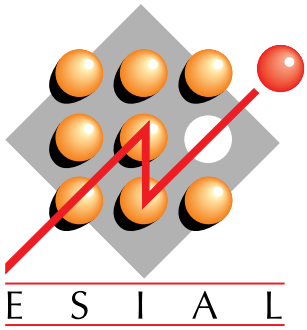

Mathématiques Numériques et Analyse de Données

ESIAL 1ère année

Notes de cours rédigées par B. Pinçon et J.-F. Scheid

2007-2008

Table des matières

Chapitre 1

Arithmétique flottante

1.1 Représentation d'un nombre en virgule flottante

1.1.1 Introduction

Cette représentation est naturelle lorsque l'on veut écrire un nombre très grand ou très petit en valeur absolue, par exemple :

$A = 6,02252\ 10^{23}$ le nombre d'Avogadro
$h = 6,625\ 10^{-34}$ $[joule\ seconde]$ la constante de Planck ($\hbar = h/2\pi$).

Cette écriture d'un nombre (parfois appelée "notation scientifique") comporte deux parties :
- la mantisse (ici $6,02252$ et $6,625$) ;
- la partie exposant (23 et -34, le 10 est ici obligatoire car c'est la base choisie pour écrire la mantisse et l'exposant).

On dit que la représentation est *normalisée* quand la mantisse est de la forme[1] :

$$c_0, c_1c_2c_2c_3 \dots \text{ avec } c_0 \neq 0$$

Remarques :

1. Tout nombre réel (sauf zéro) peut s'écrire avec cette notation en virgule flottante normalisée avec en général un nombre de chiffres infini pour la mantisse (résultat provenant de la densité de $\mathbb{Q}$ dans $\mathbb{R}$) ; on peut aussi remarquer que certains nombres peuvent s'écrire de deux façons, par exemple $0,999\cdots = 1$.
2. Un changement de base peut introduire quelques bizarreries dans l'écriture d'un nombre alors que celle-ci est anodine dans la base initiale, par exemple :

$$(0,2)_{10} = (0,\ 0011\ 0011\ 0011\ \dots)_2.$$

1.1.2 Définition formelle d'un système fini de représentation en virgule flottante

Dans un ordinateur on est obligé de restreindre le nombre de chiffres pour les mantisses et de limiter l'étendue des exposants. En choissisant ces limites, on définit un ensemble fini de nombres $\mathbb{F}(\beta, p, e_{min}, e_{max})$ où :
- β est l'entier ($\beta \geq 2$) définissant la base ;
- p est le nombre de chiffres de la mantisse ;
- e_{min} est l'exposant minimum et e_{max} l'exposant maximum.

Cet ensemble correspond à tous les nombres réels x qui s'écrivent :

$$x = s\left(\sum_{i=0}^{p-1} c_i\beta^{-i}\right)\beta^e, \text{ où } \begin{cases} s = \pm 1 \text{ le signe} \\ 0 \leq c_i \leq \beta - 1, \forall i \\ e_{min} \leq e \leq e_{max} \end{cases}$$

[1] une autre définition souvent rencontrée est $0, c_1c_2c_2c_3 \dots$ avec $c_1 \neq 0$

la représentation étant normalisée si $c_0 \neq 0$. Pour représenter 0 on utilise une écriture spéciale en ne mettant que des 0 dans la mantisse (ce qui est logique) et un exposant de $e_{min} - 1$[2].

Exemple : le sous-ensemble des nombres normalisés positifs (plus zéro) de $\mathbb{F}(2, 3, -1, 2)$ (cf figure 1.1).

0		
$1,00\ 2^{-1}$	$=$	$(0,5)_{10}$
$1,01\ 2^{-1}$	$=$	$(0,625)_{10}$
$1,10\ 2^{-1}$	$=$	$(0,750)_{10}$
$1,11\ 2^{-1}$	$=$	$(0.875)_{10}$
$1,00\ 2^{0}$	$=$	1
$1,01\ 2^{0}$	$=$	$(1,25)_{10}$
$\vdots$	$\vdots$	$\vdots$
$1,11\ 2^{2}$	$=$	$(7)_{10}$

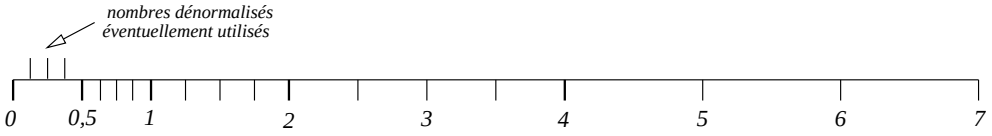

FIG. 1.1 – Les nombres normalisés positifs (plus zéro) de $\mathbb{F}(2, 3, -1, 2)$

Remarques et notations :

- Dans les intervalles $[\beta^e, \beta^{e+1}]$ et $[-\beta^{e+1}, -\beta^e]$ l'incrément entre deux nombres flottants est constant et égal à β^{1-p+e}.
- Pendant assez longtemps on a utilisé uniquement les nombres normalisés (plus zéro) de ces ensembles. Cependant, comme la figure le montre, il en résulte un "vide" entre le plus petit nombre normalisé et le zéro. L'utilisation des nombres dénormalisés non redondants (cf figure 1.1) permet "d'aller vers zéro" plus graduellement.
- On notera M le plus grand nombre positif de $\mathbb{F}(\beta, p, e_{min}, e_{max})$:

$$M = \left(\sum_{i=0}^{p-1} (\beta - 1)\beta^{-i} \right) \beta^{e_{max}} = (1 - \beta^{-p})\beta^{e_{max}+1},$$

  m le plus petit nombre normalisé positif :

$$m = (1, 00..0)\beta^{e_{min}} = \beta^{e_{min}},$$

  et μ le plus petit nombre dénormalisé (> 0) :

$$\mu = (0, 00..1)\beta^{e_{min}} = \beta^{1-p+e_{min}}.$$

- On notera $\oplus$, $\ominus$, $\otimes$, $\oslash$ les opérations d'addition, de soustraction, de multiplication et de division, effectuées par l'ordinateur.
- Dans les systèmes flottants actuels on rajoute des nombres spéciaux comme $+inf$, $-inf$ (inf comme infini) et NaN (pour Not a Number) qui ont une représentation spéciale (utilisant en particulier un exposant de $e_{max} + 1$). Enfin, on notera que, du fait du bit de signe, le zéro a deux représentations[3] qui conduisent à des résultats différents sur quelques calculs (par exemple $1 \oslash +0$ donnera $+inf$ alors que $1 \oslash -0$ donnera $-inf$).

[2] ce qui semble naturel si on pense à l'algorithme de comparaison de deux nombres flottants normalisés

[3] que l'on notera $+0$ et -0 : si mathématiquement c'est le même nombre, informatiquement non !

1.1.3 Approximation d'un nombre réel par un nombre flottant

Étant donné $x \in \mathbb{R}$, en général $x \notin \mathbb{F}(\beta, p, e_{min}, e_{max})$ et il faut associer à x une approximation $fl(x) \in \mathbb{F}(\beta, p, e_{min}, e_{max})$:

1. Lorsque $|x| > M$ ce n'est *a priori* pas possible (on parle souvent d' "overflow") mais dans les systèmes flottants actuels, on associe à x un nombre spécial :
$$\begin{cases} fl(x) = +inf & \text{si } x > M \\ fl(x) = -inf & \text{si } x < -M \end{cases}$$
et les calculs peuvent continuer...
2. Lorsque $x \neq 0$ est tel que $|x| < m$ et que seuls les nombres normalisés sont utilisés (plus zéro) alors $fl(x) = \pm 0$, selon le signe de x (on parle alors d' "underflow"). Si les nombres dénormalisés sont utilisés, la troncature vers zéro a lieu uniquement pour $|x| < \mu$ et pour $\mu \leq |x| < m$ on procède comme dans la suite, c-a-d que $fl(x)$ est le nombre flottant le plus proche de x.
3. Lorsque $m \leq |x| \leq M$, l'approximation $fl(x)$ est le nombre flottant le plus proche de x, c-a-d que si :
$$x = s \left(\sum_{i=0}^{+\infty} x_i \beta^{-i} \right) \beta^e$$
alors :
   (a) si $x_p < \beta/2$ (rmq : on choisit toujours des bases paires), on arrondit "en dessous" :
$$fl(x) = s \left(\sum_{i=0}^{p-1} x_i \beta^{-i} \right) \beta^e \tag{1.1}$$
   (b) si $x_p \geq \beta/2$ avec au moins un indice $j > p$ tel que $x_j \neq 0$, on arrondit "au dessus" :
$$fl(x) = s \left(\sum_{i=0}^{p-2} x_i \beta^{-i} + (x_{p-1} + 1)\beta^{-(p-1)} \right) \beta^e \tag{1.2}$$
   (c) si $x_p = \beta/2$ avec $x_j = 0$, $\forall j > p$ (x est à égale distance entre 2 nombres flottants), il existe principalement deux façons d'arrondir :
      i. $fl(x)$ est donné par (1.2)
      ii. on arrondit de sorte que le dernier chiffre de la représentation soit pair, c-a-d que :
         A. si x_{p-1} est pair alors $fl(x)$ est donné par (1.1),
         B. si x_{p-1} est impair alors $fl(x)$ est donné par (1.2).

   Cette dernière méthode plus compliquée (mode arrondi prescrit par la norme IEEE 754) permet de stabiliser certains calculs (voir un exemple en annexe).

Remarque : dans le temps, certains ordinateurs effectuaient simplement une troncature (pas d'arrondi), c-a-d que $fl(x)$ était donné par (1.1) dans tous les cas. Je ne pense pas qu'il existe encore des ordinateurs de ce type.

Exemples : on va travailler avec l'ensemble de flottant $\mathbb{F}(10, 8, -38, 37)$ qui a l'avantage d'être en base 10 et qui est assez proche du jeu de flottants dénommé *simple précision* disponible sur la plupart des machines ($\mathbb{F}(2, 24, -126, 127)$). Les nombres caractéristiques de cet ensemble sont : $M = 9,9999999\ 10^{37}$, $m = 10^{-38}$ et $\mu = 10^{-45}$.

$fl(10^{38})$	=	$+inf$
$fl(1,2345678\ 10^{-41})$	=	$\begin{cases} +0 \text{ si on n'utilise pas les nombres dénormalisés} \\ 0,0012346\ 10^{-38} \text{ si on les utilise} \end{cases}$
$fl(\underline{9,9999999})$	=	$9,9999999$
$fl(\underline{9,99999999})$	=	10
$fl(\underline{1,55555555})$	=	$1,5555556$ avec les deux modes d'arrondi
$fl(\underline{1,00000005})$	=	$\begin{cases} 1,0000001 \text{ avec le mode arrondi classique} \\ 1,0000000 \text{ avec le mode arrondi IEEE} \end{cases}$

1.1.4 Erreur entre un réel et son représentant flottant : le epsilon machine

Soit donc $x \in \mathbb{R}$ tel que $|x| \in [m, M]$. Si $|x| \in [\beta^e, \beta^{e+1}]$ son représentant $fl(x)$ dans $\mathbb{F}(\beta, p, e_{min}, e_{max})$ s'écrira :

$$fl(x) = \pm \overbrace{c_o \ , \ c_1 \ \ldots \ c_{p-1}}^{p \text{ chiffres}} \times \beta^e$$

ou encore :

$$fl(x) = \pm \overbrace{1 \ , \ 0 \ \ldots \ 0}^{p \text{ chiffres}} \times \beta^{e+1}$$

si $|x|$ est suffisamment proche de β^{e+1}. D'après la définition de la fonction fl, l'erreur entre x et $fl(x)$, sera bornée par :

$$ea = |x - fl(x)| \leq \overbrace{0 \ , \ 0 \ \ldots \ 0}^{p \text{ chiffres}} \frac{\beta}{2} \ \times \ \beta^e$$

$$ea = |x - fl(x)| \leq \frac{1}{2}\beta^{1-p+e}$$

on dit souvent que $ea \leq \frac{1}{2}ulp$ (ulp pour unit in the last place). Cette majoration est constante pour tout les $|x| \in [\beta^e, \beta^{e+1}]$ et l'erreur relative commise peut être majorée en divisant ea par le plus petit nombre de cet intervalle soit β^e :

$$er = \frac{|x - fl(x)|}{|x|} \leq \frac{ea}{\beta^e} = \frac{1}{2}\beta^{1-p}$$

Cette majoration[4] indépendante de e est un nombre caractéristique de l'ensemble $\mathbb{F}(\beta, p, e_{min}, e_{max})$ qui porte le nom de "epsilon machine" :

Définition : on appelle epsilon machine, la quantité $\epsilon_m = \frac{1}{2}\beta^{1-p}$.
Remarques :

- Lorsque $|x| \in [\beta^e, \beta^{e+1}]$ est proche de β^{e+1} l'erreur relative maximale est en fait proche de $\frac{1}{2}\beta^{-p}$ (cf figure 1.2 qui donne la forme exacte de l'erreur absolue (en traits pointillés) et de l'erreur relative).
- Si la machine utilise des nombres dénormalisés et que $|x| \in [\mu, m[$ alors on peut simplement dire que $|fl(x) - x| \leq \frac{1}{2}\mu$, mais l'erreur relative n'est plus bornée par ϵ_m : l'erreur relative maximum passe de ϵ_m au voisinage de m à $\frac{1}{2}$ au voisinage de μ !

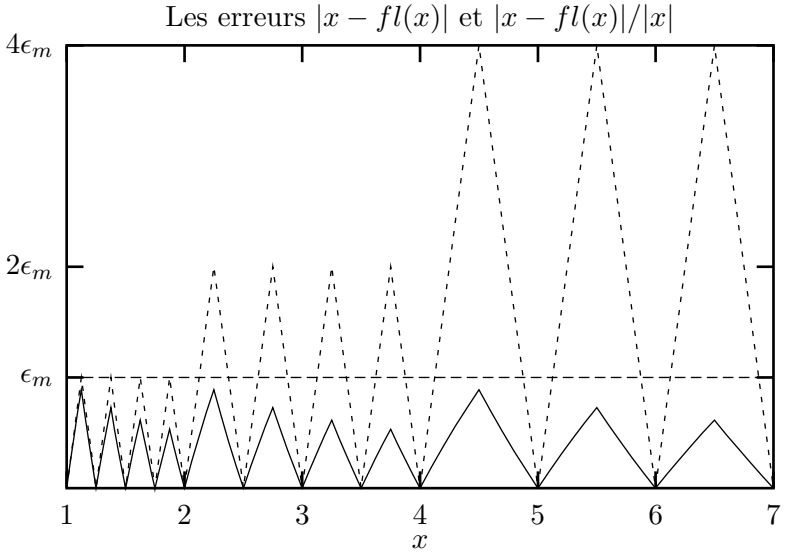

FIG. 1.2 – Erreurs pour l'approximation d'un réel dans $\mathbb{F}(2, 3, -1, 2)$

[4]en fait l'erreur relative maximum est égale à $\epsilon_m/(1 + \epsilon_m)$, (cf figure 1.2) valeur très proche de ϵ_m dans les cas qui nous intéressent !

Corollaire : Si $|x| \in [m, M]$ alors on a :

$$fl(x) = x(1+\epsilon), \quad \text{avec } |\epsilon| \leq \epsilon_m,$$

ce qui est une autre façon pour exprimer l'erreur relative. C'est cette relation qui est principalement utilisée pour les analyses d'erreur.

1.2 La norme IEEE 754

1.2.1 Introduction

Cette norme a apporté une grande amélioration pour les calculs avec les nombres flottants[5]. Elle spécifie en particulier :

1. les deux ensembles de flottants :
   (a) $\mathbb{F}(2, 24, -126, 127)$ dont le codage tient sur 4 octets,
   (b) $\mathbb{F}(2, 53, -1022, 1023)$ tenant sur 8 octets,
2. l'utilisation des nombres spéciaux $+inf$, $-inf$, NaN,
3. l'utilisation du mode arrondi sophistiqué vu précédemment[6],
4. l'utilisation des nombres dénormalisés (cette obligation est la moins respectée par les F.P.U.),
5. et enfin le fait que les 4 opérations de base[7] doivent être réalisées avec la précision maximum possible ; c-a-d que si $\cdot$ est l'une des 4 opérations et $\odot$ l'opération machine correspondante, et si u et v sont deux nombres flottants tels que $|u \cdot v| \in [m, M]$ alors on doit obligatoirement obtenir :

$$u \odot v = fl(u \cdot v) \tag{1.3}$$

Remarques :

1. Cette dernière condition est réalisable si les calculs intermédiaires (pour réaliser l'opération en question) sont menés avec deux chiffres et un bit "d'information" supplémentaires (l'un des exercices du TD montre déjà l'apport d'un chiffre supplémentaire pour les soustractions).
2. Le fait de choisir la base 2 (quelques anciens "gros ordinateurs", comme les IBM 370 utilisaient plutôt la base 16) permet d'obtenir (pour la même quantité de mémoire pour la mantisse) un epsilon machine plus petit : avec 24 chiffres binaires $\epsilon_m \simeq 6\ 10^{-8}$ alors qu' avec 6 chiffres hexadécimaux $\epsilon_m \simeq 4.8\ 10^{-7}$.

1.2.2 Valeurs caractéristiques des flottants IEEE

Les voici résumées dans un tableau (il s'agit de valeurs approchées) :

ensembles de flottants	$\mathbb{F}(2, 24, -126, 127)$	$\mathbb{F}(2, 53, -1022, 1023)$
ϵ_m	$5.9604645\ 10^{-8}$	$1.1102230246251565\ 10^{-16}$
m	$1.1754944\ 10^{-38}$	$2.2250738585072014\ 10^{-308}$
M	$3.4028235\ 10^{38}$	$1.7976931348623157\ 10^{308}$

[5] Les premières machines dont la F.P.U. (Floating Point Unit, souvent appelé coprocesseur mathématique chez nous) respectait (au moins en partie...) cette norme, furent les fameuses "stations de travail" (Sun, HP,...) ainsi que les PC (bug du pentium mis à part) ; lorsqu'elles sont arrivées en masse dans l'industrie, les laboratoires et les centres de calcul, les ingénieurs ont refait tourner un certain nombre de programmes dont les résultats comportaient, parait-il, des différences significatives par rapport à ceux effectués avec les gros ordinateurs de l'époque (CRAY, IBM, CONTROL DATA, etc...) ! Seuls, les VAX de chez DIGITAL (que l'on classait comme mini-ordinateurs) intégraient déjà une excellente F.P.U. capable d'effectuer en particulier, des calculs en quadruple précision (flottants sur 16 octets).

[6] La norme impose aussi que les autres modes d'arrondi puissent être activés, ce qui peut être intéressant pour certaines applications.

[7] ainsi que l'extraction de racines carrées

En langage C, les déclarations respectives `float x;` et `double y;` correspondent la plupart du temps[8] à $x \in \mathbb{F}(2, 24, -126, 127)$ (souvent appelé simple précision) et $y \in \mathbb{F}(2, 53, -1022, 1023)$ (double précision). Scilab quant à lui, utilise uniquement des nombres en double précision.

1.2.3 Remarques sur le codage des flottants IEEE

Si on examine les prérequis de mémoire pour coder un nombre de $\mathbb{F}(2, 24, -126, 127)$, on compte donc :
- 1 bit pour le signe,
- 24 bits pour la mantisse,
- et 8 bits pour l'exposant (qui permettent de coder 256 entiers, par exemple ceux de l'intervalle [-127, +128],

ce qui nous fait 33 bits et donc un de trop ! L'astuce est la suivante : comme on travaille essentiellement avec des nombres normalisés pour lesquels le premier chiffre c_0 est toujours égal à 1 (du fait que $\beta = 2$), on peut donc *a priori* s'en passer (c_0 est appelé le "bit caché" (hidden bit)). Ce truc semble empêcher l'utilisation des nombres dénormalisés : comment les circuits peuvent-ils reconnaître un tel nombre ? L'astuce consiste alors à utiliser l'exposant associé au zéro $e_{min} - 1 = -127$[9]. De même le codage des $+inf$, $-inf$ et NaN utilise l'exposant $e_{max} + 1$. Voici un tableau récapitulatif, dans lequel f désigne $c_1\ c_2\ \ldots\ c_{p-1}$:

exposant	f	nombre flottant associé
$e = e_{min} - 1$	$f = 0$	± 0
$e = e_{min} - 1$	$f \neq 0$	$\pm 0, f \times 2^{e_{min}}$
$e_{min} \leq e \leq e_{max}$	$f \neq 0$	$\pm 1, f \times 2^{e}$
$e = e_{max} + 1$	$f = 0$	$\pm inf$
$e = e_{max} + 1$	$f \neq 0$	NaN

Enfin, le codage de l'exposant est obtenu en rajoutant un biais de 127 pour les "floats" $e_{code} = e + 127$) et de 1023 pour les "doubles" ($e_{code} = e + 1023$).

1.2.4 L'arithmétique avec les nombres spéciaux

Lorsque les calculs (ou simplement l'introduction de données) conduisent à un dépassement de capacité, c-a-d quand x (la donnée ou le résultat d'un calcul) est tel que $|x| > M$, l'ordinateur continue les calculs avec les nombres flottants spéciaux $\pm inf$. Voici quelques règles de calculs (les autres sont évidentes ou se déduisent avec des changements de signe) :

avec x tel que	opération	résultat
$x \neq -inf$ et $x \neq NaN$	$+inf \oplus x$	$+inf$
$x \neq 0$ et $x \neq NaN$	$+inf \otimes x$	$sgn(x)inf$
$x \neq \pm inf$ et $x \neq NaN$	$+inf \oslash x$	$sgn(x)inf$
$x \neq \pm inf$ et $x \neq NaN$	$x \oslash +inf$	$sgn(x)0$

Ces règles sont naturelles puisqu'elles sont la copie des règles mathématiques[10] en remplaçant $\pm inf$ par $\pm\infty$). Pour gérer les cas d'indétermination $\pm inf \oslash \pm inf$, $\pm inf \otimes \pm 0$, $\pm 0 \oslash \pm 0$, ..., la norme a proposé l'utilisation du NaN. On obtient aussi un NaN lorsque l'on essaie d'extraire la racine carrée d'un nombre strictement négatif ou de calculer le logarithme d'un nombre négatif. Bref lorsqu'apparait un NaN il a une très grande tendance à se reproduire puisque tout calcul comportant un tel nombre donne aussi un NaN.

[8] L'examen du fichier `limits.h` (qui se trouve généralement dans `/usr/include` ou dans `/usr/local/include`) permet de savoir à quoi correspondent plus précisemment `float` et `double` et de récupérer en particulier les valeurs m et M avec les constantes nommées `FLT_MIN, DBL_MIN, FLT_MAX, DBL_MAX`.

[9] Si la séquence de bits $c_1\ c_2\ \ldots\ c_{23}$ est nulle il s'agit de zéro, sinon d'un nombre dénormalisé et il faut alors rajouter 1 à l'exposant (cf tableau).

[10] sauf la troisième avec $x = 0$

Le fonctionnement habituel d'un programme est de continuer quoi qu'il arrive. En général, on peut, à l'aide de certains mécanismes[11] savoir si une condition "d'exception" sur les calculs flottants est intervenue. Les conditions d'exception définies par l'IEEE sont les suivantes :

overflow : dépassement de capacité (non du à une division par zéro),

underflow : troncature vers zéro ou apparition d'un nombre dénormalisé,

division par zéro ,

opération invalide : une opération produisant un NaN,

opération inexacte : c'est la plus bénigne puisqu'elle apparait lorsque $fl(x) \neq x$.

Par exemple avec Scilab, la fonction `ieee(mode)` permet théoriquement[12] de régler le comportement vis à vis de ces exceptions :

- avec `mode=0` une exception génère une erreur (le script ou la fonction en cours s'arrête alors brutalement) ;
- avec `mode=1` une exception est signalée par un message mais les calculs continuent ;
- avec `mode=2` les exceptions ne sont pas signalées, les calculs continuent.

Avec l'instruction `mode = ieee()`, on récupère le comportement courant.

Certains compilateurs permettent aux programmes (sans avoir à écrire une ligne de code supplémentaire) d'afficher un message à la fin de leur déroulement prévenant de la survenue de ces exceptions[13]. L'intérêt principal est de savoir si une condition *d'overflow* est arrivée : en effet les résultats obtenus en fin de calcul ne feront pas forcément apparaître des $\pm inf$ et donc sembleront normaux, alors que l'erreur relative sur ces résultats sera généralement grande (un cas élémentaire : $x = M \oslash (M \oplus 1.0)$ donne $x = +0$ alors que la valeur exacte est voisine de 1). De même si une condition *d'underflow* est intervenue on peut aussi avoir de gros doutes sur la précision des résultats[14].

1.3 Quelques conséquences sur les calculs

1.3.1 Introduction

Si l'arithmétique d'un ordinateur est conforme à la norme IEEE alors on déduit de (1.3) et de l'étude sur fl que si u et v sont deux nombres flottants tels que $|u \cdot v| \in [m, M]$ alors :

$$u \odot v = fl(u \cdot v) = (u \cdot v)(1 + \epsilon) \text{ avec } |\epsilon| \leq \epsilon_m. \tag{1.4}$$

D'une manière générale, les ordinateurs ne respectant pas la norme, assurent quand même l'inégalité précédente avec $|\epsilon| \leq \alpha\epsilon_m$ où α est un petit entier (2, 3, ...qui peut varier selon le type de l'opération).

Avant de passer à des considérations générales, regardons les exemples suivants qui utilisent l'ensemble de flottants $\mathbb{F}(10, 8, -38, 37))$. Pour le moment on ne regardera pas les problèmes dus aux "*overflow* et *underflow*" (les calculs suivants seraient identiques dans $\mathbb{F}(10, 8, -\infty, +\infty))$:

1. L'addition n'est plus associative : dans un système flottant, il existe de nombreux cas pour lesquels :

$$(u \oplus v) \oplus w \neq u \oplus (v \oplus w)$$

par exemple :

$$(11111113 \oplus -11111111) \oplus 7,5111111 = 9,5111111$$

[11] Lorsqu'une F.P.U. détecte une condition d'exception, un bit spécial (correspondant au type de l'exception) est positionné à 1 : la lecture de ce bit permet alors la détection de l'exception, le programme devant remettre ce bit à zéro s'il veut redétecter cette même exception pour un calcul ultérieur. Par ailleurs, la norme recommande plutôt l'utilisation de "trap handlers" qui permettent une gestion des exceptions plus agréable pour le programmeur.

[12] Apparemment seuls les divisions par zéro et certaines conditions d'overflow sont prises en compte.

[13] Ceci un minimum qui semble assez simple à réaliser puisqu'il suffit d'aller lire les bits spéciaux juste à la fin du déroulement du programme, mais apparemment, peu de compilateurs le font ; à ce sujet notons que le constructeur SUN fournit avec ses compilateurs un ensemble de fonctions qui permet d'exploiter complètement la norme IEEE de ses F.P.U..

[14] Cependant si cette condition est obtenue alors que l'on cherche une convergence vers zéro c'est plutôt bon signe !

où le résultat est exact, alors que :

$$11111113 \oplus (-11111111 \oplus 7,5111111) = 10$$

où une erreur a été générée du fait que :

$$fl(-11111111 + 7,5111111) = -11111103 \neq -11111103,4888889 = -11111111 + 7,5111111.$$

Cet exemple montre que si chaque opération individuelle est menée avec une précision relative d'au moins ϵ_m, après 2 opérations successives cette précision peut être fortement dégradée (la deuxième façon de mener le calcul conduit à une erreur relative d'environ $0,05$ alors qu'ici $\epsilon_m = 5\ 10^{-8}$).

2. La multiplication n'est plus associative mais de manière nettement moins dramatique que l'addition, par exemple :

$$(1,2345678\ 10^7 \otimes 8,7654321\ 10^7) \otimes 1,3579246\ 10^7 = 1,4694808\ 10^{22}$$

alors que :

$$1,2345678\ 10^7 \otimes (8,7654321\ 10^7 \otimes 1,3579246\ 10^7) = 1,4694809\ 10^{22}$$

le résultat exact étant $\underline{1,469480852}1222713562948\ 10^{22}$. Cet exemple n'est pas un cas particulier puisque l'on verra par la suite qu'un calcul uniquement à base de multiplications et de divisions n'est pas entaché d'une erreur importante : la règle étant que si on effectue n calculs *n'engendrant ni overflow ni underflow*, alors l'erreur relative maximum est quasi bornée par $n\epsilon_m$ (et sera nettement plus petite d'un point de vue statistique).

1.3.2 La soustraction de deux nombres proches

Qu'est-ce qui pose problème dans notre premier exemple ? Et bien la principale source d'instabilité numérique avec les calculs flottants provient de la soustraction de deux nombres proches[15]. Ce qui est paradoxal c'est que la soustraction de deux flottants suffisamment proches est exacte. On peut en effet montrer la :

Proposition : En arithmétique IEEE, si u et v sont deux flottants tels que $v/2 \leq u \leq 2v$, alors :

$$u \ominus v = u - v$$

Preuve : admise. Ce résultat est valable lorsque l'on utilise les nombres dénormalisés ; dans le cas contraire l'énoncé reste vrai tant que le résultat $u - v$ est un flottant normalisé.

En fait même si la soustraction de deux nombres très proches est effectuée exactement, elle révèle, en les amplifiant, les erreurs commises sur les calculs précédents (ou plus simplement sur l'introduction de données). C'est ce qui se passe avec la deuxième méthode du premier exemple : la petite erreur commise initialement sur $-11111111 + 7,5111111$ est amplifiée par la deuxième addition. Considérons 2 quantités réelles assez proche X et Y (mais telles que $X \neq Y$), résultats théoriques que l'on obtient à partir de données sur lesquelles on effectue des calculs. On va quantifier l'écart entre ces deux nombres par rapport à leur magnitude, c-à-d que l'on suppose que $|X - Y| = r|X|$ ($\simeq r|Y|$) où r est "petit" par exemple $r = 10^{-2}, 10^{-3}, \ldots$. Dans la pratique, on peut déjà introduire des erreurs initiales uniquement avec le codage informatique des données[16] puis les calculs seront généralement entachés d'erreur. On obtiendra alors deux flottants x et y :

$$\begin{cases} x = X + \delta x \\ y = Y + \delta y \end{cases}$$

Supposons que x et y soient suffisamment proches, on a donc :

$$x \ominus y = x - y = (X - Y) + \delta x - \delta y,$$

[15] ou de l'addition de deux quantités proche en valeur absolue mais de signe opposé !

[16] lorsque les données initiales sont issues de mesures physiques, elles contiennent déjà une part d'incertitude généralement beaucoup plus grande que les erreurs introduites par le codage informatique

et l'erreur relative sera égale à :

$$er = \frac{|(x \ominus y) - (X - Y)|}{|X - Y|} = \frac{|\delta x - \delta y|}{|X - Y|}.$$

Faisons une hypothèse optimiste : les calculs conduisant aux nombres flottants x et y ont été assez précis et une analyse nous dit que l'erreur relative sur x et y est bornée par $5\epsilon_m$ (c-à-d que $|\delta x/X| \leq 5\epsilon_m$ et $|\delta y/Y| \leq 5\epsilon_m$). En développant un peu les calculs, le cas le moins favorable va donner une erreur relative de :

$$er_{max} = 5\epsilon_m \frac{|X + Y|}{|X - Y|} \simeq 10\epsilon_m \frac{|X|}{|X - Y|} = \frac{10\epsilon_m}{r}.$$

Cette formule fait apparaître le phénomène d'amplification : les erreurs (relatives) sur les calculs antérieurs (inférieures à $5\epsilon_m$) sont multipliés par le coefficient $1/r$. Si l'on veut une erreur relative raisonnable (disons inférieure à tol), les calculs doivent être mené avec un système flottant pour lequel :

$$\epsilon_m \leq \frac{r\ tol}{10}.$$

1.3.3 Calculs d'erreurs classiques

Séquence de multiplications et divisions

Considérons le calcul $x = abc/(de)$ qui, écrit sous la forme `x = (a*b*c)/(d*e)` dans la plupart des langages de programmation, sera effectué selon la séquence :

$$x_c = ((a \otimes b) \otimes c) \oslash (d \otimes e)$$

Dans la suite nous aurons besoin de simplifier des expressions comme celle qui apparaît ci-dessous à gauche par une expression plus simple comme celle de droite :

$$\frac{\prod_{i=1}^{k}(1 + \epsilon_i)}{\prod_{i=k+1}^{n}(1 + \epsilon_i)} = 1 + \delta \tag{1.5}$$

où les ϵ_i sont tels que $|\epsilon_i| < \epsilon_m$, et où k peut varier de 0 (tous les facteurs sont au dénominateur) à n (tous les facteurs sont au nominateur). Le lemme suivant montre que si n n'est pas trop grand alors $|\delta|$ est au maximum de l'ordre de $n\epsilon_m$.

Lemme de simplification : sous les conditions $n\epsilon \leq 0,1$, $\epsilon < 5\ 10^{-3}$, et $|\epsilon_i| < \epsilon$, la valeur de δ dans l'équation (1.5) vérifie :

$$|\delta| < 1,06\ n\epsilon.$$

Rmq : en simple précision $\epsilon \simeq 6\ 10^{-8}$ la condition $n\epsilon \leq 0,1$ est respectée jusqu'à des valeurs de n de l'ordre de $1,6\ 10^6$.

Preuve : voir en annexe.

Revenons à notre calcul, on a :

$$\begin{aligned}
a \otimes b &= ab(1 + \epsilon_1) \\
(a \otimes b) \otimes c &= (a \otimes b)c(1 + \epsilon_2) = abc(1 + \epsilon_1)(1 + \epsilon_2) \\
(d \otimes e) &= de(1 + \epsilon_3) \\
((a \otimes b) \otimes c) \oslash (d \otimes e) &= \frac{((a \otimes b) \otimes c)}{d \otimes e}(1 + \epsilon_4) \\
x_c &= \frac{(1 + \epsilon_1)(1 + \epsilon_2)(1 + \epsilon_4)}{(1 + \epsilon_3)}x
\end{aligned}$$

où $|\epsilon_i| \leq \epsilon_m$. D'après le lemme, ceci s'écrit plus simplement :

$$x_c = x(1 + \delta) \ \text{ avec } |\delta| \leq 4\epsilon_m\ 1,06$$

On voit donc que ces calculs n'engendrent pas d'erreurs importantes[17] tant qu'il n'y a pas *d'overflow* ou *d'underflow*.

[17] avec n calculs on a une erreur relative "quasi" bornée par $n\epsilon_m$ mais celle-ci sera plus petite d'un point de vue statistique : pour obtenir $n\epsilon_m$ il faut que chaque calcul soit effectué avec l'erreur maximum de ϵ_m et que toutes les erreurs s'additionnent dans le même sens !

Séquence d'additions et soustractions

Considérons le calcul $s = x_1 + x_2 + x_3 + x_4$ qui, écrit sous cette même forme en C, C++, f77, f90, Scilab, etc[18]..., sera effectué selon la séquence :

$$s = ((x_1 \oplus x_2) \oplus x_3) \oplus x_4$$

On obtient donc (en supposant l'absence *d'overflow* ou *d'underflow*) :

$$\begin{aligned} x_1 \oplus x_2 &= (x_1 + x_2)(1 + \epsilon_1) \\ (x_1 \oplus x_2) \oplus x_3 &= ((x_1 \oplus x_2) + x_3)(1 + \epsilon_2) \\ &= (x_1 + x_2)(1 + \epsilon_1)(1 + \epsilon_2) + x_3(1 + \epsilon_2) \\ ((x_1 \oplus x_2) \oplus x_3) \oplus x_4 &= (((x_1 \oplus x_2) \oplus x_3) + x_4)(1 + \epsilon_3) \end{aligned}$$

où $|\epsilon_i| < \epsilon_m$. En développant la dernière équation, et en utilisant le lemme, on obtient finalement :

$$\begin{aligned} s_c &= x_1(1 + \epsilon_1)(1 + \epsilon_2)(1 + \epsilon_3) \\ &+ x_2(1 + \epsilon_1)(1 + \epsilon_2)(1 + \epsilon_3) \\ &+ x_3(1 + \epsilon_2)(1 + \epsilon_3) \\ &+ x_4(1 + \epsilon_3) \\ &= s + x_1\delta_3 + x_2\delta_3 + x_3\delta_2 + x_4\delta_1 \end{aligned}$$

avec $|\delta_i| \leq i\epsilon_m \; 1,06$. La généralisation évidente est que si $s = x_1 + x_2 + \cdots + x_n$ alors :

$$s_c = s + x_1\delta_{n-1} + x_2\delta_{n-1} + \sum_{i=3}^{n} x_i\delta_{n+1-i}$$

Cette formule nous donne plusieurs indications :

1. On peut majorer l'erreur absolue de la façon suivante :

$$|s_c - s| \leq \left(\sum_{i=1}^{n} |x_i|\right) (n-1)\epsilon_m \; 1,06$$

   et si $s \neq 0$ on a la majoration suivante pour l'erreur relative :

$$\frac{|s_c - s|}{|s|} \leq \left(\frac{\sum_{i=1}^{n} |x_i|}{|s|}\right) (n-1)\epsilon_m \; 1,06$$

   Par conséquent si tous les x_i sont de même signe $\sum_{i=1}^{n} |x_i| = |s|$ et l'erreur relative est petite (avec grosso modo la même règle que pour les multiplications et divisions[19]). De même si $\sum_{i=1}^{n} |x_i|/|s|$ n'est pas trop grand, le calcul reste précis. Dans le cas contraire, on rencontrera le phénomène d'amplification des erreurs d'arrondi.

2. La formule suggère un algorithme plus stable : comme les premiers nombres additionnés collectent par la suite toutes les erreurs d'arrondi suivantes, on peut essayer de minimiser l'erreur absolue en commençant à sommer les nombres les plus petits en valeur absolue (en faisant aussi deux sommes distinctes, l'une pour les nombres positifs, l'autre pour les nombres négatifs). Ce qui donne l'algorithme suivant :
   (a) tri (croissant) du "vecteur" x (coût d'un tri $O(n\ln(n))$)
   (b) recherche de l'indice[20] k tel que $x_k > 0$ (coût d'une recherche dichotomique $O(ln(n))$)

[18] de même qu'en Perl si on donne des dollars à nos variables : `$s = $x1 + $x2 + $x3 + $x4`

[19] on a quand même une précision meilleure puisque l'on a majoré chaque erreur relative par $(n-1)\epsilon_m \; 1,06$

[20] on supposera que l'algorithme renvoie $k = n + 1$ si tous les x_i sont négatifs

(c) calcul de $s_+ = x_k \oplus x_{k+1} \oplus \cdots \oplus x_n$ (somme nulle si $k = n+1$)

(d) calcul de $s_- = x_{k-1} \oplus x_{k-2} \cdots \oplus x_1$ (somme nulle si $k = 1$)

(e) calcul de $s_c^{bis} = s_+ \oplus s_-$

En fait il existe un algorithme très astucieux du à W. M. Kahan[21] encore plus robuste (vis à vis des erreurs) et plus rapide (car il reste en $O(n)$ opérations). Cet algorithme est explicité plus loin dans la section "*Recettes*".

