ERREURS DE CALCUL DES ORDINATEURS

ROUNDOFF ERRORS ON COMPUTERS

Jocelyne Erhel

IRISA-INRIA
Campus de Beaulieu
35042 Rennes Cedex
e-mail : erhel@irisa.fr

Septembre 1990

Abstract

This report describes floating-point arithmetic on computers and shows the effect of roundoff errors in scientific computations. The first part is devoted to floating-point formats, rounding modes, and implementation of the usual arithmetic operations. In the second part we introduce the concepts of stability, condition, error analysis. These are illustrated by means of numerous examples.

Résumé

Ce rapport a pour objectifs de décrire l'arithmétique à virgule flottante des ordinateurs et de mettre en évidence l'effet des erreurs d'arrondi dans les logiciels scientifiques. La première partie explique en détails la représentation des réels, les modes d'arrondi, et la mise en œuvre des opérations arithmétiques usuelles. La seconde partie introduit les notions de stabilité, conditionnement, et analyse d'erreur, en les illustrant par de nombreux exemples.

Table des matières

1 Introduction

1.1 Evolution du calcul scientifique

Le calcul scientifique représente l'ensemble des techniques nécessaires à la résolution des problèmes physiques tels que l'étude de la trajectoire d'un satellite, la conception d'une centrale nucléaire, l'étude de la structure d'une molécule, etc... Ces problèmes qui étaient auparavant résolus par expérimentation physique sont maintenant traités par calcul. Pour l'illustrer, considérons l'industrie aéronautique.

Il y a vingt ans, la définition d'un profil d'avion ne pouvait se traiter que par la construction successive de maquettes que l'on plaçait ensuite dans un tunnel de soufflerie pour en étudier l'aérodynamisme. Maintenant les concepteurs ont recours à la simulation numérique : il s'agit de simuler l'expérience par un modèle mathématique dont on a à calculer une solution. L'avantage de cette approche est double : d'abord d'ordre économique, car la réalisation de maquettes et des outils nécessaires à leur étude est chère; ensuite parce que la simulation numérique accélère le processus de conception. En effet, une fois la définition du modèle mathématique et sa méthode de résolution connues, un changement de profil ne correspond qu'à un changement de données pour le programme ce qui simplifie beaucoup la mise au point. L'usage de la soufflerie se limite alors à la validation finale du profil obtenu.

Les techniques du calcul scientifique ont évolué dans deux directions:

- L'accélération de la vitesse de calcul des ordinateurs. Deux facteurs ont contribué à cet essor: l'amélioration de la qualité de l'électronique et la nouvelle organisation des architectures. A la fin des années soixante, le CDC 7600 atteignait presque la vitesse de 10 Mflops (1 Mégaflops = 1 million d'opérations flottantes par seconde); aujourd'hui le supercalculateur CRAY Y-MP atteint 2660 Mflops, le facteur de gain en vingt ans a été de plus de 250.

- L'amélioration des algorithmes, avec réduction de la complexité et/ou meilleure adéquation à l'architecture. On considère qu'en quinze ans le gain a atteint dans ce domaine un facteur 100.

Le gain total a donc été en quinze ans d'environ 10 000. Grâce à cette accélération de puissance, il n'est plus de domaine qui n'essaie d'introduire le calcul scientifique pour résoudre ses problèmes; la liste suivante ne donne donc qu'une idée de la diversité des préoccupations:

- exploration pétrolière,
- conceptions de systèmes électroniques,
- industries nucléaires,
- aéronautique,
- domaine spatial,

- industries automobiles,
- industries pharmaceutiques,
- chimie quantique,
- météorologie,
- modèles d'économies,
- synthèse d'images ...

1.2 Limites de l'arithmétique flottante

Les nombres réels ne peuvent qu'être approchés sur un ordinateur. Toute représentation des nombres réels par un ensemble fini de valeurs est jugée selon au moins deux critères:

- l'amplitude de l'intervalle des nombres représentés (écart entre le plus petit et le plus grand nombre);
- la distribution des écarts entre deux nombres représentés consécutivement.

La première représentation qui a été utilisée est celle de la virgule fixe; elle est caractérisée par un écart constant entre les nombres représentés. Cela entraîne une majoration d'erreur absolue constante pour les arrondis. Comme cette représentation ne permet pas d'avoir à la fois une bonne précision sur les petites valeurs et une large amplitude de la représentation, on a inventé la représentation flottante qui permet d'avoir plutôt une erreur relative constante dans les arrondis. La représentation flottante correspond à la notation scientifique où l'on se fixe le nombre de chiffres significatifs et où un nombre est de la forme $\pm 10^e \times m$ avec le nombre décimal m qui vérifie $1 \leq m < 10$ et qui possède un nombre fixe de décimales. Pratiquement, pour un ordinateur, on choisit en général une autre base que la base décimale; le plus souvent, c'est la base deux qui est utilisée. La représentation est donc caractérisée par la base, le nombre de chiffres de la *mantisse* m et le nombre de chiffres de l'entier e qui est appelé *exposant*.

Une fois, la représentation flottante choisie, on définit les différents modes d'arrondis, c'est-à-dire le moyen de projeter les réels sur l'ensemble des flottants. L'arrondi le plus courant, parce qu'il minimise les erreurs d'arrondis est l'*arrondi au plus près*: comme son nom l'indique il consiste à représenter un réel par le nombre flottant le plus proche.

L'ensemble des nombres flottants est ensuite muni d'opérations qui essaient de traduire au mieux les quatre opérations dans les réels. Malheureusement, on verra que les opérations flottantes ne conservent pas les propriétés des opérations réelles, par exemple l'addition n'est pas associative. En particulier, deux phénomènes qui se produisent lors d'additions et de soustractions, l'*absorption* et l'*élimination*, sont responsables de bien des erreurs dans les exécutions de résolutions numériques.

1.3 Qualification des problèmes et des algorithmes

Les algorithmes sont plus ou moins sensibles à ces erreurs d'arrondi, ainsi qu'aux erreurs sur les données initiales. Un algorithme est stable si l'erreur sur le résultat reste petite malgré les erreurs d'arrondi. Un problème est stable si une petite perturbation des données n'entraîne qu'une petite perturbation des résultats. Le conditionnement d'un problème mesure cette stabilité, en évaluant le facteur d'amplification d'une perturbation des données. L'analyse d'erreur d'un algorithme évalue l'erreur sur un résultat due aux erreurs d'arrondi.

Nous verrons sur des exemples comment un problème instable peut conduire à des résultats complètement faux, et comment un problème mal conditionné peut amplifier les erreurs sur les données initiales. D'autres exemples illustreront la sensibilité des algorithmes aux erreurs d'arrondi, et montreront comment deux formulations équivalentes mathématiquement ont des comportements numériques très différents.

Part I
Arithmétique flottante

2 Représentation des nombres réels

2.1 Numération des entiers et des décimaux

Il nous est familier de compter les entiers en base 10, et de les écrire à l'aide de chiffres compris entre 0 et 9. De manière générale, on peut représenter un entier x en base b par l'expression suivante:

$$\begin{cases} x = a_{N-1}\ b^{N-1} + \ldots + a_1\ b + a_0 \\ \text{avec}\ \ 0 \leq a_i < b \ \ i = 0, \ldots, N-1 \end{cases}$$

Exemples:

$$\begin{aligned} \text{en base 10:} \quad 1536 = &\quad 1 \times 10^3 + 5 \times 10^2 + 3 \times 10 + 6 \\ \text{en base 2:} \quad 1010 = &\quad 2^3 + 2^1 = 10 \end{aligned}$$

Pour représenter les entiers relatifs, une solution simple consiste à ajouter un signe devant la représentation. Pour coder des entiers compris strictement entre $-b^N$ et b^N, il faut alors un code pour le signe, et N chiffres.
Exemple: en base 10, pour coder les nombres entiers strictement compris entre -1000 et 1000, il faut le signe et 3 chiffres.
Remarque : une autre manière de coder les entiers relatifs est la représentation dite en *complément à la base*, qui ne sera pas décrite ici.

Cette *numération de position* s'étend facilement aux *nombres décimaux*. On introduit la notion de *virgule*, les chiffres avant la virgule correspondent aux puissances positives de la base, et les chiffres après la virgule aux puissances négatives :

$$\begin{cases} x = a_{N-1}\ b^{N-1} + \ldots + a_1\ b + a_0 + a_{-1}\ b^{-1} + \ldots + a_{-p}\ b^{-p} \\ \text{avec}\ \ 0 \leq a_i < b\ \ i = -p, \ldots, N-1 \end{cases}$$

Les nombres décimaux sont en principe écrits en base 10. Par extension, on appelle nombre décimal un nombre écrit en base b et dont le nombre de chiffres aprés la virgule est fini.
Exemples:

$$\begin{aligned} \text{en base 10:}\quad 15.36 = &\ \ 1 \times 10 + 5 + 3 \times 1/10 + 6 \times 1/100 \\ \text{en base 2:}\quad 1.011 = &\ \ 1 + 1/4 + 1/8 = 11/8 \end{aligned}$$

2.2 Contraintes pour représenter les réels

Les nombres réels ne peuvent être représentés par un nombre décimal avec un nombre fini de chiffres, et ne peuvent être qu'approchés.

Pour coder en machine l'ensemble des nombres réels, noté $\mathcal{R}$, on ne dispose que d'un nombre fini de chiffres. Il faut donc essayer de représenter les réels par un ensemble fini de nombres, noté $\mathcal{F}$. Ceci a deux conséquences importantes:

- Cet ensemble est borné: les nombres représentables sont compris dans l'intervalle $[xmin, xmax]$.
- L'écart entre deux nombres représentables consécutifs est un intervalle qui contient une infinité de réels. Tous les réels de cet intervalle doivent être approchés par l'une de ses bornes. On constate donc que la largeur de l'intervalle doit être assez petite pour assurer une bonne précision, c'est-à-dire une petite différence entre un réel et sa valeur approchée.

