ING1 – Génie Informatique

Introduction à l'analyse numérique – Notes de cours

Romain Dujol, Laurence Lamoulie, Chrysostome Baskiotis

2014 – 2015

Table des matières

Chapitre 1

Représentation des nombres en machines : le standard IEEE-754

La simulation numérique est largement utilisé dans l'industrie dans l'objectif de réduire les expérimentations réelles ainsi que les coûts finaux. Elle utilise les modèles physiques et mathématiques qui sont implémentés dans un calculateur.

Avant même de considérer les calculs effectués sur un ordinateur, il nous faut considérer comment les objets de ces calculs, les nombres, sont stockés. En effet, comment représenter une infinité de nombres réels dans un ordinateur qui, par construction, ne dispose que d'une capacité finie de stockage ? Dans la plupart des cas, on ne peut donc représenter qu'une approximation d'un nombre. Et tous les calculs qui suivront sont également approchés.

1.1 Représentations d'un nombre réel

1.1.1 Représentations d'un entier naturel

Théorème 1.1 (Numération d'un entier). Soit β un entier naturel supérieur ou égal à deux.

Pour tout entier naturel n, il existe un unique entier naturel non nul p et un unique p-uplet $(b_0,\dots,b_{p-1})$ d'entiers de $[\![0,\beta-1]\!]$ tels que $n=\sum_{k=0}^{p-1} b_k\beta^k$.

La donnée de p et de $(b_0,\dots,b_{p-1})$ est appelée **représentation de n en base β** et on écrira :

$$n=(b_{p-1}\cdots b_1 b_0)_\beta$$

L'entier b_{p-1} est appelé **chiffre de poids fort de n**.
L'entier b_0 est appelé **chiffre de poids faible de n**.

Exemple (Bases usuelles).

- $\beta = 2$: représentation binaire
- $\beta = 7$: pour les semaines
- $\beta = 8$: représentation octale
- $\beta = 10$: représentation décimale
- $\beta = 12$: pour les mois
- $\beta = 16$: représentation héxadécimale
- $\beta = 24$: pour les heures
- $\beta = 60$: pour les secondes et les minutes

fonction int2base(β : entier, n : entier) : Tableau de entier
préconditions : $\beta \geq 2$ et $n > 0$
  $k \leftarrow 0$
  $m \leftarrow n$
  répéter
    $b_k \leftarrow m \mod \beta$
    $m \leftarrow m \operatorname{div} \beta$ /* *Division entière* */
    $k \leftarrow k + 1$
  jusqu'à $m = 0$
  retourner $(b_{k-1}, \ldots, b_1, b_0)$
fin fonction

Algorithme 1: Calcul des chiffres de la représentation d'un entier naturel n en base β

Exemple (Représentation de 13 *en bases* 2, 4, 8 *et* 16*).*

$13 = 2 \times 6 + \mathbf{1}$
$6 = 2 \times 3 + \mathbf{0}$
$3 = 2 \times 1 + \mathbf{1}$
$1 = 2 \times 0 + \mathbf{1}$
Donc $13 = (1101)_2$

$13 = 4 \times 3 + \mathbf{1}$
$3 = 4 \times 0 + \mathbf{3}$
Donc $13 = (31)_4$

$13 = 8 \times 1 + \mathbf{5}$
$5 = 8 \times 0 + \mathbf{1}$
Donc $13 = (15)_8$

$13 = 16 \times 0 + \mathbf{13}$
Donc $13 = (D)_{16}$

Remarque. Dans le cas des représentations en base 2, 4, 8, 16, ..., on peut remarquer le comportement par bloc des représentations :

$(11|01)_2$
$(\ 3|\ 1)_4$
$(\ \ D\ \ )_{16}$

$(001|101)_2$
$(\ \ 1|\ \ 5)_8$

De manière générale, pour passer d'une représentation en base b à une représentation en base b^p, on évalue par bloc de p chiffres en partant du chiffre de poids faible.

1.1.2 Représentations d'un nombre réel positif

Définition 1.1 (Partie entière. Partie fractionnaire). Soit x un nombre réel.
On appelle **partie entière de x**, notée $\lfloor x \rfloor$, le plus grand entier relatif inférieur ou égal à x :

$$\lfloor x \rfloor = \max\{n \in \mathbb{Z},\ n \leq x\}$$

On appelle **partie fractionnaire de x**, notée $[x]$, le nombre réel défini par $[x] = x - \lfloor x \rfloor$.

Remarque. Pour tout réel positif x, $[x] \in [0,1[$.

ATTENTION. $\lfloor 2{,}5 \rfloor = 2$, mais $\lfloor -2{,}5 \rfloor = -3$

Pour représenter un nombre réel positif x dans une base b :
- on calcule la représentation de la partie entière $\lfloor x \rfloor$ de x (voir sous-section précédente) ;
- on calcule la représentation de la partie fractionnaire $[x]$ de x ;
- on combine les deux représentations.

Théorème 1.2 (Numération d'un réel de $[0,1[$).
Soit β un entier naturel supérieur ou égal à deux.
Pour tout nombre réel x de $[0,1[$, il existe une unique suite $(b_k)_{k\geq 1} \in [\![0, \beta-1]\!]^{\mathbb{N}}$ tels que

$$x = \sum_{k=1}^{+\infty} b_k \beta^{-k}$$

La donnée de $(b_k)_{k\geq 1}$ est appelée **représentation de x en base β** et on écrira :

$$x = (0, b_1 b_2 \cdots)_\beta$$

IMPORTANT. Cela suggère donc qu'une représentation peut être infinie.

ATTENTION. Si $b_k = \beta - 1$ pour tout entier naturel non k, alors $(0, b_1 b_2 \cdots)_b = 1$.

Démonstration. $(0, b_1 b_2 \cdots)_\beta = \sum_{k=1}^{+\infty} (\beta-1)\beta^{-k} = (\beta-1)\sum_{k=0}^{+\infty} \beta^{-k-1} = \frac{\beta-1}{\beta} \sum_{k=0}^{+\infty} (\beta^{-1})^k = \frac{\beta-1}{\beta} \frac{1}{1-\beta^{-1}} = 1.$ □

fonction frac2base(β : entier, x : réel) : Tableau de entier
préconditions : $\beta \geq 2$ et $x \in [0,1[$
  $k \leftarrow 1$
  $y \leftarrow x$
  répéter
    $b_k \leftarrow \lfloor y \times \beta \rfloor$
    $y \leftarrow [y \times \beta]$
    $k \leftarrow k+1$
  jusqu'à $y = 0$
  retourner $(b_1, b_2, \ldots)$
fin fonction

Algorithme 2: Calcul des chiffres de la représentation d'un réel de $[0,1[$ en base b

ATTENTION. L'algorithme présenté ici peut ne jamais terminer. Il faut ajouter un contrôle sur le nombre maximal de chiffres permis.

Exemple (Représentation de 0,1875 *en base* 2, 4, 8 *et* 16*).*

$0{,}1875 \times 2 = \mathbf{0}{,}375$
$0{,}375 \times 2 = \mathbf{0}{,}75$
$0{,}75 \times 2 = \mathbf{1}{,}5$
$0{,}5 \times 2 = \mathbf{1}{,}0$
$0{,}1875 = (0{,}0011)_2$

$0{,}1875 \times 4 = \mathbf{0}{,}75$
$0{,}75 \times 4 = \mathbf{3}{,}0$
$0{,}1875 = (0{,}03)_4$

$0{,}1875 \times 8 = \mathbf{1}{,}5$
$0{,}5 \times 8 = \mathbf{4}{,}0$
$0{,}1875 = (0{,}14)_8$

$0{,}1875 \times 16 = \mathbf{3}{,}0$
$0{,}1875 = (0{,}3)_8$

Remarque. Le comportement par blocs observé dans le cas des entiers naturels est également observable ici.

Exemple (Représentation de 0,2 *en base* 2, 4, 8 *et* 16*).*

$0{,}2 \times 2 = \mathbf{0}{,}4$
$0{,}4 \times 2 = \mathbf{0}{,}8$
$0{,}8 \times 2 = \mathbf{1}{,}6$
$0{,}6 \times 2 = \mathbf{1}{,}2$
$0{,}2 \times 2 = \ldots$
$0{,}2 = (0{,}00110011\cdots)_2$

$0{,}2 \times 4 = \mathbf{0}{,}8$
$0{,}8 \times 4 = \mathbf{3}{,}2$
$0{,}2 \times 4 = \ldots$
$0{,}2 = (0{,}0303\cdots)_4$

$0{,}2 \times 8 = \mathbf{1}{,}6$
$0{,}6 \times 8 = \mathbf{4}{,}8$
$0{,}8 \times 8 = \mathbf{6}{,}4$
$0{,}4 \times 8 = \mathbf{3}{,}2$
$0{,}2 \times 8 = \ldots$
$0{,}2 = (0{,}14631463\cdots)_8$

$0{,}2 \times 16 = \mathbf{3}{,}2$
$0{,}2 \times 16 = \ldots$
$0{,}2 = (0{,}33\cdots)_8$

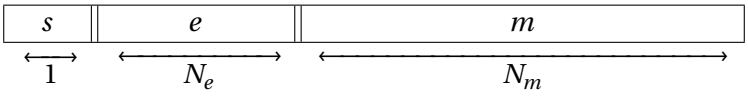
Exemple (Représentation de 13,1875 *en bases* 2, 4, 8 *et* 16*).*
$13{,}1875 = (1101{,}0011)_2 = (31{,}03)_4 = (15{,}14)_8 = (D{,}3)_{16}$

Exemple (Représentation de 13,2 *en bases* 2, 4, 8 *et* 16*).*
$13{,}2 = (1101{,}00110011\cdots)_2 = (31{,}0303\cdots)_4 = (15{,}14631463\cdots)_8 = (D{,}33\cdots)_{16}$

1.2 Description de la norme `IEEE-754`

La norme `IEEE-754` représente un nombre sur N bits divisés en trois parties :
- le signe (sur un bit) ;
- l'exposant (sur N_e bits) : celui n'est pas directement stocké, mais stocké « par excès » ;
- la mantisse (sur N_m bits) : seule la partie après la virgule est stockée, le 1 avant la virgule est implicite.

Donc $N = 1 + N_e + N_m$.



1.2.1 Classification des représentations

On distingue trois types de nombres : les nombres normalisés, les nombres sous-normalisés et les valeurs spéciales.

Définition 1.2 (Nombre normalisé). Un nombre `IEEE-754` est dit **normalisé** si et seulement si le codage binaire de son exposant e a au moins deux bits différents.

Auquel cas, le nombre représenté est $\pm 1, m \cdot 2^{E_{\min}+e-1}$, i.e. $(-1)^s \cdot (1 + m \cdot 2^{-N_m}) \cdot 2^{E_{\min}+e-1}$.

Remarque. Les valeurs $e = 0 = (0 \cdots 0)_2$ and $e = 2^{N_e} - 1 = (1 \cdots 1)_2$ sont donc réservées et l'intervalle des exposants autorisés est

$$[\![E_{\min}, E_{\max}]\!] = [\![-(2^{N_e-1} - 2), 2^{N_e-1} - 1]\!]$$

Proposition 1.1 (Bornes des nombres normalisés).
On suppose que l'exposant et la mantisse sont respectivement codés sur N_e et N_m bits.

Le plus petit nombre positif normalisé $v_{\min}$ a pour codage :

0	00 01	00 00

et vaut donc $v_{\min} = 2^{E_{\min}}$.

Le plus grand nombre positif normalisé $v_{\max}$ a pour codage :

0	11 10	11 11

et vaut donc $v_{\max} = (2 - 2^{-N_m}) \cdot 2^{E_{\max}}$.

Définition 1.3 (Nombre sous-normalisé). Un nombre `IEEE-754` est dit **sous-normalisé** si et seulement si :
- le codage binaire de son exposant e est nul ;
- le codage binaire de sa mantisse m est non nul.

Auquel cas, le nombre représenté est $(-1)^s \cdot (0 + m \cdot 2^{-N_m}) \cdot 2^{E_{\min}}$.

Proposition 1.2 (Bornes des nombres sous-normalisés).
On suppose que l'exposant et la mantisse sont respectivement codés sur N_e et N_m bits.

Le plus petit nombre sous-normalisé a pour codage :

0	00 00	00 01

et vaut donc $2^{E_{\min}-N_m}$.

Le plus grand nombre sous-normalisé a pour codage :

0	00 00	11 11

et vaut donc $(1-2^{-N_m})\cdot 2^{E_{\min}}$.

Définition 1.4 (Valeurs spéciales). En fonction du codage binaire, on dispose des valeurs spéciales suivantes :

- | 0 | 00 ..... 00 | 00 ..... 00 | : « zéro positif » ;
  | 1 | 00 ..... 00 | 00 ..... 00 | : « zéro négatif » ;
- | 0 | 11 ..... 11 | 00 ..... 00 | : « infini positif » ;
  | 1 | 11 ..... 11 | 00 ..... 00 | : « infini négatif » ;
- | 0/1 | 11 ..... 11 | $\neq 0$ | : « `NaN` » (*Not a Number*).

1.2.2 Précision d'un format `IEEE-754`

Définition 1.5 (Précision d'un format). On appelle **précision** d'un format `IEEE-754` le plus petit nombre réel positif ε tel que les représentations de 1 et $1+\varepsilon$ soient distinctes.

Remarque. Cette précision est souvent appelée « ε-machine ».

IMPORTANT. Si deux nombres ont une erreur relative strictement inférieure à cette précision, alors *leurs représentations normalisées sont identiques.*

Proposition 1.3.
On suppose que l'exposant et la mantisse sont respectivement codés sur N_e et N_m bits.

Alors la précision est égale à 2^{-N_m}.

Démonstration. On a $1=(-1)^0\cdot(1+0\cdot 2^{-N_m})\cdot 2^0$. La représentation de 1 est donc

0	01 11	00 00

On en déduit que le plus petit strictement supérieur à 1 dont la représentation est différente est :

0	01 11	00 01

c'est-à-dire $(-1)^0(1+1\cdot 2^{-N_m})\cdot 2^0 = 1+2^{-N_m}$. □

Format	N	N_e	N_m	$E_{\min}$	$E_{\max}$	$\varepsilon_{\text{mach}}$	*Nombre normalisé*		*Nombre sous-normalisé*	
							$v_{\min}$	$v_{\max}$	*min*	*max*
Simple	32	8	23	-126	127	$1.2\cdot10^{-7}$	$1.2\cdot10^{-38}$	$3.4\cdot10^{+38}$	$1.2\cdot10^{-45}$	$1.2\cdot10^{-38}$
Double	64	11	52	-1022	1023	$2.2\cdot10^{-16}$	$2.2\cdot10^{-308}$	$1.8\cdot10^{+308}$	$4.9\cdot10^{-324}$	$2.2\cdot10^{-308}$
Quadruple	128	15	112	-16382	16383	$1.9\cdot10^{-34}$	$3.4\cdot10^{-4932}$	$6.0\cdot10^{+4931}$	$6.5\cdot10^{-4966}$	$3.4\cdot10^{-4932}$

TABLE 1.1 – Trois formats `IEEE-754` : simple, double, quadruple

1.2.3 Codage

Proposition 1.4 (Condition suffisante de normalisation).
Tout nombre de l'ensemble $[-v_{\max},-v_{\min}]\cup[v_{\min},v_{\max}]$ est approchable par une représentation `IEEE-754` normalisée.