3. On peut interpréter le calcul comme une somme exacte avec les données perturbées $x_1(1+\delta_{n-1})$, ..., $x_n(1+\delta_1)$. Cette interprétation, nommée *analyse inverse*, est particulièrement intéressante car elle permet de distinguer la stabilité (ou l'instabilité) du problème mathématique sous-jacent (cf section suivante) de la stabilité (ou de l'instabilité) numérique de l'algorithme qui résout le problème mathématique.

1.3.4 Notions sur le conditionnement d'un problème

Soit donc le problème mathématique qui consiste à calculer une quantité scalaire réelle y dépendant régulièrement de données x_1, x_2, ..., x_n. C'est à dire que :

$$y = f(x_1, x_2, \ldots, x_n),\ f \in C^1(\Omega),\ \Omega \text{ un ouvert non vide de } \mathbb{R}^n$$

Pour simplifier l'exposé on considère uniquement le cas $n = 1$ (en posant $x = x_1$). On veut mesurer la sensibilité de f au voisinage de x, c'est à dire mesurer la variation de la sortie $y_p = y + \Delta y$ ($y = f(x)$), en fonction d'une perturbation Δx sur x ($x_p = x + \Delta x$). Effectuons un développement de Taylor au voisinage de x :

$$y_p - y = \Delta y = f'(x)\Delta x + o(\Delta x).$$

Si la perturbation Δx est suffisamment petite, le reste est négligeable et alors :

$$\Delta y \simeq f'(x)\Delta x \tag{1.6}$$

Si on s'intéresse à des perturbations dont l'amplitude maximum est connue ($|\Delta x| \leq \rho$) et que l'on ne veut pas négliger le reste dans un premier temps, le théorème des accroissements finis nous donne :

$$|\Delta y| \leq \max_{z \in B(x,\rho)} |f'(z)| \Delta x \tag{1.7}$$

La dérivée de f en x est donc notre coefficient d'amplification pour l'erreur absolue. Comme on s'intéresse plutôt à l'erreur relative, on obtient (en supposant donc que $y \neq 0$ et $x \neq 0$), dans le premier cas :

$$\left|\frac{\Delta y}{y}\right| \simeq \left|\frac{x f'(x)}{f(x)}\right| \left|\frac{\Delta x}{x}\right| \tag{1.8}$$

et dans le second :

$$\left|\frac{\Delta y}{y}\right| \leq \left(\max_{z \in B(x,\rho)} |f'(z)| \left|\frac{x}{f(x)}\right|\right) \left|\frac{\Delta x}{x}\right| \tag{1.9}$$

nous venons de faire apparaître nos coefficients d'amplification que l'on appelle généralement conditionnement de f au point x.

Définition : on appelle conditionnement au point $x \neq 0$ d'une fonction f dérivable en x et telle que $f(x) \neq 0$, la quantité :

$$\kappa_{(f,x)} = \left|\frac{x f'(x)}{f(x)}\right|$$

La quantité :

$$\kappa_{(f,B(x,\rho))} = \max_{z \in B(x,\rho)} |f'(z)| \left|\frac{x}{f(x)}\right|$$

[21] spécialiste en arithmétique flottante mondialement reconnu et l'un des pères de la norme IEEE 754

pourra s'appeler conditionnement de f dans le voisinage $B(x,\rho)$ de x. On dira que le calcul de $y = f(x)$ est *mal conditionné* si $\kappa_{(f,x)}$ est grand.

Exemple : conditionnement pour la fonction $f(x) = x - a$, avec $a \neq 0$ fixé :

$$\kappa_{(f,x)} = \left| \frac{x}{x-a} \right|$$

Au voisinage de a le conditionnement est donc élevé.

Remarque : avec le codage informatique des données dans l'ordinateur on introduit nécessairement des erreurs relatives de l'ordre de ϵ_m ; si le problème à résoudre a un conditionnement très élevé (disons de l'ordre de $1/\epsilon_m$), alors (avec une résolution exacte à partir des données perturbées) on obtiendra :

$$\frac{|\Delta y|}{|y|} \approx 1$$

ceci avec l'algorithme le plus précis possible ! Mais qu'est ce qu'un algorithme précis (on parle en fait d'algorithme *numériquement stable*). On considère généralement deux définitions, l'une issue de l'*analyse directe* de l'erreur (comme nous l'avons fait jusqu'à présent) et l'autre provenant de l'*analyse inverse* où l'on reporte les erreurs dues à l'algorithme sur les données (comme dans la réinterprétation de la formule de l'erreur pour une somme). Soit $\mathcal{A}$ un algorithme qui résoud le problème $y = f(x)$ (c-à-d qui étant donné x calcule une approximation y_c de $y = f(x)$: mathématiquement c'est aussi une fonction de x : $y_c = f_{\mathcal{A}}(x)$).

Définition 1 : On dira que $\mathcal{A}$ est numériquement stable (au sens de l'analyse directe) si y_c reste proche de y :

$$\frac{|y_c - y|}{|y|} \leq C\epsilon_m$$

avec C pas trop grande.

Définition 2 : On dira que $\mathcal{A}$ est numériquement stable (au sens de l'analyse inverse) s'il calcule exactement la fonction f sur une donnée légèrement perturbée, c-à-d si :

$$y_c (= f_{\mathcal{A}}(x)) = f(x(1+\epsilon)), \quad \text{avec } |\epsilon| \leq C\epsilon_m$$

où la constante C n'est pas trop grande.

Remarques :

1. La taille tolérée pour les constantes C dépend souvent du problème traité : pour un problème simple on sous-entend C inférieur à quelques dizaines ; C sert en fait à comparer la stabilité d'algorithmes résolvant le même problème (voir aussi la dernière remarque).
2. La deuxième définition a l'avantage de séparer la stabilité /instabilité du problème mathématique de celle de l'algorithme. Ainsi la soustraction est toujours stable selon la deuxième définition (lorsque le résultat est proche de 0 c'est le problème qui est mal conditionné).
3. Une analyse inverse est souvent plus simple à mener qu'une analyse directe dès que le problème n'est pas élémentaire (résolution d'un système linéaire, etc...). Cependant pour pouvoir obtenir par la suite une estimation ou une majoration de l'erreur sur y il faut connaître le conditionnement du problème ce qui n'est pas forcément simple.
4. D'une manière générale on a fixé des limites floues pour les constantes C et sur $\kappa_{(f,x)}$ puisque nous avons dit :
   – qu'un problème est "mal conditionné" si $\kappa_{(f,x)}$ est "grand",
   – qu'un algorithme est stable si C est "petite",

mais les valeurs admissibles dépendent de la précision cherchée sur le calcul ainsi que des systèmes flottants que l'on peut utiliser (la caractéristique importante étant ϵ_m). Par exemple si le conditionnement d'un problème est borné par 100000 ce qui peut sembler grand, et si on a un algorithme dont la stabilité, au sens de la définition 2, a une constante d'environ 1000 (algorithme peu stable) alors l'erreur en sortie sera bornée par :

$$\left|\frac{\Delta y}{y}\right| \leq 100000 \times 1000\epsilon_m = 10^7\epsilon_m$$

Donc si on travaille en simple précision ($\epsilon_m \simeq 6\ 10^{-8}$) le résultat risque de n'avoir aucun chiffre significatif alors qu'en double précision ($\epsilon_m \simeq 10^{-16}$), on obtient une erreur relative inférieure à 10^{-9} ce qui peut être largement suffisant !

1.3.5 Stabilité d'un algorithme : un exemple simple

On veut évaluer la fonction :

$$f(x) = \frac{1}{x} - \frac{1}{x+1} = \frac{1}{x(x+1)}$$

pour $x > 0$. Calculons d'abord le conditionnement en un point x :

$$f'(x) = -\frac{2x+1}{x^2(x+1)^2}$$

$$\kappa_{(f,x)} = \left|\frac{xf'(x)}{f(x)}\right| = \frac{|2x+1|}{|x+1|}.$$

Comme on considère $x > 0$, on obtient la majoration suivante indépendante de x :

$$\kappa_{(f,x)} = \frac{|2x|+1}{|x|+1} \leq \frac{2|x|+2}{|x|+1} = 2.$$

Voici donc une fonction qui ne devrait pas poser de problème ! Cependant un mauvais algorithme peut détruire cette propriété. Par exemple, l'algorithme trivial obtenu à partir de la première écriture de f :

$$f_{\mathcal{A}_1}(x) = (1 \oslash x) \ominus (1 \oslash (x \oplus 1))$$

fait apparaître une soustraction de deux nombres proches lorsque $|x|$ est grand devant 1. Cette soustraction amplifiera alors les petites erreurs commises sur les opérations précédentes (2 divisions et une addition). Par contre l'algorithme basé sur la deuxième écriture de f ne posera pas de problème. Cf TD pour une analyse plus complète.

Remarque : souvent on peut améliorer la stabilité en choisissant la "bonne formule" ; celle-ci peut d'ailleurs dépendre du domaine des paramètres (Cf TD).

1.4 Recettes et remèdes

1.4.1 Géneralités

Lorsque l'on envisage une application critique qui utilise des calculs avec des flottants l'idéal serait :

1. d'avoir une estimation (même très grossière) du conditionnement des différents problèmes à résoudre ;
2. d'analyser (de façon directe ou indirecte) les algorithmes utilisés.

À partir de ces informations on peut alors estimer si les systèmes flottants dont on dispose (en général simple et double précision) sont adéquats pour mener les calculs avec une précision suffisante. Si ces systèmes flottants ne sont pas adaptés, on peut alors essayer :

1. d'utiliser des algorithmes plus stables (mais généralement plus lents) ;
2. d'utiliser la quadruple précision (flottants sur 16 octets) si c'est possible ; attention actuellement la quadruple précision est obtenue de façon logicielle[22] , c'est donc beaucoup beaucoup plus lent que les calculs en simple ou en double précision ; cette solution[23] (envisageable avec environ la moitiée

[22] c'est pour cette raison que les "vieux" VAX étaient encore recherchés il n'y a pas si longtemps : ils disposaient de calculs en quadruple effectués au niveau hardware !

[23] c'est effectivement une solution si cette précision suffit !

des compilateurs disponibles) est assez pratique car vous n'avez que très peu de modifications à apporter au code ;

3. si la quadruple précision n'est pas disponible ou bien insuffisante, on peut utiliser des bibliothèques[24] qui proposent des calculs (plus l'évaluation des fonctions classiques, sin, exp, etc...) avec un système flottant arbitraire.

Remarque : souvent il suffit d'utiliser une précision supérieure uniquement localement pour les calculs comportant un risque d'instabilité.

Pour les problèmes classiques (résolution de systèmes linéaires, interpolation polynômiale, interpolation par spline cubique, problèmes de moindres carrés, problèmes de valeurs propres, etc...) les conditionnements sont connus et parfois on peut en obtenir une approximation grossière à moindre coût. De même la stabilité pour certains algorithmes classiques est bien connue, mais en dehors de ces sentiers battus il peut être assez difficile de maîtriser la situation :

- si théoriquement il est assez simple de calculer le(s) conditionnement(s), les évaluer peut être hasardeux : un calcul de conditionnement peut être très mal conditionné lui-même !
- une analyse *directe* ou *indirecte* des algorithmes est en général très difficile et particulièrement pénible !

Ainsi, dans beaucoup de cas on est incapable de savoir si les chiffres obtenus en fin de calcul sont crédibles : un ingénieur spécialiste du problème traité par l'ordinateur (par exemple un calcul de structure) pourra souvent dire si les résultats sont aberrants ou encore comparer avec une expérience. On peut aussi effectuer deux calculs, l'un en simple précision, l'autre en double : si les résultats sont cohérents cela nous conforte sur leur précision mais n'enlève pas tous les doutes car une instabilité forte peut faire en sorte que les résultats soient presque identiques mais complètement faux !

Pour pallier à ces inconvénients, il existe quelques solutions issues de la recherche et dont certaines commencent à être opérationnelles :

1. l'arithmétique d'intervalle : cette technique consiste à remplacer chaque nombre flottant par un intervalle I_x. On définit les opérations habituelles sur ces intervalles de manière à ce que le calcul de $I_z = I_x \odot I_y$ vérifie la propriété suivante : $\forall x \in I_x$ et $\forall y \in I_y$ alors $z = x.y \in I_z$. L'avantage de ce système est que l'on est sûr d'encadrer les résultats cherchés avec un coût supplémentaire assez faible (en gros on effectue 2 fois plus de calculs). L'inconvénient est que, par la nature même de ce procédé, les bornes que l'on obtient sont la plupart du temps sans commune mesure avec les erreurs effectuées par un calcul classique : soit X le résultat théorique exact cherché, on obtient en sortie un intervalle $[a, b]$ tel que $X \in [a, b]$ mais l'erreur absolue "suggérée" à savoir $|b - a|$ sera géneralement beaucoup plus grande que l'erreur obtenue par le calcul classique (par contre cette dernière est difficile à estimer puisqu'il faut connaître le conditionnement du problème et la stabilité des algorithmes mis en jeux). Voici un site internet dédié cette technique : `http://www.cs.utep.edu/interval-comp/main.html`
2. des outils stochastiques ont été étudiés et développés par une équipe française (comprenant J. Vignes et J.M. Chesneaux). Ces outils ont été intégrés dans un logiciel, CADNA[25], qui permet (entre autres) de donner le nombres de chiffres significatifs pour tous les calculs et de détecter les instabilités.

1.4.2 Problèmes d'overflow

Une remarque générale pour éviter les problèmes d'overflow et d'underflow est de "normaliser" les équations traitées en utilisant des unités adéquates pour lesquelles les grandeurs mise en jeux sont raisonnables.

Parfois au cours d'un programme on est obligé de calculer des quantités intermédiaires dont la valeur absolue dépasse M alors que celle du résultat cherché est OK. On peut dans ce cas effectuer localement les calculs avec un système flottant dont l'étendue des exposants est plus grande. Souvent une réécriture

[24] dont d'excellentes sont libres et/ou gratuites par exemple mpfun de *David H. Bailey* et gmp de chez gnu (http ://www.swox.com/gmp/)

[25] pour en savoir plus : `http ://www-anp.lip6.fr/chpv/francais/cadna/`

des formules de calcul peut fonctionner aussi. Par exemple si vous devez utiliser un langage qui ne fournit pas la fonction tangente hyperpolique, dont la définition classique est :

$$\mathrm{th}(x)\left(=\tanh(x)=\frac{\mathrm{sh}(x)}{\mathrm{ch}(x)}\right)=\frac{e^x-e^{-x}}{e^x+e^{-x}}$$

il ne faut surtout pas l'évaluer en recopiant cette formule ! Rappelons que $\mathrm{th}(x)\in[-1,1]$ (les bornes étant atteintes pour $x=\pm\infty$) mais comme $e^{89}>M$ en simple précision et $e^{710}>M$ en double, le programme renverra NaN pour la plupart des nombres flottants (le calcul de th(89) en simple précision conduit à $+inf\oslash+inf$ soit NaN). Il est cependant assez facile de contourner ce problème en utilisant la formulation suivante :

$$\mathrm{th}(x)=\begin{cases}\dfrac{1-e^{-2x}}{1+e^{-2x}} & \text{pour } x\geq 0,\\ \dfrac{e^{2x}-1}{e^{2x}+1} & \text{pour } x<0.\end{cases} \tag{1.10}$$

Celle-ci est d'ailleurs améliorable dans le cas où $|x|$ est suffisamment petit (on obtient alors une soustraction de deux nombres proches et l'erreur sur le calcul de e^{2x} ou e^{-2x} sera amplifiée). Le développement de Taylor en zéro :

$$\mathrm{th}(x)=x-\frac{x^3}{3}+O(x^5)$$

nous fait penser que pour $|x|$ suffisamment petit, mieux vaudra évaluer avec $\mathrm{th}(x)\simeq x$ plutôt que d'utiliser les formules précédentes. On obtiendra alors l'algorithme (dans lequel th denote la fonction calculée par l'ordinateur) :

si $|x|\leq\tau$ alors $th(x)=x$
sinon $th(x)$ est donné les formules (1.10).

Le problème est d'évaluer la limite τ pour laquelle on change d'algorithme ce qui demande de connaître la précision du calcul de la fonction exponentielle. Supposons que celle-ci est bien programmée et que l'erreur relative est bornée par $2\epsilon_m$. L'analyse de l'erreur avec (1.10) conduit à :

$$er_1(x)=\left|\frac{\mathrm{th}(x)-th(x)}{\mathrm{th}(x)}\right|\simeq\left|\frac{\epsilon}{x}\right|,\quad\text{avec } |\epsilon|\leq\epsilon_m,$$

où ϵ vient uniquement de l'erreur commise sur exp (les autres sont négligeables). Pour la formule $th(x)=x$ l'erreur relative est :

$$er_2(x)=\left|\frac{\mathrm{th}(x)-th(x)}{\mathrm{th}(x)}\right|=\left|\frac{\mathrm{th}(x)-x}{\mathrm{th}(x)}\right|\simeq\left|\frac{x^2}{3}\right|$$

Prenons le cas où $|\epsilon|=\epsilon_m$ (une valeur "moyenne" serait sans doute plus proche de $\epsilon_m/2$ mais cela n'a pas trop d'importance). Le choix optimal est obtenu quand les deux erreurs sont égales (faire un dessin pour comprendre) ce qui nous donne :

$$\tau=\sqrt[3]{3\epsilon_m}$$

Remarque : en double précision on obtient $\tau\simeq 7\ 10^{-6}$ (et $5,5\ 10^{-6}$ si on utilise $|\epsilon|=\epsilon_m/2$). On peut alors facilement calculer l'erreur relative au voisinage de τ ce qui donne environ $1,6\ 10^{-11}$: si on veut faire mieux il faut améliorer l'approximation en utilisant par exemple $th(x)=x*(1-x^2/3)$ (il faut aussi recalculer une bonne valeur pour τ) ou d'autres techniques !

1.4.3 Quelques algorithmes

L'algorithme de sommation de W. Kahan

Le calcul de la somme $s=\sum_{i=1}^{n}x_i$ est effectué par l'algorithme :

$S:=x_1$
$C:=0$
pour $i:=2$ **à** n
  $y:=x_i-C$

$t := S + y$
$C := (t - S) - y$
$S := t$
fin pour

Explication intuitive de l'algorithme :

1. à chaque itération on calcule un "petit coefficient correcteur" C qui sera appliqué à l'itération suivante (pour la première itération ce coef est 0) ;
2. à l'itération i avant d'incrémenter la somme ($s := s + x_i$), on applique ce coefficient correcteur à x_i pour obtenir $y := x_i - C$ (pour le moment pensez que y est très proche de x_i) ;
3. on effectue ensuite l'incrémentation de la somme avec $t := S + y$ (t est une variable temporaire car on a encore besoin de s) : moralement dans cette somme les bits de poids faible de la mantisse de y sont perdus (disons que $y = y_h + y_b$ avec $|y_h| > |y_b|$ où y_b est la partie de y qui est perdue lors du calcul de t) ;
4. pour récuperer y_b on effectue la troisième instruction, où l'on obtient : $C = (S + y_h \ominus S) \ominus y \simeq -y_b$, le prochain coefficient correcteur.

Simple mais il fallait y penser ! Une démonstration précise est bien sûr assez compliquée mais on peut montrer que la quantité S obtenue par cet algorithme vérifie (sous la condition de ne pas avoir d'overflow ou d'underflow) :

$$S = \sum_{i=1}^{n} x_i(1 + \epsilon_i), \quad \text{avec } |\epsilon_i| \leq 2\epsilon_m + O(n\epsilon_m^2)$$

Si on néglige le terme en $O(n\epsilon_m^2)$ on obtient donc un algorithme plus stable que l'algorithme classique.
Rmq : Quand on programme cet algorithme il ne faut surtout pas utiliser d'options d'optimisation qui permettent au compilateur de réarranger les calculs pour plus de rapidité. Ces réarrangements sont basés sur des propriétés d'associativité ou autres qui sont souvent légèrement fausses en arithmétique flottante. En général ce n'est pas trop important mais comme l'algorithme ci-dessus exploite justement les défauts (ou les qualités) de l'arithmétique flottante cela le détruit complètement : en optimisant, le compilateur peut simplifier la troisième instruction de la boucle ($C := (t - S) - y$) en utilisant l'instruction précédente, ce qui donne dans un premier temps : $C := ((S + y) - S) - y$ et dans un deuxième $C := 0$. Finalement on se retrouve alors avec l'algorithme classique d'une somme !

Choix du pas pour le calcul d'une dérivée

Dans de nombreux algorithmes qui travaillent sur des fonctions (minimisation, recherche d'une racine, etc...) on a besoin d'évaluer la dérivée de nombreuses fois. La bonne méthode est d'obtenir une expression analytique de cette dérivée (par exemple en utilisant un logiciel de calcul formel) ou encore, en utilisant un logiciel de différentiation automatique, qui, à partir du sous-programme (écrit en C, C++, Fortran, etc) calculant la fonction, vous sort le sous-programme qui calcule la dérivée. Cependant ceci n'est pas toujours possible et l'on doit alors recourir à un calcul approché, par exemple avec la formule :

$$f'(x) = \frac{f(x + h) - f(x)}{h} + O(h),$$

en négligeant le reste $O(h)$, ou encore avec la formule symétrique :

$$f'(x) = \frac{f(x + h) - f(x - h)}{2h} + O(h^2).$$

Pour obtenir une bonne précision, il faut prendre h petit, mais si h est trop petit, le calcul consistant à soustraire 2 nombres proches, on va considérablement amplifier les erreurs commises nécessairement sur la fonction f (d'autre part pour h suffisamment petit $x \oplus h = x$). Bref il faut trouver un compromis. On peut trouver une valeur raisonnable pour h si on a une idée de l'erreur commise sur f et si on connait une approximation de $f''(x)$ pour la première formule et de $f^{(3)}(x)$ pour la deuxième. Prenons la première formule (les explications se transposent aisément pour la deuxième), on calcule la quantité :

$$dF = (F(x \oplus h) \ominus F(x)) \oslash h$$

où F désigne la fonction évaluée par la machine (selon un certain algorithme). Supposons que :

1. $F(x) = f(x)(1+\delta)$ avec $|\delta| \leq C\epsilon_m$,
2. $F(x \oplus h)$ et $F(x)$ sont assez proches (hypothèse naturelle),
3. que h est une puissance de 2 (pas d'erreur dans la division),
4. et que l'on puisse négliger l'erreur sur $x \oplus h$ par rapport à l'erreur sur le calcul de f (cette hypothèse sert à simplifier le calcul mais est raisonnable si l'on choisit h pas trop petit par rapport à x).

Développons le calcul d'erreur :

$$\begin{aligned} df &= \frac{F(x+h) - F(x)}{h} \\ &= \frac{f(x+h)(1+\delta_1) - f(x)(1+\delta_2)}{h} \\ &= \frac{f(x+h) - f(x)}{h} + \frac{f(x+h)\delta_1 - f(x)\delta_2}{h} \\ &= f'(x) + \frac{h}{2}f''(x) + O(h^2) + \frac{f(x+h)\delta_1 - f(x)\delta_2}{h} \end{aligned}$$

Supposons maintenant que l'on puisse négliger le terme en $O(h^2)$, et que l'on puisse simplifier le dernier terme avec $f(x+h)\delta_1 \simeq f(x)\delta_1$, ces deux nouvelles hypothèses étant cohérentes si h est suffisamment petit (mais pas trop pour que l'hypothèse précédente soit valide !). On obtient alors :

$$df \simeq f'(x) + \frac{h}{2}f''(x) + \frac{f(x)}{h}\delta$$

avec $|\delta| \leq 2C\epsilon_m$. Le deuxième terme est l'erreur de méthode (le fait d'approcher $f'(x)$ par $(f(x+h) - f(x))/h$) et le troisième l'erreur dans le calcul de $(f(x+h) - f(x))/h$ par ordinateur. L'erreur absolue est donc bornée par :

$$|df - f'(x)| \leq |\frac{h}{2}f''(x) + \frac{f(x)}{h}\delta| \leq |\frac{h}{2}f''(x)| + |\frac{f(x)}{h}\delta|$$

Faisons une hypothèse sur $|\delta|$: en moyenne $|\delta| = C\epsilon_m$, on cherche alors la valeur de h qui minimise la fonction :

$$g(h) = |\frac{h}{2}f''(x)| + |\frac{f(x)}{h}C\epsilon_m|$$

et il est aisé de voir que le minimum est obtenu pour :

$$|\frac{h}{2}f''(x)| = |\frac{f(x)}{h}C\epsilon_m|$$

ce qui nous donne finalement :

$$h = \sqrt{\left|\frac{2C\epsilon_m f(x)}{f''(x)}\right|}$$

(pour respecter l'une des hypothèses on peut prendre la puissance de 2 la plus proche).

Ce calcul très empirique[26] donne néanmoins des valeurs correctes pour le pas h (il faut cependant avoir une idée pour la constante C et une approximation de $f''(x)$).

[26] on peut effectuer un calcul moins empirique et beaucoup plus compliqué utilisant des probabilités mais on obtient à peu près le même résultat

Annexe

Exemple de comportements différents pour les 2 modes d'arrondi : avec $\beta = 10$ et $p = 8$ et les 2 flottants $u = 1,0000000$ et $v = 5,5555555\ 10^{-1}$:

$$\begin{aligned} u \oplus v &= fl(\underline{1,55555555}) \\ &= 1,5555556 \text{ pour les 2 modes d'arrondi} \\ (u \oplus v) \ominus v &= fl(1,5555556 - 0,55555555) \\ &= fl(\underline{1,00000005}) \\ &= 1,0000001 \text{ avec l'arrondi classique} \\ &= 1,0000000 \text{ avec l'arrondi IEEE} \end{aligned}$$

Si on poursuit l'opération : $(((u \oplus v) \ominus v) \oplus v) \ominus v$ etc...avec le mode arrondi classique on obtient un nombre qui croît lentement mais sûrement.

Preuve du Lemme de simplification : On minore et majore l'expression de gauche en prenant les cas extrèmes :

$$\frac{(1-\epsilon)^k}{(1+\epsilon)^{n-k}} \leq \frac{\prod_{i=1}^{k}(1+\epsilon_i)}{\prod_{i=k+1}^{n}(1+\epsilon_i)} \leq \frac{(1+\epsilon)^k}{(1-\epsilon)^{n-k}}$$

On commence par la minoration, comme :

$$\frac{1}{1+\epsilon} = (1 - \frac{\epsilon}{1+\epsilon}) \geq 1 - \epsilon$$

on a donc :

$$(1-\epsilon)^n \leq \frac{(1-\epsilon)^k}{(1+\epsilon)^{n-k}}$$

(c-a-d que le cas le plus défavorable est obtenu pour $k = n$). Posons $f(\epsilon) = (1-\epsilon)^n$, et appliquons le théorème des accroissements finis :

$$\begin{aligned} f(\epsilon) &= f(0) + \epsilon f'(\xi),\ \xi \in]0, \epsilon[\\ (1-\epsilon)^n &= 1 - n\epsilon(1-\xi)^{n-1} \ \geq 1 - n\epsilon \end{aligned}$$

On vient donc de montrer que :

$$1 - n\epsilon \leq \frac{\prod_{i=1}^{k}(1+\epsilon_i)}{\prod_{i=k+1}^{n}(1+\epsilon_i)}.$$

Passons maintenant à la majoration, comme :

$$\frac{1}{1-\epsilon} = 1 + \epsilon + \epsilon^2 + \cdots \geq 1 + \epsilon$$

on en déduit que le cas le plus défavorable est obtenu pour $k = 0$:

$$\frac{(1+\epsilon)^k}{(1-\epsilon)^{n-k}} \leq \frac{1}{(1-\epsilon)^n}$$

En posant $\varepsilon = \epsilon/(1-\epsilon)$:

$$\left(\frac{1}{1-\epsilon}\right)^n = (1 + \frac{\epsilon}{1-\epsilon})^n = (1+\varepsilon)^n$$

On utilise maintenant l'inégalité $\ln(1+x) \leq x$ et la croissance de l'exponentielle pour obtenir :

$$(1+\varepsilon)^n = e^{n\ln(1+\varepsilon)} \leq e^{n\varepsilon} = 1 + n\varepsilon + \frac{(n\varepsilon)^2}{2!} + \frac{(n\varepsilon)^3}{3!} + \ldots$$

puis l'inégalité $(k+1)! \geq 2^k$:

$$1 + n\varepsilon \left(1 + \frac{n\varepsilon}{2!} + \frac{(n\varepsilon)^2}{3!} + \dots \right) \leq 1 + n\varepsilon \left(1 + \frac{n\varepsilon}{2} + \left(\frac{n\varepsilon}{2} \right)^2 + \dots \right) = 1 + n\varepsilon \left(\frac{1}{1 - n\varepsilon/2} \right)$$

Revenons à ϵ et utilisons les hypothèses $n\epsilon \leq 0.1$ et $\epsilon \leq 5\ 10^{-3}$:

$$1 + n\varepsilon \left(\frac{1}{1 - n\varepsilon/2} \right) = 1 + n\epsilon \left(\frac{1}{1 - \epsilon - n\epsilon/2} \right) \leq 1 + n\epsilon \left(\frac{1}{1 - 5\ 10^{-3} - 0,1/2} \right) \leq 1 + 1,06\ n\epsilon$$

Chapitre 2

Résolution de systèmes linéaires

2.1 Introduction

Un grand nombre de problèmes d'ingénierie conduisent, après modélisation puis éventuellement une procédure de discrétisation, à résoudre des systèmes linéaires : calculs sur des réseaux électriques ou hydrauliques en régime stationnaire, calculs de structures, etc... D'autre part, ce type de problème intervient aussi lorsque l'on met en œuvre d'autres méthodes numériques : par exemple la méthode de Newton (qui permet de calculer un zéro d'une fonction de $\mathbb{R}^n \to \mathbb{R}^n$) conduit à résoudre une succession de systèmes linéaires.

On cherche donc à résoudre le problème : "étant donnés une matrice A et un vecteur b, trouver le vecteur x solution de" :

$$Ax = b \quad \text{avec} \quad \begin{cases} A \in \mathcal{M}_{nn}(\mathbb{K}) \\ x \in \mathbb{K}^n \\ b \in \mathbb{K}^n \end{cases} \tag{2.1}$$

Dans la plupart des cas $\mathbb{K} = \mathbb{R}$, mais comme résoudre des systèmes linéaires avec des nombres complexes arrive aussi dans des cas pratiques (exemple : calcul de réseau électrique en régime sinusoïdal permanent) on supposera donc que $\mathbb{K} = \mathbb{R}$ ou $\mathbb{C}$.

On admettra que la matrice A est inversible, c-a-d qu'elle possède l'une des propriétés équivalentes suivantes (et donc toutes ces propriétés !) :

1. $\det(A) \neq 0$,
2. $\forall b \in \mathbb{K}^n, \exists! x \in \mathbb{K}^n$ tel que $Ax = b$,
3. $KerA = \{0\}$,
4. $ImA = \mathbb{K}^n$,
5. 0 n'est pas valeur propre de A.

Dans la pratique, le problème (2.1) peut être très mal conditionné (vis à vis du système flottant utilisé), et la solution numérique obtenue sera généralement de mauvaise qualité. Une "bonne" méthode numérique doit être capable de détecter ce genre de problème pour pouvoir prévenir l'utilisateur. Dans la suite nous allons simplement apprendre le b.a.ba sur quelques méthodes sans nous occuper de ce problème (mathématiquement il fait appel à l'outil des normes matricielles, qui sort de l'objectif de ce cours élémentaire !).

2.2 La méthode de Gauss (rappels)

Elle consiste à effectuer des transformations successives ($n-1$ en tout) sur (2.1) qui conduisent à un système linéaire équivalent dont la matrice est triangulaire supérieure : le résoudre est alors très simple !

Écrivons l'équation matricielle (2.1) comme système d'équations linéaires :

$$\left\{\begin{array}{rcl} a_{11}x_1 + a_{12}x_2 + \ldots\cdots + a_{1n}x_n &=& b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots\cdots + a_{2n}x_n &=& b_2 \\ \vdots & & \vdots \\ \vdots & & \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots\cdots + a_{nn}x_n &=& b_n \end{array}\right. \tag{2.2}$$

- Si $a_{11} = 0$, on cherche alors $a_{k1} \neq 0$ et l'on échange les équations 1 et k : ceci est toujours possible puisque si tous les coefficients de la première colonne de A sont nuls alors $\det(A) = 0$.
- Supposons donc $a_{11} \neq 0$ (ce qui est donc toujours possible modulo un échange de lignes), l'idée de la méthode est de soustraire la première équation multipliée par le "bon" coef à la deuxième équation, de façon à éliminer la variable x_1 dans la nouvelle (deuxième) équation ainsi obtenue. On recommence ensuite cette manip sur l'équation 3, puis sur la 4, etc, et enfin sur la dernière. Le coefficient qui permet d'éliminer la variable x_1 dans l'équation i se calcule par :

$$coef_i = \frac{a_{i1}}{a_{11}}$$

et la nouvelle équation i obtenue $equ_i^{(new)} = equ_i - coef_i \times equ_1$ est :

$$(a_{i2} - \frac{a_{i1}}{a_{11}}a_{12})x_2 + \cdots + (a_{in} - \frac{a_{i1}}{a_{11}}a_{1n})x_n = b_i - \frac{a_{i1}}{a_{11}}b_1$$

Nous venons d'effectuer la première étape de la méthode de Gauss, le nouveau système linéaire obtenu est :

$$A^{(1)}x = b^{(1)} : \left\{\begin{array}{rcl} a_{11}x_1 \quad +a_{12}x_2 + \ldots\cdots + a_{1n}x_n &=& b_1 \\ 0 \quad +a_{22}^{(1)}x_2 + \ldots\cdots + a_{2n}^{(1)}x_n &=& b_2^{(1)} \\ \vdots \qquad\qquad \vdots & & \vdots \\ \vdots \qquad\qquad \vdots & & \vdots \\ 0 \quad +a_{n2}^{(1)}x_2 + \ldots\cdots + a_{nn}^{(1)}x_n &=& b_n^{(1)} \end{array}\right. \tag{2.3}$$

avec :

$$a_{ij}^{(1)} = a_{ij} - a_{1j}\frac{a_{i1}}{a_{11}} \text{ pour } 2 \leq i,j \leq n$$

$$b_i^{(1)} = b_i - b_1\frac{a_{i1}}{a_{11}} \text{ pour } 2 \leq i \leq n.$$

Remarques :

1. la transformation qui permet d'obtenir le nouveau système (2.3) est inversible : il est très facile de retrouver l'ancien système à partir du nouveau ; on verra plus loin que cette transformation est une application linéaire de $\mathbb{K}^n$ dans $\mathbb{K}^n$ que l'on notera matriciellement $M^{(1)}$, on a donc : $A^{(1)} = M^{(1)}A$ et $b^{(1)} = M^{(1)}b$;
2. les manipulations sur les équations du système sont équivalentes à des manipulations sur les lignes de la matrice A et du vecteur b : ainsi éliminer l'inconnue x_1 des équations $2, 3, \ldots, n$ correspond à faire apparaître des zéros sous la diagonale dans la première colonne de la matrice $(A^{(1)})$ avec les opérations :

$$ligne_i^{(new)} = ligne_i - coef_i \times ligne_1 \text{ pour } 2 \leq i \leq n.$$

Il faut bien sûr appliquer la même transformation sur le vecteur b.

Et la suite ? La deuxième étape revient à faire exactement la même chose sur le sous-système constitué par les équations $2, 3, \ldots, n$. Comme ce sous-système ne dépend pas de la variable x_1, c'est un système de $n-1$ équations à $n-1$ inconnues, sur lequel on applique la même technique :

- si le coefficient $a_{22}^{(1)} \neq 0$ on procède tout de suite à la phase d'élimination de l'inconnue x_2 dans les équations $3, 4, \ldots, n$;

– sinon on cherche un coefficient non nul :

$$a_{i2}^{(1)} \neq 0, \qquad 2 < i \leq n$$

puis on échange les équations 2 et i et l'on procède alors à la phase d'élimination.

Remarque : on peut toujours trouver $i \in [2, n]$ tel que $a_{i2}^{(1)} \neq 0$; en effet comme la matrice $A^{(1)}$ a le découpage par blocs suivant :

$$A^{(1)} = \left(\begin{array}{c|c} a_{11} & a_{12} \dots a_{1n} \\ \hline 0 & \\ \vdots & \widetilde{A}^{(1)} \\ 0 & \end{array}\right)$$

le calcul de son déterminant (en développant par rapport à la première colonne) donne :

$$\det(A^{(1)}) = a_{11} \det(\widetilde{A}^{(1)}).$$

D'autre part comme $A^{(1)}$ est obtenue à partir de A sur laquelle on a appliqué une transformation linéaire inversible ($A^{(1)} = M^{(1)}A$), son déterminant est non nul. Ainsi, comme $a_{11} \neq 0$, la première colonne de la matrice $\widetilde{A}^{(1)}$ ne peut être nulle (sinon son déterminant serait nul et par conséquent celui de $A^{(1)}$ aussi), c-a-d qu'il existe bien $i \in [2, n]$ tel que $a_{i2}^{(1)} \neq 0$. On admettra que ce résultat se généralise pour les autres étapes, c-a-d qu'à chaque étape k on peut trouver un coefficient non nul :

$$a_{ik}^{(k-1)} \neq 0, \quad \text{avec } i \in [k, n]$$

et donc que la méthode de Gauss marche à tous les coups sur une matrice inversible (en arithmétique exacte !).