Or, sur un mot binaire de N bits, on ne peut coder que 2^N nombres différents. Donc il faut établir un compromis entre la plage de nombres représentables et la précision.

Dans la suite, on ne considère que des ensembles symétriques par rapport à 0; en particulier, on a $xmin = -xmax$.

2.3 Représentation en virgule fixe

Un bit est réservé pour coder le signe d'un nombre, comme dans le cas des entiers relatifs.

Les nombres réels sont approchés par des nombres décimaux dont le nombre total de chiffres ainsi que le nombre de chiffres après la virgule sont déterminés. Cette représentation est appelée à *virgule fixe*, car la position de la virgule est la même pour tous les nombres. Soit N le nombre total de chiffres, et p le nombre de chiffres après la virgule. Tout nombre représentable x s'écrit:

$$x = a_{N-p-1}\ b^{N-p-1} + \ldots + a_1\ b + a_0 + a_{-1}\ b^{-1} + \ldots + a{-p}\ b^{-p}$$

longueur	représentation	précision
1.23 m	1.23	0.01=1 cm
1.23 km	1230.00	0.01=1 cm
1.23 cm	0.01	0.01=1 cm

Table 1 : Représentation en virgule fixe

L'ensemble des nombres représentables est symétrique par rapport à 0.

Le plus grand nombre représentable est alors:

$$\begin{aligned} xmax &= (b-1)\ b^{N-p-1} + \ldots + (b-1) + \ldots + (b-1)\ b^{-p} \\ &= b^{N-p} - b^{-p} \end{aligned}$$

Les intervalles entre deux nombres représentables consécutifs sont de longueur constante. Cette longueur, notée ϵ, correspond ici au plus petit nombre représentable strictement positif.

$$\epsilon = b^{-p}$$

Exemple: Soit un système de mesure des longueurs, dont l'unité est le mètre, la plus grande longueur mesurable est légèrement inférieure à 10 kilomètres, et la précision est le centimètre. Ce système correspond aux paramètres suivants:

$$\begin{aligned} b &= 10 \\ p &= 2 \\ N &= 6 \end{aligned}$$

Pour toute longueur, quel que soit son ordre de grandeur, la précision de la mesure est de 1 cm, comme le montre la table 1. En d'autres termes, la *précision absolue* est constante. D'autre part, la plage des longueurs mesurables est réduite, de 1 cm à 10 km. On ne dispose donc que d'un seul instrument de mesure, qui est une règle de longueur 10 km, graduée en cm.

De manière générale, la précision absolue de la représentation en virgule fixe est constante, et déterminée par ϵ. Soit x un réel et $\Box x$ sa valeur approchée. Selon le mode d'arrondi choisi, on a

$$\begin{aligned} \forall x \in \mathcal{R}\ \cap\ [-xmax, xmax] \quad |x - \Box x| &\leq \epsilon \\ \text{ou} \quad |x - \Box x| &\leq \epsilon/2 \end{aligned}$$

Les ordres de grandeur représentables sont fixes, déterminés par le nombre de chiffres avant la virgule. Or sur un ordinateur, les données d'entrée peuvent avoir des ordres de grandeur très différents, et surtout ceux des résultats intermédiaires sont inconnus. Ces deux inconvénients ont conduit à adopter un autre système de représentation, où l'ordre de grandeur d'un nombre est explicité.

2.4 Représentation en virgule flottante normalisée

Là encore, un bit est réservé pour coder le signe d'un nombre.

Les nombres réels sont approchés par des nombres décimaux décomposés en *exposant* e et *mantisse* m, de la façon suivante:

$$x = b^e \times m$$

l'exposant est un nombre entier relatif, et la mantisse est un nombre décimal. Cette notation est appelée à *virgule flottante*, car la position de la virgule varie suivant la valeur de l'exposant. On notera que cette représentation n'est pas unique.

Exemple : en base 10, le nombre 3.14 peut s'écrire aussi 314 10^{-2} ou encore 0.0314 10^2.

On constate qu'une représentation avec des 0 à gauche est moins précise, dans la mesure où le nombre de chiffres de la mantisse est déterminé.

Afin de choisir une représentation unique et précise, on impose que la mantisse soit *normalisée*, c'est-à-dire que le premier chiffre non nul soit celui des unités. Autrement dit, la mantisse est comprise entre 1 (inclus) et la base b (exclus). Cette convention implique qu'il faut choisir une représentation spéciale pour le nombre 0.

Le nombre de chiffres de la mantisse est fixe. L'exposant est codé sous forme d'un entier relatif, avec également un nombre fixe de chiffres. L'ensemble des nombres flottants normalisés $\mathcal{F}$ est défini par les paramètres suivants:

- la base b,
- la précision ou le nombre de chiffres de la mantisse p,
- l'exposant minimal E_{min},
- l'exposant maximal E_{max}.

Tout nombre non nul x de $\mathcal{F}$ est alors défini par son *signe* s, son *exposant* e, et sa *mantisse* m, et s'écrit:

$$\begin{cases} \forall x \in \mathcal{F} \ x \neq 0 \ x = (-1)^s b^e m \\ \text{avec} \ \ 1 \leq m < b \end{cases}$$

et m s'écrit:

$$\begin{cases} m = a_0 + a_1 * b^{-1} + a_2 * b^{-2} + \ldots + a_{p-1} * b^{-(p-1)} \\ \text{avec} \ \ 0 \leq a_i \leq b-1 \ i = 0, ..p-1 \\ \text{et} \ \ a_0 \neq 0 \end{cases}$$

Le nombre nul a un exposant et une mantisse nuls:

$$x = 0 \Rightarrow e = m = 0$$

longueur	représentation	précision
1.23 m	1.23	0.01=1 cm
1.23 km	$1.23\ 10^3$	$0.01\ 10^3$=10 m
1.23 cm	$1.23\ 10^{-2}$	$0.01\ 10^{-2}$=0.1 mm

Table 2 : Représentation en virgule flottante

Exemple : Soit un système de mesure des longueurs, dont l'unité est le mètre, la plus grande longueur mesurable est légèrement inférieure à 10 kilomètres, la plus petite longueur mesurable est le millimètre, et la précision pour 1m est le centimètre. Ce système correspond aux paramètres suivants:

$$\begin{aligned} b &= 10 \\ p &= 3 \\ E_{max} &= 3 \\ E_{min} &= -3 \end{aligned}$$

La précision absolue de la mesure dépend de l'ordre de grandeur de celle-ci. C'est la précision relative qui est constante, égale ici au 1/100, comme le montre la table 2. D'autre part, la plage des longueurs mesurables varie cette fois de 1 mm à 10 km. Mais on dispose de plusieurs instruments de mesure, qui sont des règles graduées au 1/100. Par exemple, on dispose d'une règle de longueur 10cm graduée tous les 0.1mm, d'une autre règle de longueur 10m graduée tous les cm, d'une autre encore de longueur 10km, graduée tous les 10m.

De manière générale, on peut faire les remarques suivantes:

- L'ensemble $\mathcal{F}$ est symétrique par rapport à 0.
- La plage des nombres strictement positifs représentables varie du plus petit nombre u, au plus grand nombre $xmax$, qui valent:

  $$\begin{aligned} xmax &= b^{E_{max}} * (b - b^{-(p-1)}) \\ u &= b^{E_{min}} \end{aligned}$$

  $xmax$ est appelé le seuil d'*overflow* et u est appelé le seuil d'*underflow*.
- L'ensemble des nombres entiers de l'intervalle $[-xmax, xmax]$ est inclus dans $\mathcal{F}$.
- Les intervalles entre deux nombres représentables sont de longueur variable, suivant l'ordre de grandeur de ces nombres. Cette représentation garantit la même précision relative, non la même précision absolue. La longueur de l'intervalle pour l'unité, ou encore le plus petit nombre réel x tel que $1 + x$ soit représentable en machine est appelé ϵ *machine*.

Selon le mode d'arrondi choisi, la précision relative de l'approximation d'un réel x par sa représentation flottante $\Box x$ vaut ϵ ou $\epsilon/2$.

$$\epsilon \;=\; b^{-(p-1)}$$

$$\begin{aligned} \forall x \in \mathcal{R} \; \cap \; [-xmax, xmax] \quad |x - \Box x| \quad &\leq \epsilon \; |x| \\ \text{ou} \quad |x - \Box x| \quad &\leq \epsilon/2 \; |x| \end{aligned}$$

Remarque : le plus petit nombre réel x tel que la représentation flottante de $1 + x$ soit différente de 1 peut être inférieur à ϵ. En effet, on peut avoir

$$\begin{aligned} 0 < x < \epsilon \\ \Box(1 + x) = 1 + \epsilon \end{aligned}$$

- Il y a un trou entre 0 et le plus petit nombre positif u. Ceci pose des problèmes pour effectuer des opérations sur des nombres très petits. Afin de combler en quelque sorte ce trou, c'est-à-dire de le subdiviser en intervalles en rajoutant des nombres flottants, ont été définis les *nombres dénormalisés*. Dans ce cas, l'exposant vaut E_{min}, et les premiers chiffres de la mantisse (non nulle) peuvent être nuls. On divise ainsi l'intervalle $[0, u]$ en intervalles égaux de longueur $ud = b^{E_{min}-(p-1)}$, et le plus petit nombre strictement positif représentable est ud.

2.5 Choix de la base

L'être humain calcule en base 10. Les entrées et sorties d'un calculateur doivent donc être converties en base 10. Mais la représentation des nombres peut être dans une autre base.

Les calculatrices de poche utilisent en général la base 10 pour éviter les conversions qui seraient trop nombreuses.

Certains gros calculateurs scientifiques (IBM par exemple) utilisent la base 16 afin de regrouper les bits 4 par 4 pour effectuer les opérations arithmétiques.