Algorithme 2.1 (Calcul de l'approximation normalisée d'un nombre réel x).
Prérequis : $|x|\in[v_{\min},v_{\max}]$

1. On calcule la représentation binaire de $|x|$:
   — l'exposant binaire e_{bin} sera le nombre de décalages nécessaires pour amener la virgule juste après le premier bit à 1 : ce nombre est compté positivement pour un décalage vers la gauche et négativement pour un décalage vers la droite ;
   — la mantisse binaire m_{bin} sera le résultat de ce décalage.
2. L'exposant stocké est la représentation binaire de $e=e_{\text{bin}}+1-E_{\min}$.
3. La mantisse stockée est les N_m premiers bits de la représentation binaire de $m=m_{\text{bin}}-1$.

Exemple (Représenter 13,1875 *en simple précision).* La représentation binaire de 13,1875 est $(1101{,}0011)_2$. D'où $e_{\text{bin}}=3$ et $m_{\text{bin}}=(1{,}1010011)_2$.

L'exposant stocké est alors $e=3+1-(-126)=130$ dont la représentation binaire est $(10000010)_2$ et on conclut que la représentation de 13,1875 au format `IEEE-754` simple précision est :

0	10000010	10100110000000000000000

Exemple (Représenter 13,2 *en simple précision).* La représentation binaire de 13,2 est

$$(1101{,}001100110110011\cdots)_2$$

D'où $e_{\text{bin}}=3$ et $m_{\text{bin}}=(1{,}10100110011001100110011\cdots)_2$.

L'exposant stocké est alors $e=3+1-(-126)=130$ dont la représentation binaire est $(10000010)_2$ et on conclut qu'une représentation *approchée* (puisqu'une troncature à vingt-trois bits a été nécessaire) de 13,2 au format `IEEE-754` simple précision est :

0	10000010	10100110011001100110011

- valeur exacte de la représentation ci-avant : 13,19999980926513671875
- erreur absolue commise : $\frac{2^{-20}}{5}$, soit environ $1{,}9 \cdot 10^{-7}$
- erreur relative commise : $\frac{2^{-20}}{66}$, soit environ $1{,}4 \cdot 10^{-8}$ (ce qui est bien plus petit que la précision du format)

Modes d'arrondi Tout comme pour les nombres décimaux, il existe plusieurs façons d'arrondir un nombre binaire et de l'appliquer au calcul de sa représentation `IEEE-754`.

Définition 1.6 (Modes d'arrondi).
On suppose que l'exposant et la mantisse sont respectivement codés sur N_e et N_m bits.
Soit x un nombre écrit sous la forme $x = (-1)^s \cdot (1 + m_{\text{bin}} \cdot 2^{-N_m}) \cdot 2^{e_{\text{bin}}}$ avec

$$m_{\text{bin}} = b_0 b_1 \cdots b_{N_m-1} b_{N_m} \cdots \quad \in [0, 2^{N_m}[$$

On définit les modes d'arrondi suivants.

Vers zéro La mantisse stockée est $m = b_0 b_1 \cdots b_{N_m-1}$.

Vers $-\infty$ La mantisse stockée est $m = b_0 b_1 \cdots b'_{N_m-1}$ avec $b'_{N_m-1} = \begin{cases} b_{N_m-1} & \text{si } s = 0 \\ b_{N_m-1} + 1 & \text{si } s = 1 \end{cases}$.

Vers $+\infty$ La mantisse stockée est $m = b_0 b_1 \cdots b'_{N_m-1}$ avec $b'_{N_m-1} = \begin{cases} b_{N_m-1} + 1 & \text{si } s = 0 \\ b_{N_m-1} & \text{si } s = 1 \end{cases}$.

Au plus près La mantisse stockée est $m = b_0 b_1 \cdots b'_{N_m-1}$ avec $b'_{N_m-1} = b_{N_m-1} + b_{N_m}$.
Si $b'_{N_m-1} = 2$, la retenue est propagée et peut engendrer, si nécessaire, une modification de l'exposant.

Remarque. La norme `IEEE-754` recommande le mode d'arrondi au plus près.

Exemple (Modes d'arrondi de 13,2*).*

· vers zéro :	0	10000010	10100110011001100110011
· vers $-\infty$:	0	10000010	10100110011001100110011
· vers $+\infty$:	0	10000010	10100110011001100110100
· au plus près :	0	10000010	10100110011001100110011

Définition 1.7 (Erreurs de représentation). Soit x un nombre réel.
On appelle **nombre machine** de x, noté $\hat{x}$, la représentation machine de x.
On appelle **erreur (absolue) de représentation de x**, notée Δx, le nombre réel défini par $\Delta x = \hat{x} - x$.
On appelle **erreur relative de représentation de x**, notée $\iota(x)$, le nombre réel défini par $\iota(x) = \dfrac{\Delta x}{\hat{x}}$.
On appelle **erreur relative de précision de x**, notée $\eta(x)$, le nombre réel défini par $\eta(x) = \dfrac{\Delta x}{x}$.

Remarque. On a évidemment $\widehat{\hat{x}} = \hat{x}$.

Proposition 1.5 (Évaluation de l'erreur de représentation en `IEEE-754`).
On suppose que l'exposant et la mantisse sont respectivement codés sur N_e et N_m bits.
Alors pour tout nombre réel x admettent une représentation normalisée :

$$\frac{\Delta x}{|x|} \leq \begin{cases} 2^{-N_m} & \text{si le mode d'arrondi est au plus près} \\ 2 \cdot 2^{-N_m} & \text{sinon} \end{cases}$$

IMPORTANT. Donc l'erreur de représentation de tout nombre dans un mode d'arrondi au plus près est majorée par la précision du système de représentation.

Exercices

Exercice 1.1. *Convertir les nombres suivants en héxadécimal.*

1. 15,275
2. 358,937
3. 387,62
4. 5233,618

Exercice 1.2.

1. *Calculer la représentation de* 5,75 *au format* `IEEE-754` *simple précision, puis double précision.*
2. *Même question pour* 0,1.

Chapitre 2

Perturbations de systèmes linéaires

2.1 Définitions

Pour étudier des perturbations d'un système linéaire, nous devons disposer d'une mesure numérique de ces perturbations qui portent sur deux types d'objets : des vecteurs et des matrices.

2.1.1 Norme vectorielle

2.1.1.1 Définitions

Définition 2.1 (Norme). Soit E un $\mathbb{R}$-espace vectoriel.
Une **norme sur E** est une application $\|\cdot\|$ de E dans $\mathbb{R}$ qui vérifie les trois propriétés suivantes :

1. $\forall x \in E, (\|x\| = 0 \iff x = 0_E)$
2. $\forall \lambda \in \mathbb{R}, \forall x \in E, \|\lambda x\| = |\lambda| \|x\|$
3. $\forall x \in E, \forall y \in E, \|x+y\| \leq \|x\| + \|y\|$ (« *inégalité triangulaire* »)

Si il existe une telle application, alors le couple $(E, \|\cdot\|)$ est un **espace vectoriel normé**.

Remarque. Donc $\|0_E\| = 0$ et $\|-x\| = |-1| \cdot \|x\| = \|x\|$.

Exemple. $E = \mathbb{R}^n$ est un espace vectoriel normé. Citons trois normes à connaître :

— $\forall x = (x_1, \ldots, x_n) \in \mathbb{R}^n, \|x\|_1 = \sum_{i=1}^{n} |x_i|$

— $\forall x = (x_1, \ldots, x_n) \in \mathbb{R}^n, \|x\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}$ (*norme « euclidienne »*)

— $\forall x = (x_1, \ldots, x_n) \in \mathbb{R}^n, \|x\|_\infty = \max_{1 \leq i \leq n} |x_i|$

2.1.1.2 Propriétés

Proposition 2.1. Une norme est toujours positive.

Démonstration. Soit $x \in E$, alors $0 = \|0_E\| = \|x + (-x)\| \leq \|x\| + \|-x\| = 2\|x\|$. Donc $\|x\| \geq 0$. □

Corollaire (Deuxième inégalité triangulaire). Soit $\|\cdot\|$ une norme sur E.
Alors pour tous vecteurs x et y de E, $\big|\|x\| - \|y\|\big| \leq \|x - y\|$.

Démonstration. On a $\|x\| = \|(x-y)+y\| \leq \|x-y\| + \|y\|$, donc $\|x\| - \|y\| \leq \|x-y\|$
et $\|y\| = \|(y-x)+x\| \leq \|y-x\| + \|x\|$, donc $\|y\| - \|x\| \leq \|y-x\| = \|x-y\|$

On peut alors conclure que $\big|\|x\| - \|y\|\big| = \max\{\|x\| - \|y\|, \|y\| - \|x\|\} \leq \|x-y\|$. □

Définition 2.2 (Normes équivalentes). Soit $\|\cdot\|_a$ et $\|\cdot\|_b$ deux normes sur E.
On dit que $\|\cdot\|_a$ et $\|\cdot\|_b$ sont **équivalentes** si et seulement si il existe deux réels strictement positifs α et β tels que

$$\forall x \in E,\ \alpha\|x\|_a \leq \|x\|_b \leq \beta\|x\|_a$$

Remarque. Les coefficients α et β doivent être indépendants de x.

Proposition 2.2. Cette relation définit une relation d'équivalence sur l'ensemble des normes sur E.

Exemple. Les trois normes $\|\cdot\|_1$, $\|\cdot\|_2$ et $\|\cdot\|_\infty$ sont équivalentes :

$$\forall x \in \mathbb{R}^n,\ \|x\|_\infty \leq \|x\|_2 \leq \|x\|_1 \leq n\,\|x\|_\infty$$

Théorème 2.1 (Équivalence des normes en dimension finie). Si E est un espace vectoriel de dimension finie, alors toutes les normes sur E sont équivalentes entre elles.

2.1.2 Norme matricielle

2.1.2.1 Norme d'algèbre

Définition 2.3 (Algèbre vectorielle). Soit $\mathbb{K}$ un corps commutatif, E un ensemble, $+$ et $\times$ deux lois de compositions internes et $\cdot$ une opération externe.
On dit que $(E, +, \cdot, \times)$ est une **$\mathbb{K}$-algèbre (vectorielle)** (ou **une algèbre (vectorielle) sur** $\mathbb{K}$) si les propriétés suivantes sont vérifiées :
— $(E, +, \cdot)$ est un $\mathbb{K}$-espace vectoriel.
— $(E, +, \times)$ est un anneau.
— $\forall \lambda \in \mathbb{K},\ \forall \mu \in \mathbb{K},\ \forall x \in E,\ \forall y \in E,\ (\lambda \cdot x) \times (\mu \cdot y) = (\lambda\mu) \cdot (x \times y)$

Exemple.

Pour tout $\mathbb{K}$-espace vectoriel E, $(\mathscr{L}(E),+,\cdot,\circ)$ est une $\mathbb{K}$-algèbre.

Pour tous entiers naturels non nuls n et p, $(\mathscr{M}_n(\mathbb{K}),+,\cdot,\times)$ est une $\mathbb{K}$-algèbre.

Définition 2.4 (Norme d'algèbre). Soit $(E,+,\cdot,\times)$ une $\mathbb{R}$-algèbre vectorielle.
Une **norme d'algèbre sur E** est une application $\|\cdot\|$ de E dans $\mathbb{R}$ telle que :

1. $\|\cdot\|$ est une norme sur E ;
2. $\|\cdot\|$ est *sous-multiplicative* : $\forall x\in E,\ \forall y\in E,\ \|x\times y\|\leq\|x\|\cdot\|y\|$.

Proposition 2.3. Soit $(E,+,\cdot,\times)$ une $\mathbb{R}$-algèbre vectorielle non réduite à l'élément nul et $\|\cdot\|$ une norme d'algèbre sur E. Alors $\|1_E\|\geq 1$ où 1_E désigne l'élément neutre de $\times$.

Démonstration. Soit $x\in E\backslash\{0_E\}$. Alors $\|x\|=\|x\times 1_E\|\leq\|x\|\cdot\|1_E\|$: comme $\|x\|$ est un réel strictement positif, on en déduit que $1\leq\|1_E\|$. □

2.1.2.2 Norme subordonée

Lemme. Soit $(E,\|\cdot\|_E)$ et $(F,\|\cdot\|_F)$ deux espaces vectoriels normés de dimension finie.
Toute application linéaire f de E dans F est bornée sur la sphère unité de E et y atteint ses bornes.
De plus $\sup\limits_{\substack{x\in E\\ \|x\|_E=1}}\|f(x)\|_F=\sup\limits_{x\in E\backslash\{0_E\}}\dfrac{\|f(x)\|_F}{\|x\|_E}$.

Démonstration. Soit f une application linéaire de E dans F. Comme E et F sont de dimension finie, f est continue. La sphère unité est un fermé borné de E, donc un compact. On en conclut que f est bornée sur la sphère unité et y atteint ses bornes.
Soit $x\in E\backslash\{0_E\}$. Alors $\dfrac{\|f(x)\|_F}{\|x\|_E}=\left\|\dfrac{f(x)}{\|x\|_E}\right\|_F=\left\|f\left(\dfrac{x}{\|x\|_E}\right)\right\|_F$. Comme $\left\|\dfrac{x}{\|x\|_E}\right\|_E=\dfrac{\|x\|_E}{\|x\|_E}=1$, on conclut. □

Définition 2.5 (Norme subordonnée).
Soit $(E,\|\cdot\|_E)$ et $(F,\|\cdot\|_F)$ deux espaces vectoriels normés de dimension finie.
On appelle **norme subordonée à $\|\cdot\|_E$ et $\|\cdot\|_F$**, notée $|||\cdot|||_{E,F}$, l'application

$$\begin{array}{rccc}|||\cdot|||_{E,F}: & \mathscr{L}(E,F) & \to & \mathbb{R}\\ & f & \mapsto & \sup\limits_{\substack{x\in E\\ \|x\|_E=1}}\|f(x)\|_F=\sup\limits_{x\in E\backslash\{0_E\}}\dfrac{\|f(x)\|_F}{\|x\|_E}\end{array}$$

Remarque. Si $E=F$ et $\|\cdot\|_F=\|\cdot\|_E$, alors $|||\mathrm{id}_E|||_{E,E}=\sup\limits_{\substack{x\in E\\ \|x\|_E=1}}\|x\|_E=1$.

Proposition 2.4 (Norme subordonnée).

Soit $(E,\|\cdot\|_E)$ et $(F,\|\cdot\|_F)$ deux espaces vectoriels normés de dimension finie.

Alors $|||\cdot|||_{E,F}$ est une norme sur $\mathscr{L}(E,F)$ et

$$\forall f\in\mathscr{L}(E,F),\ \|f(x)\|_F\leq |||f|||\cdot\|x\|_E$$

Démonstration.

1. $|||0_{\mathscr{L}(E,F)}|||_{E,F}=\sup\limits_{\substack{x\in E\\ \|x\|_E=1}}\|0_F\|_F=\sup\limits_{\substack{x\in E\\ \|x\|_E=1}}0=0$

   Soit $f\in\mathscr{L}(E,F)$ tel que $0=|||f|||_{E,F}=\sup\limits_{x\in E\setminus\{0_E\}}\dfrac{\|f(x)\|_F}{\|x\|_E}$.

   Alors pour tout vecteur x non nul de E, $\dfrac{\|f(x)\|_F}{\|x\|_E}\leq 0$: donc $\|f(x)\|_F=0$.

   Comme f est linéaire, $f(0_E)=0_F$: on en conclut que $f=0_{\mathscr{L}(E,F)}$.