Et la fin ? Au bout de $n-1$ étapes (qui consistent donc toutes à faire la même chose sur des systèmes linéaires de plus en plus petits), on obtient le système linéaire équivalent $A^{(n-1)}x = b^{(n-1)}$, où la matrice $A^{(n-1)}$ est triangulaire supérieure :

$$A^{(n-1)} = \begin{pmatrix} a_{11} & \times & \dots & \times \\ 0 & a_{22}^{(1)} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \times \\ 0 & \dots & 0 & a_{nn}^{(n-1)} \end{pmatrix}$$

avec ses éléments diagonaux (les pivots) $a_{ii}^{(i-1)}$ tous non nuls. Le système linéaire se calcule alors par la "remontée" :

pour $i = n, n-1, \dots, 1$
$\quad x_i = \left(b_i^{(i-1)} - \sum_{j=i+1}^{n} a_{ij}^{(i-1)} x_j\right) / a_{ii}^{(i-1)}$
fin pour

2.3 L'interprétation matricielle de la méthode de Gauss

2.3.1 Quelques notations

– Soit B une matrice de format (n, m) (n lignes et m colonnes), on notera :

B^j le vecteur colonne formé par la j$^{\text{ème}}$ colonne de B ;

B_i le vecteur ligne formé par la i$^{\text{ème}}$ ligne de B ;

avec ces notations, on a les découpages par blocs de la matrice B :

$$B = \left(\begin{array}{c} B_1 \\ \hline B_2 \\ \hline \vdots \\ \hline B_n \end{array}\right) = \left(\begin{array}{c|c|c|c} B^1 & B^2 & \dots & B^m \end{array}\right)$$

– Dans un espace $\mathbb{K}^n$ on notera e^j les vecteurs de la base canonique ($(e^j)_i = \delta_{ij}$) :

$$e^j = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \; j^{\text{ème}} \text{ composante}$$

On remarque facilement que (B étant la matrice précédente) :

$$Be^j = B^j$$

où ici e^j est le j$^{\text{ème}}$ vecteur de la base canonique de $\mathbb{K}^m$, et que :

$$(e^i)^\top B = B_i$$

où e^i est le i$^{\text{ème}}$ vecteur de la base canonique de $\mathbb{K}^n$.

2.3.2 Action d'une matrice de la forme $M = I + z(e^k)^\top$

Soit B une matrice (n, m), on regarde la transformation opérée par la multiplication MB où $z \in \mathbb{K}^n$, e^k est le k$^{\text{ème}}$ vecteur de la base canonique de $\mathbb{K}^n$ et I l'identité dans $\mathbb{K}^n$. La matrice M correspond à une matrice identité dans laquelle on a ajouté le vecteur z dans la k$^{\text{ème}}$ colonne (faire le calcul !).

$$MB = (I + z(e^k)^\top)B = B + z(e^k)^\top B = B + zB_k$$

En décomposant la matrice MB ligne par ligne, on obtient :

$$MB = \begin{pmatrix} \hline B_1 + z_1 B_k \\ \hline B_2 + z_2 B_k \\ \hline \vdots \\ \hline B_n + z_n B_k \\ \hline \end{pmatrix}$$

c'est à dire qu'à chaque ligne i de la matrice B initiale, on a ajouté la ligne k multipliée par le coefficient z_i (i$^{\text{ème}}$ composante du vecteur z). D'autre part si $z_k = 0$ alors la matrice M est nécessairement inversible d'inverse :

$$L = (M^{-1}) = I - z(e^k)^\top$$

En effet :

$$\begin{aligned} LM &= (I - z(e^k)^\top)(I + z(e^k)^\top) \\ &= I \underbrace{-z(e^k)^\top + z(e^k)^\top}_{0} - z(e^k)^\top z(e^k)^\top \\ &= I - z\left((e^k)^\top z\right)(e^k)^\top \\ &= I - z\underbrace{(z|e^k)}_{z_k=0}(e^k)^\top \\ &= I \end{aligned}$$

(la notation $(x|y)$ désignant le produit scalaire). Avec cet outil, on obtient facilement les matrices qui permettent de passer d'une étape à l'autre dans la méthode de Gauss (l'échange des lignes pour trouver un pivot non nul, est lui obtenu par une matrice de permutation élémentaire dont nous parlerons plus loin) :

– pour l'étape 1 : $A^{(1)} = M^{(1)}A$ et $b^{(1)} = M^{(1)}b$ avec $M^{(1)} = I - z^{(1)}(e^1)^\top$ le vecteur $z^{(1)}$ étant :

$$z^{(1)} = \begin{pmatrix} 0 \\ a_{21}/a_{11} \\ \vdots \\ a_{n1}/a_{11} \end{pmatrix}$$

En effet $M^{(1)}$ effectue bien les opérations attendues sur les lignes :

$$\begin{cases} A_1^{(1)} = A_1 - 0 \times A_1 \\ A_i^{(1)} = A_i - z_i \times A_1 \quad \text{pour } 2 \leq i \leq n \end{cases}$$

(et on a le même effet sur b bien sûr).

– pour une étape k quelconque ($1 \leq k \leq n-1$), la matrice $A^{(k-1)}$ étant de la forme :

$$\begin{pmatrix} a_{11} & \times & \times & \times & \times & \times \\ 0 & a_{22}^{(1)} & \times & \times & \times & \times \\ \vdots & \ddots & \ddots & \times & \times & \times \\ \vdots & \dots & 0 & a_{kk}^{(k-1)} & \times & \times \\ \vdots & \dots & \vdots & \vdots & \times & \times \\ 0 & \dots & 0 & a_{nk}^{(k-1)} & \times & \times \end{pmatrix}$$

si $a_{kk}^{(k-1)} \neq 0$ [1] on peut procéder à l'élimination (de la variable x_k dans les équations $k+1, \dots, n$) en utilisant la matrice $M^{(k)} = I - z^{(k)}(e^{(k)})^\top$ avec :

$$z^{(k)} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ a_{k+1,k}^{(k-1)}/a_{kk}^{(k-1)} \\ \vdots \\ a_{n,k}^{(k-1)}/a_{kk}^{(k-1)} \end{pmatrix}$$

Remarquons qu'avec ces vecteurs $z^{(k)}$ particuliers, les matrices $M^{(k)}$ sont inversibles d'inverse $I + z^{(k)}(e^{(k)})^\top$, et qu'elles sont aussi triangulaires inférieures puisque les k premières composantes des $z^{(k)}$ sont nulles. De plus comme les coefficients diagonaux de ces matrices triangulaires sont tous égaux à 1, on a aussi $\det(M^{(k)}) = 1$ (de même pour leurs inverses !).

2.3.3 Théorème 1 : La décomposition $A = LU$

Soit une matrice A (n, n) inversible et telle qu'à chaque étape de la méthode de Gauss on ait (sans procéder à des échanges de lignes) :

$$a_{kk}^{(k-1)} \neq 0 \qquad \forall k \in [1, n-1].$$

Alors il existe une matrice triangulaire inférieure L à diagonale unité (c-a-d que $l_{ii} = 1,\ \forall i$) et une matrice triangulaire supérieure inversible U telles que :

$$A = LU$$

cette décomposition étant unique.

[1] dans le cas contraire il faut au préalable, échanger la ligne k avec une ligne $i > k$ de façon à obtenir un pivot non nul.

Preuve abrégée : On pose $U = A^{(n-1)}$ la dernière matrice obtenue (qui possède donc les propriétés attendues, sauf qu' il reste à montrer que $a_{nn}^{(n-1)}$ est non nul), on a :

$$U = A^{(n-1)} = M^{(n-1)}A^{(n-2)} = M^{(n-1)}M^{(n-2)}A^{(n-3)} = M^{(n-1)}M^{(n-2)}\dots M^{(1)}A \tag{2.4}$$

où les matrices $M^{(k)}$ sont de la forme $M^{(k)} = I - z^{(k)}(e^k)^\top$ avec $z_i^{(k)} = 0$ pour $i \in [1, k]$. Si on prend les déterminants de l'équation ci-dessus :

$$\det(U) = \prod_{i=1}^{n} a_{ii}^{(i-1)} = \left(\prod_{i=1}^{n-1} \det(M^{(i)})\right) \det(A) = \det(A)$$

car $\det(M^{(i)}) = 1$, par conséquent on a nécessairement $a_{nn}^{(n-1)} \neq 0$. Les inverses de ces matrices sont appelées $L^{(k)}$ et donc $L^{(k)} = I + z^{(k)}(e^k)^\top$ d'après un résultat précédent. En multipliant (2.4) à gauche par successivement $L^{(n-1)}$, $L^{(n-2)}$, …, $L^{(1)}$, on obtient :

$$A = L^{(1)}L^{(2)}\dots L^{(n-1)}U \tag{2.5}$$

On pose alors $L = L^{(1)}L^{(2)}\dots L^{(n-1)}$. Comme produit de matrices triangulaires inférieures à diagonale unité, L est aussi triangulaire inférieure à diagonale unité. Il ne reste plus qu'a montrer l'unicité : soit donc une autre décomposition $A = \widetilde{L}\widetilde{U}$, il vient :

$$\begin{aligned} LU &= \widetilde{L}\widetilde{U} \\ \widetilde{L}^{-1}LU &= \widetilde{U} \\ \widetilde{L}^{-1}L &= \widetilde{U}U^{-1} \end{aligned}$$

D'après les résultats sur les matrices triangulaires (vus en TD) :
- la matrice $\widetilde{L}^{-1}$ est triangulaire inférieure à diagonale unité ;
- la matrice U^{-1} est triangulaire supérieure ;
- il s'ensuit que $\widetilde{L}^{-1}L$ est une matrice triangulaire inférieure à diagonale unité et que la matrice $\widetilde{U}U^{-1}$ est triangulaire supérieure.

Par conséquent l'égalité $\widetilde{L}^{-1}L = \widetilde{U}U^{-1}$ ne peut avoir lieu que si : $\widetilde{L}^{-1}L = \widetilde{U}U^{-1} = I$, soit :

$$\begin{cases} \widetilde{L} = L \\ \widetilde{U} = U \end{cases}$$

Pour le calcul effectif de la matrice L on a un résultat assez fort : la matrice L s'obtient sans aucun calcul supplémentaire puisque :

$$L = I + \sum_{k=1}^{n-1} z^{(k)}(e^k)^\top = I + \left(\begin{array}{c|c|c|c|c} z^{(1)} & z^{(2)} & \dots & z^{(n-1)} & 0 \end{array}\right)$$

ce résultat se montrant en généralisant le calcul suivant : le produit de deux matrices $L^{(k)}L^{(k')}$ avec $k < k'$ est égal à :

$$L^{(k)}L^{(k')} = I + z^{(k)}(e^k)^\top + z^{(k')}(e^{k'})^\top$$

(faire le calcul).

Remarques :

1. Cette première décomposition repose sur l'hypothèse suivante : à chaque étape de la méthode de Gauss on suppose que $a_{kk}^{(k-1)} \neq 0$ (sans avoir à effectuer des échanges de lignes). Par exemple la matrice :

   $$\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$$

   ne convient pas pour cette méthode. On verra cependant que pour certaines matrices qui interviennent dans la pratique, cette condition est vérifiée (la décomposition étant de plus stable vis à vis des erreurs d'arrondi numérique dues aux calculs avec les flottants).

2. Cependant comme *a priori* il semble que l'on ait très peu de chance de tomber pile sur un zéro, cette méthode parait convenir dans la plupart des cas (il suffit de faire un test pour vérifier si le pivot est nul et arrêter la méthode au cas où, en gérant l'exception (message d'erreur, positionnement d'une variable booléenne, etc...)). En fait pour une matrice générale cet algorithme peut être très mauvais car un petit pivot peut amplifier les erreurs d'arrondi. La méthode standard consiste, à chaque étape k, à rechercher d'abord le pivot maximum en valeur absolue (ou module dans le cas complexe) dans la colonne k (à partir de la ligne k) :

$$\max_{i\in[k,n]} \left|a_{ik}^{(k-1)}\right|$$

(que l'on suppose atteind en i_0 par exemple) et à échanger les lignes k et i_0. Matriciellement cela revient à multiplier en premier par une matrice de permutation $P^{(k)}$ avant de procéder à la phase d'élimination (correspondant à la multiplication par la matrice $M^{(k)}$) :

$$A^{(k)} = M^{(k)}P^{(k)}A^{(k-1)}.$$

Cette stratégie est appelée "méthode de Gauss à pivot partiel". Elle conduit à une décomposition du type :

$$PA = LU$$

où P est une matrice de permutation.

2.3.4 Quelques autres résultats théoriques

Théorème 2 : Une condition nécessaire et suffisante pour que A inversible, admette une décomposition $A = LU$ est que toutes les sous-matrices principales de A, $[A]_k$ $1 \leq k \leq n-1$ soient inversibles.
Remarque : la sous-matrice principale d'ordre k (notée $[A]_k$) est la sous-matrice obtenue en ne prenant que les k premières lignes et colonnes de A :

$$[A]_k = (a_{ij})_{\substack{1\leq i\leq k\\ 1\leq j\leq k}}$$

Démonstration : admise. Grâce à ce résultat on peut montrer qu'une matrice à diagonale strictement dominante et qu'une matrice définie positive (cf théorème 5) admettent une telle décomposition.

Théorème 3 (la décomposition $PA = LU$) : Soit A une matrice inversible, alors il existe une matrice triangulaire supérieure U, une matrice triangulaire inférieure à diagonale unité L et une matrice de permutation P telles que :

$$PA = LU$$

Démonstration : admise. Même si au final la programmation de cette décomposition est simple, le travail mathématique à fournir est suffisamment important pour qu'on l'oculte dans ce cours élémentaire. En particulier nous avons vu précédemment qu'une étape revenait à multiplier la matrice obtenue à l'étape précédente par deux matrices (l'une de permutation et l'autre d'élimination) :

$$A^{(k)} = M^{(k)}P^{(k)}A^{(k-1)}.$$

Vous pouvez alors commencer à effectuer la même démarche que pour la décomposition $A = LU$ mais on n'obtient pas directement le résultat cherché !

2.3.5 Utilisation d'une décomposition pour résoudre un système linéaire

Lorsque l'on a obtenu une décomposition $A = LU$, résoudre un système linéaire consiste à résoudre deux systèmes triangulaires : $Ax = b \Leftrightarrow LUx = b$, on pose $Ux = y$ et l'on résoud :

(i) $Ly = b$ on obtient y (résolution d'un système triangulaire inférieur)
(ii) $Ux = y$ on obtient x (résolution d'un système triangulaire supérieur)

Voici l'algorithme que l'on appelle généralement "descente-remontée" :

pour $i = 1, 2, \ldots, n$
  $y_i = \left(b_i - \sum_{j=1}^{i-1} l_{ij} y_j\right)$
fin pour
pour $i = n, n-1, \ldots, 1$
  $x_i = \left(y_i - \sum_{j=i+1}^{n} u_{ij} x_j\right) / u_{ii}$
fin pour

Il est simple de calculer le nombre d'opérations : pour la "descente" le coût de l'itération i est de $i-1$ multiplications et $i-1$ additions/soustractions, soit :

$$C_{descente} = \sum_{i=1}^{n} (i-1) \text{ mult.} + \sum_{i=1}^{n} (i-1) \text{ add.} = \frac{n(n-1)}{2} \text{ mult. } + \frac{n(n+1)}{2} \text{ add.}$$

Pour la remontée, on rajoute en plus n divisions. Au total on obtient (en ne gardant que les termes en $O(n^2)$) :

$$C_{desc/rem} \simeq \frac{1}{2}n^2 \text{ multiplications et} \frac{1}{2}n^2 \text{ additions}$$

Dans le cas d'une décomposition $PA = LU$, la méthode ci-dessus est précédée de l'application de la permutation sur le second membre : $Ax = b \Leftrightarrow PAx = Pb \Leftrightarrow LUx = Pb$.

2.3.6 Coût de la méthode de Gauss/LU

Commençons par expliquer ce qui diffère entre la méthode de Gauss et la décomposition $A = LU$ ou $PA = LU$ d'une matrice. En fait presque rien : pour obtenir une décomposition ($A = LU$ ou $PA = LU$) on procède avec une méthode de Gauss qui agit uniquement sur la matrice (pas d'opérations sur un second membre) et qui **enregistre les coefficients d'élimination successifs** pour former la matrice L (qui s'obtient directement avec ces coefficients), et, dans le cas d'une méthode avec échange de lignes (pivot partiel qui conduit à une décomposition $PA = LU$), on enregistre aussi les permutations successives (en fait un seul tableau d'entiers de dimension n suffit). Dans la méthode de Gauss classique on procède aussi à la fin à la résolution du système triangulaire supérieur. Pour évaluer le coût nous allons écrire ci-dessous un algorithme naïf (on ne fait aucun test sur le pivot) pour la décomposition $A = LU$ qui travaille sur place dans le tableau contenant initialement la matrice A. En sortie la partie triangulaire supérieure contient la matrice U, et la partie triangulaire strictement inférieure, la matrice L (comme la diagonale contient des 1 inutile de les stocker). Ici nous écrivons une forme classique qui permet de compter facilement les opérations (on utilise pas d'expression matricielle comme en langage Scilab) :

pour $k := 1$ **à** $n-1$ la boucle des $n-1$ étapes
  pour $i := k+1$ **à** n les opérations sur les lignes
    $A(i,k) := A(i,k)/A(k,k)$ calcul et stockage du coef d'élimination $z_i^{(k)}$
    pour $j := k+1$ **à** n mise à jour de la ligne i
      $A(i,j) := A(i,j) - A(i,k) \times A(k,j)$
    fin pour
  fin pour
fin pour

Le compte des opérations peut se faire de la façon suivante : à chaque étape k on calcule :

1. les coefficients d'élimination : en tout $n-k$ divisions
2. les modifications sur la matrice courante (opérations sur les lignes) : la boucle interne est effectuée $(n-k)^2$ fois en tout, ce qui nous donne donc : $(n-k)^2$ additions/soustractions[2] et multiplications ;

[2] à partir de maintenant, pour les calculs de complexité, on parlera d'addition même si l'opération est une soustraction : les temps de calcul sont les mêmes.

D'où :

$$C_{LU} = \sum_{k=1}^{n-1}(n-k)^2 \text{ additions et multiplications et } \sum_{k=1}^{n-1}(n-k) \text{ divisions}$$

Pour calculer aisément, on effectue un petit changement de variable en posant $l = n - k$, ce qui donne :

$$\begin{aligned} C_{LU} &= \sum_{l=1}^{n-1} l^2 \text{ additions et multiplications et } \sum_{l=1}^{n-1} l \text{ divisions} \\ C_{LU} &= \frac{n(n-1)(2n-1)}{6} \text{ additions et multiplications et } \frac{n(n-1)}{2} \text{ divisions} \end{aligned}$$

Dans ce calcul, on a utilisé l'identité :

$$\sum_{l=1}^{n} l^2 = (2n^3 + 3n^2 + n)/6 = n(n+1)(2n+1)/6.$$

Pour simplifier un peu, on ne prend en compte que les termes en $O(n^3)$, d'où :

$$C_{LU} \simeq \frac{1}{3}n^3 \text{ additions et multiplications}$$

Le coût de la décomposition $PA = LU$ avec la stratégie du pivot partiel n'est que légèrement supérieur car la recherche du coefficient maximum intervient avec un coût en $O(n^2)$ (de même que les opérations d'échanges de lignes, et le coût total de la mise à jour de la permutation peut se faire avec seulement $O(n)$ opérations). De même le coût de la méthode de Gauss classique, est juste un peu plus élevé puisque les opérations sur le second membre et la résolution finale du système triangulaire supérieur rajoutent simplement environ n^2 multiplications et additions.

2.3.7 A quoi sert la formalisation de la méthode de Gauss en décomposition sur la matrice ?

Dans la pratique, on a souvent à résoudre des systèmes linéaires qui comportent tous la même matrice :

$$Ax^{(i)} = b^{(i)} \qquad 1 \leq i \leq m \tag{2.6}$$

et où les vecteurs $b^{(i)}$ ne sont pas tous connus au même moment[3], par exemple $b^{(i+1)}$ s'obtient avec un calcul qui utilise $x^{(i)}$. Cette situation est très courante dans les problèmes d'ingénierie : en calcul des structures (en régime stationnaire), la matrice A (après discrétisation) dépend uniquement de la structure étudiée alors que le vecteur b modélise les efforts extérieurs apportés sur la structure [4]. On a envie de faire plusieurs calculs en "chargeant" la structure différemment, c-a-d que finalement, on optient typiquement le problème (2.6). Si on utilise bêtement une méthode de Gauss classique à chaque fois, on refait toujours les mêmes opérations sur la matrice A alors que c'est inutile ! On obtient alors un coût calcul d'environ :

$$C_{meth1} \simeq m \times \frac{1}{3}n^3.$$

Alors que si on procède d'abord par une décomposition, puis par m "descentes-remontées" on obtient :

$$C_{meth2} \simeq \frac{1}{3}n^3 + m \times n^2$$

ce qui est beaucoup plus rapide. Par exemple, si on considère que le temps d'accès à une donnée en mémoire est à peu près constant[5], on peut estimer le temps calcul (de la décomposition et de Gauss) avec :

$$T_{decompo} \simeq \frac{1}{3}n^3 \times \tau$$

[3] car il est simple de généraliser la méthode de Gauss classique pour qu'elle s'adapte à plusieurs seconds membres.

[4] Dans ce type de problème la solution x correspond souvent aux déformations aux points d'un maillage qui discrétise la structure ; lorsque celles-ci sont obtenues, un post-traitement peu coûteux permet alors d'avoir les valeurs des contraintes.

[5] ceci est faux du fait des mémoires caches : une donnée en mémoire cache est plus rapidement accessible qu'en mémoire principale ; mais le calcul suivant est quand même représentatif du fait que les algorithmes travaillent sur un tableau du même type à chaque fois.

où τ représente un temps moyen pour faire une addition et une multiplication et les transferts mémoire $\leftrightarrow$ C.P.U. associés, et :

$$T_{desc/rem} \simeq n^2 \times \tau$$

sera le temps calcul pour effectuer une descente-remontée. Le rapport entre les deux est égal à :

$$T_{decompo}/T_{desc/rem} \simeq \frac{n}{3}$$

ainsi avec une matrice 900×900 l'opération de descente-remontée est 300 fois plus rapide que la factorisation ! Voici un petit test avec Scilab sur Oceanos (actuellement la machine la plus puissante du réseau Scinfo) :

```
-->A = rand(900,900);

-->b = rand(900,1);

-->stacksize(2000000);  //pour augmenter la memoire de Scilab

-->timer(); x = A\b ; timer()
 ans  =

    15.85
```

La routine met donc environ 16 sec (correspondant en fait au calcul de $PA = LU$ puis à une descente-remontée). On pourra alors obtenir un temps d'environ :

$$\frac{16}{300} \simeq 0.053 sec$$

pour une descente-remontée. Prenons maintenant $m = 30$, la méthode naïve 1 prendra environ $16 \times 30 = 480$ sec $= 8$ mn alors que la méthode 2 prendra environ $16 + 30 \times 16 / 300 = 17{,}6$ sec. Pour une application critique (par exemple temps réel) une telle différence compte ! Petite remarque : si Scilab permet de récupérer quelques décompositions types, il ne possède pas de primitive de descente-remontée[6] ! Pour cela il faut lier à Scilab un sous-programme f77 ou une fonction C qui effectue ce calcul (c'est pas trop compliqué à faire).

2.4 Deux autres décompositions

On considère maintenant des matrices réelles et symétriques, c-a-d telles que :

$$A^\top = A$$

munies aussi d'autres propriétés. On essaie alors d'adapter la décomposition $A = LU$ à ces spécificités.

2.4.1 La décomposition $A = LDL^\top$

Théorème 4 : Soit A une matrice réelle symétrique inversible dont toutes les sous-matrices principales $[A]_k$ $k \in [1, n-1]$ sont inversibles. Alors il existe une matrice L triangulaire inférieure à diagonale unité et une matrice diagonale D telles que :

$$A = LDL^\top$$

cette décomposition étant unique.

Démonstration : avec l'hypothèse sur les sous-matrices principales, le théorème 2 nous dit que A admet une unique décomposition $A = LU$. On pose $D = diag(u_{ii})$ la matrice diagonale obtenue en prenant la diagonale de U, on a alors :

$$U = DV$$

[6] Il y en quand même une pour les matrices creuses.

où V est une matrice triangulaire supérieure à diagonale unité. Donc $A = LDV$, et en utilisant la symétrie de A il vient :

$$\begin{aligned} A &= A^\top && (2.7)\\ L(DV) &= V^\top D^\top L^\top = V^\top (DL^\top) && (2.8)\\ LU &= \widetilde{L}\widetilde{U} && (2.9) \end{aligned}$$

où l'on a posé $\widetilde{L} = V^\top$ et $\widetilde{U} = DL^\top$. On remarque alors que $\widetilde{L}$ est une matrice triangulaire inférieure à diagonale unité alors que $\widetilde{U}$ est une matrice triangulaire supérieure. Par unicité de la décomposition $A = LU$ on obtient alors :

$$L = \widetilde{L} = V^\top.$$

L'unicité est laissée en exercice.

2.4.2 La décomposition de Cholesky $A = CC^\top$

Théorème 5 : Soit A une matrice réelle symétrique définie positive. Alors il existe une unique matrice triangulaire inférieure C dont tous les éléments diagonaux sont strictement positifs telle que :

$$A = CC^\top.$$

Remarque : une matrice est dite définie positive si et seulement si

$$(Ax|x) = x^\top Ax > 0,\ \forall x \neq 0.$$

Il s'ensuit qu'une telle matrice est nécessairement inversible. Pour cela on remarque que le noyau de A ne contient que le vecteur nul :

$$Ax = 0 \Rightarrow (Ax|x) = (0|x) = 0 \Rightarrow x = 0$$

car sinon on contredit la propriété de définie positivité. D'autre part on peut remarquer que les sous-matrices principales sont aussi définie positive et donc inversibles. En effet, soit $\tilde{x} \in \mathbb{R}^k$, on définit un vecteur x de $\mathbb{R}^n$ en rajoutant à $\tilde{x}$, $n-k$ composantes toutes nulles :

$$x = \left(\begin{array}{c} \tilde{x} \\ \hline 0 \end{array}\right)$$

Si $\tilde{x} \neq 0$ il en est de même pour x, on a alors :

$$(Ax|x) > 0 \Leftrightarrow x^\top Ax > 0 \Leftrightarrow \left(\begin{array}{c|c} \tilde{x}^\top & 0 \end{array}\right) \left(\begin{array}{c|c} [A]_k & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right) \left(\begin{array}{c} \tilde{x} \\ \hline 0 \end{array}\right) > 0 \Leftrightarrow \tilde{x}^\top [A]_k \tilde{x} > 0$$

en effectuant un produit par blocs (excellent exercice à faire).

Démonstration : D'après ce qui précède une matrice définie positive admet donc une décomposition $A = LU$ et comme A est symétrique, elle a possède aussi une décomposition $A = LDL^\top$. Nous allons montrer que les d_{ii} sont strictement positifs. Soit y^i le vecteur tel que :

$$L^\top y^i = e^i,$$

(ce qui ne pose pas de problème car $L^\top$ est inversible). D'autre part on a nécessairement $y^i \neq 0$ et donc :

$$(Ay^i|y^i) > 0.$$

En remplaçant A par $LDL^\top$ il vient :

$$(Ay^i|y^i) = (LDL^\top y^i|y^i) = (y^i)^\top LDL^\top y^i = ((y^i)^\top L)D(L^\top y^i) = (L^\top y^i)^\top D(L^\top y^i) = (e^i)^\top De^i = d_{ii}$$

d'où $d_{ii} > 0$. On pose alors $\Lambda = diag(\sqrt{d_{ii}})$ et donc $D = \Lambda\Lambda = \Lambda\Lambda^\top$ d'où :

$$A = LDL^\top = L\Lambda\Lambda^\top L^\top = (L\Lambda)(L\Lambda)^\top$$

et en posant $C = L\Lambda$ on a bien le résultat cherché. L'unicité est laissée en exercice.

2.5 C'est fini ?

Pour compléter cette introduction aux méthodes de résolution de systèmes linéaires, voici quelques informations pour votre culture générale :

- Très souvent les matrices obtenues en pratique sont creuses, c-a-d qu'elles comportent beaucoup de zéros (cf la matrice du devoir). Pour optimiser le temps de calcul, il faut adapter les algorithmes à ce cas, ce qui peut être assez difficile. D'autre part les matrices ne sont pas stockées dans des tableaux bidimensionnels (inutile de stocker des floppées de zéros). Scilab possède une méthode $PA = LU$ adaptée aux matrices creuses, ainsi qu'une structure spéciale pour le stockage de ces matrices qui convient bien aux problèmes de "taille moyenne".
- Lorsque la taille d'un système linéaire creux est très grande (disons $n > 20000$) les méthodes de type Gauss sont inadéquates (sauf pour des structures creuses très spécifiques) et l'on a plutôt recours à des méthodes itératives qui ne donnent pas une solution exacte avec un nombre fini d'itérations : on arrête les itérations lorsque le norme du résidu :

  $$||Ax^{(k)} - b||$$

  est jugée suffisamment petite.
- En plus des concepts mathématiques de ces méthodes pour matrices creuses, interviennent des méthodes informatiques comme la théorie des graphes qui permettent de les optimiser.

Chapitre 3

Interpolation polynomiale

3.1 Introduction

Le problème (général) de l'interpolation consiste à déterminer une courbe passant par des points donnés dans le plan et issus par exemple de mesures expérimentales. Dans ce chapitre, la courbe cherchée est le graphe d'une fonction et plus exactement d'un polynôme. Comme nous allons le voir, le degré de ce polynôme est déterminé par le nombre de points. Donnons-nous $(n+1)$ points deux à deux distincts $x_0,\ x_1,\ \ldots,\ x_n$ et $(n+1)$ valeurs réelles $y_0,\ y_1,\ \ldots,\ y_n$. On cherche alors un polynôme tel que son graphe passe par les points $(x_i,\ y_i)$ pour $i = 0, \ldots, n$. Plus précisement, le problème de l'interpolation polynomiale consiste à trouver un polynôme de degré au plus égale à n, qui prend la valeur y_i au point x_i (voir la figure 3.1 suivante). Si on note $\mathbb{P}_n$ l'espace vectoriel des polynômes de degré $\leq n$, le problème

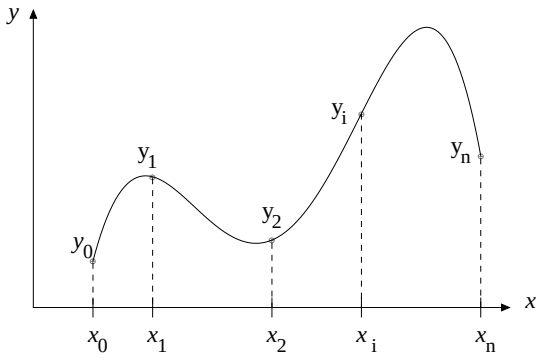

FIG. 3.1 – Exemple d'interpolation polynomiale.

de l'interpolation polynomiale peut alors se formuler de la façon suivante :

$$(3.1) \quad \begin{cases} \text{Trouver } p \in \mathbb{P}_n \text{ vérifiant} \\ p(x_i) = y_i \quad \text{pour } i = 0, 1, \ldots, n. \end{cases}$$

On va d'abord montrer que le problème (3.1) est bien posé ; c'est l'objet du résultat suivant.

Théorème 1 : *Polynôme d'interpolation*

Si les $(n+1)$ points $x_0,\ x_1,\ \ldots,\ x_n$ sont distincts alors il existe un et un seul polynôme p de degré $\leq\ n$ tel que $p(x_i) = y_i$, $i = 0, ..., n$. Ce polynôme est appelé polynôme d'interpolation aux points $(x_i,\ y_i)$.

L'unicité d'un tel polynôme d'interpolation peut s'établir de la façon suivante. Supposons en effet qu'il existe deux polynômes p et q de degré $\leq n$ tels que $p(x_i) = q(x_i) = y_i$, pour $i = 0, \ldots, n$. Par conséquent, les x_i pour $i = 0, \ldots, n$, sont racines du polynômes $p-q$. Donc le polynôme r défini par $r(x) = \prod_{i=0}^{n}(x - x_i)$ divise $p - q$. Or $\deg(r) = n + 1$ et $\deg(p - q) \leq n$, donc nécessairement $p - q \equiv 0$.

La démonstration de l'existence d'un polynôme d'interpolation quant à elle, peut se faire en utilisant soit la base canonique de l'espace vectoriel $\mathbb{P}_n$, soit les polynômes de Lagrange. Les sections suivantes détaillent successivement ces deux approches. Bien que ces méthodes fournissent toutes deux une démonstration de l'existence du polynôme d'interpolation, on verra que d'un point de vue pratique, seule la méthode de Lagrange est à retenir.

3.2 Base canonique - Système de Van-der-Monde

La famille $\{1,\ x,\ x^2,\ \ldots,\ x^n\}$ forme une base de l'espace vectoriel $\mathbb{P}_n$. On cherche alors le polynôme p sous la forme $p(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n$, c'est à dire que l'on cherche à déterminer les valeurs réelles a_i pour $i = 0, 1, \ldots, n$. Le problème de l'interpolation polynomiale (3.1) peut s'écrire sous forme matricielle. On note

$$y = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix}, \qquad a = \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix},$$

les vecteur de $\mathbb{R}^{n+1}$ formés respectivement par les valeurs y_i données et par les coefficients du polynôme p dans la base canonique de $\mathbb{P}_n$. On introduit également la matrice M appelée matrice de Van-der-Monde, à $(n+1)$ lignes et $(n+1)$ colonnes, définie par

$$M = \begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{pmatrix}.$$

Sous forme matricielle, le problème (3.1) s'écrit alors

$$Ma = y \tag{3.2}$$

Le système linéaire (3.2) admet une et une seule solution. En effet, on peut montrer que le déterminant de la matrice M est donné par $\det(M) = \prod_{j<i}(x_i - x_j)$. Puisque les $(n+1)$ points sont supposés distincts, on a nécessairement $\det(M) \neq 0$ et par conséquent la matrice M est inversible.

Remarque : Dans la pratique, on ne résout pas le système (3.2) pour calculer les coefficients a_i du polynôme p car c'est numériquement instable (au sens où des petites perturbations sur les données $(x_i,\ y_i)$ peuvent conduire à des résultats différents) et en plus, le coût de calcul pour résoudre le système (3.2) est en $O\left((n+1)^3\right)$ si on utilise une méthode de Gauss par exemple (voir le chapitre sur les systèmes linéaires).

3.3 Polynôme de Lagrange

Nous allons voir une autre façon de résoudre le problème d'interpolation (3.1) en introduisant des polynômes particuliers appelés polynôme de Lagrange.

Définition : *Polynômes de Lagrange*

Pour les points $x_0,\ x_1,\ \ldots,\ x_n$ donnés et distincts, les n polynômes de Lagrange de degré n sont définis par

$$L_i(x) = \frac{(x-x_0)(x-x_1)\cdots(x-x_{i-1})(x-x_{i+1})\cdots(x-x_n)}{(x_i-x_0)(x_i-x_1)\cdots(x_i-x_{i-1})(x_i-x_{i+1})\cdots(x_i-x_n)} = \prod_{\substack{0 \leq j \leq n \\ j \neq i}} \frac{(x-x_j)}{(x_i-x_j)}, \tag{3.3}$$

pour $i = 0, 1, \ldots, n$ et $x \in \mathbb{R}$.

La fonction L_i est un polynôme de degré n tel que

$$\boxed{L_i(x_j) = \begin{cases} 1 & \text{si } i = j, \\ 0 & \text{si } i \neq j. \end{cases}}$$

Le polynôme de Lagrange L_i est donc le polynôme d'interpolation associé aux points $\{x_0, \ldots, x_n\}$, qui vaut 1 en x_i et 0 ailleurs. Cette propriété est parfois prise comme définition des polynômes de Lagrange. La Figure 3.2 montre un exemple des trois polynômes de Lagrange de degré 2 associés aux points $x_0 = -1$, $x_1 = 0$, $x_2 = 2$.

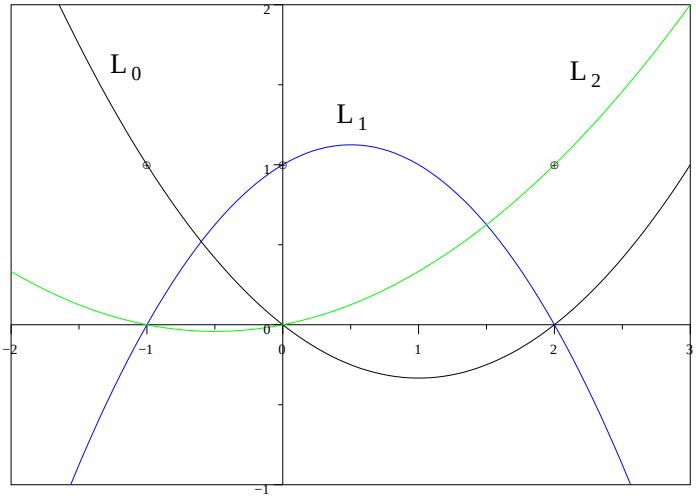

FIG. 3.2 – Les trois polynômes de Lagrange de degré 2 associés à $x_0 = -1$, $x_1 = 0$, $x_2 = 2$.

La famille $\{L_0, L_1, \ldots, L_n\}$ forme une base de l'espace vectoriel $\mathbb{P}_n$. Le polynôme p d'interpolation de Lagrange aux points $(x_i,\ y_i)$ pour $i = 0, \ldots, n$ est alors donné par

$$p(x) = \sum_{i=0}^{n} y_i L_i(x). \tag{3.4}$$

Les polynômes de Lagrange forment une base de $\mathbb{P}_n$ telle que la décomposition du polynôme d'interpolation p dans cette base, est simple (les coefficients sont les y_i connus).

Coût d'évaluation : Pour calculer chaque polynôme de Lagrange (cf. equation (3.3)), on effectue $4n - 1$ opérations. Le calcul de $p(x)$ avec la formule (3.4) nécessite, quant à lui, $2n + 1$ opérations. Au total, le calcul de $p(x)$ pour un x donné, coûte donc $(n + 1)(4n - 1) + 2n + 1 = 4n(n + 5/4) \simeq 4n^2 = O(n^2)$ opérations pour n grand.

Remarque : Il est possible de réduire ce coût en faisant un pré-calcul de $\prod_{0 \leq j \leq n}(x - x_j)$ et d'évaluer les L_i en divisant la quantité précédente par $(x - x_i)$. Mais alors on peut rencontrer des problèmes numériques lorsque x est proche de x_i.

3.4 Base de Newton - Différences divisées

Nous allons voir à présent une méthode efficace (en terme de nombre d'opérations) permettant de trouver une expression du polynôme d'interpolation dans une base particulière de $\mathbb{P}_n$, la base de Newton. On se donne $n + 1$ points distincts $x_0,\ x_1,\ \ldots,\ x_n$ et on considère les images de ces points par une fonction f (continue), c'est-à-dire $y_0 = f(x_0)$, $y_1 = f(x_1)$, $\ldots$, $y_n = f(x_n)$.

Définition : *Base de Newton*

On appelle base de Newton relative à $\{x_0, \ldots, x_n\}$, la base formée des $n+1$ polynômes

$$1, \quad (x-x_0), \quad (x-x_0)(x-x_1), \quad \ldots \quad , \quad (x-x_0)(x-x_1)\ldots(x-x_{n-1}).$$

Cette définition contient en fait un résultat, à savoir que la famille des polynômes en question constitue bien une base de l'espace vectoriel $\mathbb{P}_n$. Tout polynôme $p \in \mathbb{P}_n$ peut s'écrire sous la forme

$$p(x) = a_0 + a_1(x-x_0) + a_2(x-x_0)(x-x_1) + \cdots + a_n \prod_{i=0}^{n-1}(x-x_i), \tag{3.5}$$

où les nombres réels a_i constituent les coefficients de p dans la base de Newton.