Cependant la base 2 reste la plus utilisée, pour diverses raisons. Les chiffres doivent être codés par des bits pour des raisons technologiques. La base 2 semble donc naturelle, et est la plus facile à réaliser. En outre, le premier chiffre de la mantisse est non nul, et vaut donc 1 en base 2. On peut donc ne pas le mémoriser et gagner ainsi un chiffre: c'est la règle du *premier bit implicite*. Grâce à cette convention, c'est la base 2 qui garantit la meilleure précision, pour un nombre de bits par mot donné, et pour une plage représentable donnée.

2.6 Norme IEEE

Lorsque les calculateurs ont des systèmes à virgule flottante différents, l'usager se trouve confronté à des problèmes de portabilité. Un même programme

paramètre	simple précision	double précision
base b	2	2
nombre de bits	32	64
précision p	24	53
exposant minimal E_{min}	-126	-1022
exposant maximal E_{max}	+127	+1023
biais de l'exposant	127	1023

Table 3 : Formats de la norme IEEE

peut donner des résultats corrects sur une machine et faux sur une autre. Une méthode itérative peut converger dans un cas et pas dans l'autre. La norme IEEE a permis de définir un standard pour représenter les nombres flottants. Cette norme contient non seulement le format des nombres, mais aussi les modes d'arrondi possibles, le traitement des *exceptions* (overflow, division par 0, etc ...), les conversions (en décimal par exemple), et l'arithmétique. Si la plupart des microprosseurs respectent cette norme, beaucoup de gros calculateurs (IBM, CRAY, par exemple) ont conservé des formats et des arithmétiques différents, ce qui n'est pas sans poser des problèmes d'arrondi.

La norme IEEE définit deux formats, appelés *simple précision* et *double précision*, résumés dans la table 3. Les flottants sont codés en base 2, avec le premier bit implicite, sur 32 (simple précision) ou 64 bits (double précision).

Lors des opérations arithmétiques, on est amené à additionner les exposants. Or il est plus facile d'additionner des entiers positifs. On introduit donc un biais, égal à $-E_{min}+1$, de façon à ce que tous les exposants soient positifs. Le plus petit exposant E_{min} est alors codé comme 1. En simple précision, les exposants varient de 1 à 254 (les exposants 0 et 255 sont réservés pour des nombres spéciaux).

Par convention, l'exposant du nombre 0 reste nul.

La norme IEEE introduit également des nombres spéciaux, qui permettent de traiter les exceptions. NaN signifie " Not a Number", et est le résultat d'une opération illicite, telle que 0/0. $+\infty$ et $-\infty$ traitent le cas des overflows, après une division par 0 par exemple. Les valeurs des nombres flottants sont résumées dans la table 4 pour la simple précision.

Dans cette table, *fraction* signifie la valeur de la mantisse sans le premier bit implicite, toujours égal à 1 (sauf pour $e = 0$). La mantisse est donc égale à 1.f (ou à 0.f).

3 Arrondis d'un nombre réel

La représentation des réels est une application de $\mathcal{R}$ vers $\mathcal{F}$ appelée *arrondi*. Cet arrondi, noté $\Box$, doit satisfaire les propriétés suivantes:

exposant e	fraction f	valeur
255	$\neq 0$	NaN Not a Number: erreur
255	0	$(-1)^s\ \infty$ $\pm$ infini
$0 < e < 255$		$(-1)^s\ 2^{e-127}\ (1.f)$ nombre flottant normalisé
0	$\neq 0$	$(-1)^s\ 2^{-126}\ (0.f)$ nombre dénormalisé
0	0	$(-1)^s\ 0$ $\pm$ nombre nul

Table 4 : Simple précision IEEE

$$
\begin{array}{ll}
\forall x \in \mathcal{F} & \Box x = x \\
\forall x \in \mathcal{R} & -xmax \leq x \leq xmax \Rightarrow \Box x - x \text{ est petit} \\
\forall x, y \in \mathcal{R} & -xmax \leq x \leq y \leq xmax \Rightarrow \Box x \leq \Box y
\end{array}
$$

3.1 Modes d'arrondi

La norme IEEE définit quatre types d'arrondi: la *troncature*, ou arrondi vers 0, qui est le plus simple à réaliser, les arrondis *par excès* ou vers $+\infty$, et *par défaut* ou vers $-\infty$, et l'arrondi *au plus près* qui est le plus précis. Les arrondis par excès et par défaut servent essentiellement à réaliser une arithmétique d'intervalles. Les différents modes d'arrondi sont illustrés sur la figure 1 et sont définis pour tout nombre réel x par les formules suivantes:

- arrondi par défaut: $\bigtriangledown x = \max y \in \mathcal{F}\ y \leq x$,
- arrondi par excès: $\triangle x = \min y \in \mathcal{F}\ y \geq x$,
- troncature: $\bigcirc x = \bigtriangledown x$ si $x \geq 0$, et $\bigcirc x = \triangle x$ si $x \leq 0$,
- arrondi au plus près: $\forall y \in \mathcal{F}\ |x - \Diamond x| \leq |x - y|$.

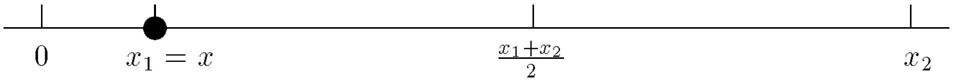
Il est facile de voir que l'arrondi au plus près offre la meilleure précision. En effet, on obtient les formules d'encadrement suivantes, où e désigne l'exposant de $\Box x$, et $\Box$ est l'un des arrondis précédents.

$$
\begin{array}{lrcll}
\forall \Box \in \bigtriangledown, \triangle, \bigcirc & |x - \Box x| & \leq & b^{e-(p-1)} & \leq \epsilon\ |x| \\
 & |x - \Diamond x| & \leq & 1/2 * b^{e-(p-1)} & \leq \epsilon/2\ |x|
\end{array}
$$

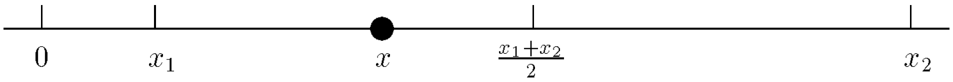
Définition : la précision d'un arrondi $\Box$ est le plus petit nombre réel u tel que

$$\forall x \in \mathcal{R}\ \cap\ [-xmax, xmax]\ \ |x - \Box x|\ \leq u\ |x|$$

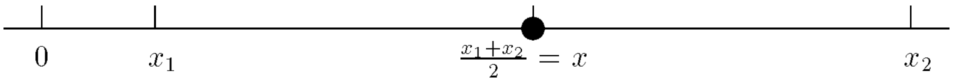
$x_1 \in \mathcal{F} \quad x_2 \in \mathcal{F} \qquad x \in \mathcal{R} \quad 0 \leq x_1 \leq x < x_2$



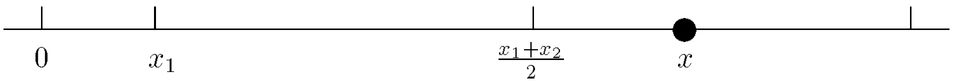
1^{er} cas: $x = x_1$
$\bigtriangledown x = \bigcirc x = \bigtriangleup x = \Diamond x = x_1 \quad (x \in \mathcal{F} \Rightarrow \Box x = x)$



2^e cas: $x_1 < x < (x_1 + x_2)/2$
$\bigtriangledown x = \bigcirc x = \Diamond x = x_1 \quad \bigtriangleup\, x = x_2$



3^e cas: $x = (x_1 + x_2)/2$
$\bigtriangledown x = \bigcirc x = x_1 \quad \bigtriangleup\, x = x_2 \quad \Diamond x = x_1 \; ou \; x_2$



4^e cas: $(x_1 + x_2)/2 < x < x_2$
$\bigtriangledown x = \bigcirc x = x_1 \quad \bigtriangleup\, x = \Diamond x = x_2$

Figure 1 : Arrondis d'un réel x

valeur avec 5 chiffres	arrondis avec 3 chiffres		
	par défaut	par excès	au plus près
1.2500	1.25	1.25	1.25
1.2534	1.25	1.26	1.25
1.2550	1.25	1.26	1.26
1.2556	1.25	1.26	1.26
9.9951	9.99	10.00	10.00

Table 5 : Exemples d'arrondi en base 10

La précision de l'arrondi au plus près vaut $\epsilon/2$, tandis que celle des autres arrondis vaut ϵ.

Exemples: Soit le système de mesure précédent, c'est-à-dire :

$$\begin{aligned} b &= 10 \\ p &= 3 \\ E_{max} &= 3 \\ E_{min} &= -3 \end{aligned}$$

La table 5 montre les différents arrondis pour diverses longueurs. Les chiffres après le p^e permettent de déterminer la position du nombre dans l'intervalle flottant qui le contient.
En général, il faut arrondir le résultat d'une opération arithmétique, qui est un nombre représentable avec une précision supérieure à p. Combien de chiffres faut-il garder dans le résultat pour garantir un arrondi correct?

3.2 Arrondi d'une précision q à la précision p

Soit $q \geq p$, et x un nombre flottant écrit avec q chiffres. Il s'agit de déterminer l'arrondi de x avec p chiffres. Rappelons que $\epsilon = b^{-(p-1)}$.

$$x = (-1)^s * b^e * (a_0 + a_1 * b^{-1} + a_2 * b^{-2} + \ldots + a_{q-1} * b^{-(q-1)})$$

Soit m la partie de la mantisse de x avec p chiffres, et h le reste de la mantisse:

$$\begin{aligned} m &= a_0 + a_1 * b^{-1} + a_2 * b^{-2} + \ldots + a_{p-1} * b^{-(p-1)} \\ h &= a_p * b^{-p} + a_{p+1} * b^{-(p+1)} + \ldots + a_{q-1} * b^{-(q-1)} \end{aligned}$$

On a donc

$$\begin{aligned} & x = (-1)^s \ b^e \ (m + h) \\ & \text{avec} \ \ 1 \leq m \leq b - \epsilon \\ & \text{et} \ \ 0 \leq h < \epsilon \end{aligned}$$

x est dans l'intervalle $[x_1, x_2]$, où x_1 et x_2 sont deux nombres flottants consécutifs:

$$\begin{array}{lrcl} x_1 \in \mathcal{F} & x_1 & = & (-1)^s * b^e * m \\ x_2 \in \mathcal{F} & x_2 & = & (-1)^s * b^e * (m + \epsilon) \end{array}$$

En outre, on a les égalités:

$$\begin{array}{rcl} x - x_1 & = & (-1)^s * b^e * h \\ x_2 - x & = & (-1)^s * b^e * (\epsilon - h) \end{array}$$

Selon le mode d'arrondi, il faut avoir plus ou moins d'information sur h pour déterminer si l'arrondi de x vaut x_1 ou x_2.

