la factorisation, et de montrer comment elle est utilisée pour résoudre le problème de moindres carrés.

Théorème A.4. *Soit $A \in \mathbf{R}^{m\times n}$ une matrice de rang n. Il existe une matrice orthogonale $Q \in \mathbf{R}^{m\times m}$ et une matrice triangulaire supérieure inversible telles que*

$$A = Q \begin{pmatrix} R \\ 0 \end{pmatrix} \tag{A.13}$$

On peut montrer (voir [23]) que le nombre d'opérations flottantes est (au premier ordre) $2mn^2 - \frac{2}{3}n^3$. Le calcul de la factorisation QR est donc plus cher que la formation, et la factorisation, de la matrice des équations normales. Le sur-coût n'est toutefois que d'un facteur 2 (l'ordre de complexité est le même). De plus, (voir les références citées au début de ce chapitre), les propriétés numériques de la méthode QR sont supérieures (dues au fait que l'on travaille avec des matrices orthogonales), et en font la méthode de choix pour une implémentation robuste. En pratique, les bibliothèques numériques fournissent toutes une version de la résolution de problèmes de moindres carrés basées sur la factorisation QR. Dans les programmes interactifs comme Scilab où Matlab, la commande `x= A\z` résolvent le problème de moindres carrés quand A est rectangulaire, et sont basées sur la factorisation QR de A.

La décomposition QR possède une importante interprétation en terme d'orthogonalisation. Pour préciser cela, partitionnons la matrice Q en $Q = (Q_1\, Q_2)$, avec $Q_1 \in \mathbf{R}^{m\times n}$, $Q_1 \in \mathbf{R}^{(m-n)\times n}$. Les colonnes de Q_1 forment une base orthogonale de l'image de A, celle de Q_2 une base orthonormale de $\operatorname{Im} A^{\perp}$, et l'on a la factorisation réduite

$$A = Q_1 R. \tag{A.14}$$

Cette factorisation montre que, pour $k \leq n$, les k premières colonnes de Q forment une base orthonormale du sous-espace engendré par les k premières colonnes de A. De plus, comme R est inversible, ces sous–espace sont égaux pour chaque k.

La factorisation QR construit donc une base orthonormale pour *tous* les sous espaces engendrés par les colonnes de A. On trouve le résultat obtenu habituellement par l'orthogonalisation de Gram–Schmidt. Il y a toutefois deux différences notables :

— l'algorithme de Gram–Schmidt est notoirement *instable* : les vecteurs calculés par cette méthode ne sont pas numériquement orthogonaux. L'algorithme QR calcule le même résultat de façon numériquement stable (voir ci-dessous). Il est possible d'utiliser une variante (l'algorithme de Gram–Schmidt modifié), qui possède de meilleures propriétés de stabilité, mais celles de l'algorithme QR sont encore meilleures.
— L'algorithme de Gram–Schmidt ne calcule que la matrice que nous avons notée Q_1. Comme nous l'avons vu, l'algorithme QR fournit en plus la matrice Q_2. Selon les applications, cette différence peut ou on être importante.

Remarque A.2. La relation (A.14) montre que R n'est autre que le facteur de Cholesky de A^TA. En effet, $A^TA = R^TQ_1^TQ_1R = R^TR$, puisque Q_1 est orthogonale. Cette remarque montre alors que R est déterminée de façon unique par A (si A est de rang n), et donc $Q_1 = AR^{-1}$ l'est également. Par contre, Q_2 n'est déterminée que par la condition que $(Q_1\ Q_2)$ soit orthogonale, et n'est donc pas unique.

Une fois la factorisation QR obtenue, il est facile de résoudre le problème de moindres carrés (A.7).

Théorème A.5. *Soir* $A = Q\begin{pmatrix} R \\ 0 \end{pmatrix}$ *la décomposition* QR *de la matrice* A. *La solution du problème de moindres carrés est donnée par la résolution du système linéaire*

$$Rx = z_1, \tag{A.15}$$

avec $Q^Tz = \begin{pmatrix} z_1 \\ z_2 \end{pmatrix}$.