Lorsque le polynôme p interpole f aux points $(x_i,\ y_i = f(x_i))$, $i = 0, \ldots, n$, on cherche à déterminer les coefficients a_i de p dans la base de Newton. On voit alors dans ce cas que $a_0 = f(x_0) = y_0$ et $a_1 = \dfrac{f(x_1)-f(x_0)}{x_1-x_0} = \dfrac{y_1-y_0}{x_1-x_0}$. Et la question naturelle est : que valent les a_i pour tout $i = 0, \ldots, n$? Comme nous allons le voir, la réponse est donnée par les différences divisées.

Notation et Définition. *Différences divisées*

Soient $\{x_k\}_{k\geq 0}$ des points deux à deux distincts. Les différences divisées de la fonction f sont définies par récurrence de la façon suivante.

Différence divisée d'ordre 0 : $\quad f[x_i] = f(x_i) = y_i$

Différence divisée d'ordre 1 : $\quad f[x_i, x_j] = \dfrac{f(x_i)-f(x_j)}{x_i-x_j} = \dfrac{y_i-y_j}{x_i-x_j}, \quad i \neq j$

Différence divisée d'ordre $k \neq 0$: $\quad f[x_l, \ldots, x_{l+k}] = \dfrac{f[x_l, \ldots, x_{l+k-1}] - f[x_{l+1}, \ldots, x_{l+k}]}{x_l - x_{l+k}}$ (3.6)

Donnons tout de suite une propriété des différences divisées.

Proposition 1 :

i) On a, pour $k \geq 1$, $\quad f[x_l, \ldots, x_{l+k}] = \displaystyle\sum_{i=l}^{l+k} \frac{f(x_i)}{\displaystyle\prod_{\substack{l \leq j \leq l+k \\ j \neq i}} (x_i - x_j)}$.

ii) Les valeurs des différences divisées ne dépendent pas de l'ordre des points.

La démonstration du point *i)* se fait par récurrence (exercice). Le point *ii)* quant à lui, est une conséquence directe de *i)* et affirme que par exemple $f[x_0, x_1, x_2] = f[x_1, x_0, x_2] = f[x_2, x_1, x_0] = \cdots$.

Exemple : Pour $h > 0$, on a $f[x_0-h, x_0, x_0+h] = \dfrac{f(x_0+h) - 2f(x_0) + f(x_0-h)}{2h^2}$, ce qui correspond à la moitié de la différence finie centrée pour la dérivée seconde de f.

On a en fait le résultat suivant qui fait le lien entre les différences divisées et les coefficients du polynôme d'interpolation dans la base de Newton.

Théorème 2 :

1) *Formule de Taylor discrète.*

Pour $x \neq x_i$, $i = 0, \ldots, n$, on a

$$f(x) = f(x_0) + f[x_0, x_1](x - x_0) + \cdots + f[x_0, \ldots, x_n] \prod_{i=0}^{n-1} (x - x_i) + f[x_0, \ldots, x_n, x] \prod_{i=0}^{n} (x - x_i). \quad (3.7)$$

2) Le polynôme d'interpolation p de degré $\leq n$ de f en $x_0, \ \ldots, \ x_n$ s'écrit

$$p(x) = f(x_0) + f[x_0, x_1](x - x_0) + \cdots + f[x_0, \ldots, x_n] \prod_{i=0}^{n-1} (x - x_i), \quad (3.8)$$

c'est-à-dire que les coefficients a_i de p dans la base de Newton sont exactement les différences divisées i.e. $a_i = f[x_0, \ldots, x_i]$.

Démonstration : 1) La démonstration se fait par récurrence. Pour $n = 0$, par définition on a $f(x) = f(x_0) + f[x_0, x](x - x_0)$. On suppose à présent la formule (3.7) vraie jusqu'au rang $n-1$. Compte tenu de ce que les différences divisées ne dépendent pas de l'ordre des points, on peut écrire :

$$f[x_0, x_1, \ldots, x_n, x] = f[x, x_0, x_1, \ldots, x_n] = \frac{f[x_0, x_1, \ldots, x_n] - f[x, x_0, x_1, \ldots, x_{n-1}]}{x_n - x}$$

d'où on déduit que $f[x_0, x_1, \ldots, x_{n-1}, x] = f[x, x_0, x_1, \ldots, x_{n-1}] = f[x_0, x_1, \ldots, x_n] + (x - x_n) f[x_0, x_1, \ldots, x_n, x]$. On multiplie alors par le produit $\prod_{i=0}^{n-1}(x - x_i)$ pour obtenir

$$f[x_0, x_1, \ldots, x_{n-1}, x] \prod_{i=0}^{n-1} (x - x_i) = f[x_0, x_1, \ldots, x_n] \prod_{i=0}^{n-1} (x - x_i) + f[x_0, x_1, \ldots, x_n, x] \prod_{i=0}^{n} (x - x_i),$$

ce qui établit la formule de Taylor discrète au rang n.

2) On va utiliser le résultat intermédiaire suivant.

Lemme 1 :

Soient α et β deux réels distincts. On considère les deux polynômes d'interpolation q et r de degré $\leq \ n$ associés respectivement à $\{(x_0, y_0), \ldots, (x_{n-1}, y_{n-1}), (\alpha, a)\}$ et $\{(x_0, y_0), \ldots, (x_{n-1}, y_{n-1}), (\beta, b)\}$. Alors le polynôme d'interpolation p de degré $\leq n+1$ associé à $\{(x_0, y_0), \ldots, (x_{n-1}, y_{n-1}), (\alpha, a), (\beta, b)\}$ est donné par $p(x) = \dfrac{(\beta - x)q(x) - (\alpha - x)r(x)}{\beta - \alpha}$.

Démonstration du lemme : Il suffit de vérifier que $\deg(p) \leq n+1$ et que $p(x_i) = y_i$ pour $i = 0, \ldots, n-1$ et $p(\alpha) = a$, $p(\beta) = b$. □

La démonstration du point 2) du Théorème 2 se fait alors par récurrence sur n.
Pour $n = 1$, on a $p(x) = f(x_0) + f[x_0, x_1](x - x_0)$ car alors $\deg(p) \leq 1$ et $p(x_0) = f(x_0)$ et $p(x_1) = f(x_1)$ donc p est bien l'unique polynôme d'interpolation de f en x_0, x_1.

Supposons à présent que le polynôme d'interpolation p_n de degré $\leq n$ de f aux points $(x_0, y_0), \cdots, (x_n, y_n)$ soit donné par $p_n(x) = f(x_0) + f[x_0, x_1](x - x_0) + \cdots + f[x_0, \ldots, x_n] \prod_{i=0}^{n-1}(x - x_i)$. Soit q_n le polynôme d'interpolation de degré $\leq n$ de f aux points $(x_0, y_0), \cdots, (x_{n-1}, y_{n-1}), (x_{n+1}, y_{n+1})$. Par hypothèse de récurrence on a

$$q_n(x) = f(x_0) + f[x_0, x_1](x - x_0) + \cdots + f[x_0, \ldots, x_{n-1}] \prod_{i=0}^{n-2} (x - x_i) + f[x_0, \ldots, x_{n-1}, x_{n+1}] \prod_{i=0}^{n-1} (x - x_i).$$

D'après le Lemme 1, on a

$$\begin{aligned} p_{n+1}(x) &= \frac{(x_{n+1}-x)p_n(x)-(x_n-x)q_n(x)}{x_{n+1}-x_n} \\ &= \frac{(x_{n+1}-x_n)p_n(x)-(x_n-x)(p_n(x)-q_n(x))}{x_{n+1}-x_n} \\ &= p_n(x)+\frac{(x_n-x)}{(x_{n+1}-x_n)}\prod_{i=0}^{n-1}(x-x_i)\Big(f[x_0,\cdots,x_n]-f[x_0,\cdots,x_{n-1},x_{n+1}]\Big). \qquad (3.9) \end{aligned}$$

Par ailleurs, compte tenu du fait que les différences divisées ne dépendent pas de l'ordre des points, on a

$$\begin{aligned} f[x_0,\cdots,x_n,x_{n+1}] &= f[x_n,\cdots,x_0,x_{n+1}] \\ &= \frac{f[x_n,\cdots,x_0]-f[x_1,\cdots,x_0,x_{n+1}]}{x_n-x_{n+1}} \\ &= \frac{f[x_0,\cdots,x_n]-f[x_0,\cdots,x_{n-1},x_{n+1}]}{x_n-x_{n+1}} \end{aligned}$$

et en utilisant (3.9), on obtient $p_{n+1}(x)=p_n(x)+f[x_0,\cdots,x_n,x_{n+1}]\prod_{i=0}^{n}(x-x_i)$. □

3.5 Calcul des différences divisées

On utilise un tableau pour calculer les coefficients du polynôme d'interpolation dans la base de Newton c'est-à-dire - compte tenu du Théorème 2 - pour calculer les différences divisées. Les différences divisées d'ordre k sont calculées à partir des différences divisées d'ordre $k-1$ par la relation de récurrence (3.6) (et non pas en utilisant la Proposition 1). On construit ainsi le tableau suivant :

x_0	$y_0=\boxed{f(x_0)}$											
		$\searrow$										
x_1	$y_1=f(x_1)$	$\rightarrow$	$\boxed{f[x_0,x_1]}$									
		$\searrow$		$\searrow$								
x_2	$y_2=f(x_2)$	$\rightarrow$	$f[x_1,x_2]$	$\rightarrow$	$\boxed{f[x_0,x_1,x_2]}$							
		$\searrow$		$\searrow$		$\searrow$						
x_3	$y_3=f(x_3)$	$\rightarrow$	$f[x_2,x_3]$	$\rightarrow$	$f[x_1,x_2,x_3]$	$\rightarrow$	$\boxed{f[x_0,x_1,x_2,x_3]}$					
$\vdots$	$\vdots$		$\vdots$					$\ddots$				
$\vdots$	$\vdots$	$\searrow$	$\vdots$	$\searrow$						$\searrow$		
x_n	$y_n=f(x_n)$	$\rightarrow$	$f[x_{n-1},x_n]$	$\rightarrow$	$f[x_{n-2},x_{n-1},x_n]$	$\cdots$	$\cdots$	$\cdots$	$\rightarrow$	$\boxed{f[x_0,\ldots,x_n]}$		

Les coefficients de p dans la base de Newton sont lus en haut sur la diagonale. Si on ajoute un point supplémentaire $(x_{n+1},\ y_{n+1})$ il faut effectuer $n+1$ calculs en plus pour obtenir $f[x_0,\ldots,x_n,x_{n+1}]$.

Exemple : Déterminer les coefficients dans la base de Newton du polynôme d'interpolation de degré 3 tel que $p(-2)=-31$, $p(0)=-1$, $p(1)=-1$ et $p(4)=83$.

		a_0	a_1	a_2	a_3
x_i	y_i	$\swarrow$	$\swarrow$	$\swarrow$	$\swarrow$
-2	**-31**				
0	-1	**15**			
1	-1	0	**-5**		
4	83	28	7	**2**	

On obtient ainsi $p(x)=-31+15(x+2)-5(x+2)x+2(x+2)x(x-1)$. □

Une fois les coefficients du polynôme obtenus, il faut évaluer la valeur de ce polynôme en une valeur x donnée. On utilise pour cela le schéma d'Horner.

3.5.1 Evaluation du polynôme - Schéma d'Horner

Dans la base de Newton, on a $p(x) = a_0 + a_1(x-x_0) + \cdots + a_n(x-x_0)\dots(x-x_{n-1})$. On veut calculer $p(x)$ pour une valeur x donnée. Si on calcule par différences divisées les coefficients de p dans la base de Newton et qu'on évalue ensuite le polynôme en x, le nombre d'opérations effectuées est alors le suivant :

- Evaluation des coefficients :

1ère colonne : $2n$ soustractions + n divisions = $3n$ opérations.

2ème colonne : $3(n-1)$ opérations.

$\vdots$

n-ième colonne : 3 opérations.

Le coût total pour calculer les coefficients dans la base de Newton par différence divisées est donc de $3\left(n + (n-1) + \cdots 1\right) = \dfrac{3n(n+1)}{2} \simeq \dfrac{3n^2}{2}$ opérations pour n grand.

- Evaluation du polynôme par un schéma de Horner : Pour évaluer le polynôme p en un point x donné, on écrit l'expression de p dans la base de Newton sous la forme suivante (schéma de Horner) :

$$\boxed{p(x) = \left(\cdots\left(\left((x-x_{n-1})a_n + a_{n-1}\right)(x-x_{n-2}) + a_{n-2}\right)(x-x_{n-3}) + \cdots\right)(x-x_0) + a_0.}$$

On effectue ainsi $2n$ additions et n multiplications, soit un coût de $3n$ opérations pour le schéma d'Horner.

Le nombre total d'opérations effectuées pour calculer dans la base de Newton la valeur du polynôme p en un point x donné, est donc $\dfrac{3n(n+1)}{2} + 3n = \dfrac{3n(n+3)}{2} \simeq \dfrac{3n^2}{2} = O(n^2)$ opérations pour n grand.

Remarque : Une évaluation directe de $p(x)$ à partir de l'expression (3.5) (une fois les coefficients connus) requière $O(n^2)$ opérations. D'autres schémas moins coûteux (et différents de Horner) existent pour évaluer un polynôme. Par exemple, une alternative possible au schéma d'Horner est l'algorithme suivant :

$p \leftarrow a_0, \quad y \leftarrow 1$
`Pour` i `de 1 à` n
  $y \leftarrow y * (x - x_{i-1})$
  $p \leftarrow p + a_i * y$

On voit alors que le coût est de $4n$ opérations (contre $3n$ pour Horner).

3.6 Erreur d'interpolation

On va à présent analyser l'erreur que l'on commet lorsqu'on remplace une fonction f par son polynôme d'interpolation p associé aux points $\{x_0, \dots, x_n\}$ distincts. D'après le Théorème 2 de la section 3.4, on sait que $f(x) - p(x) = f[x_0, x_1, \dots, x_n, x] \prod_{i=0}^{n}(x - x_i)$ et grâce à la Proposition 1, on peut obtenir une expression de $f[x_0, x_1, \dots, x_n, x]$. Malheureusement celle-ci fait intervenir la valeur $f(x)$ qu'on ne connait pas. Le théorème suivant fournit une estimation de la différence.

Théorème 3 : *Erreur d'interpolation*

Soit f une fonction $(n+1)$ fois dérivable et soit p son polynôme d'interpolation de degré n associés aux points $x_0, \dots, x_n$ distincts. Alors pour tout $x \in \mathbb{R}$, il existe un réel $\theta_x \in]\min(x, x_i), \max(x, x_i)[$ tel que

$$f(x) - p(x) = \frac{1}{(n+1)!}\Pi_{n+1}(x)\, f^{(n+1)}(\theta_x), \qquad \text{avec} \qquad \Pi_{n+1}(x) = \prod_{i=0}^{n}(x - x_i).$$

Démonstration. Si $x = x_i$, alors $\Pi_{n+1}(x) = 0$ et la formule est juste. Fixons à présent $x \neq x_i$ et considérons q le polynôme d'interpolation de f en x, x_0, ..., x_n. On a évidemment $f(x) - p(x) = q(x) - p(x)$ et $q - p$ est un polynôme de degré $\leq n+1$ qui s'annule aux $(n+1)$ points x_0, ..., x_n. Par conséquent, il existe une constante α telle que $q(t) - p(t) = \alpha \prod_{i=0}^{n}(t - x_i) = \alpha \Pi_{n+1}(t)$, pour tout $t \in \mathbb{R}$. Il reste à montrer que $\alpha = \dfrac{1}{(n+1)!} f^{(n+1)}(\theta_x)$. Posons $r(t) = f(t) - q(t) = f(t) - p(t) - \alpha \Pi_{n+1}(t)$. On remarque que la fonction r s'annule $(n+2)$ fois en x, x_0, x_1, ..., x_n. En appliquant $(n+1)$ fois le théorème de Rolle, on en déduit que la dérivée r' s'annule $(n+1)$ fois dans l'intervalle $I =]\min(x, x_i), \max(x, x_i)[$. On peut à nouveau appliquer n fois le théorème de Rolle et ainsi de suite ... De cette façon, en appliquant par récurrence le théorème de Rolle, on obtient que la $(n+1)$-ième dérivée $r^{(n+1)}$ s'annule une fois dans I, c'est-à-dire qu'il existe $\theta_x \in I$ tel que $r^{(n+1)}(\theta_x) = 0$. On en déduit que

$$0 = r^{(n+1)}(\theta_x) = f^{(n+1)}(\theta_x) - p^{(n+1)}(\theta_x) - \alpha \Pi_{n+1}^{(n+1)}(\theta_x).$$

Or $p^{(n+1)} \equiv 0$ car $\deg(p) \leq n$ et $\Pi_{n+1}^{(n+1)} \equiv (n+1)!$. On obtient donc $\alpha = \dfrac{1}{(n+1)!} f^{(n+1)}(\theta_x)$ ce qui prouve le théorème. □

3.7 Problème de convergence de l'interpolation

On a vu à la section précédente qu'on pouvait estimer l'erreur entre une fonction f et son polynôme d'interpolation. La question qu'on se pose à présent est de savoir si l'erreur d'interpolation diminue lorsqu'on augmente le nombre n de points d'interpolation et à la limite quand n tend vers l'infini, est-ce-que p converge (en un sens à préciser) vers f ?

D'après le Théorème 3, l'erreur d'interpolation dépend essentiellement de deux termes ; d'une part de la fonction Π_{n+1} qui ne dépend que de la répartition des points x_i et non de f et d'autre part de la dérivée $(n+1)$-ième de f qui au contraire ne dépend pas des x_i. On se place désormais sur un intervalle $[a, b]$ avec $a = x_0 < x_1 < \cdots < x_n = b$. On peut alors estimer la fonction Π_{n+1} pour des abscisses x_i quelconques.

Lemme 2 : Pour des points $x_0, \cdots, x_n$ quelconques, on a l'estimation

$$\max_{x \in [a,b]} |\Pi_{n+1}(x)| \leq h^{n+1} n! \quad \text{où} \quad h = \max_i h_i \text{ avec } h_i = |x_{i+1} - x_i|.$$

Démonstration. Pour x fixé, on considère l'indice k tel que $x \in [x_k, x_{k+1}]$. On a ainsi $|\Pi_{n+1}(x)| = \prod_{i=0}^{n} |x - x_i| \leq \prod_{i=0}^{k} |x_{k+1} - x_i| \times \prod_{j=k+1}^{n} |x_j - x_k|$. Par ailleurs, on a :

$$\begin{aligned}
\text{pour } i = 0, \cdots, k, \qquad |x_{k+1} - x_i| &\leq |x_{k+1} - x_k| + |x_k - x_{k-1}| + \cdots + |x_{i+1} - x_i| \\
&\leq (k + 1 - i)h \\
\text{pour } j = k+1, \cdots, n, \qquad |x_j - x_k| &\leq |x_j - x_{j-1}| + |x_{j-1} - x_{j-2}| + \cdots + |x_{k+1} - x_k| \\
&\leq (j - k)h
\end{aligned}$$

On obtient ainsi

$$\begin{aligned}
|\Pi_{n+1}(x)| &\leq \prod_{i=0}^{k} \Big((k+1-i)h\Big) \times \prod_{j=k+1}^{n} \Big((j-k)h\Big) \\
&\leq h^{k+1} \prod_{i=0}^{k}(k+1-i) \times h^{n-k} \prod_{j=k+1}^{n}(j-k) \qquad (3.10)
\end{aligned}$$

On a, de plus

$$\begin{aligned}
\prod_{i=0}^{k}(k+1-i) &= (k+1) \times k \times (k-1) \times \cdots \times 2 \times 1 = (k+1)! \qquad (3.11) \\
\prod_{j=k+1}^{n}(j-k) &= 1 \times 2 \times 3 \times \cdots \times (n-k) = (n-k)! \\
&\leq 1 \times (2+k) \times (3+k) \times \cdots \times n = \frac{n!}{(k+1)!} \qquad (3.12)
\end{aligned}$$

Ainsi, par les relations (3.10)-(3.11)-(3.12), on obtient bien l'estimation cherchée. □

Pour une fonction f qui est $n+1$ fois dérivable, on considère son polynôme d'interpolation p aux points $\{x_0, \ldots, x_n\}$ et d'après le Théorème 3, on a

$$\max_{x\in[a,b]} |f(x)-p(x)| \leq \frac{1}{(n+1)!} \max_{x\in[a,b]} |\Pi_{n+1}(x)| \max_{x\in[a,b]} \left|f^{(n+1)}(x)\right|.$$

Compte tenu du Lemme 2, on obtient alors l'estimation d'erreur suivante :

$$\boxed{\max_{x\in[a,b]} |f(x)-p(x)| \leq \frac{h^{n+1}}{(n+1)} \max_{x\in[a,b]} \left|f^{(n+1)}(x)\right|.} \tag{3.13}$$

On fait remarquer que le paramètre h dépend de n.

On va à présent s'intéresser à des distributions particulières des points x_i afin de pouvoir exploiter l'estimation d'erreur (3.13).

3.7.1 Points d'interpolation équidistants

Commençons par étudier le cas de points équidistants. On considère une subdivision régulière de l'intervalle $[a,b]$ en $(n+1)$ points $(n \geq 1)$:

$$x_0 = a, \quad x_1 = a+h, \quad x_2 = a+2h, \quad \ldots \quad , \quad x_n = a+nh, \quad \text{avec } h = \frac{b-a}{n}.$$

Dans ce cas, on a $\dfrac{h^{n+1}}{(n+1)} = \dfrac{1}{(n+1)}\left(\dfrac{b-a}{n}\right)^{n+1} \longrightarrow 0$ quand $n \to +\infty$. Compte tenu de l'estimation d'erreur (3.13), on serait donc tenté de penser que l'erreur entre f et son polynôme d'interpolation p tend vers 0 quand $n \to +\infty$. Mais attention car la $(n+1)$-ième dérivée de f dépend de n et en fait peut croître très rapidement avec n. En général, p ne converge pas vers f lorsque n tend vers $+\infty$. L'exemple suivant illustre une telle situation de non-convergence.

Exemple : *(Runge)*
On considère la fonction $f(x) = \dfrac{1}{1+x^2}$ sur l'intervalle $[-5,5]$. On note p_n le polynôme d'interpolation de f aux $n+1$ points équidistants dans l'intervalle $[-5,5]$. On observe alors (cf. Figure 3.3) quand n augmente, des problèmes d'oscillations aux extrémités de l'intervalle. En fait $\left|f^{(n)}(5)\right|$ devient rapidement grand avec n. On montre que pour $|x| \geq 3.83\cdots$, on a $|f(x)-p(x)| \to +\infty$ quand $n \to +\infty$. □

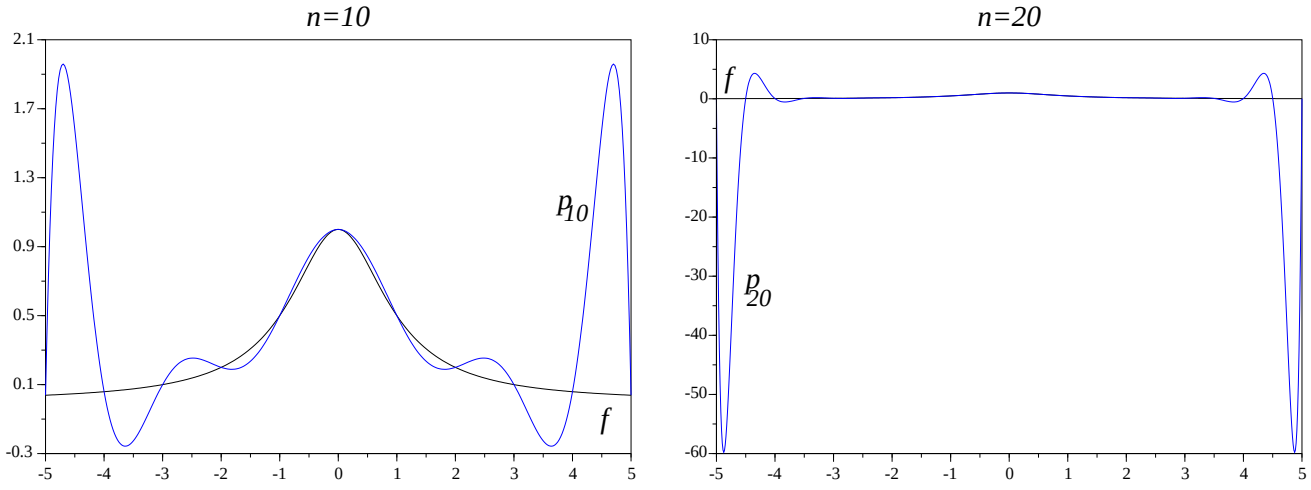

FIG. 3.3 – Phénomène de Runge : "explosion" de l'interpolation polynomiale avec points équidistants.

Nous venons de voir qu'en général, il n'y a pas de convergence du polynôme d'interpolation lorsqu'on choisit les points d'interpolation répartis de façon uniforme dans un intervalle fermé borné. Mais existe-t-il une répartition (évidemment non uniforme) des points d'interpolation pour lesquels il y convergence ? Une réponse est fournie par les abscisses de Tchebichev.

3.7.2 Abscisses de Tchebichev

Le problème est le suivant : étant donnés l'intervalle $[-1, 1]$ et un entier $n \in \mathbb{N}$, trouver $n+1$ points $\{x_0, x_1, \ldots, x_n\}$ distincts qui minimisent $\|\Pi_{n+1}\|_\infty = \max_{x\in[-1,1]} |\Pi_{n+1}(x)|$.
L'existence de ces points est donnée par le théorème suivant.

Théorème 4 : *Abscisses de Tchebichev*

1) La fonction $T_{n+1}(x) = \cos((n+1)\arccos(x))$ définie pour $x \in [-1, 1]$ est un polynôme de degré $n+1$ en x, appelé polynôme de Tchebichev. Ce polynôme admet exactement $n+1$ racines distinctes dans $[-1, 1]$, appelées abscisses de Tchebichev.

2) Les $(n+1)$ racines du polynôme de Tchebichev T_{n+1} minimisent la quantité $\|\Pi_{n+1}\|_\infty$.

On montre facilement par récurrence que les T_n vérifient

$$\begin{cases} T_0(x) = 1, \qquad T_1(x) = x, \\ T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x). \end{cases}$$

Ces relations permettent d'établir que T_{n+1} est bien un polynôme de degré $n+1$. Cherchons à présent les racines de ce polynôme. On cherche les x_i tels que $\cos((n+1)\arccos(x_i)) = 0$. On a donc $(n+1)\arccos(x_i) = \frac{\pi}{2} + i\pi = \frac{(2i+1)\pi}{2}$ pour $i \in \mathbb{Z}$, d'où $x_i = \cos\left(\frac{(2i+1)\pi}{2(n+1)}\right)$. On voit en fait qu'il y a $n+1$ racines distinctes et puisque $\deg(T_{n+1}) = n+1$, ce sont nécessairement les seules racines.

Ainsi, les $n+1$ abscisses de Tchebichev sont données par

$$\boxed{x_i = \cos\left(\frac{(2i+1)\pi}{2(n+1)}\right), \quad i = 0, \ldots, n.}$$

Exemple : Pour $n = 2$, les 3 abscisses de Tchebichev sont $x_0 = \cos(\frac{\pi}{6}) = \frac{\sqrt{3}}{2}$, $x_1 = \cos(\frac{\pi}{2}) = 0$, $x_0 = \cos(\frac{5\pi}{6}) = -\frac{\sqrt{3}}{2}$. □

Les points x_i sont répartis symétriquement autour de 0 (car $x_{n-i} = -x_i$) et de façon plus dense au voisinage de 1 et -1. En fait, lorsque n croît, les abscisses se concentrent autour des bords de l'intervalle (cf. Figure 3.4)

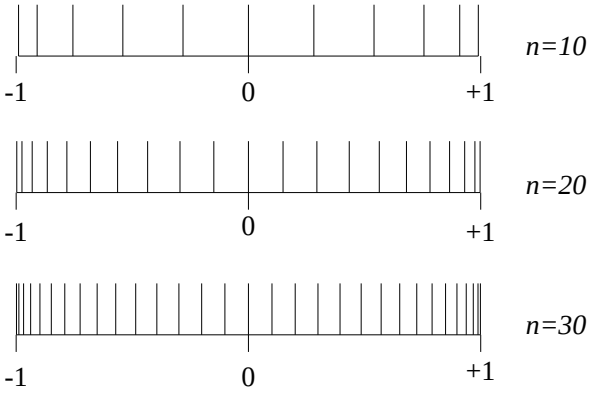

FIG. 3.4 – Abscisses de Tchebichev.

Abscisses de Tchebichev sur un intervalle $[a,b]$: Il suffit de prendre $t_i = \dfrac{(a+b)}{2} + \dfrac{(b-a)}{2} x_i$, où les x_i sont les abscisses de Tchebichev données sur l'intervalle $[-1,1]$.

Revenons maintenant au problème de la convergence du polynôme d'interpolation. En choisissant les abscisses de Tchebichev comme points d'interpolation, on sait maintenant que la quantité $\|\Pi_{n+1}\|_\infty$ est la plus petite possible donc converge vers 0 quand $n \to +\infty$ puisque c'est le cas avec des points équidistants. Mais est-ce suffisant pour assurer la convergence ? Si la $(n+1)$-ième dérivée de la fonction f ne croît pas trop vite avec n alors la réponse est positive. La figure 3.5 montre ce que donne les abscisses de Tchebichev avec l'exemple de Runge (cf. section 1.8.1).

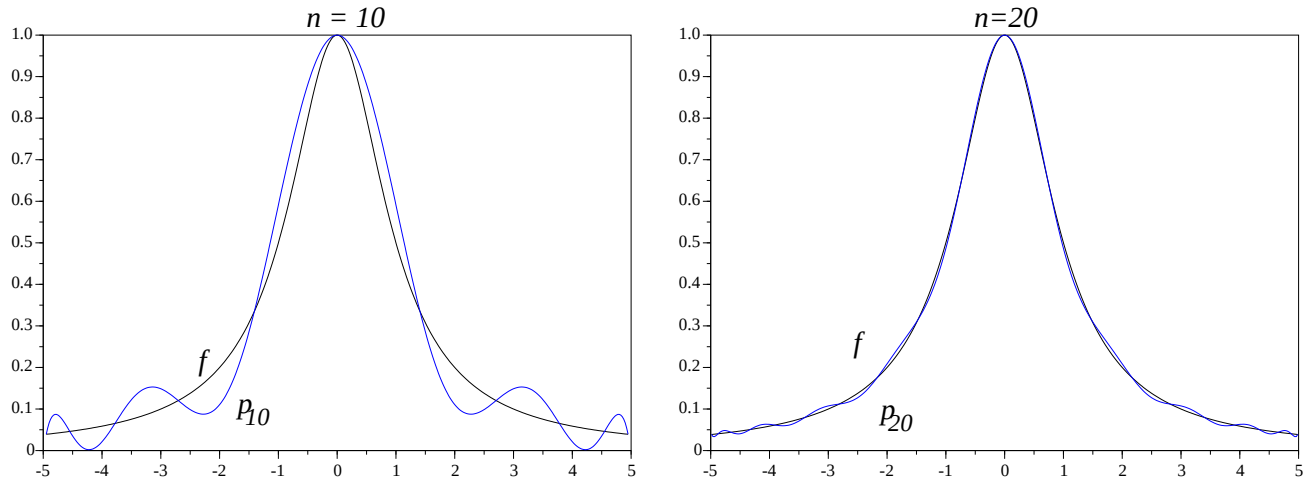

FIG. 3.5 – Interpolation avec les abscisses de Tchebichev.

En fait, on peut montrer (théorème de Faber) que si f est analytique sur $[a,b]$ (f développable en série entière), alors le polynôme d'interpolation de f associé aux abscisses de Tchebichev converge uniformément vers f sur l'intervalle $[a,b]$.

Remarque : Si on ordonne de façon croissante (ou décroissante) les abscisses de Tchebichev et qu'on utilise les différences divisées avec un schéma de Horner pour calculer le polynôme d'interpolation, des instabilités numériques peuvent alors apparaître autour des extrémités de l'intervalle, lorsque le nombre de points n est grand (du fait notamment de soustractions de quantités très proches : cf. Chapitre 1, Arithmétique flottante). Une façon d'y remédier est de répartir les abscisses de Tchebichev alternativement de gauche à droite de l'origine.

3.8 Interpolation de Lagrange-Hermite

Dans certains problèmes, on est amené à chercher un polynôme qui interpole une fonction f en des points donnés (interpolation de Lagrange) ainsi que les dérivées en ces points (pentes données). C'est l'interpolation de Lagrange-Hermite. Plus précisement, on se donne des triplets (x_i, y_i, y'_i), pour $i = 0, \ldots, n$, où $y_i = f(x_i)$ et $y'_i = f'(x_i)$ sont connus (f' désigne la dérivée de f). On cherche alors un polynôme p (polynôme d'interpolation de Lagrange-Hermite) tel que :

$$\begin{cases} p(x_i) = y_i \\ p'(x_i) = y'_i, \quad \text{pour } i = 0, \ldots, n. \end{cases}$$

On voit clairement qu'on dispose de $2(n+1)$ équations. Il faut donc $2(n+1)$ inconnues et par conséquent on cherche un polynôme de degré $\leq 2n+1$.

Le polynôme d'interpolation de Lagrange-Hermite est donné par

$$p(x) = \sum_{i=0}^{n} y_i H_i(x) + \sum_{i=0}^{n} y'_i K_i(x), \tag{3.14}$$

où les polynômes de Lagrange-Hermite H_i et K_i sont définis par

$$\begin{aligned} H_i(x) &= \big(1 - 2(x - x_i)L'_i(x_i)\big)\, L_i^2(x), \\ K_i(x) &= (x - x_i)L_i^2(x) \end{aligned}$$

et ceci pour $i = 0, \ldots, n$. On vérifie que H_i et K_i sont bien des polynômes de degré $2n + 1$.

3.9 Interpolation polynomiale par morceaux

3.9.1 Interpolation de Lagrange par morceaux

Les inconvénients majeurs de l'interpolation polynomiale de Lagrange, sont d'une part le coût élevé si le nombre n de points d'interpolation est grand (coût en $O(n^2)$ pour la base de Newton avec un schéma d'Horner) et d'autre part des effets de bords mal maitrisés. Une alternative possible consiste à effectuer des interpolations par morceaux. Supposons qu'on veuille interpoler une fonction f sur un intervalle $[a, b]$. L'interpolation par morceaux consiste - comme son nom l'indique - à interpoler f sur des sous-intervalles de $[a, b]$. On limite le degré m d'interpolation ($m = 2$ par exemple) dans chaque sous-intervalle et on regroupe les données. De cette façon, on obtient un raccord de continuité C^0 pour l'interpolation (cf. Figure 3.6).

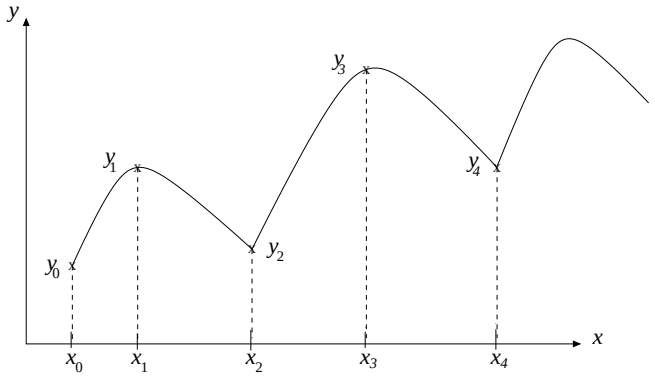

FIG. 3.6 – Interpolation de Lagrange par morceaux.

Convergence de l'interpolation de Lagrange par morceaux.

On va montrer que l'interpolation par morceaux est un processus convergent contrairement à l'interpolation polynômiale *globale*. Considérons pour cela un nombre entier m **fixé** ($m = 1, 2$ ou 3) et subdivisons l'intervalle $[a, b]$ en sous-intervalles de la forme $I_i = [x_{im}, x_{(i+1)m}]$. Soit alors p_i le polynôme de degré $\leq m$ qui interpole f sur le sous-intervalle I_i. D'après l'estimation (3.13) sur l'erreur d'interpolation, on a $\max_{x\in I_i} |f(x) - p_i(x)| \leq \dfrac{h^{m+1}}{(m+1)} \max_{x\in[a,b]} \left|f^{(n+1)}(x)\right|$. où $h = \max_i |x_{i+1} - x_i|$. Si on note p la fonction définie sur tout $[a, b]$ et obtenue par recollement des polynômes p_i, on obtient

$$\max_{x\in[a,b]} |f(x) - p(x)| \leq \frac{h^{m+1}}{(m+1)} \max_{x\in[a,b]} \left|f^{(n+1)}(x)\right|. \tag{3.15}$$

Par conséquent, on obtient bien la convergence (uniforme) de p vers f lorsque h tend vers 0 car m est désormais fixe. Le fait que h tende vers zéro traduit le fait que le nombre de sous-intervalles sur lesquels on effectue des interpolations par morceaux tendent vers l'infini tout en maintenant m points dans chacun

de ces sous-intervalles. L'estimation (3.15) montre que l'erreur d'interpolation par morceaux est d'ordre $\mathcal{O}(h^{m+1})$ où m est le degré des polynômes d'interpolation. On fera enfin remarquer qu'on ne fait aucune hypothèse sur la répartition des points d'interpolation et que ceux-ci peuvent très bien être choisis de façon équidistante.

3.9.2 Interpolation de Lagrange-Hermite par morceaux

L'inconvénient de l'interpolation de Lagrange par morceaux est de fournir une interpolation seulement continue (raccord C^0). En conservant l'idée d'interpolation par morceaux, on peut améliorer ce raccord avec l'interpolation de Lagrange-Hermite.
La méthode consiste à faire de l'interpolation de Lagrange-Hermite non pas sur tout l'intervalle $[a, b]$ mais sur chaque sous-intervalle $[x_i, x_{i+1}]$. On obtient ainsi des polynômes d'interpolation de degré 3 sur chacun des sous-intervalles $[x_0, x_1]$, $[x_1, x_2]$, ..., $[x_{n-1}, x_n]$. Puis on regroupe les données. Puisqu'on a un raccordement de polynômes de degré 3 ainsi que de leurs dérivées, on obtient globalement un raccord C^1 pour l'interpolation (continue ainsi que la dérivée).