- arrondi par défaut:

$$\begin{array}{rrcl} \text{si} \ \ x > 0 & \bigtriangledown x & = & x_1 \\ \text{si} \ \ x < 0 \ \ \text{et} \ \ h = 0 & \bigtriangledown x & = & x_1 \\ \text{si} \ \ x < 0 \ \ \text{et} \ \ h \neq 0 & \bigtriangledown x & = & x_2 \end{array}$$

- arrondi par excès:

$$\begin{array}{rrcl} \text{si} \ \ x > 0 \ \ \text{et} \ \ h = 0 & \bigtriangleup x & = & x_1 \\ \text{si} \ \ x > 0 \ \ \text{et} \ \ h \neq 0 & \bigtriangleup x & = & x_2 \\ \text{si} \ \ x < 0 & \bigtriangleup x & = & x_1 \end{array}$$

- arrondi par troncature:

$$\forall h \quad \bigcirc x \ = \ x_1$$

- arrondi au plus près:

$$\begin{array}{rrcl} \text{si} \ \ h < \epsilon/2 & \diamond x & = & x_1 \\ \text{si} \ \ h > \epsilon/2 & \diamond x & = & x_2 \\ \text{si} \ \ h = \epsilon/2 & \diamond x & = & x_1 \ \ \text{ou} \ \ x_2 \end{array}$$

L'arrondi par troncature ne requiert donc aucune information. Par contre, pour déterminer l'arrondi par défaut ou par excès, il faut savoir si h est non nul. Dans le cas de l'arrondi au plus près, il faut comparer h à $\epsilon/2$.

Pour déterminer si un nombre décimal est nul, on lui associe un *sticky bit* :

Définition : soit y un nombre positif décimal, avec q chiffres. En base b différente de 2, le sticky bit de y, noté s_y, est la somme des chiffres de y. En base 2, le sticky bit de y est le OU logique des chiffres de y.

$$\begin{array}{l} y = c_0 + c_1 * b^{-1} + c_2 * b^{-2} + \ldots + c_{q-1} * b^{-(q-1)} \\ \text{en base b:} \ \ s_y = c_0 + c_1 + \ldots + c_{q-1} \\ \text{en base 2:} \ \ s_y = c_0 \ \text{OU} \ c_1 \ \text{OU} \ldots \ \text{OU} \ c_{q-1} \end{array}$$

Pour déterminer si y est non nul, il suffit de savoir si son sticky bit est non nul.

$$y \neq 0 \Leftrightarrow s_y \neq 0$$

Donc il suffit de calculer le sticky bit de h pour calculer les arrondis par défaut ou par excès de x.

En ce qui concerne l'arrondi au plus près, il faut déterminer le signe de $h - \epsilon/2$. Si la base b est paire, $\epsilon/2$ est représentable, et la comparaison est réalisable. Par contre, si la base b est impaire, $\epsilon/2$ n'est pas représentable par un nombre décimal, et il n'est pas possible de comparer h à $\epsilon/2$.

Supposons donc que la base b est paire. Alors $b/2$ est un chiffre entier non nul ($0 < b/2 < b$), et on peut représenter $\epsilon/2$:

$$\epsilon/2 = b/2 * b^{-p}$$

On calcule facilement $h - \epsilon/2$:

$$\begin{array}{rcl} h - \epsilon/2 & = & (a_p - b/2) * b^{-p} + h_1 \\ h_1 & = & a_{p+1} * b^{-(p+1)} + \ldots + a_{q-1} * b^{-(q-1)} \\ 0 & \leq h_1 & < b^{-p} \end{array}$$

Pour comparer h à $\epsilon/2$, il faut connaître la valeur du chiffre a_p, et si a_p vaut $b/2$, déterminer si h_1 est nul. Dans ce cas, il faut établir une règle pour choisir entre les deux arrondis possibles. En base 2, il faut savoir si le *bit d'arrondi* a_p vaut 0 ou 1, et s'il vaut 0, si le sticky bit de h_1 vaut 0 ou 1.
en base b paire :

$$\begin{array}{rcl} s_{h_1} & = & a_{p+1} + \ldots + a_{q-1} \\ h < \epsilon/2 & \Leftrightarrow & a_p < b/2 \\ h = \epsilon/2 & \Leftrightarrow & a_p = b/2 \text{ et } s_{h_1} = 0 \\ h > \epsilon/2 & \Leftrightarrow & a_p > b/2 \text{ ou } (a_p = b/2 \text{ et } s_{h_1} \neq 0) \end{array}$$

en base 2:

$$\begin{array}{rcl} s_{h_1} & = & a_{p+1} \text{ OU } \ldots \text{ OU } a_{q-1} \\ h < \epsilon/2 & \Leftrightarrow & a_p = 0 \\ h = \epsilon/2 & \Leftrightarrow & a_p = 1 \; et \; s_{h_1} = 0 \\ h > \epsilon/2 & \Leftrightarrow & a_p = 1 \; et \; s_{h_1} = 1 \end{array}$$

Lorsque x est au milieu de l'intervalle $[x_1,x_2]$, c'est-à-dire lorsque h vaut $\epsilon/2$, on utilise une règle dite de *parité* pour choisir entre les deux bornes de l'intervalle.

Puisque x_1 et x_2 diffèrent de ϵ, leurs derniers chiffres (rang p) diffèrent de 1. La règle de parité choisit comme arrondi au plus près le nombre x_1 ou x_2 dont le dernier chiffre est pair. Cette convention permet d'équilibrer l'arrondi, c'est-à-dire d'arrondir aussi souvent à droite qu'à gauche. Il faut

valeur avec 5 chiffres	arrondis avec 3 chiffres		
	par défaut	par excès	au plus près
1.0100	1.01	1.01	1.01
1.0101	1.01	1.10	1.01
1.0110	1.01	1.10	1.10
1.0111	1.01	1.10	1.10
1.1111	1.11	2*1.00	2*1.00

Table 6 : Exemples d'arrondi en base 2

donc connaître la parité du dernier chiffre de x. En base 2, il faut savoir si le p^e bit, qui est le *bit de parité*, vaut 0 ou 1.

Pour arrondir sur p bits un nombre représenté avec q bits, il faut donc connaître, en base 2 :

- arrondi par défaut ou par excès: le OU logique des bits après le p^e bit, c'est-à-dire le sticky bit de h.
- troncature: rien.
- arrondi au plus près: le $(p+1)^e$ bit, qui est le bit d'arrondi, et le OU logique des bits suivants, qui est le sticky bit de h_1, et le p^e bit, qui est le bit de parité.

Remarque : Après arrondi, il se peut que la mantisse ne soit plus normalisée, s'il y a eu propagation d'une sorte de retenue. Il faut alors *normaliser* le flottant obtenu, en divisant la mantisse par b, et en ajoutant 1 à l'exposant.

Exemple : la table 6 donne les arrondis de divers nombres à 5 chiffres, dans un système binaire à 3 chiffres. Dans certains cas, il a fallu normaliser l'arrondi.

4 Arithmétique flottante

Pour calculer sur l'ensemble $\mathcal{F}$, il faut y définir une arithmétique. En général, $\mathcal{F}$ est non clos pour les opérations arithmétiques de $\mathcal{R}$. Le résultat d'une addition de deux nombres représentables n'est pas toujours représentable. Il faut donc arrondir ce résultat. On définit des opérations sur $\mathcal{F}$ en leur imposant la propriété suivante: le résultat d'une opération $\odot$ dans $\mathcal{F}$ de deux nombres représentables est l'arrondi du résultat de l'opération correspondante $\cdot$ dans $\mathcal{R}$:

$$\forall x, y \in \mathcal{F} \; x \odot y = \Box(x \cdot y) \tag{1}$$

Définition : une opération arithmétique flottante $\odot$ est correcte si elle satisfait la propriété (1).

Remarque: cette propriété n'est en fait pas vérifiée sur tous les calculateurs (voir plus loin l'exemple du CRAY).

Il faut noter que $\mathcal{F}$ ne possède pas la structure algébrique de $\mathcal{R}$. En particulier, les opérations ne sont en général pas associatives, comme on le verra plus loin.

La norme IEEE impose que cette règle (1) s'applique aux quatre opérations usuelles (+,-,$\times$,/), ainsi qu'à la racine carrée, et à la congruence (remainder). Nous allons étudier le cas des 4 opérations usuelles.

Soit x et y deux nombres flottants non nuls, dont les représentations sont les suivantes:

$$x = (-1)^{s_x}\ b^{e_x}\ m_x \text{ avec } 1 \leq m_x < b$$
$$y = (-1)^{s_y}\ b^{e_y}\ m_y \text{ avec } 1 \leq m_y < b$$

4.1 Multiplication

Soit $\times$ la multiplication dans l'ensemble des réels $\mathcal{R}$ et $\otimes$ la multiplication dans l'ensemble des flottants $\mathcal{F}$. On a donc:

$$x \times y = (-1)^{s_x+s_y}\ b^{e_x+e_y}\ (m_x \times m_y) \text{ avec } 1 \leq (m_x \times m_y) < b^2$$

Remarque: cette opération peut signaler un overflow ($e_x + e_y > E_{max}$), ou un underflow ($e_x + e_y < E_{min}$).

