République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

UNIVERSITE M0HAMMED KHIDER BISKRA
FACULTE DES SCIENCES ET DES SCIENCES DE L'INGENEUR
DEPARTEMENT DES MATHEMATIQUE

Mémoire

Présenté en vue de J'obtention du diplôme de
Magister en Mathématiques

(Option: Analyse Mathématique)

Intitulé

Arithmétique en Virgule Flottante et Méthodes de Correction

Présenté Par

BENABBA FADHILA

Soutenu le / / 2005 ,devant. le Jury composé de :

Pr Brahim MEZERDI (Prof. à l'Université de Biskra)	Président
Dr Lamine MELKEMI (M. C. à l'Université de Batna)	Rapporteur
Dr Said BAHLALI (M. C. à l'Université de Biskra)	Examinateur
Dr Abdelkader ZIADI (M. C. à l'Université de Sétif)	Examinateur

Année Universitaire: 2005/2006

REMERCIEMENTS

Je tiens à exprimer ma reconnaissance à Monsieur Lamine MELKEMI Maître de Conférences à l'Université de Batna qui par sa direction professionnelle de ce mémoire, l'aide constante qu'il m'a prodiguée et ses conseils fructueux j'ai pu réaliser ce travail. Qu'il trouve ici mes remerciements les plus sincères.

C'est un grand plaisir pour moi de remercier vivement Monsieur Brahim MEZERDI Professeur à l'Université de Biskra pour l'honneur qu'il me fait en acceptant de présider le Jury.

C'est avec la même sincérité que j'adresse mes remerciements à Messieurs Said BAHLALI Maître de Conférences à l'Université de Biskra et Abdelkader ZIADI Maître de Conférences à l'Université de Sétif d'avoir accepté d'examiner ce mémoire et de faire partie du Jury.

Enfin je dis à toutes les personnes qui m'ont aidé directement ou indirectement à la réalisation de ce mémoire Merci.

مختصر.-(الحساب بالأعداد ذات الفاصلة المتنقلة و الطرق التصحيحية)

في هذه المذكرة نهتم بالحساب التقريبي بواسطة الأعداد المكتوبة بالفاصلة المتنقلة مع الدقة المنتهية. نتعرض إلى النموذج المعروف و إلى كيفية إنجاز العمليات الأولية. نشير إلى أن النظام الذي سنقدمه هو المعمول من طرف كل الحواسيب. نهتم بعدها بالطرق التصحيحية- التي تترصد بقدر الإمكان وبالشكل الذي سنبينه- الأخطاء المتراكمة في مراحل الطريقة الأولى. لقد تبين-كما سنرى- أن هذا المنهج يؤدي في أغلب الأحيان إلى تحسينات مفيدة في النتائج التقريبية المتحصل عليها من ذي قبل.

Abstract.- (Floating-point Arithmetic and correction methods)

In this memory we are interested to the floating-point arithmetic with finite precision. We present the different numerical aspects such as overflow, rounding and the standard model. Then we consider the correction methods that attempt to gather the rounding errors accumulated in the different steps of the first method. It turns out, as we will see, that such an approach often leads to important improvements in the yet obtained computed results.

Résumé-

Dans ce mémoire nous nous intéressons d'abord à l'arithmétique en virgule flottante avec précision finie. Nous présentons les différents aspects numériques comme les dépassements, les arrondis et le modèle standard. Ensuite nous considérons les méthodes de correction qui tentent de rassembler les erreurs d'arrondi accumulées dans les étapes de la première méthode. Il s'est avéré que cette approche conduit le plus souvent à d'importantes améliorations dans les solutions approchées déjà obtenues.

SOMMAIRE

Chapter 1

Introduction

Pour analyser un algorithme numérique, on se base logiquement sur le principe communément admis par la communauté des algorithmiciens et numériciens disant que, d'une façon générale, la performance d'une méthode numérique est caractérisée par quatre critères que nous exposons brièvement dans ce qui suit. Précisons au passage qu'informellement, on entend par algorithme toute structure formée d'un ensemble fini et ordonné (pas nécessairement totalement ordonné) de tâches élémentaires réalisant des opérations sur des éléments (données et/ou résultats intermédiaires) d'un certain ensemble, et ce pour obtenir les solutions requises par le problème à résoudre. Par définition un algorithme numérique manipule, entre autres, des nombres réels via des opérations comme $+, -, \times, /, \ldots$

(a) *Le critère du nombre d'opérations.* En ne tenant compte que des opérations coûteuses; c'est-à-dire celles qui requièrent un temps de calcul plus élelvé, il paraît tout à fait clair qu'un algorithme est d'autant plus rapide, donc plus performant, que le nombre d'opérations utilisées est réduit. En guise de motivation, rappelons les algorithmes permettant de calculer le produit de deux polynômes denses de degré n. Il est à présent connu que l'algorithme direct utilise $(n+1)^2$ multiplications alors qu'un autre algorithme moins évident peut être conçu de manière à ne demander que $O(n \log n)$ opérations. Cet algorithme fait appel à la FFT (Fast Fourier Transform) découverte en 1967 par Cooley et Tukey et qui réalise rapidement, comme son nom l'indique, la transformation discrète de Fourier. Pour plus de précisions sur ce point lié à la rapidité des algorithmes, on réfère le lecteur aux fameux livres de référence de Knuth[35], Aho

et al[1] et Cormen et al[14].

(b)*Le critère du nombre de données et résultats.* Comme précisé au début, un algorithme numérique manipule des paramètres, c'est-à-dire des données, des résultats et si besoin, des résultats intermédiaires. Tous ces paramètres vont naturellement avoir besoin d'être stockés en mémoire. D'habitude, on demande à l'algorithme d'afficher les données et les résultats si bien que le nombre de cases mémoires allouées soit nécessairement supérieur ou égal au nombre de ces données et ces résultats. Par conséquent, l'algorithme est d'autant plus économique, donc plus performant, que le nombre de cases mémoire allouées pour stocker les résultats intermédiaires est réduit. Bien qu'il s'agisse de cas rares on est parfois contraint à employer des techniques non triviales permettant de reduire sensiblement le stockage. C'est ainsi que grâce aux structures de déplacement par exemple on peut représenter l'inverse d'une matrice Toeplitz en déclarant un tableau linéaire uniquement (voir [40]).

(c) *Le critère du parallélisme.* Rappelons qu'un algorithme est vu comme un ensemble fini et ordonné de tâches élémentaires. Si T_1 et T_2 sont deux tâches comparables, alors $T_1 \prec T_2$ signifie que la réalisation de la tâche T_1 doit précéder celle de T_2. Il s'ensuit que deux tâches non comparables peuvent être réalisées en parallèle. L'algorithme est donc plus adapté au calcul parallèle si le nombre de tâches non comparables est important. Avec les technologies actuelles notamment la technologie VLSI (Very Large Scale Integration) où il est possible d'intégrer des milliers de milliers de transistors dans une même puce de Sillicium, la conception d'architectures et d'algorithmes parallèles est de plus en plus encourageante et motivante. Pour ainsi dire, le calcul parallèle est à l'ordre du jour. De ce fait, les algorithmes adaptés au parallélisme sont qualifiés de performants de par l'ultra-rapidité qu'ils peuvent assurer. Dans ce domaine voir par exemple[2], [14], [15]...

(d) *Le critère de la stabilité numérique.* Les critères(a)-(c) considérés jusque là s'appliquent à tous les algorithmes, numériques ou non numériques. Ce critère, par contre, s'intéresse exclusivement à l'aspect numérique des algorithmes. Dans ce cas, la performance d'un algorithme numérique est analysée à partir de la robustesse face à la propagation des erreurs d'arrondi provoquées par des réalisations approximatives des opérations élémentaires. En clair, on souhaite obtenir d'un tel algorithme des résultats approchés aussi précis que possible, auquel

cas l'algorithme est jugé efficace et performant. Il est à noter que le terme "stabilité numérique" est utilisé également dans le domaine de perturbation. Dans cette situation, un problème numérique (et non un algorithme) est dit numériquement stable si de petites perturbations sur les données entraînent des perturbations raisonnables sur les résultats. Pour simplifier on dira qu'un algorithme est numériquement stable s'il est, dans un sens qui sera précisé plus loin, robuste face à la propagation des erreurs d'arrondi. Pour plus de détails concernant ce critère, on renvoie le lecteur aux livres [26], [30], [46], [47], ...

Bien entendue, on souhaite la situation idéale dans laquelle l'algorithme conçu satisfait tous ces critères. Il peut se faire néanmoins (malheureusement c'est la règle) qu'un algorithme soit rapide sans qu'il soit numériquement stable ou bien numériquement stable sans être rapide etc... C'est le cas par exemple de l'algorithme d'inversion d'une matrice par partitionnement qui peut dans certains cas être performant du point de vue complexité mais hélas du point de vue numérique, est en règle générale fragile en face de l'accumulation des erreurs d'arrondi. Il est à préciser à cet égard qu'un compromis permettant de définir la performance à partir d'une combinaison de ces critères est difficile à admettre pour la simple raison qu'un algorithme conduisant à des solutions approximatives approchant d'une manière non réaliste les résultats exacts n'est pas à même d'être mis en pratique et n'a alors qu'un intérêt purement théorique. Une façon de pallier à cette remarque un peu pessimiste consiste à essayer de corriger les résultats approchés. Ces méthodes appelées raisonnablement méthodes de correction (ou d'amélioration) et qui constituent le centre d'intérêt de ce mémoire de Magister reposent sur l'idée de suivre les étapes de l'algorithme et de déterminer dans chaque étape l'erreur commise ou même sa valeur approchée. On obtient de cette façon une erreur approchée finale qui sera rajoutée à la solution approchée déjà obtenue. Cette approche, si elle est réalisable, donne souvent lieu à une amélioration. Pour plus de précisions concernant ces méthodes, on doit disposer d'outils permettant de déterminer les erreurs et d'analyser leur comportement; ceci impose la connaissance de l'arithmétique en virgule flottante avec précision finie.

Ce mémoire de Magister est donc logiquement composé de quatre chapitres. Les chapitres 2 et 3 sont consacrés à l'arithmétique des ordinateurs alors que dans les chapitres 4 et 5 on présente des méthodes de correction pour améliorer les résultats de certaines méthodes connues.

Au chapitre 2, on considére les systèmes de numération en les présentant comme un moyen adéquat de coder les nombres. Au passage nous avons jugé le célebre et très important codage de Hauffman comme une occasion heureuse de le rappeler pour appuyer l'intérêt d'utiliser le codage à longueur variable. Le thème qui nous intéresse le plus est bien sûr l'introduction de l'ensemble machine ou bien la structure machine $\mathbb{F}_\beta(p, e_{\min}, e_{\max})$ où chaque nombre réel non nul est approché par une représentation normalisée en virgule flottante avec précision finie p. Nous verrons que les opérations usuelles ne sont pas malheureusement stables dans cet ensemble si bien qu'on soit poussé à utiliser des approximations.

Au chapitre 3, nous menons une étude globale des arrondis et expliquons les raisons de considérer l'arrondi pair. Nous abordons ensuite les questions liées à l'analyse de l'effet des erreurs d'arrondi en mettant en évidence le modèle standard après avoir montré la façon par laquelle les opérations élémentaires sont réalisées [31]. Comme l'objet c'est de concevoir des méthodes de correction, on considère en particulier la réalisation des opérations avec récupération de l'erreur. Pour cela, on se base sur les travaux effectués par [16], [45].

Au troisième chapitre on considère le problème des récurrences linéaires du premier ordre. Comme la sommation d'un nombre fini de nombres n'est qu'un cas particulier des récurrences linéaires, nous avons présenté un formalisme clair expliquant comment sans recourir aux calculs en double précision, on arrive à améliorer la somme finale. Nous avons montré en particulier que sous des conditions raisonnables, on peut améliorer sensiblement la somme via une méthode de correction parallèle demandant un temps de calcul très réduit. En ce qui concerne les récurrences d'une manière générale, nous montrons que les résultats issus des méthodes parallèles ne peuvent pas être corrigés. Nous proposons alors de corriger la méthode séquentielle directe en utilisant une méthode parallèle.

Le quatrième chapitre est consacré à l'amélioration itérative appliquée aux systèmes linéaires. Nous rappelons la méthode en montrant qu'elle corrige en effet la solution et montrons informellement que d'une manière générale la méthode de Newton- Raphson peut être déduite d'une approche de correction. Nous abordons ensuite le sujet qui nous préoccupe le plus et qui concerne l'application de la technique de l'amélioration itérative aux matrices Toeplitz. Après avoir vue les approches éventuelles, nous proposons en conclusion la méthode d'amélioration

que nous recommandons le plus puisqu'elle assure une stabilité numérique acceptable en utilisant un temps de calcul réduit à condition que la méthode de départ ne soit pas abusivement instable.

Chapter 2

Représentation des nombres

2.1 Codage et systèmes de numération.

Le codage est naturellement pressenti dans toute expression formulée dans une langue étrangère, et par le décodage l'opération inverse qui consiste à déchiffrer et traduire l'expression dans une langue plus familière. De là, on déduit clairement que le codage est un sujet universel touchant pratiquement tous les domaines de la vie.

Sachant que les ordinateurs construits jusque là ou en voie de construction travaillent en binaire (c'est-à-dire basés sur l'alphabet $\{0,1\}$), tout ordinateur reconnait, comprend et interprète donc les objets qu'il manipule via des représentations binaires convenables au contexte dans lequel ils sont définis. En d'autres termes, de tels objets sont identifiés par les ordinateurs en lisant des mots de taille aussi réduite que possible et composés de 0 et 1. C'est ainsi que sont développés et commercialisés des codages efficaces comme par exemple le code de Gray, le code ASCII...

Concernant les nombres, il est logique de chercher les codages qui sont en adéquation avec les structures qui font des nombres réels un corps totalement ordonné.

Coder un objet signifie donc l'exprimer par une chaîne formée de 0 et 1. Soit $\mathcal{A}$ un alphabet (c'est-à-dire un ensemble fini) et soit $\mathcal{A}^*$ le monoide généré par $\mathcal{A}$: $x \in \mathcal{A}^* \Leftrightarrow$.

$$x = \# \; ou \; (\exists \; q \in \mathbb{N}^* \; et \; a_1, a_2, ..., a_q \in \mathcal{A} \; / \; x = a_1 a_2 ... a_q)$$

Pour rappel, le monoide $\mathcal{A}^*$ est une structure algébrique composée d'une part de l'ensemble noté encore $\mathcal{A}^*$ des mots formés à partir de l'alphabet $\mathcal{A}$ et de l'autre de la concaténation des mots comme opération de base. Le symbole conventionnel # désigne le mot vide qui joue le rôle de l'élément neutre par rapport à la concaténation (c'est-à-dire $\#w = w \; \forall w \in \mathcal{A}^*$). Ceci étant, considérons une fonction

$$c : \; \mathcal{A} \longrightarrow \{0,1\}^+ \tag{2.1}$$

de l'alphabet $\mathcal{A}$ à valeurs dans le monoide libre (privé du mot vide)

$$\{0,1\}^+ = \{0,1\}^* - \{\#\}$$

généré par $\{0,1\}$.

Définition 2.1. On dit que la fonction c considérée dans (2.1) est un codage de $\mathcal{A}$ si c est une injection de $\mathcal{A}$ sur $\{0,1\}^+$.

Cette définition est bien sûr établie dans le souci d'éviter le cas où deux éléments de $\mathcal{A}$ aient une même représentation. Auquel cas, il sera difficile voir impossible de décoder l'élément en question.

Associons, par ailleurs, à chaque caractère $a \in \mathcal{A}$, l'entier naturel $|c(a)| = q$ tel que:

$$c(a) = b_1 b_2 ... b_q \; / \; b_1, b_2, ..., b_q \in \{0,1\}$$

Dans le cas où $a \mapsto |c(a)|$ est constante, on dit que c est un codage de longueur fixe. Autrement, c'est un codage de longueur variable.

Dans notre contexte, il est naturel de construire des codages réalisant une certaine optimalité notamment en matière d'économie dans le stockage en mémoire. A titre d'exemple, on aimerait avoir un codage c^* vérifiant:

$$\max_{a\in\mathcal{A}} |c^*(a)| \leq \max_{a\in\mathcal{A}} |c(a)| \quad (\forall\ c \text{ codage de } \mathcal{A}) \tag{2.2}$$

On peut énoncer dans ce cas le résultat classique suivant:

Théorème 2.1.- Soit N le nombre d'éléments de $\mathcal{A}$ et supposons que $N = 2^q$. Alors il existe un codage c^* de longueur fixe (de longueur q) réalisant (2.2). □

D'après ce théorème, il faut au moins des chaînes de $\lceil \log_2 N \rceil$ bits pour coder N objets. Malgré cette motivation incitant à considérer les codages de longueur fixe, il convient quand même de noter que les codages de longueur variable sont parfois beaucoup plus utiles notamment dans les situations où l'apparition de certains objets est plus fréquente que d'autres. Pour s'en convaincre, prenons un exemple emprunté à Cormen et al [14, page 331].

Exemple 2.1: Supposons qu'un fichier ne contient que les lettres A, B, C, D, E, F avec les fréquences d'apparition suivantes:

A	B	C	D	E	F
45.10^3	13.10^3	12.10^3	16.10^3	9.10^3	5.10^3

Le fichier contient donc 100 000 caractères qu'il importe de les stocker économiquement en mémoire. Analysons les deux codages suivants:

X	A	B	C	D	E	F
$c_1(X)$	000	001	010	011	100	101
$c_2(X)$	0	101	100	1101	000	1100

On observe qu'avec le codage de longueur fixe c_1, on a besoin de 300 000 cases mémoire binaires pour mémoriser le fichier alors qu'avec le codage de longueur variable c_2, on a besoin pour stoker en mémoire le dit fichier de

$$(45.1 + 13.3 + 12.3 + 16.3 + 9.4 + 5.4).1000 = 224\ 000$$

cases mémoire binaires, soit 76 000 cases mémoire de moins.

Supposons que c donnée par (2.1) soit un codage de longueur fixe et identifions toute chaîne $b_q b_{q-1}...b_1 \in \{0,1\}^q$ à $b_r b_{r-1}...b_1$ où $1 \leq r \leq q$, $b_r = 1$ et $b_q = b_{q-1} = \ ... = b_{r+1} = 0$ et 00...0 à 0. On obtient ainsi un codage de longueur variable de $\mathcal{A}$ que, pour le représenter, il suffit de

$$1 + \sum_{k=1}^{q} k2^{k-1} = N \log_2 N - (N-2)$$

cases mémoires binaires, au lieu de $N \log_2 N$, requises dans le cas du codage de longueur fixe.

Le système de numération binaire, décimal ou dans n'importe quelle base $\beta \geq 2$ n'est qu'une forme de codage adapté aux nombres. Nous savons que dans une base $\beta \geq 2$, tout entier naturel N s'écrit d'une façon unique comme suit:

$$N = d_{q-1}\beta^{q-1} + d_{q-2}\beta^{q-2} + \ ... + d_0, \ d_j \in \{0,1,...,\beta-1\} \ (\forall j = 0 : (q-1)) \tag{2.3}$$

on écrit alors $N = (d_{q-1}d_{q-2}...d_0)_\beta$ et on dit que N est représenté dans la base β. Le système binaire correspond à $\beta = 2$, le système décimal correspond à $\beta = 10$, le système octal correspond à $\beta = 8$, le système héxadécimal correspond à $\beta = 16$ etc...On observe que si dans (2.2), $d_{q-1} \neq 0$, on a

$$\beta^{q-1} \leq N \leq \beta^q - 1$$

de sorte que la taille de N soit comprise entre $\log_\beta(N+1)$ et $1 + \log_\beta N$.

Quelle que soit la base adoptée, on est tenu en pratique à allouer autant de cases mémoire que la technologie et sa gestion le permettent pour représenter les nombres. Comme l'ensemble des nombres est infini, on comprend de suite que l'ordinateur aussi puissant soit il ne peut reconnaître certains nombres qu'approximativement. Pire encore, il peut dans certains cas afficher des messages de dépassement (overflow/underflow). Cet aspect sera abordé à la cinquième section.

Il importe de remarquer ici le caractère arbitraire de la base β dans notre introduction aux systèmes de numération. On s'attendrait peut-être au cas; visiblement logique; $\beta = 2$ préféré au

demeurant par le système IEEE 754 (et même 854) [31]. Cependant, il est intéressant d'étudier le problème dans sa généralité [9]. De plus, certains systèmes utilisent le système décimal comme le IEEE standard 854 ou le système héxadécimal comme c'est le cas dans l'IBM370. A cet égard, une question mérite d'être tirée au clair. Comment les nombres sont alors représentés alors que les machines travaillent en binaire? Pour répondre à cette question, supposons en guise de simplicité que $\beta = 16$. Les β-chiffres $0, 1, 2, ..., 15$ sont alors codés en utilisant de préférance le système binaire. Ensuite il suffit de considérer les mots composés de "sous-mots" de taille 4 si le codage est de longuer fixe. Par exemple le nombre $1329 = (531)_{16}$ peut être représenté en machine comme suit

$$(0101)(0011)(0001)$$

Dans le système binaire ce même nombre s'écrit

$$(10011001101)_2$$

Bien que cet exemple puisse être matière à comparer entre les différentes bases de numération, notre propos ici consiste, néanmoins, juste à montrer la possibilité d'utiliser des bases autres que le système binaire.

Il est raisonnable de signaler pour clore cette section qu'il existe des codages signés des nombres. Il s'agit d'une approche comme tant d'autres qui mérite d'être tenue en compte mais qui sont encore au stade d'exploration. C'est pour cette raison que nous ne l'avons pas considéré ici. Voir pour cela à titre d'exemple [38].

2.2 Représentation en virgule flottante des nombres réels.

Il est connu que la représentation (2.3) peut être étendue aux nombres réels. Dans ce contexte, s'il est vrai que la représentation en virgule fixe:

$$\begin{aligned} x &= \pm(n_1 n_2 ... n_p \,.\, d_1 d_2 ...)_\beta \qquad (2.4) \\ &= \sum_{j=1}^{p} n_j \beta^j + \sum_{k\geq 1} d_k \beta \qquad (n_j, d_k \in \{0, 1, ..., \beta - 1\}) \end{aligned}$$

soit, surtout dans le cas décimal $\beta = 10$, la plus utilisée dans la vie courante, il n'en demeure pas moins que, pour des raisons qui vont être apparentes, on préfère dans le domaine de l'arithmétique des ordinateurs, la représentation en virgule flottante justifiée par le fait que tout nombre non nul $x \neq 0$ peut être décomposé d'une façon unique comme suit:

$$\begin{aligned} x &= \pm(m_0.m_1 m_2 ...)_\beta \beta^e \qquad (2.5) \\ (\forall j &\geq 0) m_j \in \{0, 1, ..., \beta - 1\},\ m_0 \neq 0, e \in \mathbb{Z} \end{aligned}$$

Cette écriture est justement ce qu'on appelle la représentation (normalisée) en virgule flottante de x. Le nombre

$$m(x) = (m_0.m_1 m_2 ...)_\beta = \sum_{j\geq 0} m_j \beta^{-j} \qquad (2.6)$$

s'appelle la mantisse de x et $e(x) = e$ s'appelle l'exposant de x. On observe que $1 \leq m(x) < \beta$ si bien qu'il importe de décrire les nombres réels > 0 par l'intervalle semi ouvert $[1, \beta[$ et ses décalages $\beta^e[1, \beta[,\ e \in \mathbb{Z}$. Ainsi dans cette représentation, tout nombre réel x est identifié par son signe $sgn = sgn(x) \in \{-, 0, +\}$, un nombre $m = m(x) \in [1, \beta[$ et par un entier relatif $e = e(x)$. Dans ce cas, ou $x = 0$ (qui a lieu si $sgn = 0$) ou bien $x = (sgn)m\beta^e$. Mathématiquement, on peut dire qu'on a une bijection; bien mieux un isomorphisme dont la structure sera naturellement comprise à partir du contexte ; de $\mathbb{R}^*$ sur le produit cartésien $\{-, +\} \times [1, \beta[\times \mathbb{Z}$.

En tout état de cause, une hypothèse que nous formulons dans ce qui suit est exigée

Hypothèses 2.1. (i) Dans la représentation en virgule fixe (2.4), on suppose que:

$$(\forall i \geq 1)\ (\exists j \geq i)\ d_j \neq \beta - 1$$

(ii) Dans la représentation en virgule flottante (2.5), on suppose que:

$$(\forall i \geq 1)\ (\exists j \geq i)\ m_j \neq \beta - 1 \qquad \text{(H)}$$

On peut alors énoncer le résultat suivant justifiant l'écriture (2.5)

Théorème 2.2. Sous l'hypothèse (H) ainsi formulée, la représentation (2.5) existe et est unique.

Démonstration. Rappelons d'abord que $\lfloor y \rfloor$ désigne la partie entière de y, c'est-à-dire le plus grand entier $\leq y$. Ensuite, formons les récurrences suivantes à partir d'un $m \in [1, \beta[$

$$\begin{aligned} y_0 &= m, \ \ b_0 = \lfloor y_0 \rfloor \\ y_k &= (y_{k-1} - b_{k-1}) * \beta, \ \ k \geq 1 \\ b_k &= \lfloor y_k \rfloor \end{aligned} \qquad (2.7)$$

On vérifie alors que pour tout $e \in \mathbb{Z}$,

$$\begin{aligned} x &= \pm m\beta^e \\ &= \pm(b_0.b_1b_2...)_\beta \beta^e \end{aligned}$$

est la représentation normalisée en virgule flottante de x puisque $b_0 = \lfloor y_0 \rfloor \geq 1$.Inversement, si l'on se donne $x > 0$ (quitte à remplacer x par $-x$) défini par (2.5), on s'aperçoit qu'il existe un et un seul $e \in \mathbb{Z}$ tel que $\frac{x}{\beta^e} \in [1, \beta[$. En effet, l'unicité découle du fait que

$$\beta^{-1}(\frac{x}{\beta^e}) < 1 \text{ et } \beta(\frac{x}{\beta^e}) \geq \beta$$

et l'existence découle du fait que

$$(\frac{x}{\beta^k} - \frac{x}{\beta^{k+1}}) = \frac{\beta - 1}{\beta}(\frac{x}{\beta^k}) \to 0$$

quand $k \to \infty$ de manière à ce qu'il existe k_0 tel que

$$k \geq k_0 \Rightarrow (\frac{x}{\beta^k} - \frac{x}{\beta^{k+1}}) \leq \frac{\beta - 1}{2}$$

Par ailleurs, si l'on applique les récurrences (2.7) à $m = \frac{x}{\beta^e}$, on montre par une récurrence aisée que

$$(\forall k \in \mathbb{N})\ b_k = m_k$$

où les b_k sont déterminés par les récurrences (2.7) et les m_k sont donnés par (2.5). On vient ainsi de démontrer d'une manière constructive l'unicité de la représentation en virgule flottante (2.5).□

Il importe de noter dans cette décomposition en virgule flottante le caractère infini de la mantisse et celui de l'ensemble des entiers $\mathbb{Z}$. Voulant représenter les nombres en machine, cet aspect ne facilite, en aucun cas, la tâche.

Bien qu'en général, les nombres ne soient pas écrits suivant le format (2.5), il n'empêche qu'il n'est pas difficile de le voir directement à partir de l'écriture en virgule fixe (2.4). Exposons à cet effet la définition suivante.

Définition 2.2. Soit

$$x = \sum_{j=-\infty}^{p} a_j \beta^j, \quad (a_j \in \{0, 1, ..., \beta - 1\}\ \forall j \leq p,\ a_p \neq 0)$$

un nombre et soit $j \leq p$. On dit que a_j est un chiffre significatif s'il existe $i \leq j$ tel que $a_i \neq 0$.

Il en découle directement que si q désigne le plus petit indice q pour lequel $a_q \neq 0$, on a

$$x = (a_p.a_{p-1}...a_q)\beta^p$$

Par exemple la relation $0.000900107 = 9.00107 \times 10^{-4}$ peut être vue comme découlée des six chiffres significatifs $9, 0, 0, 1, 0, 7$.