Evaluation du polynôme d'interpolation de Lagrange-Hermite.

Il est possible d'utiliser l'interpolation de Lagrange et les différences divisées pour déterminer le polynôme d'interpolation de Lagrange-Hermite sur un sous-intervalle de la forme $[x_i, x_{i+1}]$. En effet, plaçons nous pour fixer les idées, sur le sous-intervalle $[x_0, x_1]$. On cherche le polynôme d'interpolation de degré 3 tel que

$$\begin{aligned} p(x_0) = y_0, \quad p(x_1) = y_1, \\ p'(x_0) = y'_0, \quad p'(x_1) = y'_1, \end{aligned}$$

où on suppose que $y_i = f(x_i)$ et $y'_i = f'(x_i)$ c'est-à-dire qu'on connait les valeurs d'une fonction f ainsi que les dérivées aux points x_i. On introduit alors 2 points supplémentaires $\varepsilon < \varepsilon'$ dans le sous-intervalle $[x_0, x_1]$ et destinés à tendre respectivement vers x_0 et x_1. On considère alors le polynôme d'interpolation de Lagrange de f aux points x_0, ε, ε', x_1 (on oublie Hermite). On peut évaluer ce polynôme par différences divisées puis ensuite faire tendre ε vers x_0 et ε' vers x_1. On remarque que $f[x_0, \varepsilon] = \dfrac{f(\varepsilon) - y_0}{\varepsilon - x_0} \longrightarrow f'(x_0) = y'_0$ quand $\varepsilon \to x_0$. De même, on a $f[\varepsilon', x_1] = \dfrac{f(\varepsilon') - y_1}{\varepsilon' - x_1} \longrightarrow y'_1$ quand $\varepsilon' \to x_1$. On peut alors calculer les différences divisées dans un tableau où on a "doublé" les points (x_0, y_0) et (x_1, y_1).

Exemple : Cherchons le polynôme de degré 3 tel que

$$\begin{aligned} p(0) = 1, \quad p(1) = 0, \\ p'(0) = 0, \quad p'(1) = 2. \end{aligned}$$

On calcule alors les différences divisées par le tableau suivant :

x_i	y_i				y'_0 donné (non calculé)
0	**1**				
0	1	**0**			
1	0	-1	**-1**		
1	0	2	3	**4**	
		y'_1 donné (non calculé)			

On obtient ainsi $p(x) = 1 + 0 \times (x - 0) - 1 \times (x - 0)^2 + 4 \times (x - 0)^2(x - 1) = 1 - x^2 + 4x^2(x - 1)$.

□

L'avantage de cette méthode est de fournir une interpolation plus régulière que l'interpolation de Lagrange par morceaux mais en revanche elle requière la connaissance des dérivées y'_i. Or dans la pratique, ces quantités ne sont pas toujours données ou connues.

3.10 Splines cubiques

On peut en fait s'affranchir de la donnée des dérivées y_i' de la fonction f aux points x_i. De plus, on peut augmenter le degré de régularité de l'interpolation par morceaux, pour obtenir un raccord C^2 (continuité des dérivées secondes de l'interpolation). C'est ce qui est réalisé par les splines cubiques.

Pour $(x_i,\, y_i)$, $i = 0, \ldots, n$ donnés, il s'agit de chercher une fonction $s \in C^2$ telle que $s(x_i) = y_i$ pour $i = 0, \ldots, n$ et telle que sa restriction s_i sur l'intervalle $[x_i, x_{i+1}]$ soit un polynôme vérifiant

- *Raccordement C^2 :*
$$\begin{cases} s_{i-1}'(x_i) = s_i'(x_i) \\ s_{i-1}''(x_i) = s_i''(x_i), \quad \text{pour } i = 1, \ldots, n-1 \end{cases}$$

- *Conditions aux bords :*
$$\begin{cases} s_0''(x_0) = s_{n-1}''(x_n) = 0 & \text{(conditions naturelles)} \\ \quad \text{ou bien} \\ s_0'(x_0) = y_0' \text{ donné},\ s_{n-1}'(x_n) = y_n' \text{ donné} & \text{(condition d'Hermite)} \\ \quad \text{ou bien} \\ s_0'(x_0) = s_{n-1}'(x_n) \text{ et } s_0''(x_0) = s_{n-1}''(x_n) & \text{(conditions périodiques)} \\ \quad \text{ou bien} \\ s_0^{(3)}(x_1) = s_1^{(3)}(x_1) \text{ et } s_{n-2}^{(3)}(x_{n-1}) = s_{n-1}^{(3)}(x_{n-1}) & \text{(conditions "not-a-knot")} \\ \quad \text{i.e. raccord } C^3 \text{ en } x_1 \text{ et } x_{n-1}. \end{cases}$$

On voit alors que pour qu'un tel polynôme existe et soit unique, il faut chercher s_i comme un polynôme de degré 3 sur $[x_i, x_{i+1}]$[1]. En fait, on montre qu'un tel polynôme existe bien et est unique. La fonction s obtenue en "recollant" les polynômes s_i sur chacun des intervalles $[x_i, x_{i+1}]$ est alors par construction une fonction de classe C^2 qui prend les valeurs y_i aux points x_i.

3.10.1 Calcul numérique des splines cubiques (conditions naturelles)

On se place désormais sur l'intervalle $[a, b]$ et on considère $(n+1)$ points $\{x_i\}_{i=0}^n$ tels que $a = x_0 < x_1 < \cdots < x_n = b$. A ces $(n+1)$ points, on associe $(n+1)$ valeurs $y_0, \cdots, y_n$. On cherche alors la spline cubique s avec des conditions naturelles i.e.

$$\boxed{\begin{array}{l} s \in C^2([a,b]) \text{ et } s_i \equiv s_{|_{[x_i, x_{i+1}]}} \in \mathbb{P}_3 \text{ pour } i = 0, ..., n-1 \text{ telle que} \\ \\ \quad \begin{cases} s(x_i) = y_i, \text{ pour } i = 0, ..., n \\ s''(a) = s''(b) = 0. \end{cases} \end{array}}$$

On note $h_i = x_{i+1} - x_i$ et $M_i = s_i''(x_i)$. Puisque les dérivées secondes de s se raccordent (continues en x_{i+1}), on a $s_i''(x_{i+1}) = s_{i+1}''(x_{i+1})$ et par conséquent $M_{i+1} = s_i''(x_{i+1})$.
Par ailleurs sur $[x_i, x_{i+1}]$, s_i est un polynôme de degré 3 et donc s_i'' est un polynôme de degré 1 donné par

$$s_i''(x) = M_i \frac{(x_{i+1} - x)}{h_i} - M_{i+1} \frac{(x_i - x)}{h_i},$$

pour $x \in [x_i, x_{i+1}]$. En intégrant deux fois, on obtient

$$s_i(x) = M_i \frac{(x_{i+1} - x)^3}{6h_i} - M_{i+1} \frac{(x_i - x)^3}{6h_i} + C_i(x_i - x) + D_i(x_{i+1} - x),$$

où les paramètres C_i et D_i sont déterminés par les relations d'interpolation :

$$y_i = M_i \frac{h_i^2}{6} + D_i h_i, \quad y_{i+1} = M_{i+1} \frac{h_i^2}{6} - C_i h_i.$$

[1] La continuité fournit $2n$ équations, les raccords des dérivées premières et secondes en fournissent $2(n-1)$ et enfin les conditions de bords en donnent 2, soit au total $4n$ équations. Si on cherche des polynômes de degré p sur chaque sous-intervalle, il y a $n(p+1)$ inconnues. Pour qu'il y ait autant d'équations que d'inconnues, il faut et il suffit que $p = 3$.

On peut déterminer les dérivées premières des s_i et les évaluer en x_{i+1} :

$$s_i'(x_{i+1}) = M_{i+1}\frac{h_i}{2} - \frac{1}{h_i}\left(y_i - M_i\frac{h_i^2}{6} - y_{i+1} + M_{i+1}\frac{h_i^2}{6}\right) \tag{3.16}$$

et

$$s_{i+1}'(x_{i+1}) = -M_{i+1}\frac{h_{i+1}}{2} - \frac{1}{h_{i+1}}\left(y_{i+1} - M_{i+1}\frac{h_{i+1}^2}{6} - y_{i+2} + M_{i+2}\frac{h_{i+1}^2}{6}.\right) \tag{3.17}$$

On écrit à présent les conditions de raccords des dérivées premières aux points x_i pour $i = 0, ..., n-1$ i.e. $s_i'(x_{i+1}) = s_{i+1}'(x_{i+1})$. On obtient ainsi en égalant (3.16) et (3.17) :

$$M_i\frac{h_i}{6} + M_{i+1}\frac{h_i + h_{i+1}}{3} + M_{i+2}\frac{h_{i+1}}{6} = \frac{1}{h_i}(y_i - y_{i+1}) + \frac{1}{h_{i+1}}(y_{i+2} - y_{i+1}), \quad \text{pour } i = 0, ..., n-2.$$

Or par hypothèse $M_0 = M_n = 0$ (conditions naturelles). Les inconnues $u = (M_1, ..., M_{n-1})^T \in \mathbb{R}^{n-1}$ vérifient donc le système linéaire

$$\boxed{Au = b},$$

avec $b = (b_i)_{i=1,...,n-1}$ et

$$b_i = \frac{1}{h_{i-1}}(y_{i-1} - y_i) + \frac{1}{h_i}(y_{i+1} - y_i)$$

et la matrice $A \in \mathcal{M}_{(n-1)\times(n-1)}$ est une matrice tridiagonale, donnée par

$$A = \frac{1}{6}\begin{pmatrix} 2(h_0+h_1) & h_1 & & & 0 \\ h_1 & 2(h_1+h_2) & h_2 & & \\ & \ddots & \ddots & \ddots & \\ & & & & h_{n-2} \\ 0 & & & h_{n-2} & 2(h_{n-2}+h_{n-1}) \end{pmatrix}$$

La matrice A est symétrique et à diagonale strictement dominante, par conséquent A est symétrique définie positive et en particulier elle est inversible.

3.10.2 Propriétés et estimations d'erreurs des splines cubiques

On donne à présent des estimations d'erreurs pour la spline cubique d'interpolation s d'une fonction f suffisamment régulière avec des conditions naturelles. La spline s vérifie

$$\begin{cases} s(x_i) = f(x_i), & i = 0, ..., n \\ s''(a) = 0,\ s''(b) = 0 & \text{(conditions naturelles).} \end{cases}$$

On note $h_i = x_{i+1} - x_i$ pour $i = 0, ..., n-1$ et $h = \max_i h_i$. Les résultats d'erreurs sont les suivants :

Proposition 2 :

1. Si $f \in C^2([a,b])$ alors on a $\displaystyle\int_a^b |s''(x)|^2 dx \leq \int_a^b |f''(x)|^2 dx$.
   L'égalité a lieu si et seulement si $f = s$.
2. On note $\beta = h/\min_i h_i$. Si $f \in C^4([a,b])$ alors

$$\|s^{(k)} - f^{(k)}\|_\infty \leq C_k\ h^{4-k}\ \|f^{(4)}\|_\infty, \quad \text{pour } k = 0,\ 1,\ 2,\ 3,$$

   avec $C_0 = 5/384$, $C_1 = 1/24$, $C_2 = 3/8$ et $C_3 = (\beta + \beta^{-1})/2$.

La première propriété traduit le fait que parmi toute les fonctions d'interpolation, la spline cubique s est celle qui a la plus petite "énergie". La deuxième propriété indique que la spline cubique s ainsi que ses dérivées première et seconde convergent uniformément vers f et ses dérivées lorsque h tend vers zéro. Il en est de même de la dérivée troisième pourvu que β soit borné uniformément.

3.10.3 Splines paramétriques

La problématique est à présent la suivante : on souhaite interpoler des points du plan qui ne sont pas nécessairement sur une courbe définie par le graphe d'une fonction. Ainsi, au lieu de considérer une fonction f comme on l'a fait jusquà présent, on considère maintenant une courbe plane paramétrée par $P : [0,T] \to \mathbb{R}^2$; $P(t) = \big(x(t), y(t)\big)$. On va interpoler cette courbe par une courbe spline en des points donnés $P_i = (x_i, y_i)$, $i = 0, \cdots, n$, du plan. Pour cela, on introduit les $(n+1)$ points $0 = t_0 < t_1 < \cdots < t_n$ définis par

$$\boxed{t_0 = 0}, \qquad \boxed{t_i = \sum_{k=1}^{i} l_k}, \text{ pour } i = 1, \cdots, n, \tag{3.18}$$

où l_k est la longueur du segment $P_{k-1}P_k$ c'est-à-dire

$$\boxed{l_k = \sqrt{(x_k - x_{k-1})^2 + (y_k - y_{k-1})^2}.} \tag{3.19}$$

Le paramètre t_i représente donc la longueur cumulée des segments joignant le point P_0 au point P_i.

On détermine alors les deux splines (cubiques) d'interpolation s_x et s_y associées respectivement à (t_i, x_i) et (t_i, y_i) pour $i = 0, \cdots, n$ et qui interpolent respectivement $x(t)$ et $y(t)$.

La courbe paramétrique

$$\boxed{\begin{array}{rcl} S : [0, t_n] & \to & \mathbb{R}^2 \\ t & \mapsto & S(t) = \Big(s_x(t), s_y(t)\Big) \end{array}}$$

est appelée **spline** (cubique) **paramétrique**.

La figure 3.7 montre une interpolation par une spline cubique paramétrique. On fait enfin remarquer que la spline paramétrique dépend du choix des paramètres t_i. D'autres choix sont possibles et qui conduisent évidemment à des splines paramétriques différentes.

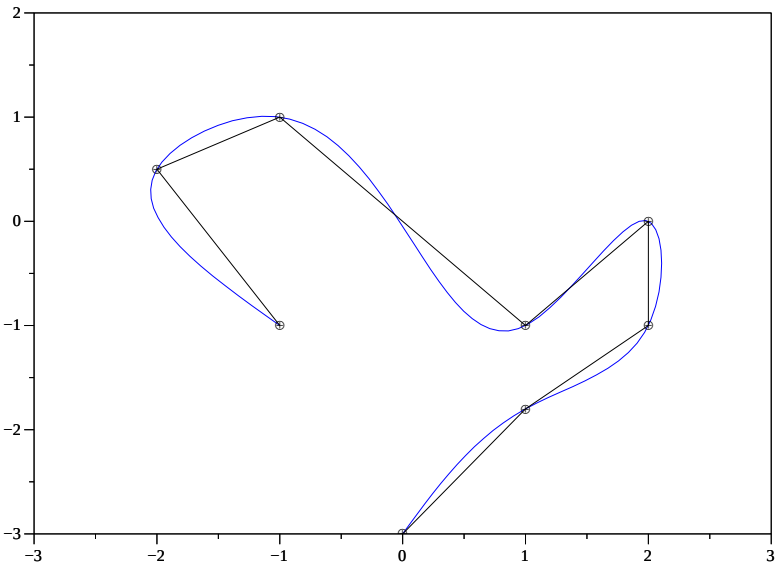

FIG. 3.7 – Interpolation par spline cubique paramétrique.

3.11 Courbes de Bézier

Il ne s'agit plus véritablement d'interpolation mais plutôt d'un procédé de construction de courbes ou formes régulières, utile pour des problèmes de CAO. A partir d'un certain nombre de points donnés du plan, on veut construire une courbe régulière (un polynôme) contrôlée par ces points.

3.11.1 Construction géométrique

Commençons par une construction avec trois points P_0, P_1, P_2 du plan. Pour un paramètre t *fixé* dans l'intervalle $[0,1]$, on définit les deux points $P_{0,1}(t) = (1-t)P_0 + tP_1$ appartenant au segment P_0P_1 et $P_{1,1}(t) = (1-t)P_1 + tP_2$ appartenant au segment P_1P_2. On joint ensuite $P_{0,1}$ à $P_{1,1}$ par un segment et on construit sur ce segment le point $P_{0,2}$ par $P_{0,2}(t) = (1-t)P_{0,1}(t) + tP_{1,1}(t)$. Maintenant, si le paramètre t parcourt tout l'intervalle $[0,1]$, alors le point $P_{0,2}$ décrit une courbe appelée *courbe de Bézier*[2] associée à P_0, P_1, P_2 (cf ; Figure 3.8). On note $B_2(t)$ cette courbe définie pour $t \in [0,1]$.

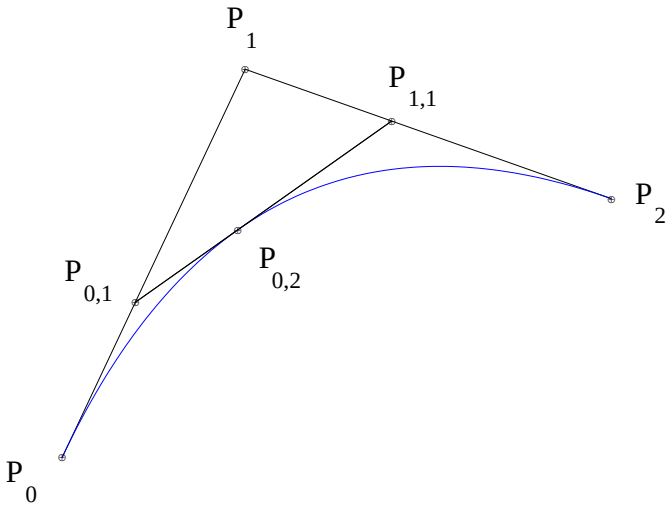

FIG. 3.8 – Construction d'une courbe de Bézier à partir des 3 points P_0, P_1, P_2.

On peut trouver une expression de la courbe de Bézier associée à P_0, P_1, P_2, en fonction de t. En effet, on a

$$\begin{aligned} B_2(t) = P_{0,2}(t) &= (1-t)P_{0,1}(t) + tP_{1,1}(t) \\ &= (1-t)[(1-t)P_0 + tP_1] + t[(1-t)P_1 + tP_2] \\ &= (1-t)^2 P_0 + 2t(1-t)P_1 + tP_2. \end{aligned}$$

On voit ainsi que la courbe de Bézier $B_2(t)$ associée aux trois points est un polynôme de degré 2 en t.

Cette construction se généralise facilement avec $(n+1)$ points du plan $P_0, \cdots, P_n$. On définit les points $P_{i,j}(t)$ de la façon suivante :

$$\begin{aligned} P_{i,1}(t) &= (1-t)P_i + tP_{i+1}, && i = 0, \cdots, n-1 \\ &\vdots && \vdots \\ P_{i,j}(t) &= (1-t)P_{i,j-1}(t) + tP_{i+1,j-1}(t), && i = 0, \cdots, n-j \\ &\vdots && \vdots \\ P_{0,n}(t) &= (1-t)P_{0,n-1}(t) + tP_{1,n-1}(t) \end{aligned}$$

On obtient alors que $B_n(t) = P_{0,n}(t)$ est la **courbe de Bézier** associée à $P_0, \cdots, P_n$.

[2] Pierre Bézier (1910-1999), ingénieur à la Régie Renault. Il étudia les courbes qui portent son nom dans les années 50.

3.11.2 Calcul numérique des courbes de Bézier

Comme pour la courbe de Bézier associée à trois points, on peut obtenir une expression de B_n en fonction de t grâce à des polynômes particuliers, les polynômes de Bernstein.

Définition : *Polynômes de Bernstein*

Les $(n+1)$ polynômes de Bernstein sont définis pour $t \in [0,1]$ et $k = 0, \cdots, n$, par

$$b_{n,k}(t) = C_n^k \; t^k (1-t)^{n-k} = \frac{n!}{k!(n-k)!} \; t^k (1-t)^{n-k}$$

Les polynômes de Bernstein vérifient les relations de récurrences suivantes

$$\begin{aligned} b_{n,0}(t) &= (1-t)^n \\ b_{n,k}(t) &= (1-t) b_{n-1,k}(t) + t b_{n-1,k-1}(t), \quad k = 1, \cdots, n. \end{aligned}$$

De plus, la famille $\{b_{n,0}, \cdots, b_{n,n}\}$ forme une base de $\mathbb{P}_n$.

La courbe de Bézier B_n associée aux points $P_0, \cdots, P_n$, s'exprime alors à partir des polynômes de Bernstein de la façon suivante :

$$B_n(t) = \sum_{k=0}^{n} P_k b_{n,k}(t), \qquad 0 \leq t \leq 1. \tag{3.20}$$

Il s'agit donc d'une moyenne pondérée des points P_k avec des poids $b_{n,k}$. La Figure 3.9 montre la courbe

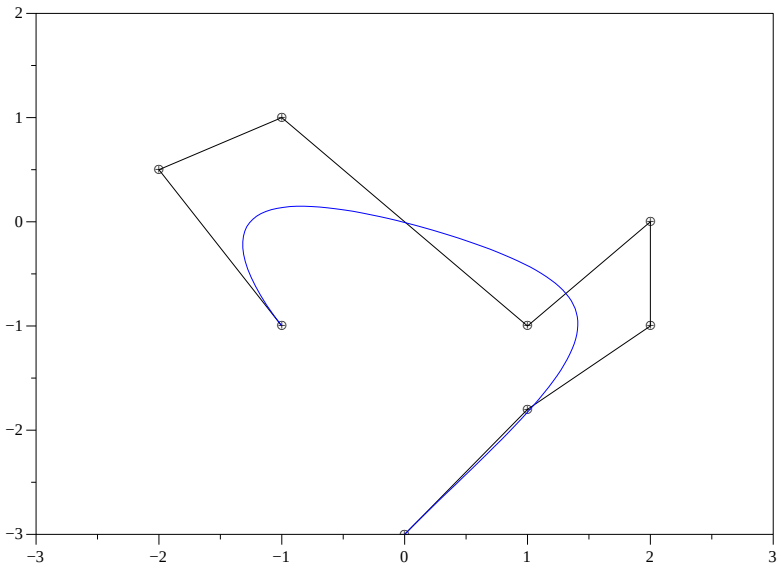

FIG. 3.9 – Courbe de Bézier.

de Bézier obtenue avec les huit points déjà utilisés à la section précédente pour déterminer une spline paramétrique (cf. Figure 3.7).

Chapitre 4

Problèmes de moindres carrés

4.1 Introduction

On est souvent amené à résoudre des systèmes linéaires comportants plus d'équations que d'inconnues :

$$Ax = b \quad \text{avec} \quad \begin{cases} A \in \mathcal{M}_{mn}(\mathbb{R}) \\ x \in \mathbb{R}^n \\ b \in \mathbb{R}^m \\ m > n \end{cases} \tag{4.1}$$

Sauf cas exceptionnels (4.1) n'admet pas de solutions et l'on défini alors une "pseudo-solution" $\bar{x}$ qui réalise le minimum de la fonction (erreur) :

$$E(x) = ||Ax - b||^2. \tag{4.2}$$

Cette solution (dont on verra qu'elle est unique si $KerA = \{0\}$) est appelée solution au sens des moindres carrés de (4.1). Revenons à l'équation (4.1), celle-ci a (au moins) une solution si et seulement si :

$$b \in ImA$$

ce qui est une condition exceptionnelle si b est pris au hasard dans $\mathbb{R}^m$. En effet l'image de A est un sous-espace vectoriel de $\mathbb{R}^m$ de dimension maximale $n < m$. Ceci se voit facilement en écrivant le produit Ax avec une décomposition de A suivant ses colonnes :

$$Ax = \left(\begin{array}{c|c|c|c} A^1 & A^2 & \dots & A^n \end{array} \right) \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \sum_{j=1}^{n} x_i A^j$$

la famille de vecteurs de $\mathbb{R}^m$ constituée par les colonnes de A apparait alors comme une famille génératrice de ImA. Par ailleurs, on supposera que ces colonnes sont des vecteurs linéairement indépendants, ce qui est équivalent à dire que $KerA = \{0\}$, et donc que $\mathcal{A} = (A^1, A^2, \dots, A^n)$ est une base de ImA. Ceci est une hypothèse réaliste : n vecteurs pris au hasard dans un espace de dimension $\geq n$ sont généralement linéairement indépendants (pour comprendre intuitivement ces notions faites des dessins). On évoquera cependant le cas dégénéré $dim(KerA) \geq 1$. Dans ce cas si $\bar{x}$ est une solution de (4.1) ou de (4.2) alors $\bar{x}+x^*$ où $x^* \in KerA$ est aussi une solution puisque $A(\bar{x}+x^*) = A\bar{x}+Ax^* = A\bar{x}$ et donc $E(\bar{x}+x^*) = E(\bar{x})$.

4.1.1 Exemple : un problème de lissage de points

Soient deux grandeurs physiques t et y liées entre-elles par une relation du type $y = f(t)$. On a quelques idées sur le type de fonction f qui convient, par exemple une droite, un polynôme, un polynôme

trigonométrique, etc... :

$$\begin{aligned} f(t) &= x_0 + x_1 t \\ f(t) &= x_0 + x_1 t + \dots x_n t^n \\ f(t) &= x_0 + x_1 sin(\frac{2\pi t}{T}) + x_2 cos(\frac{2\pi t}{T}) + x_3 sin(\frac{4\pi t}{T}) + \dots \\ f(t) &= x_0 e^{-\tau t} \end{aligned}$$

et l'on cherche alors à déterminer les paramètres (notés x_i) du modèle. On se contente uniquement de modèles qui sont linéaires en leurs paramètres. Par exemple, on pourrait aussi chercher à déterminer la période T dans le troisième modèle, et le taux de décroissance τ dans le dernier. Dans ce cas les méthodes de résolution se compliquent et nous ne les étudierons pas dans ce chapitre. On suppose donc la donnée de n fonctions ϕ_j et la fonction f cherchée est une combinaison linéaire de ces fonctions :

$$f(t) = x_1\phi_1(t) + x_2\phi_2(t) + \cdots + x_n\phi_n(t) = \sum_{j=1}^{n} x_j\phi_j(t) \tag{4.3}$$

Pour estimer les n paramètres x_j on effectue m mesures $(t_i, y_i)_{1\leq i\leq m}$, puis on essaie de minimiser la somme des écarts quadratiques entre ce que donne le modèle et les mesures :

$$\min_{x\in\mathbb{R}^n} \sum_{i=1}^{m} (f(t_i, x) - y_i)^2.$$

On peut aussi utiliser les écarts en valeur absolue ou d'autres choix encore mais la résolution du problème est plus difficile.

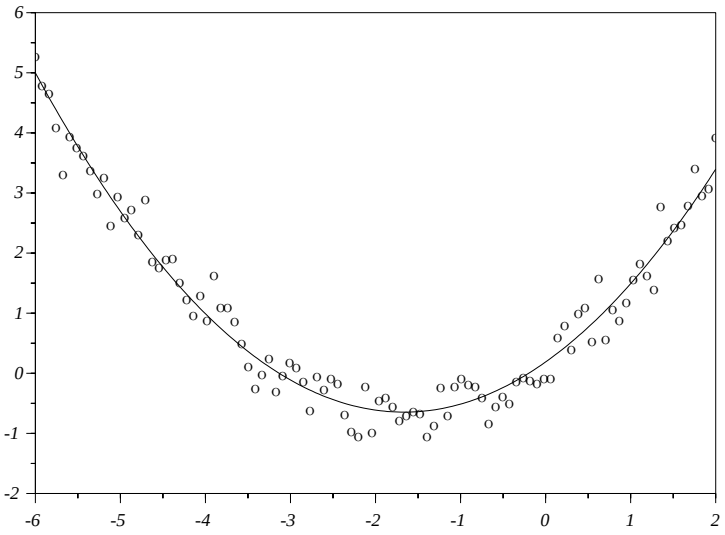

FIG. 4.1 – Approximation par un polynôme du second degré de données bruitées

On appelle fonction d'erreur la somme des écarts quadratiques :

$$E(x) = \sum_{i=1}^{m} (f(t_i, x) - y_i)^2. \tag{4.4}$$

Développons E en utilisant le modèle linéaire (en ses paramètres x_i) général (4.3), il vient :

$$E(x) = \sum_{i=1}^{m} (\sum_{j=1}^{n} x_j\phi_j(t_i) - y_i)^2$$

Si on introduit la matrice A de dimension (m, n) et telle que $a_{ij} = \phi_j(t_i)$, on remarque que sa i$^{\text{ème}}$ ligne est égale à :

$$A_i = \begin{pmatrix} \phi_1(t_i) & \phi_2(t_i) & \dots & \phi_n(t_i) \end{pmatrix}$$

et que :

$$f(t_i, x) = \sum_{j=1}^{n} x_j \phi_j(t_i) = A_i x = (Ax)_i$$

d'où l'expression pour la fonction erreur :

$$E(x) = \sum_{i=1}^{m} ((Ax)_i - y_i)^2 = \sum_{i=1}^{m} ((Ax - y)_i)^2 = ||Ax - y||^2,$$

on a donc obtenu un problème de la forme (4.2).

Définition 1 : Soient la matrice $A \in \mathcal{M}_{mn}(\mathbb{R})$, $m \geq n$, et le vecteur $b \in \mathbb{R}^m$ donnés. On appelle problème de moindres carrés, le problème : Trouver $x \in \mathbb{R}^n$ réalisant :

$$\min_{x \in \mathbb{R}^n} ||Ax - b||^2 \tag{4.5}$$

Hypothèse fondamentale : on supposera que les colonnes de la matrice A sont linéairement indépendantes, c-a-d $KerA = \{0\}$.

Dans la suite nous allons montrer deux approches possibles pour résoudre ce type de problème :

1. Résoudre en cherchant un point $\bar{x}$ qui annule la dérivée : *a priori* ce n'est pas forcément un minimum (on peut aussi obtenir un maximum ou un point selle) mais on montrera qu'il existe un et un seul point réalisant
$$E'(\bar{x}) = 0$$
et que ce point est bien le minimum de E.
2. Résoudre en utilisant une méthode pour orthogonaliser les colonnes de A : on verra en effet que si on connait une b.o.n. (base orthonormée) de ImA alors la solution s'obtient facilement.

Avant d'aller regarder d'un peu plus près la fonction E, voici quelques petits rappels sur les matrices de passage...

4.1.2 Matrices de changement de base

On considère un espace vectoriel E sur $\mathbb{K}$ de dimension n, et deux bases $\mathcal{U} = (u^1, u^2, \dots, u^n)$ et $\mathcal{V} = (v^1, v^2, \dots, v^n)$ de cet espace. On suppose aussi que l'on connait la décomposition des vecteurs de $\mathcal{V}$ dans $\mathcal{U}$ et l'on note :

$$v^j = \sum_{i=1}^{n} p_{ij} u^i, \text{ pour } j = 1, \dots, n$$

On notera P la matrice :

$$P = (p_{ij})_{\substack{1 \leq i \leq n \\ 1 \leq j \leq n}}$$

On se pose le problème d'élaborer une relation liant les composantes d'un vecteur quelconque x de E dans la base $\mathcal{U}$ à celles dans la base $\mathcal{V}$:

$$x = \sum_{i=1}^{n} x_i^{\mathcal{U}} u^i = \sum_{j=1}^{n} x_j^{\mathcal{V}} v^j \tag{4.6}$$

On introduit aussi la notation suivante : les composantes de x dans $\mathcal{U}$ (resp. dans $\mathcal{V}$) forment un vecteur de $\mathbb{K}^n$ que l'on notera :

$$[x]_{/\mathcal{U}} = [x_1^{\mathcal{U}}, \dots, x_n^{\mathcal{U}}]^\top$$

(resp. $[x]_{/\mathcal{V}}$). *Rmq* : avec cette notation on a : $P^j = [v^j]_{/\mathcal{U}}$.

La relation cherchée s'obtient facilement en partant de l'équation (4.6) et en exprimant les v^j en fonction des u^i :

$$\begin{aligned}
\sum_{i=1}^{n} x_i^{\mathcal{U}} u^i &= \sum_{j=1}^{n} x_j^{\mathcal{V}} v^j \\
\sum_{i=1}^{n} x_i^{\mathcal{U}} u^i &= \sum_{j=1}^{n} x_j^{\mathcal{V}} \left(\sum_{i=1}^{n} p_{ij} u^i \right) \\
\sum_{i=1}^{n} x_i^{\mathcal{U}} u^i &= \sum_{j=1}^{n} \sum_{i=1}^{n} \left(x_j^{\mathcal{V}} p_{ij} u^i \right) \\
\sum_{i=1}^{n} x_i^{\mathcal{U}} u^i &= \sum_{i=1}^{n} \sum_{j=1}^{n} \left(p_{ij} x_j^{\mathcal{V}} u^i \right) \\
\sum_{i=1}^{n} x_i^{\mathcal{U}} u^i &= \sum_{i=1}^{n} \left(\sum_{j=1}^{n} p_{ij} x_j^{\mathcal{V}} \right) u^i
\end{aligned}$$

et en égalisant "composantes par composantes" il vient :

$$x_i^{\mathcal{U}} = \sum_{j=1}^{n} p_{ij} x_j^{\mathcal{V}}, \text{ pour } i = 1, \ldots, n$$

ce qui peut se noter matriciellement :

$$[x]_{/\mathcal{U}} = P[x]_{/\mathcal{V}}$$

Conclusion : **Le fait de connaître la décomposition des vecteurs de $\mathcal{V}$ dans la base $\mathcal{U}$ (relations que l'on synthétise par la matrice P) permet de calculer facilement les composantes d'un vecteur dans la base $\mathcal{U}$ si on connait ses composantes dans la base $\mathcal{V}$.** Parfois on donne des noms à la matrice P comme "matrice de changement de base de $\mathcal{U}$ vers $\mathcal{V}$" (ce nom est aussi parfois donné à la transposée de cette matrice ainsi qu'à la matrice inverse) mais comme il règne une certaine confusion dans cette appellation, il vaut mieux retenir la phrase précédente ou savoir refaire le calcul ci-avant ! Si on dispose de la matrice P et des composantes de x dans la base $\mathcal{U}$ on peut alors obtenir celles dans $\mathcal{V}$ en résolvant le système linéaire $P[x]_{/\mathcal{V}} = [x]_{/\mathcal{U}}$.

4.2 Écriture de E comme une forme quadratique

La fonction $E : x \in \mathbb{R}^n \longmapsto ||Ax - b||^2 \in \mathbb{R}$ se développe en écrivant la norme au carré comme un produit scalaire :

$$\begin{aligned}
E(x) &= (Ax - b|Ax - b) \\
&= (Ax|Ax) - 2(Ax|b) + (b|b) \\
&= x^\top A^\top Ax - 2(A^\top b)^\top x + ||b||^2 \\
&= x^\top Gx - 2h^\top x + c
\end{aligned}$$

avec $G = A^\top A$ une matrice (n, n) symétrique, $h = A^\top b$ un vecteur de $\mathbb{R}^n$ et $c = ||b||^2$ une constante positive. Nous venons d'écrire E comme une forme quadratique. La matrice G est non seulement symétrique car :

$$G^\top = (A^\top A)^\top = A^\top (A^\top)^\top = A^\top A = G$$

(petit rappel $(AB)^\top = B^\top A^\top$), mais aussi définie positive, puisque :

$$(Gx|x) = (Ax|Ax) = ||Ax||^2 \geq 0, \ \forall x \in \mathbb{R}^n$$

et comme $KerA = \{0\}$ il vient :

$$(Gx|x) = 0 \Leftrightarrow ||Ax|| = 0 \Leftrightarrow Ax = 0 \Leftrightarrow x = 0$$

(rappel : la norme d'un vecteur est nulle si et seulement si le vecteur est nul).

A quoi ressemble la fonction E : une forme quadratique est la généralisation d'un polynôme du second degré dans le cas multidimensionnel. Comme la matrice G est définie positive, on obtient en fait un paraboloïde dont "le sommet est dirigé vers le bas", ce sommet constituant le point où E est minimale. Ceci peut s'expliquer rigoureusement avec un peu d'algèbre linéaire. Toute matrice symétrique est diagonalisable dans une base orthonormée, c-a-d qu'il existe une matrice orthogonale P (ses vecteurs colonnes p^j sont les vecteurs propres de G et forment une b.o.n. de $\mathbb{R}^n$) telle que :

$$\Lambda = P^\top GP \Leftrightarrow G = P\Lambda P^\top$$

la matrice diagonale Λ étant formée des n valeurs propres λ_i (les valeurs propres multiples apparaissant autant de fois que leur multiplicité) de G. L'inverse d'une matrice orthogonale carrée s'obtient simplement en prenant la transposée, en effet :

$$(P^\top P)_{ij} = P_i^\top P^j = (p^i)^\top p^j = (p_j|p_i) = \delta_{ij}$$

(l'élément en position (i, j) du produit de 2 matrices AB est égal à $(AB)_{ij} = A_i B^j$). D'où le résultat attendu $P^\top P = I$. On passe maintenant dans la b.o.n. des vecteurs propres, c-a-d que l'on va travailler avec $y = P^\top x$, il vient :

$$\begin{aligned} E(x) &= x^\top P\Lambda P^\top x - 2h^\top x + c \\ &= (P^\top x)^\top \Lambda P^\top x - 2h^\top PP^\top x + c \\ &= y^\top \Lambda y - 2(P^\top h)^\top y + c \\ &= y^\top \Lambda y - 2w^\top y + c \end{aligned}$$

où w est le vecteur h exprimé dans la base des vecteurs propres ($w = P^\top h$). Comme Λ est une matrice diagonale, l'expression obtenue pour E (en tant que fonction de y) est beaucoup plus simple :

$$E(x) = E(Py) = \hat{E}(y) = \sum_{i=1}^{n} (\lambda_i y_i^2 - 2w_i y_i) + c$$

Comme la matrice G est définie positive, ses valeurs propres sont strictement positives :

$$0 < (Gp^i|p^i) = \lambda_i (p^i|p^i) = \lambda_i$$

Continuons le développement de E :

$$\begin{aligned} \hat{E}(y) &= \sum_{i=1}^{n} (\lambda_i (y_i^2 - 2\frac{w_i}{\lambda_i} y_i)) + c \\ &= \sum_{i=1}^{n} \left(\lambda_i \left(y_i - \frac{w_i}{\lambda_i} \right)^2 - \frac{w_i^2}{\lambda_i} \right) + c \\ &= \sum_{i=1}^{n} \lambda_i \left(y_i - \frac{w_i}{\lambda_i} \right)^2 + \tilde{c} \end{aligned}$$

où $\tilde{c} = c - \sum_{i=1}^{n} w_i^2/\lambda_i$. Cette expression montre que le minimum de E (vue comme fonction de y) est atteind au point $\bar{y}$ tel que :

$$\bar{y}_i = \frac{w_i}{\lambda_i},\ i = 1, 2, \ldots, n \Leftrightarrow y = \Lambda^{-1} w = \Lambda^{-1} P^\top h$$

et donc au point :

$$\bar{x} = P\Lambda^{-1} P^\top h$$

en tant que fonction de x. Comme le calcul des valeurs propres et vecteurs propres est généralement beaucoup plus difficile (et cher) que la solution d'un problème de moindres carrés, on n'utilise pas cette méthode. On vient quand même de montrer que la solution existe et est unique. En effet tout autre choix pour y et donc pour x conduit à une valeur de la fonction telle que :

$$\hat{E}(y) = \underbrace{\sum_{i=1}^{n} \lambda_i \left(y_i - \frac{w_i}{\lambda_i} \right)^2}_{>0} + \tilde{c} > E(\bar{x}) = \tilde{c}.$$

Mais revenons à notre paraboloïde, est-ce bien ce que nous venons d'obtenir ? Oui car la matrice orthogonale P peut être choisie telle que $\det(P) = 1$ (le déterminant d'une matrice orthogonale est toujours égal à 1 ou -1 et si le déterminant est égal à -1 il suffit de permuter deux colonnes pour obtenir 1) et le changement de base $y = P^\top x$ est donc une rotation généralisée dans l'espace $\mathbb{R}^n$, c'est à dire que la forme de l'objet constitué par E n'a pas été modifiée. Maintenant en procédant à des coupes $\hat{E}(y) = Cte$, on obtient :

1. l'ensemble vide si $Cte < \tilde{c}$ la valeur minimale de E ;
2. le point $\bar{y}$ pour $Cte = \tilde{c}$,
3. un ellipsoïde (une ellipse si $n = 2$) pour $Cte > \tilde{c}$ (à faire en exercice),

ce qui correspond bien (du moins pour $n = 2$) à un paraboloïde dont le sommet est dirigé vers le bas ! Et que se passe-t-il si $dim(KerA) \geq 1$? Et bien 0 est alors valeur propre de G et en dimension 2 (en considérant une valeur propre nulle et l'autre strictement positive) on obtient un cylindre parabolique (cf dessin 4.2) (il y a alors une infinité de solutions correspondant à la droite D sur le dessin).