Le résultat possède une mantisse avec $q = 2p$ chiffres. Il faut arrondir et normaliser le résultat de la multiplication, afin d'obtenir un nombre représentable. On distingue deux cas (voir figure 2):

- $1 \leq m_x \times m_y < b$
  Alors il suffit d'arrondir le résultat selon les règles décrites précédemment.

  $$x \otimes y \quad = \quad \Box(x \times y)$$

- $b \leq m_x \times m_y < b^2$
  Alors il faut normaliser le résultat en le divisant par b, donc en décalant les chiffres d'un rang vers la droite. On obtient une mantisse avec $q+1$ chiffres. Puis il faut arrondir le résultat à p chiffres.

  Remarque: la normalisation entraîne un nouveau risque d'overflow.

  $$x \times y \quad = \quad (-1)^{s_x+s_y}\ b^{e_x+e_y+1}\ \frac{m_x \times m_y}{b}$$
  $$x \otimes y \quad = \quad \Box(x \times y)$$

Des exemples sont donnés dans la table 7, pour un système décimal avec 3 chiffres de mantisse.

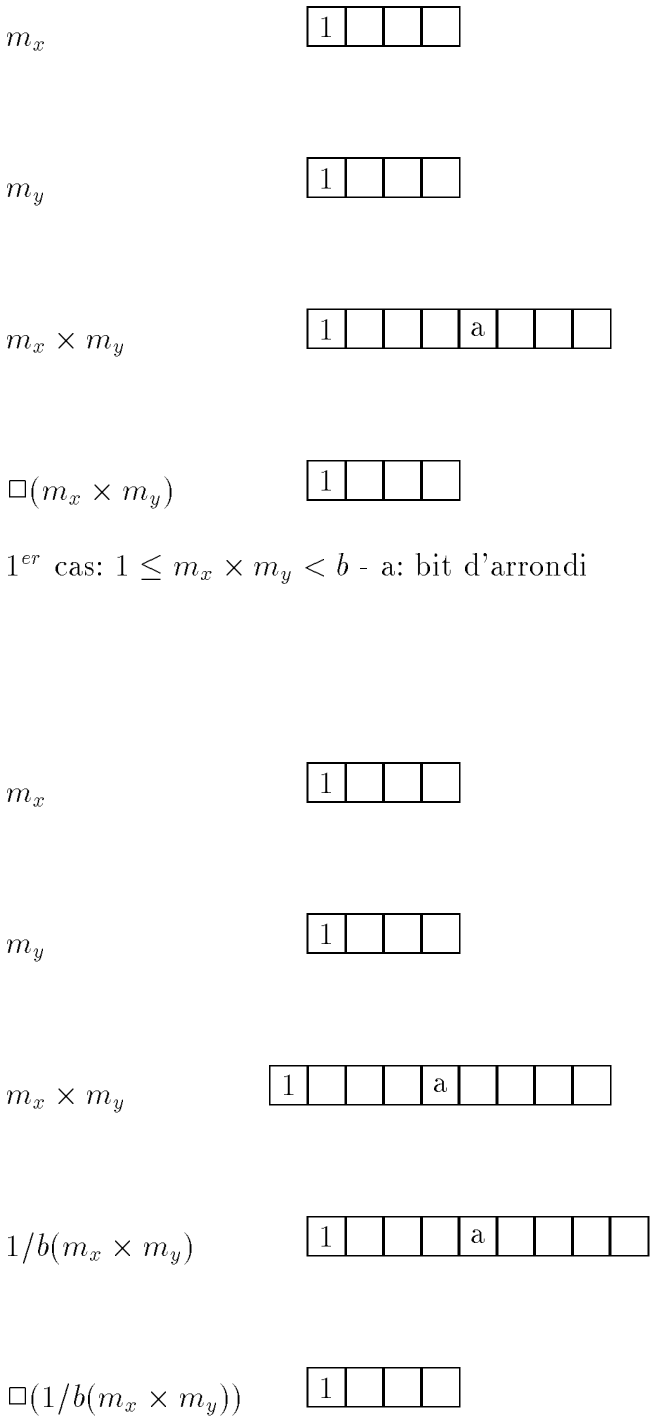


Figure 2 : Multiplication flottante en binaire avec 4 bits de mantisse

x	y	$x \times y$	$x \otimes y$
1.25	4.57	5.7125	5.71
2.50	6.33	15.825	10*1.58

Table 7 : Multiplication en base 10 avec 3 chiffres

4.2 Division

Soit / la division dans $\mathcal{R}$ et $\oslash$ la division dans $\mathcal{F}$. On a donc:

$$x/y = (-1)^{s_x+s_y}\ b^{e_x-e_y}\ (m_x/m_y) \text{ avec } 1/b \leq (m_x/m_y) < b$$

Remarque: cette opération peut signaler un overflow ($e_x - e_y > E_{max}$), ou un underflow ($e_x - e_y < E_{min}$).

Le résultat de la division possède une mantisse avec un nombre infini de chiffres. En fait, il suffira de garder un nombre fini de chiffres pour obtenir l'arrondi correct. Il faut normaliser et arrondir ce résultat, afin d'obtenir un nombre représentable. On distingue deux cas (voir figure 3):

- $1 \leq m_x/m_y < b$
  Alors il suffit d'arrondir le résultat selon les règles décrites précédemment. Il suffit donc de garder le bit d'arrondi et le sticky bit pour l'arrondi au plus près.

  $$x \oslash y \quad = \quad \square(x/y)$$

- $1/b \leq m_x \times m_y < 1$
  Alors il faut normaliser le résultat en multipliant la mantisse par b, donc en décalant les chiffres d'un rang vers la gauche. Il faut donc garder le $(p+1)^e$ chiffre de la mantisse, que l'on appelle le *bit de garde*. Puis il faut arrondir le résultat, en gardant en plus le bit d'arrondi et le sticky bit.

  Remarque: la normalisation entraîne un nouveau risque d'underflow.

  $$\begin{aligned} x/y &= (-1)^{s_x+s_y}\ b^{e_x-e_y-1}\ (b * m_x/m_y) \\ x \oslash y &= \square(x/y) \end{aligned}$$

On constate que pour effectuer correctement la division, il faut utiliser un bit de garde.

Des exemples sont donnés dans la table 8, pour un système décimal avec 3 chiffres de mantisse.

4.3 Addition de nombres de même signe

Soit + l'addition dans $\mathcal{R}$ et $\oplus$ l'addition dans $\mathcal{F}$. On suppose que $|x| \geq |y|$, donc que $e_x > e_y$ ou ($e_x = e_y$ et $m_x \geq m_y$). Il faut commencer par aligner x

m_x $\quad \boxed{1}\boxed{}\boxed{}\boxed{}$

m_y $\quad \boxed{1}\boxed{}\boxed{}\boxed{}$

m_x/m_y $\quad \boxed{1}\boxed{}\boxed{}\boxed{}\boxed{\text{a}}\boxed{\text{s}}$

$\square(m_x/m_y)$ $\quad \boxed{1}\boxed{}\boxed{}\boxed{}$

1^{er} cas: $1 \leq m_x/m_y < b$ - a: bit d'arrondi - s: sticky bit

m_x $\quad \boxed{1}\boxed{}\boxed{}\boxed{}$

m_y $\quad \boxed{1}\boxed{}\boxed{}\boxed{}$

m_x/m_y $\quad \boxed{0}\boxed{1}\boxed{}\boxed{}\boxed{\text{g}}\boxed{\text{a}}\boxed{\text{s}}$

$b(m_x/m_y)$ $\quad \boxed{1}\boxed{}\boxed{}\boxed{\text{g}}\boxed{\text{a}}\boxed{\text{s}}$

$\square(b(m_x/m_y))$ $\quad \boxed{1}\boxed{}\boxed{}\boxed{}$

2^e cas: $1/b < m_x/m_y < b$ - g:bit de garde - a: bit d'arrondi - s:sticky bit

Figure 3 : Division flottante en binaire avec 4 bits de mantisse

x	y	x/y	$x \oslash y$
10*1.25	4.57	2.6805	2.68
2.50	6.33	0.39494	3.95

Table 8 : Division en base 10 avec 3 chiffres

x	y	$x+y$	$x \oplus y$
1.25	4.57	5.82	5.82
8.50	6.33	14.83	10*1.5

Table 9 : Addition en base 10 avec 3 chiffres

et y en décalant m_y de $n = e_x - e_y$ chiffres vers la droite. On a donc:

$$\begin{aligned} m1_y &= b^{e_y - e_x}\ m_y \\ x + y &= (-1)^{s_x}\ b^{e_x}\ (m_x + m1_y) \text{ avec } 1 + b^{-n} \leq m_x + m1_y < b(1 + b^{-n}) \end{aligned}$$

Le résultat de l'addition possède une mantisse avec $q = p + n$ chiffres. Il faut normaliser et arrondir ce résultat, afin d'obtenir un nombre représentable. On distingue deux cas (voir figure 4):

- $1 + b^{-n} \leq m_x + m1_y < b$
  Alors il suffit d'arrondir le résultat selon les règles décrites précédemment.

  $$x \oplus y = \square(x + y)$$

- $b \leq m_x + m1_y < b\ (1 + b^{-n})$
  Alors il faut normaliser le résultat en divisant la mantisse par b, donc en décalant les chiffres d'un rang vers la droite. On obtient une mantisse avec $q + 1$ chiffres. Il suffit alors d'arrondir le résultat.

  Remarque: la normalisation entraîne un risque d'overflow.

  $$\begin{aligned} x + y &= (-1)^{s_x}\ b^{e_x + 1}\ \frac{m_x + m1_y}{b} \\ x \oplus y &= \square(x + y) \end{aligned}$$

Des exemples sont donnés dans la table 9, pour un système décimal avec 3 chiffres de mantisse.