Grâce à cette notion de chiffres significatifs (voir [10]), d'autres définitions équivalentes de la représentation normalisée en virgule flottante peuvent être formulées. La représentation la plus utilisée dans la littérature, en plus du format (2.5), consiste à écrire le nombre $x \neq 0$ comme suit:

$$\begin{aligned} x &= \pm(0.m_0 m_1 m_2 ...)_\beta \beta^{e'} \\ (\forall j &\geq 0) m_j \in \{0, 1, ..., \beta - 1\},\ m_0 \neq 0, e' \in \mathbb{Z} \end{aligned}$$

Dans cette section, nous n'avons considéré que la représentation normalisée en virgule flottante. La représentation non normalisée est similaire et peut être formulée en écrivant $x \in \mathbb{R}$ comme suit:

$$\begin{aligned} x &= \pm(m_0.m_1 m_2 ...)_\beta \beta^{e} \\ (\forall j &\geq 0) m_j \in \{0, 1, ..., \beta - 1\}, e \in \mathbb{Z} \end{aligned}$$

On remarque que la condition $m_0 \neq 0$ est omise. On verra après des situations où on est contraint à recourir aux représentations non normalisées.

2.3 Représentation des nombres avec précision finie.

On a précisé au début que, la capacité mémoire d'un ordinateur étant limitée, la machine ne peut reconnaître, hélas, qu'une partie finie de $\mathbb{R}$. On est donc contraint à se restreindre à considérer une partie finie, aussi représentative que possible, des nombres réels. A présent, il est communément admis que la représentation en virgule flottante exposée à la section précédente

délivre la partie finie la plus convaincante. Une telle partie peut être notée par $\mathbb{F}_\beta(p, e_{\min}, e_{\max})$ et est définie comme suit: $x \in \mathbb{F}_\beta(p, e_{\min}, e_{\max}) \Leftrightarrow$

$$\begin{aligned} x &= 0 \; ou \; x = \pm(m_0.m_1m_2...m_{p-1})_\beta \times \beta^e \\ \text{t.q. } (\forall j &= 0 : p-1) \; m_j \in \{0, 1, ..., \beta - 1\}, m_0 \neq 0, \; e_{\min} \leq e \leq e_{\max} \end{aligned} \tag{2.8}$$

Il s'agit bien de l'écriture en virgule flottante similaire à celle présentée dans (2.5). La différence réside dans le nombre fini p; appelé précision; de chiffres qui forment la mantisse et dans la restriction à $[e_{\min}, e_{\max}]$ au lieu de $\mathbb{Z}$.

Définitition 2.3. L'entier $p \geq 1$ ainsi introduit s'appelle la précision.

Pour mieux cerner la définition de cet ensemble, il convient de le voir comme étant l'ensemble des nombres réels ayant au plus p chiffres significatifs et dont l'exposant est compris entre $e_{\min}$ et $e_{\max}$. Sachant que le nombre des mantisses possible est égal à $(\beta - 1)\beta^{p-1}$, on en déduit que

$$card(\mathbb{F}_\beta(p, e_{\min}, e_{\max})) = 1 + 2(\beta - 1)\beta^{p-1}(e_{\max} - e_{\min} + 1)$$

D'après la section 2.1, il faut alors à peu près $1 + \log \beta^p + \log(e_{\max} - e_{\min} + 1)$ bits pour représenter en machine l'ensemble $\mathbb{F}_\beta(p, e_{\min}, e_{\max})$.

Il est tout à fait clair que nous avons les inclusions suivantes:

$$\mathbb{F}_\beta(1, e_{\min}, e_{\max}) \subset \mathbb{F}_\beta(2, e_{\min}, e_{\max}) \subset \mathbb{F}_\beta(3, e_{\min}, e_{\max}) \subset \; ...$$

de sorte qu'on puisse parler de précisions différentes. En pratique, on se donne une précision de travail disons p; dans ce cas p est appelé précision simple; et il peut se faire qu'on ait besoin de travailler en précision double $2p$ ou en une précision étendue $p' > p$. Une telle approche est recommandée (en général comme dernier recours) afin d'améliorer la précision d'un résultat.

Exemple 2.2. Les éléments de $\mathbb{F}_2(1, -3, 6_{\max})$ sont

$$0, \pm 0.125, \pm 0.25, \pm 0.5, \pm 1, \pm 2, \pm 4, \pm 8, \pm 16, \pm 32, \pm 64$$

et les éléments de $\mathbb{F}_2(3, e_{\min}, e_{\max}) \cap [1, 2[$ sont

$$1, 1.25,\ 1.5,\ 1.75$$

Supposons qu'une machine sur laquelle nous travaillons utilise un système en virgule flottante de base β. L'algorithme suivant réalise la décomposition en virgule flottante (2.7), c'est-à-dire qu'étant donné un nombre nul x, il délivre sa mantisse $m(x)$ et son exposant $e(x)$.

Algorithme 2.1.- décomp-flot$(x)= (m(x), e(x))$.

Dans cet algorithme, x est donné. Il peut calculer directement son signe. C'est-à-dire s'il est positif, négatif ou nul. On suppose donc sans perdre en généralité que x est un nombre strictement positif quitte à remplacer x par $-x$. Ceci étant, l'algorithme procéde à la détermination de la mantisse $m(x)$ et l'exposant $e(x)$.

1. $m = x$
2. $e = 0$
3. Tant que $((m < 1)$ ou $(m \geq \beta))$
4. Si $(m < 1)$
5. $m = \beta * m$
6. $e = e - 1$
7. Sinon
8. $m = m/\beta$
9. $e = e + 1$
10. $m(x) = m$ et $e(x) = e$.

L'importance de cet algorithme réside dans le fait que la machine est conçue de manière à faciliter la tâche à l'utilisateur. Ainsi, elle reçoit et délivre les nombres en les écrivant dans le système décimal alors que dans bien des cas, elle travaille dans le système binaire. A l'aide de cet algorithme, l'utilisateur peut calculer, si besoin, à la fois la mantisse et l'exposant du nombre traité.

Il est à noter que cet algorithme révèle une caractérisation importante de l'exposant d'un nombre non nul qu'on peut exposer dans le résultat suivant

Théorème 2.3. Soit $x > 0$. Alors

$$e(x) = e \Leftrightarrow \beta^e \leq x < \beta^{e+1} \Leftrightarrow e = \lfloor \log_\beta x \rfloor$$

Démonstration. Ecrivons $x = m(x)\beta^{e(x)}$. Comme $1 \leq m(x) < \beta$, il vient

$$e(x) = e \Longrightarrow \beta^e \leq x < \beta^{e+1}$$

Réciproquement, si $\beta^e \leq x < \beta^{e+1}$, on écrit $x = m\beta^e$ si bien que $1 \leq m = \frac{x}{\beta^e} < \beta$. L'unicité de la représentation normalisée en virgule flottante donne $m = m(x)$ et $e(x) = e$. Des inégalités

$$\beta^{e(x)} \leq x < \beta^{e(x)+1}$$

on déduit que $e(x) \leq \log_\beta x < e(x) + 1$ de sorte que $e(x) = \lfloor \log_\beta x \rfloor$. □

Dans un même ordre d'idées, il importe de déterminer la précision p adoptée dans le cas où elle n'est pas mentionnée dans les brochures accompagnant la machine. Pour comprendre l'algorithme qui suit et qui détermine la précision, considérons d'abord l'exemple suivant

Exemple 2.3. Supposons que $\beta = 2$ et étudions le nombre $\frac{4}{3}$. On a dans le système binaire

$$\frac{4}{3} = 1.010101010...$$

On saura après que si la précision $p = 5$

$$\frac{4}{3} \simeq 1.0101$$

et que si $p = 6$

$$\frac{4}{3} \simeq 1.01011$$

Pour généraliser cet exemple, remarquons d'abord que

$$\frac{\beta^2}{\beta^2-1} = (1.010101...)_\beta.$$

On est alors en mesure de décrire l'algorithme permettant de déterminer la précision.

Algorithme 2.2. Précision.

Cet algorithme détermine la précision p adoptée par la machine en supposant qu'elle travaille dans un système de base β.

1. $m = \frac{\beta^2}{\beta^2-1}$
2. $b = 1$
3. $q = 1$
4. $ms = (m - b) * \beta$
5. $bs = \lfloor ms \rfloor$
6. Tant que $(b \neq bs)$
7. $\quad m = ms$
8. $\quad b = bs$
9. $\quad ms = (m - b) * \beta$
10. $\quad bs = \lfloor ms \rfloor$
11. $\quad q = q + 1$
12. Si $(b = bs = 0)$ $p = q - 1$ sinon $p = q + 1$

Cet algorithme applique en fait les récurrences (2.7) à la valeur appartenant à

$$\mathbb{F}_\beta(p, e_{\min}, e_{\max})$$

qui approche le mieux $\frac{\beta^2}{\beta^2-1}$ et repose sur l'idée de l'exemple.

Le problème principal dans l'ensemble $\mathbb{F}_\beta(p, e_{\min}, e_{\max})$, comme d'ailleurs c'est le cas de tout ensemble fini de nombres, réside dans la non stabilité des opérations de base. Plus clairement, dans la plupart des cas

$$x \; op \; y \quad \notin \quad \mathbb{F}_\beta(p, e_{\min}, e_{\max})$$
$$op \quad \in \quad \{+, -, *, /, ...\}$$

bien que $x, y \in \mathbb{F}_\beta(p, e_{\min}, e_{\max})$. Nous verrons après comment faire face à cette difficulté en recourant à des opérations élémentaires approximatives. Pour l'instant on se contente d'étudier l'opération de dualité $x \mapsto \beta^{e(x)+1} - x$; mais auparavant énonçons le résultat suivant qui peut être vu comme une caractérisation de $\mathbb{F}_\beta(p, e_{\min}, e_{\max})$.

Théorème 2.4. Soit $x \in \mathbb{R}$. Alors

$$x \in \mathbb{F}_\beta(p, e_{\min}, e_{\max}) \Longleftrightarrow x\beta^{p-e(x)-1} \in \mathbb{Z}$$

Démonstration. Supposons que $x \neq 0$ et

$$\begin{aligned} x &= \pm(m_0.m_1m_2...)_\beta \beta^{e(x)} \\ (\forall j &\geq 0) m_j \in \{0, 1, ..., \beta - 1\}, \; m_0 \neq 0, e \in [e_{\min}, e_{\max}] \end{aligned}$$

donc

$$x\beta^{p-e(x)-1} = \pm(m_0m_1m_2...m_{p-1}.m_pm_{p+1}...)$$

par suite l'equivalence s'ensuit aussitôt. □

Dans le résultat suivant, nous montrons que l'opération $x \longmapsto \beta^{e(x)+1} - x$ assure une certaine invariance

Théorème 2.5. Soientt $x \in \mathbb{F}_\beta(p, e_{\min}, e_{\max})$ un nombre "flottant" positif et

$$k \in \{0, 1, ..., p-1\}$$

tel que

$$m_k \neq \beta - 1, m_j = \beta - 1, j = 0 : (k-1)$$

Alors

$$\beta^{e(x)+1} - x \in \mathbb{F}_\beta(p-k, e_{\min}, e_{\max}),\ e(\beta^{e(x)+1} - x) = e(x) - k$$

pourvu que $e(x) - k \geq e_{\min}$

Démonstration. Supposons sans perdre en généralité que $e(x) = 0$ et écrivons

$$\begin{aligned} x &= (m_0.m_1 m_2 ... m_{p-1})_\beta \\ (\forall j &\geq 0) m_j \in \{0, 1, ..., \beta - 1\},\ m_0 \neq 0, m_{p-1} \neq 0 \end{aligned}$$

Posons alors

$$\bar{m}_{p-1} = \beta - m_{p-1},\ \bar{m}_i = \beta - 1 - m_i$$

de sorte que

$$\beta - x = (\bar{m}_0.\bar{m}_1 \bar{m}_2 ... \bar{m}_{p-1})_\beta$$

Par conséquent $\beta - x \in \mathbb{F}_\beta(p, e_{\min}, e_{\max})$. De plus si $m_j = \beta - 1, 0 \leq j < k$ et $m_k \neq \beta - 1$ on a

$$\beta - x = (0.00...0\bar{m}_k ... \bar{m}_{p-1})_\beta$$

et forcément

$$\beta - x \in \mathbb{F}_\beta(p-k, e_{\min}, e_{\max}), e(\beta - x) = -k$$

Le théorème est alors démontré.□

Dans ce théorème, on a exclu les nombres dont la mantisse est $m = \beta - \beta^{1-p}$. Dans ce cas, on observe que $\beta - m = \beta^{1-p}$.

Chapter 3

Arrondis et Analyse de l'Erreur

3.1 Les différents types d'arrondi.

Rappelons que les ensembles

$$\mathbb{F}_\beta(p, e_{\min}, e_{\max})$$

étudiés dans la section précédente sont introduits bien entedu dans le but d'approximer au mieux l'ensemble des nombres réels. Dans cette perspective, il est visiblement logique d'approximer tout nombre réel x par un autre nombre réel $\hat{x}$ représentable en machine et le plus proche possible de x. En clair, on cherche $\hat{x} \in \mathbb{F}_\beta(p, e_{\min}, e_{\max})$ tel que

$$|x - \hat{x}| \leq |x - y| \quad (\forall y \in \mathbb{F}_\beta(p, e_{\min}, e_{\max})) \tag{3.1}$$

Pour construire cet $\hat{x}$, on procède tout simplement à une troncature appropriée de la mantisse $m(x)$ définie dans (2.5) et (2.6) en ne retenant bien sûr que p chiffres significatifs. Dans le but de comprendre les avantages de l'arrondi adopté par la communauté des numériciens et informaticiens, nous avons jugé utile d'étudier dans la globalité les arrondis possibles, y compris ceux qui ne remplissent pas à fortiorie la condition de minimalité (3.1). On remarquera après que la notion d'arrondi est étroitement liée à celle de la partie entière d'un nombre. Dans ce cadre, nous aurons besoin du résultat élémentaire suivant:

Lemme 3.1. Soit $x > 0$. Alors $x \in [\lfloor x \rfloor, \lfloor x \rfloor + 1[$ et $]\lfloor x \rfloor, \lfloor x \rfloor + 1[\cap \mathbb{N} = \varnothing$.

Démonstration. Par définition, $\lfloor x \rfloor$ désigne le plus grand entier $\leq x$. Il est donc logique que le successeur $\lfloor x \rfloor + 1$ de $\lfloor x \rfloor$ soit $> x$. D'autre part, il est évident que tout intervalle ouvert dont les bornes sont deux entiers successifs ne contient pas des entiers.□

Arrondi par défaut.

On note cet arrondi par rd_p . Soit

$$\begin{aligned} x &= \pm(m_0.m_1m_2...)_\beta\beta^e \\ (\forall j &\geq 0) m_j \in \{0, 1, ..., \beta - 1\},\ m_0 \neq 0, e \in [e_{\min}, e_{\max}] \end{aligned}$$

un nombre réel non nul. Alors on pose

$$rd_p(x) = \pm(m_0.m_1m_2...m_{p-1})_\beta\beta^e$$

Bien sûr, $rd_p(x)$ et x sont du même signe. On en déduit directement que

$$\begin{aligned} |x - rd_p(x)| &= (\sum_{j=p}^{\infty} m_j\beta^{-j})\beta^e \\ &< (\beta - 1)\beta^{-p}\frac{1}{1 - \beta^{-1}}\beta^e = \beta.\beta^{-p}\beta^e \end{aligned} \tag{3.2}$$

L'inégalité stricte provient de l'hypothèse (H) formulée dans la deuxième section.

Exemple 3.1. $(p = 4, \beta = 10)$

$$rd_4(0.023569085) = 0.02356$$

C'est-à-dire on tronque en ne retenant que 4 chiffres significatifs.

Cet arrondi est simple à réaliser puisqu'il suffit de retenir les premiers p chiffres de la mantisse d'un nombre pour dire qu'on a réalisé un arrondi par défaut. Dans certains ouvrages,

Lemme 3.1. Soit $x > 0$. Alors $x \in [\lfloor x \rfloor, \lfloor x \rfloor + 1[$ et $]\lfloor x \rfloor, \lfloor x \rfloor + 1[\cap \mathbb{N} = \varnothing$.

Démonstration. Par définition, $\lfloor x \rfloor$ désigne le plus grand entier $\leq x$. Il est donc logique que le successeur $\lfloor x \rfloor + 1$ de $\lfloor x \rfloor$ soit $> x$. D'autre part, il est évident que tout intervalle ouvert dont les bornes sont deux entiers successifs ne contient pas des entiers.□

Arrondi par défaut.

On note cet arrondi par rd_p . Soit

$$\begin{aligned} x &= \pm(m_0.m_1m_2...)_\beta\beta^e \\ (\forall j \geq 0) m_j &\in \{0,1,...,\beta-1\},\ m_0 \neq 0, e \in [e_{\min}, e_{\max}] \end{aligned}$$

un nombre réel non nul. Alors on pose

$$rd_p(x) = \pm(m_0.m_1m_2...m_{p-1})_\beta\beta^e$$

Bien sûr, $rd_p(x)$ et x sont du même signe. On en déduit directement que

$$\begin{aligned} |x - rd_p(x)| &= (\sum_{j=p}^{\infty} m_j\beta^{-j})\beta^e \\ &< (\beta-1)\beta^{-p}\frac{1}{1-\beta^{-1}}\beta^e = \beta.\beta^{-p}\beta^e \end{aligned} \tag{3.2}$$

L'inégalité stricte provient de l'hypothèse (H) formulée dans la deuxième section.

Exemple 3.1. $(p = 4, \beta = 10)$

$$rd_4(0.023569085) = 0.02356$$

C'est-à-dire on tronque en ne retenant que 4 chiffres significatifs.

Cet arrondi est simple à réaliser puisqu'il suffit de retenir les premiers p chiffres de la mantisse d'un nombre pour dire qu'on a réalisé un arrondi par défaut. Dans certains ouvrages,

on définit l'arrondi à partir du résultat suivant

Lemme 3.2.- Soit $x \neq 0$. Alors $rd_p(x) = \pm(\lfloor \beta^{p-1} m(x) \rfloor \beta^{1-p})\beta^{e(x)}$.

Démonstration. Supposons que

$$\begin{aligned} x &= \pm(m_0.m_1m_2...)_\beta \beta^e \\ (\forall j &\geq 0) m_j \in \{0,1,...,\beta-1\},\ m_0 \neq 0, e \in [e_{\min}, e_{\max}] \end{aligned}$$

On observe que

$$\beta^{p-1}(m_0.m_1m_2...)_\beta = (m_0m_1...m_{p-1}.m_pm_{p+1}...)_\beta$$

donc

$$\lfloor \beta^{p-1}(m_0.m_1m_2...)_\beta \rfloor \beta^{1-p} = (m_0.m_1...m_{p-1})_\beta$$

Le lemme s'ensuit aussitôt.□

Cet arrondi, important de par sa simplicité, ne vérifie pas malheureusement la condition de minimalité (3.1). Pour pallier à cet inconvénient, on va établir une condition "relaxée"

Théorème 3.3. On a:

$$|x - rd_p(x)| = \min_{y \in \mathbb{F}_\beta(p,e_{\min},e_{\max}) \& |y| \leq |x|} |x - y| \tag{3.3}$$

Démonstration. Supposons sans perdre en généralité que $x > 0$ et écrivons $x = m(x)\beta^{e(x)}$. Par ailleurs, soit y un nombre tel que

$$y \in \mathbb{F}_\beta(p, e_{\min}, e_{\max}), y \leq x$$

Notre but consiste à montrer que $y \leq rd_p(x)$. Supposons par absurde que $y > rd_p(x)$. On en déduit que

$$\beta^{e(x)} \leq y < \beta^{e(x)+1} \Longrightarrow e(y) = e(x)$$

Par conséquent $m(y) \leq m(x)$.D'après le théorème 2.4, on a $\beta^{p-1}m(y) \in \mathbb{N}$ de sorte que

$$\beta^{p-1}m(y) \leq \lfloor \beta^{p-1}m(x) \rfloor$$

en vertu du lemme 3.1. Enfin, on obtient

$$y = m(y)\beta^{e(y)} \leq \lfloor \beta^{p-1}m(x) \rfloor \beta^{1-p}\beta^{e(x)} = rd_p(x)$$

On vient de montrer que $[rd_p(x), x] \cap \mathbb{F}_\beta(p, e_{\min}, e_{\max}) = \{rd_p(x)\}$; ce qui prouve le théorème.□

Il importe d'apprécier davantage la majoration stricte (3.2). Pour cela, construisons un x donné par (2.5) en supposant qu'il est positif, $e(x) = 0$ et que

$$(\forall j = p : (k + p - 1)) \quad m_j = \beta - 1$$

Alors cet x admet la décomposition suivante

$$x = rd_p(x) + \beta\beta^{-p}(1 - \beta^{-k}) + R_k, \quad R_k < \beta\beta^{-p}\beta^{-k}$$

si bien que

$$x - rd_p(x) = \beta\beta^{-p} - (\beta^{-\kappa} - \frac{\cdots\kappa}{\beta\beta^{-p}})\beta\beta^{-p}$$

et que le second terme tende vers $\beta\beta^{-p}$ quand k tend vers $+\infty$. Par conséquent, on a:

$$\sup \left\{ \frac{|x - rd_p(x)|}{\beta^{e(x)}} / x \in \mathbb{R} \& e(x) \in [e_{\min}, e_{\max}] \right\} = \beta\beta^{-p}$$

Arrondi par excès.

Cet arrondi est noté par rd_p^+ et est défini comme suit. Soit

$$\begin{aligned} x &= \pm(m_0.m_1m_2...)_\beta\beta^e \\ (\forall j \geq 0) & m_j \in \{0,1,...,\beta-1\},\ m_0 \neq 0, e \in [e_{\min}, e_{\max}] \end{aligned}$$

Alors, on pose

$$rd_p^+(x) = \pm(m_0.m_1m_2...m_{p-1} + \beta^{1-p})_\beta\beta^e$$

Là aussi, x et $rd_p^+(x)$ sont du même signe. On observe que si $x > 0$,

$$rd_p^+(x) = rd_p(x) + \beta\beta^{-p}\beta^e \tag{3.4}$$

de sorte que

$$\begin{aligned} rd_p^+(x) - x &= \left[\beta\beta^{-p} - \frac{x - rd_p(x)}{\beta^e}\right]\beta^e \\ &\leq \beta\beta^{-p}\beta^e \end{aligned}$$

et l'égalité ait lieu si $x = rd_p(x)$. Si $x < 0$, on obtient

$$rd_p^+(x) = rd_p(x) - \beta\beta^{-p}\beta^e$$

par suite

$$\begin{aligned} x - rd_p^+(x) &= \left[\beta\beta^{-p} - \frac{rd_p(x) - x}{\beta^e}\right]\beta^e \\ &\leq \beta\beta^{-p}\beta^e \end{aligned}$$

et enfin

$$\left|rd_p^+(x) - x\right| \leq \beta\beta^{-p}\beta^e$$

et que l'égalité a lieu si $x = rd_p(x)$.

Contrairement à l'arrondi par défaut qui se réalise par une simple troncature, l'arrondi par excès exige pour sa réalisation une addition; ceci requiert un temps de calcul relativement non négligeable.

Exemple 3.2. $p = 6, \beta = 10$. On a: $0.000498999108 = 4.98999108 \times 10^{-4}$

$$\begin{aligned} rd_6^+(0.000498999108) &= (4.98999 + 10^{-5})10^{-4} \\ &= 0.000499 \end{aligned}$$

Malgré ce caractère relativement coûteux de cet arrondi, il est à signaler qu'il est similaire à l'arrondi par défaut comme l'explique le résultat suivant

Lemme 3.4. Soit $x \neq 0$. Alors $rd_p^+(x) = \pm(\lfloor \beta^{p-1} m(x) \rfloor + 1)\beta^{1-p}\beta^{e(x)}$.

Démonstration. Ce lemme résulte directement du lemme 2.7 et de (3.4).□

Enonçons le résultat suivant qui est similaire au théorème 3.3 et vérifie une propriété de minimalité importante.

Théorème 3.5. On a pour tout $x > 0$ tel que $x > rd(x)$

$$|x - rd_p^+(x)| = \min_{y \in \mathbb{F}_\beta(p, e_{\min}, e_{\max}) \& |y| \geq |x|} |x - y| \tag{3.5}$$

*Démonstration.*Supposons sans perdre en généralité que $x > 0$ et écrivons $x = m(x)\beta^{e(x)}$. Par ailleurs, soit y un nombre tel que

$$y \in \mathbb{F}_\beta(p, e_{\min}, e_{\max}), y > x$$

Notre but consiste à montrer que $y \geq rd_p^+(x)$. Supposons par absurde que $y < rd_p^+(x)$. On en déduit que

$$\beta^{e(x)} < y < \beta^{e(x)+1} \Longrightarrow e(y) = e(x)$$

en vertu du théorème 2.3 et du fait que $rd_p^+(x) \leq \beta^{e(x)+1}$. Par conséquent $m(y) > m(x)$ et; puisque $\beta^{p-1}m(y) \in \mathbb{N}$ d'après le théorème 2.4;

$$\beta^{p-1}m(y) \geq \lfloor \beta^{p-1}m(x) \rfloor + 1$$

selon le lemme 3.1. Enfin, on obtient

$$y = m(y)\beta^{e(y)} \geq (\lfloor \beta^{p-1}m(x) \rfloor + 1)\beta^{1-p}\beta^{e(x)} = rd_p^+(x)$$

On vient de montrer que $]x, rd_p^+(x)] \cap \mathbb{F}_\beta(p, e_{\min}, e_{\max}) = \{rd_p^+(x)\}$; ce qui prouve le théorème.□

Les Arrondis minimaux.

Soit

$$T : \mathbb{R} \to \mathbb{F}_\beta(p, e_{\min}, e_{\max})$$

une application de $\mathbb{R}$ à valeurs dans l'ensemble $\mathbb{F}_\beta(p, e_{\min}, e_{\max})$

Définition 3.1. On dit que T est un arrondi minimal si pour tout $x \in \mathbb{R}$

$$|x - T(x)| = \min_{y \in \mathbb{F}_\beta(p, e_{\min}, e_{\max})} |x - y| \tag{3.6}$$

Les arrondis par défaut et par excès ne sont pas minimaux; mais jouent un rôle décisif dans la détermination des arrondis minimaux; et ce grâce aux propriétés suivantes qu'ils jouissent

Théorème 3.6. Soit $x > 0$. Alors

(a) $rd_p^+(x) - rd_p(x) = \beta\beta^{-p}\beta^{e(x)}$

(b) $rd_p(x) \leq x < rd_p^+(x)$

(c) $[rd_p(x), rd_p^+(x)] \cap \mathbb{F}_\beta(p, e_{\min}, e_{\max}) = \{rd_p(x), rd_p^+(x)\}$.□

Pour $x > 0$, posons

$$\bar{x} = \frac{rd_p(x) + rd_p^+(x)}{2}. \tag{3.7}$$

On peut énoncer le résultat fondamental suivant

Théorème 3.7. Soit $T : \mathbb{R} \longrightarrow \mathbb{F}_\beta(p, e_{\min}, e_{\max})$. Alors T est un arrondi minimal si et seulement si

$$T(x) = \begin{cases} rd_p(x) & \text{si } |x|<|\bar{x}| \\ rd_p^+(x) & \text{si } |x|>|\bar{x}| \end{cases} \tag{3.8}$$

et $T(\bar{x}) \in \{rd_p(x), rd_p^+(x)\}$.

Démonstration. On montre facilement que tout T vérifiant (2.16) avec

$$T(\bar{x}) \in \{rd_p(x), rd_p^+(x)\}.$$

est un arrondi minimal. De même, la réciproque découle sans peine du théorème 3.6.□

Curieusement, le théorème 3.7 dit qu'à quelques détails près, il y a une certaine unicité dans la classe des arrondis minimaux.

Corollaire 3.8. Soient T et T' deux arrondis minimaux. Alors

$$(\forall x \in \mathbb{R}, x \neq \bar{x}) \;\; T(x) = T'(x).\square$$

En conséquence, le choix d'un arrondi minimal dépend uniquement de son évaluation en $\bar{x}$.