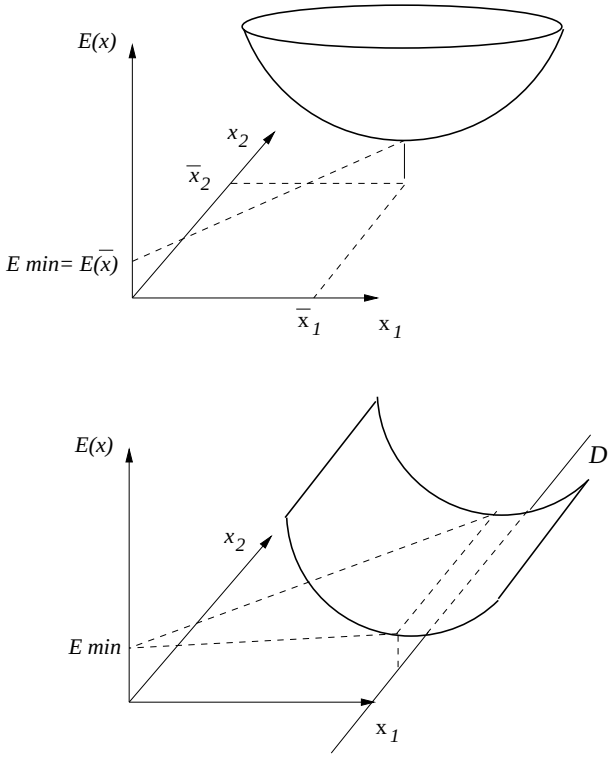

FIG. 4.2 – Formes quadratiques : le cas λ_1, $\lambda_2 > 0$ et le cas $\lambda_1 > 0$, $\lambda_2 = 0$

4.3 Résolution du problème par recherche du zéro de la dérivée

4.3.1 Rappels ou compléments de calcul différentiel

Dans ce qui suit, on considère une fonction $f : x \in \mathbb{R}^n \longmapsto f(x) \in \mathbb{R}$.

Définition 2 : On dit que $\bar{x}$ est un minimum local de f si et seulement si il existe une boule ouverte $B(\bar{x}, r)$ telle que : $\forall x \in B(\bar{x}, r),\ f(x) \geq f(\bar{x})$. Le point $\bar{x}$ est un minimum local strict si $\forall x \in B(\bar{x}, r),\ x \neq \bar{x},\ f(x) > f(\bar{x})$.

Définition 3 : On dit que $\bar{x}$ est un minimum global de f si et seulement si $\forall x \in \mathbb{R}^n,\ f(x) \geq f(\bar{x})$. Le point $\bar{x}$ est le minimum global strict si $\forall x \in \mathbb{R}^n,\ x \neq \bar{x},\ f(x) > f(\bar{x})$.

Définition 4 : On dit que la fonction f est dérivable en x si et seulement si, il existe $v \in \mathcal{L}(\mathbb{R}^n \to \mathbb{R})$ (une forme linéaire) telle que :

$$f(x+h) = f(x) + vh + r(h)||h||$$

avec :

$$r(h) : \mathbb{R}^n \to \mathbb{R} \text{ telle que } \lim_{||h|| \to 0} r(h) = 0$$

Remarques :

1. matriciellement une forme linéaire est un vecteur ligne ;
2. v est appelé dérivée de f en x et noté $f'(x)$ ou encore gradient de f en x et noté $\nabla f(x)$; dans notre cas $f'(x)$ est le vecteur (ligne) des dérivées partielles de f au point x :

$$f'(x) = \left(\begin{array}{ccc} \frac{\partial f}{\partial x_1}(x) & \ldots & \frac{\partial f}{\partial x_n}(x) \end{array} \right)$$

3. intuitivement une fonction est dérivable en x si l'hyper-surface définie par $z = f(x)$ admet un plan tangent au point $(x, f(x))$, où ce qui est équivalent peut être approchée (suffisamment près) par une application affine ; cette application affine est l'approximation au premier ordre de f en x c'est à dire $f_a(y) = f(x) + f'(x)(y - x)$.

Définition 5 : On dit que la fonction f admet une dérivée directionnelle en x dans la direction d si et seulement si la limite suivante :

$$\lim_{\epsilon \to 0} \frac{f(x + \epsilon d) - f(x)}{\epsilon}$$

existe. On peut noter $Df(x, d)$ cette limite.

Remarques :

1. La notion de dérivée directionnelle est simple à appréhender : on se choisit un point x dans l'espace ambient et une direction d : on regarde alors comment la fonction varie sur la droite $D_{x,d}$ définie par $y \in D_{x,d} \Leftrightarrow \exists t \in \mathbb{R}$ tel que $y = x + td$. La fonction f restreinte sur cet ensemble devient alors une fonction d'une variable (t) et on cherche à calculer son "taux d'accroissement" en $t = 0$ (c'est à dire en x).
2. On peut montrer que :

$$Df(x, d) = f'(x)d$$

et que les dérivées partielles sont les dérivées directionnelles dans la direction des vecteurs de la base canonique, c-a-d :

$$\frac{\partial f}{\partial x_i}(x) = Df(x, e^i).$$

L'intérêt de cette remarque est qu'il est souvent plus facile de calculer les dérivées partielles via cette définition que par la méthode usuelle. Par exemple si avec la méthode habituelle on montre facilement que la dérivée de la fonction $f_1(x) = (y|x)$ (y est un vecteur donné de $\mathbb{R}^n$) est :

$$f_1'(x) = y^\top$$

c'est plus difficile avec la fonction $f_2(x) = (Bx|x)$ (où B est une matrice réelle (n, n)). Pour celle-ci, en utilisant la méthode suggérée on obtient assez rapidement que :

$$f_2'(x) = x^\top (B + B^\top).$$

Théorème 1 : On suppose que f est dérivable sur $\mathbb{R}^n$. Une condition nécessaire pour que $\bar{x}$ soit un minimum local de f est que :

$$f'(\bar{x}) = 0.$$

Démonstration : Comme $\bar{x}$ est un minimum local de f, il existe une boule $B(\bar{x}, r)$ telle que $f(x) \geq f(\bar{x}),\ \forall x \in B(\bar{x}, r)$. f étant dérivable on a :

$$f(x) = f(\bar{x} + h) = f(\bar{x}) + f'(\bar{x})h + r(h)||h||$$

Prenons alors $h = \tau d$, où τ est un scalaire et $d \in \mathbb{R}^n$ une direction que l'on se fixe arbitrairement (avec $||d|| = 1$). On a pour tout τ tel que $|\tau| < r,\ x = \bar{x} + h \in B(\bar{x}, r)$, donc :

$$f(x) - f(\bar{x}) \geq 0 \tag{4.7}$$

$$f'(\bar{x})\tau d + r(\tau d)|\tau| \geq 0 \tag{4.8}$$

On en déduit donc que :

$$\tau > 0 \Rightarrow f'(\bar{x})d + r(\tau d) \geq 0$$

et donc que :

$$\lim_{\tau \to 0+} f'(\bar{x})d \geq 0$$

alors que pour $\tau < 0$:

$$\tau < 0 \Rightarrow f'(\bar{x})d - r(\tau d) \leq 0$$

et donc que :

$$\lim_{\tau \to 0-} f'(\bar{x})d \leq 0$$

D'où $f'(\bar{x})d = 0$ mais comme la direction d est arbitraire (avec cependant $||d|| = 1$) on en déduit que $f'(\bar{x}) = 0$ (par exemple en prenant successivement $d = e^1, d = e^2$, etc...).

Définition 6 : La fonction f est dite convexe si et seulement si :

$$\forall x, y \in \mathbb{R}^n,\ \forall \theta \in [0,1] \text{ on a } f((1-\theta)x + \theta y) \leq (1-\theta)f(x) + \theta f(y)$$

et strictement convexe si et seulement si :

$$\forall x, y \in \mathbb{R}^n, x \neq y,\ \forall \theta \in]0,1[\text{ on a } f((1-\theta)x + \theta y) < (1-\theta)f(x) + \theta f(y)$$

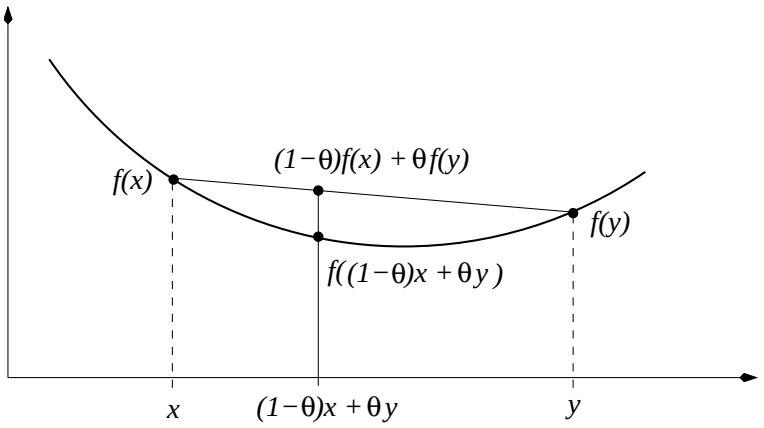

FIG. 4.3 – Illustration pour la convexité (avec $\theta \simeq 1/3$)

Théorème 2 : Si f est dérivable et strictement convexe et si il existe un point $\bar{x}$ vérifiant $f'(\bar{x}) = 0$ alors $\bar{x}$ est le minimum global de f.

Démonstration : On écrit la propriété de convexité de f avec le point $\bar{x}$ et un point quelconque $x \neq \bar{x}$: $\forall \theta \in]0,1[$ on a :

$$\begin{array}{rcl} f((1-\theta)\bar{x}+\theta x) & < & (1-\theta)f(\bar{x})+\theta f(x) \\ f(\bar{x}+\theta(x-\bar{x}))-f(\bar{x}) & < & \theta(f(x)-f(\bar{x})) \end{array}$$

En divisant par θ on fait apparaître à gauche une future dérivée directionnelle (dans la direction $x-\bar{x}$) :

$$F(\theta) = \frac{f(\bar{x}+\theta(x-\bar{x}))-f(\bar{x})}{\theta} \; < \; f(x)-f(\bar{x})$$

et en passant à la limite lorsque $\theta \to 0+$, il vient :

$$\lim_{\theta \to 0+} F(\theta) = Df(\bar{x}, x-\bar{x}) = f'(\bar{x})(x-\bar{x}) \leq f(x)-f(\bar{x})$$

et comme $f'(\bar{x}) = 0$, on obtient :

$$f(\bar{x}) \leq f(x), \; \forall x \in \mathbb{R}^n.$$

Ce qui montre que $\bar{x}$ est un minimum global de f. Montrons qu'il est unique : soit $x^* \neq \bar{x}$ un autre minimum global, c'est à dire que $f(\bar{x}) = f(x^*) = m$ alors en utilisant la stricte convexité de f il vient :

$$f(\frac{1}{2}\bar{x}+\frac{1}{2}x^*) < \frac{1}{2}f(\bar{x})+\frac{1}{2}f(x^*) = m$$

on aurait donc trouvé un point $\hat{x} = \frac{1}{2}\bar{x}+\frac{1}{2}x^*$ pour lequel la valeur de f est strictement inférieure à son minimum global, ce qui est une contradiction. D'où $\bar{x} = x^*$.

4.3.2 Application des résultats précédents sur notre fonction E

Comme E est une forme quadratique, c-a-d un polynôme du second degré en les variables $x_1, x_2, \ldots, x_n$, E est indéfiniment dérivable et donc dérivable. Par conséquent une condition nécessaire pour que $\bar{x}$ soit un minimum est que :

$$E'(\bar{x}) = 0. \tag{4.9}$$

La dérivée de E est :

$$E'(x) = x^\top(G+G^\top) - 2h^\top = 2(x^\top G^\top - h^\top)$$

car G est symétrique, l'équation (4.9) s'écrit alors :

$$\bar{x}^\top G^\top - h^\top = 0$$

en transposant (pour se ramener à un système linéaire classique) et en écrivant $G = A^\top A$ et $h = A^\top b$, on obtient finalement que :

$$A^\top A\bar{x} = A^\top b \tag{4.10}$$

système d'équations linéaires que l'on appelle généralement *équations normales* du problème de moindres carrés. Comme la matrice G est définie positive, elle est en particulier inversible et la solution du système linéaire (4.10) existe et est unique.

On montre maintenant que la fonction E est strictement convexe. Pour cela on forme la différence q :

$$q = (1-\theta)E(x)+\theta E(y) - E((1-\theta)x+\theta y)$$

Pour simplifier une partie des calculs, il est relativement simple de voir que le terme linéaire et le terme constant de E ne vont pas intervenir. On a donc directement :

$$q = (1-\theta)(Gx|x)+\theta(Gy|y) - (G((1-\theta)x+\theta y)|(1-\theta)x+\theta y)$$

Il faut développer le dernier terme (du fait que G est symétrique on a $(Gx|y) = (Gy|x)$) :

$$\begin{array}{rcl} q & = & (1-\theta)(Gx|x)+\theta(Gy|y)-(1-\theta)^2(Gx|x)-2(1-\theta)\theta(Gx|y)-\theta^2(Gy|y) \\ & = & (1-\theta)\theta(Gx|x)+\theta(1-\theta)(Gy|y)-(1-\theta)\theta(Gx|y)-(1-\theta)\theta(Gy|x) \\ & = & (1-\theta)\theta\left((Gx|x-y)+(Gy|y-x)\right) \\ & = & (1-\theta)\theta(G(x-y)|x-y) \end{array}$$

et donc $q > 0$ pour $\theta \in]0,1[$ et pour $x \neq y$ (on utilise le fait que G est définie positive).

Conclusion : le point $\bar{x}$ unique solution des équations normales est bien le minimum global de E.

4.4 Résolution du problème par orthogonalisation

4.4.1 Projection orthogonale sur un sous-espace vectoriel

Théorème 3 : Soit E un espace vectoriel sur $\mathbb{R}$ de dimension m et S un sous espace vectoriel de E de dimension n (donc $n \leq m$).E est muni d'un produit scalaire (noté $(.|.)$), on note $||.||$ la norme associée ($||x|| = \sqrt{(x|x)}$). Le problème :

$$\min_{y \in S} ||y - b||^2$$

où $b \in E$ est donné, admet une solution unique $\bar{y}$, appelée projection orthogonale de b sur S et le vecteur résidu $r = b - \bar{y}$ appartient au sous espace vectoriel orthogonal à S (noté $S^{\perp}$), c-a-d $(b - \bar{y}|v) = 0,\ \forall v \in S$.

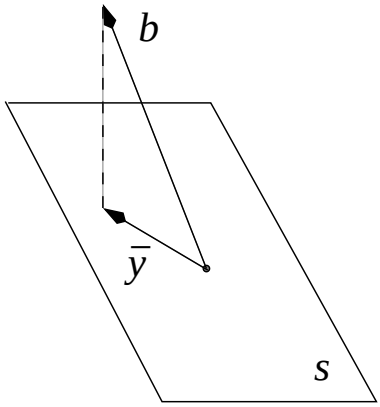

FIG. 4.4 – Projection orthogonale sur un sous-espace

Démonstration : soit $\mathcal{S} = (s^1, s^2, \ldots, s^n)$ une b.o.n. (base orthonormée) de S. On la complète par $m - n$ vecteurs pour former une b.o.n. de E notée $\mathcal{E} = (s^1, \ldots, s^n, s^{n+1}, \ldots, s^m)$ (il est clair que les $m - n$ derniers vecteurs forment une b.o.n. de $S^{\perp}$). Tout vecteur $y \in S$ s'écrit de manière unique :

$$y = \sum_{i=1}^{n} y_i s^i$$

et le vecteur b a aussi une décomposition unique sur la base $\mathcal{E}$:

$$b = \sum_{i=1}^{m} b_i s^i.$$

Développons le terme $||y - b||^2$:

$$\begin{aligned} ||y - b||^2 &= (y - b|y - b) \\ &= (\sum_{i=1}^{n}(y_i - b_i)s^i - \sum_{i=n+1}^{m} b_i s^i | \sum_{i=1}^{n}(y_i - b_i)s^i - \sum_{i=n+1}^{m} b_i s^i) \\ &= \sum_{i=1}^{n}(y_i - b_i)^2 + \sum_{i=n+1}^{m} b_i^2 \end{aligned}$$

car comme $\mathcal{E}$ est une b.o.n. on a $(s^i|s^j) = \delta_{ij}$. Cette quantité est trivialement minimisée pour $\bar{y} = \sum_{i=1}^{n} b_i s^i$ et tout autre choix pour y conduit à :

$$||y - b||^2 > ||\bar{y} - b||^2.$$

D'autre part le vecteur résidu $r = \sum_{i=n+1}^{m} b_i s^i$ est bien dans $S^{\perp}$.

4.4.2 Application : un premier algorithme basé sur cette idée

En posant $y = Ax$ le problème de moindres carrés (4.5) s'écrit :

$$\min_{y \in ImA} ||y - b||^2 \tag{4.11}$$

par conséquent si on connait une b.o.n. $\mathcal{S}$ de ImA on obtient facilement la solution dans cette base en utilisant le résultat précédent :

$$\bar{y} = \sum_{i=1}^{n} (b|s^i) s^i$$

Rmq : on notera $[\bar{y}]_{/\mathcal{S}}$ le vecteur des composantes de $\bar{y}$ dans la base $\mathcal{S}$, c-a-d $[\bar{y}]_{/\mathcal{S}} = ((b|s^1), \ldots, (b|s^n))^\top$. D'autre part, comme :

$$y = Ax = \sum_{i=1}^{n} x_i A^i$$

le vecteur x est le vecteur des composantes de y dans la base $\mathcal{A} = (A^1, A^2, \ldots, A^n)$ formée par les colonnes de A (avec la notation précédemment introduite $x = [y]_{/\mathcal{A}}$). Ce qui nous intéresse en fait, c'est la solution du problème (4.11) dans cette base. Comme les colonnes de A ne sont généralement pas orthonormées, il nous faut donc :

1. trouver une b.o.n. $\mathcal{S}$ de ImA ;
2. et obtenir une matrice de passage entre cette base et la base $\mathcal{A}$.

Un moyen connu depuis fort longtemps pour orthonormer une base est le procédé ou algorithme de Gram-Schmidt : dans le cas de deux vecteurs (cf figure (4.5)) u^1 et u^2, on obtient directement v^1 en normalisant u^1 puis v^2 est obtenu en deux temps :

1. on "enlève" de u^2 sa "composante" suivant v^1 : $\tilde{v}^2 = u^2 - (u^2|v^1)v^1$;
2. on normalise le vecteur $\tilde{v}^2$ ainsi obtenu : $v^2 = \tilde{v}^2 / ||\tilde{v}^2||$.

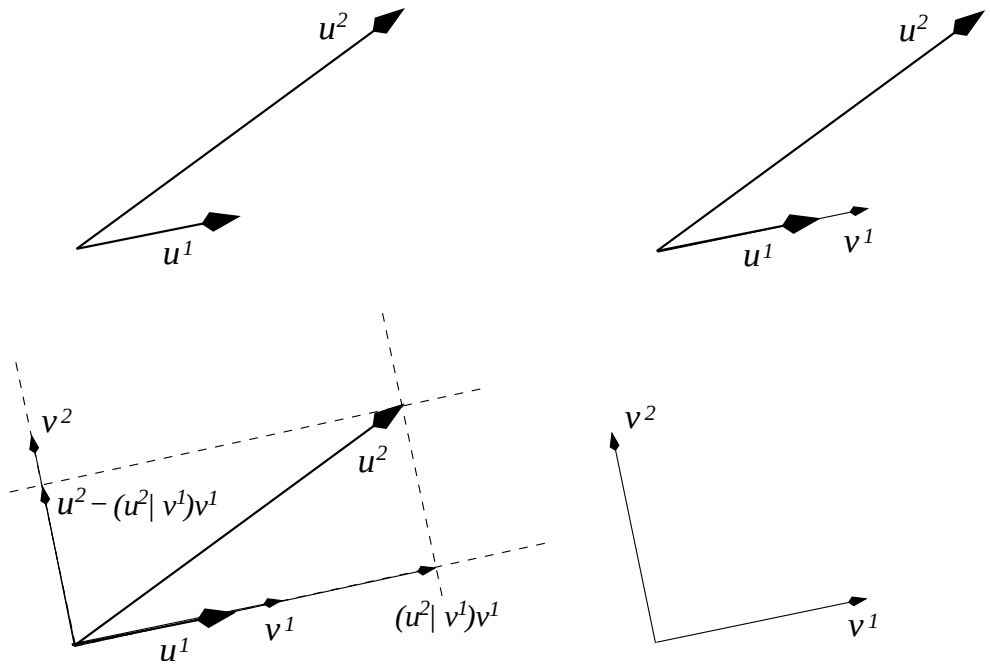

FIG. 4.5 – Illustration du procédé de Gram-Schmidt

Dans le cas général cet algorithme s'écrit :

pour $j = 1, 2, \ldots, n$
$\tilde{s}^j = A^j - \sum_{i=1}^{j-1}(A^j|s^i)s^i$
$s^j = \tilde{s}^j/||\tilde{s}^j||$
fin pour

Preuve : Comme $s^1 = A^1/||A^1||$, s^1 est normé. Supposons que pour $j \leq n$ les vecteurs $s^1, s^2, \ldots, s^{j-1}$ forment une famille orthonormée , la construction de s^j implique-t-elle que la famille de vecteurs $s^1, s^2, \ldots, s^j$ est orthonormée ? Si oui, on prouve bien (c'est une preuve par récurrence finie) que ça marche. Effectuons donc le produit scalaire de $\tilde{s}^j$ avec s^k, $k \in [1, j-1]$:

$$\begin{aligned}(\tilde{s}^j|s^k) &= (A^j - \sum_{i=1}^{j-1}(A^j|s^i)s^i|s^k) \\ &= (A^j|s^k) - \sum_{i=1}^{j-1}(A^j|s^i)\underbrace{(s^i|s^k)}_{\delta_{ik}} \\ &= (A^j|s^k) - (A^j|s^k) = 0\end{aligned}$$

D'autre part $\tilde{s}^j$ ne peut être nul : en effet il est assez simple de voir (on peut le démontrer rigoureusement par récurrence) que les vecteurs $s^1, s^2, \ldots, s^{j-1}$ engendrent le même espace vectoriel E_{j-1} que les vecteurs $A^1, A^2, \ldots, A^{j-1}$. Par conséquent si $\tilde{s}^j = 0$ il vient que :

$$A^j = \sum_{i=1}^{j-1}(A^j|s^i)s^i$$

c'est à dire que A^j est dans E_{j-1} et donc il existe des scalaires α_i tels que :

$$A^j = \sum_{i=1}^{j-1}\alpha_i A^i$$

ce qui veut dire que les vecteurs $A^1, \ldots, A^j$ ne sont pas linéairement indépendants (d'où la contradiction).

Grâce à cet algorithme[1] on peut donc obtenir une b.o.n. de ImA. Mais quant est-il de la matrice de passage ? Celle-ci s'obtient sans aucun calcul supplémentaire : en notant t_{ij} les produits scalaires $(A^j|s^i)$ et t_{jj} la norme $||\tilde{s}^j||$, les égalités :

$$\left\{\begin{array}{l}\tilde{s}^j = A^j - \sum_{i=1}^{j-1}(A^j|s^i)s^i \\ s^j = \tilde{s}^j/||\tilde{s}^j||\end{array}\right.$$

impliquent que :

$$A^j = \sum_{i=1}^{j} t_{ij}s^i$$

et l'on peut donc former la matrice de passage :

$$T = \begin{array}{c} \begin{array}{ccccc} A^1 & \ldots & A^j & \ldots & A^n \end{array} \\ \left(\begin{array}{ccccc} t_{11} & \ldots & t_{1j} & \ldots & t_{1n} \\ 0 & \ddots & \vdots & \ldots & \times \\ \vdots & \ddots & t_{jj} & \ldots & \times \\ \vdots & & \ddots & \ddots & \times \\ 0 & \ldots & \ldots & 0 & t_{nn} \end{array}\right) \end{array} \begin{array}{c} s^1 \\ \vdots \\ s^j \\ \vdots \\ s^n \end{array}$$

[1] Programmé naïvement il a un très mauvais comportement vis à vis des erreurs d'arrondi numérique ; par contre en l'écrivant différemment on obtient l'algorithme de Gram-Schmidt modifié, complètement équivalent sur le plan mathématique mais dont le comportement numérique est bien meilleur.

qui est triangulaire supérieure. Cette forme est la bienvenue car on peut voir facilement que :

$$T[\bar{y}]_{/\mathcal{A}} = T\bar{x} = [\bar{y}]_{/\mathcal{S}}$$

il faut encore résoudre un système linéaire pour obtenir $\bar{x}$. Mais comme la matrice est triangulaire ceci ne coûte pas très cher.

Résumé de l'algorithme :

1. Appliquer l'algorithme de Gram-Schmidt sur la matrice A : on obtient la base $\mathcal{S} = (s^1, \ldots, s^n)$ et la matrice T ;
2. Effectuer le calcul de $[\bar{y}]_{/\mathcal{S}} = ((b|s^1), \ldots, (b|s^n))^\top$;
3. Résoudre le système triangulaire supérieur $Tx = [\bar{y}]_{/\mathcal{S}}$.

4.4.3 Interprétation matricielle de l'algorithme 1

En notant S la matrice obtenue en juxtaposant les vecteurs de $\mathcal{S}$: $S = (s^1|s^2|\ldots|s^n)$, on peut voir que l'algorithme de Gram-Schmidt correspond à une décomposition de la matrice A comme produit d'une matrice orthogonale $S(m,n)$ et d'une matrice triangulaire supérieure $T(n,n)$:

$$A = ST$$

Enfin le calcul de $[\bar{y}]_{/\mathcal{S}}$ peut se noter matriciellement :

$$[\bar{y}]_{/\mathcal{S}} = S^\top b$$

La conséquence de ceci est que si l'on obtient une décomposition de ce type (par un moyen quelconque) $A = \widetilde{Q}\widetilde{R}$ alors les colonnes de $\widetilde{Q}$ forment une b.o.n. de ImA et on peut résoudre le problème de moindres carrés par :

$$\widetilde{R}x = \widetilde{Q}^\top b$$

Le but de ce qui suit est d'obtenir une telle décomposition par un procédé numériquement plus stable que l'algorithme de Gram-Schmidt[2].

4.4.4 Algorithme 2 : la décomposition $A = QR$ par les transformations de Householder

Théorème 4 : Soit A une matrice (m,n), $m \geq n$, il existe une matrice $Q(m,m)$ orthogonale et une matrice $R(m,n)$ triangulaire supérieure telle que :

$$A = QR$$

Remarque : vu les dimensions des matrices Q et R cette décomposition semble différente de celle obtenue par Gram-Schmidt. En fait on s'y ramène facilement en décomposant par blocs :

$$Q = (\widetilde{Q}|\widehat{Q}), \text{ et } R = \left(\frac{\widetilde{R}}{0}\right)$$

où $\widetilde{Q}$ est (m,n), $\widehat{Q}$ est $(m, m-n)$, alors que $\widetilde{R}$ est (n,n) et triangulaire supérieure, 0 étant la matrice nulle $(m-n, n)$. On obtient alors :

$$QR = \widetilde{Q}\widetilde{R} + \widehat{Q}0 = \widetilde{Q}\widetilde{R}$$

En fait l'algorithme qui permet d'obtenir la décomposition $A = QR$ fournit non seulement une b.o.n. de ImA (les colonnes de $\widetilde{Q}$) mais aussi une b.o.n. de $(ImA)^\perp$ (les colonnes de $\widehat{Q}$).

Idées de la construction :

[2] même dans sa version modifiée.

1. Comme dans la méthode de Gauss, on essaie de mettre des zéros sous la diagonale de la matrice A ;
2. Cependant ceci est obtenu à l'aide de matrices orthogonales $H^{(k)}$. Une étape de l'algorithme s'écrivant matriciellement :
$$A^{(k)} = H^{(k)} A^{(k-1)}$$
Au bout de n étapes on obtient la matrice R : $R = A^{(n)}$. Et l'on a :
$$R = H^{(n)} \dots H^{(1)} A$$
Comme le produit de deux matrices orthogonales est aussi une matrice orthogonale on obtient la décomposition annoncée en posant $Q^\top = H^{(n)} \dots H^{(1)}$.
3. Les matrices $H^{(k)}$ sont des transformations de Householder qui sont des symétries par rapport à un hyperplan vectoriel. Elles sont définies de la manière suivante :
$$H = I - \frac{2uu^\top}{||u||^2},\ u \neq 0$$
A chaque étape k, il s'agit alors de choisir le vecteur u de façon à faire apparaître les zéros dans la colonne k sous la diagonale.

Et pour finir voici comment choisir u avec le **Théorème 4 :** Soit $x \in \mathbb{R}^p$ avec $x \neq \alpha e^1$, alors il existe deux transformations de Householder $H^{(-)}$ et $H^{(+)}$ telles que :

$$H^{(-)}x = -||x||e^1,\ H^{(-)} \text{ s'obtient avec } u = x + ||x||e^1$$
$$H^{(+)}x = +||x||e^1,\ H^{(+)} \text{ s'obtient avec } u = x - ||x||e^1$$

Démonstration : à faire en exercice.

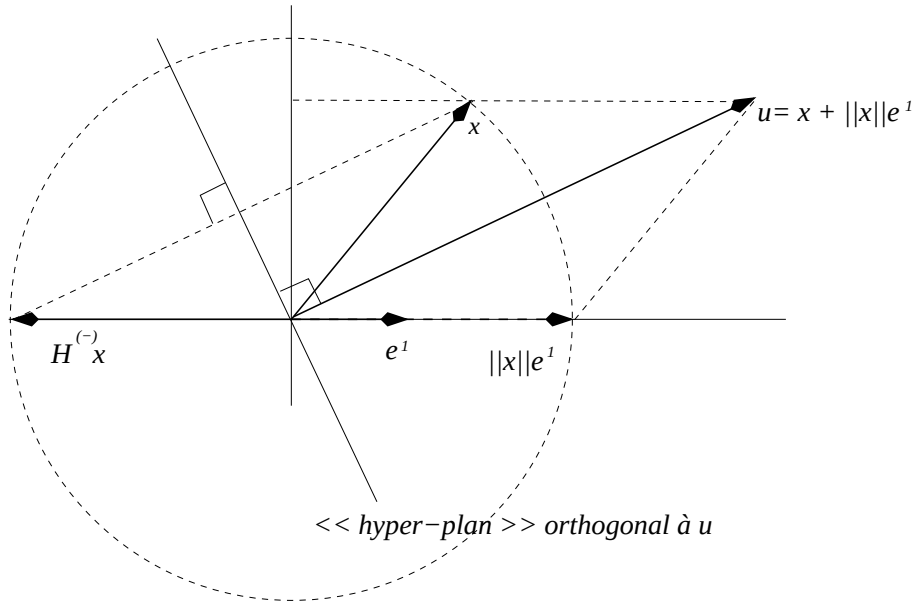

FIG. 4.6 – Une transformée de Householder en action

Remarque : dans Matlab, l'instruction `x = A\b` résoud un problème de moindres carrés avec ce dernier algorithme : l'opérateur `\` rend donc transparent à un utilisateur un certain nombre de choses...

Chapitre 5

Introduction à l'analyse de données : Analyse en composantes principales.

5.1 Introduction

L'analyse de données développée dans ce chapitre porte sur des données *quantitatives* reliant un échantillon d'individus à un certain nombre de critères ou caractères. Les individus et les critères sont en général nombreux et le but de l'analyse de données est de pouvoir traiter ces nombreuses données pour analyser les relations existantes entre les individus et entre les variables. En particulier, on veut représenter graphiquement les relations entre variables d'une part et visualiser en même temps les individus qui sont en relation avec ces variables. Le nombre important de variables nécessite de trouver un moyen spécifique pour visualiser et analyser ces données. Evidemment, ce moyen doit être suffisamment représentatif des relations existantes entre individus et variables. C'est en fait ce que permet de réaliser l'Analyse en Composantes Principales (ACP). L'aspect quantitatif des données (des nombres !) conditionne l'utilisation de cette méthode d'analyse particulière. Le but de ce chapitre est d'exposer le contenu de l'ACP en le justifiant autant que possible.

On suppose donc que l'on dispose de données numériques concernant un nombre n d'*individus* en rapport avec un nombre p de critères que l'on appellera désormais *variables*. Ces données sont regroupées dans un tableau où les individus sont rangés *par lignes* et les variables *par colonnes*. A ce tableau des données, on associe naturellement une matrice X de taille $n \times p$.

Tout au long de ce chapitre, on a choisi un exemple volontairement simple (sans doute un peu trop), mais qui permet de décrire et d'illustrer simplement la méthode et les concepts mis en oeuvre. Cet exemple concerne une étude réalisée auprès de 24 personnes ayant acheté une piscine. Cette étude doit permettre de déterminer l'influence de plusieurs facteurs sur l'achat d'une piscine et de dégager ainsi un profil "type" d'acheteur. Les critères retenus dans l'étude sont le revenu du propriétaire, la surface du terrain et un critère psychologique d'attitude à l'égard du soleil, traduit par une note de 0 à 10. En fait, un des buts de l'étude est de savoir si ce critère "soleil" est déterminant ou non dans l'achat d'une piscine. Les données brutes recueillies sont reportées dans le Tableau 5.1.

Avant de décrire la méthode, on commence par rappeler quelques éléments de statistiques élémentaires ainsi que quelques notions et résultats sur les valeurs et vecteurs propres de matrice ; ces résultats sont nécessaires à l'exposé de la méthode d'ACP.

Individus	**Revenu** (kF)	**Superficie** (m^2)	**Soleil**
1	9.3	30.2	8
2	10.2	40.1	10
3	9.7	35.3	6
4	11.5	45.1	4
5	12	38	5
6	14.2	50.1	9
7	18.6	60.2	10
8	28.4	100.4	3
9	13.2	25.1	2
10	10.8	40.7	7
11	22.7	68.4	9
12	12.3	60.3	8
13	8.2	38.2	2
14	7.8	40.2	0
15	11.4	44.8	1
16	16.3	50.6	4
17	12.4	42.5	8
18	11.5	60.3	0
19	6.8	39.7	6
20	10.4	35.4	4
21	14.2	42.5	3
22	11.5	38.4	2
23	8.4	30.2	4
24	9.1	25.7	5
moyenne	12.538	45.1	5
écart-type	4.793	15.826	3.028

$$X = \begin{pmatrix} 9.3 & 30.2 & 8 \\ 10.2 & 40.1 & 10 \\ 9.7 & 35.3 & 6 \\ 11.5 & 45.1 & 4 \\ 12 & 38 & 5 \\ 14.2 & 50.1 & 9 \\ 18.6 & 60.2 & 10 \\ 28.4 & 100.4 & 3 \\ 13.2 & 25.1 & 2 \\ 10.8 & 40.7 & 7 \\ 22.7 & 68.4 & 9 \\ 12.3 & 60.3 & 8 \\ 8.2 & 38.2 & 2 \\ 7.8 & 40.2 & 0 \\ 11.4 & 44.8 & 1 \\ 16.3 & 50.6 & 4 \\ 12.4 & 42.5 & 8 \\ 11.5 & 60.3 & 0 \\ 6.8 & 39.7 & 6 \\ 10.4 & 35.4 & 4 \\ 14.2 & 42.5 & 3 \\ 11.5 & 38.4 & 2 \\ 8.4 & 30.2 & 4 \\ 9.1 & 25.7 & 5 \end{pmatrix}$$

TAB. 5.1 – *Les données initiales et la matrice associée.*

5.2 Quelques rappels de statistiques

Dans ce paragraphe, x et y désignent deux vecteurs de $\mathbb{R}^n$; les composantes d'un vecteur x sont notées x_i.

⋆ *Produit scalaire dans* $\mathbb{R}^n$: $(x \,|\, y) = \sum_{i=1}^{n} x_i y_i$. Si A est une matrice carrée d'ordre n à coefficients réels, on a la propriété

$$(Ax \,|\, y) = (x \,|\, A^T y), \tag{5.1}$$

où A^T désigne la matrice transposée.

⋆ *Norme euclidienne dans* $\mathbb{R}^n$: $\|x\| = (x \,|\, x)^{1/2} = \sum_{i=1}^{n} x_i^2$.

⋆ *Moyenne de* x : $\bar{x} = E(x) = \frac{1}{n} \sum_{i=1}^{n} x_i$.

⋆ *Variance de* x : $V(x) = \frac{1}{n} \sum_{i=1}^{n} (x_i - \bar{x})^2 = \frac{1}{n} \|x - \bar{x}\mathbf{1}\|^2$ où $\mathbf{1}$ désigne le vecteur de $\mathbb{R}^n$ dont toutes les composantes sont égales à 1 i.e. $\mathbf{1} = (1, \cdots, 1)^T \in \mathbb{R}^n$. La variance vérifie la propriété suivante

$$V(x) = \frac{1}{n} \sum_{i=1}^{n} x_i^2 - \bar{x}^2 = \frac{1}{n} \|x\|^2 - \bar{x}^2. \tag{5.2}$$

La variance est donc la différence entre la moyenne des carrés et le carré de la moyenne.