2. Soit $f\in\mathscr{L}(E,F)$ et $\lambda\in\mathbb{R}$. Alors

$$|||\lambda f|||_{E,F}=\sup_{\substack{x\in E\\ \|x\|_E=1}}\|\lambda f(x)\|_F=\sup_{\substack{x\in E\\ \|x\|_E=1}}[|\lambda|\cdot\|f(x)\|_F]=\sup_{\substack{x\in E\\ \|x\|_E=1}}[|\lambda|\cdot\|f(x)\|_F]=|\lambda|\sup_{\substack{x\in E\\ \|x\|_E=1}}\|f(x)\|_F=|\lambda|\cdot|||f|||_{E,F}$$

3. Soit f et g deux applications linéaires de E dans F. On note $\mathbb{S}_E=\{x\in E,\ \|x\|_E=1\}$. Alors :

$$\forall x\in\mathbb{S}_E,\ \|f(x)+g(x)\|_F\leq\|f(x)\|_F+\|g(x)\|_F\leq|||f|||_{E,F}+|||g|||_{E,F}$$

   Donc $|||f+g|||_{E,F}=\sup\limits_{x\in\mathbb{S}_E}\|f(x)+g(x)\|_F\leq|||f|||_{E,F}+|||g|||_{E,F}$.

Comme f est linéaire, on a $f(0_E)=0_F$ et $\|f(0_E)\|_F=\|0_F\|_F=0\leq 0\leq|||f|||_{E,F}\cdot 0=|||f|||_{E,F}\cdot\|0_E\|_F$.

Soit $x\in E\setminus\{0_E\}$. Alors $\dfrac{\|f(x)\|_F}{\|x\|_E}\leq\sup\limits_{z\in E\setminus\{0_E\}}\dfrac{\|f(z)\|_F}{\|z\|_E}=|||f|||_{E,F}$. D'où le résultat demandé. □

Théorème 2.2 (Norme subordonnée sur l'espace des endomorphismes).

Soit $(E,\|\cdot\|_E)$ un espace vectoriel normé de dimension finie.

Alors la norme $|||\cdot|||_{E,E}$ subordonnée à $\|\cdot\|_E$ est une norme d'algèbre sur $\mathscr{L}(E)=\mathscr{L}(E,E)$.

Démonstration. On sait d'après la proposition 2.4 que $|||\cdot|||_{E,E}$ est une norme sur $\mathscr{L}(E)$.

Soit f et g deux endomorphismes de E et x un vecteur non nul de E. En utilisant l'inégalité établie dans la proposition 2.4, il vient que :

$$\|(g\circ f)(x)\|_E=\|g[f(x)]\|_E\leq|||g|||_{E,E}\cdot\|f(x)\|_E\leq|||g|||_{E,E}\cdot\big(|||f|||_{E,E}\cdot\|x\|_E\big)=\big(|||g|||_{E,E}\cdot|||f|||_{E,E}\big)\cdot\|x\|_E$$

Donc $\dfrac{\|(g\circ f)(x)\|_E}{\|x\|_E}\leq|||g|||_{E,E}\cdot|||f|||_{E,E}$ pour tout $x\in E\setminus\{0_E\}$, puis

$$|||g\circ f|||_{E,E}=\sup_{x\in E\setminus\{0_E\}}\frac{\|(g\circ f)(x)\|_E}{\|x\|_E}\leq|||g|||_{E,E}\cdot|||f|||_{E,E}$$

Donc $|||\cdot|||_{E,E}$ est bien une norme d'algèbre sur $\mathscr{L}(E)$. □

2.1.3 Cas $E = \mathbb{R}^n$ et $F = \mathbb{R}^m$

Alors on peut aisément identifier une application linéaire de $\mathbb{R}^n$ dans $\mathbb{R}^m$ avec sa matrice dans les bases canoniques de $\mathbb{R}^n$ et $\mathbb{R}^m$.

Définition 2.6 (Norme matricielle).
Soit n et m des entiers naturels non nul, $\|\cdot\|_n$ une norme sur $\mathbb{R}^n$ et $\|\cdot\|_m$ une norme sur $\mathbb{R}^m$.
Pour toute matrice $A \in \mathcal{M}_{nm}(\mathbb{R})$, on note $|||A|||_{n,m} = \sup\limits_{\substack{X\in\mathbb{R}^n \\ \|X\|_n=1}} \|AX\|_m = \sup\limits_{x\in\mathbb{R}^n\setminus\{0_{\mathbb{R}^n}\}} \dfrac{\|AX\|_m}{\|X\|_n}$
L'application $|||\cdot|||_{n,m} : \mathcal{M}_{mn}(\mathbb{R}) \to \mathbb{R}$ est appelée **norme matricielle subordonée à $\|\cdot\|_n$ et $\|\cdot\|_m$**.

Remarque. Si $m = n$ et $\|\cdot\|_m = \|\cdot\|_n = \|\cdot\|$, alors $|||I_n||| = \sup\limits_{\substack{X\in\mathbb{R}^n \\ \|X\|=1}} \|IX\| = \sup\limits_{\substack{X\in\mathbb{R}^n \\ \|X\|=1}} \|X\| = 1$.

Théorème 2.3.
Soit n et m des entiers naturels non nul, $\|\cdot\|_n$ une norme sur $\mathbb{R}^n$ et $\|\cdot\|_m$ une norme sur $\mathbb{R}^m$.
La norme matricielle subordonée à $\|\cdot\|_n$ et $\|\cdot\|_m$ est une norme d'algèbre sur $\mathcal{M}_{n,m}(\mathbb{R})$.

Démonstration. On note respectivement $\mathcal{B}$ et $\mathcal{B}'$ les bases canoniques de $\mathbb{R}^n$ et $\mathbb{R}^m$. On note également l'application $\Phi : \begin{array}{ccc} \mathcal{L}(\mathbb{R}^n,\mathbb{R}^m) & \to & \mathcal{M}_{mn}(\mathbb{R}) \\ f & \mapsto & \mathrm{mat}_{\mathcal{B}} f \end{array}$.
Il est bien connu que Φ est un isomorphisme d'algèbres. De plus $|||\Phi(\cdot)|||_{n,m}$ est une norme subordonée aux normes $\|\cdot\|_{\mathbb{R}^n} : \begin{array}{ccc} \mathbb{R}^n & \to & \mathbb{R} \\ x & \mapsto & \|\mathrm{mat}_{\mathcal{B}} x\| \end{array}$ et $\|\cdot\|_{\mathbb{R}^m} : \begin{array}{ccc} \mathbb{R}^m & \to & \mathbb{R} \\ x & \mapsto & \|\mathrm{mat}_{\mathcal{B}'} x\| \end{array}$, donc une norme d'algèbre sur $\mathcal{L}(\mathbb{R}^n,\mathbb{R}^m)$. On conclut par isomorphisme. □

Exemple (Normes matricielles subordonnées aux normes fondamentales).

— $\forall A \in \mathcal{M}_n(\mathbb{R}),\ |||A|||_1 = \max\limits_{1\le j\le n} \sum\limits_{i=1}^{n} |a_{ij}|$

— $\forall A \in \mathcal{M}_n(\mathbb{R}),\ |||A|||_2 = \sqrt{\max\limits_{\lambda\in \mathrm{Sp}\, A\,{}^tA} |\lambda|}$

— $\forall A \in \mathcal{M}_n(\mathbb{R}),\ |||A|||_\infty = \max\limits_{1\le i\le n} \sum\limits_{j=1}^{n} |a_{ij}|$

Définition 2.7 (Norme de FROBENIUS). Soit n un entier naturel non nul.
Pour toute matrice carrée d'ordre n, on définit la **norme de FROBENIUS de A**, notée $\|A\|_{\mathrm{F}}$, par

$$\|A\|_{\mathrm{F}} = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}^2}$$

Proposition 2.5. Soit n un entier naturel non nul.

La norme de FROBENIUS est une norme d'algèbre sur $\mathscr{M}_n(\mathbb{R})$, mais n'est pas une norme matricielle subordonnée.

Démonstration. Remarquons que la norme de FROBENIUS d'une matrice est la norme euclidienne du vecteur obtenu en « aplatissant » la matrice. Donc on établit aisément que $\|\cdot\|_{\mathrm{F}}$ est une norme sur $\mathscr{M}_n(\mathbb{R})$.

Lemme. Soit n un entier naturel non nul.

Soit $(\alpha_1,\dots,\alpha_n)$ et $(\beta_1,\dots,\beta_n)$ deux n-uplets de réels positifs. Alors $\displaystyle\sum_{k=1}^{n}\alpha_k\beta_k \leq \left(\sum_{k=1}^{n}\alpha_k\right)\cdot\left(\sum_{k=1}^{n}\beta_k\right)$.

Démonstration. Montrons le lemme par récurrence pour n entier naturel non nul.

— Si $n=1$, le résultat est évident.

— Si $n=2$, alors $(\alpha_1+\alpha_2)(\beta_1+\beta_2)=\alpha_1\beta_1+\alpha_1\beta_2+\alpha_2\beta_1+\alpha_2\beta_2 \geq \alpha_1\beta_1+\alpha_2\beta_2$.

— Si le lemme est vérifié au rang $n\geq 2$, soit alors $(\alpha_1,\dots,\alpha_{n+1})$ et $(\beta_1,\dots,\beta_{n+1})$ deux $(n+1)$-uplets de réels positifs. Alors en utilisant l'hypothèse de récurrence puis le lemme au rang 2, il vient que :

$$\sum_{k=1}^{n+1}\alpha_k\beta_k=\sum_{k=1}^{n}\alpha_k\beta_k+\alpha_{n+1}\beta_{n+1}\leq\left(\sum_{k=1}^{n}\alpha_k\right)\cdot\left(\sum_{k=1}^{n}\beta_k\right)+\alpha_{n+1}\beta_{n+1}\leq\left(\sum_{k=1}^{n}\alpha_k+\alpha_{n+1}\right)\cdot\left(\sum_{k=1}^{n}\beta_k+\beta_{n+1}\right)$$

Donc le lemme est vérifié au rang $n+1$. □

Soit $A=(a_{ij})_{\substack{1\leq i\leq n\\ 1\leq jn}}$ et $B=(b_{ij})_{\substack{1\leq i\leq n\\ 1\leq jn}}$ deux matrices carrées d'ordre n. Alors en utilisant le lemme précédent, il vient que :

$$\begin{aligned}\|AB\|_{\mathrm{F}}^2=\sum_{i=1}^{n}\sum_{j=1}^{n}(AB)_{ij}^2=\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}a_{ik}^2b_{kj}^2=\sum_{k=1}^{n}\sum_{i=1}^{n}\sum_{j=1}^{n}a_{ik}^2b_{kj}^2=\sum_{k=1}^{n}\left(\sum_{i=1}^{n}a_{ik}^2\right)\left(\sum_{j=1}^{n}b_{kj}^2\right)\\ \leq\left(\sum_{k=1}^{n}\sum_{i=1}^{n}a_{ik}^2\right)\cdot\left(\sum_{k=1}^{n}\sum_{j=1}^{n}b_{kj}^2\right)=\|A\|_{\mathrm{F}}^2\cdot\|B\|_{\mathrm{F}}^2\end{aligned}$$

Donc $\|\cdot\|_{\mathrm{F}}$ est sous-multiplicative.

Comme $\|I_n\|_{\mathrm{F}}=\sqrt{n}$, $\|\cdot\|_{\mathrm{F}}$ n'est pas une norme subordonnée. □

Remarque. Le fait que la norme de FROBENIUS ne soit pas une norme subordonnée justifie la non-utilisation de la notation $|||\cdot|||$.

Théorème 2.4. Soit n un entier naturel et $\|\cdot\|$ une norme sur $\mathscr{M}_n(\mathbb{R})$. Alors

$$\forall A\in\mathscr{M}_n(\mathbb{R}),\ \|A\|\geq\rho(A)$$

où $\rho(A)=\max_{\lambda\in\mathrm{Sp}\,A}|\lambda|$ est le **rayon spectral de A**.

Démonstration. ADMIS □

2.2 Analyse d'erreurs

2.2.1 Contexte et définitions

Définition 2.8 (Système linéaire). Soit n et m deux entiers naturels non nuls.

Un **système linéaire à m équations et n inconnues** est la donnée d'une matrice $A \in \mathscr{M}_{mn}(\mathbb{R})$ et d'un vecteur $b \in \mathbb{R}^m$.

Résoudre un système linéaire (A, b), c'est déterminer l'ensemble $\{x \in \mathbb{R}^n,\ Ax = b\}$.

Un système linéaire est dit **de CRAMER** si et seulement si A est carrée (i.e. $m = n$) et inversible.

Il s'agit donc d'étudier le comportement des solutions d'un système linéaire de CRAMER (A, b) de le cas de perturbations de A et/ou de b.

Théorème 2.5. Soit (A, b) un système linéaire de CRAMER.

Alors il existe un unique vecteur $x = (x_1, \dots, x_n) \in \mathbb{R}^n$ tel que $Ax = b$ avec :

$$\forall i \in [\![1, n]\!],\ x_i = \frac{\det(A_i)}{\det(A)}$$

où A_i est la matrice obtenue en remplaçant la $i^{\text{ème}}$ colonne de A par b.

Définition 2.9 (Conditionnement d'une matrice).

Soit n un entier naturel et $\|\cdot\|$ une norme d'algèbre sur $\mathscr{M}_n(\mathbb{R})$.

Pour toute matrice carrée d'ordre n inversible, on définit le **conditionnement (ou nombre-condition) de A**, noté $\kappa(A)$, par $\kappa(A) = \|A\| \cdot \|A^{-1}\|$.

Proposition 2.6. Soit n un entier naturel et $\|\cdot\|$ une norme d'algèbre sur $\mathscr{M}_n(\mathbb{R})$. Alors :

1. $\forall \lambda \in \mathbb{R}^*,\ \kappa(\lambda I_n) = 1$, si la norme considérée est une norme matricielle subordonnée
2. $\forall A \in \mathsf{GL}_n(\mathbb{R}),\ \kappa(A) \geq 1$
3. $\forall A \in \mathsf{GL}_n(\mathbb{R}),\ \forall k \in \mathbb{N}^*,\ \kappa(A^k) \leq \kappa(A)^k$

Théorème 2.6. Soit n un entier naturel et $\|\cdot\|$ une norme d'algèbre sur $\mathscr{M}_n(\mathbb{R})$. Alors

$$\forall A \in \mathsf{GL}_n(\mathbb{R}),\ \forall \Delta A \in \mathscr{M}_n(\mathbb{R}),\ \left(\frac{\|\Delta A\|}{\|A\|} < \frac{1}{\kappa(A)} \Longrightarrow A + \Delta A \in \mathsf{GL}_n(\mathbb{R}) \right)$$

Remarque. Donc la boule ouverte de centre A et de rayon $\dfrac{\|A\|}{\kappa(A)}$ est une zone de sécurité dans laquelle les matrices perturbées $A + \Delta A$ restent inversibles, comme A.
On rappelle que si $A + \Delta A$ devient singulière (non inversible), le système associé devient dégénéré.

2.2.2 Perturbations du second membre

Théorème 2.7 (Perturbations du second membre).
Soit (A, b) un système linéaire de CRAMER d'ordre n et x l'unique vecteur de $\mathbb{R}^n$ tel que $Ax = b$.
Soit Δb un vecteur quelconque de $\mathbb{R}^n$. On note Δx l'unique vecteur de $\mathbb{R}^n$ tel que $A(x+\Delta x) = b+\Delta b$.
Soit $\|\cdot\|$ une norme sur $\mathbb{R}^n$ et $|||\cdot|||$ sa norme matricielle subordonée. Alors

$$\frac{\|\Delta x\|}{\|x\|} \leq \kappa(A) \cdot \frac{\|\Delta b\|}{\|b\|}$$

avec $\kappa(A) = |||A||| \cdot |||A^{-1}|||$ le conditionnement de A pour la norme matricielle subordonnée.