Supposons que

$$\begin{aligned} x &= \pm(m_0.m_1m_2...)_\beta\beta^e \\ (\forall j \geq 0) m_j &\in \{0, 1, ..., \beta-1\}, \; m_0 \neq 0, e \in [e_{\min}, e_{\max}] \end{aligned}$$

On vérifie aisément que $\bar{x}$ défini par (3.7) s'écrit de la façon suivante

$$\bar{x} = \pm(m_0.m_1m_2...m_{p-1} + \frac{\beta}{2}\beta^{-p})_\beta\beta^e$$

Supposons que β est impair et posons $c = \left\lfloor \frac{\beta}{2} \right\rfloor = \frac{\beta-1}{2}$. Alors on peut montrer que

$$\frac{\beta}{2} = (c.ccc...)_\beta$$

si bien que $\bar{x}$ puisse admettre une infinité de chiffres significatifs. En pratique, on traite des nombres ayant un nombre fini de chiffres significatifs de sorte que si β est impair, le nombre ne peut guère être détectable par la machine. Pour cette raison, on suppose désormais et sauf mention contraire que la base β est un entier pair.

Grâce au théorème fondamental 3.4, tout arrondi minimal

$$T : \mathbb{R} \longrightarrow \mathbb{F}_\beta(p, e_{\min}, e_{\max})$$

appliqué à un nombre

$$\begin{aligned} x &= \pm(m_0.m_1m_2...)_\beta\beta^e \\ (\forall j &\geq 0) m_j \in \{0, 1, ..., \beta - 1\},\ m_0 \neq 0, e \in [e_{\min}, e_{\max}] \end{aligned}$$

est obtenu en étudiant le chiffre m_p et son voisinage. On distingue quatre cas

cas 1, $m_p > \frac{\beta}{2}$: Dans ce cas, on a $T(x) = rd_p^+(x)$

cas 2, $m_p < \frac{\beta}{2}$: Dans ce cas, on a $T(x) = rd_p(x)$

cas 3, $m_p = \frac{\beta}{2}$ et $\exists j > p / m_j \neq 0$: Dans ce cas, on a $T(x) = rd_p^+(x)$

Avant de parler du quatrième cas, illustrons ces cas en considérant l'exemple suivant

Exemple 3.3. $p = 6, \beta = 10$. Si T est un arrondi minimal, on a

$$\begin{aligned} T(1300996) &= 1301 \\ T(0.0254999499) &= 0.0254999 \\ T(14.65985000001) &= 14.6599 \end{aligned}$$

Le quatrième cas où $m_p = \frac{\beta}{2}$ et $(\forall j > p)\ m_j = 0$ correspond à la situation où $x = \bar{x}$ défini dans (3.7). D'après le théorème 3.7, toute décision $T(\bar{x}) = rd_p(x)$ ou $rd_p^+(x)$ n'affecte pas la

minimalité de l'arrondi T. C'est pourquoi cette situation, ou plutôt ce dilemme, a fait l'objet de débats entre différentes écoles. On suggère par exemple de poser

$$T_{usuel}(\bar{x}) = rd_p^+(x)$$

pour tout x. Cette démarche peut s'appuyer sur une condition de continuité en $\bar{x}$ et donc sur la tendance de croire que la probabilité qu'il existe $j > p$ tel que $m_j \neq 0$ est grande. Aussi, elle peut être justifiée par le fait que cet arrondi assure une certaine optimalité:

$$T_{usuel}(\bar{x}) \geq T(\bar{x}) \quad (\forall T \text{ arrondi minimal})$$

En tous les cas, c'est de cette façon que l'arrondi des ordinateurs VAX (marque commerciale de Digital Equipment Corporation) opère.

On recommande également de départager $rd_p(x)$ et $rd_p^+(x)$ d'une manière équitable en utilisant par exemple l'arrondi pair que nous notons fl_p (ou fl) et que nous adoptons tout au long de ce travail.

Définition 3.2. L'arrondi pair fl_p est l'arrondi minimal tel que

$$fl_p(\bar{x}) = \begin{cases} rd_p(x) \text{ si } m_{p-1} \text{ est pair} \\ rd_p^+(x) \text{ si } m_{p-1} \text{ est impair} \end{cases} \tag{3.9}$$

S'il n'y a aucune confusion à craindre, on écrit $fl = fl_p$.

Dans [42], Reiser et Knuth avancent l'argumentation suivante sur laquelle on s'appuie pour préférer cet arrondi à l'arrondi usuel.

Théorème 3.9 Soit x et y deux nombres flottants. Formons la suite récurrente suivante:

$$x_0 = x, \quad x_n = fl_p[fl_p(x_{n-1} - y) + y)] \qquad (n \geq 1)$$

Alors $(\forall n \geq 0)$ on a $x_n = x$ ou $x_n = x_1$.□

Exemple 3.4. $p = 3, \beta = 10, x = 1, y = -0.555$. En appliquant l'arrondi usuel, on n'obtient pas la propriété du théorème 3.9. En effet

$$\begin{aligned} x_0 - y &= 1.555 \simeq 1.56 \Longrightarrow \\ x_1 &= 1.56 - 0.555 = 1.005 \simeq 1.01 \\ x_1 - y &= 1.565 \simeq 1.57 \Longrightarrow \\ x_2 &= 1.57 - 0.555 = 1.015 \simeq 1.02 \end{aligned}$$

En appliquant fl par contre, on obtient

$$\begin{aligned} fl(x_0 - y) &= fl(1.555) = 1.56 \\ x_1 &= fl(1.56 - 0.555) \\ &= fl(1.005) = 1 \end{aligned}$$

Puisque fl est un arrondi minimal, on a alors d'après le théorème 2.12

$$fl(x) = fl_p(x) = \begin{cases} rd_p(x) & \text{si } |x| < |\bar{x}| \\ rd_p^+(x) & \text{si } |x| > |\bar{x}| \end{cases} \tag{3.10}$$

de manière à déduire que

$$\begin{aligned} |x - fl(x)| &\leq \min\{|x - rd_p(x)|, |x - rd_p^+(x)|\} \\ &\leq \frac{|rd_p^+(x) - rd_p(x)|}{2} = \frac{\beta}{2}\beta^{-p}\beta^{e(x)} \end{aligned}$$

et grâce à (3.9)

$$|\bar{x} - fl(\bar{x})| = \frac{|rd_p^+(x) - rd_p(x)|}{2} = \frac{\beta}{2}\beta^{-p}\beta^{e(x)}$$

Définition 3.3. Le nombre

$$\varepsilon = \frac{\beta}{2}\beta^{-p} \tag{3.11}$$

s'appelle l'unité de précision. On l'appelle également l'unité de l'erreur d'arrondi.

On peut alors affirmer que

$$(\forall x \in \mathbb{R})\ \frac{|x - fl(x)|}{\beta^{e(x)}} \leq \varepsilon$$

3.2 Situations de dépassement.

Lorsqu'un nombre en traitement n'appartient pas à l'envelppe convexe de l'ensemble-machine $\mathbb{F}_\beta(p, e_{\min}, e_{\max})$ il est naturel de dire qu'on est dans une situation de dépassement. Il est clair, par ailleurs que le maximum et le minimum des nombres strictement positifs de $\mathbb{F}_\beta(p, e_{\min}, e_{\max})$ sont respectivement

$$\max p = (\beta - \beta^{1-p})\beta^{e_{\max}}, \quad \min p = \beta^{e_{\min}}$$

de sorte que si

$$x \geq \beta^{e_{\max}+1}$$

on dit qu'on est dans une situation de sur-dépassement (overflow) et si

$$x < \beta^{e_{\min}}$$

on est dans une situation de sous-dépassement (underflow). Il est à noter à ce propos que 0 admet conventionnellement la représentation (qui va être remise en cause après)

$$0 \simeq (\beta - 1)\beta^{e_{\min}-1}$$

La situation de sous-dépassement (underflow) est en fait sujette à quelques anomalies méritant d'être éviter. Considérons l'exemple suivant

Exemple 3.5. $\beta = 10, p = 3, e_{\min} = -98$. posons

$$x = 6.87 \times 10^{-97},\ y = 6.81 \times 10^{-97}$$

On observe que x et y sont à priori deux nombres sensiblement différents en dehors de la zone du sous-dépassement. Mais étrangement

$$\begin{aligned} x - y &= 0.06 \times 10^{-97} \\ &= 6 \times 10^{-99} \simeq 0 \end{aligned}$$

L'anomalie se voit dans la réalisation de cette boucle

$$\textbf{Tant que } (x \neq y)\ z = \frac{1}{x - y}$$

et dans la fausseté de l'équivalence $x = y \Leftrightarrow x - y = 0$.

Afin d'éviter ce genre d'anomalies, on se propose de recourir à la représentation non-normalisée en virgule flottante des nombres [13],[25]. Cette non-normalisation; appelée également dénormalisation; n'est autorisée que si l'exposant est égal à $e_{\min}$. C'est ainsi qu'on peut admettre dans l'ensemble $\mathbb{F}_\beta(p, e_{\min}, e_{\max})$ les nombres

$$\begin{aligned} x &= \pm(m_0.m_1m_2...m_{p-1})_\beta \times \beta^{e_{\min}} \\ (\forall j &= 0 : (p-1)),\ 0 \leq m_j \leq \beta - 1, m_j \in \mathbb{N} \end{aligned}$$

et par conséquent, le minimum de $\mathbb{F}_\beta(p, e_{\min}, e_{\max})$ n'est plus $\beta^{e_{\min}}$, mais

$$\min p = \beta^{e_{\min}-p+1}$$

Dans ce cas, le zéro peut avoir la représentation suivante

$$0 \simeq (\beta - 1)\, \beta^{e_{\min} - p}$$

Bien mieux, tout nombre x tel que $|x| \leq (\beta - 1)\, \beta^{e_{\min} - p}$ est conventionnellement approché par 0.

Grâce à cette approche, appelée sous-dépassement graduel ou progressif (en Anglais: gradual underflow) [18], [25], l'anomalie de l'exemple précédent n'aura pas lieu.

Exemple 3.6. $\beta = 10, p = 3, e_{\min} = -98$. Comme l'exemple précédent, posons

$$x = 6.87 \times 10^{-97},\ y = 6.81 \times 10^{-97}$$

Avec le sous dépassement graduel, on obtient:

$$x - y = 0.06 \times 10^{-97} = 0.6 \times 10^{-98}$$

On peut penser que la situation de dépassement survient dans le cas où les opérands sont proches de $\max p$ ou $\min p$.Auquel cas, toute opération peut causer un débordement. Malheureusement, il s'est avéré que dans certains cas, même si les données sont relativement loins des bornes, les situations de débordement peuvent avoir lieu.

Exemple 3.7. Supposons que $x \geq \sqrt{\frac{\beta}{2}}\beta^{\frac{e_{\max}}{2}}$ et $y > 0$ deux nombres flottants. On veut calculer la norme de Pythagore

$$D = \sqrt{x^2 + y^2}$$

Il est logique de procèder comme suit:

$$\begin{aligned} a &= x^2 \\ b &= y^2 + a \\ D &= \sqrt{b} \end{aligned}$$

Toutefois, cette approche donne lieu à un sur-dépassement (overflow) dès qu'on calcule

$$a = x^2 \geq \beta^{e_{\max}+1}$$

Il importe de remarquer ici que D, étant du même ordre que x et y, est relativement loin de la région du sur-dépassement. Pour remédier à ce genre de comportements, il est nécessaire de recourir à d'autres méthodes ayant plus de robustesse en face des situations de dépassement.

Exemple 3.8. Comme dans l'exemple 3.7, on veut réaliser la norme de Pythagore

$$D = \sqrt{x^2 + y^2}$$

où x et y sont deux nombres positifs flottants par une méthode qui résiste aux situations de dépassement. Pour cela, on propose la méthode suivante. On pose

$$t = \begin{cases} \frac{y}{x} \text{ si } 0 < y \leq x \\ \frac{x}{y} \text{ si } 0 < x \leq y \end{cases}, \quad s = \begin{cases} x \text{ si } 0 < y \leq x \\ y \text{ si } 0 < x \leq y \end{cases}$$

On calcule alors D en utilisant la formule suivante

$$D = s\sqrt{1 + t^2}$$

On observe d'emblé qu'il y a un risque de sous-dépassement. En effet, si $y < x$ et

$$e(y) - e(x) \leq \frac{e_{\min} - p + 1}{2}$$

le nombre t^2 sera dans la zone du sous-dépassement et sera pris pour 0; ceci provient du fait que

$$\begin{aligned} t^2 &= 0.d_1 d_2 \ldots \times \beta^{2(e(y)-e(x))} \\ &= d_1.d_2 d_3 \ldots \times \beta^{e_{\min}-p} \simeq 0 \end{aligned}$$

Dans ce cas, on obtient l'approximation suivante

$$D \simeq x$$

de D dont la précision est tout à fait acceptable. On a

$$\begin{aligned} D - x &= \frac{y^2}{D + x} \\ &\leq \frac{y^2}{x} = xt^2 \\ &< x\beta\beta^{e_{\min}}\beta^{-p} \end{aligned}$$

En pratique $e_{\min} << -p$ si bien que l'erreur relative soit très réduite et par suite x approche acceptablement la norme de Pythagore.

On constate, dans cette analyse, que la situation du sous-dépassement est toujours moins dangereuse que la situation du sur-dépassement. Il s'agit d'une règle générale qui peut être bénéfique en transformant, si besoin est, les problèmes qui sont éventuellement sujets aux sur-dépassements, aux problèmes dont des sous-dépassements peuvent avoir lieu [25].

C'est pour cette raison d'ailleurs qu'en pratique, on conçoit des systèmes, comme les IEEE standard [31] par exemple, de sorte que $|e_{\min}| > e_{\max}$ et; en conséquence; l'inverse de $\beta^{e_{\min}}$ ne se trouve pas dans la situation du sur-dépassement. Bien que l'inverse de $\beta^{e_{\max}}$ puisse être dans la région du sous-dépassement, le problème est valablement résolu en lui donnant 0 comme valeur approchée.

3.3 Erreur relative et ulp.

Il est évident qu'un algorithme numérique est implémenté dans le but d'obtenir des résultats qui sont des nombres réels. Hélas, à cause de nombreuses limitations naturelles, l'obtention des valeurs exactes de ces résultats relève de l'impossible; et ce malgré les travaux tentés çà et là notamment dans le cadre du calcul formel. Entre autres causes, citons en particulier le fait que la capacité mémoire de l'ordinateur est finie; ce qui contraint le compilateur, dans

l'implémentation, d'arrondir après chaque réalisation d'une opération élémentaire. Cette propagation des erreurs d'arrondi donne naturellement lieu à une valeur finale approchée (ou calculée) du résultat final désiré.

Ainsi, au lieu d'une valeur exacte $x \in \mathbb{R}$ d'un résultat, on n'obtient à partir de la machine qu'une valeur approchée $\hat{x} \in \mathbb{F}_\beta(p, e_{\min}, e_{\max})$ (on dit également valeur calculée). De toute évidence, il est intéressant d'avoir un moyen efficace permettant de mesurer l'erreur commise. Dans ce contexte, il est bien connu qu'en comparant l'erreur absolue $|x - \hat{x}|$ à l'erreur relative $\frac{|x-\hat{x}|}{|x|}$, on est dans une situation plus confortable avec l'erreur relative qu'avec l'erreur absolue. Il est également connu que ceci est dû à l'invariance de l'erreur relative par rapport aux changements de l'échelle. Ainsi, on a du mal à accepter l'approximation

$$1234 \simeq 234$$

alors qu'on admet volontier l'écriture

$$10000001000 \simeq 10000000000$$

malgré le fait que l'erreur absolue, étant égale à 1000, soit la même dans les deux cas. Notre attitude vis à vis de ces deux approximations est néanmoins très bien justifiée en introduisant l'erreur relative. En effet, on observe dans la première approximation que l'erreur relative est

$$\frac{1000}{1234} \simeq 0.81$$

alors que dans la seconde, l'erreur relative est

$$\frac{1000}{10000001000} \simeq 10^{-7}$$

La différence est sans appel! Pour avoir un formalisme général de l'erreur, on parle de l'erreur

$$|x - \hat{x}| \text{ relativement à } |y| > 0$$

qui est tout simplement

$$\frac{|x-\hat{x}|}{|y|}$$

En ce qui concerne notre étude où la valeur approchée $\hat{x}$ de x est telle que

$$\hat{x} \in \mathbb{F}_\beta(p, e_{\min}, e_{\max})$$

on distingue trois types d'erreurs relatives

(a) L'erreur $|x-\hat{x}|$ relativement à $|x|$. Posons $E = \frac{|x-\hat{x}|}{|x|} (x \neq 0)$

(b) L'erreur $|x-\hat{x}|$ relativement à $|\hat{x}|$, Posons $\hat{E} = \frac{|x-\hat{x}|}{|\hat{x}|} (\hat{x} \neq 0)$

(c) L'erreur $|x-\hat{x}|$ relativement à $\beta^{e(x)}$, Posons $E' = \frac{|x-\hat{x}|}{\beta^{e(x)}}$. On suppose ici que $e(x) = e(\hat{x})$

On énonce alors le résultat suivant:

Théorème 3.10. (a) Supposons que $E < 1$ (en général $<<1$). Alors

$$\frac{E'}{1+E} \leq \hat{E} \leq \frac{E'}{1-E}$$

Supposons que $\hat{E} < 1$ (en général $<<1$). Alors

$$\frac{\hat{E}}{1+\hat{E}} \leq E \leq \frac{\hat{E}}{1-\hat{E}}$$

On a sous l'hypothèse $e(x) = e(\hat{x})$, $E' = |m(x) - m(\hat{x})|$ et

$$\begin{aligned} \beta^{-1}E' &\leq E \leq E' \\ \beta^{-1}E' &\leq \hat{E} \leq E' \end{aligned}$$

Démonstration. Pour montrer (a), on observe que

$$\begin{aligned} |x-\hat{x}| &= |x|E \\ &\leq |x-\hat{x}|E + |\hat{x}|E \end{aligned}$$

Comme $E < 1$, on obtient directement $\hat{E} \leq \frac{E}{1-E}$. De même

$$\begin{aligned} |x - \hat{x}| &= |\hat{x}|\hat{E} \\ &\leq |x - \hat{x}|\hat{E} + |x|\hat{E} \end{aligned}$$

par suite

$$\hat{E} \geq \frac{|x - \hat{x}|}{|x - \hat{x}| + |x|} = \frac{E}{E + 1}$$

Le point (b) peut être prouvé d'une façon similaire. Le point (c) découle du fait que

$$1 \leq m(x) < \beta, \ 1 \leq m(\hat{x}) < \beta$$

Puisque $E' = m(x)E = m(\hat{x})\hat{E}$, les inégalités s'en déduisent aussitôt. □

Ce théorème dit que dans les conditions normales, E, $\hat{E}$ et E' sont équivalentes au sens que l'utilisation de l'une donne des résultats équivalents à ceux obtenus en utilisant l'autre. Il est important de remarquer à cet égard que $E' = |m(x) - m(\hat{x})|$ si bien que l'erreur relative devient une erreur absolue.

On mesure également l'erreur au moyen de la notion de l'unité dans la dernière place (notée "*ulp*") actuellement à la mode. le terme "*ulp*" est composé d'initiales remplaçant; en guise d'abréviation; "Unit in Last Place". Disons d'emblée que 1 *ulp* est le poids du plus petit chiffre significatif de la mantisse d'un nombre $z \in \mathbb{F}_\beta(p, e_{\min}, e_{\max})$. Posons

$$ulp(z) = \beta^{e(z)-p+1} \tag{3.12}$$

On introduit alors la définition suivante:

Définition 3.4. Soit $x \in \mathbb{R}$ et soit $y \in \mathbb{F}_\beta(p, e_{\min}, e_{\max})$. Posons

$$r = \frac{|y - x|}{ulp(y)} \tag{3.13}$$

On dit alors que "y approche x avec r ulp".

On remarque que (en posant $y = \hat{x}$) $r = E'\beta^{p-1} = \frac{E'}{2\varepsilon}$ où ε désigne l'unité de précision. Il est donc clair que plus r est réduit plus l'approximation $x \simeq y$ est jugée bonne.

Exemple 3.9.

(a) $p = 5, \beta = 10, x = 5.3652, y = 5.3651$. Alors y approche x avec 1 ulp.

(b) $p = 4, \beta = 10, x = 1025.7, y = 1024$. Alors y approche x avec 17 ulp.

(c) $p = 3, \beta = 10, x = 4.501402, y = 4.50$.Alors y approche x avec 1.402 ulp.

(d) $p = 3, \beta = 2$,Il est clair que $1.25 = (1.01)_2$ et $1.125 = (1.001)_2$ de sorte qu'on déduise que 1.25 approche 1.125 avec 1 ulp.

(e) $\beta = 10$. Alors -9 approche 1 avec 10^p ulp.

On voit dans cet exemple surtout la concordance entre la définition 3.4de la phrase " x approche y avec r ulp" et de l'aspect littéraire de cette même phrase. En ce sens, cette mesure consiste à évaluer le nombre des chiffres contaminés en démarrant des chiffres significatifs aux poids plus petit. Le point (e) de cet exemple compare un nombre positif à un nombre négatif et à première vue au moins, le résulat donne l'impression qu'il est rarissime qu'un nombre positif soit approché par un nombre négatif et vice versa.

Une notion classique qui peut être considérée comme duale à la notion de l'ulp ainsi introduite, consiste à évaluer le nombre de chiffres exacts à commencer par les chiffres significatifs au poids plus fort. Dans cette orientation, soit

$$\begin{aligned} t &= t_0.t_1t_2... \\ (\forall j)\ 0 &\leq t_j < \beta \end{aligned}$$

un nombre de $[0, 1]$. On écrit alors

$$o(t) = k$$

si $t_j = 0(\forall j = 0 : (k-1))$ et $t_k \neq 0$. On montre tout de suite que $o(t) = \lfloor -\log_\beta t \rfloor$. Ceci étant, pour $x, y \in [1, 2[$, on dit que y approche x avec k chiffres exacts si $o(x - y) = k$. A l'aide de cette définition, on juge que l'approximation $x \simeq y$ est d'autant plus bonne que k est près de la précision p (voir [10]).

Nous venons donc de présenter des moyens permettant de mesurer l'erreur d'une approximation arbitraire. A propos de cette approximation, nous n'avons rien dit jusque là. Pourtant c'est elle qui peut donner des indications précises sur sa qualité. Pour cela, rappelons que $fl = fl_p$ désigne l'arrondi pair introduit dans la définition 3.2 avec qui, bien entendue, on peut considérer l'approximation élémentaire suivante qu'on peut voir dans un certain sens comme inéluctable à cause du fait qu'elle n'est exacte que si $x \in \mathbb{F}_\beta(p, e_{\min}, e_{\max})$.

$$(x \in \mathbb{R}) \quad x \simeq fl(x)$$

Les approximations que nous considérons ici proviennent de la propagation des erreurs d'arrondi. En conséquence, elles peuvent être vues comme générées par l'approximation élémentaire ainsi introduite et dont l'appréciation semble plus maîtrisable. Commençons notre analyse de cette approximation élémentaire par de simples majorations. Pour $x \neq 0$, on a

$$\begin{aligned} |x - fl(x)| &\leq \min\{|x - rd_p(x)|, |x - rd_p^+(x)|\} \\ &\leq \frac{|rd_p^+(x) - rd_p(x)|}{2} = \frac{\beta}{2}\beta^{-p}\beta^{e(x)} \end{aligned}$$

En conséquence

$$\frac{|x - fl(x)|}{\beta^{e(x)}} \leq \varepsilon \tag{3.14}$$

où $\varepsilon = \frac{\beta}{2}\beta^{-p}$ désigne l'unité de l'erreur d'arrondi introduite dans la définition 3.3. En termes d'*ulp*, la formule (3.14) peut être interprétée en disant que $fl(x)$ approche x avec au plus $\frac{1}{2}$ *ulp*. On peut généraliser (3.14) et énoncer le résultat suivant.

Théorème 3.11. Pour tout $x \in \mathbb{R}^*$, on a:

(a) $|x - fl(x)| \leq \varepsilon\beta^{e(x)}$

(b) $|x - fl(x)| \leq \varepsilon|x|$

(c) $|x - fl(x)| \leq \varepsilon|fl(x)|$.

Démonstration. Le point (a) est déjà établi. Montrons (b). On a:

$$\frac{|x - fl(x)|}{|x|} = \frac{|x - fl(x)|}{m(x)\beta^{e(x)}} \leq \frac{\varepsilon}{m(x)} \leq \varepsilon$$

puisque $m(x) \geq 1$. Un raisonnement similaire permet de montrer (c)

$$\frac{|x - fl(x)|}{|fl(x)|} = \frac{|x - fl(x)|}{m(fl(x))\beta^{e(fl(x))}} \leq \varepsilon$$

puisque $e(fl(x)) = e(x)$ ou $e(x) + 1$. □

Nous procèdons maintenant à montrer que les majorations (a) et (c) sont atteintes tandis que la majoration (c) peut être quelque peu raffinée.

Théorème 3.12. Posons $y = 1 + \frac{\beta}{2}\beta^{-p}$. Alors

$$fl(y) = fl(1 + \frac{\beta}{2}\beta^{-p}) = 1 \tag{3.15}$$

Dans ce cas, on a

$$|y - fl(y)| = \frac{\beta}{2}\beta^{-p} = \varepsilon = \varepsilon fl(y)$$

ce qui montre que les majorations (a) et (c) du théorème 3.11 sont atteintes.

Démonstration. L'égalité (3.15) découle directement de la définition 3.2 de l'arrondi pair. En effet, on est dans le cas où $m_p = \frac{\beta}{2}$ et $m_{p-1} = 0$ de sorte que $fl(y) = rd_p(y) = 1$. □

Concernant l'erreur $|x - fl(x)|$ relativement à $|x|$, on peut obtenir une majoration plus fine que celle obtenue dans (b) du théorème 3.11.

Théorème 3.13. Pour tout $x \in \mathbb{R}^*$, on a:

$$\frac{|x - fl(x)|}{|x|} \leq \frac{\varepsilon}{1 + \varepsilon} \tag{3.16}$$

et cette majoration est atteinte.

Démonstration. On a

$$\frac{|x - fl(x)|}{|x|} = \frac{|x - fl(x)|}{m(x)\beta^{e(x)}} \leq \frac{\varepsilon}{m(x)}$$

Supposons sans perdre en généralité que $x \in [1, 2[$ et écrivons

$$\begin{aligned} x &= rd_p(x) + \sum_{j \geq p} d_j \beta^{-j} \quad (0 \leq d_j \leq \beta - 1) \\ &\geq 1 + \sum_{j \geq p} d_j \beta^{-j} \end{aligned}$$

Si $fl(x) = rd_p(x)$, on a nécessairement

$$\begin{aligned} d_p &\leq \frac{\beta - 2}{2} \text{ ou} \\ d_p &= \frac{\beta}{2} \text{ et } (\forall j > p)\ d_j = 0 \end{aligned}$$

(rappelons qu'on a supposé que β est pair). On montre alors que

$$\sum_{j \geq p} d_j \beta^{-j} \leq \frac{\beta}{2} \beta^{-p} = \varepsilon$$

En conséquence

$$\begin{aligned} \frac{|x - fl(x)|}{|x|} &= \frac{\sum_{j \geq p} d_j \beta^{-j}}{rd_p(x) + \sum_{j \geq p} d_j \beta^{-j}} \\ &\leq \frac{\sum_{j \geq p} d_j \beta^{-j}}{1 + \sum_{j \geq p} d_j \beta^{-j}} \leq \frac{\varepsilon}{1 + \varepsilon} \end{aligned}$$

puisque la fonction $t \longmapsto \frac{t}{1+t}$ est croissante dans $[0, +\infty[$. Si $fl(x) = rd_p^+(x)$, on a nécessairement

$$x \geq \frac{rd_p(x) + rd_p^+(x)}{2} = rd_p(x) + \varepsilon$$

par conséquent

$$x \geq 1 + \varepsilon$$

et

$$\frac{|x - fl(x)|}{|x|} \leq \frac{\varepsilon}{x} \leq \frac{\varepsilon}{1+\varepsilon}$$

Cette majoration est atteinte en posant simplement $x = 1 + \frac{\beta}{2}\beta^{-p}$ auquel cas $fl(x) = 1$. Dans ce cas, on vérifie aisément que

$$\frac{x - fl(x)}{x} = \frac{\varepsilon}{1+\varepsilon}.\square$$

Dans notre modeste recherche bibliographique, nous n'avons connaissance d'aucun article ou ouvrage parlant de la majoration (2.24). Dans son excellent livre: "The accuracy and stability of numerical algorithms" [30]consacré en partie à ce domaine, Higham a exposé la majoration (b) du théorème 3.11 et a précisé qu'elle est stricte. Il est à dire; quand même; que la majoration (3.16) est comparable (surtout lorsque $\varepsilon << 1$) à celle de (b) du théorème 3.11 et n'est par conséquent que théoriquement intéressante. On conclut que les résultats ainsi établis suggèrent, dans l'analyse (basée sur l'erreur relative) d'une approximation, d'exprimer l'erreur relative sous forme d'un multiple de ε ou de $\frac{\varepsilon}{1+\varepsilon}$ d'après le théorème 3.13.