Preuve. Du fait que la matrice Q est orthogonale, nous avons

$$\|Ax - z\|_2^2 = \left\| Q\begin{pmatrix} R \\ 0 \end{pmatrix} x - z \right\|_2^2 = \left\| \begin{pmatrix} R \\ 0 \end{pmatrix} x - Q^Tz \right\|_2^2$$

Posons $Q^Tz = \begin{pmatrix} z_1 \\ z_2 \end{pmatrix}$ comme indiqué dans l'énoncé du théorème, il vient

$$\|Ax - z\|_2^2 = \|Rx - z_1\|_2^2 + \|z_2\|_2^2 .$$

Le deuxième terme est constant (indépendant de x), et le premier est minimisé en prenant $x = R^{-1}z_1$, ce qui est possible puisque R est inversible. Pour conclure, notons que le terme $\|z_e\|_2^2$ représente la valeur du résidu à la solution. ■

A.5 Méthodes itératives classiques

Bibliographie

[1] Grégoire Allaire and Mahmoud Sidi Kaber. *Algèbre linéaire numérique : Cours et exercices.* Ellipses, 2002.

[2] O. Axelsson and V. A. Barker. *Finite element solution of boundary value problems*, volume 35 of *Classics in Applied Mathematics.* Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2001. Theory and computation, Reprint of the 1984 original.

[3] Owe Axelsson. *Iterative solution methods.* Cambridge University Press, Cambridge, 1994.

[4] Richard Barrett, Michael Berry, Tony F. Chan, and et al. *Templates for the solution of linear systems : building blocks for iterative methods.* Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1994.

[5] Michele Benzi. Preconditioning techniques for large linear systems : a survey. *J. Comput. Phys.*, 182(2) :418–477, 2002.

[6] Åke Björck. *Numerical methods for least squares problems.* Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1996.

[7] Ke Chen. *Matrix preconditioning techniques and applications*, volume 19 of *Cambridge Monographs on Applied and Computational Mathematics.* Cambridge University Press, Cambridge, 2005.

[8] Philippe G. Ciarlet. *Introduction à l'analyse numérique matricielle et à l'optimisation.* Collection Mathématiques Appliquées pour la Maîtrise. Masson, Paris, 1982.

[9] T. Davis and Y. Hu. The University of Florida Sparse Matrix Collection. *ACM Transactions on Mathematical Software*, 38(1) :1–25, 2011.

[10] Gene H. Golub and Charles F. Van Loan. *Matrix computations.* Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, Baltimore, MD, 4ème edition, 2013.

[11] Xavier Gourdon. *Algèbre.* Ellipses, 2009.

[12] Anne Greenbaum. *Iterative methods for solving linear systems*, volume 17 of *Frontiers in Applied Mathematics.* Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1997.

[13] Anne Greenbaum, Vlastimil Pták, and Zdeněk Strakoš. Any nonincreasing convergence curve is possible for GMRES. *SIAM J. Matrix Anal. Appl.*, 17(3) :465–469, 1996.

[14] C. T. Kelley. *Iterative methods for linear and nonlinear equations*, volume 16 of *Frontiers in Applied Mathematics.* Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1995.

[15] P. Lascaux and R. Théodor. *Analyse numérique matricielle appliquée à l'art de l'ingénieur. Tome 1.* Masson, Paris, 2 edition, 1993. Méthodes directes.

[16] P. Lascaux and R. Théodor. *Analyse numérique matricielle appliquée à l'art de l'ingénieur. Tome 2.* Masson, Paris, 2 edition, 1994. Méthodes itératives.

[17] Brigitte Lucquin. *Équations aux dérivées partielles et leurs approximations.* Mathématiques à l'université. Ellipses, 2004.

[18] Roger Mansuy and Rached Mneimné. *Algèbre linéaire. Rédution des endomorphismes.* Vuibert, 2012.

[19] Gérard Meurant. *Computer solution of large linear systems*, volume 28 of *Studies in mathematics and its applicaitons.* Elsevier North-Holland, 2 edition, 1999.

[20] Youcef Saad and Martin H. Schultz. GMRES : a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 7(3) :856–869, 1986.

[21] Yousef Saad. *Iterative methods for sparse linear systems.* Society for Industrial and Applied Mathematics, Philadelphia, PA, 2 edition, 2003.

[22] Denis Serre. *Matrices*, volume 216 of *Graduate Texts in Mathematics.* Springer, New York, second edition, 2010. Theory and applications.

[23] Lloyd N. Trefethen and David Bau, III. *Numerical linear algebra.* Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1997.

[24] Henk A. van der Vorst. *Iterative Krylov methods for large linear systems*, volume 13 of *Cambridge Monographs on Applied and Computational Mathematics.* Cambridge University Press, Cambridge, 2009. Reprint of the 2003 original.

[25] A. J. Wathen. Preconditioning. *Acta Numer.*, 24 :329–376, 2015.