Démonstration. On a $b+\Delta b = A(x+\Delta x) = Ax + A\Delta x = b + A\Delta x$: donc $\Delta b = A\Delta x$ et $\Delta x = A^{-1}\Delta b$.
On déduit que $\|\Delta x\| \cdot \|b\| = \|A^{-1}\Delta b\| \cdot \|Ax\| \leq (|||A^{-1}||| \cdot \|\Delta b\|) \cdot (|||A||| \cdot \|x\|) = \kappa(A) \cdot \|\Delta b\| \cdot \|x\|$, d'où le résultat. □

IMPORTANT. Donc un conditionnement $\kappa(A)$ faible assure qu'une petite perturbation sur b perturbe peu la solution x.

2.2.3 Perturbations de la matrice du système

Théorème 2.8 (Perturbations de la matrice du système).
Soit (A, b) un système linéaire de CRAMER d'ordre n et x l'unique vecteur de $\mathbb{R}^n$ tel que $Ax = b$.
Soit ΔA une matrice carrée d'ordre n telle que $A + \Delta A$ soit inversible. On note Δx l'unique vecteur de $\mathbb{R}^n$ tel que $(A+\Delta A) \cdot (x+\Delta x) = b$.
Soit $\|\cdot\|$ une norme sur $\mathbb{R}^n$ et $|||\cdot|||$ sa norme matricielle subordonée. Alors

$$\frac{\|\Delta x\|}{\|x\|} \leq \kappa(A) \cdot \frac{|||\Delta A|||}{|||A|||}$$

avec $\kappa(A) = |||A||| \cdot |||A^{-1}|||$ le conditionnement de A pour la norme matricielle subordonnée.

Démonstration. On a $b = (A+\Delta A)(x+\Delta x) = Ax + \Delta A \cdot x + A\Delta x + \Delta A \cdot \Delta x = b + \Delta A \cdot x + A\Delta x + \Delta A \cdot \Delta x$.
Donc $0 = \Delta A \cdot x + A\Delta x + \Delta A \cdot \Delta x$, puis $A\Delta x = -\Delta A \cdot x - \Delta A \cdot \Delta x \simeq -\Delta A \cdot x$, c'est-à-dire $\Delta x \simeq -A^{-1}\Delta A \cdot x$.
On en déduit que $|||A||| \cdot \|\Delta x\| \simeq |||A||| \cdot \|A^{-1}\Delta A \cdot x\| \leq |||A||| \cdot |||A^{-1}||| \cdot |||\Delta A||| \, \|x\| = \kappa(A) \, |||\Delta A||| \cdot \|x\|$, d'où le résultat. □

IMPORTANT. Donc un conditionnement $\kappa(A)$ faible assure qu'une petite perturbation sur A perturbe peu la solution x.

2.2.4 Perturbations des paramètres du système

Théorème 2.9 (Perturbations des paramètres du système).
Soit (A, b) un système linéaire de CRAMER d'ordre n et x l'unique vecteur de $\mathbb{R}^n$ tel que $Ax = b$.
Soit ΔA une matrice carrée d'ordre n telle que $A + \Delta A$ soit inversible et Δb un vecteur quelconque de $\mathbb{R}^n$. On note Δx l'unique vecteur de $\mathbb{R}^n$ tel que $(A + \Delta A) \cdot (x + \Delta x) = b + \Delta b$.
Soit $\|\cdot\|$ une norme sur $\mathbb{R}^n$ et $|||\cdot|||$ sa norme matricielle subordonée. Alors

$$\frac{\|\Delta x\|}{\|x\|} \leq \kappa(A) \cdot \left[\frac{|||\Delta A|||}{|||A|||} + \frac{\|\Delta b\|}{\|b\|} \right]$$

avec $\kappa(A) = |||A||| \cdot |||A^{-1}|||$ le conditionnement de A pour la norme matricielle subordonnée.

Démonstration. On a $b + \Delta b = (A + \Delta A)(x + \Delta x) = Ax + \Delta A \cdot x + A\Delta x + \Delta A \cdot \Delta x = b + \Delta A \cdot x + A\Delta x + \Delta A \cdot \Delta x$.
Donc $\Delta b = \Delta A \cdot x + A\Delta x + \Delta A \cdot \Delta x$, puis $A\Delta x = \Delta b - \Delta A \cdot x - \Delta A \cdot \Delta x \simeq \Delta b - \Delta A \cdot x$, c'est-à-dire $\Delta x \simeq A^{-1}(\Delta b - \Delta A \cdot x)$.
On en déduit que $\|b\| \cdot |||A||| \cdot \|\Delta x\| \simeq \|b\| \cdot |||A||| \cdot \|A^{-1}(\Delta b - \Delta A \cdot x)\|$

$$\begin{aligned} &\leq \|b\| \cdot |||A||| \cdot |||A^{-1}||| \cdot \|\Delta b - \Delta A \cdot x\| = \|b\| \kappa(A)(\|\Delta b\| + |||\Delta A||| \cdot \|x\|) \\ &= \kappa(A)(\|Ax\| \cdot \|\Delta b\| + \|b\| \cdot |||\Delta A||| \cdot \|x\|) \\ &\leq \kappa(A)(|||A||| \cdot \|x\| \cdot \|\Delta b\| + \|b\| \cdot |||\Delta A||| \cdot \|x\|) \end{aligned}$$

On en conclut que $\|b\| \cdot |||A||| \cdot \|\Delta x\| \leq \kappa(A) \cdot \|x\| \cdot (|||A||| \cdot \|\Delta b\| + \|b\| \cdot |||\Delta A|||)$ et on en déduit le résultat. □

Remarque. Si on perturbe A et b, la perturbation sur la solution est la somme de chacune des perturbations prise séparément.

2.2.5 Opérations vectorielles et matricielles

Théorème 2.10 (Produit scalaire).
Soit n un entier naturel, $x = (x_1, \ldots, x_n)$ et $y = (y_1, \ldots, y_n)$ deux vecteurs de $\mathbb{R}^n$.
Alors $|\Delta({}^t x\, y)| \leq \sum_{k=1}^{n} \gamma_{n+2-k} \cdot (x_k\, y_k)$ où $\gamma_k = \dfrac{k\varepsilon_{\text{mach}}}{1 - k\varepsilon_{\text{mach}}}$ pour tout $k \in [\![1, n]\!]$ et $\varepsilon_{\text{mach}}$ est la précision du système de représentation utilisé.

Remarque. La suite $(\gamma_k)_{k \in \mathbb{N}}$ étant croissante, la propagation se concentre sur les premiers termes.

Théorème 2.11 (Produit matrice-vecteur).
Soit n et m des entiers naturels non nul, $\|\cdot\|_n$ une norme sur $\mathbb{R}^n$ et $\|\cdot\|_m$ une norme sur $\mathbb{R}^m$.
On note $|||\cdot|||_{n,m}$ la norme matricielle subordonnée à $\|\cdot\|_n$ et $\|\cdot\|_m$.

$$\forall A \in \mathscr{M}_{n,m}(\mathbb{R}),\ \forall x \in \mathbb{R}^n,\ |\Delta(A\cdot x)| \leq \frac{n\varepsilon_{\text{mach}}}{1-n\varepsilon_{\text{mach}}} \cdot |||A|||_{n,m} \cdot \|x\|_n$$

Théorème 2.12 (Produit matriciel). Soit m, n et p des entiers naturels non nuls, $\|\cdot\|_n$ une norme sur $\mathbb{R}^n$, $\|\cdot\|_m$ une norme sur $\mathbb{R}^m$ et $\|\cdot\|_p$ une norme sur $\mathbb{R}^p$.
On note $|||\cdot|||_{n,m}$ la norme matricielle subordonnée à $\|\cdot\|_n$ et $\|\cdot\|_m$ et $|||\cdot|||_{m,p}$ la norme matricielle subordonnée à $\|\cdot\|_m$ et $\|\cdot\|_p$.

$$\forall A \in \mathscr{M}_{pm}(\mathbb{R}),\ \forall B \in \mathscr{M}_{mn}(\mathbb{R}),\ |\Delta(A\cdot B)| \leq \frac{n\varepsilon_{\text{mach}}}{1-n\varepsilon_{\text{mach}}} \cdot |||A|||_{m,p} \cdot |||B|||_{n,m}$$

2.2.6 Préconditionnement

Définition 2.10 (Préconditionnement).
Préconditionner un système linéaire (A,b), c'est déterminer un système linéaire (A',b') équivalent à (A,b) — c'est-à-dire qui a exactement les mêmes solutions — tel que $\kappa(A') < \kappa(A)$.
Un **préconditionneur de (A,b)** est une matrice carrée inversible P telle que $\kappa(P^{-1}A) < \kappa(A)$. Auquel cas, $(P^{-1}A, P^{-1}b)$ est le **système préconditionné**.

Remarque. On rappelle que réduire le conditionnement d'un système linéaire permet de réduire sa sensibilité aux erreurs sur ses paramètres.

Le préconditionneur idéal théoriquement est $P = A$: dans la pratique, il faudrait alors calculer l'inverse de A…autant résoudre le système linéaire lui-même !

On choisit donc des « approximations » P de A pour lesquelles le calcul de l'inverse de P est peu coûteux :

- P est la matrice diagonale composée des éléments diagonaux de A : P est appelé préconditionneur *diagonal* (ou *de* JACOBI) ;
- P est la part triangulaire inférieure ou supérieure de A ;
- P peut être dérivée de factorisations incomplètes de A (LU, CHOLESKY, …)
- …

On peut également réaliser une suite de préconditionnements pour obtenir une matrice P tel que $\|PA-I\|$ soit suffisamment petit.

Perturbations de systèmes linéaires : Exercices

Normes vectorielles et matricielles

Exercice 2.1.

1. *Calculer* $\|x\|_1$, $\|x\|_2$ *et* $\|x\|_\infty$ *pour* $x = (1,0,1,4)$.
2. *Calculer* $|||A|||_1$, $|||A|||_2$ *et* $|||A|||_\infty$ *pour* $A = \begin{pmatrix} 2 & 1 & 1 \\ 2 & 3 & 2 \\ 1 & 1 & 2 \end{pmatrix}$.
3. *Montrer que* $|||A|||_2 = 1$ *pour toute matrice orthogonale* A.

Exercice 2.2. *Pour tout entier naturel non nul* p*, on définit la norme* $\|\cdot\|_p : \begin{array}{ccc} \mathbb{R}^n & \to & \mathbb{R} \\ x & \mapsto & \left(\sum_{k=1}^{p} |x_i|^p\right)^{1/p} \end{array}$.

1. *Montrer que* $\forall x \in \mathbb{R}^n$, $\|x\|_\infty \leq \|x\|_p \leq n^{1/p}\|x\|_\infty$.
2. *En déduire que* $\lim_{p\to+\infty} \|x\|_p = \|x\|_\infty$.

Préconditionnement

Exercice 2.3. *Prouver la proposition 2.6 page 18.*

Exercice 2.4. *Soit* A *une matrice carrée d'ordre* n *inversible et* B *une approximation de* A^{-1} *telle qu'il existe une norme d'algèbre* $\|\cdot\|$ *sur* $\mathcal{M}_n(\mathbb{R})$ *telle que* $\rho = \|I - AB\| < 1$.

On définit les suites récurrentes $(B_k)_{k\in\mathbb{N}}$ *et* $(\Delta A_k)_{k\in\mathbb{N}}$ *par* $\begin{cases} B_0 = B \\ \forall k \in \mathbb{N},\ \Delta A_k = I - AB_k \\ \forall k \in \mathbb{N},\ B_{k+1} = B_k(I + \Delta A_k) \end{cases}$

1. (a) *Montrer que pour tout entier naturel* k*, on a* $B_k = A^{-1}\left(I - \Delta A_0^{2^k}\right)$.

   (b) *En déduire que* $\|B_k - A^{-1}\| \leq \|B\| \dfrac{\rho^{2^m}}{1-\rho}$.

2. *Soit* $A = \begin{pmatrix} 1 & 0{,}42 & 0{,}54 & 0{,}66 \\ 0{,}42 & 1 & 0{,}32 & 0{,}44 \\ 0{,}54 & 0{,}32 & 1 & 0{,}22 \\ 0{,}66 & 0{,}44 & 0{,}22 & 1 \end{pmatrix}$.

   (a) *En utilisant la procédure décrite ci-dessus, améliorer le calcul de* A^{-1} *en* SciLAB.

   (b) *Peut-on quantifier l'amélioration obtenue ?*

Chapitre 3

Méthodes directes de résolution de systèmes linéaires

La complexité d'un calcul de déterminant est factorielle, c'est-à-dire sur-exponentielle. La formule générale de résolution du théorème 2.5 page 18 devient donc numériquement irréaliste.

Dans ce chapitre, nous présenterons des algorithmes de résolution qui s'appuient sur la possibilité de décomposer un système en une composition de systèmes linéaires plus simples (qui seront présentés en début de chapitre), ce qui permet d'obtenir une complexité en $\mathrm{O}(n^3)$.

3.1 Résolution de systèmes linéaires

3.1.1 Systèmes linéaires « simples »

3.1.1.1 Systèmes linéaires diagonaux

Proposition 3.1. Soit n un entier naturel non nul et D une matrice diagonale d'ordre n.

Alors pour tout vecteur $b \in \mathbb{R}^n$, le système (D, b) est de CRAMER si et seulement tous les coefficients diagonaux de D sont non nuls. Auquel cas :

$$D\,x = b \iff \forall\, i \in [\![1, n]\!]\,,\ x_i = \frac{b_i}{D_{ii}}$$

Proposition 3.2 (Complexité linéaire de la résolution d'un système linéaire diagonal).

Soit n un entier naturel non nul. Alors l'algorithme 3 page suivante nécessite $\mathrm{O}(n)$ opérations arithmétiques et $\mathrm{O}(n)$ incrémentations de compteur.

3.1.1.2 Systèmes linéaires triangulaires

Proposition 3.3. Soit n un entier naturel non nul et T une matrice triangulaire d'ordre n.

Alors pour tout vecteur $b \in \mathbb{R}^n$, le système (T, b) est de CRAMER si et seulement tous les coefficients diagonaux de T sont non nuls.

fonction diagSolve(D : Matrice de réel, b : Tableau de réel) : Tableau de réel
préconditions : D carrée d'ordre n et diagonale
  pour $i \leftarrow 1$ **à** n **faire**
    $x_i \leftarrow \dfrac{b_i}{d_{ii}}$
  fin pour
  retourner x
postconditions : $Dx = b$
fin fonction

Algorithme 3: Résolution d'un système linéaire diagonal

fonction upperSolve(U : Matrice de réel, b : Tableau de réel) : Tableau de réel
préconditions : U carrée d'ordre n et triangulaire supérieure
  $x_n \leftarrow \dfrac{b_n}{u_{nn}}$
  pour $i \leftarrow n-1$ **à** 1 **par** -1 **faire**
    $x_i \leftarrow b_i$
    pour $j \leftarrow i+1$ **à** n **faire**
      $x_i \leftarrow x_i - u_{ij} \cdot x_j$
    fin pour
    $x_i \leftarrow \dfrac{x_i}{u_{ii}}$
  fin pour
  retourner x
postconditions : $Ux = b$
fin fonction

Algorithme 4: Résolution d'un système linéaire triangulaire supérieur

Proposition 3.4 (Complexité quadratique de la résolution d'un système linéaire triangulaire).
Soit n un entier naturel non nul. Alors l'algorithme 4 nécessite $O(n^2)$ opérations arithmétiques et $O(n^2)$ incrémentations de compteur.

Remarque. On obtient évidemment un algorithme de performance identique pour un système linéaire triangulaire inférieur.