Comme prélude aux opérations flottantes que nous étudierons dans la prochaîne section, les majorations (b), (c) et la majoration (3.16) peuvent être reformulées de manière à faciliter la manipulation des valeurs approchées en ne perdant pas de vue les valeurs exactes.

Corollaire 3.14. Pour tout $x \in \mathbb{R}$,

$$fl(x) = x(1+\alpha), \quad (|\alpha| \leq \frac{\varepsilon}{1+\varepsilon}) \tag{3.17}$$

$$fl(x) = \frac{x}{1+\delta}, \quad (|\delta| \leq \epsilon) \tag{3.18}$$

Démonstration.- Il suffit de poser pour $x \in \mathbb{R}^*$,

$$\alpha = \frac{x - fl(x)}{x} \text{ et } \delta = \frac{x - fl(x)}{fl(x)}$$

et d'appliquer les théorèmes 3.11 et 3.13. □

Comparons maintenant la notion de l'erreur relative à celle de l'*ulp* et considérons l'exemple suivant

Exemple 3.10. $p = 3, \beta = 10, x = 11.05, y = 11$. On a

$$\begin{aligned} ulp(y) &= 10^{1-3+1} = 0.1 \\ x - y &= 0.05 \end{aligned}$$

donc y approche x avec 0.5 *ulp* et l'erreur $|x - y|$ relativement à $|x|$ est $\simeq 0.00452$. D'autre part, on a $16x = 176.8 = 1.768 \times 10^2$ et $16y = 1.76 \times 10^2$ de sorte que $16y$ approche $16x$ avec 8 *ulp* tandis que l'erreur $|x - y|$ relativement à $|x|$ reste toujours la même $\simeq 0.00452$.

Le théorème suivant donne un généralisation naturelle de cet exemple et motive l'option de préférer la base binaire $\beta = 2$.

Théorème 3.15. Supposons que y approche x avec r *ulp* et soit $a > 0$ tel que $e(ay) = 1 + e(y)$. Alors ay approche ax avec $\frac{a}{\beta} r$ *ulp*.

Démonstration. On a

$$\frac{|ax - ay|}{\beta^{e(ay)-p+1}} = \frac{a}{\beta} \frac{|x - y|}{\beta^{e(y)-p+1}} = \frac{a}{\beta} r. \quad \square$$

Posons $a = \beta^2 - 1$ et $x = 1 + \beta^{-k}$ $(k \geq 2)$ (donc $e(x) = 0$). Alors si $c = \beta - 1$

$$ax = cc.\underbrace{00...0}_{k-2}cc$$

de sorte que $e(ax) = 1$. En conséquence, le facteur $\frac{a}{\beta}$ peut être aussi grand que $\beta - \beta^{-1}$. Cette possibilté (fréquente) est due manifestement au : "Phénomène Wobble" (Wobble: Oscillation).

C'est dans le but d'éviter le Wobbling qu'on recommande d'utiliser le système binaire ($\beta = 2$).

Pour comprendre ce phénomène, on considère les deux nombres flottants successifs suivants

$$\begin{aligned} y &= c.cc...cc \times \beta^{e-1} \text{et } \ x = \beta^e \\ (c &= \beta - 1) \end{aligned}$$

Alors

$$ulp(y) = \beta^{e-p} \text{ et } ulp(x) = \beta ulp(y)$$

Posons

$$\begin{aligned} z &= c.cc...cc\frac{\beta}{2} \times \beta^{e-1} \text{ et} \\ z' &= c.cc...cc\frac{\beta - 2}{2}\beta^{e-1} \end{aligned}$$

On observe alors que $fl(z) = x$, $fl(z') = y$ et $\frac{|z-z'|}{\beta^{e-1}} = 1.5\beta^{-p}$. Malgré cette erreur permettant d'écrire approximativement $z \simeq z'$, on voit d'un autre côté que x approche z avec $\frac{1}{2\beta}$ ulp tandis que y approche z' avec $\frac{\beta-2}{2\beta}$ ulp soit une oscillation (un Wobbling) d'un facteur (the Wobble) β (à remarquer que y approche z avec $\frac{1}{2}$ ulp).

3.4 Opérations flottantes de base et modèle standard.

Les algorithmes numériques sont généralement construits sur la base d'opérations élémentaires. En démarrant de données qui sont naturellement des nombres et en réalisant ces opérations élémentaires dans un ordre préscrit par l'algorithme, on obtient les résultats désirés. Comme précisé auparavant, à cause des erreurs d'arrondi survenues après l'implémentation de chaque opération élémentaire, nous n'obtenons que des valeurs approchées (calculées) de ces résultats. Le problème qui consiste à apprécier les valeurs approchées obtenues dépend donc de la manière de réaliser les opérations élémentaires. S'inspirant du corollaire 3.14 de la section précédente, on est enthousiasmé à adopter, dans l'analyse de l'erreur des algorithmes, le modèle standard que nous exposons tout de suite:

Modèle Standard.- Dans l'analyse de l'erreur des algorithmes numériques, on adopte le modèle standard

$$x \; flop \; y = (x \; op \; y)(1+\alpha), \qquad |\alpha| \leq \frac{\varepsilon}{1+\varepsilon} \tag{3.19}$$

ou la version modifiée

$$x \; flop \; y \; = \frac{(x \; op \; y)}{1+\delta}, \qquad |\delta| \leq \varepsilon \tag{3.20}$$

où $flop$ est l'opération qui approche $op \in \{+, -, \times, /, \sqrt{*}, ...\}$.

La façon la plus naturelle de réaliser $flop$ de manière à satisfaire (3.19) et (3.20); c'est d'ailleurs la façon adoptée par les IEEE standard; consiste à poser

$$x \; flop \; y = fl(x \; op \; y) \tag{3.21}$$

C'est-à-dire qu'on réalise d'abord l'opération $x \; op \; y$ d'une manière exacte et on effectue ensuite l'arrondi à la précision exigée; en l'occurence p; en appliquant l'arrondi pair fl introduit dans la définition 3.2.

Définition 3.5. On appelle opération flottante $flop$ approchant une opération élémentaire op, une opération stable dans $\mathbb{F}_\beta(p, e_{\min}, e_{\max})$ et vérifiant (3.19) et (3.20).

Addition et soustraction avec arrondi exact.

Soit

$$x = m(x)\beta^{e(x)} \text{ et } y = m(y)\beta^{e(y)}$$

deux nombres positifs de $\mathbb{F}_\beta(p, e_{\min}, e_{\max})$. Pour réaliser la somme

$$z = x + y$$

de $x, y \in \mathbb{F}_\beta(p, e_{\min}, e_{\max})$, on commence par une étape d'alignement à un même exposant de sorte que si $e(x) \geq e(y)$, on ait

$$z = (m(x) + \beta^{e(y)-e(x)}m(y))\beta^{e(x)}$$

En pratique, on effectue sur $m(y)$ un décalage à gauche par $e(x) - e(y)$ zéros. On normalise (si besoin est) et applique ensuite l'arrondi fl à z pour obtenir la valeur approchée $fl(z)$ de z.

Exemple 3.11. $p = 4, \beta = 10, x = 9.999 \times 10^{14}, y = 8.991 \times 10^{11}$.

On calcule $z = x + y$ en suivant les étapes suivantes

1. Alignement : $y = 0.008991 \times 10^{14}$ (Décalage à gauche par $e(x) - e(y) = 3$ zéros)

2.Sommation des mantisses:

$$m(x) + 10^{-3}m(y) = 10.007991 \Longrightarrow z = 1.0007991 \times 10^{15}$$

3. Arrondi : $fl(z) = 1.001 \times 10^{15}$

En effectuant un décalage par $e(x) - e(y)$ rend le temps de calcul dépendant de x et y donc éventuellement coûteux. Si $e(x) = 90$ et $e(y) = 30$, on doit réaliser un décalage par 60 zéros. Pour éviter ces opérations coûteuses, on énonce le résultat suivant:

Théorème 3.16. Si $e(x) - e(y) \geq p + 1$ et $z = x + y$, $fl(z) = fl(x + y) = x$.

Démonstration. Posons

$$x = m_0.m_1...m_{p-1} \times \beta^{e(x)}$$
$$y = m'_0.m'_1...m'_{p-1} \times \beta^{e(y)}$$

On a $\beta^{e(y)-e(x)}m(y) = \underbrace{0.00...0}_{\geq p+1}m'_0m'_1...m'_{p-1}$ donc le chiffre à la position $p+1$ de $m(x) + \beta^{e(y)-e(x)}m(y)$ vaut 0 de telle manière que $fl(\ m(x) + \beta^{e(y)-e(x)}m(y)) = m(x)$. Le résultat est donc prouvé. □

La réalisation de la soustraction avec arrondi exact se fait exactement comme l'addition avec arrondi exact à ceci près que + est remplacée par −. Pour calculer $z = x - y$, on aligne au même exposant, on effectue la soustraction sur les mantisses et on applique l'arrondi fl. Pour $e(x) \geq e(y)$, on utilise la formule

$$fl(z) = fl[(m(x) - \beta^{e(y)-e(x)}m(y))\beta^{e(x)}]$$

Exemple 3.12. $p = 4, \beta = 10, x = 1.005 \times 10^{14}, y = 8.991 \times 10^{11}$.

On calcule $z = x - y$ en suivant les étapes suivantes

1. Alignement : $y = 0.008991 \times 10^{14}$ (Décalage à gauche par $e(x) - e(y) = 3$ zéros)

2.Soustraction des mantisses:

$$m(x) - 10^{-3}m(y) = 0.996009 \Longrightarrow z = 9.96009 \times 10^{13}$$

3. Arrondi : $fl(z) = 9.96 \times 10^{15}$

Pour éviter la situation où le temps dépendrait de x et y, le théorème 3.16 admet; heureusement; un analogue pour la soustraction qu'on expose dans ce qui suit.

Théorème 3.17. Si $e(x) - e(y) \geq p + 1$ et $z = x - y$, $fl(z) = fl(x - y) = x$.

Démonstration. Posons

$$\begin{aligned} x &= m_0.m_1...m_{p-1} \times \beta^{e(x)} \\ y &= m'_0.m'_1...m'_{p-1} \times \beta^{e(y)} \end{aligned}$$

On a $\beta^{e(y)-e(x)}m(y) = \underbrace{0.00...0}_{\geq p+1}m'_0m'_1...m'_{p-1}$ donc le chiffre à la position $p+1$ de $m(x) - \beta^{e(y)-e(x)}m(y)$ vaut $\beta - 1$ et le chiffre à la position p est $(m_{p-1} - 1) \bmod \beta$. Si $m_{p-1} = 0$, $(m_{p-1} - 1) \bmod \beta = \beta - 1$ comme l'illustre l'exemple suivant

Exemple 3.13. $p = 4, \beta = 10, x = 1.01, y = 1. \times 10^{-7}$. On a

$$\begin{aligned} x - y &= 1.01 - 0.0000001 \\ &= 1.0099999 \Longrightarrow fl(x-y) = 1.01 = x \end{aligned}$$

On en déduit que $fl(\ m(x) - \beta^{e(y)-e(x)}m(y)) = m(x)$ d'où le résultat. □

Multiplication avec arrondi exact.

Soit à multiplier deux nombres flottants

$$\begin{aligned} x &= m(x)\beta^{e(x)} = m_0.m_1...m_{p-1} \times \beta^{e(x)} \\ y &= m(y)\beta^{e(y)} = m'_0.m'_1...m'_{p-1} \times \beta^{e(y)} \end{aligned}$$

qu'on suppose sans perdre en généralité positifs. Bien sûr, le produit $z = x.y$ de x et y vérifie

$$z = m(x)m(y)\beta^{e(x)+e(y)}$$

de telle sorte que le tout revient à calculer le produit des mantisses $m(x)m(y)$. Après normalisation si besoin, on applique enfin l'arrondi fl.

Exemple 3.14. $p = 4, \beta = 10, x = 4.104 \times 10^{6},\ y = 3.864 \times 10^{-11}$. On a

$$\begin{aligned} z &= x.y \\ &= 15.857856 \times 10^{-5} \\ &= 1.5857856 \times 10^{-4} \end{aligned}$$

Par conséquent $fl(z) = fl(x.y) = 1.586 \times 10^{-4}$

Théorème 3.18. Supposons que $x, y \in \mathbb{F}_\beta(p, e_{\min}, e_{\max})$. Alors

$$z = x.y \in \mathbb{F}_\beta(2p, e_{\min}, e_{\max})$$

et $e(m(x)m(y)) = 0$ ou 1.□

Dans bien des cas, on travaille en précision double afin d'améliorer les précisions des résultats approchés. Le théorème 3.18 rappelle que la réalisation en précision double de la multiplication de deux nombres flottants revient à sa réalisation exacte.

Les autres opérations élémentaires.

Les autres opérations élémentaires comme la division, la racine carrée, et même les fonctions élémentaires cos, sin, ... ne peuvent pas être calculées d'une manière exacte. Tout ce qu'on peut dire dans cette direction, c'est de supposer qu'on puisse implémenter une telle opération en précision double $2p$ ou en précision étendue $q \geq p$ (ou $q \geq 2p$) de telle manière qu'un autre arrondi au p chiffres significatifs donne l'approximation souhaitée par le modèle standard (3.19)-(3.20). Dans le cadre de ce travail, nous supposons que la machine dispose d'algorithmes en software ou cablés (hardware) réalisant les opérations élémentaires. Le choix de la précision étendue q se fait en principe de sorte que le chiffre à la position $q+1$ du nombre en traitement; disons x; soit $< \frac{\beta}{2}$ auquel cas, on a $fl_p(x) = fl_p(fl_q(x))$.

3.5 Analyse de l'erreur.

Après cet aperçu (nécessairement non exhaustif) sur les notions de base de l'arithmétique en virgule flottante, on peut dire qu'on dispose maintenant d'un tant soit peu d'outils nous permettant d'analyser l'erreur des algorithmes numériques.

Dans notre contexte, analyser l'erreur d'un algorithme numérique délivrant une valeur approchée $\hat{z}$ d'un résultat z veut dire l'étude de l'erreur relative $\frac{|z-\hat{z}|}{|z|}$ et/ou l'estimation du nombre d'*ulp* avec quoi $\hat{z}$ approche z. Il est bien de préciser qu'il s'agit là de l'analyse de l'erreur en composantes par opposition à l'analyse de l'erreur en normes appliquée en particulier dans les situations où on est intéressé par l'appréciation de l'approximation $\hat{v} \simeq v$ où

$$v \in \mathbb{R}^n, \quad \hat{v} \in \mathbb{F}_\beta(p, e_{\min}, e_{\max})^n$$

sont deux vecteurs. Dans ce cas, l'analyse de l'erreur revient à l'estimation de $\frac{\|z-\hat{z}\|}{\|z\|}$ où $\|.\|$ désigne une norme de $\mathbb{R}^n$.

Perturbation et stabilité numérique.

Bien entendu, un algorithme numérique est appelé à résoudre un problème numérique. En l'appliquant à des données $d_1, d_2, ..., d_m$, il délivre les résultats $r_1, r_2, ..., r_n$ escomptés par le problème en question. On peut dire que l'algorithme évalue d'une manière directe ou indirecte des fonctions f_i aux poins $d_1, d_2, ..., d_m$ de telle sorte que

$$r_i = f_i(d_1, d_2, ..., d_m)$$

Négligeons un moment les erreurs d'arrondi de façon à admettre que l'algorithme calcule exactement les r_i et effectuons une petite perturbation sur les données

$$d_k \longmapsto \tilde{d}_k$$

Naturellement, on obtient

$$\tilde{r}_i = f_i(\tilde{d}_1, \tilde{d}_2, ..., \tilde{d}_m)$$

On définit dans ces conditions le nombre

$$M = \lim_{h \searrow 0} \sup_{\substack{|\tilde{d}_k - d_k| \leq h|d_k \\ k=1:m|}} \{C \ / \ |\tilde{r}_i - r_i| \leq hC|r_i|, i = 1:n\} \tag{3.22}$$

Le nombre M est appelé conditionnement (ou constante de stabilité) du problème en traitement. Il permet d'estimer le nombre de chiffres contaminés dans les résultats suite aux petites perturbations $d_k \longmapsto \tilde{d}_k$. Comme cette analyse n'a rien à voir avec les erreurs d'arrondi et incombe au problème lui même, la définition suivante est donc tout à fait justifiée

Définition 3.6. Appelons M le conditionnement d'un problème numérique PBN tel que

$$r_i = f_i(d_1, d_2, ..., d_m), \ (i = 1:n)$$

On dit qu'un algorithme qui résoud PBN est numériquement stable si au lieu des résultats $r_1, ..., r_n$, il délivre des résultats $\hat{r}_1, ..., \hat{r}_n$ tels que

$$|\hat{r}_i - r_i| \leq c_{n,m} M \varepsilon |r_i|$$

où $c_{n,m}$ (polynôme en n, m) est une constante raisonnablement réduite et ε est l'unité de précision.

Ce bref commentaire à propos de la stabilité numérique (perturbation et conditionnement du problème) donne déjà une première impression sur la complexité de l'analyse de l'erreur et sur les difficultés d'obtenir des résultats acceptablement précis. En effet un algorithme résolvant un problème numérique peut être vu comme une première étape suivi d'un algorithme résolvant un "sous problème " numérique. Vraisemblablement, le problème de trouver méthodiquement un sous problème ayant le conditionnement le plus réduit est inextricable de sorte à faire prévaloir l'approche heuristique dans la conception d'algorithmes valables numériquement.

Produit de n nombres flottants.

Considérons le problème suivant

$$P = x_1.x_2...x_n$$

de multiplier n nombres

$$x_1, x_2, ..., x_n \in \mathbb{F}_\beta(p, e_{\min}, e_{\max})$$

Pour le résoudre, on peut proposer les récurrences suivantes.

$$\begin{aligned} g_1 &= x_1,\ g_k = g_{k-1}.x_k\ (k = 2:n) \\ P &= g_n \end{aligned}$$

A cause des erreurs d'arrondi, on obtient des $\hat{g}_k$ au lieu des g_k ($\hat{g}_1 = g_1$) tels qu'en adoptant le modèle standard

$$\begin{aligned} \hat{g}_k &= \hat{g}_{k-1}.x_k(1+\delta_{k-1})\ \ (k = 2:n) \\ |\delta_{k-1}| &\leq \varepsilon \end{aligned}$$

En fin on obtient au lieu de P sa valeur approchée $\hat{P}$ tel que

$$\begin{aligned} \hat{P} &= P(1+\delta_1)(1+\delta_2)...(1+\delta_{n-1}) \\ &= P(1+\theta_{n-1}) \end{aligned}$$

Comme $|\delta_k| \leq \varepsilon\ (1 \leq k < n)$, on montre que

$$|\theta_{n-1}| \leq \frac{(n-1)\varepsilon}{1-(n-1)\varepsilon} \text{ pour } (n-1)\varepsilon < 1$$

En supposant que les situations de dépassement n'ont pas lieu, on peut alors conclure que

$\hat{P}$ approche P avec au plus $\frac{n-1}{2}$ *ulp*

Sommation de n nombres.

Considérons le problème suivant

$$S = x_1 + x_2 + ... + x_n$$

d'additionner n nombres

$$x_1, x_2, ..., x_n \in \mathbb{F}_\beta(p, e_{\min}, e_{\max})$$

Pour le résoudre, on peut proposer les récurrences suivantes.

$$\begin{aligned} S_1 &= x_1,\ S_k = S_{k-1} + x_k\ (k = 2:n) \\ S &= S_n \end{aligned}$$

En adoptant le modèle, on obtient au lieu des S_j les valeurs approchées $\hat{S}_j$ telles que

$$\begin{aligned} \hat{S}_1 &= x_1,\ \hat{S}_k = (\hat{S}_{k-1} + x_k)(1 + \rho_{k-1}) \qquad (k = 2:n) \\ \hat{S} &= \hat{S}_n,\quad |\rho_{k-1}| \leq \varepsilon \end{aligned}$$

Posons

$$\begin{aligned} 1 + \phi_k &= \prod_{j=k}^{n-1}(1 + \rho_j),\ \phi_n = 0 \Longrightarrow \\ |\phi_k| &\leq \frac{(n-k)\varepsilon}{1 - (n-k)\varepsilon} \end{aligned}$$

On vérifie alors que

$$\begin{aligned} \hat{S}_1 &= x_1,\ (1+\phi_k)\hat{S}_k = (1+\phi_{k-1})\hat{S}_{k-1} + x_k(1+\psi_k) \quad (k=1:n) \\ \hat{S} &= \hat{S}_n,\ \ (1+\psi_k) = (1+\phi_k)(1+\rho_{k-1}),\ \rho_0 = 0 \end{aligned}$$

et

$$\begin{aligned} \hat{S} &= \hat{S}_n = x_1(1+\psi_1) + x_2(1+\psi_2) + ... + x_n(1+\psi_n) \\ &= S + E,\ \ |E| \leq \sum_{k=1}^{n} \frac{(n-k+1)\varepsilon}{1-(n-k)\varepsilon}|x_k| \end{aligned} \tag{3.23}$$

Par conséquent

$$|\hat{S} - S| \leq \frac{n\varepsilon}{1-(n-1)\varepsilon}(|x_1| + |x_2| + ... + |x_n|) \tag{3.24}$$

D'après cette analyse, on se permet de conclure que la somme S de n nombres flottants positifs donne une valeur approchée $\hat{S}$ telle que

$$\hat{S} \text{ approche } S \text{ avec au plus } \tfrac{n}{2}\ ulp$$

et d'après (3.23), on est tenté de dire que lorsqu'il s'agit de nombres positifs, il est plus judicieux de trier d'abord les nombres de manière croissante:

$$x_1 \leq x_2 \leq ... \leq x_n$$

et effectuer ensuite la sommation demandée.

Exemple 3.15. Supposons que $p = 3$, $\beta = 10$ et

$$\begin{aligned} x_1 &= 1.,\ x_2 = 10^{-2},\ x_3 = 3. \times 10^{-3},\ x_4 = 4. \times 10^{-3}, \\ x_5 &= 2. \times 10^{-3}, x_6 = 10^{-3} \\ S &= x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \end{aligned}$$

L'application de l'algorithme précédent en utilisant l'addition avec arrondi exact donne

$$S \simeq \hat{S}^1 = 1.01$$

Toutefois, en réarrangeant les x_i de sorte que

$$\begin{aligned} x_1 &= 10^{-3} < x_2 = 2. \times 10^{-3} < x_3 = 3. \times 10^{-3} < x_4 = 4. \times 10^{-3} \\ &< x_5 = 1. \times 10^{-2} < x_6 = 1. \end{aligned}$$

l'application de l'algorithme en utilisant toujours l'addition avec arrondi exact donne $\hat{S}$ telle que

$$S = \hat{S} = 1.02$$

Le principe qu'on peut dégager de cet exemple consiste à éviter l'addition de deux nombres de grandeurs sensiblement différentes. Il est plutôt conseillé de sommer d'abord les petits nombres pour faire un nombre pouvant se mesurer aux grands nombres. On s'attend bien sûr à ce que cette règle ne s'applique pas en général comme le montre l'exemple suivant

Exemple 3.16. $p = 3, \beta = 10,$

$$\begin{aligned} x_1 &= 1.,\ x_2 = 1. \times 10^4,\ x_3 = -1. \times 10^4,\ x_4 = -1.. \\ S &= x_1 + x_2 + x_3 + x_4 \end{aligned}$$

L'application de l'algorithme en utilisant l'addition et soustraction avec arrondi exact donne

$$S \simeq \hat{S}^1 = -1.$$

Dans ce cas où les x_k n'ont pas un même signe, on préconise de trier les $|x_k|$ dans le sens décroissant

$$|x_1| = |1 \times 10^4| \geq |x_2| = |-10^4.| \geq |x_3| = |1.| \geq |x_4| = |-1.|$$

On obtient alors

$$S = \hat{S} = 0$$

Phénomène de cancellation.

Expliquons d'emblée qu'on entend par phénomène de cancellation (on dit encore phénomène de compensation) le problème de soustraire deux nombres flottants positifs contaminés $\hat{A}$ et $\hat{B}$ tels que $\hat{A} \simeq \hat{B}$. Et ce même si la soustraction se fait d'une manière exacte. Le mot "contaminés" signifie que $\hat{A} = A(1+\theta)$ et $\hat{B} = B(1+\theta')$ approchent deux nombres A et B. Ainsi, au lieu de $D = A - B$, on aura $\hat{D} = \hat{A} - \hat{B}$ et l'erreur relative sera

$$\frac{|\hat{D} - D|}{|D|} \leq \mathrm{O}(\varepsilon)\frac{|A| + |B|}{|A - B|}$$

On a $\hat{A} \simeq \hat{B}$ donc $A \simeq B$ qui peut signifier par exemple que $|A - B| = \mathrm{O}(\varepsilon)$ si bien que l'erreur relative

$$\frac{|\hat{D} - D|}{|D|} \leq O(1)(|A| + |B|)$$

n'exclut pas (plutôt attend) une valeur approchée $\hat{D}$ de D qui est sérieusement imprécise.

Exemple 3.17. $p = 3,\ \beta = 10,\ x = 1.65,\ y = 1.64$. On veut calculer

$$D = x^2 - y^2$$

Si $A = x^2$ et $B = y^2$, on a

$$\begin{aligned} A &= 2.7225, B = 2.6896 \\ fl(A) &= \hat{A} = 2.72, fl(B) = \hat{B} = 2.69 \end{aligned}$$

donc

$$D = 3.29 \times 10^{-2} \text{ et } \hat{D} = fl(A) - fl(B) = 3. \times 10^{-2}$$

et $\hat{D}$ approche D avec 29 ulp. Au lieu de procéder de cette manière passant par le phénomène de cancellation (soustraction de nombres voisins contaminés), il est bien de remarquer que

$$D = (x - y)(x + y)$$

Il est clair que x et y sont voisins mais ils ne sont pas contaminés (D. Goldberg [25] appelle cette situation cancellation bénigne). On a dans ce cas

$$x - y = 1. \times 10^{-2} \text{ et } x + y = 3.29$$

si bien qu'on obtienne $D = \hat{D} = 3.29 \times 10^{-2}$.

Dans la quasi totalité des livres d'analyse numérique où une partie est consacrée à l'analyse de l'erreur et au calcul approché, on met en garde contre ce phénomène de cancellation et recommande, dans la construction des algorithmes, d'éviter, autant que faire ce peut, la soustraction de nombres voisins. Il ne faut pas surtout comprendre de là qu'on recommande de minimiser le nombre de soustractions dans l'algorithme à concevoir. A titre d'exemple, il est même très déconseillé, dans la réalisation de la sommation

$$S = x_1 + x_2 + ... + x_n$$

de n nombres qui sont de signes différents, de procéder comme suit:

1. Déterminer $\mathcal{P} = \{i / x_i \geq 0\}$
2. Calculer $S^1 = \sum_{i \in \mathcal{P}} x_i$

3. Déterminer $\mathcal{N} = \{i/x_i < 0\}$

4. Calculer $S^2 = \sum_{i\in\mathcal{N}}(-x_i)$

5. $S = S^1 - S^2$

Est ce que le pivotement est nécessaire dans les E.G?

Dans la résolution des systèmes linéaires en utlisant les éliminations de Gauss (en abrégé E.G), on préconise le pivotement (partiel) comme approche menant à des solutions acceptablement précises. La réponse définitive à cette question; qui au demeurant reste un sujet de recherche motivant même actuellement; sort quelque peu du cadre qu'on s'est fixé dans ce travail. Ici, nous voulons juste reproduire l'exemple qu'on trouve dans [30],[46] montrant la nécessité de recourir au pivotement.