⋆ *Ecart-type de* x : $\sigma_x = \sigma(x) = \sqrt{V(x)} = \frac{1}{\sqrt{n}} \|x - \bar{x}\mathbf{1}\|$.

Un vecteur a même écart-type que ce vecteur auquel on a enlevé sa moyenne :

$$\sigma(x) = \sigma(x - \bar{x}\mathbf{1}). \tag{5.3}$$

⋆ *Covariance de x et y :* $\operatorname{cov}(x,y) = \sigma_{xy} = \dfrac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y}) = \dfrac{1}{n}(x - \bar{x}\mathbf{1} \,|\, y - \bar{y}\mathbf{1})$. La covariance vérifie les propriétés suivantes

$$\sigma_{xy} = \frac{1}{n}(x \,|\, y) - \bar{x}\bar{y} \quad \text{et } \sigma_{xx} = \sigma_x^2 = V(x). \tag{5.4}$$

⋆ *Coefficient de corrélation linéaire* entre deux vecteurs x et y :

$$r(x,y) = \frac{\sigma_{xy}}{\sigma_x \sigma_y}.$$

On montre qu'on a toujours $-1 \leq r \leq 1$ et que $|r(x,y)| = 1$ si et seulement s'il existe a, b, $c \in \mathbb{R}$ tels que $ax_i + by_i + c = 0$, pour tout $i = 1, \cdots, n$. Le coefficient de corrélation r mesure donc le caractère linéaire d'une relation entre les deux vecteurs x et y. En fait, on montre sans difficulté que

$$r(x,y) = \cos(\theta), \tag{5.5}$$

où θ est l'angle entre les vecteurs $(x - \bar{x}\mathbf{1})$ et $(y - \bar{y}\mathbf{1})$.

5.3 Quelques rappels sur les valeurs et vecteurs propres

Dans ce paragraphe, A désigne une matrice carrée d'ordre n à coefficients réels. Les coefficients de A sont notés a_{ij}.

⋆ *Valeurs propres et vecteurs propres.*

- Les n valeurs propres de A notées $\lambda_1, \lambda_2, ..., \lambda_n$, sont les n racine réelles ou complexes du polynôme caractéristique de A défini par $p(\lambda) = \det(A - \lambda I)$ (polynôme de degré n en λ). Les coefficients de A étant réels, si λ est valeur propre de A alors la valeur conjuguée $\overline{\lambda}$ est aussi valeur propre.
- Un vecteur $u \neq 0$ de $\mathbb{C}^n$ est un vecteur propre de A associé à la valeur propre $\lambda \in \mathbb{C}$ si et seulement si $Au = \lambda u$. Un vecteur propre est donc connu à une constante multiplicative près. On prendra toujours (sauf mention contraire) les vecteurs propres normés i.e. de norme unité.
- Si A est *symétrique* alors toutes les valeurs propres de A sont *réelles*.
- Les matrices A et A^T ont même valeurs propres.
- La trace de A vérifie $\operatorname{trace}(A) := \sum_{i=1}^{n} a_{ii} = \sum_{i=1}^{n} \lambda_i$.
- Soient M et N deux matrices de tailles respectives $n \times p$ et $p \times n$. Alors les matrices produits MN et NM ont mêmes valeurs propres *non-nulles*. De plus, si $u \neq 0$ est un vecteur propre de MN associé à $\lambda \neq 0$ et alors $Nu \neq 0$ est un vecteur propre de NM associé à λ.

⋆ *Diagonalisation.*

- On dit que A est semblable à une matrice B si et seulement s'il existe une matrice P inversible telle que $A = PBP^{-1}$. Dans ce cas, les matrices A et B ont mêmes valeurs propres.
- A est diagonalisable si et seulement si A est semblable à une matrice diagonale et dans ce cas, il existe une matrice P inversible telle que $A = PDP^{-1}$ avec $D = \operatorname{diag}(\lambda_1, ..., \lambda_n)$ matrice diagonale formée des valeurs propres de A. De plus, la i-ième colonne de P est un vecteur propre de A associé à λ_i. Une autre caractérisation est que A est diagonalisable si et seulement si A possède n vecteurs propres linéairement indépendants (qui forment alors une base de $\mathbb{C}^n$).
- Si A est symétrique alors toutes les valeurs propres de A sont réelles et A est diagonalisable avec $A = QDQ^T$ où $D = \operatorname{diag}(\lambda_1, ..., \lambda_n)$ et Q est une matrice orthogonale ($Q^{-1} = Q^T$). Les n vecteurs propres (normés) de A constituant les colonnes de la matrice Q, sont orthogonaux (orthonormés) Ils forment donc une base orthonormée de $\mathbb{R}^n$.

⋆ *Interprétation géométrique.*

Pour simplifier, on se place en dimension 2 et on considère le cercle centré en 0 et de rayon 1. L'image de ce cercle par une application linéaire représentée (dans la base canonique) par une matrice M *symétrique*, est une ellipse centrée en 0 dont les deux axes sont donnés par les vecteurs propres de M. Par l'exemple, la figure 5.1 montre bien que l'ensemble des points $y = Mx$ avec $\|x\| = 1$ définit l'ellipse centrée en 0 dont les axes sont les vecteurs propres de M et les longueurs des demi-axes sont les valeurs propres correspondantes. Ici $M = \begin{pmatrix} 5 & 4 \\ 4 & 1 \end{pmatrix}$ qui a pour valeurs propres $\lambda_1 = \overline{OP_1} = 7.47$, $\lambda_2 = \overline{OP_2} = -1.47$ et les vecteurs propres associés sont définis par $\overrightarrow{OP_1}$ et $\overrightarrow{OP_2}$.

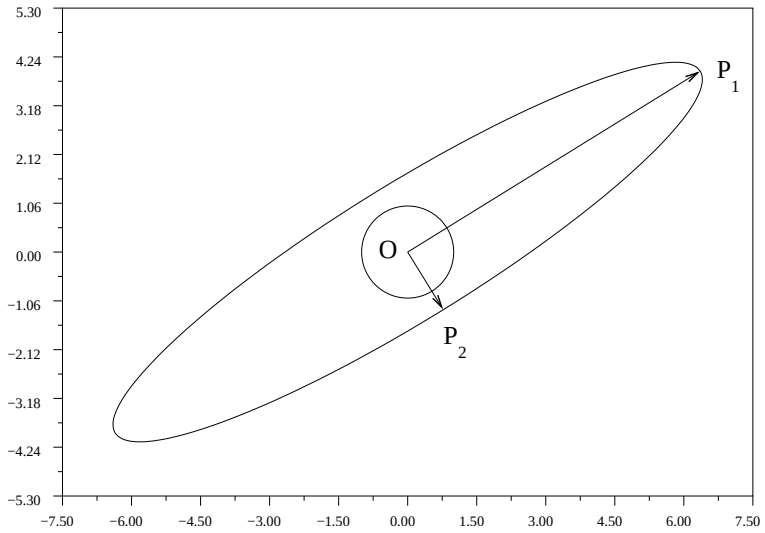

FIG. 5.1 – *Image du cercle unité par une application linéaire.*

⋆ *Quotient de Rayleigh - relations de Courant-Fischer.*

Si la matrice A est symétrique, on définit pour $x \in \mathbb{R}^n, x \neq 0$, le quotient de Rayleigh par la quantité scalaire

$$R_A(x) = \frac{(Ax \,|\, x)}{(x \,|\, x)}.$$

Cette quantité fournit la caractérisation suivante de la plus grande et de la plus petite valeur propre de A. Si on range les valeurs propres de A de façon décroissante : $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$, alors on a les relations suivantes, appelées relations de Courant-Fischer.

$$\lambda_1 = \max_{\substack{x \in \mathbb{R}^n \\ x \neq 0}} R_A(x), \qquad \lambda_n = \min_{\substack{x \in \mathbb{R}^n \\ x \neq 0}} R_A(x). \tag{5.6}$$

5.4 Variables centrées réduites

Les données *initiales* sont représentées par une matrice $X \in \mathcal{M}_{n\times p}(\mathbb{R})$ (en général $n \geq p$) où n est le nombre d'individus et p le nombre de variables. Les colonnes x^j de X correspondent donc aux variables et les lignes x_i aux individus. Les vecteurs x^j $(j = 1, \cdots, p)$ sont donc des vecteurs (colonnes) de $\mathbb{R}^n$ alors que les x_i $(i = 1, \cdots, n)$ sont des vecteurs (lignes) de $\mathbb{R}^p$. Les coefficients de la matrice X seront notés indifféremment x_{ij} ou bien x_i^j.

Dans l'analyse des données, on veut éviter deux effets. D'une part, les effets dus aux moyennes et d'autre part les effets de dispersion dus au fait qu'en général les variables ont des unités de mesure différentes et non comparables. On veut donc rendre les données plus *homogènes*. Pour cela, on va centrer puis réduire les variables.

Individus	**Revenu**	**Superficie**	**Soleil**
1	- 0.675	- 0.942	0.991
2	- 0.488	- 0.316	1.651
3	- 0.592	- 0.619	0.330
4	- 0.216	0	- 0.330
5	- 0.112	- 0.449	0
6	0.347	0.316	1.321
7	1.265	0.954	1.651
8	3.309	3.494	- 0.661
9	0.138	- 1.264	- 0.991
10	- 0.363	- 0.278	0.661
11	2.12	1.472	1.321
12	- 0.050	0.960	0.991
13	- 0.905	- 0.436	- 0.991
14	- 0.988	- 0.310	- 1.651
15	- 0.237	- 0.019	- 1.321
16	0.785	0.348	- 0.330
17	- 0.029	- 0.164	0.991
18	- 0.216	0.960	- 1.651
19	- 1.197	- 0.341	0.330
20	- 0.446	- 0.613	- 0.330
21	0.347	- 0.164	- 0.661
22	- 0.216	- 0.423	- 0.991
23	- 0.863	- 0.942	- 0.330
24	- 0.717	- 1.226	0
moyenne	0	0	0
écart-type	1	1	1

$$Z = \begin{pmatrix} - 0.675 & - 0.942 & 0.991 \\ - 0.488 & - 0.316 & 1.651 \\ - 0.592 & - 0.619 & 0.330 \\ - 0.216 & 0 & - 0.330 \\ - 0.112 & - 0.449 & 0 \\ 0.347 & 0.316 & 1.321 \\ 1.265 & 0.954 & 1.651 \\ 3.309 & 3.494 & - 0.661 \\ 0.138 & - 1.264 & - 0.991 \\ - 0.363 & - 0.278 & 0.661 \\ 2.12 & 1.472 & 1.321 \\ - 0.050 & 0.960 & 0.991 \\ - 0.905 & - 0.436 & - 0.991 \\ - 0.988 & - 0.310 & - 1.651 \\ - 0.237 & - 0.019 & - 1.321 \\ 0.785 & 0.348 & - 0.330 \\ - 0.029 & - 0.164 & 0.991 \\ - 0.216 & 0.960 & - 1.651 \\ - 1.197 & - 0.341 & 0.330 \\ - 0.446 & - 0.613 & - 0.330 \\ 0.347 & - 0.164 & - 0.661 \\ - 0.216 & - 0.423 & - 0.991 \\ - 0.863 & - 0.942 & - 0.330 \\ - 0.717 & - 1.226 & 0 \end{pmatrix}$$

TAB. 5.2 – *Les données centrées réduites et la matrice associée.*

On centre les variables afin d'éviter les effets de moyennes : pour chaque variable, on retranche la moyenne à chacune de ses composantes. On obtient ainsi une matrice Y (de même taille que X) dont les colonnes y^j sont données par

$$\boxed{y^j = x^j - E(x^j)\mathbf{1}} \quad \text{soit encore} \quad y_i^j = x_i^j - E(x^j). \tag{5.7}$$

On a évidemment que la moyenne de y^j est *nulle* : les variables y^j sont *centrées*.

On réduit ensuite les variables afin d'éviter les effets de dispersion : pour chaque variable centrée, on divise chaque composante par l'écart-type de la variable (centrée ou non car $\sigma(x^j) = \sigma(y^j)$, cf. la propriété (5.3)). On obtient ainsi une matrice Z (de même taille que X) dont les colonnes z^j sont données par

$$z^j = \frac{y^j}{\sigma(x^j)},$$

soit sous forme matricielle

$$\boxed{Z = YD_{1/\sigma}} \quad \text{où} \quad D_{1/\sigma} = \begin{pmatrix} 1/\sigma(x^1) & & 0 \\ & \ddots & \\ 0 & & 1/\sigma(x^p) \end{pmatrix}. \tag{5.8}$$

La matrice $D_{1/\sigma}$ est diagonale et de taille $p \times p$. Le fait de diviser par l'écart-type rend les nouvelles variables "sans dimension" car l'écart-type d'une variable a la même unité (ou dimension) que celle-ci. Les variables z^j sont donc sans unité et leurs écart-types valent 1. De plus, leurs moyennes sont toujours nulles. Les variables z^j sont *centrées* et *réduites* (cf. tableau 5.2).

5.5 Représentation des individus

On cherche à représenter le *nuage* des points-individus $\{x_i \in \mathbb{R}^p, i = 1, \cdots, n\}$. La notion de moyenne du nuage se traduit ici par celle de centre de gravité (ou barycentre) et est défini par le vecteur g de $\mathbb{R}^p$ donné par

$$g = \frac{1}{n}\sum_{i=1}^{n} x_i.$$

Nous allons développer et étudier la notion d'inertie du nuage, cette notion étant à la base de l'ACP.

5.5.1 Inertie par rapport à un point

Soit y un vecteur donné de $\mathbb{R}^p$. On définit l'inertie $I(y)$ du nuage de points $\{x_i\}$ par rapport à y, par

$$I(y) = \frac{1}{n}\sum_{i=1}^{n} d^2(x_i, y), \tag{5.9}$$

où $d(x, y) = \|x - y\|$ est la distance euclidienne entre x et y (cf. figure 5.2).

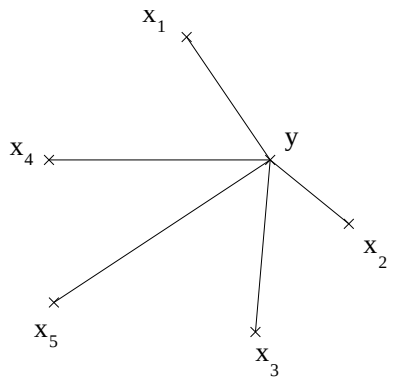

FIG. 5.2 – *Distance euclidienne du nuage $\{x_i\}$ au point y.*

La question que l'on se pose à présent est de déterminer le vecteur y pour que l'inertie soit *minimale*. C'est l'objet du résultat suivant.

Proposition 1 : *Théorème de Huygens.*
Pour tout $y \in \mathbb{R}^p$, on a

$$I(y) = I(g) + d^2(g, y). \tag{5.10}$$

Par conséquent, l'inertie du nuage de points par rapport au centre de gravité, est minimale.

Démonstration. On part de la définition de l'inertie. On a

$$\begin{aligned} I(y) &= \frac{1}{n}\sum_i \|x_i - y\|^2 = \frac{1}{n}\sum_i (x_i - y \,|\, x_i - y) \\ &= \frac{1}{n}\sum_i (x_i - g + g - y \,|\, x_i - g + g - y) \\ &= \frac{1}{n}\sum_i \left(\|x_i - g\|^2 + 2(x_i - g \,|\, g - y) + \|g - y\|^2\right) \\ &= I(g) + \frac{2}{n}\sum_i (x_i - g \,|\, g - y) + \|g - y\|^2. \end{aligned}$$

Or $\sum_i (x_i - g \,|\, g - y) = (\sum_i (x_i - g) \,|\, g - y) = 0$, car $\sum_i (x_i - g) = 0$ par définition du centre de gravité. On obtient ainsi l'identité recherchée. Il est alors clair qu'en prenant $y = g$, l'inertie est minimale. □

5.5.2 Inertie par rapport à une droite

Soit Δ une droite donnée de $\mathbb{R}^p$. On définit l'inertie du nuage des points-individus par rapport à la droite Δ, par la quantité

$$I(\Delta) = \frac{1}{n}\sum_{i=1}^{n} d^2(x_i, \Delta), \tag{5.11}$$

où $d(x_i, \Delta) = d(x_i, p_i) = \|x_i - p_i\|$ avec p_i le projeté orthogonal de x_i sur la droite Δ (cf. figure 5.3).

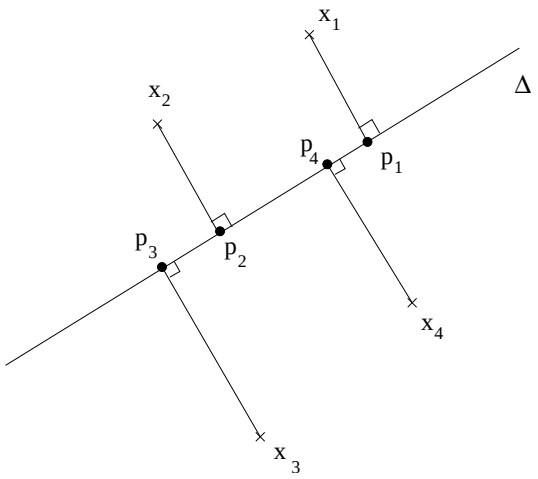

FIG. 5.3 – *Projection orthogonale du nuage de points sur la droite* Δ.

5.5.3 Minimisation de l'inertie

Le principe de l'ACP pour représenter les individus, consiste à minimiser l'inertie $I(\Delta)$ du nuage des points-individus, par rapport à la droite Δ. Autrement dit, il s'agit d'un problème de minimisation qui s'écrit

$$(P_\Delta) \quad \boxed{\text{Trouver } \Delta \text{ qui réalise } \min_{\Delta \in \mathbb{R}^p} I(\Delta).}$$

On va résoudre le problème (P_Δ) en travaillant désormais avec les variables centrées réduites représentées par la matrice Z. L'inertie du nuage par rapport à Δ s'écrit donc

$$I(\Delta) = \frac{1}{n}\sum_{i=1}^{n} d^2(z_i, \Delta)$$

et le centre de gravité est le vecteur nul i.e. $g = 0$ car les variables sont centrées. La droite Δ de $\mathbb{R}^p$ est caractérisée par un vecteur directeur $u \neq 0$ et sa distance à l'origine $g = 0$. On notera a le projeté orthogonal de g sur Δ (cf. figure 5.4). La droite Δ est ainsi complètement déterminée par u et a. De plus, tout point P associé à un vecteur p, qui appartient à la droite Δ peut s'écrire $p = \alpha u + a$ où $\alpha \in \mathbb{R}$.

Pour résoudre le problème (P_Δ), on a besoin du résultat suivant.

Proposition 2 :

Pour toute droite Δ de $\mathbb{R}^p$, on a

$$I(\Delta) = I(g) + d^2(g, a) - \frac{1}{n}\frac{\|Zu\|^2}{\|u\|^2}, \tag{5.12}$$

où u est un vecteur directeur de Δ et a est le projeté orthogonal de g sur Δ.

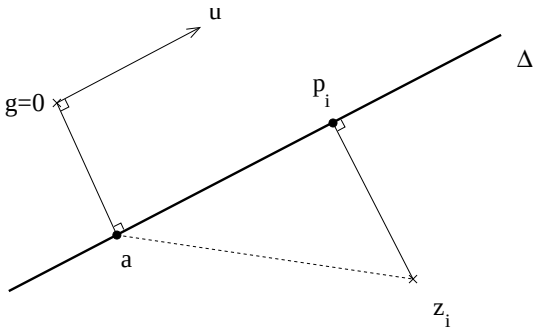

FIG. 5.4 – *Projection orthogonale du centre de gravité g sur la droite Δ.*

Démonstration. On part de la définition de l'inertie par rapport à Δ. On a $I(\Delta) = \dfrac{1}{n}\sum_i d^2(z_i, p_i)$. Par le théorème de Pythagore (toujours vrai dans $\mathbb{R}^p$!), on a la relation (cf. figure 5.4) $d^2(a, z_i) = d^2(z_i, p_i) + d^2(a, p_i)$ et donc

$$I(\Delta) = \frac{1}{n}\sum_i d^2(a, z_i) - \frac{1}{n}\sum_i d^2(a, p_i) = I(a) - \frac{1}{n}\sum_i d^2(a, p_i).$$

Or d'après le théorème de Huygens (5.10) relatif à l'inertie du nuage par rapport à un point, on a $I(a) = I(g) + d^2(g, a)$ et par conséquent

$$I(\Delta) = I(g) + d^2(g, a) - \frac{1}{n}\sum_i \|a - p_i\|^2. \tag{5.13}$$

Par ailleurs, on a $p_i = \alpha_i u + a$ où $\alpha_i \in \mathbb{R}$. Le scalaire α_i peut être déterminé en utilisant la définition du projeté orthogonal, c'est-à-dire $(z_i - p_i \,|\, p_i - a) = 0$. En effet, de cette dernière relation, on déduit que

$$0 = (z_i - p_i \,|\alpha_i u) = \alpha_i (z_i - a - \alpha_i u \,|\, u),$$

d'où l'on tire (si $\alpha_i \neq 0$) que $(z_i \,|\, u) - (a|u) - \alpha_i \|u\|^2 = 0$. Or $(a \,|\, u) = 0$ donc il vient

$$\alpha_i = \frac{(z_i \,|\, u)}{\|u\|^2}.$$

Enfin, on a $\|a - p_i\|^2 = \alpha_i^2 \|u\|^2 = \dfrac{(z_i \,|\, u)^2}{\|u\|^2}$. Il suffit alors de remarquer que $(z_i \,|\, u) = (Zu)_i$ et que $\sum_i ((Zu)_i)^2 = \|Zu\|^2$. La relation (5.13) devient ainsi

$$I(\Delta) = I(g) + d^2(g, a) - \frac{1}{n}\frac{\|Zu\|^2}{\|u\|^2},$$

ce qui termine la démonstration. □

Grâce à la Proposition 2, on est en mesure de résoudre le problème de minimisation (P_Δ). C'est l'objet du résultat suivant.

Proposition 3 : *Minimisation de l'inertie du nuage par rapport à une droite.*

1. La droite qui minimise l'inertie du nuage passe par le centre de gravité ($g = a = 0$).
2. Un vecteur directeur de la droite qui minimise l'inertie du nuage est le vecteur propre normé de la matrice carrée symétrique $\dfrac{1}{n}Z^T Z$ de taille $p \times p$, associé à la plus grande valeur propre.

Démonstration. Il est immédiat d'après (5.12) que si l'inertie est minimale alors nécessairement $a = g$ c'est-à-dire qu'on peut se restreindre aux droites passant par le centre de gravité du nuage. L'expression de l'inertie se réduit alors à

$$I(\Delta) = I(g) - \frac{1}{n}\frac{\|Zu\|^2}{\|u\|^2}.$$

Ainsi minimiser $I(\Delta)$ (par rapport à u) est équivalent à maximiser $\frac{1}{n}\frac{\|Zu\|^2}{\|u\|^2}$.
Or, on a $\|Zu\|^2 = (Zu \,|\, Zu) = (Z^TZu \,|\, u)$, de sorte qu'on est ramené à maximiser par rapport à $u \neq 0$, la quantité $\frac{1}{n}\frac{(Z^TZu \,|\, u)}{\|u\|^2}$. Cette quantité n'est rien d'autre que le quotient de Rayleigh de la matrice *carrée symétrique* $\frac{1}{n}Z^TZ$ d'ordre p. Par conséquent d'après les relations de Courant-Fischer (5.6), le maximum vaut λ_1 la plus grande valeur propre de la matrice $\frac{1}{n}Z^TZ$. De plus, le maximum est atteint pour le vecteur propre associé à λ_1. □

Remarque. Les valeurs propres de la matrice $\frac{1}{n}Z^TZ$ sont toutes positives ou nulles (car $\frac{1}{n}\|Zu\|^2 = \lambda\|u\|^2$ si $u \neq 0$ est un vecteur propre associé à la valeur propre λ).

5.5.4 Axes principaux

La matrice $\frac{1}{n}Z^TZ$ de taille $p \times p$ étant symétrique, on ordonne ses p valeurs propres (réelles et positives ou nulles) de façon décroissante $\lambda_1 \geq \cdots \geq \lambda_p \geq 0$ et on note $u^1, \cdots, u^p$ les vecteurs propres *normés* correspondants.
Ces vecteurs propres sont orthonormés deux à deux et forment donc une base orthonormée (b.o.n) de $\mathbb{R}^p$. De plus, on a la diagonalisation

$$\frac{1}{n}Z^TZ = QDQ^T,$$

où D est la matrice diagonale de taille $p \times p$ formée des valeurs propres et Q est la matrice orthogonale de taille $p \times p$ dont les colonnes sont formées des vecteurs propres normés.

Définition 1 : *Axes principaux.*

On appelle *axes principaux d'inertie* les vecteurs propres normés $u^1, \cdots, u^p$ de la matrice $\frac{1}{n}Z^TZ$.

5.5.5 Matrice de variance-covariance, matrice de corrélation

Dans ce paragraphe, on fait le lien entre la matrice $\frac{1}{n}Z^TZ$ précédente et la matrice de corrélation des données X initiales. On rappelle que les variables initiales sont notées x^j et forment les colonnes de X.

Définition 2 :

1. On appelle *matrice de variance-covariance* des données X, la matrice *symétrique* V de taille $p \times p$ définie par

$$V = \begin{pmatrix} \sigma_{11} & \sigma_{12} & \cdots & \sigma_{1p} \\ \sigma_{12} & \sigma_{22} & \cdots & \sigma_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{1p} & \sigma_{2p} & \cdots & \sigma_{pp} \end{pmatrix} = \frac{1}{n}Y^TY, \tag{5.14}$$

où $\sigma_{ij} = \sigma_{x^ix^j} = \mathrm{cov}(x^i, x^j)$ est la covariance de x^i et x^j.

2. On appelle <u>*matrice de corrélation*</u> des données X, la matrice *symétrique* R de taille $p \times p$ définie par

$$R = \begin{pmatrix} 1 & r_{12} & \cdots & r_{1p} \\ r_{12} & 1 & \cdots & r_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ r_{1p} & r_{2p} & \cdots & 1 \end{pmatrix}, \tag{5.15}$$

où $r_{ij} = r(x^i, x^j)$ est le coefficient de corrélation linéaire entre x^i et x^j.

On rappelle que les données centrées et réduites Z sont obtenues à partir de Y par (5.8) c'est-à-dire $Z = YD_{1/\sigma}$. On a donc

$$\frac{1}{n}Z^TZ = \frac{1}{n}(D_{1/\sigma}^T Y^T)YD_{1/\sigma} = D_{1/\sigma}VD_{1/\sigma}. \tag{5.16}$$

Par ailleurs, on a le lien suivant entre la matrice de corrélation et les données centrées réduites :

$$R = \frac{1}{n}Z^TZ. \tag{5.17}$$

En effet, $\frac{1}{n}\left(Z^TZ\right)_{i,j} = \frac{1}{n}(z^i \,|\, z^j) = \frac{1}{n}\left(\frac{x^i - E(x^i)\mathbf{1}}{\sigma(x^i)} \,|\, \frac{x^j - E(x^j)\mathbf{1}}{\sigma(x^j)}\right) = \frac{\mathrm{cov}(x^i, x^j)}{\sigma(x^i)\sigma(x^j)} = r(x^i, x^j)$, ce qui établit bien (5.17). Par conséquent, en combinant (5.16) et (5.17), on obtient

$$\boxed{R = \frac{1}{n}Z^TZ = D_{1/\sigma}VD_{1/\sigma},} \tag{5.18}$$

où la matrice $D_{1/\sigma}$ est donnée par (5.8).

On remarquera enfin que les coefficients de corrélation sont inchangés lorsqu'on centre et réduit les variables. Ainsi, on a

$$\boxed{r(x^i, x^j) = r(x^i, z^j) = r(z^i, z^j)}. \tag{5.19}$$

Exemple

Dans l'exemple étudié (cf. tableaux 5.1 et 5.2), la matrice de corrélation vaut (avec $n = 24$)

$$R = \frac{1}{n}\, Z^TZ = \begin{pmatrix} 1 & 0.845 & 0.170 \\ 0.845 & 1 & 0.084 \\ 0.170 & 0.084 & 1 \end{pmatrix}$$

et admet comme diagonalisation

$$R = QDQ^T \text{ avec } Q = \begin{array}{c} \begin{matrix} q^1 & q^2 & q^3 \end{matrix} \\ \begin{pmatrix} -0.698 & -0.711 & -0.090 \\ -0.688 & 0.700 & -0.193 \\ -0.200 & 0.073 & 0.977 \end{pmatrix} \end{array} \text{ et } D = \begin{pmatrix} 1.881 & 0 & 0 \\ 0 & 0.151 & 0 \\ 0 & 0 & 0.968 \end{pmatrix}.$$

Les 2 premières valeurs propres retenues sont donc $\lambda_1 = 1.881$ et $\lambda_2 = 0.968$ et les axes principaux sont $u^1 = q^1$, $u^2 = q^3$.

5.5.6 Représentation graphique des individus

Pour représenter les individus, on calcule les coordonnées des individus z_i centrés-réduits, dans la base orthonormée $\mathcal{U} = \{u^1, \cdots, u^p\}$ de $\mathbb{R}^p$ contituée des vecteurs propres de R . Ainsi, si on note $[z_i]_{\mathcal{E}}$ (resp. $[z_i]_{\mathcal{U}}$) le vecteur *colonne* de $\mathbb{R}^p$ formé des coordonnées de z_i dans la base canonique $\mathcal{E}$ (resp. dans la base $\mathcal{U}$) de $\mathbb{R}^p$, alors on a $[z_i]_{\mathcal{E}} = Q[z_i]_{\mathcal{U}}$. Puisque la matrice Q est orthogonale, on obtient

$$\boxed{[z_i]_{\mathcal{U}} = Q^T[z_i]_{\mathcal{E}}.} \tag{5.20}$$

Pour la représentation des individus, on ne retient en fait que les 2 premiers vecteurs propres u^1 et u^2 de R associés aux 2 plus grandes valeurs propres λ_1 et λ_2. Graphiquement, on ne représente que les projections des z_i dans le plan $(0, u^1, u^2)$ c'est-à-dire qu'on ne retient que les 2 premières composantes de $[z_i]_{\mathcal{U}}$ (voir la figure 5.5). On notera que l'origine 0 du repère orthonormé $(0, u^1, u^2)$ correspond au centre de gravité du nuage des points-individus initiaux x_j.

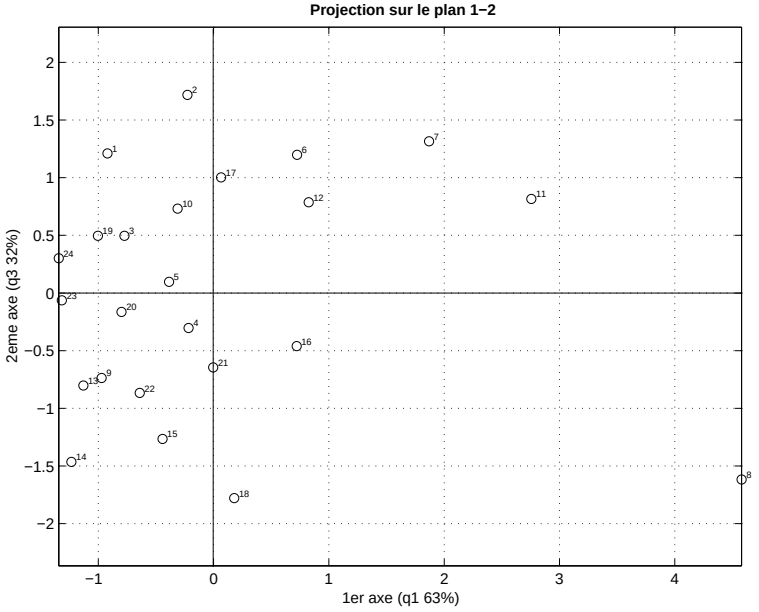

FIG. 5.5 – *Représentation des individus dans le plan formé par les deux premiers axes principaux.*

Qualité de la représentation. Un des objectifs de l'ACP étant de représenter les individus dans un espace de dimension plus petit que p, il faut apprécier la perte d'information ainsi subie. Un critère généralement utilisé est celui du *pourcentage d'inertie totale expliquée (ITE).* Il correspond à la proportion des deux premières valeurs propres λ_1 et λ_2 retenues ; il s'exprime donc par

$$ITE = \frac{\lambda_1 + \lambda_2}{\sum_{i=1}^{p} \lambda_i} = \frac{\lambda_1 + \lambda_2}{\text{trace}(R)} = \frac{\lambda_1 + \lambda_2}{p}. \tag{5.21}$$

Pour l'exemple étudié (voir la fin du § 5.5.5), on obtient les proportions suivantes des valeurs propres.

valeurs propres	%
$\lambda_1 = 1.881$	62.7
$\lambda_2 = 0.968$	32.3
$\lambda_3 = 0.151$	5.0

TAB. 5.3 – *Pourcentage des valeurs propres.*

Le pourcentage d'inertie totale expliquée est obtenu en faisant la somme des parts d'inertie des deux premières valeurs propres. On obtient ainsi $ITE = 62.7\% + 32.3\% = 95\%$. Ce résultat indique que le nuage des points-individus est très aplati sur un sous-espace à deux dimensions (un "galet" aplati en dimension 3) et donc que la représentation dans le plan des deux premiers axes principaux est très satisfaisante.

5.6 Composantes principales

Les p composantes principales c^i $(i = 1, \cdots, p)$ sont des vecteurs de $\mathbb{R}^n$ définis par combinaison linéaire des variables centrées réduites :

$$\boxed{c^i = Zu^i}. \tag{5.22}$$

En fait, la première composante principale c^1 est caractérisée par le résultat suivant.

Proposition 4 : La première composante principale $c^1 \in \mathbb{R}^n$ est la variable la plus liée aux variables x^j au sens où elle réalise

$$\max_{c\in\mathbb{R}^n} \sum_{j=1}^{p} r^2(c, x^j). \tag{5.23}$$

Démonstration. Par définition et propriété du coefficient de corrélation (cf. (5.19)), on a $r^2(c, x^j) = r^2(c, z^j) = \dfrac{\big(\mathrm{cov}(c, z^j)\big)^2}{V(c)V(z^j)}$. Or $V(z^j) = 1$ et $V(c) = \dfrac{1}{n}\|c - \bar{c}\mathbf{1}\|^2$. Par ailleurs, $\mathrm{cov}(c, z^j) = \dfrac{1}{n}(c - \bar{c}\mathbf{1} \,|\, z^j)$ car les z^j sont à moyenne nulle. Par conséquent

$$r^2(c, z^j) = \frac{1}{n}\frac{(c - \bar{c}\mathbf{1} \,|\, z^j)^2}{\|c - \bar{c}\mathbf{1}\|^2} \tag{5.24}$$

La relation précédente montre qu'on peut se restreindre aux vecteurs c à moyenne nulle et dans ce cas on a

$$r^2(c, z^j) = \frac{1}{n}\frac{(c \,|\, z^j)^2}{\|c\|^2}$$

Par ailleurs, on a $(c \,|\, z^j) = (c \,|\, Ze^j) = (Z^T c \,|\, e^j) = (Z^T c)_j$, où e^j désigne le j-ième vecteur de la base canonique de $\mathbb{R}^p$. On obtient donc

$$\sum_{i=1}^{p} r^2(c, z^j) = \frac{1}{n}\sum_{i=1}^{p} \frac{\big((Z^T c)_j\big)^2}{(c \,|\, c)} = \frac{1}{n}\frac{\|Z^T c\|^2}{(c \,|\, c)} = \frac{1}{n}\frac{(ZZ^T c \,|\, c)}{(c \,|\, c)}.$$

D'après les relations de Courant-Fischer (cf. (5.6)), le maximum est atteint pour le vecteur propre c^1 de la matrice $\dfrac{1}{n}ZZ^T$ de taille $n \times n$ associé à la plus grande valeur propre. Par rapport à R, cette matrice est obtenu en permutant le produit de la transposée Z^T avec Z. Par conséquent, on a bien que $c^1 = Zu^1$ où u^1 est le premier vecteur propre de R (voir les rappels sur les valeurs propres en section 5.3).

□

Les composantes principales définies par (5.22) ont les propriétés suivantes.

Proposition 5 : Le vecteur c^i est non nul si et seulement si $\lambda_i \neq 0$; si toutes les valeurs propres de R sont non nulles, les p composantes principales c^i sont orthogonales deux-à-deux. De plus, la variance de c^i vaut

$$V(c^i) = \lambda_i \tag{5.25}$$

Démonstration. Calculons le produit scalaire entre deux composantes : on a $(c^i \,|\, c^j) = (Zu^i \,|\, Zu^j) = (Z^T Z u^i \,|\, u^j) = n\lambda_i (u^i \,|\, u^j) = n\lambda_i \delta_{ij}$ ce qui établit bien l'orthogonalité pour des valeurs propres non nulles. On obtient en particulier $\frac{1}{n}\|c^i\|^2 = \lambda_i$ ce qui prouve bien que c^i est nulle ssi λ_i l'est. Les composantes étant centrées (moyenne nulle) on en déduit la variance $V(c^i) = \lambda_i$. □

5.7 Représentation des variables

On relie les composantes principales c^i aux variables initiales x^j en calculant les coefficients de corrélation linéaire $r(c^i, x^j)$ pour tout $i, j = 1, \cdots, p$. D'après (5.25), on a

$$r(c^i, x^j) = r(c^i, z^j) = \frac{\mathrm{cov}(c^i, z^j)}{\sigma(c^i)\sigma(z^j)} = \frac{1}{n}\frac{(c^i \,|\, z^j)}{\sqrt{\lambda_i}}.$$

Or $(c^i \,|\, z^j) = (c^i \,|\, Ze^j) = (Z^T c^i \,|\, e^j) = (Z^T c^i)_j = (Z^T Z u^i)_j = n(Ru^i)_j = n\lambda_i u^i_j$, ce qui entraine, pour tout $i, j = 1, \cdots, p$,

$$\boxed{r(c^i, x^j) = \sqrt{\lambda_i}\; u^i_j}. \tag{5.26}$$

Pour représenter les variables, on ne retient que les deux premières composantes principales c^1 et c^2 correspondants aux deux plus grandes valeurs propres λ_1 et λ_2 de la matrice R. On considère alors le plan $(0, c^1, c^2)$ où chaque variable x^j est repéré par un point d'abscisse $r(c^1, x^j)$ et d'ordonnée $r(c^2, x^j)$ c'est-à-dire que l'on représente les corrélations de chaque variable x^j par rapport aux deux composantes principales c^1 et c^2.