3.1.2 Algorithme du pivot de GAUSS (Rappel)

L'algorithme du pivot de GAUSS est rappelé page suivante.

ATTENTION. *Dans l'algorithme 5 page suivante, le test « $a_{ii} = 0$ » est numériquement instable et laisse la possibilité à des erreurs numériques non contrôlées.*
Il faut préférer un test « $|a_{ii}| < \varepsilon$ » où ε désigne une petite valeur strictement positive.

Proposition 3.5 (Complexité cubique de la méthode du pivot de GAUSS).
Soit n un entier naturel non nul. Alors l'algorithme 5 page suivante nécessite $O(n^3)$ opérations arithmétiques et $O(n^3)$ incrémentations de compteur.

Entrées : A (matrice $n \times n$), b (vecteur de $\mathbb{R}^n$)
Sortie : x (vecteur de $\mathbb{R}^n$)
pour $i \leftarrow 1$ **à** $n-1$ **faire**
  si $a_{ii} = 0$ **alors** /* *le pivot actuel est nul* */
    i_1 est choisi dans $\{\ell \in [\![i+1, n]\!],\ a_{\ell i} \neq 0\}$
    $L_i \longleftrightarrow L_{i_1}$
  fin si
  pour $k \leftarrow i+1$ **à** n **faire**
    $L_k \leftarrow L_k - \dfrac{a_{ki}}{a_{ii}} L_i$
  fin pour
fin pour
Résoudre $Ax = b$ /* *A est à présent triangulaire supérieure* */

Algorithme 5: Méthode du pivot de GAUSS

3.2 Factorisation LU

3.2.1 Introduction

Définition 3.1 (Factorisation LU). Soit n un entier naturel et A une matrice carrée d'ordre n.
On dit que ***A* admet une factorisation LU** si et seulement si il existe :
— une matrice L carrée d'ordre n triangulaire inférieure,
— une matrice U carrée d'ordre n triangulaire supérieure
telles que $A = LU$.

$$\underset{A}{\begin{pmatrix} * & \cdots & \cdots & * \\ \vdots & \ddots & \iddots & \vdots \\ \vdots & \iddots & \ddots & \vdots \\ * & \cdots & \cdots & * \end{pmatrix}} = \underset{L}{\begin{pmatrix} * & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & 0 \\ * & \cdots & \cdots & * \end{pmatrix}} \underset{U}{\begin{pmatrix} * & \cdots & \cdots & * \\ 0 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & * \end{pmatrix}}$$

Si les coefficients diagonaux de L sont tous égaux à un, c'est une **factorisation de DOOLITTLE** :

$$\begin{pmatrix} * & \cdots & \cdots & * \\ \vdots & \ddots & \iddots & \vdots \\ \vdots & \iddots & \ddots & \vdots \\ * & \cdots & \cdots & * \end{pmatrix} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ * & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ * & \cdots & * & 1 \end{pmatrix} \begin{pmatrix} * & \cdots & \cdots & * \\ 0 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & * \end{pmatrix}$$

Si les coefficients diagonaux de U sont tous égaux à un, c'est une **factorisation de CROUT** :

$$\begin{pmatrix} * & \cdots & \cdots & * \\ \vdots & \ddots & \iddots & \vdots \\ \vdots & \iddots & \ddots & \vdots \\ * & \cdots & \cdots & * \end{pmatrix} = \begin{pmatrix} * & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & 0 \\ * & \cdots & \cdots & * \end{pmatrix} \begin{pmatrix} 1 & * & \cdots & * \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & * \\ 0 & \cdots & 0 & 1 \end{pmatrix}$$

Remarque. Nous considérerons systématiquement la factorisation de DOOLITTLE.

IMPORTANT. La factorisation LU est fortement liée à la méthode du pivot de GAUSS. En effet :

— la matrice U est la matrice du système triangulaire supérieur obtenue à l'issue de l'application de la méthode ;

— la matrice L contient les informations sur les opérations de pivotage qui ont été réalisées.

3.2.2 Algorithme de calcul d'une factorisation LU

Avant de présenter un algorithme de calcul de factorisation LU, nous allons d'abord nous intéresser aux applications possibles de cette factorisation.

3.2.2.1 Applications de la factorisation LU

Résolution de système linéaire Si on dispose d'une factorisation LU de A, alors on peut résoudre le système $Ax = b$, c'est-à-dire $L(Ux) = b$ en deux temps :

1. on résout le système linéaire triangulaire inférieur $Ly = b$;
2. puis on résout le système linéaire triangulaire supérieur $Ux = y$.

Cette méthode est présentée dans l'algorithme 6.

fonction linsolveByLU(A : Matrice de réel, b : Tableau de réel) : Tableau de réel
préconditions : A carrée d'ordre n admettant une factorisation LU
  $(L, U) \leftarrow \mathrm{LU}(A)$
  $y \leftarrow \mathrm{lowerSolve}(L, b)$
  $x \leftarrow \mathrm{upperSolve}(U, y)$
  retourner x
postconditions : $Ax = b$
fin fonction

Algorithme 6: Résolution de système linéaire par factorisation LU

Remarque. Si la même matrice A est utilisée pour de nombreuses résolutions, sa factorisation LU peut être réutilisée pour chaque système sans avoir à la recalculer.

Calcul de déterminant Si on dispose d'une factorisation de DOOLITTLE de A, alors

$$\det A = \det(LU) = (\det L) \cdot (\det U)$$

L est triangulaire, donc $\det L$ est le produit de ses coefficients diagonaux : donc $\det L = \prod_{i=1}^{n} l_{ii} = 1$.

U est triangulaire, donc $\det U$ est le produit de ses coefficients diagonaux : donc $\det U = \prod_{i=1}^{n} u_{ii}$.

Donc $\det A = \prod_{i=1}^{n} u_{ii}$: cette méthode est présentée dans l'algorithme 7 page suivante.

fonction detByLU(A : Matrice de réel) : réel
préconditions : A carrée d'ordre n admettant une factorisation LU
  $(L,U) \leftarrow \mathsf{LU}(A)$
  $d \leftarrow 1$
  pour $i \leftarrow 1$ **à** n **faire**
    $d \leftarrow d \times u_{ii}$
  fin pour
  retourner d
postconditions : $d = \det A$
fin fonction

Algorithme 7: Calcul de déterminant par factorisation LU

3.2.2.2 Algorithme « en place »

Suite aux remarques qui viennent d'être faites, on peut aller plus loin : une fois la factorisation LU, la connaissance de A devient inutile.

Il existe alors un algorithme, dit « en place » [1], qui consiste à stocker les données de la factorisation en écrasant A. En effet, dans le cas d'une factorisation de DOOLITTLE, après avoir appliqué la version « en place » de l'algorithme (cf. figure 3.1) :

— la partie triangulaire inférieure stricte (i.e. diagonale non incluse) contiendra les éléments de L [2] ;

— la partie triangulaire supérieure, diagonale incluse, contiendra les éléments de U.

Cette méthode est présenté dans l'algorithme 8 page suivante.

Remarque. La matrice effectivement stockée est $(L - I_n) + U$.

$$\begin{pmatrix} u_{1,1} & u_{1,2} & \cdots & u_{1,n} \\ l_{2,1} & u_{2,2} & & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ l_{n,1} & \cdots & l_{n,n-1} & u_{n,n} \end{pmatrix}$$

FIGURE 3.1 – Stockage de la factorisation LU dans l'algorithme « en place »

Remarque. On peut définir un stockage équivalent dans le cas d'une factorisation de CROUT.

1. traduction littérale (mais maladroite) du terme anglais *in place*
2. il est inutile de stocker la diagonale car on sait que tous ses éléments valent un dans une factorisation de DOOLITTLE

procedure LUInPlace(A : Matrice de réel)
préconditions : A carrée d'ordre n admettant une factorisation LU : $A = LU$
pour $j \leftarrow 1$ **à** $n-1$ **faire**
pour $i \leftarrow j+1$ **à** n **faire**
$a_{ij} \leftarrow \dfrac{a_{ij}}{a_{jj}}$
pour $k \leftarrow j+1$ **à** n **faire**
$a_{ik} \leftarrow a_{ik} - a_{ij}a_{jk}$
fin pour
fin pour
fin pour
postconditions : $A = (L - I_n) + U$
fin procedure

Algorithme 8: Calcul de factorisation de DOOLITTLE « en place »

Théorème 3.1 (Analyse d'erreurs sur la factorisation LU). Soit n un entier naturel non nul, A une matrice carrée d'ordre n admettant une factorisation LU : $A = L \cdot U$.

$$|\Delta(L \cdot U)| \leq \frac{n\varepsilon_{\text{mach}}}{1 - n\varepsilon_{\text{mach}}} \cdot |||L|||_n \cdot |||U|||_n$$

où $|||\cdot|||_n$ est la norme subordonnée à $\|\cdot\|_n$.

Proposition 3.6 (Complexité cubique d'un calcul de factorisation LU « en place »).
Soit n un entier naturel non nul. Alors l'algorithme 8 nécessite $O(n^3)$ opérations arithmétiques et $O(n^3)$ incrémentations de compteur.

Corollaire (Complexité cubique d'une résolution de système linéaire par factorisation LU).
Soit n un entier naturel non nul. Alors les algorithmes 6 page 26 et 7 page précédente nécessite $O(n^3)$ opérations arithmétiques et $O(n^3)$ incrémentations de compteur.

Remarque. Il s'agit donc d'une grande amélioration par rapport à l'algorithme « naïf » — i.e. avec les mineurs — de complexité $O(n!)$.

Démonstration. Décomposons les trois étapes de l'algorithme 6 :

1. la factorisation LU est en $O(n^3)$;
2. la résolution d'un système triangulaire inférieur est en $O(n^2)$;
3. la résolution d'un système triangulaire supérieur est en $O(n^2)$.

D'où la complexité annoncée. □

3.2.3 Condition d'existence d'une factorisation LU

Théorème 3.2 (Existence et unicité d'une factorisation LU).
Soit n un entier naturel non nul et A une matrice carrée d'ordre n.
Pour tout $p \in [\![1, n]\!]$, on note A_p la matrice carrée d'ordre p définie par $A_p = (a_{ij})_{1 \leq i,j \leq p}$.

A admet une factorisation LU $\iff$ les matrices A_p sont inversibles pour tout $p \in [\![1, n-1]\!]$.

Si les matrices A_p sont inversibles pour tout $p \in [\![1, n]\!]$, alors la factorisation LU est unique.

ATTENTION. Il existe des matrices inversibles qui ne vérifient pas ces conditions.

Exemple. $A = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix}$. En effet $\det A_2 = \begin{vmatrix} 1 & 1 \\ 1 & 1 \end{vmatrix} = 0$, donc A_2 n'est pas inversible.

Corollaire (Lien avec la méthode du pivot de GAUSS).
Soit n un entier naturel non nul et A une matrice carrée d'ordre n.
A admet une factorisation LU si et seulement si lors de l'application de la méthode du pivot de GAUSS (algorithme 5 page 25), aucun échange de ligne n'est nécessaire [3].

3.2.4 Algorithme de calcul d'une factorisation LU avec pivot partiel

Si des échanges de lignes sont nécessaires durant l'application de la méthode du pivot de GAUSS, on peut envisager de faire ces échanges préalablement pour obtenir une matrice qui vérifie les conditions du théorème 3.2 : c'est en substance le contenu du théorème 3.3 page suivante.

Définition 3.2 (Matrice de permutation). Soit n un entier naturel non nul.
Une **matrice de permutation d'ordre *n*** est une matrice carrée d'ordre n telle que :
- seul un élément par ligne est non nul et il vaut un ;
- seul un élément par colonne est non nul et il vaut un.

Exemple. $P = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$ est une matrice de permutation d'ordre quatre.

3. c'est-à-dire que la condition $a_{ii} = 0$ n'est jamais vérifiée

Théorème 3.3 (Condition d'existence de factorisation LU avec pivot partiel).
Soit n un entier naturel non nul et A une matrice carrée d'ordre n inversible.
Alors il existe une matrice de permutation P telle que PA admette une factorisation LU.
La donnée de P, L et U constitue une **factorisation LU avec pivot partiel**.

Nous présentons dans l'algorithme 9 page suivante une modification de l'algorithme 8 page 28 dite « de pivot maximal ». Cette méthode consiste à systématiquement échanger la ligne courante avec celle qui donnera le pivot le plus grand en valeur absolue, assurant ainsi une bien meilleure stabilité numérique.

L'algorithme permet également de calculer P au fur et à mesure des permutations.

Proposition 3.7 (Complexité cubique d'un calcul de factorisation LU avec pivot maximal).
Soit n un entier naturel non nul. Alors l'algorithme 9 page suivante nécessite $O(n^3)$ opérations arithmétiques et $O(n^3)$ incrémentations de compteur.

3.3 Factorisation de CHOLESKY

Définition 3.3 (Factorisation de CHOLESKY).
Soit n un entier non nul et A une matrice carrée d'ordre n.
On dit que A **admet une factorisation de CHOLESKY** (ou **une factorisation** $LL^\top$) si et seulement il existe une matrice L carrée triangulaire inférieure telle que $A = LL^\top$.

ATTENTION. La matrice L définie ici n'a rien à voir avec la matrice L issue de la factorisation LU.

Théorème 3.4 (Existence et unicité d'une factorisation de CHOLESKY).
Soit n un entier non nul et A une matrice carrée d'ordre n.
Alors A admet une factorisation de CHOLESKY si et seulement si A est symétrique et positive.
De plus si A est inversible, alors la factorisation est unique.

Proposition 3.8 (Complexité cubique d'un calcul de factorisation de CHOLESKY).
Soit n un entier naturel non nul. Alors l'algorithme 10 page 32 nécessite $O(n^3)$ opérations arithmétiques et $O(n^3)$ incrémentations de compteur.

fonction LUPivotMax(A : Matrice de réel) : Matrice de réel
préconditions : A carrée d'ordre n admettant une factorisation LU avec pivot partiel : $PA = LU$
  /* *Initialisation du vecteur de permutation :* $\pi = \begin{pmatrix} 1 & 2 & \cdots & n \end{pmatrix}$ */
  pour $j \leftarrow 1$ **à** n **faire**
    $\pi_j \leftarrow j$
  fin pour
  pour $j \leftarrow 1$ **à** $n-1$ **faire**
    /* *Recherche du pivot maximal* j_1 */
    $j_1 \leftarrow j$
    pour $i \leftarrow j+1$ **à** n **faire**
      si $|a_{j_1 k}| < |a_{ik}|$ **alors**
        $j_1 \leftarrow i$
      fin si
    fin pour
    /* *Échange des lignes* j *et* j_1 *de* A *et mise à jour de* π */
    pour $k \leftarrow j$ **à** n **faire**
      $a_{jk} \longleftrightarrow a_{j_1 k}$
      $\pi_j \longleftrightarrow \pi_{j_1}$
    fin pour
    /* *Étape LU standard* */
    pour $i \leftarrow j+1$ **à** n **faire**
      $a_{ij} \leftarrow \dfrac{a_{ij}}{a_{jj}}$
      pour $k \leftarrow j+1$ **à** n **faire**
        $a_{ik} \leftarrow a_{ik} - a_{ij} a_{jk}$
      fin pour
    fin pour
  fin pour
  /* *Construction de la matrice de permutation* */
  $P \leftarrow \mathrm{O}_n$ /* *Matrice nulle d'ordre n* */
  pour $j \leftarrow 1$ **à** n **faire**
    $p_{i,\pi_i} \leftarrow 1$
  fin pour
  retourner P
postconditions : P est une matrice de permutation d'ordre n
postconditions : $PA = (L - I_n) + U$
fin fonction

Algorithme 9: Calcul d'une factorisation LU avec pivot partiel

fonction cholesky(A : Matrice de réel) : Matrice de réel
préconditions : A carrée d'ordre n symétrique et positive
  $L \leftarrow \mathrm{O}_n$ /* *Matrice nulle d'ordre n* */
  $l_{11} \leftarrow \sqrt{a_{11}}$
  pour $i \leftarrow 2$ **à** n **faire**
    pour $j \leftarrow 1$ **à** $i-1$ **faire**
      $l_{ij} \leftarrow a_{ij}$
      pour $k \leftarrow 1$ **à** $j-1$ **faire**
        $l_{ij} \leftarrow l_{ij} - l_{ik} l_{jk}$
      fin pour
      $l_{ij} \leftarrow \dfrac{l_{ij}}{l_{jj}}$
    fin pour
    $l_{ii} \leftarrow a_{ii}$
    pour $k \leftarrow 1$ **à** $i-1$ **faire**
      $l_{ii} \leftarrow l_{ii} - l_{ik}^2$
    fin pour
    $l_{ii} \leftarrow \sqrt{l_{ii}}$
  fin pour
  retourner L
postconditions : L carrée d'ordre n triangulaire inférieure
postconditions : $A = LL^\top$
fin fonction

Algorithme 10: Calcul d'une factorisation de CHOLESKY

3.4 Autres factorisations

3.4.1 Factorisation QR

Théorème (Factorisation QR). Soit n un entier naturel non nul.
Pour toute matrice carrée d'ordre n, il existe :
— une matrice Q carrée d'ordre n orthogonale (i.e. $Q^{-1} = Q^\top$),
— une matrice R carrée d'ordre n triangulaire supérieure
telles que $A = QR$.
La donnée de Q et R constitue **une factorisation QR de A**.