Exemple 3.18. $p = 3, \beta = 10$. Soit à résoudre le système linéaire 2×2 suivant

$$\begin{bmatrix} 10^{-3} & -1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

La solution exacte est $x = \frac{1}{1+10^{-3}}, y = -\frac{1}{1+10^{-3}}$ et

$$fl(x) = -fl(y) = 0.99 \times 10^{-1}$$

Par contre, en appliquant les E.G., on obtient

$$L = 10^3, U = 1 + 10^3 \rightarrow fl(U) = 1$$

Le problème revient donc à résoudre le système triangulaire

$$\begin{bmatrix} 10^{-3} & -1 \\ 0 & 10^3 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 \\ 10^3 \end{bmatrix}$$

dont la résolution donne

$$\hat{y} = 1 \text{ et } \hat{x} = 0$$

En appliquant le pivotement, on aura à résoudre le système linéaire équivalent

$$\begin{bmatrix} 1 & 1 \\ 10^{-3} & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Les E.G.; cette fois; donnent

$$L = 10^{-3} \text{ et } U = -1 - 10^{-3} \rightarrow fl(U) = -1$$

Le problème revient donc à résoudre le système triangulaire

$$\begin{bmatrix} 1 & 1 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Sa résolution donne

$$\tilde{x} = 1,\ \tilde{y} = -1$$

Il est clair que $\tilde{x}$ et $\tilde{y}$ approchent respectivement x et y avec au plus 0.1 *ulp*.

3.6 Opérations élémentaires avec récupération de l'erreur.

Les erreurs d'arrondi proviennent donc du fait qu'il est impératif de travailler en précision finie. En appliquant une opération élémentaire (unaire, binaire,...); disons $*$ et supposons qu'elle est binaire; à deux nombres flottants

$$x, y \in \mathbb{F}_\beta(p, e_{\min}, e_{\max})$$

on obtient un nombre $z = x * y$ qui dans la plupart du temps n'est pas flottant

$$z = x * y \notin \mathbb{F}_\beta(p, e_{\min}, e_{\max})$$

La seule solution pour réaliser cette opération consiste donc à arrondir le résultat:

$$\hat{z} = fl(z)$$

On vient donc de construire une opération approchée de $*$ que nous notons

$$\circledast \simeq * \tag{3.25}$$

et appelons l'opération $*$ avec arrondi exact. On comprend de là que les quatres opérations usuelles avec arrondi exact sont notées :

$$\oplus,\ \ominus,\ \otimes,\ \oslash$$

En tout état de cause, on a:

$$\hat{z} = z + e,\ |e| \leq \varepsilon|z|$$

Les questions qu'on se pose ici s'articulent autour de la calculabilité de l'erreur e. Il est à préciser que même si e est représentable, il n'y a aucun espoir à améliorer davantage l'approximation $z \simeq \hat{z}$. Seulement, le calcul de l'erreur e: comme on le verra plus loin; peut être exploité à d'autres fins. Commençons notre étude par la soustraction $-$ et son homologue la soustraction avec arrondi exact $\ominus$.

Sur la stabilité de "$-$" dans $F_\beta(p, e_{\min}, e_{\max})$.

Heureusement, il peut se faire dans certains cas, bien que rares, que l'application de la soustraction produit des résultats exacts. Sans aucun doute, le résultat le plus cité, au demeurant le plus exploité, dans cette perspective, est le théorème de Sterbenz[45] que nous exposons dans ce qui suit.

Théorème 3.19 (Théorème de Sterbenz) Supposons que

$$x, y \in \mathbb{F}_\beta(p, e_{\min}, e_{\max}) \text{ et} \tag{3.26}$$
$$\frac{y}{2} \leq x \leq 2y$$

Alors $x - y \in \mathbb{F}_\beta(p, e_{\min}, e_{\max})$.

Démonstration. Supposons sans perdre en généralité que

$$0 \leq y \leq x \leq 2y$$

on en déduit directement que $e(x) = e(y)$ ou $e(y)+1$. Quitte à effectuer un décalage approprié, on suppose que $e(y) = 0$. On remarque que si $e(x) = e(y)$, le résultat est immédiat. Si par contre $e(x) = e(y) + 1$, on aura

$$x = x_0.x_1...x_{p-1} \times \beta \text{ et } y = y_0.y_1...y_{p-1}$$

de sorte que

$$\begin{aligned} x - y &= (x_0.x_1...x_{p-1} - 0.y_0y_1...y_{p-1})\beta \\ &= z_0.z_1...z_{p-1}z_p\beta \end{aligned}$$

Or $0 \leq x - y \leq y < \beta$, il vient alors

$$0 \leq z_0.z_1...z_{p-1}z_p = \frac{x-y}{\beta} < 1$$

par suite $z_0 = 0$ et $x - y \in \mathbb{F}_\beta(p, e_{\min}, e_{\max})$.□

L'importance (plutôt l'élégance) de ce résultat; attribué à Sterbenz[45]; réside surtout dans le fait que la condition posée (3.26) ne dépend ni de la mantisse, ni de l'exposant , ni même de la base. Ce qui facilite sa vérification. Parmi les généralisations de ce théorème, on peut énoncer le résultat suivant dû à Ferguson[22]

Théorème 3.20. Supposons que

$$\begin{aligned} x, y &\in \mathbb{F}_\beta(p, e_{\min}, e_{\max}) \text{ et} \\ & e(x - y) \leq \min(e(x), e(y)) \end{aligned} \tag{3.27}$$

Alors $x - y \in \mathbb{F}_\beta(p, e_{\min}, e_{\max})$.

Démonstration. La démonstration peut être; à quelques détails près; calquée sur la démonstration du théorème 2.24. □

On se propose maintenant de présenter une autre version du théorème 3.19 dans le cas binaire où $\beta = 2$. Posons

$$W_k = \mathbb{F}_\beta(p, e_{\min}, e_{\max}) \bigcap [1 - 2^{-k}, 2 - 2^{-k}],\ k = 1 : (p-1)$$

On peut alors énoncer le résultat suivant.

Théorème 3.21. Soient $x, y \in \mathbb{F}_2(p, e_{\min}, e_{\max})$ tels que

$$\exists j \in \mathbb{Z} \text{ et } k = 1 : (p-1) \ / \ \ x, y \in 2^j W_k$$

Alors $x - y \in \mathbb{F}_2(p, e_{\min}, e_{\max})$.

Pour voir que ce théorème généralise bien le théorème 3.19, observons que W_1 est composé de nombres flottants appartenant à $[\frac{1}{2}, \frac{3}{2}]$ parmi lesquels, on peut construire

$$y = 0.1...1 \simeq \frac{1}{2} \text{ et } \ x = 1.011...1 \simeq \frac{3}{2}$$

ne vérifiant pas la condition (3.26).

Démonstration. Supposons que $x \geq y$. Les intervalles "flottants" W_k sont construits de sorte que l'exposant $e(x)$ de x soit $e(y)$ ou $e(y) + 1$. Par ailleurs $1 \leq x < 2 - 2^{-k}$ veut dire que

$$x = 1.0...0 \times \times ... \times$$

Si $y = 0.1 \times ... \times$, on obtient

$$x - y = 0.m_0...m_{p-1}$$

par suite $x - y \in \mathbb{F}_2(p, e_{\min}, e_{\max})$.□

Il est bien de remarquer; enfin; que si en plus de

$$\begin{aligned} x, y &\in \mathbb{F}_\beta(p, e_{\min}, e_{\max}),\ x \geq y \\ y &\notin \mathbb{F}_\beta(p-1, e_{\min}, e_{\max}) \end{aligned}$$

$e(x) = e(y) + 2$, on vérifie aisément que $x - y \notin \mathbb{F}_\beta(p, e_{\min}, e_{\max})$.

Découpage de la somme.

Considérons la somme

$$s = x + y$$

de deux nombres flottants

$$x,\ y \in \mathbb{F}_\beta(p, e_{\min}, e_{\max})$$

En arrondissant, on obtient

$$\hat{s} = fl(s) = s + e,\ |e| \leq \frac{\varepsilon}{1+\varepsilon}|s|$$

On observe que même si $s \in \mathbb{F}_\beta(p', e_{\min}, e_{\max})$ avec $p' >> p$ (cas tout à fait possible), l'erreur e sera comme $\hat{s}$ appartient à $\mathbb{F}_\beta(p, e_{\min}, e_{\max})$ et mieux encore; comme on va le voir maintenant; peut être déterminée à partir de $\hat{s}$, x et y et indépendamment des caractéristiques de l'arithmétique en virgule flottante adoptée (mantisse, exposant, précision).

Théorème 3.22. Supposons que $\beta = 2$ et $x \geq y \geq 0$. Alors

$$\begin{aligned} \hat{y} &= \hat{s} - x \in \mathbb{F}_2(p, e_{\min}, e_{\max}) \text{ et} \\ e &= \hat{y} - y \in \mathbb{F}_2(p, e_{\min}, e_{\max}) \end{aligned}$$

Démonstration. Puisque $x \geq y \geq 0$, on a $x \leq s \leq 2x$ et puisque $\beta = 2$, $2x \in \mathbb{F}_2(p, e_{\min}, e_{\max})$

par suite $\hat{s} \leq 2x$ car le contraire aboutirait au fait qu'il existe un flottant autre que $\hat{s} = rd_p^+(s)$ dans $]s, \hat{s}]$ qui est $2x$. Un argument similaire montre que $\hat{s} \geq x$. Le théorème de Sterbenz donne alors

$$\hat{y} = \hat{s} - x \in \mathbb{F}_2(p, e_{\min}, e_{\max})$$

Par ailleurs un suivi du déroulement de la réalisation de l'opération $\oplus$ (addition avec arrondi exact) montre bien que l'erreur $e \in \mathbb{F}_2(p, e_{\min}, e_{\max})$.□

Avant d'analyser algorithmiquement ce théorème, considérons d'abord le cas de la soustraction

$$d = x - y$$

de deux nombres flottants positifs

$$x,\ y \in \mathbb{F}_\beta(p, e_{\min}, e_{\max})$$

En arrondissant, on obtient

$$\hat{d} = fl(d) = d + e,\ |e| \leq \frac{\varepsilon}{1+\varepsilon}|d|$$

Théorème 3.23. Supposons que $\beta = 2$ et $x \geq y \geq 0$. Alors si $x \leq 2\hat{y}$, $e = 0$ sinon

$$\begin{aligned} \hat{y} &= x - \hat{d} \in \mathbb{F}_2(p, e_{\min}, e_{\max}) \text{ et} \\ e &= y - \hat{y} \in \mathbb{F}_2(p, e_{\min}, e_{\max}) \end{aligned}$$

Démonstration. Le fait que $e = 0$ si $x \leq 2\hat{y}$ découle du théorème de Sterbenz. Sinon, on vérifie facilement que

$$d \leq x \leq 2d$$

L'argumentation donnée dans la preuve du théorème 2.27 peut être appliquée de sorte à montrer sans peine que

$$\hat{d} \leq x \leq 2\hat{d}$$

et; en vertue du théorème de Sterbenz; que $\hat{y} = x - \hat{d} \in \mathbb{F}_2(p, e_{\min}, e_{\max})$. Dans ce cas également, on vérifie que l'erreur $e \in \mathbb{F}_2(p, e_{\min}, e_{\max})$ $\square$

Les théorèmes 3.22 et 3.23 délivre la méthode à construire pour réaliser le découpage suivant

$$\begin{aligned} (x, y) &\longmapsto (\hat{s}, e) \\ \hat{s} &= fl(s) \\ x + y &= \hat{s} + e \end{aligned} \tag{3.28}$$

Cette méthode utilise, en plus de $\oplus$ et $\ominus$, des comparaisons.

Découpage de la multiplication.

Pour la multiplication

$$w = x \times y$$

de deux nombres flottants x et y, la méthode préconisée consiste à calculer exactement w. C'est ce qu'on fait puisque nous considérons des opérations avec arrondi exact. Seulement dans notre situation, on calcule $fl(w)$ sans détruire w quitte à faire appel à un travail en double précision. Grâce au théorème 3.18, on déduit aisément que

$$v = w - fl(w) \in \mathbb{F}_\beta(p, e_{\min}, e_{\max})$$

par suite

$$x \times y = w = w_H + w_B \quad / w_H, w_B \in \mathbb{F}_\beta(p, e_{\min}, e_{\max}) \tag{3.29}$$

De fait, la décomposition (3.29) peut se faire de différentes manières à condition de pouvoir travailler en double précision et d'être équipé de certains moyens nécessaires comme les arrondis par exemple. Il importe de remarquer par delà que contrairement au découpage de la somme (3.28) où il est exigé que $\hat{s} = fl(s)$, dans notre cas on s'intéresse uniquement à la transformation de $\times$ à $+$ et on peut s'en passer d'une telle exigence .

Il est possible de transformer la multiplication $w = x \times y$ de deux nombres flottants x et y à une somme de quatres flottants en évitant le travail en double précision. On suppose dans ce cas que la précision $p = 2q$ est pair et est connue. On réalise alors les découpages suivants sur x et y.

$$x = rd_q(x) + x_B \quad \text{et } y = rd_q(y) + y_B$$

de sorte que

$$\begin{aligned} w &= w_1 + w_2 + w_3 + w_4 \\ w_1 &= rd_q(x)rd_q(y),\ w_2 = rd_q(x)y_B \\ w_3 &= x_B rd_q(y),\ w_4 = x_B y_B \end{aligned}$$

D'après le théorème 3.18, $w_1, w_2, w_3, w_4 \in \mathbb{F}_\beta(p, e_{\min}, e_{\max})$. On montre bien que rd_q peut être remplacée par fl_q. Cette méthode est inspirée de celle de Dekker[16] qui est plus technique en supposant connue, en plus de la précision paire, l'arrondi pair fl_p seulement.

Chapter 4

Méthodes de Correction pour les Récurrences Linéaires

4.1 Méthodes usuelles de sommation.

Soient $x_1, x_2, ..., x_N$ N nombres appartenant au système $\mathbb{F}_\beta(p, e_{\min}, e_{\max})$ (dont la définition est présentée à (2.8) du chapitre 2) et posons

$$s = x_1 + x_2 + ... + x_N \tag{4.1}$$

Comme + est une opération commutative et associative, on peut envisager différentes façons de calculer cette somme. Rappelons à la lumière du chapitre précédent que si ces méthodes sont mathématiquement équivalents, il n'en demeure pas qu'elles ne le sont pas du point de vue numérique. Dans cette vision, il est important donc d'explorer les différentes méthodes réalisant (4.1). Pour ce faire, soient σ et

$$A_1, A_2, ..., A_k$$

une permutation et une partition de $\{1, 2, ..., N\}$ respectivement. On peut alors dire que tout algorithme calculant s de (4.1) a typiquement la forme suivante

$$\begin{aligned} z_j &= \sum_{i \in A_j} x_{\sigma(i)} \\ s &= z_1 + z_2 + ... + z_k \end{aligned}$$

Bien que la complexité des algorithmes soit considérée, nous concentrons notre attention surtout sur l'aspect numérique puisque la précision des résultats constitue l'objet principal autour duquel ce mémoire s'articule. En guise de clarté et sans nuire à la généralité, nous exposons et étudions deux algorithmes typiques réalisant la sommation (4.1).

4.2 Méthode récursive pour la sommation.

Pour calculer la somme s donnée par (4.1), l'approche récurrente suivante

$$\begin{cases} s_1 = x_1 \\ s_i = s_{i-1} + x_i \quad i \geq 2 \end{cases} \tag{4.2}$$

est naturelle et peut être implémentée comme suit:

Algorithme 4.1.

Cet algorithme calcule s donné par (4.1) via la récurrence (4.2)

$s = x_1$

Pour $i = 2 : N$

$s = s + x_i$

Complexité. $(N - 1)$ additions

En fait, nous avons considéré ce probème à la section 3.5 du chapitre précédent et avons obtenu comme résultat la formule (3.23) que nous rappelons ici. Nous avions dit que dans le système arithmétique en virgule flottante $\mathbb{F}_\beta(p, e_{\min}, e_{\max})$ on obtient à la place des s_i de (4.2) les valeurs approchées $\hat{s}_i$ de sorte que

$$\begin{aligned}
\hat{s} &= \hat{s}_N = s + E \\
|E| &\leq \sum_{k=1}^{N} \frac{(N-k+1)\varepsilon}{1-(N-k)\varepsilon}|x_k| \\
&= \frac{N\varepsilon}{1-(N-1)\varepsilon}|x_1| + \frac{(N-1)\varepsilon}{1-(N-2)\varepsilon}|x_2| + ... + \varepsilon|x_N| \\
&\leq \frac{N\varepsilon}{1-(N-1)\varepsilon}(|x_1| + |x_2| + ... + |x_N|)
\end{aligned}$$

En particulier si les x_i sont positifs, on obtient l'estimation suivante

$$\hat{s} = \hat{s}_N = s(1+\theta), \quad |\theta| \leq \frac{N\varepsilon}{1-(N-1)\varepsilon}$$

prouvant une précision acceptable réalisée par la valeur approchée $\hat{s}$ de s. Dans le cas général, on constate l'intervention du conditionnement

$$\kappa_S = \frac{\sum_{i=1}^{N}|x_i|}{|\sum_{i=1}^{N} x_i|}$$

du problème de sommation lors de de l'évaluation de l'erreur d'arrondi. Pour obtenir une solution avec une bonne précision, il est nécessaire de procéder de manière à ce que κ_S soit multiplié par ε^k, $k \geq 1$. On peut par exemple se proposer de réaliser cet algorithme en double précision. Dans ce cas, si on suppose que $\varepsilon\kappa_S \leq 1$, on en déduit que

$$\hat{s} = \hat{s}_N = s(1+\theta), \quad |\theta| \leq \frac{N\varepsilon}{1-(N-1)\varepsilon^2}$$

Toutefois le travail en double précision conduit d'un côté à un algorithme moins rapide que l'algorithme 4.1 et de l'autre rend l'algorithme non portable (c'est-à-dire qu'il dépend de la machine et de ses caractéristiques).

Avant de clore cette sous section, il importe de noter que la méthode récursive (4.2) de sommation ainsi considérée est de quelques utilités en pratique. A titre d'exemple étudions la méthode d'Euler suivante (voir aussi [39])

$$\begin{cases} y_0 \\ y_{i+1}=y_i+hf(t_i,y_i) \quad (i=0:N-1) \end{cases} \tag{4.3}$$

proposée dans le but de résoudre le problème de Cauchy

$$y(0) = y_0, \quad y' = f(t,y) \quad (t \in [0,T])$$

où f est une fonction de deux variables $(t,x) \in \mathbb{R}_+ \times \mathbb{R}$ suffisamment régulière. Il s'ensuit que, de par la dépendance liant les y_i entre eux, la méthode récursive (4.2) se trouve inéluctable. En pratique, on choisit N de sorte que $h = \frac{T}{N} = 2^m$ soit une puissance de 2 si bien que la multiplication $hf(t_i,y_i)$ se réduise à un simple décalge. Etudions brièvement l'influence de l'erreur d'arrondi sur le procédé (4.3). On a

1. Une erreur d'arrondi $\rho_i \leq \rho$ sur le calcul de $f(t_i,\hat{y}_i)$
2. Une erreur d'arrondi $\alpha_i \leq \alpha$ sur le calcul de $\hat{y}_{i+1}$

En conséquence

$$\begin{aligned} \hat{y}_{i+1} &= \hat{y}_i + h(f(t_i,\hat{y}_i)+\rho_i)+\alpha_i \\ &\hat{y}_i + hf(t_i,\hat{y}_i)+h\rho_i+\alpha_i \end{aligned}$$

En supposant que f est $S-$Lipschitzienne en y, on obtient

$$\begin{aligned} \max_{i=0:N}|\hat{y}_i - y_i| &\leq S(\sum_{i=0}^{N-1} h|\rho_i| + |\alpha_i|) \\ &\leq S(T\rho + N\alpha) \end{aligned}$$

On peut alors montrer que

$$\begin{aligned} \max_{i=0:N}|\hat{y}_i - y(t_i)| &\leq S(T\rho + N\alpha + CTh) \\ &= ST\rho + ST[Ch + \frac{\alpha}{h}] \end{aligned}$$

Cette majoration dit que les erreurs d'arrondi sur les y_i exigent du pas h de la méthode de ne pas déscendre en dessous d'un seuil $h_{opt} \approx \sqrt{\frac{\alpha}{C}}$.

4.3 Méthode parallèle pour la sommation.

Pour calculer la somme s donnée par (4.1), on adopte également l'approche suivante. On suppose d'abord que $N = 2^m$ quitte à rajouter autant de zéros que la situation le demande. Ensuite, on considère les récurrences

$$\begin{aligned} x_j^0 &= x_j \ (j = 1 : N) \\ x_j^k &= x_{2j-1}^{k-1} + x_{2j}^{k-1} \quad (\ k = 1 : m, j = 1 : 2^{m-k}) \\ s &= x_1^m \end{aligned} \tag{4.4}$$

qu'on peut implanter via l'algorithme suivant

Algorithme 4.2.

Cet algorithme calcule s en réalisant les récurrences (4.4)

Pour $k = 1 : m$

  Pour $j = 1 : 2^{m-k}$

    $x_j = x_{2j-1} + x_{2j}$

Complexité: N-1 additions

On observe que cet algorithme utilise le même nombre d'opérations ($N - 1$ additions) que celui de l'algorithme 4.1. Cependant cet algorithme a l'avantage d'être plus adapté au traitement en parallèle. En effet, on introduit dans ce contexte la boucle parallèle **ParPour** signifiant que les instructions situées au nid de **ParPour** sont exécutées en parallèle. Ainsi la boucle suivante

$$\begin{aligned} \textbf{ParPour } j &= 1 : 2^{m-k} \\ x_j &= x_{2j-1} + x_{2j} \end{aligned}$$

est à interpréter comme suit: Les x_j (à gauche de l'instruction) sont calculés simultanément. En supposant qu'on dispose de $N-1$ ALU (ou additionneurs) $A_1, A_2, ..., A_{N-1}$, chacune des instructions $x_j = x_{2j-1} + x_{2j}$ pour $j = 1 : 2^{m-k}$ est réalisée au niveau du processeur A_j. Le résultat de l'addition $x_{2j-1} + x_{2j}$ est ensuite stocké dans x_j. De cette façon on peut dire que le temps d'exécution de cette boucle parallèle est égal à une unité de temps $1T(+)$ pour réaliser une addition flottante. Cette explication; inspirée du modèle PRAM (*Parallel Random Access Machine)*; du parallélisme est à cheval entre la pratique et la théorie et est de fait loin de la réalité. Pour une rélalisation parallèle pratique, on doit tenir compte de plusieurs paramètres (inhérents à la technologie) parmi lesquels on peut citer; à titre d'exemple; l'interconnexion entre les processeurs et la mémoire. Pour plus de détails, nous référons le lecteur intéressé aux livres [2], [14], [15].

À la lumière de ce point, on peut proposer l'algorithme parallèle suivant

...rithme 4.3.

...algorithme est parallèle et calcule s en réalisant les récurrences (3.4). $T(+)$ désigne le ...quis pour additionner deux nombres flottants.

...r $k = 1 : m$

ParPour $j = 1 : 2^{m-k}$

$x_j = x_{2j-1} + x_{2j}$

Complexité: $mT(+)$, $N - 1 = 2^m - 1$ additionneurs

Avec le progrès technologique actuel, ce type d'algorithmes est tout à fait réalisable pourvu que le nombre de processeurs ne soit pas trop élevé.

L'analyse de l'erreur de cet algorithme est similaire à celle menée pour la sommation récursive de la section précédente. En tenant compte des erreurs d'arrondi sous le modèle standard, les récurrences (4.4) deviennent

$$\begin{aligned} \tilde{x}_j^0 &= x_j \ (j = 1:N) \\ \tilde{x}_j^k &= \left(\tilde{x}_{2j-1}^{k-1} + \tilde{x}_{2j}^{k-1}\right)(1+\delta_{jk}) \quad (\ k = 1:m, j = 1:2^{m-k}, |\delta_{jk}| \le \varepsilon) \\ \tilde{s} &= \tilde{x}_1^m \end{aligned} \tag{4.5}$$

où $\tilde{y}$ désigne ici la valeur approchée de y. On a le résultat suivant

Théorème 4.1. Pour tout $k = 1:m$ on a

$$\left|\tilde{x}_j^k - x_j^k\right| \le \frac{k\varepsilon}{1-k\varepsilon}(|x_{2^k j}| + |x_{2^k j-1}| + \cdots + |x_{2^k(j-1)+1}|)$$

pour tout $j = 1:2^{m-k}$. En particulier

$$|\tilde{s} - s| \le \frac{\lceil \log_2 N \rceil \varepsilon}{1 - \lceil \log_2 N \rceil \varepsilon}(|x_N| + |x_{N-1}| + \cdots + |x_1|) \tag{4.6}$$

Démonstration. Si $k = 1$ le théorème est immédiat. Si le théorème est vrai pour $k-1$, on aura

$$\left|\tilde{x}_{2j}^{k-1} - x_{2j}^{k-1}\right| \le \frac{(k-1)\varepsilon}{1-(k-1)\varepsilon}(|x_{2^k j}| + |x_{2^k j-1}| + \cdots + |x_{2^k j-2^{k-1}+1}|)$$

$$\left|\tilde{x}_{2j-1}^{k-1} - x_{2j-1}^{k-1}\right| \le \frac{(k-1)\varepsilon}{1-(k-1)\varepsilon}(|x_{2^k j-2^{k-1}}| + |x_{2^k j-2^{k-1}-1}| + \cdots + |x_{2^k(j-1)+1}|)$$

Par ailleurs cela permet d'écrire

$$\begin{aligned} \left|\tilde{x}_j^k - x_j^k\right| &\le \left|\tilde{x}_{2j-1}^{k-1} - x_{2j-1}^{k-1}\right| + \left|\tilde{x}_{2j}^{k-1} - x_{2j}^{k-1}\right| + \varepsilon|\tilde{x}_{2j-1}^{k-1} + \tilde{x}_{2j}^{k-1}| \\ &\le (1+\varepsilon)\left|\tilde{x}_{2j-1}^{k-1} - x_{2j-1}^{k-1}\right| + (1+\varepsilon)\left|\tilde{x}_{2j}^{k-1} - x_{2j}^{k-1}\right| \\ &\quad + \varepsilon(|x_{2^k j} + x_{2^k j-1} + \cdots + x_{2^k(j-1)+1}|) \end{aligned}$$

En utilisant l'hypothèse de récurrence on obtient

$$\begin{aligned}\left|\tilde{x}_j^k - x_j^k\right| &\leq \left(\frac{(k-1)\varepsilon(1+\varepsilon)}{1-(k-1)\varepsilon} + \varepsilon\right)(|x_{2^k j}| + |x_{2^k j-1}| + \cdots + |x_{2^k(j-1)+1}|) \\ &\leq \frac{k\varepsilon}{1-k\varepsilon}(|x_{2^k j}| + |x_{2^k j-1}| + \cdots + |x_{2^k(j-1)+1}|). \quad \square\end{aligned}$$

En résumé, appelons $\hat{s}$ et $\tilde{s}$ les valeurs approchées de la somme s donnée par (4.1) en appliquant la méthode récursive (4.2) et la méthode dichotomique (parallèle) (4.4), respectivement. On vient de montrer que

Méthode récursive (4.2)	*Méthode parallèle (4.4)*
$\lvert\hat{s}-s\rvert \leq \frac{N\varepsilon}{1-N\varepsilon}\kappa_S\lvert s\rvert$	$\lvert\tilde{s}-s\rvert \leq \frac{\lceil\log_2 N\rceil\varepsilon}{1-\lceil\log_2 N\rceil\varepsilon}\kappa_S\lvert s\rvert$

où $\kappa_S = \frac{\sum_{i=1}^N |x_i|}{|\sum_{i=1}^N x_i|}$ désigne le conditionnement du problème (4.1).

Dans les deux situations, on observe que l'analyse de l'erreur menée considère le pire des cas et suppose que si le conditionnement $\kappa_S >> 1$, on ne peut pas assurer des résultats approchés avec de bonnes précisions. Si par contre κ_S est comparable à 1 (c'est le cas lorsque les x_i sont positifs), l'analyse de l'erreur menée montre que la méthode parallèle donne un résultat relativement précis.