4.4 Addition de nombres de signes contraires

Additionner deux nombres de signes contraires revient à soustraire leurs valeurs absolues. Soit $-$ la soustraction dans $\mathcal{R}$ et $\ominus$ la soustraction dans $\mathcal{F}$.

m_x	1						
$m1_y$	0	0	1				
$m_x + m1_y$	1				a		
$\Box(m_x + m1_y)$	1						

1^{er} cas: $1 \leq m_x + m1_y < b$ - a: bit d'arrondi

m_x		1						
$m1_y$		0	0	1				
$m_x + m1_y$	1				a			
$1/b(m_x + m1_y)$		1				a		
$\Box(1/b(m_x \times m_y))$		1						

2^e cas: $b \leq m_x + m1_y < b^2$ - a: bit d'arrondi

Figure 4 : Addition flottante en binaire avec 4 bits de mantisse

On suppose que $|x| \geq |y|$, donc que $e_x > e_y$ ou ($e_x = e_y$ et $m_x \geq m_y$). Il faut commencer par aligner x et y en décalant m_y de $n = e_x - e_y$ chiffres vers la droite. On a donc:

$$\begin{aligned} m1_y &= b^{e_y - e_x}\ m_y \\ x - y &= (-1)^{s_x}\ b^{e_x}\ (m_x - m1_y) \text{ avec } 1 - b^{1-n} < m_x - m1_y < b - b^{-n} \end{aligned}$$

Le résultat de la soustraction possède une mantisse avec $q = p+n$ chiffres. Il faut normaliser et arrondir ce résultat, afin d'obtenir un nombre représentable. On distingue trois cas, suivant la valeur de n (voir figures 5,6,7):

- $n = 0$
  Alors soit $m_x = m_y$, et on obtient le résultat exact 0, soit $m_x > m_y$. Dans ce cas, il faut décaler le résultat de $k < p$ chiffres vers la gauche, en complétant à droite par des 0, puisque la mantisse du résultat a p chiffres. On obtient là aussi le résultat exact.

  Remarque: en fait, le résultat est exact à la précision machine. En fait, si x et y sont eux-mêmes des résultats de calcul, donc des arrondis de nombres réels, alors les zéros que l'on rajoute à droite sont en réalité des chiffres non nuls, perdus dans les arrondis précédant la soustraction. On étudiera plus loin les erreurs d'arrondi liées à ce phénomène.

$$x \ominus y \quad = \quad x - y$$

- $n = 1$
  Alors soit $1 \leq m_x - m1_y < b$, et il faut arrondir le résultat en utilisant le $(p+1)^e$ bit, soit $0 < m_x - m1_y < 1$, et il faut normaliser le résultat en multipliant la mantisse par b^k, avec $k < p$, donc décaler le résultat de k chiffres vers la gauche. Il faut alors garder le $(p+1)^e$ chiffre, qui est le bit de garde, et compléter par des 0, puisque le résultat a $p+1$ chiffres. On obtient dans ce cas le résultat exact, dans la mesure où x et y sont exacts.

$$x \ominus y \quad = \quad \Box(x - y)$$

- $n \geq 2$
  Dans ce cas, $1 - b^{1-n} \geq 1 - b^{-1} \geq b^{-1}$, d'où $b^{-1} < m_x + m1_y < b$. Il faut éventuellement normaliser le résultat en multipliant la mantisse par b, donc en décalant d'au plus un chiffre vers la gauche, et en utilisant le bit de garde. On obtient alors une mantisse avec q ou $q-1$ chiffres, qu'il suffit d'arrondir.

$$x \ominus y \quad = \quad \Box(x - y)$$

On constate que pour effectuer correctement la soustraction, il est nécessaire d'utiliser un bit de garde.

Des exemples sont donnés dans la table 10, pour un système décimal avec 3 chiffres de mantisse.

m_x	1			
$m1_y$	1			
$m_x - m1_y$	0	0	0	0

1^{er} cas: $n = 0$ et $m_x = m1_y$ - résultat nul

m_x	1			
$m1_y$	1			
$m_x - m1_y$	0	0	0	1
$b^3(m_x - m1_y)$	1	0	0	0
$\square[b^3(m_x - m1_y)]$	1	0	0	0

2^e cas: $n = 0$ et $1/b^k \leq m_x - m1_y < 1/b^{(k-1)}$ - (ici k=3) - résultat exact

Figure 5 : Soustraction flottante en binaire avec 4 bits de mantisse (cas n=0)

m_x	1				
$m1_y$	0	1			
$m_x - m1_y$	1				a
$\Box(m_x - m1_y)$	1				

1^{er} cas: $n = 1$ et $1 \leq m_x - m1_y < b$ - a: bit d'arrondi

m_x	1				
$m1_y$	0	1			
$m_x - m1_y$	0	0	1		g
$b^2(m_x - m1_y)$	1		g	0	
$\Box(b^2(m_x - m1_y))$	1		g	0	

2^e cas: $n = 1$ et $1/b^k < m_x - m1_y < b$ - (ici k=2) - g: bit de garde - résultat exact

Figure 6 : Soustraction flottante en binaire avec 4 bits de mantisse (cas n=1)

m_x	1					
$m1_y$	0	0	1			
$m_x - m1_y$	1				a	
$\square(m_x - m1_y)$	1					

1^{er} cas: $n \geq 2$ (ici $n = 2$) et $1 \leq m_x - m1_y < b$ - a: bit d'arrondi

m_x	1					
$m1_y$	0	0	1			
$m_x - m1_y$	0	1			g	a
$b(m_x - m1_y)$	1			g	a	
$\square(b(m_x - m1_y))$	1			g		

2^e cas: $n \geq 2$ (ici $n = 2$) et $1/b < m_x - m1_y < 1$ - g: bit de garde - a: bit d'arrondi

Figure 7 : Soustraction flottante en binaire avec 4 bits de mantisse (cas $n \geq 2$)

x	y	$x - y$	$x \ominus y$
8.50	6.33	2.17	2.17
8.50	8.43	0.07	(1/100)*7.00
8.50	0.763	7.737	7.74
1.50	0.763	0.737	(1/10)*7.37
1.50	0.006	1.494	1.49
1.00	0.006	0.994	9.94

Table 10 : Soustraction en base 10 avec 3 chiffres

4.5 Exemple d'arithmétique incorrecte

Les calculateurs qui respectent la norme IEEE effectuent les opérations arithmétiques correctement. Par contre, si le calculateur n'utilise pas par exemple de bit de garde, les résultats peuvent être faux et conduire à des erreurs assez graves.

Par exemple, l'arithmétique du CRAY n'est pas conforme à la norme. Les nombres sont codés en base 2 sur 64 bits, avec 15 bits d'exposant et 48 bits de mantisse, sans premier bit implicite; la mantisse est comprise entre 1 et 2. Aucune des opérations n'est arrondie correctement, ce qui laisse supposer qu'il n'y a pas utilisation de sticky bit ou de bit d'arrondi. Mais ce qui est plus grave est une division incorrecte et l'absence de bit de garde dans la soustraction. La division est implémentée comme la multiplication par l'inverse, ce qui introduit deux erreurs d'arrondi consécutives.

Si l'on convertit dans un entier le résultat de la division des deux flottants 14. et 7., on obtient... 1! Il faut donc être prudent lorsqu'on convertit des flottants dans des entiers. En effet, le résultat de la division est $2 - \epsilon$, qui est converti à l'entier inférieur 1.

Si l'on compare $2 - \epsilon$ à 2, on constate bien que les deux flottants sont différents. Mais si on les soustrait, on obtient... 0!.. au lieu de ϵ qui est pourtant représentable en machine. Cette incohérence peut conduire à des divisions par 0., alors que le programme teste si le diviseur est non nul!

5 Propriétés algébriques de l'arithmétique flottante

Il est utile de se demander quelles propriétés algébriques de $\mathcal{R}$ sont conservées dans $\mathcal{F}$. L'absence de certaines propriétés induit une accumulation des erreurs d'arrondi dans une suite de calculs qui peut être importante. Dans ce qui suit, on suppose que l'arithmétique de $\mathcal{F}$ est conforme à la norme.

5.1 Multiplication et division

La multiplication dans $\mathcal{F}$ est commutative:

$$\forall x, y \in \mathcal{F} \quad x \otimes y = \Box(x \times y) = \Box(y \times x) = y \otimes x$$

1 est élément neutre de la multiplication:

$$\forall x \in \mathcal{F} \quad x \otimes 1 = \Box(x \times 1) = \Box(x) = x$$

Par contre, il existe des flottants qui ne possèdent pas d'inverse flottant.

$$\begin{array}{lll} \forall x \in \mathcal{F} & 1 \oslash x & = 1/x \ (1+\alpha) \ \text{ avec } \ |\alpha| < \epsilon \\ & x \otimes (1 \oslash x) & = \Box(x \times 1/x \ (1+\alpha)) = \Box(1+\alpha) = 1 \ \text{ ou } \ \neq 1 \end{array}$$

Enfin, la multiplication dans $\mathcal{F}$ n'est pas associative:

$\forall x, y, z \in \mathcal{F}$

$$\left\{ \begin{array}{l} (x \otimes y) \otimes z = \Box(\Box(x \times y) \times z) = (x \times y \times z) \ (1+\alpha) \ (1+\beta) \\ \text{avec } \ |\alpha| < \epsilon \text{ et } |\beta| < \epsilon \end{array} \right.$$

$$\left\{ \begin{array}{l} x \otimes (y \otimes z) = \Box(x \times \Box(y \times z)) = (x \times y \times z) \ (1+\gamma) \ (1+\delta) \\ \text{avec } \ |\gamma| < \epsilon \text{ et } |\delta| < \epsilon \end{array} \right.$$

5.2 Addition et soustraction

L'addition dans $\mathcal{F}$ est commutative:

$$\forall x, y \in \mathcal{F} \quad x \oplus y = \Box(x + y) = \Box(y + x) = y \oplus x$$

0 est élément neutre de la multiplication:

$$\forall x \in \mathcal{F} x \oplus 0 = \Box(x + 0) = \Box(x) = x$$

Tout flottant a un opposé qui est flottant:

$$\forall x \in \mathcal{F} \quad (-x) \in F \ et \ x \oplus (-x) = \Box(x + (-x)) = \Box(0) = 0$$

L'addition flottante n'est pas associative:

$\forall x, y, z \in \mathcal{F}$

$$\left\{ \begin{array}{l} (x \oplus y) \oplus z = \Box(\Box(x + y) + z) = x + y + z + \alpha + \beta \\ \text{avec } \ |\alpha| < \epsilon |x + y| \ \text{ et } \ |\beta| < \epsilon |x + y + z + \alpha| \end{array} \right.$$

$$\left\{ \begin{array}{l} x \oplus (y \oplus z) = \Box(x + \Box(y + z)) = (x + y + z + \gamma + \delta \\ \text{avec } \ |\gamma| < \epsilon |y + z| \ \text{ et } \ |\delta| < \epsilon |x + y + z + \gamma| \end{array} \right.$$

La multiplication n'est pas distributive par rapport à l'addition.