Remarque. Il s'agit donc d'une factorisation sans condition particulière d'existence.

Comme la factorisation LU est une traduction matricielle de la méthode du pivot de GAUSS, la factorisation QR est une traduction matricielle de la méthode d'orthonormalisation de GRAM-SCHMIDT.

Résolution de système linéaire Si on dispose d'une factorisation QR de A, alors on peut résoudre le système $Ax = b$, c'est-à-dire $Q(Rx) = b$ en deux temps :

1. on résout le système linéaire $Qy = b$ par $y = Q^\top b$;
2. puis on résout le système linéaire triangulaire supérieur $Rx = y$.

3.4.2 Factorisation de SCHUR

Théorème (Factorisation de SCHUR). Soit n un entier naturel non nul.
Pour toute matrice carrée d'ordre n, il existe :

— une matrice Q carrée d'ordre n orthogonale (i.e. $Q^{-1} = Q^\top$),

— une matrice U carrée d'ordre n triangulaire supérieure

telles que $A = QUQ^\top$.

La donnée de Q et U constitue **une factorisation de SCHUR de A**.

Remarque. Il s'agit donc d'une factorisation sans condition particulière d'existence.

Corollaire. Soit n un entier naturel non nul et A une matrice carrée d'ordre n.

Si $A = QUQ^\top$ est la factorisation de SCHUR de A, alors les valeurs propres sont les coefficients diagonaux de U.

Démonstration. On a $A = QUQ^{-1}$: donc A est semblable à U. Donc les spectres de A et U sont identiques (multiplicités incluses). Comme U est triangulaire, on peut conclure. □

Résolution de système linéaire Si on dispose d'une factorisation de SCHUR de A, alors on peut résoudre le système $Ax = b$, c'est-à-dire $QUQ^{-1}x = b$ en trois temps :

1. on résout le système linéaire $Qz = b$ par $z = Q^\top b$;
2. puis on résout le système linéaire triangulaire supérieur $Uy = z$;
3. enfin on résout le système linéaire $Q^{-1}x = y$ par $x = Qy$.

Méthodes directes de résolution de systèmes linéaires : Exercices

Exercice 3.1. *Écrire l'algorithme de résolution d'un système triangulaire inférieur.*

Exercice 3.2. *Soit n un entier naturel non nul et A une matrice carrée d'ordre n inversible.*

1. *On suppose que A admet une factorisation LU.*
   Montrer qu'il existe un algorithme de complexité $\mathrm{O}(n^3)$ pour calculer A^{-1}.
2. *Est-ce toujours possible si A admet une factorisation LU avec pivot partiel ?*

Exercice 3.3. *Soit ε un réel quelconque et* $A = \begin{pmatrix} 1 & 1-\varepsilon & 3 \\ 2 & 2 & 2 \\ 3 & 6 & 4 \end{pmatrix}$

1. *Déterminer les valeurs de ε pour lesquelles A admet une factorisation LU.*
2. *Déterminer les valeurs de ε pour lesquelles A n'est pas inversible.*
   Pour cesdites valeurs, A admet-elle une factorisation LU ?

Exercice 3.4. *Soit n un entier naturel non nul.*
On note A la matrice carrée d'ordre n définie par $A = (i^{j-1})_{1 \leq i,j \leq n}$.

1. *Implémenter l'algorithme de calcul de factorisation LU en* SciLAB.
   Le modifier pour qu'il compte les nombres d'opérations arithmétiques effectuées.
2. *Calculer le nombre d'opérations du calcul de la factorisation LU de A pour $n \in \{10, 20, 30, 40, 50, 60\}$.*
   Représenter graphiquement le résultat.
3. *Retrouver l'ordre de complexité de la méthode.*
   Indication. On pourra utiliser :
   — l'option `logflag` de la fonction `plot2d` pour afficher des échelles logarithmiques ;
   — la fonction `reglin` qui permet de calculer les coefficients de régression linéaire d'un nuage de points.

Exercice 3.5. *Soit* $A = \begin{pmatrix} 1 & 1+5\cdot 10^{-16} & 3 \\ 2 & 2 & 20 \\ 3 & 6 & 4 \end{pmatrix}$.

1. *Vérifier que A est inversible et qu'elle admet une factorisation* LU.
2. *Calculer la factorisation* LU *de A à l'aide du code* SciLAB *de l'exercice précédent.*
   Vérifier que $A - LU = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 4 \end{pmatrix}$. *Expliquer.*
3. *Analyser la différence entre ce résultat et celui obtenu avec la fonction* SciLAB `lu`.

Chapitre 4

Méthodes de descente

Les méthodes directes ont des limitations, notamment lorsque le problème est de grande taille. Une alternative réside dans l'utilisation d'algorithmes itératifs.

De tels algorithmes génèrent une suite de solutions approchées qui converge vers la solution du système linéaire. Nous présenterons ici une famille d'algorithmes itératifs : les méthodes de descente.

4.0 Motivation

Proposition 4.1 (Fonctionnelle quadratique).
Soit $n \in \mathbb{N}^*$, $A \in \mathscr{M}_n(\mathbb{R})$ et $b \in \mathscr{M}_{n1}(\mathbb{R})$ et $J : \begin{array}{ccc} \mathscr{M}_{n1}(\mathbb{R}) & \to & \mathbb{R} \\ x & \mapsto & \frac{1}{2}\left(x^\top A x\right) - x^\top b \end{array}$.

Alors J est de classe $\mathscr{C}^\infty$ sur $\mathscr{M}_{n1}(\mathbb{R})$ et :

$$\forall x \in \mathscr{M}_{n1}(\mathbb{R}), \qquad \nabla J(x) = \frac{A + A^\top}{2} x - b \qquad \text{et} \qquad \nabla^2 J(x) = \frac{A + A^\top}{2}$$

Corollaire (Cas d'une matrice symétrique). Si A est symétrique, alors

$$\forall x \in \mathscr{M}_{n1}(\mathbb{R}), \qquad \nabla J(x) = Ax - b \qquad \text{et} \qquad \nabla^2 J(x) = A$$

Théorème 4.1 (Cas d'une matrice symétrique définie positive).
Soit n un entier naturel non nul, $A \in \mathscr{M}_n(\mathbb{R})$ et $b \in \mathscr{M}_{n1}(\mathbb{R})$.
Si A est symétrique définie positive alors $J : \begin{array}{ccc} \mathscr{M}_{n1}(\mathbb{R}) & \to & \mathbb{R} \\ x & \mapsto & \frac{1}{2}\left(x^\top A x\right) - x^\top b \end{array}$ admet un unique minimum local et global qui est l'unique solution du système $Ax = b$.

DONC RÉSOUDRE $Ax = b$ REVIENT À MINIMISER J.

Remarque. Les matrices symétriques définies positives apparaissent fréquemment dans les problèmes numériques, notamment lors de résolution d'équations aux dérivées partielles.

IMPORTANT. Si A n'est pas symétrique définie positive, mais qu'elle est inversible, alors $A^\top A$ est une matrice symétrique définie positive. On peut alors considérer le système linéaire $(A^\top A, A^\top b)$.

Toutefois, comme $\kappa(A^\top A) = kappa(A)^2$, un système linéaire (A,b) mal conditionné engendrera un système $(A^\top A, A^\top b)$ encore plus mal conditionné !

Les algorithmes que nous allons présenter dans ce chapitre s'inspirent d'algorithmes usuels en optimisation non linéaire et vont ainsi générer une suite de vecteurs $(x_k)_{k\in\mathbb{N}}$ telle que la suite $(J(x_k))_{k\in\mathbb{N}}$ soit strictement décroissante.

Entrées : x_0 (vecteur-colonne de longueur n)
Sorties : x (vecteur-colonne de longueur n)
$k \leftarrow 0$
répéter
  $k \leftarrow k+1$
  d_{k-1} est choisi dans $\{d \in \mathbb{R}^n, \exists \alpha > 0,\ J(x_{k-1} + \alpha \cdot d) \leq J(x_{k-1})\}$ /* *Direction de descente* */
  α_{k-1} est choisi dans $\{\alpha > 0,\ J(x_{k-1} + \alpha \cdot d_{k-1}) \leq J(x_{k-1})\}$ /* *Pas de descente* */
  $x_k \leftarrow x_{k-1} + \alpha_{k-1} \cdot d_{k-1}$
jusqu'à $J(x_k) \geq J(x_{k-1})$
$x \leftarrow x_{k-1}$

Algorithme 11: Principe général d'une méthode de descente

Remarque. En pratique, ce principe est évidemment inapplicable tel quel. Il faudra en effet :
— affiner la condition d'arrêt sur le critère ;
— ajouter un contrôle sur le nombre d'itérations.

Définition 4.1 (Direction de descente).
Soit n un entier naturel, U une partie de $\mathcal{M}_{n1}(\mathbb{R})$ et J une application de U dans $\mathbb{R}$.
Le vecteur $d \in \mathcal{M}_{n1}(\mathbb{R})$ est une **direction de descente pour J en x** si et seulement si

$$\forall \gamma > 0,\ \exists \alpha \in]0, \gamma[,\ J(x + \alpha \cdot d) \leq J(x)$$

Remarque. Il existe au moins deux cas usuels :
— méthode de CAUCHY ou méthode de plus grande pente : $d = -\nabla J(x)$ est toujours une direction de descente pour J en x (sauf si x est un minimum local strict) ;
— méthode de NEWTON : $d = -\left[\nabla^2 J(x)\right]^{-1} \nabla J(x)$ n'est pas toujours une direction de descente pour J en x, mais donne une convergence plus rapide si c'est le cas.

DANS LE RESTE DU CHAPITRE, ON CONSIDÈRE UNIQUEMENT LA FONCTIONNELLE QUADRATIQUE :

$$J : x \mapsto \frac{1}{2} x^\top A x + x^\top b$$

4.1 Méthode de plus grande pente

À chaque itération de l'algorithme 11 page précédente, on choisit donc $d_{k-1} = -\nabla J(x_{k-1}) = b - Ax_{k-1}$ en ligne 6.

Entrées : x_0 (vecteur-colonne de longueur n), α_0, α_1, … (suite de réels), ε (réel)
Sorties : x (vecteur-colonne de longueur n)
préconditions : $\forall k \in \mathbb{N}$, $\alpha_k > 0$ et $\varepsilon > 0$
$k \leftarrow 0$
tant que $\|Ax_k - b\| > \varepsilon$ **faire**
  $k \leftarrow k+1$
  $x_k \leftarrow x_{k-1} + \alpha_k \cdot (b - Ax_{k-1})$
fin tant que
$x \leftarrow x_k$

Algorithme 12: Méthode de plus grande pente (version générale)

Remarque. Le test sur la norme du gradient se justifie par le fait que

$$J(x_k) - J(x_{k-1}) = \nabla J(x_{k-1})^\top [-\alpha_{k-1} \nabla J(x_{k-1})] + \mathrm{o}\left(\alpha_{k-1}^2\right) = -\alpha_{k-1} \|\nabla J(x_{k-1})\|^2 + \mathrm{o}\left(\alpha_{k-1}^2\right)$$

En remarquant que
$$\begin{aligned} d_k = b - Ax_k &= b - A[x_{k-1} + (x_k - x_{k-1})] \\ &= b - Ax_{k-1} - A(x_k - x_{k-1}) = d_{k-1} - A[\alpha_{k-1} d_{k-1}] \\ &= d_{k-1} - \alpha_{k-1} A d_{k-1}, \end{aligned}$$
on peut modifier l'algorithme de manière à ce qu'il mette à jour la direction de descente :

Entrées : x_0 (vecteur-colonne de longueur n), α_0, α_1, … (suite de réels), ε (réel)
Sorties : x (vecteur-colonne de longueur n)
préconditions : $\forall k \in \mathbb{N}$, $\alpha_k > 0$ et $\varepsilon > 0$
$k \leftarrow 0$
$d_0 \leftarrow b - Ax_0$
tant que $\|d_k\| > \varepsilon$ **faire**
  $k \leftarrow k+1$
  $x_k \leftarrow x_{k-1} + \alpha_k \cdot d_{k-1}$
  $d_k \leftarrow d_{k-1} - \alpha_k A d_{k-1}$
fin tant que
$x \leftarrow x_k$

Algorithme 13: Méthode de plus grande pente (version générale)

4.1.1 Version à pas constant

À chaque itération de l'algorithme 11 page 36, on fixe $\alpha_{k-1} = \alpha$ en ligne 7.

fonction constantSD(A : Matrice de réel, b : Tableau de réel, x_0 : Tableau de réel, α : réel, ε : réel) : Tableau de réel
  $x \leftarrow x_0$
  $d \leftarrow b - Ax$
  tant que $\|d\| > \varepsilon$ **faire**
    $x \leftarrow x + \alpha d$
    $d \leftarrow d - \alpha Ad$
  fin tant que
  retourner x
fin fonction

Algorithme 14: Méthode de plus grande pente (version à pas constant)

4.1.2 Version à pas variable optimal

À chaque itération de l'algorithme 11 page 36, on choisit en ligne 7 :

$$\alpha_{k-1} = \underset{h>0}{\operatorname{argmin}} J\big(x_{k-1} - h \cdot \Delta J(x_{k-1})\big)$$

Proposition 4.2 (Pas optimal pour une fonctionnelle quadratique). Soit n un entier naturel non nul, A une matrice carrée d'ordre n symétrique définie positive et $b \in \mathscr{M}_{n1}(\mathbb{R})$.

On note $J : \begin{array}{ccc} \mathscr{M}_{n1}(\mathbb{R}) & \to & \mathbb{R} \\ x & \mapsto & \frac{1}{2}\big(x^\top Ax\big) - x^\top b \end{array}$.

Alors $\underset{h>0}{\operatorname{argmin}} J\big(x - h \cdot \nabla J(x)\big) = \dfrac{\nabla J(x)^\top \nabla J(x)}{\nabla J(x)^\top A \nabla J(x)}$ avec $\nabla J(x) = Ax - b$.