Il a été établi que, dans la classe des algorithmes calculant s sans recourir à aucune opération élémentaire autre que + et n'utilisant que $(N-1)$ additions, la méthode parallèle est l'algorithme le plus optimal dans le sens suivant. Au pire des cas, la méthode parallèle fournit une valeur approchée plus précise de s [48]. Dans ce contexte il est bien de rappeler la remarque faite par Wilkinson[49] disant que l'analyse de l'erreur théorique pèche en général par un excès de pessimisme et conduit à des facteurs multiplicatifs exagérés par rapport à la réalité.

4.4 Méthodes de compensation.

Les méthodes de compensations ont été découvert indépendemment par de nombreux auteurs dans les années 70 et ont pour but d'améliorer la précision de la valeur approchée de la somme

s au prix d'un coût algorithmique plus élevé [41], [39]„ [34],[7],[21],... Malgré le caractère classique de cette méthode, elles ont connu récemment un regain d'attention surtout après les travaux de synthèses de Higham[28],[30]. Nous tentons d'expliquer ces méthodes à partir d'un formalisme mathématique clair. Pour ce faire, faisons appel à l'addition flottante composée (3.28) introduite au chapitre précédent:

$$x \oplus y = (fl_2(x+y), e)$$

signifiant que non seulement $fl_2(x+y)$ est calculé mais également l'erreur e en utilisant l'approche suggérée dans les théorèmes 3.22 et 3.23. Cette opération peut être réalisée via la procédure suivante suivante:

Procédure *PFC(données: x, y; résultats: z, e)*

1. $z = x + y$
2. *si* $|x| \geq |y|$ alors
3. $\quad yh = z - x$
4. $\quad e = yh - y$
5. *sinon*
6. $\quad xh = z - y$
7. $\quad e = xh - x$

Cette procédure réalise l'addition flottante composée. C'est-à-dire qu'elle réalise la décomposition suivante:

$$\hat{z} = z + e, \quad |e| \leq \varepsilon |z|$$

Appliquons cette opération à la récurrence (4.2)

$$\begin{cases} s_1 = x_1 \\ s_i = s_{i-1} + x_i \end{cases} \quad i \geq 2$$

qui dans le système des nombres en virgule flottante avec cette addition composée devient

$$\begin{cases} s_1 = x_1, \ e_1 = 0 \\ \hat{s}_i = fl_2(\hat{s}_{i-1} + x_i), \ e_i \end{cases} \quad i \geq 2$$

avec

$$|e_i| \leq \varepsilon |\hat{s}_i| \tag{4.7}$$

L'idée consiste alors à calculer la somme des erreurs

$$F = e_2 + e_3 + \cdots + e_N$$

Si par exemple l'algorithme 4.1 est appliqué, on obtient au lieu de F sa valeur approchée $\hat{F}$ tel que

$$|\hat{F} - F| \leq \frac{\varepsilon N}{1 - \varepsilon N}[|e_2| + \cdots + |e_N|]$$

En utilisant (4.7) on obtient

$$\begin{aligned} |\hat{F} - F| &\leq \frac{\varepsilon^2 N}{1 - \varepsilon N}[|\hat{s}_2| + \cdots |\hat{s}_N|] \\ &\leq \frac{\varepsilon^2 N^2}{1 - \varepsilon N}[|x_1| + \cdots + |x_N|] \end{aligned}$$

En utilisant le principe disant que les estimations à priori qu'on obtient péchent en général par un excès de pessimisme et sont toujours supérieurs des bornes réelles et en utilisant le fait que

$$\frac{\varepsilon^2 N^2}{1 - \varepsilon N} = \varepsilon^2 N^2 + \mathrm{O}(\varepsilon^3)$$

on peut se permettre comme dans Golub et Van Loan[26] de prétendre que

$$|\hat{F} - F| \leq \varepsilon^2 N^2 [|x_1| + \cdots + |x_N|] \tag{4.8}$$

Ces estimations nous motivent à proposer, dans ce qui suit, la méthode typique de compensation:

Algorithme 4.4.

Cet algorithme calcule s donné par (4.1) via la récurrence (4.2)

1. $s = x_1$
2. *Pour* $i = 2 : N$

3. $PFC(s, x_i; s, e_i)$

4. $F = e_2$

5. *Pour* $i = 3 : N$

6. $F = F + e_i$

7. $s = s - F$

Complexité. $(4N - 5)$ ± et $(N - 1)$ comparaisons

Dans le système des nombres en virgule flottante, on a, au lieu de la somme des erreurs F, formée à partir des lignes 4, 5 et 6, sa valeur approchée $\hat{F}$ vérifiant la majoration (4.8). On a également une valeur approchée $\hat{s}$ de s après exécution des lignes 1-3 qui d'après la section 1 vérifie

$$|\hat{s} - s| \leq \frac{N\varepsilon}{1 - N\varepsilon}(|x_1| + |x_2| + ... + |x_N|)$$

et une autre valeur approchée qu'on note $\hat{s}_{\text{corrigé}}$ de s obtenue après exécution de la ligne 7. Il est tout à fait clair que

$$\hat{s}_{\text{corrigé}} = (\hat{s} - \hat{F})(1 + \delta), \quad |\delta| \leq \varepsilon \tag{4.9}$$

Bien entendu, on souhaite que $\hat{s}_{\text{corrigé}}$ soit plus proche que $\hat{s}$ de s.

Théorème 4.2: Supposons que

$$\varepsilon N^2 \kappa_S \leq 1 \tag{4.10}$$

Alors

$$|\hat{s}_{\text{corrigé}} - s| \leq (2\varepsilon + \varepsilon^2)|s|$$

Démonstration. On a en fait pour chaque $i = 2 : N$, les valeurs approchées $\tilde{s}_i$, $\hat{s}_i$ de s tels que

$$\tilde{s}_i = \hat{s}_{i-1} + x_i$$

et

$$\begin{aligned} \hat{s}_i &= \tilde{s}_i + e_i \\ &= \hat{s}_{i-1} + x_i + e_i \end{aligned}$$

Par conséquent

$$\hat{s}_i = (x_1 + \cdots + x_i) + (e_1 + \cdots e_i)$$

et

$$\hat{s} = s + F$$

Il en découle que

$$\begin{aligned} \hat{s}_{\text{corrigé}} - s &= (\hat{s} - \hat{F})(1+\delta) - s(1+\delta) + \delta s \\ &= (F - \hat{F})(1+\delta) + \delta s \end{aligned}$$

par suite

$$\begin{aligned} |\hat{s}_{\text{corrigé}} - s| &\leq \varepsilon(1+\varepsilon)|s| + \varepsilon|s| \\ &= (2\varepsilon + \varepsilon^2)|s|.\square \end{aligned}$$

Ce théorème affirme que si la condition (4.10) est satisfaite, l'algorithme 4.4 délivre une valeur approchée $\hat{s}$ de la somme s telle que

$$\hat{s} = s(1+\alpha) \quad / \quad |\alpha| \leq 2\varepsilon + \varepsilon^2$$

Supposons que $\varepsilon = 2^{-100}$, $N < 2^{25}$ et le problème de sommation est si mal conditionné que $\kappa_S = 2^{50}$. Alors la condition (4.10) est satisfaite puisque, dans ce cas de figure, on a

$$\varepsilon N^2 \kappa_S \leq 1$$

Cet exemple montre que la condition (4.10) exigée par le théorème 4.2 est, en pratique, tout à fait raisonnable.

Pour convaincre davantage, précisons que le formalisme ainsi présenté est essentiellement fondé sur l'idée de récupérer les erreurs après chaque addition élémentaire et ensuite sommer ces erreurs de manière à obtenir une valeur comparable à la somme approchée obtenue si bien qu'en la rajoutant on obtient une amélioration! On en déduit de suite que cette approche recèle

d'autres performances dépendant de la façon de réaliser la somme des erreurs

$$F = e_2 + e_3 + \cdots + e_N$$

En effet, si au lieu de la méthode récursive utilisée dans l'algorithme 4.4, on calcule F via la méthode dichotomique (parallèle) (4.4), l'estimation (4.8) devient dans ce cas

$$|\hat{F} - F| \leq \varepsilon^2(\log^2 N)[|x_1| + \cdots + |x_N|] \tag{4.11}$$

et ce en utilisant la majoration (4.6) issue de l'analyse de l'erreur de la méthode parallèle. Dans ce cas, il suffit de modifier les lignes 4-6 de l'algorithme 4.4 et les remplacer par les instructions réalisant les récurrences (4.4). On propose alors l'algorithme suivant:

Algorithme 4.5.

Cet algorithme calcule s donné par (4.1) via la récurrence (4.4)

1. $s = x_1$
2. *Pour* $i = 2 : N$
3. $\quad PFC(s, x_i; s, e_i)$
4. Pour $k = 1 : m$
5. $\quad$ Pour $j = 1 : 2^{m-k}$
6 $\quad\quad e_j = e_{2j-1} + e_{2j}$
7. $F = e_N$
8. $s = s - F$

Complexité. $(4N - 5)$ + et $(N - 1)$ comparaisons

L'analyse de l'erreur de cet algorithme peut se faire d'une manière similaire à celle du théorème 4.2. On énonce le résultat suivant.

Corollaire 4.3 Supposons que

$$(\log^2 N)\varepsilon_0 \leq 1 \tag{4.12}$$

Alors

$$|\tilde{s}_{\text{corrigé}} - s| \leq (2\varepsilon + \varepsilon^2)|s|$$

où $\tilde{s}_{\text{corrigé}}$ est la valeur approchée de s obtenue par l'algorithme 4.5. □

La condition (4.12) est évidemment plus raisonnable que (4.10) puisque $\log N << N$. Supposons que $\varepsilon = 2^{-100}$, $\kappa_S = 10^9$ (c'est-à-dire que le problème de sommation (4.1) est très mal conditionné) et $N \leq 2^{2^{20}}$. Alors la condition (4.12) est satisfaite. Fort de ce constat et des avantages du parallélisme, nous préconisons alors d'utiliser la méthode parallèle (4.4) dans le calcul de la somme des erreurs.

A partir de cette étude, il devient clair que l'idée consiste à calculer la somme des erreurs d'une manière efficace. Il en découle donc que l'approche présentée peut être vue comme itérative en appliquant la méthode de compensation au calcul de la somme des erreurs. Ce schéma itératif donne lieu aux algorithmes de distillation [34].

4.5 Amélioration itérative et produit scalaire

De fait, la méthode de compensation pour calculer la somme s donnée par (4.1) est, dans la littérature, réalisée via l'algorithme suivant (voir [28],[30])

Algorithme 4.6.

Méthode de Compensation

1. $s = 0; e = 0$
2. Pour $i = 1 : N$
3. $\quad temp = s$
4. $\quad y = x_i + e$
5. $\quad s = temp + y$
6. $\quad e = (temp - s) + y$

Complexité: $4N$ $(\pm)$

On observe que cet algorithme, n'utilisant pas les comparaisons, est moins coûteux que les algorithmes 4.4 et 4.5 proposés dans la section précédente. Néanmoins la détermination exacte de l'erreur commise dans l'addition n'est pas garantie si bien qu'on s'attendrait à une contre-performance en matière de la précision de la valeur approchée du résultat s. Il est à noter quand-même que le résultat suivant établi notamment par Knuth[35] entre autres (voir aussi [41]) est le fait motivant de sugérer cet algorithme.

Théorème 4.4. Appelons $\hat{s}$ la valeur approchée de s obtenue par l'algorithme 6. Alors

$$\hat{s} = \sum_{i=1}^{n}(1+\alpha_i)x_i \ , \ |\alpha_i| \leq 2\varepsilon + O(N\varepsilon^2)$$

En particulier

$$|\hat{s} - s| \leq (2\varepsilon + O(N\varepsilon^2))\kappa_S|s|$$

Démonstration. La démonstration est trop technique. On réfère le lecteur intéressé à [35] et [25].□

La majoration obtenue par ce théorème est certes plus intéressante que celles obtenues par la méthode récursive et la méthode parallèle et encourage à penser que l'algorithme 4.6 est numériquement plus stable que les algorithmes 4.1 et 4.2. Cependant on voit dans ces trois algorithmes 4.1, 4.2 et 4.6, le conditionnement κ_S est omniprésent et est visiblement inévitable. De ce fait ces algorithmes ne sont intéressants que si le problème de calculer (4.1) est bien conditionné. C'est-à-dire que κ_S est comparable à 1. Par exemple lorsque les x_i sont positifs. On conclut alors que les algorithmes 4.4 et 4.5, pouvant être appliqués même aux problèmes mal conditionnés, sont du point de vue numérique nettement plus performants que les algorithmes 4.1, 4.2 et 4.6.

En cherchant à trouver les origines conduisant aux méthodes de compensation proposées dans la littérature, on est tenté de penser que l'idée se base sur le calcul même approché de l'erreur commise dans la réalisation de l'addition. Nous allons ici explorer ici une autre idée qui à notre avis n'est pas considérée jusque là. Il s'agit de l'approche de l'amélioration itérative que nous allons aborder au chapitre suivant. L'idée découle de la formalisation en termes de

matrices de la méthode récursive (4.2). Plus précisement, les récurrences (4.2) peuvent être vues comme la méthode classique de résolution du système bidiagonal suivant

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ -1 & 1 & 0 & \cdots & 0 \\ 0 & -1 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_N \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}$$

L'idée que nous proposons consiste à appliquer l'approche de l'amélioration itérative à ce système. Ceci nous amène à considérer l'algorithme suivant :

Algorithme 4.7

Calcul de s par l'amélioration itérative

1. $s_0 = 0$; $e_0 = 0$
2. Pour $i = 1 : N$
3. $\quad s_i = s_{i-1} + x_i$
4. $\quad g_i = s_i - s_{i-1}$
5. $\quad e_i = e_{i-1} + (g_i - x_i)$
6. $\quad s_i = s_i + e_i$

Complexité: $5N(+)$

A première vue, cet algorithme est coûteux en utilisant $5N$ additions au lieu de $4N$ et est présenté de façon à l'inscrire dans un autre cadre. Heureusement, il se trouve que cet algorithme peut être optimisé et transformé à l'algorithme 4.6. Procédons à la démonstration de cette observation.

Remarquons d'abord que a ligne 6 peut être omise et la ligne 3 peut être remplacée par les affectations

$$g_i = x_i + e_{i-1}, \; s_i = s_{i-1} + g_i$$

Ensuite les lignes 4-5 peuvent être remplacées par l'affectation

$$e_i = e_{i-1} + (s_i - s_{i-1})$$

On obtient alors la version modifiée suivante qui est équivalente à l'algorithme 4.6.

Algorithme 4.8.

Version modifiée de l'Algorithme 4.7.

1. $s_0 = 0$; $e_0 = 0$
2. Pour $i = 1 : N$
3. $\quad y_i = x_i + e_{i-1}$
4. $\quad s_i = s_{i-1} + y_i$
5. $\quad e_i = e_{i-1} + (s_i - s_{i-1})$

Complexité: $4N(\pm)$

En conclusion, on peut qualifier la méthode de l'amélioration itérative ainsi décrite parmi les méthodes de compensation. Nous allons exploiter cette idée dans la section prochaine pour proposer des algorithmes numériques pour les récurrences linéaires. Auparavant, il importe de remarquer que les méthodes de compensation étudiées et présentées dans ce chapitre peuvent être appliquées au calcul du produit scalaire

$$\begin{aligned} c &= < u, v > \\ &= u_1 v_1 + \cdots + u_n v_n \end{aligned} \tag{4.13}$$

qui constitue une opération clé dans le calcul des matrices puisque les opérations matricielles comme le produit de deux matrices les décomposition LU, de Gram-Schmidt... sont toutes générées par le produit scalaire. Il est donc très important de calculer d'une manière précise l'opération (4.13).

Pour ce faire, l'idée est faire appel à la méthode de Dekker[16] présentée au chapitre précédent et qui consiste à transformer la multiplication en une addition. Pour simplifier, on suppose

que la machine réalise exactement la multiplication $a \times b$ de deux nombres flottants a et b de sorte que la décomposition suivante soit possible

$$a \times b = f + q; \quad fl(a \times b) = f \tag{4.14}$$

Exemple 2.1. $\beta = 10,\ p = 4,\ a = b = 9.999$. On a

$$\begin{aligned} c &= 99.980001 \\ &= 9.9980001 \times 10 \\ &= f + q \end{aligned}$$

où

$$f = 9.998 \times 10 \quad \text{et} \quad q = 10^{-6}$$

Cet exemple illustre le fait que le produit $a \times b$ de

$$a, b \in \mathbb{F}_\beta(p, e_{\min}, e_{\max})$$

contient au plus $2p$ chiffres significatifs si bien que la décomposition flottante (4.14) soit toujours valable.

Ceci étant, réalisons la décomposition flottante de chaque terme $u_k v_k$ du produit scalaire (4.13) et écrivons

$$u_k v_k = f_k + q_k \ / \ \ fl(u_k v_k) = f_k$$

Il en découle que le produit scalaire (4.13) devient

$$c = f_1 + q_1 + \cdots + f_n + q_n \tag{4.15}$$

C'est-à-dire un simple problème de sommation de $2n$ nombres flottants. En conséquence le tout revient à appliquer les méthodes de compensation notamment les algorithmes 4.4 et 4.5 afin

d'obtenir une valeur approximative précise. Notons par

$$\kappa_{ps} = \frac{\sum_{k=1}^{n} |u_k v_k|}{|c|}$$

le conditionnement du problème de calculer (4.13). On peut alors affirmer que si

$$\varepsilon(1+\log n)^2 \kappa_{ps} \leq 1 \tag{4.16}$$

et la sommation (4.15) est calculé via l'algorithme 4.5, on obtient à la place de c sa valeur approchée $\hat{c}$ tel que

$$|\hat{c} - c| \leq (2\varepsilon + \varepsilon^2)|c|$$

Corollairement à ce résultat, on déduit aussitot que si le conditionnement

$$\kappa_{pm} = \max_{i,j;C_{ij}\neq 0} \frac{\sum_{k=1}^{r} |A_{ik}B_{kj}|}{|C_{ij}|}$$

du produit $C = A.B$ de deux matrices $A \in \mathbb{R}^{m\times r}$ et $B \in \mathbb{R}^{r\times n}$ vérifie

$$\varepsilon(1+\log r)^2 \kappa_{pm} \leq 1$$

alors le calcul de C via l'algorithme 4.5 donne la matrice approchée $\tilde{C}$ vérifiant

$$(\forall i = 1:m, j = 1:n) \quad |\tilde{C}_{ij} - C_{ij}| \leq (2\varepsilon + \varepsilon^2)|C_{ij}|$$

4.6 Calcul précis des récurrences linéaires

Nous considérons dans cette section le problème de calculer les récurrences linéaires du type suivant

$$z_k = a_k \times z_{k-1} + b_k \ ; \ k = 1:N \tag{4.17}$$

où z_0 et les a_k et b_k $(k = 1:N)$ sont donnés et appartiennent à $\mathbb{F}_\beta(p, e_{\min}, e_{\max})$. Il s'agit d'un problème simple auquel, du point de vue algorithmique, on peut peut proposer différentes

méthodes. La plus directe est bien sûr l'approche séquentielle.

Algorithme 4.9.

Calcul direct des z_k donnés par (4.7)

1. $z_0 = z$
2. Pour $k = 1 : N$
3. $z_k = a_k \times z_{k-1} + b_k$

Complexité: N $(\times)$ et N $(+)$

On peut également citer la méthode de réduction cyclique qui repose sur l'observation que

$$\begin{aligned} z_{2k} &= (a_{2k}a_{2k-1})z_{2(k-1)} + a_{2k}b_{2k-1} + b_{2k} \\ z_{2k-1} &= a_{2k-1}z_{2(k-1)} + b_{2k-1};\ k = 1 : \frac{N}{2} \end{aligned} \tag{4.18}$$

On suppose ici que $N = 2^m$ est une puissance de 2 et par récurrence que le problème des récurrences (4.7) soit réglé pour $\frac{N}{2} = 2^{m-1}$ de sorte que les z_{2k} soient de cette façon délivrés. Les z_{2k+1} sont ensuite obtenus par un calcul direct. Cette approche utilisée dans plusieurs contextes, entre autres la réalisation de l'addition binaire, a un avantage sur la méthode séquentielle (Algorithme 4.9) si elle est réalisée en parallèle. En effet appelons MA le temps pour réaliser l'accumulation $d = c + a \times b$ et $T(m)$ le temps utilisé pour réaliser en parallèle (4.18). On voit tout de suite que si les z_{2k} sont déjà calculés alors les

$$z_{2k-1} = a_{2k-1}z_{2(k-1)} + b_{2k-1};\ k = 1 : \frac{N}{2}$$

sont calculés en parallèle en utilisant $1MA$ et 2^{m-1} processeurs (accumulateurs). Les z_{2k} sont calculés d'une manière récurrente en utilisant donc une temps de calcul parallèle qui vaut

$T(m-1)$. On aura alors

$$\begin{aligned} T(m) &= T(m-1)+1 \\ &= mMA \\ &= (\log N)MA \end{aligned}$$

Si $P(m)$ désigne le nombre de processeurs utilisés à cet effet, on aura de même

$$\begin{aligned} P(m) &= P(m-1)+2^{m-1} \\ &= \sum_{j=1}^{m} 2^{j-1} = N-1 \text{ processeurs} \end{aligned}$$

Toujours dans le même cadre algorithmique, il est important d'observer que (4.17) revient à un simple problème de résolution d'un système bidiagonal inférieur si bien que les méthodes de résolution des systèmes triangulaires puissent être appliquées dans notre situation. Ainsi l'algorithme 4.9 peut être vu comme la méthode des éliminations successives. D'une manière plus formelle, on s'aperçoit aisément que le problème de réaliser (4.17) est équivalent au système bidiagonal $(n+1)\times(n+1)$ suivant

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ -a_1 & 1 & 0 & \cdots & 0 \\ 0 & -a_2 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & -a_n & 1 \end{bmatrix} \begin{bmatrix} z_0 \\ z_1 \\ \vdots \\ z_N \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_N \end{bmatrix} \tag{4.19}$$

Dans [30], [29], Higham a mené une étude de synthèse des méthodes de résolution des systèmes triangulaires et a analyser l'effet de la propagation des erreurs d'arrondi de chacune de ces méthodes (voir aussi [43]). Parmi les méthodes décrites, on constate en particulier la méthode des éliminations successives et la méthode parallèle "fan-in". Concernant l'analyse de l'erreur, on peut conclure que si $\hat{x}$ désigne la valeur approchée de la solution x du système triangulaire inférieur

$$Lx = b$$

obtenue via la méthode parallèle "fan-in" alors

$$||b - L\hat{x}||_\infty \leq d_n\varepsilon||\ |L||L^{-1}||L||L^{-1}||L||x|||_\infty + \mathrm{O}(\varepsilon^2)$$

On peut également exprimer les récurrences (4.17) comme produit d'un nombre fini des matrices du type 2×2 qui pour un numéricien représente une importante formulation (voir [30]). En effet, on a le résultat suivant dont la preuve découle d'une récurrence immédiate.

Théorème 4.5. On a

$$\begin{pmatrix} 1 \\ z_j \end{pmatrix} = \begin{bmatrix} 1 & 0 \\ b_j & a_j \end{bmatrix} \begin{bmatrix} 1 & 0 \\ b_{j-1} & a_{j-1} \end{bmatrix} \cdots \begin{bmatrix} 1 & 0 \\ b_1 & a_1 \end{bmatrix} \begin{pmatrix} 1 \\ z_0 \end{pmatrix} \tag{4.20}$$

□

Cette formulation nous permet de construire le conditionnement du problème (4.17) en perturbant quelque peu les données. En posant

$$M_j = \begin{bmatrix} 1 & 0 \\ b_j & a_j \end{bmatrix}$$

on peut observer que le conditionnement κ_{rl} vérifie

$$\kappa_{rl} = \max_{j=1:N} \frac{|||M_j|.|M_{j-1}| \cdots |M_1|\binom{1}{|z_0|}||_\infty}{||M_j.M_{j-1} \cdots M_1\binom{1}{z_0}||_\infty}$$

Ainsi si $\hat{z}$ désigne la valeur approchée de z en utilisant la méthode de réduction cyclique (4.18), on montre que

$$\hat{z}_j = z_j - e_j \ / \ |e_j| \leq (\log N)\varepsilon\kappa_{rl}|z_j| \quad (\forall j = 1:N)$$

Il suffit pour celà d'utiliser les méthodes standards notamment le lemme 4.7 présenté dans [30, p.81]. De même si $\hat{z}$ désigne la valeur approchée de z en utilisant la méthode directe, on montre que

$$[illegible] \quad (\forall j = 1:N) \tag{4.21}$$

Comme l'objet est de développer des méthodes de correction, notre quête consiste alors de déterminer; ne serait ce que approximativement; les erreurs e_j définies par (4.21) et les rajouter ensuite aux $\hat{z}_j$ en vue d'obtenir une amélioration. Pour ce faire, on remarque que si l'on applique la méthode directe

$$\hat{z}_j = a_j \hat{z}_{j-1} + b_j - r_j$$

où r_j est l'erreur provenu du calcul de $a_j \hat{z}_{j-1} + b_j$. On en déduit directement que

$$e_j = a_j e_{j-1} + r_j$$

Malheureusement on ne peut pas déterminer d'une manière exacte l'erreur r_j. On ne peut obtenir qu'une valeur approximative $\hat{r}_j$ telle que $r_j = \hat{r}_j + O(\varepsilon^2)$ et ce en utilisant l'addition flottante composée et la transformation (4.14) de $\times$ à $+$. Par conséquent, on ne peut pas calculer exactement les e_j et nous n'avons dans un premier temps que des valeurs approchées $\tilde{e}_j$ telles que

$$\tilde{e}_j = a_j \tilde{e}_{j-1} + \hat{r}_j$$

Puisque $e_0 = 0$, $r_j = \hat{r}_j + O(\varepsilon^2)$ on vérifie (avec une constante multiplicative plus grande) que

$$e_j = \tilde{e}_j + O(\varepsilon^2) \tag{4.23}$$

D'autre part, l'application parallèle de la méthode de réduction cyclique aux récurrences (4.22) donne des valeurs approchées $\hat{e}_j$ de $\tilde{e}_j$ telles que

$$\begin{aligned} |\hat{e}_j - \tilde{e}_j| &\leq (\log N)\varepsilon\kappa_{rl}|\tilde{e}_j| \\ &< (\log N)\varepsilon\kappa_{rl}|e_j| + O(\varepsilon^3) \end{aligned}$$

en utilisant (4.23). par conséquent

$$\begin{aligned} |\hat{e}_j - e_j| &\leq |\hat{e}_j - \tilde{e}_j| + |e_j - \tilde{e}_j| \\ &\leq (\log N)\varepsilon\kappa_{rl}|e_j| + \mathrm{O}(\varepsilon^3) + \mathrm{O}(\varepsilon^2) \\ &\leq (\log N)^2\varepsilon^2\kappa_{rl}^2|z_j| + \mathrm{O}(\varepsilon^3) + \mathrm{O}(\varepsilon^2) \\ &\quad + \mathrm{O}(\varepsilon^2) \end{aligned}$$

Enfin les valeurs approchées $\hat{z}_j$ de z_j obtenues en appliquant la méthode de réduction cyclique sont alors corrigées et donc améliorées par les $\tilde{z}_j$ telles que

$$\tilde{z}_j = fl(\hat{z}_j + \hat{e}_j)$$

D'après (4.21)

$$\begin{aligned} \hat{z}_j + \hat{e}_j &= z_j - e_j + \hat{e}_j \\ &= z_j + \mathrm{O}(\varepsilon^2) \end{aligned}$$

On en déduit aussitôt que

$$|\tilde{z}_j - z_j| \leq \varepsilon|z_j| + \mathrm{O}(\varepsilon^2) \tag{4.24}$$

On vient d'exposer une méthode parallèle réalisant les récurrences (4.17) en utilisant N processeurs et

$$(N + \log N)\ MA$$

L'avantage par rapport aux autres algorithmes plus performants du point de vue complexité c'est que; moyennant des conditions raisonnables similaires à (4.12); notre méthode délivre des valeurs approchées $\tilde{z}_j$ des solutions z_j telles que

$$|\tilde{z}_j - z_j| \leq 2\varepsilon|z_j|$$

Le problème des récurrences linéaires est d'un intérêt pratique et théorique indéniable.