Dans l'exemple étudié, on obtient les coordonnées suivantes (cf. Tab. 5.4).

variables	coord. c_1 $r(c^1, x^j)$	coord. c_2 $r(c^2, x^j)$
x^1	0.9570	-0.0887
x^2	0.9434	-0.1897
x^3	0.2746	0.9612

TAB. 5.4 – *Coordonnées des variables dans le plan des composantes principales.*

On regroupe alors les résultats dans une figure où l'on fait également apparaître le cercle centré en 0 et de rayon 1, appelé *cercle des corrélations*. Les coefficients de corrélation étant toujours compris entre -1 et 1, ce cercle permet d'apprécier graphiquement les corrélations des variables avec les composantes principales. En fait, on peut montrer que les variables représentées sont toujours à l'intérieur du cercle des corrélations et que la représentation décrite ci-dessus n'est rien d'autre qu'une projection (pour un produit scalaire[1] autre que l'usuel) des variables centrées réduites z^j de $\mathbb{R}^n$ dans le plan formé des deux premières composantes principales. Or dans $\mathbb{R}^n$, les variables centrées réduites (donc de variances égales à 1) sont sur la sphère centrée en 0 et de rayon 1 (pour la norme associée au produit scalaire définissant la projection). Le cercle des corrélations est donc l'intersection de la sphère de $\mathbb{R}^n$ avec le plan formé par c^1 et c^2 et les projections sont nécessairement dans le cercle.

Ainsi, deux variables proches et situées près du cercle des corrélations sont nécessairement proches dans l'espace entier $\mathbb{R}^n$. En revanche, deux variables proches entre elles mais loin du cercle peuvent être très éloignées dans $\mathbb{R}^n$. Enfin, deux variables éloignées dans le plan des deux composantes principales, le sont nécessairement dans l'espace tout entier $\mathbb{R}^n$. Il faut bien garder à l'esprit que la représentation est en fait une projection et qu'il peut donc y avoir un effet de perspective.

Les résultats obtenus avec l'exemple étudié (cf. Tab. 5.4), sont reportés sur la figure 5.6.

[1] Le produit scalaire considéré est $(\cdot \,|\, \cdot)_n = \frac{1}{n}(\cdot \,|\, \cdot)$

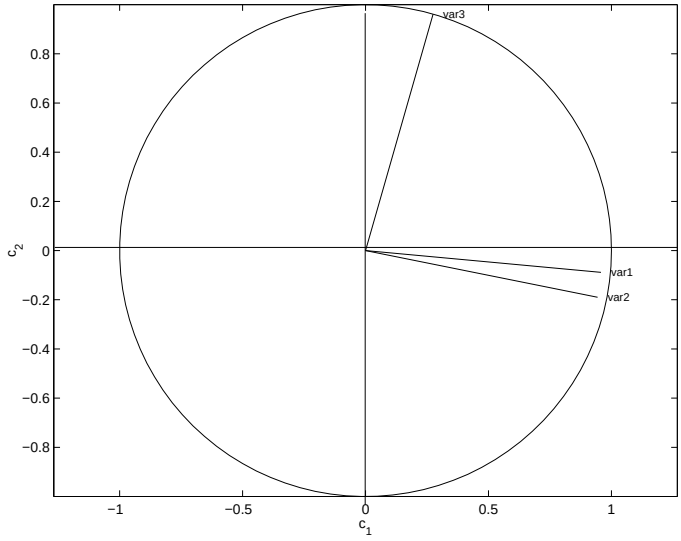

FIG. 5.6 – *Représentation des variables dans le plan formé par les deux composantes principales.*

Enfin, on peut regrouper le graphique de représentation des individus et celui des variables dans une même figure, ce qui donne la figure 5.7 pour l'exemple étudié dans ce chapitre.

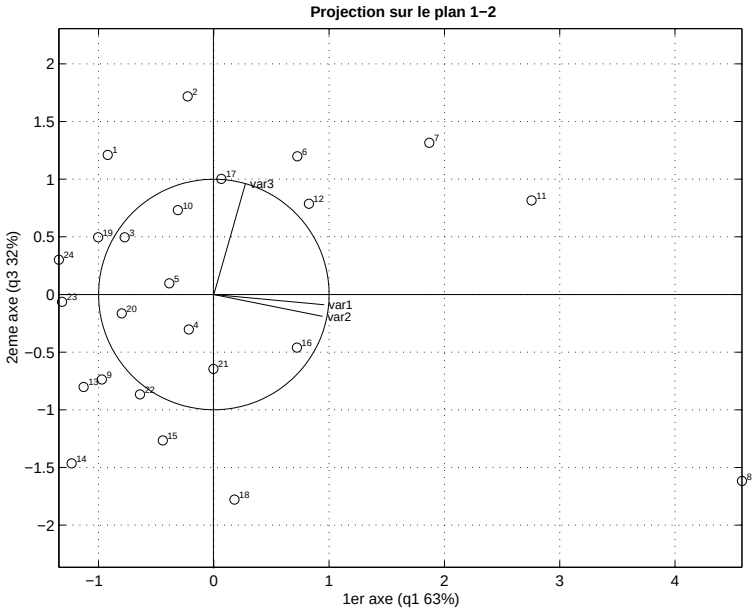

FIG. 5.7 – *Représentation des individus et des variables.*

Interprétation . Les variables x^1 et x^2 (le revenu et la superficie) sont fortement corrélées car elles sont toutes deux proches et situées à proximité du cercle des corrélations. En revanche la variable x^3 est non-corrélée avec les deux autres car la direction de la variable x^3 forme un angle approximativement droit avec les directions des deux autres variables : le facteur "soleil" est donc indépendant (non-corrélé) du revenu et de la superficie. Par ailleurs, la proximité des variables et des individus sur le graphique final (cf. figure 5.7) n'a pas réellement de signification ; c'est plutôt les directions définies par les variables qui comptent dans leurs relations avec les individus. Par exemple, dans la figure 5.7, l'individu **15** est proche de l'axe "soleil" et par conséquent il est dans la moyenne pour le revenu et la superficie mais sa note "soleil" est basse puisqu'il est éloigné de l'origine.

Chapitre 6

Introduction à l'analyse de données : Classification non hiérarchique.

6.1 Introduction

Les méthodes que nous allons étudier dans ce chapitre sont complémentaires de l'analyse en composantes principales. Notons que ce chapitre est une simple introduction au domaine de la classification non hiérarchique et on se contentera de présenter deux algorithmes très utilisés (il existe beaucoup d'autres méthodes ...).

On considère toujours des données (caractères) quantitatives portant sur n individus (p caractères par individu), résumées dans le tableau X et on cherche à partitionner ces n individus en K groupes (on dira classes dans la suite) les plus homogènes possibles avec (en général) $K << n$.

Un individu est donc caractérisé par un vecteur de $\mathbb{R}^p$. Comme pour l'ACP un traitement préliminaire (centrage et réduction par exemple) peut être effectué. On appelera néanmoins toujours X ce tableau des valeurs (après traitement préliminaire), et comme dans le chapitre précédent, X sera vu comme une matrice $n \times p$, l'individu numéro i correspondant toujours à la ligne i du tableau X, ce que l'on notera x_i (on utilisera parfois x pour désigner un individu sans préciser son numéro).

Pour classer les individus en plusieurs groupes, on a besoin de définir une distance entre deux individus quelconques (c-a-d entre deux points de $\mathbb{R}^p$) et l'on choisira la distance euclidienne. D'autres choix sont possibles, mais la distance euclidienne (en tout cas une distance issue d'un produit scalaire) est bien adaptée pour certaines raisons de simplicité mathématique et algorithmique (comme pour les moindres carrés) et convient généralement aux données modulo le traitement préliminaire habituel :

$$d(x_i, x_k) = ||x_i - x_k|| = \sqrt{\sum_{j=1}^{p} (x_i^j - x_k^j)^2}$$

Dans les algorithmes les plus simples, chaque groupe sera représenté par un individu « moyen » (le barycentre du groupe) et ainsi l'information initiale constituée des n individus, est alors résumée après classification par les K barycentres. Notons qu'une ACP sur chaque groupe peut ensuite être effectuée pour le caractériser plus précisemment que par son seul barycentre.

Le but du jeu est donc de trouver une partition des n individus en K groupes. Soit $\Pi = \{C_1, C_2, ..., C_K\}$ une telle partition (chaque classe (sous-ensemble) C_k étant non vide), on adoptera les notations suivantes :

- g_k désignera le barycentre de la classe C_k :

$$g_k = \frac{1}{n_k} \sum_{x \in C_k} x$$

  où $n_k = \#C_k$ est le nombre d'éléments de C_k ;
- I_k désignera l'inertie de la classe C_k par rapport à son barycentre :

$$I_k = \sum_{x \in C_k} d(x, g_k)^2 = \sum_{x \in C_k} ||x - g_k||^2$$

– $I(\Pi)$ désigne l'inertie totale associée à la partition Π :

$$I(\Pi) = \sum_{k=1}^{K} I_k$$

(remarque : les barycentres g_k et les inerties I_k dépendent bien sûr aussi du choix de la partition Π).

Avec ces notations, le problème de classer les n individus en K groupes, consiste à trouver une partition optimale $\bar{\Pi}$ qui réalise :

$$I(\bar{\Pi}) \leq I(\Pi), \forall \Pi \in P_{ad} \tag{6.1}$$

où P_{ad} est l'ensemble des partitions admissibles, c'est à dire l'ensemble de toutes les partitions de $\{x_1, x_2, ..., x_n\}$ en K sous-ensembles non vides.

Comme nous le verrons par la suite, ce problème est très difficile à résoudre (temps de calcul prohibitif), et on se contente généralement d'une solution « acceptable » ne réalisant pas forcément le minimum de la fonction inertie sur l'ensemble P_{ad}.

Remarques :

1. Dans le problème de la classification, on ne connaît pas, la plupart du temps, le bon nombre de classes K à utiliser ! Il n'y a d'ailleurs pas de réponses mathématiques satisfaisantes à cette question : si on veut simplement minimiser I alors $K = n$ convient, chaque classe étant alors réduite à un point... De plus les algorithmes qui permettent de classifier en changeant (au cours du déroulement de l'algorithme) le nombre de groupes sont assez difficiles à paramétrer ! Finalement il semble qu'une solution courante consiste à utiliser les algorithmes fonctionnant avec un nombre de classe fixé et de les essayer successivement avec différentes valeurs de K.
2. Les algorithmes que nous allons étudier dans ce chapitre, présentent tous un aspect stochatisque, dans le sens où ils partent d'une partition initiale, choisie au hasard. D'autre part, ils ne trouvent généralement pas la solution optimale et, pour un nombre fixe de classes K on est ainsi conduit à les essayer plusieurs fois sur des configurations initiales différentes, en retenant finalement la meilleure partition obtenue (au sens de l'inertie). Heureusement que ces algorithmes sont assez rapides !

6.2 Considérations générales sur le problème (6.1)

On peut remarquer que l'ensemble P_{ad} étant de dimension finie, un algorithme évident pour trouver une partition optimale (il peut y en avoir plusieurs), consiste à calculer l'inertie $I(\Pi)$ pour chaque $\Pi \in P_{ad}$ puis à retenir la (ou les) partition(s) optimale(s). Le problème est que le nombre d'éléments de P_{ad} à tendance à être gigantesque ! Appelons $P_{n,k}$ le nombre de partitions en k sous-ensembles non vides d'un ensemble E à n éléments et $\Pi_{n,k}(E)$ l'ensemble de toutes ces partitions. On a les résultats élémentaires suivants :

1. $P_{n,1} = 1$: en effet il est évident que $\Pi_{n,1}(E) = \{E\}$;
2. $P_{n,n} = 1$: E est partitionné en singletons et donc $\Pi_{n,n}(E) = \{\{\{e_1\}, \ldots, \{e_n\}\}\}$.

Calculons maintenant $P_{n,2}$. Rappelons que le nombre de sous-ensembles à k éléments d'un ensemble à n éléments est égal à C_n^k. Ainsi le nombre d'éléments de $\mathcal{P}(E)$ (l'ensemble des parties de E) est égal à :

$$\#\mathcal{P}(E) = \sum_{k=0}^{n} C_n^k = 2^n$$

car la formule du binôme nous donne :

$$2^n = (1+1)^n = \sum_{k=0}^{n} C_n^k 1^k 1^{n-k} = \sum_{k=0}^{n} C_n^k$$

L'ensemble $\Pi_{n,2}(E)$ est formé de paires du type $\{F, E \setminus F\}$ où F est un sous-ensemble non vide de E. Ainsi :

$$\Pi_{n,2}(E) = \bigcup_{F \in \mathcal{P}(E), F \neq \emptyset, F \neq E} \{\{F, E \setminus F\}\}$$

mais on remarque dans cette union que la même paire apparaît exactement deux fois $\{F, E \setminus F\}$ et $\{G, E \setminus G\}$ avec $G = E \setminus F$ et donc finalement :

$$\#\Pi_{n,2}(E) = P_{n,2} = \frac{2^n - 2}{2} = 2^{n-1} - 1$$

Exemple numérique : supposons que l'on cherche le meilleur partitionnement en deux groupes pour 100 individus par cette méthode de « force brute » , il faut calculer l'inertie pour les :

$$2^{99} - 1 = 633825300114114700748351602687$$

partitions possibles : no comment !

Avec ces cas simples, on peut maintenant calculer $P_{n,k}$ dans le cas général grâce à la formule de récurrence suivante :

$$P_{n,k} = P_{n-1,k-1} + kP_{n-1,k} \ , \ 1 < k < n$$

Démonstration : soit donc $\Pi_{n,k}(E)$ l'ensemble des partitions qui nous intéressent et $e \in E$ un élément quelconque de E. On peut scinder $\Pi_{n,k}(E)$ en 2 sous-ensembles distincts :

- Q_1 rassemblant les partitions dans lesquelles e apparait uniquement comme singleton (l'une des k classes de chaque partition est exactement $\{e\}$)
- et Q_2 constitué de toutes les autres partitions (dans lesquelles e n'apparait jamais dans un singleton !).

Il est clair que : $\#Q_1 = P_{n-1,k-1}$ puisque ses partitions peuvent être obtenues en prenant les partitions de $\Pi_{n-1,k-1}(E \setminus \{e\})$ et en leur rajoutant le singleton $\{e\}$. L'ensemble Q_2 lui peut se construire de la façon suivante : on forme $\Pi_{n-1,k}(E\setminus\{e\})$ et, pour chaque partition de cet ensemble, il suffit de rajouter l'élément e dans l'une quelconque des k classes, soit k choix possible par partition. D'où finalement $\#Q_2 = kP_{n,k-1}$ et le résultat cherché. □

Comme exercice (de récursivité) vous pouvez programmer cette fonction en Eiffel en utilisant la classe `LARGE_POSITIVE_INTEGER` ou encore (plus efficace) d'utiliser un algorithme « en triangle » (comme pour le triangle de Pascal).

Voici un tableau donnant quelques valeurs des $P_{n,k}$ (qui sont appelés nombre de *Stirling* de 2 ème espèce) :

$n \setminus k$	2	3	4	5	6
7	63	301	350	140	21
8	127	966	1701	1050	266
9	255	3025	7770	6951	2646
10	511	9330	34105	42525	22827
11	1023	28501	145750	246730	179487
12	2047	86526	611501	1379400	1323652
13	4095	261625	2532530	7508501	9321312
14	8191	788970	10391745	40075035	63436373

Malgré cette explosion combinatoire, quelques algorithmes permettant de trouver l'optimum sans tester toutes les partitions possibles, ont été mis au point, mais ils restent cantonnés à des valeurs de n très faibles (les plus perfectionnés permettent d'aller jusqu'à $n = 150$ à l'heure actuelle). Ainsi dans la plupart des cas pratiques, on est obligé de recourir à des algorithmes « *heuristiques* » . Néanmoins l'aspect dimension finie de ce problème intervient pour prouver que le nombre d'itérations de ces *heuristiques* est fini. Dans la pratique, la solution (en général sous optimale) est obtenue en une dizaine d'itérations, ce qui en fait des algorithmes très efficaces.

6.3 Quelques algorithmes

6.3.1 Introduction

Les deux algorithmes que nous allons détailler sont connus sous les noms de *K-means* et *H-means*, ce dernier étant appelé méthode des *centres mobiles* en français. Enfin ils apparaissent comme des cas

particuliers d'une méthode plus générale nommée *nuées dynamiques*. L'algorithme des *K-means* est une amélioration de celui des *H-means* : en général, il converge plus rapidement et on obtient une partition de meilleure qualité (au sens où l'inertie totale est plus faible). De plus, il peut améliorer une partition obtenue par les *H-means*, l'inverse étant faux. Cependant dans certains cas, partant de la même configuration initiale, l'algorithme des *H-means* peut donner une meilleure partition (qui est ensuite améliorable par *K-means* !) et il est donc bon d'avoir ces deux méthodes dans sa musette ! Notons que de nombreux auteurs donnent le nom de *K-means* à l'algorithme *H-means*...

Le résultat obtenu par ces algorithmes sur un cas particulier dépend (généralement) de la configuration (partition) initiale choisie. Nous allons tout d'abord détailler la méthode usuellement utilisée pour obtenir cette partition initiale.

6.3.2 Choix de la partition initiale

1. Tirer K nombres $m_{i_{(1\leq i\leq K)}}$ au hasard tous distincts dans l'ensemble $\{1, 2, \ldots, n\}$. Si on dispose d'une fonction $U(a, b)$ permettant de tirer uniformément un entier entre a et b (compris), on peut utiliser l'algorithme évident :

   pour k de 1 **à** K
   répéter
   $u \leftarrow U(1, n)$
   jusqu'à ce que $u \notin \{m_1, ..., m_{k-1}\}$
   $m_k \leftarrow u$
   fin pour

   On obtient ainsi un tirage uniforme d'un sous-ensemble à K élément de $\{1, 2, \ldots, n\}$.

2. Les points x_{m_i} sont choisis comme « centres » des classes initiales, la partition initiale $\Pi^{(0)} = \{C_1^{(0)}, ..., C_K^{(0)}\}$ étant obtenu de la façon suivante : la classe k est formée de tous les points qui lui sont les plus proches. En cas d'égalité de distance entre un point et plusieurs centres, le point est attribué à la classe de plus petit indice :

$$x_i \in C_k^{(0)} \iff k \text{ est le plus petit entier tel que } ||x_i - x_{m_k}|| \leq ||x_i - x_{m_l}|| \ \forall l \in [1, K]$$

   Il est clair que chaque classe C_k est non vide car elle comprend au moins le point x_{m_k}.

6.3.3 Algorithme des *H-means*

Du à *Forgy*, il est extrèmement simple et naturel, chaque itération étant constituée des deux phases suivantes :

phase de barycentrage : étant donné une partition, on calcule les barycentres de chaque classe ;

phase d'affectation : on boucle sur chaque point (individu) en le réaffectant à la classe dont le barycentre est le plus proche (avec affectation à la classe d'indice le plus petit si ambiguité). Cette phase modifie la partition de l'étape précédente.

Ce processus est itéré jusqu'à stabilisation de la partition. Il faut noter que lors de la phase d'affectation, on ne recalcule pas les barycentres lorsqu'un point change de classe. Cette remarque (et une autre) est à l'origine de l'algorithme des *K-means*. Dans certains cas, on peut simplement effectuer un nombre d'itérations donné sans forcément aller jusqu'à la stabilisation (cette stratégie s'explique du fait que l'inertie diminue assez rapidement lors des premières itérations puis plus lentement par la suite). Pour une écriture plus algorithmique, on va définir :

1. le tableau *classe* de taille n tel que $classe_i$ nous donne le numéro de la classe du point x_i ;
2. un tableau I de taille K donnant l'inertie de chaque classe ;
3. un tableau ne de taille K donnant le nombre d'éléments de chaque classe ;
4. un tableau G de taille $K \times p$, g_k servira à stocker le barycentre de la classe k ;
5. une variable *nbcht* donnant le nombre de points qui ont changé de classe lors de la phase de réaffectation.

L'algorithme donné (qui est relativement détaillé) n'est qu'une solution possible parmi d'autres, on écrira une version un peu différente en Matlab en TD... On peut remarquer aussi que le calcul du tableau des inerties n'est pas obligatoire mais il permet d'apprécier la qualité de la partition obtenue.

Algorithme *H-means*
entrées : X, classe (la partition initiale)
sorties : classe (partition finale), I, ...

répéter

phase de calcul des barycentres :

$g_k \leftarrow [0,0,0]$ et $ne_k \leftarrow 0$ pour $k = 1, 2, ..., K$
pour i de 1 à n
$k \leftarrow classe_i$
$g_k \leftarrow g_k + x_i$; $ne_k \leftarrow ne_k + 1$
fin pour
pour k de 1 à K
si $ne_k = 0$ **alors** *traiter l'exception* **fin si**
$g_k \leftarrow g_k / ne_k$
fin pour

phase d'affectation des points :

$nbcht \leftarrow 0$
$I_k \leftarrow 0$ pour $k = 1, 2, ..., K$
pour i de 1 à n
$d2min \leftarrow +\infty$; $kmin \leftarrow classe_i$
pour k de 1 à K
$temp \leftarrow ||x_i - g_k||^2$
si $temp < d2min$ **alors**
$d2min \leftarrow temp$; $kmin \leftarrow k$
fin pour
fin pour
si $kmin \neq classe_i$ **alors** le point a changé de classe
$classe_i \leftarrow kmin$; $nbcht \leftarrow nbcht + 1$
fin si
$I_{kmin} \leftarrow I_{kmin} + d2min$
fin pour

calcul de l'inertie totale :

$I_{totale} \leftarrow \sum_{k=1}^{K} I_k$

jusqu'à ce que $nbcht = 0$ si aucun point n'a changé de classe alors la stabilisation est obtenue

De façon exceptionnelle, une classe ou plusieurs classes peuvent se « vider ». Si l'on ne veut pas traiter ce cas comme une exception (arrêt du déroulement du programme, puis gestion de l'erreur ...), il est possible de continuer l'algorithme en réaffectant des points aux classes qui se sont vidées. Soit T le nombre de ces classes vides, alors on prend T points dans les autres classes (en privilégiant les classes avec une forte inertie) qui deviennent les nouveaux centres (et uniques points) de ces T classes.

Convergence de l'algorithme

On va l'établir sous l'hypothèse de non-dégénérescence, c-a-d que l'on supposera qu'aucune classe ne se vide. Cependant si on utilise la stratégie expliquée précédemment pour traiter ce problème il est simple de montrer que l'algorithme converge toujours. Par convergence, on sous-entend que la stabilisation de la partition est obtenue en un nombre fini d'itérations (et non que la solution obtenue est la (une) partition réalisant le minimum de l'inertie sur l'ensemble P_{ad}). On notera $I(\mathcal{C}, \Pi)$ l'inertie associée à un couple de

centres et une partition : ici on remarque bien que comme les barycentres ne sont pas mis à jour lors la phase d'affectation, ils sont simplement considérés comme les centres des classes. Avec cette notation, l'algorithme peut s'écrire de façon très abrégée :

$\Pi^{(0)}$ étant donnée
pour $m = 1, 2, ...$
$\mathcal{C}^{(m)}$ = barycentres des classes $\Pi^{(m-1)}$
$\Pi^{(m)}$ = partition obtenue après la phase de réaffectation

On considère alors la suite des inerties suivantes :

$$I_1 = I(\mathcal{C}^{(1)}, \Pi^{(0)}), I_2 = I(\mathcal{C}^{(1)}, \Pi^{(1)}), I_3 = I(\mathcal{C}^{(2)}, \Pi^{(1)}), I_4 = I(\mathcal{C}^{(2)}, \Pi^{(2)}), \ldots$$

où les inerties successives, $I_{2m-1} = I(\mathcal{C}^{(m)}, \Pi^{(m-1)})$ et $I_{2m} = I(\mathcal{C}^{(m)}, \Pi^{(m)})$ sont obtenues lors de l'itération m, suite, respectivement, au barycentrage puis à la réaffectation. Cette suite est décroissante. En effet, il est clair que :

$$I(\mathcal{C}^{(m)}, \Pi^{(m)}) \leq I(\mathcal{C}^{(m)}, \Pi^{(m-1)})$$

puisque le changement d'un point d'une classe à l'autre se fait sur le critère du centre le plus proche (à développer un peu ...). D'autre part, le barycentre d'un ensemble de points verifie (ici pour la classe k) :

$$\sum_{x \in C_k} ||x - g_k||^2 < \sum_{x \in C_k} ||x - y||^2 \ , \ \forall y \neq g_k$$

et donc il est aussi clair que :

$$I(\mathcal{C}^{(m)}, \Pi^{(m-1)}) \leq I(\mathcal{C}^{(m-1)}, \Pi^{(m-1)})$$

D'autre part comme cette suite est minorée (les inerties étant nécessairement positives) elle est donc convergente (suite décroissante minorée) : $\exists \bar{I} \in \mathbb{R}$ tel que $\lim_{k \to +\infty} I_k = \bar{I}$. En fait il y a mieux car on peut montrer que la suite devient stationnaire : $\exists \bar{k}$ tel que $I_k = \bar{I}$, $\forall k \geq \bar{k}$. En effet comme l'ensemble des partitions est de cardinal (élevé mais) fini, l'ensemble des centres envisagés par cet algorithme (qui sont à chaque fois les barycentres associés à une certaine partition) est aussi de cardinal fini et par conséquent l'ensemble des inerties $I(\Pi, \mathcal{C})$ est lui aussi de cardinal fini. Ainsi la suite que l'on envisage vit dans ce sous-ensemble fini de $\mathbb{R}$, et étant monotone, elle devient effectivement stationnaire à partir d'un certain rang (exercice). Si on prend m supérieur à ? alors :

$$I(\mathcal{C}^{(m-1)}, \Pi^{(m-1)} = I(\mathcal{C}^{(m)}, \Pi^{(m-1)})$$

et l'inégalité caractérisant les barycentres montre alors que, les centres n'ont pas changé ($\mathcal{C}^{(m)} = \mathcal{C}^{(m-1)}$) et comme les centres ne changent pas, les partitions non plus □.

6.3.4 Algorithme des *K-means*

Deux remarques sur l'algorithme *H-means* conduisent à l'algorithme *K-means* :

1. lorsque l'on affecte un point à une autre classe, les deux barycentres concernés ne sont pas corrigés ; ceci paraît naturel car il peut sembler onéreux de systématiquement recalculer ces barycentres à chaque changement ; nous verrons que cette opération peu se faire à moindre coût et que si les centres des classes correspondent toujours à leur barycentre, la phase (1) de l'algorithme *H-means* devient inutile.
2. en fait on peut remarquer qu'attribuer le point au centre le plus proche n'est pas toujours la meilleure option pour diminuer l'inertie ; la figure 6.1 illustre ce problème :
   (a) lors de la phase d'initialisation, les points numéros 3 et 4 ont été choisi respectivement comme centres inititiaux des classes 1 et 2 ; la classe 1 (en bleu) est donc formée par les points $\{x_1, x_2, x_3\}$ et la classe 2 ne contient que le point x_4 ; sur le dessin on a dessiné avec des croix les barycentres des deux classes ;

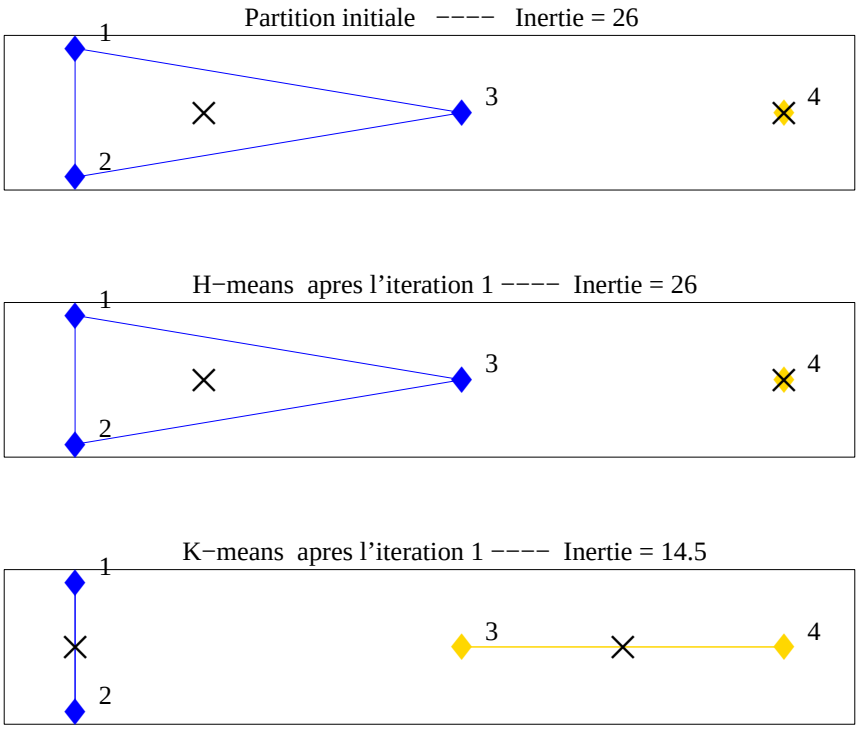

FIG. 6.1 – K-means versus H-means

(b) l'algorithme *H-means* ne change pas cette partition initiale car le point 3 est plus proche du barycentre de sa classe initiale que de celui de la classe 2 ;

(c) dans l'algorithme *K-means*, on envisage successivement pour chaque point, tous les changements de classe possibles en mesurant exactement la contribution de chaque changement à l'inertie finale (ce point sera détaillé ultérieurement), et l'algorithme découvre alors qu'attribuer le point 3 à la classe 2 permet de diminuer l'inertie ; la partition finale (obtenue aussi en 1 seule itération) est donc de meilleure qualité (au sens de l'inertie).

Ainsi l'algorithme *K-means*, après choix d'une partition initiale puis calcul des barycentres de chaque classe, consiste à effectuer jusqu'à convergence[1] des itérations constituées par une boucle sur les points :

> soit i le numéro du point courant et $l = classe_i$ sa classe actuelle, on envisage le passage de x_i dans chaque classe $k \neq l$, en calculant la modification $c_{i,k}$ de l'inertie obtenue par ce changement (si I l'inertie actuelle, alors $I + c_{i,k}$ donnerait la valeur de l'inertie suite à ce changement) ; on retient la valeur minimale des $c_{i,k}$ et la classe $\bar{k}$ correspondante[2] :
>
> $$\bar{c}_i = \min_{k \neq l} c_{i,k} \ , \quad \bar{k} = \min\{k | c_{i,k} = \bar{c}_i\}$$
>
> si $\bar{c}_i < 0$ alors le point i passe de la classe l à la classe $\bar{k}$ et l'on met à jour les barycentres des deux classes.

Remarques :

1. la convergence s'obtient lorsqu'il n'y a plus de changement de classe (ainsi pour le test d'arrêt on peut utiliser une variable qui compte le nombre de changements de classe dans la boucle sur les points) ;

[1] comme pour l'algorithme *H-means* on peut aussi fixer un nombre d'itérations maximum
[2] celle de plus petit indice si le minimum est atteind pour plusieurs choix

2. si x_i est le seul point de la classe l alors il est inutile d'envisager un changement dans une autre classe car l'inertie ne peut alors qu'augmenter (en effet l'inertie associée à la classe l est alors nulle vu que son barycentre est confondu avec le seul point de la classe). Cette remarque montre que si l'initialisation est correcte (pas de classe vide) alors l'algorithme ne peut pas dégénérer, c'est à dire qu'une classe ne peut pas se vider (contrairement à l'algorithme *H-means* [3]).

Calcul de $c_{i,k}$: le point x_i est actuellement dans la classe l et on envisage son passage dans la classe $k \neq l$; enfin on suppose que la classe l contient au moins deux points. On notera :
- $\hat{g}_l$ et $\hat{g}_k$ les barycentres après changement de classe,
- C_l et C_k les classes l et k actuelles (avant changement) et $n_l = \#C_l$, $n_k = \#C_k$ leur nombre d'éléments respectifs :
- $\hat{I}_l$ et $\hat{I}_k$ les inerties associées aux classes l et k après changement.

Tout d'abord remarquons que l'on peut exprimer assez facilement $\hat{g}_l$ et $\hat{g}_k$ en fonction de g_l et g_k (et donc avoir une mise à jour rapide des barycentres si le changement envisagé est effectif) :

$$\begin{aligned}
\hat{g}_l &= \frac{1}{n_l - 1} \sum_{x \in C_l \setminus \{x_i\}} x \\
&= \frac{1}{n_l - 1} \left(\sum_{x \in C_l} x - x_i \right) \\
&= \frac{1}{n_l - 1} (n_l g_l - x_i) \\
&= \frac{1}{n_l - 1} ((n_l - 1) g_l + g_l - x_i) \\
&= g_l + \frac{1}{n_l - 1} (g_l - x_i)
\end{aligned}$$

Un calcul analogue donne :

$$\begin{aligned}
\hat{g}_k &= \frac{1}{n_k + 1} \sum_{x \in C_k \cup \{x_i\}} x \\
&= \frac{1}{n_k + 1} (n_k g_k + x_i) \\
&= g_k + \frac{1}{n_k + 1} (x_i - g_k)
\end{aligned}$$

Passons maintenant au changement dans l'inertie : les deux termes modifiés sont les inerties des classes l et k : leurs contributions passe de $I_l + I_k$ à $\hat{I}_l + \hat{I}_k$ et ainsi $\hat{I} = I - (I_l + I_k) + (\hat{I}_l + \hat{I}_k)$ soit $c_{i,k} = (\hat{I}_l - I_l) + (\hat{I}_k - I_k)$. Développons $\hat{I}_l$:

$$\hat{I}_l = \sum_{x \in C_l \setminus \{x_i\}} d^2(\hat{g}_l, x) = \sum_{x \in C_l \setminus \{x_i\}} ||\hat{g}_l - x||^2$$

en remplaçant $\hat{g}_l$ par son expression en fonction de g_l :

$$\begin{aligned}
||\hat{g}_l - x||^2 &= ||g_l - x + \frac{g_l - x_i}{n_l - 1}||^2 \\
&= ||g_l - x||^2 + \frac{1}{(n_l - 1)^2} ||g_l - x_i||^2 + \frac{2}{n_l - 1} (g_l - x | g_l - x_i)
\end{aligned}$$

ainsi :

$$\hat{I}_l = I_l - ||g_l - x_i||^2 + \frac{1}{n_l - 1} ||g_l - x_i||^2 + \frac{2}{n_l - 1} \left(\sum_{x \in C_l \setminus \{x_i\}} g_l - x | g_l - x_i \right)$$

[3] où ce problème est néanmoins peu fréquent.

On utilise maintenant le fait que g_l est le barycentre de la classe l : $\sum_{x \in C_l} g_l - x = 0$ et donc :

$$\sum_{x \in C_l \setminus \{x_i\}} g_l - x = -(g_l - x_i)$$

et donc finalement :

$$\begin{aligned}
\hat{I}_l &= I_l - ||g_l - x_i||^2 + \frac{1}{n_l - 1}||g_l - x_i||^2 - \frac{2}{n_l - 1}(g_l - x_i | g_l - x_i) \\
\hat{I}_l &= I_l - ||g_l - x_i||^2 + \frac{1}{n_l - 1}||g_l - x_i||^2 - \frac{2}{n_l - 1}||g_l - x_i||^2 \\
\hat{I}_l &= I_l + \frac{-(n_l - 1) + 1 - 2}{n_l - 1}||g_l - x_i||^2
\end{aligned}$$

Soit :

$$\hat{I}_l - I_l = -\frac{n_l}{n_l - 1}||g_l - x_i||^2$$

Un calcul semblable nous donne :

$$\hat{I}_k - I_k = \frac{n_k}{n_k + 1}||g_k - x_i||^2$$

et donc la contribution apportée à l'inertie si le point x_i passait de la classe l à la classe k serait de :

$$c_{i,k} = \frac{n_k}{n_k + 1}||g_k - x_i||^2 - \frac{n_l}{n_l - 1}||g_l - x_i||^2$$

Avec ces derniers détails, on peut maintenant écrire un algorithme plus précis. La solution exposée reprend les notations précédemment utilisées pour l'algorithme *H-means*. D'autre part il est possible de calculer les inerties tout au début (après le calcul des barycentres) et de les mettre à jour à chaque changement de classe d'un point (il faut alors scinder le terme *temp* en deux pour apporter les modifications de I_l et $I_{\bar{k}}$) et la phase de post-traitement est alors inutile. On peut aussi rajouter une variable (qui additionnerait les termes $\bar{c}$) si l'on veut connaître la diminution de l'inertie sur une itération complète.

Algorithme *K-means*
entrées : X, classe (la partition initiale)
sorties : classe (partition finale), I, ...

initialisation : calcul des barycentres :
$g_k \leftarrow [0,0,0]$ et $ne_k \leftarrow 0$ pour $k = 1, 2, ..., K$
pour i de 1 à n
    $k \leftarrow classe_i$
    $g_k \leftarrow g_k + x_i$; $ne_k \leftarrow ne_k + 1$
fin pour
pour k de 1 à K
    $g_k \leftarrow g_k / ne_k$
fin pour

l'algorithme en question :
répéter :
    $nbcht \leftarrow 0$
    pour i de 1 à n
        $l \leftarrow classe_i$
        si $ne_l > 1$ **alors**
            $l \leftarrow classe_i$; $\bar{c} \leftarrow +\infty$
            pour k de 1 à K et $k \neq l$
                $temp \leftarrow \frac{ne_k}{ne_k+1}||g_k - x_i||^2 - \frac{ne_l}{ne_l-1}||g_l - x_i||^2$
                si $temp < \bar{c}$ **alors**
                    $\bar{c} \leftarrow temp$; $\bar{k} \leftarrow k$
                fin si
            fin pour
            si $\bar{c} < 0$ **alors** le point change de classe
                $classe_i \leftarrow \bar{k}$; $nbcht \leftarrow nbcht + 1$
                $g_l \leftarrow g_l + (g_l - x_i)/(ne_l - 1)$; $ne_l \leftarrow ne_l - 1$
                $g_{\bar{k}} \leftarrow g_{\bar{k}} + (x_i - g_{\bar{k}})/(ne_{\bar{k}} + 1)$; $ne_{\bar{k}} \leftarrow ne_{\bar{k}} + 1$
            fin si
        fin si
    fin pour
jusqu'à ce que $nbcht = 0$ si aucun point n'a changé de classe alors la stabilisation est obtenue

post traitement : calcul des inerties :
$I_k \leftarrow 0$ pour $k = 1, 2, ..., K$
pour i de 1 à n
    $k \leftarrow classe_i$
    $I_k \leftarrow I_k + ||x_i - g_k||^2$
fin pour