On obtient donc l'algorithme 15.

fonction optimalSD(A : Matrice de réel, b : Tableau de réel, x_0 : Tableau de réel, ε : réel) : Tableau de réel
  $x \leftarrow x_0$
  $d \leftarrow b - Ax$
  tant que $\|d\| > \varepsilon$ **faire**
    $\alpha \leftarrow (d^\top d)/(d^\top Ad)$
    $x \leftarrow x + \alpha d$
    $d \leftarrow d - \alpha Ad$
  fin tant que
  retourner x
fin fonction

Algorithme 15: Méthode de plus grande pente (version à pas optimal)

4.2 Méthode du gradient conjugué

4.2.1 Définition et propriétés

Définition 4.2 (Direction. Directions conjuguées). Soit n un entier naturel non nul, A une matrice carrée d'ordre n symétrique définie positive.

Une **direction** est un vecteur $d \in \mathscr{M}_{n1}(\mathbb{R})$ non nul.

Deux directions d_1 et d_2 sont dites **conjuguées par rapport à A** ou plus simplement **A-conjuguées** si et seulement si $d_1^\top A d_2 = 0$.

Proposition 4.3.

Soit n un entier naturel non nul, A une matrice carrée d'ordre n symétrique définie positive.

Toute famille de directions conjuguées deux à deux par rapport à A est libre.

Démonstration. Soit $\{d_i\}_{1\leq i\leq m}$ une famille de m directions conjuguées deux à deux par rapport à A, c'est-à-dire :

$$\forall (i,j) \in [\![1,m]\!]^2, (i \neq j) \Rightarrow d_i^\top A d_j = 0$$

Remarquons également que, comme A est définie positive, $d_i^\top A d_i$ est strictement positif pour tout $i \in [\![1,m]\!]$.

Soit $(\lambda_i)_{1\leq i\leq m} \in \mathbb{R}^m$ des réels tels que $\sum_{i=1}^{m} \lambda_i d_i = \mathrm{O}$.

Soit $k \in [\![1,m]\!]$. Alors : $0 = d_k^\top A \mathrm{O} = d_k^\top A \left(\sum_{i=1}^{m} \lambda_i d_i \right) = \sum_{i=1}^{m} \lambda_i d_k^\top A d_i = \lambda_k d_k^\top A d_k$. Comme $d_k^\top A d_k$ est non nul, on en déduit $\lambda_k = 0$ et ce pour tout $k \in [\![1,m]\!]$. Donc $\{d_i\}_{1\leq i\leq m}$ est une famille libre. □

Corollaire (Décomposition dans une base conjuguée).

Soit n un entier naturel non nul, A une matrice carrée d'ordre n symétrique définie positive.

Si $\{d_i\}_{1\leq i\leq n}$ est une famille de directions conjuguées deux à deux par rapport à A, alors c'est une base de $\mathscr{M}_{n1}(\mathbb{R})$ et

$$\forall x \in \mathscr{M}_{n1}(\mathbb{R}),\ x = \sum_{i=1}^{n} \frac{d_i^\top A x}{d_i^\top A d_i} d_i$$

Démonstration. D'après la proposition précédente, $\{d_i\}_{1\leq i\leq n}$ est une famille libre de $\mathscr{M}_{n1}(\mathbb{R})$. Comme $\dim \mathscr{M}_{n1}(\mathbb{R}) = n = \mathrm{card}\{d_i\}_{1\leq n}$, c'est une base de $\mathscr{M}_{n1}(\mathbb{R})$.

Soit $x \in \mathscr{M}_{n1}(\mathbb{R})$. Alors il existe $(x_i)_{1\leq i\leq n} \in \mathbb{R}^n$ tel que $x = \sum_{i=1}^{n} x_i d_i$ et :

$$\forall k \in [\![1,m]\!],\ d_k^\top A x = \sum_{i=1}^{n} x_i d_k^\top A d_i = x_k d_k^\top A d_k$$

car les directions $\{d_i\}_{1\leq i\leq n}$ sont conjuguées deux à deux. □

Remarque. Le résultat du corollaire permet de définir une résolution directe si l'on connaît une base de directions conjuguées deux à deux par rapport à A.

4.2.2 Algorithme du gradient conjugué

L'idée force de l'algorithme est de générer de manière itérative les directions conjuguées en partant de $-\nabla J(x_0) = b - Ax_0$ et de les intégrer dans un algorithme de descente à pas optimal.

fonction CG(A : Matrice de réel, b : Tableau de réel, x_0 : Tableau de réel, ε : réel) : Tableau de réel
  $x \leftarrow x_0$
  $r \leftarrow Ax - b$
  $d \leftarrow -r$
  tant que $\|r\| > \varepsilon$ **faire**
    $\alpha \leftarrow -(r^\top d)/(d^\top Ad)$
    $x \leftarrow x + \alpha d$
    $r \leftarrow r + \alpha Ad$
    $\beta \leftarrow (r^\top Ad)/(d^\top Ad)$
    $d \leftarrow -r + \beta d$
  fin tant que
  retourner x
fin fonction

Algorithme 16: Méthode du gradient conjugué

Théorème (Convergence). Soit n un entier naturel non nul, A une matrice carrée d'ordre n symétrique définie positive et $b \in \mathscr{M}_{n1}(\mathbb{R})$.

Alors l'algorithme 16 converge en au plus n itérations.

Démonstration. Les directions générées étant conjuguées deux à deux par rapport à A, les n premières forment une base et engendrent la solution selon la formule du corollaire 6 page précédente. □

ATTENTION. Ce résultat n'est valable que théoriquement et n'est plus garanti lorsqu'il est implémenté sur machine à cause des erreurs successives d'arrondi.

4.2.3 Version préconditionnée

Le préconditionnement consiste à réaliser un changement de variable $y = Cx$ avec C inversible. Alors $J(x) = J(C^{-1}x) = \frac{1}{2}(C^{-1}y)^\top A(C^{-1}y) - (C^{-1}y)^\top b = \frac{1}{2}y(C^{-\top}AC^{-1}y) - y^\top(C^{-\top}b)$.

Le système à résoudre est donc $C^{-\top}AC^{-1}y = C^{-\top}b$, c'est-à-dire $C^{-\top}Ax = C^{-\top}b$.

En tirant partie de cette transformation et en choisissant C de sorte que $\kappa(C^{-\top}AC^{-1})$ soit le plus faible possible, on obtient l'algorithme 17 page suivante avec $P = C^\top C$ qui est alors une matrice symétrique définie positive.

fonction PCG(A : Matrice de réel, b : Tableau de réel, x_0 : Tableau de réel, ε : réel, P : Matrice de réel) : Tableau de réel

$x \leftarrow x_0$
$r \leftarrow Ax - b$
$z \leftarrow$ solution de $Pz = r$
$d \leftarrow -r$
tant que $\|r\| > \varepsilon$ **faire**
$\alpha \leftarrow -(r^\top z)/(d^\top Ad)$
$x \leftarrow x + \alpha d$
$r \leftarrow r + \alpha Ad$
$z \leftarrow$ solution de $Pz = r$
$\beta \leftarrow (z^\top Ad)/(d^\top Ad)$
$d \leftarrow -z + \beta d$
fin tant que
retourner x
fin fonction

Algorithme 17: Méthode du gradient conjugué préconditionné

Chapitre 5

Décomposition en valeurs singulières. Problème aux moindres carrés linéaires

5.1 Décomposition en valeurs singulières

Définition 5.1 (Valeurs singulières). Soit m et n deux entiers naturels non nuls et $A \in \mathscr{M}_{np}(\mathbb{R})$.
Une **valeur singulière de A** est la racine carrée d'une valeur propre non nulle de $A^\top A$.
Un **vecteur singulier à gauche de A** est un vecteur propre de $AA^\top$.
Un **vecteur singulier à droite de A** est un vecteur propre de $A^\top A$.

Remarque. On rappelle que $AA^\top$ et $A^\top A$ sont symétriques positives : leurs valeurs propres sont positives et admettent une base orthonormale de vecteurs propres.

Proposition 5.1 (Valeurs singulières et rang). Le nombre de valeurs singulières d'une matrice est égale à son rang.

Théorème 5.1 (Décomposition en valeurs singulières).
Soit m et n deux entiers naturels non nuls et $A \in \mathscr{M}_{np}(\mathbb{R})$ de rang r. On note $(\sigma_i)_{1\leq i\leq r}$ les valeurs singulières de A.

5.2 Pseudo-inverse

5.3 Problème aux moindres carrés linéaires

Annexes

Annexe A

Analyse de propagation d'erreurs

En calcul scientifique, un algorithme consiste la plupart du temps à calculer des valeurs numériques $Y=(y_i)_{1\leq i\leq m}\in\mathbb{R}^m$ à partir d'entrées numériques $X=(x_i)_{1\leq i\leq n}\in\mathbb{R}^n$.

Définition A.1 (Opération. Opération élémentaire). On appellera **opération** la donnée de :
- deux entiers naturels non nuls n (nombre d'entrées) et m (nombre de sorties) ;
- un ouvert D de $\mathbb{R}^n$ (domaine de validité) ;
- une fonction f de D dans $\mathbb{R}^m$ de classe $\mathscr{C}^1$ sur D (l'opération proprement dite).

On appelle **opération élémentaire** toute opération qui peut s'exprimer en une instruction dans le langage utilisé.

Exemple. Ainsi les opérations arithmétiques usuelles sont des cas particuliers d'opérations élémentaires avec $n=2$ et $m=1$.

A.1 Erreurs d'une opération élémentaire

Théorème A.1 (Erreur absolue d'une opération élémentaire).
Soit $\phi : D\subset\mathbb{R}^n\to\mathbb{R}^m$ une opération élémentaire.
Alors l'erreur absolue du calcul $Y=\phi(X)$ est $\Delta Y=J_\phi(X)\cdot\Delta^I X$ où :
- $J_\phi(X)$ est la matrice jacobienne de ϕ en X ;
- ΔX est l'erreur absolue d'entrée sur X.

Notamment, pour tout entier j de $[\![1,m]\!]$, $\Delta Y_j=\sum_{i=1}^{n}\frac{\partial\phi_j}{\partial x_i}(X)\cdot\Delta X_i$

Démonstration. On a

$$\Delta Y=\phi(\widehat{X})-\phi(X)=\phi(X+\Delta X)-\phi(X)=J_\phi(X)\cdot\Delta X+\mathrm{o}(\Delta X)\simeq J_\phi(X)\cdot\Delta X$$

en négligeant les termes d'ordre strictement supérieur à un. □

A.2 Propagation d'erreurs dans un algorithme

Définition A.2 (Algorithme). Un **algorithme** est la donnée :
- — d'un entier naturel non nul p (nombre d'instructions) ;
- — de p opérations élémentaires $\phi_k : D_{k-1} \subset \mathbb{R}^{n_{k-1}} \to D_k \subset \mathbb{R}^{n_k}$ telles que $\phi_k(D_{k-1}) \subset D_k$ (les instructions proprement dites).

On définit l'opération associée à l'algorithme l'opération f définie par $f = \phi_p \circ \cdots \circ \phi_1$.

Remarque. Deux algorithmes différents peuvent conduire à la même opération.

Exemple.
$$\begin{array}{rccl} f: & \mathbb{R}^2 & \to & \mathbb{R} \\ & x & \mapsto & a^2-b^2=(a-b)(a+b) \end{array}$$

Théorème A.2 (Erreur de calcul dans un algorithme).
Soit un algorithme de p opérations élémentaires $\phi_1, \ldots, \phi_p$.
Pour tout $k \in [\![1,p]\!]$, on note $\phi^{(k)} = \phi_k \circ \cdots \circ \phi_1$ et $\psi^{(k)} = \phi_p \circ \cdots \circ \phi_{k+1}$.
Pour tout $X_0 \in D_0$ et tout $k \in [\![1,p]\!]$, on note $X_k = \phi^{(k)}(x_0)$. Alors

$$\Delta X_p = \sum_{k=1}^{p} J_{\psi^{(k)}}(X_k) \cdot [X_k \otimes \eta_k^I] + J_f(X_0) \cdot \Delta X_0$$

avec $\eta_k = \eta\big(\widehat{\phi_k(X_{k-1})}\big)$ et $\otimes$ le produit terme-à-terme entre deux vecteurs.

Méthode d'évaluation de ΔY

Étape n° 1 Identifier les opérations élémentaires $\phi^{(1)}, \ldots, \phi^{(p)}$.

Étape n° 2 Calculer les sous-algorithmes $\psi^{(1)}, \ldots, \psi^{(p)}$ et de leurs jacobiens $J_{\psi^{(1)}}, \ldots, J_{\psi^{(p)}}$

Étape n° 3 Calculer les vecteurs $\eta_1 \in D_1, \ldots, \eta_p \in D_p$ de la manière suivante :

$$\forall i \in [\![1, n_k]\!], \eta_{k,i} = \begin{cases} 0 & \text{si la } i^{\text{ème}} \text{ composante de } \phi_k(x) \text{ est égale à une composante de } x \\ \eta_{k,i} & \text{avec } |\eta_{k,i}| \leq \varepsilon_{\text{mach}} \end{cases}$$

Étape n° 4 Former $\Delta Y = \Delta X_p$ en utilisant la formule du théorème A.2.

Définition A.3 (Erreur inhérente). Soit $f : D \subset \mathbb{R}^n \to \mathbb{R}^m$ un algorithme.
Pour tout $x \in D$, on appelle **erreur inhérente de $Y = f(X)$** la quantité $\{|J_f(X) \cdot X| + |Y|\} \cdot \varepsilon_{\text{mach}}$.

Remarque. L'erreur inhérente est donc composée de :
— l'erreur $|J_f(X) \cdot X| \cdot \varepsilon_{\text{mach}}$ commise lors du calcul de f en tant qu'opération élémentaire ;
— l'erreur $|Y| \cdot \varepsilon_{\text{mach}}$ commise lors de la représentation du résultat Y.
Ces termes d'erreurs se produisent inévitablement et ce, quelque soit la décomposition de f utilisée : ils sont donc *inhérents* au calcul.

Définition A.4 (Algorithme numériquement stable).
Un algorithme $f : D \subset \mathbb{R}^n \to \mathbb{R}^m$ est dit **numériquement stable** si et seulement si pour tout $X \in D$, l'erreur inhérente de $Y = f(X)$ et l'erreur absolue de calcul $\Delta^C Y$ sont de même ordre.

IMPORTANT. L'objectif de l'Analyse Numérique est d'élaborer des algorithmes numériquement stables.

A.3 Comparaison d'algorithmes

Définition A.5 (Crédibilité d'un algorithme).
Soit A un algorithme de p opérations élémentaires $\phi_1, \ldots, \phi_p$ d'opération $f : D \subset \mathbb{R}^n \to \mathbb{R}^m$.
Soit A' un algorithme de q opérations élémentaires $\phi'_1, \ldots, \phi'_q$ de même opération f.
On dit que **A est plus crédible que A'** si pour tout $X \in D$ l'erreur maximale de calcul $Y = f(X)$ est plus faible en utilisant A qu'en utilisant A'.

Remarque. Dans l'expression d'erreur du théorème A.2 page précédente, le terme $J_f(X_0) \cdot \Delta X_0$ ne dépend pas de l'algorithme choisi : il est donc inutile de l'évaluer lors de la comparaison entre deux algorithmes.

Analyse de propagation d'erreurs : Exercices

Exercice A.1. *On considère l'opération $c = a^2 - b^2$.*

1. *Proposer deux algorithmes distincts pour le calcul de c.*
2. *Calculer Δc pour les deux algorithmes.*
3. *Un des deux algorithmes est-il plus crédible que l'autre ?*
4. *Ces algorithmes sont-ils numériquement stables ?*

Annexe B

Matrices creuses

B.1 Définition

Définition B.1 (Matrice creuse. Vecteur creux).
Une **matrice creuse** est une matrice dont le nombre d'éléments non nuls est faible devant le nombre d'éléments totaux.