Parmi les applications, on peut citer par exemple le problème de l'évaluation d'un polynôme

$$P(x) = a_1 + a_2 x + \cdots + a_N x^{N-1}$$

en un point x. Le conditionnement de ce problème est

$$\kappa_{cond} \quad \frac{|a_1| + |a_2 x| + \cdots + |a_N x^{N-1}|}{|a_1 + a_2 x + \cdots + a_N x^{N-1}|}$$

Le schéma de Horner

$$(a_1 + x(a_2 + x(a_3 + \cdots + x(a_{N-1} + x(a_N)) \cdots)))$$

est sans doute la méthode qui prévaut le plus dans ce cas et qu'on peut expliquer en termes de récurrences linéaires comme suit:

$$w_0 = a_N \ , \ w_j = a_{N-j} + x w_{j-1}$$

et $P(x) = w_{N-1}$. La méthode parallèle présentée peut être donc appliquée au schéma de Horner pour donner une valeur approchée $\hat{P}$ de $P(x)$ telle que

$$|P(x) - \hat{P}| \leq \varepsilon |P(x)| + O(\varepsilon^2) \tag{4.25}$$

En ce qui concerne la méthode parallèle proposée, on observe que le caractère parallèle intervient lors de la détermination des erreurs (4.23) et que les z_j sont calculés d'une manière séquentielle. Il s'agit plutôt d'un algorithme "séquentiel-parallèle" plus que d'un algorithme parallèle. Un algorithme parallèle "pur" coûterait $O(\log N)$ opérations. Nous terminons en expliquant pourquoi l'approche de correction n'est pas applicable à ce type d'algorithmes.

Considérons d'abord le problème de calculer la version perturbée

$$\tilde{s} = \tilde{x}_1 + \cdots + \tilde{x}_N \tag{4.26}$$

du problème de calculer (4.1) où

$$|\tilde{x}_k - x_k| \leq \varepsilon$$

et supposons que l'algorithme de sommation est réalisé sans aucune erreur d'arrondi. C'est-à-dire que l'erreur finale est égale à 0 et le résultat est $\tilde{s}$. Malheureusement, en la comparant à la solution à s on constate que

$$|\tilde{s} - s| \leq \varepsilon \kappa_S |s| \tag{4.27}$$

Pour comprendre la méthode parallèle pure appliquée à (4.17), il est bien de formuler polynômialement les récurrences linéaires de la façon suivante

$$y_j = a_j \cdots a_1 b_1 + a_j \cdots a_2 b_2 + \cdots a_j b_j + b_{j+1}$$

La méthode parallèle consiste alors à calculer d'abord tous les termes $a_j \cdots a_k b_k$ en parallèle et réaliser en suite la sommation obtenue en parallèle à l'instar de (4.4). Néanmoins on obtient un problème perturbé de sommation qui à cause de l'observation et la conclusion (4.27), l'application de la méthode de correction (et même exacte!) importe peu.

On conclut que, si l'on tient compte dans la métrique de la performance des algorithmes numériques de l'effet des erreurs d'arrondi, la méthode parallèle proposée ici est la plus performante.

Chapter 5

Méthode de l'Amélioration Itérative

D'une manière informelle, on entend par méthode de correction tout procédé permettant de déterminer au mieux les erreurs d'une certaine méthode provoquées par les arrondis et les rajouter ensuite aux résultats approchés. De cette façon on souhaite obtenir une amélioration. A la lumière de cette définition, on observe que toute méthode de correction est une méthode itérative et on est tenté de croire que l'inverse est encore vraie. Hélas, une méthode itérative basée sur la construction d'une itération

$$x_0 \text{ donné, } x_n = \Phi(x_{n-1})$$

peut théoriquement converger vers une solution a mais numériquement se transforme à une suite stationnaire de sorte que la méthode ne puisse pas améliorer la solution approchée obtenue. Considérons à titre d'exemple la suite suivante (voir [30])

$$x_1 = 1,\ x_n = x_{n-1} + \frac{1}{n^2}$$

qui converge vers $\frac{\pi^2}{6}$. Comme $x_n \nearrow +\infty$ et $\frac{1}{n^2} \searrow 0$, on en déduit que cette suite est numériquement stationnaire puisque si la valeur approchée $\hat{x}_{n-1}$ de x_{n-1} est ≥ 1 et $\frac{1}{n^2} < \varepsilon$, on aura

$$\hat{x}_n = fl(\hat{x}_{n-1} + \frac{1}{n^2}) = \hat{x}_{n-1}$$

Dans ce chapitre nous abordons cet aspect en considérant d'un côté la méthode de l'amélioration itérative et de l'autre la méthode de Newton-Raphson en montrant qu'une démarche de correction informelle conduit souvent à la méthode de Newton-Raphson.

5.1 Méthode de correction et méthode de Newton-Raphson

Typiquement, une méthode de correction suppose l'existence d'une méthode réalisant un problème numérique. Pour simplifier, Supposons que notre problème consiste à déterminer un nombre réel a tel que

$$f(a) = y$$

où f désigne une fonction réelle de classe C^∞. A cause des erreurs d'arrondi, la méthode ne donne qu'une valeur approchée $\hat{a}$ de a. D'après l'esprit de l'approche de correction on est amené à déterminer dans la mesure du possible l'erreur

$$e = a - \hat{a} \tag{5.1}$$

Une idée repose sur le théorème des accroissements finis qui permet d'affirmer qu'il existe $c \in (a, \hat{a})$ tel que

$$f(a) - f(\hat{a}) = f'(c)(a - \hat{a})$$

si bien que

$$e = \frac{y - f(\hat{a})}{f'(c)}$$

Toutefois c est inconnu et donc, on ne peut pas évaluer f' en c. Pour pallier à cet obstacle, il est raisonnable de supposer que $c \simeq \hat{a}$ par suite on obtient au lieu de e sa valeur approchée

$$\tilde{e} = \frac{y - f(\hat{a})}{f'(\hat{a})}$$

On corrige alors $\hat{a}$ en posant

$$a_{\text{corrigé}} = \hat{a} + \frac{y - f(\hat{a})}{f'(\hat{a})} \tag{5.2}$$

On s'aperçoit qu'on a corrigé $\hat{a}$ via la méthode de Newton. En effet la méthode de Newton appliquée à l'équation

$$f(x) - y = 0$$

repose sur la suite récurrente suivante

$$x_0 \text{ donné, } x_{n+1} = x_n - \frac{f(x_n) - y}{f'(x_n)} \tag{5.3}$$

Autrement dit x_{n+1} corrige x_n. On conclut alors que la méthode de Newton est une méthode de correction.

Considérons le problème de calculer l'inverse A^{-1} d'une matrice régulière A du type $n \times n$. Il est connu que la méthode de Newton pour calculer cet inverse repose sur les itérations suivantes

$$X_0 \in \mathbb{R}^{n \times n} \text{ donnée, } X_{k+1} = 2X_k - X_k A X_k \tag{5.4}$$

Nous allons montrer qu'une démarche de correction peut mener à la même formule. Supposons que nous disposons d'une méthode permettant de calculer l'inverse $J = A^{-1}$ de A et que; à cause des erreurs d'arrondi; on n'obtient qu'une solution approchée $\hat{J}$ telle que

$$A^{-1} = J = \hat{J} + \mathrm{O}(\varepsilon)$$

Comme précisé précédemment, la cible dans les méthodes de correction consiste à déterminer l'erreur

$$E = A^{-1} - \hat{J}$$

qu'on peut écrire encore

$$E = A^{-1}(I_n - A\hat{J})$$
$$= E(I_n - A\hat{J}) + \mathrm{O}(\varepsilon)(I_n - A\hat{J})$$

On a

$$(I_n - A\hat{J}) = AE = \mathrm{O}(\varepsilon)$$

de sorte que

$$\begin{aligned} E &= \hat{J}(I_n - A\hat{J}) + \mathrm{O}(\varepsilon^2) \\ &\hat{J} - \hat{J}A\hat{J} + \mathrm{O}(\varepsilon^2) \end{aligned}$$

Par conséquent on ne peut déterminer qu'une valeur approchée

$$\hat{E} = \hat{J} - \hat{J}A\hat{J}$$

de E et enfin $\hat{J}$ est corrigée de la façon suivante

$$\begin{aligned} \hat{J}_{\text{corrigé}} &= \hat{J} + \hat{E} \\ &= 2\hat{J} - \hat{J}A\hat{J} \end{aligned}$$

qui est identique à l'itération de Newton (5.4). Là encore, on renforce la thèse qui dit que la méthode de Newton est une méthode de correction.

On observe ici que

$$\begin{aligned} \hat{J}_{\text{corrigé}} &= \hat{J} + \hat{E} \\ &= \hat{J} + E + \mathrm{O}(\varepsilon^2) \\ &= A^{-1} + \mathrm{O}(\varepsilon^2) \end{aligned} \tag{5.5}$$

Bien que cette analyse indique une amélioration dans la correction, il est à signaler toutefois qu'il est supposé que le produit matriciel $\hat{J}A\hat{J}$ est calculé exactement; c'est-à-dire sans erreurs d'arrondi; et par conséquent l'analyse est à appuyer par des hypothèses supplémentaires. C'est ce que nous allons discuter plus loin.

5.2 Méthode de l'amélioration itérative

Nous considérons ici le problème de résolution d'un système linéaire

$$A.x = b \tag{5.6}$$

où $A \in \mathbb{R}^{n\times n}$ est une matrice régulière du type $n \times n$ et $b \in \mathbb{R}^n$ un vecteur d'ordre n. Pour résoudre ce système linéaire; c'est-à-dire déterminer le vecteur $x \in \mathbb{R}^n$ vérifiant l'équation (5.6), on connait de nombreuses méthodes directes. La plus fameuse d'entre elles est sans doute la méthode de Gauss avec Pivot ou bien, d'une manière équivalente, la factorisation PLU. On peut présenter cette méthode de différentes façons parmi lesquelles l'écriture de A sous la forme suivante

$$A = P.L.U \tag{5.7}$$

où $P \in \mathbb{R}^{n\times n}$ est une matrice de permutation, $L \in \mathbb{R}^{n\times n}$ est une matrice triangulaire inférieure avec $L_{ii} = 1$ et $U \in \mathbb{R}^{n\times n}$ est une matrice triangulaire supérieure régulière. A l'aide de cette factorisation, la résolution du système (5.6) devient immédiate. En effet, il suffit de procéder comme suit

1. Résoudre le système triangulaire

$$L.y = P^T.b$$

2. Résoudre le système triangulaire

$$U.x = y$$

Il est bien connu que la résolution du système linéaire (5.6) en utilisant la méthode de Gauss avec Pivot demande

$$\frac{n^3}{3} + \mathrm{O}(n^2) = \mathrm{O}(n^3)$$

opérations flottantes. Concernant l'analyse de l'erreur, on sait que; à cause des erreurs d'arrondi;

la méthode ne donne qu'une valeur approchée $\tilde{x} \in \mathbb{R}^n$ du vecteur solution x de (5.6) tel que

$$(A + \Delta A).\tilde{x} = b, \quad |\Delta A| \leq c_n \varepsilon P.|\hat{L}|.|\hat{U}| \tag{5.8}$$

où $\hat{L}$ et $\hat{U}$ sont respectivement les valeurs approchées de L et U. Dans ces estimations, on suppose pour des questions de commodité et sans perdre en généralité que $|\hat{L}|.|\hat{U}| = |L|.|U|$ et $P = I_n$. Ceci étant, on déduit directement de l'analyse backward (5.8) l'estimation forward suivante

$$|x - \tilde{x}| \leq c_n \varepsilon |A^{-1}|.|L|.|U|..|x| \tag{5.9}$$

où c_n est une constante ne dépendant que polynômialement de n. Notre objectif consiste à améliorer la précision de la solution approchée $\tilde{x}$. Pour celà, on procède comme suit.

Etape 1: On calcule d'abord

$$d = b - A.\tilde{x}$$

Le nombre d'opérations pour réaliser cette étape est: n^2 opérations. Appelons $\tilde{d}$ la valeur approchée de d. Si les calculs sont effectués en double précision, on aura

$$d - \tilde{d} : O(\varepsilon^2)$$

S'ils sont réalisés en simple précision, on aura

$$|d - \tilde{d}| \leq n\varepsilon(|b| + |A|.|\tilde{x}|)$$

Etape 2 : On résoud le système linéaire

$$A.r = \tilde{d}$$

en exploitant la factorisation LU (déjà réalisée) de A. Le nombre d'opérations exigé par cette étape est également n^2 opérations. Appelons $\tilde{r}$ la valeur approchée de r. En double précision, on a

$$r - \tilde{r} : O(\varepsilon^2)$$

et en simple précision, on obtient d'après la majoration (5.9)

$$\begin{aligned} |r - \tilde{r}| &\leq c_n\varepsilon|A^{-1}|.|L|.|U|.|r| \\ &\leq c_n\varepsilon|A^{-1}|.|L|.|U|.|A^{-1}|.|\tilde{d}| \\ &\leq c_n\varepsilon|A^{-1}|.|L|.|U|.|A^{-1}|.|A|.|x - \tilde{x}| + \mathrm{O}(\varepsilon^2) \\ &\leq \mathrm{O}(\varepsilon^2) \end{aligned}$$

Etape 3 : On corrige $\tilde{x}$ comme suit:

$$\hat{x} = \tilde{x} + \tilde{r}$$

En double précision, on a :

$$\begin{aligned} \hat{x} &= \tilde{x} + \tilde{r} \\ &= \tilde{x} + r + \mathrm{O}(\varepsilon^2) \\ &= \tilde{x} + A^{-1}\tilde{d} + \mathrm{O}(\varepsilon^2) \\ &= \tilde{x} + A^{-1}d + \mathrm{O}(\varepsilon^2) \\ &= x + \mathrm{O}(\varepsilon^2) \end{aligned}$$

et en simple précision, on a

$$\hat{x} = x + A^{-1}(\tilde{d} - d) + \mathrm{O}(\varepsilon^2)$$

Par conséquent

$$\begin{aligned} |\hat{x} - x| &\leq n\varepsilon|A^{-1}|.(|b| + |A|.|\tilde{x}|) + \mathrm{O}(\varepsilon^2) \\ &\leq c_n\varepsilon|A^{-1}|.(|b| + |A|.|x|) + \mathrm{O}(\varepsilon^2) \end{aligned}$$

et

$$|\hat{x} - x| \leq c_n\varepsilon|A^{-1}|.|A|.|x| + \mathrm{O}(\varepsilon^2) \tag{5.10}$$

On vient d'exposer la méthode de l'amélioration itérative en double précision et en simple préci-

sion. En double précision, on déduit que la méthode est numériquement stable en composantes et

$$|\tilde{x} - x| \leq c_n \varepsilon |x| + O(\varepsilon^2) \tag{5.11}$$

En simple précision, la correction $\hat{x}$ vérifie (4.10) et prouve que la méthode assure une stabilité numérique forward en composantes.

L'amélioration itérative est donc une méthode nécessaire puisque d'un côté elle demande $2n^2+O(n)$ opérations seulement qui est réduit par rapport au nombre total de $\frac{n^3}{3}+O(n^2)$ requis par la méthode de Gauss; c'est d'ailleurs la raison pour laquelle on a toléré la réalisation en double précision, si nécessaire, de l'amélioration itérative. De l'autre côté, la méthode de l'amélioration itérative assure la stabilité numérique forward (5.10) si elle est réalisée en simple précision et une stabilité numérique quasi-idéale (5.11) si elle est réalisée en double précision. Il est bien de signalé que cette stabilité numérique est assurée à condition que la constante multiplicative dissimulée dans $O(\varepsilon^2)$ dans (5.10) et (5.11) ne soit éxagérément grande. En conclusion la méthode de l'amélioration itérative est très importante, à telle enseigne qu'on recommande son enseignement en graduation.

Avant de clore ce paragraphe, il importe de mentionner qu'il existe une bibliographie (ancienne et récente) abondante sur l'approche de l'amélioration itérative exposée en premier par Wilkinson(voir [49], [50]). L'amélioration itérative appliquée aux moindres carrée est étudiée dans [5], [6] (voir [26]). L'analyse menée ici s'est inspirée de [33], [44] (voir aussi [30]). Dans [27], Higham a montré que l'application de l'amélioration itérative dans le cas de la décomposition QR de la matrice conduit également à la stabilité numérique (voir [30]).

5.3 Méthodes de correction pour les systèmes Toeplitz

Nous considérons; comme dans la section précédente; le problème de résolution des systèmes linéaires du type

$$M.a = b \tag{5.12}$$

où cette fois ci $M \in \mathbb{R}^{n\times n}$ est une matrice de Toeplitz; c'est-à-dire que

$$M = \begin{bmatrix} r_0 & r_1 & r_2 & \cdots & r_{n-1} \\ s_1 & r_0 & r_1 & \ddots & \vdots \\ s_2 & s_1 & r_0 & \ddots & r_2 \\ \vdots & \ddots & \ddots & \ddots & r_1 \\ s_{n-1} & \cdots & s_2 & s_1 & r_0 \end{bmatrix}$$

L'objet principal est de proposer des méthodes de correction pour améliorer les solutions approchées du système linéaire (5.12); avec M Toeplitz; obtenues par une certaine méthode. Afin de suggérer des méthodes de correction qui sont en adéquation avec le caractère spécial des matrices Toeplitz, nous jugeons utile de commencer, en guise de rappel, par étudier ces matrices du point de vue algébrique, algorithmique et numérique.

Supposons que

$$M = \begin{bmatrix} 1 & 2 & -1 \\ 2 & 1 & 2 \\ -1 & 2 & 1 \end{bmatrix}$$

Alors

$$M^2 = \begin{bmatrix} 6 & 2 & 2 \\ 2 & 9 & 2 \\ 2 & 2 & 6 \end{bmatrix} \text{ et } M^{-1} = \begin{bmatrix} \frac{3}{16} & \frac{1}{4} & -\frac{5}{16} \\ \frac{1}{4} & 0 & \frac{1}{4} \\ -\frac{5}{16} & \frac{1}{4} & \frac{3}{16} \end{bmatrix}$$

On en déduit que si la somme de deux matrices Toeplitz est une matrice Toeplitz, il n'en est pas de même pour la multiplication de deux matrices Toeplitz ou l'inverse d'une matrice Toeplitz régulière. Il existe heureusement un cadre approprié dans lequel on peut voir que ces opérations sont stables. Avant de construire ce cadre rappelons, pour des raisons qui vont être apparentes après, quelques aspects numériques de la transformation discrète de Fourier et des matrices circulantes.

Transformation Discrète de Fourier et FFT .

On entend par transformation discrète de Fourier (TDF) le produit matrice-vecteur $y = F_n x$

où

$$F_n = (e^{-2\pi i \frac{kj}{n}})_{k,j=0:(n-1)}, \qquad (i^2 = -1)$$

La FFT (Fast Fourier Transform), c'est-à-dire la transformation rapide de Fourier, est une approche permettant de calculer y en utilisant juste $O(n \log n)$ opérations. Ce qui constitue en complexité une réduction très importante.

Théorème 5.1 Supposons que $n = 2^q$. Alors F_n peut être décomposée comme suit

$$F_n = A_q \cdots A_1 P_n \tag{5.13}$$

où P_n est une matrice de permutation et $A_k = I_{2^{q-k}} \otimes B_{2^k}$

$$B_{2^r} = \begin{pmatrix} I_r & \Omega_r \\ I_r & -\Omega_r \end{pmatrix},$$

$$\Omega_r = diag(1, \omega_k, \cdots, \omega_k^{r-1}),\ \omega_k = e^{-\frac{2\pi i}{2^k}}$$

Démonstration. Voir VanLoan [47]. □

Ici $A \otimes B = (a_{ij}B)$ désigne le produit de Kronecker. Grâce à cette décomposition de F_n en terme de produit, on peut montrer que la FFT $y = F_n x$ est numériquement stable. Plus précisément, si $\hat{y}$ désigne la valeur approchée de $y = F_n x$ calculée via (5.13), on montre que (voir [30])

$$||\hat{y} - y||_2 \leq c_n \varepsilon ||y||_2 \tag{5.14}$$

où c_n est une constante de l'ordre de $n \log n$. Comme l'inverse F_n^{-1} de F_n vérifie

$$F_n^{-1} = \frac{1}{n} F_n^H = \frac{1}{n} (e^{2\pi i \frac{kj}{n}})_{k,j=0:(n-1)}$$

on déduit que la transformation inverse $x = F_n^{-1} y$ est également une TDF et peut être réalisée par la FFT, c'est-à-dire par la décomposition (5.13) à ceci près que ω_k est à remplacer par ω_k^{-1}. Numériquement si $\hat{x}$ désigne la valeur approchée de $x = F_n^{-1} y$ calculée via la FFT, on montre

que

$$||\hat{x} - x||_2 \leq c_n \varepsilon ||x||_2 \tag{5.15}$$

Matrices circulantes.

On appelle matrice circulante $C(w)$ générée par un vecteur $w = (w_1, \cdots, w_n)^T$ la matrice ayant la forme suivante

$$C(w) = \begin{bmatrix} w_1 & w_n & \cdots & w_2 \\ w_2 & w_1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & w_n \\ w_n & \cdots & w_2 & w_1 \end{bmatrix}$$

On dit que $C^-(w)$ est une matrice anti-circulante générée par w si

$$C^-(w) = \begin{bmatrix} w_1 & -w_n & \cdots & -w_2 \\ w_2 & w_1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & w_n \\ w_n & \cdots & w_2 & w_1 \end{bmatrix}$$

Les matrices circulantes ont la propriété importante suivante

Théorème 5.2 On a

$$F_n C(w) F_n^{-1} = D = diag(g_1, \cdots, g_n)$$

où

$$g = F_n w$$

Démonstration. Voir [23], [26]. □

Cette diagonalisation des matrices circulantes permet de réaliser des calculs rapides qui sont numériquement stables. Parmi les calculs rapides qu'on peut réaliser, on cite notamment la résolution d'un système linéaire du type $C(w)x = b$ dont l'analyse backward et forward en normes a été effectuée par Higham [30] et le produit $y = C(w)x$ d'une matrice circulante $C(w)$

et un vecteur x que nous analysons ci après. Pour réaliser ce produit on utilise l'algorithme suivant.

Algorithme 5.1

Cet algorithme réalise le produit $y = C(w)x$ de la matrice circulante $C(w)$ par x.

1. Calculer en utilisant (4.13) (c'est-à-dire la FFT) $g = F_n w$
2. Calculer en utilisant (4.13) (c'est-à-dire la FFT) $h = F_n x$
3. Calculer $z = D.h$ où $D = diag(g_1, \cdots, g_n)$
4. Calculer en utilisant (4.13) (c'est-à-dire la FFT) $y = F_n^{-1} z$

Complexité: $O(n \log n)$ opérations

Dans notre analyse "forward" en normes, on va exploiter essentiellement les estimations (5.14) et (5.15). On a

$$\begin{aligned} ||\hat{g} - g|| &\leq c_n \varepsilon ||g||_2 \\ ||\hat{h} - h|| &\leq c_n \varepsilon ||h||_2 \end{aligned}$$

où $\hat{p}$ désigne toujours la valeur calculée par l'algorithme 5.1 de p et c_n désigne une constante générique qui peut changer de valeurs le long de notre analyse. Posons

$$\hat{D} = diag(\hat{g}_1, \cdots, \hat{g}_n)$$

On a aussi

$$\begin{aligned} ||\hat{z} - \hat{D}.h|| &\leq \varepsilon ||\hat{D}.h||_2 \\ ||\hat{y} - F_n^{-1}.\hat{z}|| &\leq c_n \varepsilon ||F_n^{-1}.\hat{z}||_2 \end{aligned}$$

Observons que

$$\begin{aligned} ||D.h - \hat{D}.\hat{h}|| &= ||D(h - \hat{h}) + (D - \hat{D})\hat{h}||_2 \\ &\leq ||D||_2 ||h||_2 + ||D - \hat{D}||_2 ||\hat{h}||_2 \end{aligned}$$

On a

$$\begin{aligned} ||\hat{h}||_2 &\leq ||\hat{h} - h||_2 + ||h||_2 \\ &\leq c_n\varepsilon||h||_2 + ||h||_2 \\ &\leq (1 + c_n\varepsilon)||h||_2 \end{aligned}$$

Par ailleurs

$$\begin{aligned} ||D - \hat{D}||_2 &= ||g - \hat{g}||_\infty \\ &\leq ||g - \hat{g}||_2 \\ &\leq c_n\varepsilon||g||_2 \\ &\leq c_n\varepsilon||D||_2 \end{aligned}$$

Il s'ensuit que

$$||D.h - \hat{D}.\hat{h}||_2 \leq c_n\varepsilon||D||_2||h||_2 \tag{5.16}$$

On en déduit que

$$\begin{aligned} ||z - \hat{z}|| &\leq ||z - \hat{D}.\hat{h}||_2 + ||D.h - \hat{D}.\hat{h}||_2 \\ &\leq c_n\varepsilon||\hat{D}.\hat{h}||_2 + c_n\varepsilon||D||_2||h||_2 \end{aligned}$$

Mais d'après (5.16), $||\hat{D}.\hat{h}||_2 \leq (1 + c_n\varepsilon)||D||_2||h||_2$ de sorte que

$$\begin{aligned} ||\hat{z} - z|| &\leq c_n\varepsilon||D||_2||h||_2 \\ &c_n\varepsilon||C(w)||_2||h||_2 \end{aligned}$$

puisque $||C(w)||_2 = ||D||_2$. Puisque F_n est orthogonale et même presque unitaire, on a

$$||F_n^{-1}\hat{z} - F_n^{-1}z||_2 \leq c_n\varepsilon||C(w)||_2||h||_2$$

Par conséquent

$$\begin{aligned} ||\hat{y} - y||_2 &\leq ||\hat{y} - F_n^{-1}\hat{z}||_2 + ||F_n^{-1}\hat{z} - F_n^{-1}z||_2 \\ &\leq c_n\varepsilon||F_n^{-1}\hat{z}||_2 + c_n\varepsilon||C(w)||_2||h||_2 \\ &\leq c_n\varepsilon||C(w)||_2||h||_2 \end{aligned}$$

Comme $h = F_n x$, on obtient enfin $||h||_2 \leq n||x||_2$ et

$$||\hat{y} - y||_2 \leq c_n\varepsilon||C(w)||_2||x||_2 \tag{5.17}$$

Cette estimation montre bien la stabilité numérique forward de l'algorithme 4.1. Remarquons que la décomposition du théorème 4.1 permet de déduire que

$$F_n C(w)^{-1} F_n^{-1} = D^{-1} = [diag(g_1, \cdots, g_n)]^{-1}$$

où

$$g = F_n w$$

si bien que la résolution du système linéaire $C(w).x = y$; équivalent à $x = C(w)^{-1}.y$; via FFT donne comme solution une valeur approchée $\hat{x}$ de x telle que

$$||\hat{x} - x||_2 \leq c_n\varepsilon||C(w)^{-1}||_2.||C(w).x||_2 \tag{5.18}$$

Produit matrice Toeplitz par vecteur.

Pour calculer le produit $y = Mx$ d'une matrice Toeplitz

$$M = \begin{bmatrix} r_0 & r_1 & r_2 & \cdots & r_{n-1} \\ s_1 & r_0 & r_1 & \ddots & \vdots \\ s_2 & s_1 & r_0 & \ddots & r_2 \\ \vdots & \ddots & \ddots & \ddots & r_1 \\ s_{n-1} & \cdots & s_2 & s_1 & r_0 \end{bmatrix}$$

et d'un vecteur $x \in \mathbb{R}^n$, on peut proposer plusieurs méthodes dont la complexité est $\mathbf{O(n \log n)}$. La plus fameuse est sans doute celle qui consiste à plonger M dans une matrice circulante $C(w) \in \mathbb{R}^{(2n-1)\times(2n-1)}$ où

$$w = \left[r_0, r_1, s_2, \cdots, s_{n-1}, r_{n-1}, \cdots, r_2, r_1\right]^T$$

De cette façon, on aura le partitionnement

$$C(w) = \begin{bmatrix} M & F \\ G & L \end{bmatrix} \tag{5.19}$$

et par conséquent

$$\begin{bmatrix} y \\ * \end{bmatrix} = C(w) \begin{bmatrix} x \\ 0 \end{bmatrix} \tag{5.20}$$

Par ailleurs on sait que la norme de Frobeinus

$$||A||_F = (\sum_{i,j} |A_{ij}|^2)^{\frac{1}{2}}$$
$$\sqrt{\text{trace}(A^H A)}$$

et la norme matricielle $||A||_2$ subordonnée à la norme euclidienne sont liées par les inégalités

$$||A||_2 \leq ||A||_F \leq \sqrt{n}||A||_2$$

si bien qu'on puisse travailler, si besoin est, avec la norme de Frobeinus. Dans cette perspective, on a

$$||M||_F \leq ||C(w)||_F \leq \sqrt{(2n-1)}||M||_F$$

On en déduit que le calcul de (5.20) via FFT donne une valeur approchée $\hat{y}$ de y telle que

$$||\hat{y} - y||_2 \leq c_n \epsilon ||M||_2 ||x||_2 \tag{5.21}$$

Structures de déplacement...