5.3 Erreurs relatives sur la multiplication et la division

Soit x un réel représenté par le flottant $\overline{x}$ avec une erreur relative α et y un réel représenté par le flottant $\overline{y}$ avec une erreur relative β. On a alors une erreur relative totale γ sur le produit composée de l'erreur d'arrondi δ et des erreurs sur x et y:

$$\begin{aligned}
\overline{x} &= x(1+\alpha) \\
\overline{y} &= y(1+\beta) \\
\overline{x} \otimes \overline{y} &= \overline{x} \times \overline{y}(1+\delta) \\
&= (x \times y)(1+\alpha)(1+\beta)(1+\gamma)
\end{aligned}$$

Au premier ordre (on néglige les produits du type $\alpha \times \beta$), on obtient une erreur relative γ sur le produit $x \times y$ qui est la somme des erreurs relatives sur x et y et de l'erreur relative d'arrondi.

$$\gamma = \alpha + \beta + \delta$$

Remarque : on obtient un résultat similaire pour la division.

5.4 Erreurs relatives sur l'addition et la soustraction

Les notations sont les mêmes que précédemment. L'addition est au sens large, les signes de x et y étant quelconques.

$$\begin{aligned}
\overline{x} \oplus \overline{y} &= [x(1+\alpha) + y(1+\beta)](1+\delta) \\
&= (x+y)[1 + \delta + \frac{\alpha x + \beta y}{x+y}(1+\delta)] \\
\gamma &= \delta + \frac{\alpha x + \beta y}{x+y}(1+\delta)]
\end{aligned}$$

Si x et y sont de même signe (addition au sens propre), on peut au premier ordre majorer l'erreur relative γ sur la somme $x+y$ par la somme des erreurs relatives sur x et sur y et de l'erreur relative d'arrondi.

$$|\gamma| \leq |\alpha| + |\beta| + |\delta|$$

Par contre, si x et y sont de signes contraires (soustraction au sens propre), alors $x + y$ peut être petit devant αx et αy, et l'erreur relative γ peut être grande. Ceci se produit lorsque x et y sont proches en valeur absolue. Cette soustraction de deux termes proches provoque le phénomène d'élimination catastrophique, décrit plus loin.
Exemple : Considérons le système flottant suivant:

$$\begin{aligned}
b &= 10 \\
p &= 3 \\
E_{min} &= -3 \\
E_{max} &= +3
\end{aligned}$$

Soit les nombres flottants:

$$\begin{aligned}
\overline{x} = \quad & 1.58\ 10^2 = \quad 158 \\
\overline{y} = \quad & -1.52\ 10^2 = \quad -152 \\
\overline{x} \oplus \overline{y} \quad & = 0.06\ 10^2 \quad = 6
\end{aligned}$$

Si $\overline{x}$ et $\overline{y}$ sont les arrondis respectivement de x et y on peut avoir:

$$\begin{aligned}
x &= 157.51 \\
y &= -152.49 \\
x + y &= 5.02
\end{aligned}$$

ou bien encore :

$$\begin{aligned}
x &= 158.48 \\
y &= -151.62 \\
x + y &= 6.86
\end{aligned}$$

Les erreurs relatives sur x et y sont de l'ordre de 1/100, alors que l'erreur relative sur $x + y$ est dans les deux cas de l'ordre de 1.

Part II
Difficultés numériques

6 Erreurs d'arrondi

Chaque opération arithmétique introduit une erreur d'arrondi. Les erreurs relatives s'ajoutent dans le cas d'une multiplication et d'une division, mais ce sont les erreurs absolues qui s'ajoutent dans le cas d'une addition et d'une soustraction. Dans une suite de calculs, l'accumulation de ces erreurs peut finir par être non négligeable devant le résultat final. Deux phénomènes, qui se produisent en additionnant deux nombres, sont particulièrement importants. Ce sont *l'absorption* et *l'élimination*.

6.1 Absorption

Lorsqu'on additionne (au sens large, les deux nombres étant de signes quelconques), deux nombres d'ordre de grandeur très différents, il peut arriver que le résultat soit tout simplement le plus grand des deux nombres. En effet, en alignant les exposants, on peut perdre tous les chiffres significatifs du plus petit nombre. Pour une seule addition, ce n'est pas grave, car le petit nombre est effectivement négligeable devant le grand. Mais c'est l'accumulation de ce phénomène qui provoque des erreurs importantes. En effet, l'addition de plusieurs termes négligeables peut devenir non négligeable.

Dans ce qui suit, on considère un système flottant en base b, avec arrondi au plus près. Pour simplifier, on suppose que l'arrondi du milieu d'un intervalle flottant peut être indifféremment l'une ou l'autre des bornes de l'intervalle. La précision machine est notée ϵ.

Soit x et y deux nombres flottants, et soit e_x l'exposant de x. Le résultat de l'addition flottante $x \oplus y$ vaut x dès que y est petit devant x. Plus précisément, on a le résultat:

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

En effet (voir figure 8):

$$\begin{aligned} \forall z \in \mathcal{R} \ \ \Diamond(x+z) = x \quad &\Leftrightarrow \quad |z| \leq b^{e_x}\ \epsilon/2 \text{ et} \\ x \oplus y = x \quad &\Leftrightarrow \quad \Diamond(x+y) = x \end{aligned}$$

Cette propriété conduit à la définition suivante:

Définition: Soit deux réels x et y. Il y a *absorption* de y par x si:

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

où ϵ est la précision machine.

$$x \in \mathcal{F} \quad x > 0 \quad \text{et} \quad x_1 \in \mathcal{F} \quad x_1 = x - b^{e_x}\ \epsilon \quad \text{et} \quad x_2 \in \mathcal{F} \quad x_2 = x + b^{e_x}\ \epsilon$$

$$y_1 \in \mathcal{R} \quad y_1 = -b^{e_x}\ \epsilon/2 \quad \text{et} \quad y_2 \in \mathcal{R} \quad y_2 = b^{e_x}\ \epsilon/2$$

$$\forall y \in \mathcal{R} \quad x \oplus y = x \Leftrightarrow \Diamond(x+y) = x \Leftrightarrow y_1 \leq y \leq y_2$$

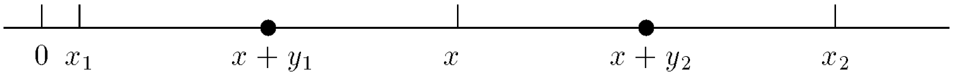


Figure 8 : Intervalle d'absorption autour de x

Remarque: Si y est absorbé par x, l'erreur absolue sur l'addition (au sens large) $x + y$ est de l'ordre de l'erreur absolue sur x. Or l'erreur sur x est une erreur d'arrondi, au moins de l'ordre de $|x|\epsilon/2$ (cette erreur peut être plus grande si x est le résultat de plusieurs opérations flottantes). Si y est absorbé par x, y est donc du même ordre que l'erreur absolue sur $x + y$, et le résultat $x + y$ est environ égal à x. On retrouve donc pratiquement le résultat précédent sur les flottants.

Exemple: en base 10, avec 4 chiffres significatifs.

$$\begin{aligned} (1.000 + 5\ 10^{-4}) + 5\ 10^{-4} &= 1.000 \\ 1.000 + (5\ 10^{-4} + 5\ 10^{-4}) &= 1.001 \end{aligned}$$

On constate sur cet exemple que le résultat d'une suite d'additions dépend de l'ordre des calculs. En effet, l'addition flottante n'est pas associative. Il faut donc choisir un ordre de calcul qui minimise les erreurs d'arrondi, en évitant les phénomènes d'absorption.

6.2 Exemple d'absorption: calcul de la série harmonique

On cherche à calculer la somme (exemple dû à J.M. Muller):

$$s = \sum_{i=1}^{n} 1/i$$

n	10 000	1 000 000	10 000 000	100 000 000
indices croissants	9.787613	14.35736	15.40368	15.40368
indices décroissants	9.787604	14.39265	16.68603	18.80792
valeur exacte	9.787606	14.39273	16.69531	18.99789

Table 11 : Calcul de la série harmonique

pour n grand. La première idée est de calculer la somme dans l'ordre croissant des indices, donc d'effectuer l'algorithme:

$$\begin{array}{l} s := 0 \\ \text{pour } i = 1 \text{ à } n \text{ faire } s := s + 1/i \end{array}$$

Mais on commence alors par les plus grands termes, et les plus petits termes sont absorbés par la somme partielle déjà calculée. On modifie donc l'algorithme pour effectuer la sommation en ordre inverse:

$$\begin{array}{l} s := 0 \\ \text{pour } i = n \text{ à } 1 \text{ faire } s := s + 1/i \end{array}$$

La table 11 donne les valeurs calculées de cette série harmonique pour différentes valeurs de n, ainsi que les valeurs exactes. On constate effectivement que le second algorithme donne de bien meilleurs résultats.