Un **vecteur creux** est un vecteur dont le nombre d'éléments non nuls est faible devant le nombre d'éléments totaux.

Exemple. Soit la matrice carrée d'ordre n $A = \begin{pmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & -1 \\ 0 & \cdots & 0 & -1 & 2 \end{pmatrix}$.

A comporte $3n-2$ éléments non nuls sur n^2 éléments au total. Donc cette matrice est creuse pour n suffisamment grand :

n	Taux d'élements non nuls dans A
10	28 %
100	2,98 %
1000	0,3 %
10000	0,03 %

Remarque. Cette matrice est un opérateur usuel qui apparaît régulièrement lors de la résolution d'une équation aux dérivées partielles par différences finies.

Définition B.2 (Largeur de bande).
Soit n un entier naturel non nul et A une matrice carrée d'ordre n.
La **largeur de bande inférieure de A** est le plus petit entier p tel que

$$\forall i \in [\![1,n]\!], \forall j \in [\![1, i-1-p]\!], a_{ij} = 0$$

La **largeur de bande supérieure de A** est le plus petit entier q tel que

$$\forall j \in [\![1,n]\!], \forall i \in [\![1, j-1-q]\!], a_{ij} = 0$$

Définition B.3 (Matrice tridiagonale). Une **matrice tridiagonale** est une matrice dont les largeurs de bande inférieure et supérieure sont toutes deux égales à un.

Exemple. $A = \begin{pmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & -1 \\ 0 & \cdots & 0 & -1 & 2 \end{pmatrix}$ a des largeurs de bande inférieure et supérieure égales à 1 : elle est donc tridiagonale.

Remarque. Les matrices à faible largeurs de bande — comme les matrices tridiagonales — sont des matrices creuses lorsque n est suffisamment grand.

Proposition B.1 (Matrices particulières).
Une matrice est triangulaire inférieure si et seulement si sa largeur de bande supérieure est nulle.
Une matrice est triangulaire supérieure si et seulement si sa largeur de bande inférieure est nulle.
Une matrice est diagonale si et seulement si ses largeurs de bande inférieure et supérieure sont toutes les deux nulles.

Toute matrice symétrique a des largeurs de bande inférieure et supérieure égales.

B.2 Résolution de systèmes linéaires « en bande »

Nous pouvons tirer parti d'une structure « en bande » d'une matrice afin d'éviter les calculs inutiles, notamment en réduisant les boucles aux indices pertinents (liés aux éléments non nuls).

B.2.1 Résolution de systèmes triangulaires

On peut donc adapter l'algorithme 4 page 24 au cas d'un système triangulaire ayant une structure « en bande » pour obtenir l'algorithme 18.

fonction bandedUpperSolve(U : Matrice de réel, b : Tableau de réel, q : entier) : Tableau de réel
préconditions : U carrée d'ordre n et triangulaire supérieure
préconditions : largeur de bande supérieure de U égale à q
  $x_n \leftarrow \dfrac{b_n}{u_{nn}}$
  pour $i \leftarrow n-1$ **à** 1 **par** -1 **faire**
    $x_i \leftarrow b_i$
    pour $j \leftarrow i+1$ **à** $\min(i+q, n)$ **faire**
      $x_i \leftarrow x_i - u_{ij} \cdot x_j$
    fin pour
    $x_i \leftarrow \dfrac{x_i}{u_{ii}}$
  fin pour
  retourner x
postconditions : $Ux = b$
fin fonction

Algorithme 18: Résolution d'un système linéaire triangulaire supérieur « en bande »

Proposition B.2. Soit n un entier non nul et q un entier. Alors l'algorithme 4 page 24 nécessite $O(nq)$ opérations arithmétiques et $O(n^2)$ incrémentations de compteur.

IMPORTANT. Cet algorithme est donc particulièrement efficace si q est très faible devant n.

Remarque. De même, pour un système triangulaire inférieur de largeur de bande inférieure p, on peut définir un algorithme analogue dont la complexité sera $O(np)$.

B.2.2 Factorisation LU

Théorème B.1 (Factorisation LU « en bande »).
Soit A une matrice carrée admettant une factorisation LU : $A = LU$. Alors :
— la largeur de bande inférieure de L est égale à celle de A ;
— la largeur de bande supérieure de U est égale à celle de A.

Remarque. Ce théorème établit donc que la structure de la matrice A est rigoureusement conservée dans les termes L et U de sa factorisation LU.

ATTENTION. Ce résultat n'est pas valable dans le cas d'une factorisation LU avec pivot partiel, car la matrice permutée PA possède des largeurs de bande différentes de celles de A.

On peut constater que, si A a des largeurs de bande faibles, la permutation peut considérablement dégrader celles-ci.

Le théorème B.1 implique également que l'algorithme de factorisation « en place » peut être maintenu dans le cas où A est stockée en tant que matrice creuse.

procedure LUInPlace(A : Matrice de réel)
préconditions : A carrée d'ordre n admettant une factorisation LU : $A = LU$
préconditions : largeur de bande inférieure de A égale à p
préconditions : largeur de bande supérieure de A égale à q
  pour $j \leftarrow 1$ **à** $n-1$ **faire**
    pour $i \leftarrow j+1$ **à** $\min(j+p, n)$ **faire**
      $a_{ij} \leftarrow \dfrac{a_{ij}}{a_{jj}}$
    fin pour
    pour $i \leftarrow j+1$ **à** $\min(j+q, n)$ **faire**
      pour $k \leftarrow j+1$ **à** $\min(j+p, n)$ **faire**
        $a_{ik} \leftarrow a_{ik} - a_{ij}a_{jk}$
      fin pour
    fin pour
  fin pour
postconditions : $A = (L - I_n) + U$
fin procedure

Algorithme 19: Calcul de factorisation LU (DOOLITTLE) « en place » pour une matrice à structure « en bande »

Proposition B.3 (Complexité cubique d'un calcul de factorisation LU « en place »).
Soit n un entier naturel non nul et p et q deux entiers. Alors l'algorithme 19 nécessite $O(npq)$ opérations arithmétiques et $O(npq)$ incrémentations de compteur.

B.2.3 Réduction de la largeur de bande

Une faible largeur de bande permettant d'améliorer les performances des méthodes de résolutions usuelles, il est pertinent d'essayer de permuter les variables d'un système linéaire afin de réduire les largeurs de bande de la matrice associée.

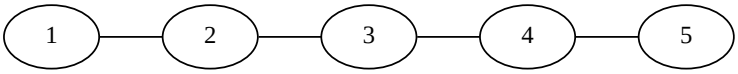
Nous présenterons un algorithme pour les matrices symétriques.

Définition B.4 (Graphe associé à une matrice symétrique). Soit n un entier naturel nn nul et A une matrice carrée d'ordre n symétrique.

Le **graphe associé à la matrice A** est le graphe simple non valué non orienté $G = (\llbracket 1, n \rrbracket, E)$ avec

$$i, j \in E \iff (i \neq j) \text{ et } a_{ij} \neq 0$$

Exemple. Le graphe associé à $A = \begin{pmatrix} 2 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{pmatrix}$ est le graphe suivant :



Remarque. Le graphe ne dépend pas explicitement des valeurs de A, mais seulement de si elles sont nulles ou pas.

Remarquons également que la réduction de largeur de bande par permutation des variables ne dépend que du placement des éléments non nuls et non de leurs valeurs. Le graphe associé à la matrice est donc le type de données idéal pour une telle opération : c'est ce que propose l'algorithme 20, dit de CUTHILL-MCKEE.

Entrées : $G = (X, E)$: graphe associé à une matrice carrée d'ordre n symétrique
Sorties : R : liste de sommets de G traduisant le nouvel ordre des variables
s est choisi dans argmin$\{\mathsf{deg}(G, x), x \in X\}$ /* *s est un sommet de degré minimal* */
$R \leftarrow \mathsf{Liste}(x)$
$i \leftarrow 1$
tant que $\mathsf{length}(R) < n$ **faire**
  $A \leftarrow \mathsf{adjacents}(G, R(i)) \backslash R$
  A est trié selon les degrés croissants.
  $R \leftarrow R :: A$ /* *Concaténation de listes* */
  $i \leftarrow i + 1$
fin tant que

Algorithme 20: Algorithme de CUTHILL-MCKEE

Remarque. Il existe également *l'algorithme de* CUTHILL-MCKEE *inversé* qui consiste à prendre la liste-miroir de R plutôt que R. Il semble empiriquement plus efficace.

B.3 Stockage

Nous présentons ici plusieurs implémentations du type matrice creuse.

B.3.1 Implémentations incrémentales

De telles implémentations permettant une mise à jour aisée des éléments d'une matrice, notamment lors d'une construction incrémentale.

B.3.1.1 Dictionnaire

On implémente alors une matrice creuse comme un tableau associatif dont les clés sont des couples (ligne, colonne). Une clé non présente correspond à un élement nul.

Type sparseMatrix = Map<(entier,entier)> de réel

Algorithme 21: Implémentation par dictionnaire

Exemple. La matrice $A = \begin{pmatrix} 2 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{pmatrix}$ est représentée par le dictionnaire suivant :

Clé	Valeur	Clé	Valeur	Clé	Valeur	Clé	Valeur	Clé	Valeur
$(1,1)$	2	$(4,4)$	2	$(3,2)$	-1	$(1,2)$	-1	$(4,5)$	-1
$(2,2)$	2	$(5,5)$	2	$(4,3)$	-1	$(2,3)$	-1		
$(3,3)$	2	$(2,1)$	-1	$(5,4)$	-1	$(3,4)$	-1		

B.3.1.2 Tableau de listes

On implémente alors une matrice creuse comme un tableau de listes. Chacune de ces listes contient les indices des éléments non nuls avec leurs valeurs.

Remarque. Une liste vide traduit donc une ligne entière de zéros.
Dans la pratique, on cherche à maintenir chaque liste triée par indice afin de faciliter la lecture.

Type sparseMatrix = Tableau de (Liste de (entier,réel))

Algorithme 22: Implémentation par tableau de listes

Exemple. La matrice $A = \begin{pmatrix} 2 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{pmatrix}$ est représentée par le tableau `t` de listes suivant :

$\texttt{t}(1) = [(1,2),(2,-1)]$

$\texttt{t}(2) = [(1,-1),(2,2),(3,-1)]$

$\texttt{t}(3) = [(2,-1),(3,2),(4,-1)]$

$\texttt{t}(4) = [(3,-1),(4,2),(5,-1)]$

$\texttt{t}(5) = [(3,-1),(4,2)]$

B.3.1.3 Liste de coordonnées

On implémente alors une matrice creuse comme une liste de triplets (ligne, colonne, valeur).

Remarque. Dans la pratique, comme dans le cas précédent, on cherche à maintenir la liste triée de manière lexicographique selon le couple (ligne,colonne).

Type sparseMatrix = Liste de (entier,entier,réel)

Algorithme 23: Implémentation par liste de coordonnées

Exemple. La matrice $A = \begin{pmatrix} 2 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{pmatrix}$ est représentée par la liste suivante :

$$[(1,1,2),(1,2,-1),(2,1,-1),(2,2,2),(2,3,-1),(3,2,-1),(3,3,2),(3,4,-1),$$
$$(4,3,-1),(4,4,2),(4,5,-1),(5,3,-1),(5,4,2)]$$

B.3.2 Implémentations avancées

De telles implémentations sont plus adaptées aux opérations matricielles, en particulier les produits matrice-vecteur et matrice-matrice.

B.3.2.1 Stockage par compression de colonnes

Remarque. Dans la littérature, cette implémentation est appelée « *Compressed Column Storage* » ou « *Compressed Sparse Column* ».

On implémente alors une matrice en utilisant la structure de données de l'algorithme 24 :
- *nombreLignes* est le nombre de lignes de la matrice ;
- *nombreColonnes* est le nombre de colonnes de la matrice ;
- *valeurs* est la liste des valeurs non nulles de la matrice, lues colonne par colonne
- pour tout indice i, *lignes*(i) est le numéro de ligne de *valeurs*(i)
- pour tout indice j, *valeurs*(*debutsColonne*(j)) est le premier élément non nul de la $j^{\text{ème}}$ colonne.

type sparseMatrix = **structure**
  nombreLignes : entier
  nombreColonnes : entier
  valeurs : Liste de entier
  lignes : Liste de entier
  debutsColonne : Liste de entier
fin type

Algorithme 24: Implémentation par compression de colonnes

Exemple. La matrice $A = \begin{pmatrix} 2 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{pmatrix}$ est représenté par la structure s suivante :
- s.*nombreLignes* = 5
- s.*nombreColonnes* = 5
- s.*valeurs* = [2,−1,−1,2,−1,−1,2,−1,−1,2,−1,−1,2]
- s.*lignes* = [1,2,1,2,3,2,3,4,3,4,5,4,5]
- s.*debutsColonne* = [1,3,6,9,12,14]

B.3.2.2 Stockage par compression de lignes

Remarque. Ici, les termes utilisés sont « *Compressed Row Storage* » ou « *Compressed Sparse Row* ».

Le principe est le même que précédemment, sauf que la lecture est faite ligne par ligne et non colonne par colonne, ce qui donne la structure de données de l'algorithme 25.

type sparseMatrix = **structure**
  nombreLignes : entier
  nombreColonnes : entier
  valeurs : Liste de entier
  colonnes : Liste de entier
  debutsLigne : Liste de entier
fin type

Algorithme 25: Implémentation par compression de lignes

Exemple. La matrice $A = \begin{pmatrix} 2 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{pmatrix}$ est représenté par la structure s suivante :

— s.*nombreLignes* = 5
— s.*nombreColonnes* = 5
— s.*valeurs* = [2,−1,−1,2,−1,−1,2,−1,−1,2,−1,−1,2]
— s.*colonnes* = [1,2,1,2,3,2,3,4,3,4,5,4,5]
— s.*debutsLigne* = [1,3,6,9,12,14]

Remarque. On obtient une structure identique à celle obtenu par compression de colonnes car la matrice est symétrique.

Matrices creuses : Exercices

Exercice B.1. *Soit* $A = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}$.

1. *Quelle est la largeur de bande de A ?*
2. *Quelle est la nouvelle la largeur de bande :*
   (a) *après application de l'algorithme de* CUTHILL-MCKEE *?*
   (b) *après application de l'algorithme de* CUTHILL-MCKEE *inversé ?*

Annexe C

Décomposition en valeurs propres

C.1 Localisation des valeurs propres

C.2 Techniques de calcul des valeurs propres

Bibliographie

[1] Chrysostome BASKIOTIS et Laurence LAMOULIE : Analyse numérique. Fascicules de cours EISTI, 2013-2014.

[2] Anna DÉSILLES : Méthodes numériques pour les grandes matrices creuses. Notes de cours EISTI, 2003-2004.

[3] Gene H. GOLUB et Charles F. VAN LOAN : *Matrix Computations*. Series in Mathematical Sciences. Johns Hopkins University Press, troisième édition, 1996.

[4] Nicholas J. HIGHAM : *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, seconde édition, 2002.

[5] Jorge NOCEDAL et Stephen J. WRIGHT : *Numerical Optimization*. Operations Research. Springer-Verlag, 1999.

[6] Josef STOER et Roland BULIRSCH : *Introduction to Numerical Analysis*, volume 12 de *Texts in Applied Mathematics*. Springer, troisième édition, 2002.

[7] WIKIPEDIA : Sparse matrix, 2015.