Le cadre convenable pour étudier les matrices Toeplitz

$$M = \begin{bmatrix} r_0 & r_1 & r_2 & \cdots & r_{n-1} \\ s_1 & r_0 & r_1 & \ddots & \vdots \\ s_2 & s_1 & r_0 & \ddots & r_2 \\ \vdots & \ddots & \ddots & \ddots & r_1 \\ s_{n-1} & \cdots & s_2 & s_1 & r_0 \end{bmatrix}$$

c'est de les définir à partir des structures de déplacement. Ainsi la matrice M satisfait l'équation de déplacement suivante

$$Z_1 M - M Z_{-1} = c_1 w^T + v e_n^T \tag{5.22}$$

où

$$Z_{-1} = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 \\ 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 1 & 0 \end{bmatrix}$$

sont les matrices de déplacement invoquées en général dans ce contexte et

$$w^T = [s_{n-1} - r_1 \ \cdots \ s_1 - r_{n-1} \ \ t_0], \ v = \begin{bmatrix} t_0 \\ r_{n-1} + s_1 \\ \vdots \\ r_1 + s_{n-1} \end{bmatrix} \tag{5.23}$$

En post-multipliant et pré-multipliant (5.22) par M^{-1} on obtient

$$\cdots = M^{-1} c_1 w^T M^{-1} + M^{-1} v e_n^T M^{-1} \tag{5.24}$$

On observe que grâce à cette structure caratérisant entièrement M^{-1}, on peut définir M^{-1} en utilisant un stockage linéaire. En effet, on a juste besoin de déclarer la première colonne M^{-1}, la dernière ligne de M^{-1}, $w^T M^{-1}$ et $M^{-1} v$. Appelons C^- la matrice anti-circulante générée

par $M^{-1}e_1$ et

$$C = \begin{bmatrix} e_n^T M^{-1} Z_1 \\ e_n^T M^{-1} Z_1^2 \\ \vdots \\ e_n^T M^{-1} Z_1^{n-1} \\ e_n^T M^{-1} \end{bmatrix}$$

On peut montrer que $(C^-)^{-1}$ est également une matrice anti-circulante et qu'elle commute avec Z_{-1} :

$$(C^-)^{-1} Z_{-1} = Z_{-1} (C^-)^{-1}$$

De même C^{-1} est circulante et qu'elle commute avec Z_1 :

$$C^{-1} Z_1 = Z_1 C^{-1}$$

En prémultipliant (5.24) par $(C^-)^{-1}$ et la post-multipliant par C^{-1} on obtient l'équation suivante

$$(C^-)^{-1} M^{-1} C^{-1} Z_1 - Z_{-1} (C^-)^{-1} M^{-1} C^{-1} = e_1 w^T M^{-1} C^{-1} + (C^-)^{-1} M^{-1} v e_n^T$$

Par identification, on s'aperçoit que

$$(C^-)^{-1} M^{-1} C^{-1} = T$$

est une matrice Toeplitz et

$$M^{-1} = C^- T C \tag{5.25}$$

Par conséquent le produit $x = M^{-1} y$ de l'inverse de M (une fois calculé) par y peut se faire en $O(n \log n)$ opérations. De plus si $\hat{x}$ designe la solution approchée de x on montre que

$$||\hat{x} - x||_2 \leq c_n \varepsilon ||C^-||_2 ||T||_2 ||C||_2 ||x||_2$$

Il peut être intéressant de noter que (5.25) est analogue à la formule de Gohberg Somencul

et mieux encore à celle de Ammar et Gader [3]. Du point de vue numérique, nous remarquons que la formule (5.25) étant écrite sous forme de produit matriciel; ce qui n'est pas le cas dans [3] et [23]; est plus expressive d'après [30].

Résolution des systèmes Toeplitz:

Pour résoudre un système linéaire quelconque $A.x = b$, on préconise d'utiliser d'abord une décomposition de A (LU, Cholesky ou QR) et procéder ensuite à la résolution du système linéaire qui grâce à la décomposition déjà effectuée devient essentiellement triangulaire. Cette démarche est motivée par le fait que la résolution du sytème avec A décomposée coûte $O(n^2)$ alors que la décomposition coûte $O(n^3)$ opérations. Bien que menant à une même conclusion, il est déconseillé de résoudre $A.x = b$ en calculant en premier lieu l'inverse A^{-1} à cause du fait que l'inversion est plus coûteuse et numériquement plus instable que la décomposition. Cette conclusion ne devrait pas laisser entendre qu'il s'agit d'une directive générale. Lorsqu'il s'agit d'un système linéaire Toeplitz $M.x = b$, on peut proposer les deux approches suivantes:

Approche 1.

1. Réaliser la décomposition $M = LU$ de M. En général, ceci coûte $O(n^2)$ opérations.

2. Résoudre le système $L(U.x) = b$. Comme ni L ni U n'est structurée, cette résolution coûte $O(n^2)$ opérations également.

Approche 2.

1. Calculer l'inverse M^{-1} de M; ce qui demande $O(n^2)$ opérations. Bien sûr il s'agit d'une opération plus coûteuse que la décomposition.

2. Calculer $x = M^{-1}b$; ce qui, en utilisant la décomposition (4.25); coûte $O(n \log n)$ opérations.

On observe que contrairement au cas général l'approche 2 peut tout à fait être recommandée même plus recommandée que l'approche 1. Pour calculer l'inverse M^{-1} de M, de nombreuses méthodes numériques ont été proposées dans la littérature (Voir [20], [26], [24], [23]). Dans ces méthodes on peut dire sur le plan numérique, que si $\tilde{B}$ désigne la valeur approchée de M^{-1}, il

existe une constante d'instabilité $K_1(M)$ telle que

$$\|\tilde{B} - M^{-1}\|_2 \leq c_n \epsilon K_1(M) \|M^{-1}\|_2 \tag{5.26}$$

Par exemple si M est symétrique définie-positive et l'inverse M^{-1} est calculé via l'algorithme de Trench alors

$$K_1(M) = \prod_{k=1}^{n} \frac{1+|\alpha_k|}{1-|\alpha_k|}$$

où les α_k sont les constantes de reflexion intervenant dans l'algorithme de Durbin ou celui de Levinson (Voir [26]).

Le premier problème qu'on pourrait soulevé ici et auquel on a pas attaché d'importance jusque là c'est le fait que $\tilde{B}$ n'est pas forcément structurée et ne vérifie pas donc l'équation (5.24) par suite (5.25) de sorte que la performance $O(n \log n)$ stipulée par le point 2 de l'approche 2 soit remise en cause. Dans ce cas, l'approche prévalue consiste à faire comme si $\tilde{B}$ vérifie (5.24). Par conséquent on travaille au lieu de $\tilde{B}$ avec la matrice structurée B vérifiant

$$BZ_1 - Z_{-1}B = \tilde{B}e_1 w^T \tilde{B} + \tilde{B}v e_n^T \tilde{B} \tag{5.27}$$

Dans ce cas

$$B = C^{-}.T.C \tag{5.28}$$

où C^{-} est la matrice anti-circulante générée par $\tilde{B}e_1$ et

$$C = \begin{bmatrix} e_n^T \tilde{B} Z_1 \\ e_n^T \tilde{B} Z_1^2 \\ \vdots \\ e_n^T \tilde{B} Z_1^{n-1} \\ e_n^T \tilde{B} \end{bmatrix}$$

et T est la matrice de Toeplitz déduite par identification dans l'équation (5.27). Dans ces

conditions, il est bien de voir à quel point l'estimation (5.26) est altérée. Posons pour celà

$$\Phi(X) \quad YZ_1 - Z_{-1}X$$

Il est clair que Φ est une application linéaire inversible dont la matrice par rapport à la base canonique est

$$\Phi = \begin{bmatrix} Z_{-1} & 0 & \cdots & 0 & I_n \\ I_n & Z_{-1} & 0 & \cdots & 0 \\ 0 & I_n & Z_{-1} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & I_n & Z_{-1} \end{bmatrix}$$

En calculant l'inverse Φ^{-1} de Φ, soit directement ou en utilisant la formule de Sherman-Morrison [26], on s'aperçoit que la norme de Frobenius $||\Phi^{-1}||_F$ de Φ^{-1} est un polynôme en n par suite

$$||\Phi^{-1}||_2 = c_n$$

est un polynôme en n. Ceci étant, on a

$$\begin{aligned} ||B - M^{-1}||_2 &\leq ||B - M^{-1}||_F \leq ||\Phi^{-1}||_2 ||\Phi(B - M^{-1})||_F \\ &= c_n ||\Phi(B - M^{-1})||_F \leq c_n ||\Phi(B - M^{-1})||_2 \end{aligned}$$

On est alors conduit à estimer $||\Phi(B - M^{-1})||_2$. On a

$$\begin{aligned} \Phi(B - M^{-1}) &= (\tilde{B}e_1 w^T \tilde{B} + \tilde{B}ve_n^T \tilde{B}) - (M^{-1}e_1 w^T M^{-1} + M^{-1}ve_n^T M^{-1}) \\ &= (\tilde{B} - M^{-1})e_1 w^T \tilde{B} + M^{-1}e_1 w^T (\tilde{B} - M^{-1}) \\ &\quad + (\tilde{B} - M^{-1})ve_n^T \tilde{B} + M^{-1}ve_n^T (\tilde{B} - M^{-1}) \end{aligned}$$

si bien que $||\Phi(B - M^{-1})||_2$ soit $\leq$

$$(||e_1 w^T B||_2 + ||M^{-1}e_1 w^T||_2 + ||ve_n^T \tilde{B}||_2 + ||M^{-1}ve_n^T||_2)||\tilde{B} - M^{-1}||_2$$

Par conséquent en utilisant (5.26)

$$||B - M^{-1}||_2 \leq c_n \epsilon K_2(M)||M^{-1}||_2 \tag{5.29}$$

où

$$K_2(M) = K_1(M)(||c_1 w^T \tilde{B}||_2 + ||M^{-1} e_1 w^T||_2 + ||v e_n^T \tilde{B}||_2 + ||M^{-1} v e_n^T||_2)$$

Nous considérons maintenant le problème de résoudre un système linéaire

$$M.x = y$$

où M est Toeplitz et en supposant qu'on dispose de la matrice B définie par l'équation de déplacement (5.27) et vérifiant la décomposition (5.28) et l'estimation (5.29). Posons

$$\bar{x} = B.y$$

de sorte que

$$||\bar{x} - x||_2 \leq c_n \epsilon K_2(M)||M^{-1}||_2||M.x||_2$$

Pour calculer $\bar{x} = B.y$, on utilise évidemment la décomposition (5.28) afin d'atteindre la performance $O(n \log n)$, ce qui numériquement donne une solution approchée $\tilde{x}$ telle que

$$||\bar{x} - \tilde{x}||_2 \leq c_n \epsilon ||C^-||_2||T||_2||C||_2||M.x||_2$$

d'où

$$||\tilde{x} - x||_2 \leq c_n \epsilon K(M)||M.x||_2 \tag{5.30}$$

où

$$K(M) = K_2(M)||M^{-1}||_2 + ||C^-||_2||T||_2||C||_2$$

Dans la correction de la solution $\tilde{x}$ nous supposons que

$$\epsilon K(M)||M||_2 \leq c_n$$

où c_n est la constante générique qui dépend polynômialement de n. Nous proposons alors l'algorithme de correction suivant

Algorithme 5.2.

Méthode d'amélioration des systèmes Toeplitz

1. $d = y - M.\tilde{x}$
2. $\bar{r} = B.d$
3. $x' = \tilde{x} + \bar{r}$

Complexité: $O(n \log n)$ opérations.

Pour analyser l'erreur de cet algorithme, appelons respectivement $\tilde{d}$, $\tilde{r}$, $\tilde{x}$ les valeurs calculées de d, $\bar{r}$ et x', et posons $r = M^{-1}.d$ si bien que $x = \tilde{x} + r$. On a

$$||\tilde{d} - d||_2 \leq c_n \tilde{\varepsilon}(||y||_2 + ||M||_2||x||_2)$$

où $\tilde{\varepsilon} = \varepsilon$ si la ligne 1 de l'algorithme 5.2 est réalisée en précision simple et $\tilde{\varepsilon} = \varepsilon^2$ si la ligne 1 de l'algorithme 5.2 est réalisée en double précision. D'après (5.30), on a

$$||\tilde{r} - M^{-1}.\tilde{d}||_2 \leq c_n \varepsilon K(M)||\tilde{d}||_2$$

On a également

$$||r - M^{-1}.\tilde{d}||_2 \leq c_n \tilde{\varepsilon}||M^{-1}||_2(||y||_2 + ||M||_2||x||_2)$$

de sorte que

$$||\tilde{r} - r||_2 \leq c_n \varepsilon K(M)||\tilde{d}||_2 + c_n \tilde{\varepsilon}||M^{-1}||_2(||y||_2 + ||M||_2||x||_2)$$

On a

$$||\tilde{d}||_2 \leq ||\tilde{d} - d||_2 + ||d||_2$$

En conséquence

$$\begin{aligned} ||\hat{r} - r||_2 &\leq c_n\bar{\varepsilon}||M^{-1}||_2(||y||_2 + ||M||_2||x||_2) \\ &+ c_n\varepsilon K(M)\bar{\varepsilon}(||y||_2 + ||M||_2||x||_2) \\ &+ c_n\varepsilon^2 K(M)||d||_2 \end{aligned}$$

et puisque $(1+\alpha)\hat{x} - x = \hat{r} - r$ avec $|\alpha| < \varepsilon$, on obtient $||x - (1+\alpha)\hat{x}||_2 = ||\hat{r} - r||_2$ si bien que

$$\begin{aligned} ||x - (1+\alpha)\hat{x}||_2 &\leq c_n\varepsilon^2(K(M)||M||_2)^2||x||_2 + \\ &c_n\bar{\varepsilon}||M^{-1}||_2(||y||_2 + ||M||_2||x||_2) + \\ &c_n\varepsilon\bar{\varepsilon}K(M)(||y||_2 + ||M||_2||x||_2) \end{aligned}$$

On distingue deux cas. Si la ligne 1 est effectuée en double précision, on obtient

$$\begin{aligned} ||x - \hat{x}||_2 &\leq ||x - (1+\alpha)\hat{x}||_2 + \varepsilon||\hat{x}||_2 \\ &\leq c_n\varepsilon||\hat{x}||_2 + O(\varepsilon^2) \end{aligned}$$

Si par contre la ligne 1 est réalisée en simple précision, on obtient

$$\begin{aligned} ||x - \hat{x}||_2 &\leq ||x - (1+\alpha)\hat{x}||_2 + \varepsilon||\hat{x}||_2 \\ &\leq c_n\varepsilon||x||_2 + c_n\varepsilon||M^{-1}||_2||M||_2||x||_2 + O(\varepsilon^2) \\ &\leq c_n\varepsilon||M^{-1}||_2||M||_2||x||_2 + O(\varepsilon^2) \end{aligned}$$

Il est possible d'adopter une autre démarche consistant à corriger directement l'inverse approché B de M en utilisant par exemple la méthode de Newton

$$B_{\text{corrigé}} = 2B - BMB \tag{5.31}$$

Dans cette perspective, on observe que $B_{\text{corrigé}}$ est une matrice structurée et vérifie l'équation

de déplacement suivante

$$\begin{aligned} B_{\text{corrigé}} Z_1 - Z_{-1} B_{\text{corrigé}} &= 2\tilde{B}e_1 w^T \tilde{B} + 2\tilde{B}ve_n^T \tilde{B} \\ &\quad - \tilde{B}e_1 w^T \tilde{B}MB - \tilde{B}ve_n^T \tilde{B}MB \\ &\quad - BM\tilde{B}e_1 w^T \tilde{B} + BM\tilde{B}ve_n^T \tilde{B} \\ &\quad + Be_1 w^T B + Bve_n^T B \end{aligned}$$

En conséquence $B_{\text{corrigé}}$ peut être représentée grâce à cette structure en utilisant un stockage linéaire et peut être calculée à partir de (5.27) et (5.28) en utilisant $O(n \log n)$ opérations. Cette performance nous encourage à réaliser (5.31) en double précision. On montre que

$$||\hat{B}_{\text{corrigé}} - B_{\text{corrigé}}||_2 \leq c_n \varepsilon^2 (||B||_2 + ||B||_2 ||M||_2 ||B||_2)$$

De cette façon on peut montrer que moyennant des hypothèses sur la stabilité de la méthode d'inversion de départ et sur la précision

$$||\hat{B}_{\text{corrigé}} - M^{-1}||_2 \leq \varepsilon ||M^{-1}||_2 + O(\varepsilon^2)$$

Comparativement à l'algorithme (5.2) on voit que la méthode de Newton est plus coûteuse puisque pour calculer $B_{\text{corrigé}}$ on est contraint de calculer

$$w^T \tilde{B}MB, BM\tilde{B}v, BM\tilde{B}e_1, BM\tilde{B}v, w^T B, Bv$$

alors que dans l'algorithme (5.2) les opérations à réaliser sont

$$Mx \text{ et } Bd$$

D'autre part on voit du point de vue numérique dans la méthode de Newton le facteur

$$(||B||_2 + ||B||_2 ||M||_2 ||B||_2)$$

qui peut compromettre davantage la stabilité numérique. On conclut que l'aspect coûteux

à lui seul suffit de recommander l'algrithme 5.2 et déconseiller la méthode de Newton dans l'amélioration des systèmes linéaires de Toeplitz.

Chapter 6

Conclusion et Perspectives

Dans ce mémoire de Magister, nous avons considéré et étudié deux thèmes d'une grande importance en analyse numérique. Le premier concerne l'arithmétique en virgule flottante et le second concerne les approches de correction. La plupart des sujets abordés ici sont rassemblés de différents articles et ouvrages et présentés de la manière qui convient à notre contexte. En tant que travail bibliographique, nous estimons que l'objectif qui nous a été fixé est presque atteint et ne peut guère être atteint à cause des efforts considérables consentis dans cette direction. De par leur importance pratique et théorique indéniable, ces thèmes sont tout le temps en plein essor.

Dans un travail de recherche, même bibliographique, on est toujours motivé à essayer d'apporter sa contribution en tentant d'améliorer un resultat, de donner une démonstration plus simple ou de présenter des méthodes plus performantes. Dans cette direction, nous prétendons avoir apporté quelques petites contributions pouvant ouvrir des perspectives prometteuses. Nous les résumons ici d'une façon succinte.

Commençons d'abord par la citation (3.16) du théorème 3.13 qui est à notre avis nouvelle et n'est citée jusque là que dans notre article. Certes, c'est une estimation qui du point de vue pratique importe peu mais nous jugeons qu'elle est théoriquement intéressante. Motivés par le théorème de Sterbenz [47] et les travaux de Dekker, nous avons proposé une simple généralisation de ce théorème en suggérant le théorème 3.21 ainsi que sa démonstration. Concernant les

chapitres 4 et 5, nous pensons à notre avis que les méthodes de correction proposées pour les récurrences ou celles proposées pour les matrices Toeplitz sont nouvelles de sorte qu'un approffondissement numérique, expérimental et rigoureux donne lieu à des travaux publiables.

REFERENCES BIBLIOGRAPHIQUES :

[1] A.V. Aho, J. E. Hopcroft et J. D. Ullman *The design and analysis of computer algorithms* Addison-Wesley, Reading MA 1974

[2] S. Akl *The design and analysis of parallel algorithms* Prentice Hall 1989

[3] G. Ammar et P. Gader *New decompositions of the inverse of a Toeplitz matrix* Signal Processing Scattering and Operator theory and Numerical Methods, Proc. Int. Symposium MTN-89 vol(3) Birkhäuser Boston 421-428 1990

[4]K. Atkinson *Elementary numerical analysis* John Wiley & Sons 1985

[5] A. Björck *Iterative refinement of least squares solutions* I. BIT, 7: 257-278, 1967

[6] A. Björck *Iterative refinement of least squares solutions* II. BIT, 8: 8-30, 1968

[7] G. Bohlender *Floating-point computation of functions with maximum accuracy* IEEE Trans. Comput. C-26(7): 621-632, 1977

[8] A. W. Bojanczyk, R. P. Brent, F. R. de Hoog et D. R. Sweet *On the stability of the Bareiss and related factorization algorithms* SIAM J. Matrix Anal. Appl. 16:40-57 1995

[9] R. P. Brent *On the precision attainable with various floating-point systems* IEEE Trans. Comput. C-22(6): 601-607, 1973

[10] C. Brezinski *Algorithmique numérique* ellipses 1988

[11] N. Brisebarre et J. M. Muller *Correctly rounded multiplication by arbitrary precision constants* Laboratoire LIP, ENS-Lyon RR 2004-44 2004

[12] P. G. Cialrlet *Introduction à l'analyse numérique matricielle et à l'optimisation* Masson, Paris 1985

[13] J. T. Coonen *Underflow and the denormalized numbers* Computer, 14:75-87 1981

[14] T. Cormen, C. Leiserson et R. Rivest *Introduction à l'algorithmique* Dunod 1994

[15] M. Cosnard et D. Trystram *Algorithmes et architectures parallèles* InterEditions 1994

[16] T. J. Dekker *A floating-point technique for extending the available prcision* Numer. Math. 18:224-242 1971

[17] J. P. Demailly *Analyse numérique et équations différentielles* Collection Grenoble Sciences, Grenoble 1991

[18] J. W. Demmel *Underflow and the reliability of numerical software* SIAM J. Sci. Statist. Comput. 5(4): 887-919 1984

[19] J. W. Demmel et Y. Hida *Accurate finite summation* Manuscrit (accessible via Internet) 2003

[20] J. Durbin *The fitting of time series models* Rev. Inst. Int. Stat. 28:233-243 1960

[21] T. O. Espelid *On floating point summation* Report N° 67 Departement of Applied Mathematics, University of Bergen (Norvège) 1978

[22] W. E. Ferguson, Jr. *Exact computation of a sum or difference with applications to argument reduction* Proc. 12^{th} IEEE symposium on Computer arithemetic Bath England S. Knowles et W. H. McAllister eds IEEE Computer Society Press 1995

[23] I. Gohberg et V. Olshevsky *Circulant, displacement and decompositions of matrices* Integral Equations Operator Theory 15(1992) 730-743

[24] I. Gohberg, T. Kailath et V. Olshevsky *Fast Gaussian elimination with partial pivoting for matrices with displacement structure* Math. of Comp. 64(1995) 1557-1576

[25] D. Goldberg *What every computer scientist should know about floating-point arithmetic?* ACM surveys vol. 23 no 1: 5-48 1991

[26] G. H. Golub et C. F. Van Loan *Matrix computations John Hopkins U. P. Baltimore 1996*

[27] N. J. Higham *Iterative refinement enhances the stability of QR factorization methods for solving linear equations* BIT 31:447-468 1991

[28] N. J. Higham *The accuracy of floating-point summation* SIAM J. Sci. Comput. 14(4):783-799 1993

[29] N. J. Higham *Stability of parallel triangular system solvers* SIAM J. Sci. Comput. 16(2):400-413 1995

[30] N. J. Higham *Accuracy and stability of numerical algorithms* SIAM Philadelphia Press 1996

[31] IEEE *IEEE Standard 754-1985 for binary floating-point arithmetic* IEEE. Reprinted in SIGPLAN 22(2): 9-25 1987

[32] M. Jankowski, A. Smoktunowicz et H. Wozniakowski *A note on floating-point summation of very many terms* J. Inform. Process. Cybernet. 19(9): 435-440 1983

[33] M. Jankowski et H. Wozniakowski *Iterative refinement implies numerical stability* BIT 17: 303-3111977

[34] W. Kahan *A survey of error analysis* Information Processing 71 north Holland Amsterdam vol 2:1214-1239 1972

[35] D. E. Knuth *The art of computer programming* 3 volumes Addison-Wesley, Reading MA 1973-1981

[38] P. Kornerup et J. M. Mulller *RN coding of numbers definitions and properties* Rapport de recherche Laboratoire LIP ENS-Lyon RR no 04-43 2004

[37] M. Lakrib *Cours d'analyse numérique* OPU 2003

[38] V. Lefèvre et P. Zimmerman *Arithmétique flottante* Rapport de recherche Laboratoire LIP ENS-Lyon RR no 5105 2004

[39] S. Linnainmaa *Analysis of some known methods of improving the accuracy of floating-point sum* BIT 16:167-202 1974

[40] L. Melkemi *Contribution à l'étude des matrices structurées* Thèse de Doctorat es sciences Batna 2000

[41] M. Pichat *Correction d'une somme en arithmétique à virgule flottante* Numer. Math. 19:400-406 1972

[42] J. H. Reiser et D. E. Knuth *Evading the drift in floating-point addition* Inf. Pocess. Letters 3(3):84-87 1975

[43] A. H. Sameh et R. P. Brent *Solving triangular systems on a parallel computer* SIAM J. Matrix Anal. Appl. 14(6):1101-1113 1977

[44] R. D. Skeel *Iterative refinement implies numerical stability for Gaussian elimination* J. ACM 26(3): 694-526 1979

[45] P. H. Sterbenz *Floating Point Computations* Prentice Hall Englwood Cliffs NJ USA 1974

[46] J. Stoer et R. Bulirsh *Introduction to numerical analysis* Springer Verlag 1983

[47] C. F. Van Loan *Computational framework for the fast Fourier transform* SIAM Philadelphia Press 1992

[48] I. V. Viten'ko *Optimum algorithms for adding and multiplying on computers with floating-point* U.S.S.R. Comput. Math. Math. Phys. 8(5):183-195 1968

[49] J. H. Wilkinson *Rounding Errors in Algebraic Processes* Her Majesty's Stationnary Office London 1963. Réimprimé par Dover, Newyork 1994

[50] J. H. Wilkinson et C. Reinsch (eds) *Linear Algebra* Volume II of Handbook for automatic computation Springer Verlag Berlin 1971

مختصر.-(الحساب بالأعداد ذات الفاصلة المتنقلة و الطرق التصحيحية)

في هذه المذكرة نهتم بالحساب التقريبي بواسطة الأعداد المكتوبة بالفاصلة المتنقلة مع الدقة المنتهية. نتعرض إلى النموذج المعروف و إلى كيفية إنجاز العمليات الأولية. نشير إلى ان النظام الذي سنقدمه هو المعمول من طرف كل الحواسيب. نهتم بعدها بالطرق التصحيحية التي تحاول جمع الأخطاء المتراكمة في مراحل الطريقة الأولى. لقد تبين-كما سنرى- أن هذا الأخير يؤدي في كثير من الأحيان إلى تحسينات مفيدة في النتائج التقريبية المتحصل عليها من ذي قبل.

Abstract.- (Floating-point Arithmetic and correction methods)

In this memory we are interested to the floating-point arithmetic with finite precision. We present the different numerical aspects such as overflow, rounding and the standard model. Then we consider the correction methods that attempt to gather the rounding errors accumulated in the different steps of the first method. It turns out, as we will see, that such an approach often leads to important improvements in the yet obtained computed results.

Résumé-

Dans ce mémoire nous nous intéressons d'abord à l'arithmétique en virgule flottante avec précision finie. Nous présentons les différents aspects numériques comme les dépassements, les arrondis et le modèle standard. Ensuite nous considérons les méthodes de correction qui tentent de rassembler les erreurs d'arrondi accumulées dans les étapes de la première méthode. Il s'est avéré que cette approche conduit le plus souvent à d'importantes améliorations dans les solutions approchées déjà obtenues.