Il est toutefois possible de faire encore mieux. La perte de précision est due au fait que 1/i est petit devant s, même pour la sommation en ordre inverse. Après alignement, les derniers chiffres de 1/i sont donc absorbés dans l'addition. Mais si l'on effectue plusieurs sommes partielles, que l'on ajoute en fin de calcul, on aura des sommes intermédiaires plus petites, et donc des termes 1/i moins négligeables en quelque sorte. On peut par exemple calculer, en sommant toujours dans l'ordre inverse

$$\begin{array}{rcl} s_0 & = & 1/m \ + \ 1/2m \ + \ 1/3m \ + \ldots \\ s_1 & = & 1/1 \ + \ 1/(m+1) \ + \ 1/(2m+1) \ + \ldots \\ s_i & = & 1/i \ + \ 1/(m+i) \ + \ 1/(2m+i) \ + \ldots \\ s_{m-1} & = & 1/(m-1) \ + \ 1/(2m-1) \ + \ldots \\ s & = & s_0 \ + \ s_{m-1} \ + \ \ldots + \ s_1 \end{array}$$

ce qui se traduit par l'algorithme:

$$\begin{array}{l} \text{pour } i = n \text{ à } 1 \text{ faire } s_{i \bmod m} := s_{i \bmod m} + 1/i \\ s := s_0 \\ \text{pour } i = m-1 \text{ à } 1 \text{ faire } s := s + s_i \end{array}$$

Bien que cet algorithme soit un peu plus complexe, il effectue le même nombre d'opérations flottantes, et prend donc à peu près le même temps de calcul. La table 12 présente les valeurs obtenues pour m=16 et m=32.

n	10 000	1 000 000	10 000 000	100 000 000
m=16	9.787605	14.39274	16.69538	19.00054
m=32	9.787606	14.39272	16.69531	18.99692
valeur exacte	9.787606	14.39273	16.69531	18.99789

Table 12 : Calcul de la série harmonique par paquets

Cette table montre immédiatement le gain en précision avec cette sommation. Dans le cas n=10 000 000, avec le même nombre d'opérations (dix millions), on obtient un résultat où tous les chiffres sont significatifs, alors que dans le premier cas (sommation dans l'ordre croissant des indices), on n'obtenait qu'un chiffre significatif.

De plus, dans le premier cas, le calcul semble montrer que la série est convergente!

6.3 Elimination

Ce phénomène se produit lorsqu'on soustrait deux nombres très voisins. Il faut alors renormaliser le résultat en ajoutant des 0 comme chiffres de droite, car les chiffres de gauche s'éliminent. Or les opérandes sont en général des résultats d'opérations précédentes qui ont été arrondis. La différence des résultats exacts ferait apparaître après renormalisation des chiffres de droite non nuls, au lieu des 0. La valeur calculée peut alors être très différente de la valeur exacte. On est dans le cas où l'on obtient une grande erreur relative sur la soustraction.

Définition:Soit x et y deux réels, et ϵ la précision machine. Il y a *élimination* de x et y si

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

Remarque: L'erreur absolue sur la somme (au sens large) $x + y$ est voisine de la somme des erreurs absolues sur x et sur y. Or celles-ci sont au moins de l'ordre d'un arrondi, c'est-à-dire $\epsilon/2\ |x|$ et $\epsilon/2\ |y|$. Donc l'erreur absolue sur $x + y$ est de l'ordre de $\epsilon/2\ (|x| + |y|)$. S'il y a élimination de x et y, l'erreur relative sur la somme $x+y$ est en général supérieure ou égale à 1.

Exemples: en base 10, avec 4 chiffres significatifs.

$$\begin{aligned} (1 + 5\ 10^{-4}) - (1 - 5\ 10^{-4}) &= 0 \\ (1 - 1) + (5\ 10^{-4} + 5\ 10^{-4}) &= 10^{-3} \end{aligned}$$

Là encore, on voit que l'ordre des opérations influe sur le résultat, conséquence de la non-associativité de l'addition.

$$
\begin{aligned}
1.00151 - 1.00149 &= 0.00002 \\
1.0024 - 1.0013 &= 0.0011 \\
1.002 - 1.001 &= 0.001
\end{aligned}
$$

Ces exemples montrent qu'une élimination peut ou non entraîner une grande erreur de calcul, suivant la combinaison des erreurs d'arrondi sur les opérandes. Dans les deux cas, il y a élimination, mais l'erreur relative sur la soustraction est de 49 dans le premier cas, et de 0.1 environ dans le second cas.

Cas particulier: Soit x et y deux réels différents, mais dont les arrondis $\overline{x}$ et $\overline{y}$ sont égaux. Alors il y a élimination de x et $-y$. En effet:

$$
\begin{array}{lll}
x - y & = \overline{x} - \overline{y} + (x - \overline{x}) - (y - \overline{y}) & = (x - \overline{x}) - (y - \overline{y}) \\
|x - y| & \leq |x - \overline{x}| + |y - \overline{y}| & \leq \epsilon/2\ (|x| + |y|)
\end{array}
$$

6.4 Exemple d'élimination: résolution d'une équation du second degré

Considérons l'équation du second degré suivante :

$$x^2 - 200\ x + 10000 \quad = \quad 0$$

Cette équation a une racine double, qui vaut 100. Si on la résout sur une machine avec le système flottant suivant: base 10, nombre de chiffres 3, exposant de 0 à 4, on obtient le résultat correct:

$$
\begin{array}{l}
\Delta = (2\ 10^2)^2 - 4 * 10^4 = 4\ 10^4 - 4\ 10^4 = 0 \\
x_1 = x_2 = 100
\end{array}
$$

Considérons maintenant une équation légèrement differente:

$$x^2 - 205\ x + 10500 \quad = \quad 0$$

Cette équation a cette fois deux racines réelles disctinctes, qui valent 100 et 105. Mais avec la même machine que précédemment, on obtient un résultat faux, car on trouve un discriminant nul:

$$
\begin{array}{l}
\Delta = (2.05\ 10^2)^2 - 4 * 1.05\ 10^2 = 4.20\ 10^4 - 4.20\ 10^4 = 0 \\
x_1 = x_2 = 100
\end{array}
$$

Par contre, si on utilise un système plus précis, avec 5 chiffres, on retrouve le résultat exact:

$$
\begin{array}{l}
\Delta = (2.05\ 10^2)^2 - 4 * 1.05\ 10^2 = 4.2025\ 10^4 - 4.20\ 10^4 = 0.0025\ 10^4 = 25 \\
x_1 = (205 - 5)/2 = 100\ et\ x_2 = (205 + 5)/2 = 105
\end{array}
$$

Dans cet exemple, augmenter la précision des calculs a permis de calculer de manière exacte le discriminant, sans phénomène d'élimination.

6.5 Elimination globale

Lorsqu'on additionne (au sens large, avec des signes quelconques) plusieurs nombres, le résultat peut être petit devant la plus grande valeur absolue. Autrement dit, les chiffres de gauche des opérandes s'éliminent globalement, et on obtient des zéros à gauche. Les chiffres significatifs sont donc les chiffres de droite. Or, les opérandes sont des résultats approchés de calcul, et ces chiffres de droite sont perdus ou incorrects après arrondi. Après normalisation du résultat final, on obtient des chiffres de droite en général faux. L'erreur relative sur le résultat risque d'être très grande. Ce phénomène d'élimination globale est l'extension à plusieurs nombres du phénomène d'élimination entre deux nombres.

Définition: Soit x_i (i=1,n) n nombres réels, et soit $s = \sum_{i=1}^{i=n} x_i$ la somme de ces nombres. Il y a élimination globale des x_i si

$$|s| \leq \epsilon/2 \sum_{i=1}^{i=n} |x_i|$$

Remarque: l'erreur relative sur le résultat est très souvent supérieure à 1 après élimination globale des opérandes.

Exemple: en base 10, avec 4 chiffres significatifs.

$$\begin{aligned} 31.5364 - 17.2456 + 5.3643 - 19.6547 &= 0.0004 \\ 31.536 - 17.246 + 5.364 - 19.655 &= -0.001 \end{aligned}$$

L'erreur relative est dans cet exemple égale à 3.5! Autrement dit, le résultat du calcul n'a aucun sens.

6.6 Exemple d'élimination globale: calcul d'une exponentielle par une série alternée

Cet exemple est dû à U.W. Kulisch et W.L. Miranker.

On sait que l'exponentielle de tout nombre complexe est la somme d'une série entière :

$$\forall z \in \mathcal{C} \quad e^z = \sum_{n=0}^{n=\infty} z^n/n!$$

Si z est un réel négatif, la série est alternée. La valeur absolue de l'erreur commise en tronquant la série à N termes est inférieure à la valeur absolue du terme d'indice (N+1). On peut donc penser que l'on peut évaluer e^z en calculant une série tronquée.

Effectuons ce calcul pour z=-20, sur un SUN en simple précision IEEE. La valeur exacte de e^{-20} est 2.06115... 10^{-9}. La précision relative des calculs est $\epsilon = 2^{-23} \simeq 10^{-7}$. On utilise l'algorithme suivant:

```
z:=20
N:=100
x:=1
s:=1
a:=0.000001
pour n=1 a N tant que |x|>a*s faire
  x:=-x*z/n
  s:=s+x
fin faire
```

Et on obtient les résultats de la figure 9.

On obtient un résultat de l'ordre de 10^{-1} au lieu de 10^{-9}! Ceci est dû aux arrondis sur les calculs de x, qui rendent les soustractions imprécises.

La plus grande valeur de x est pour $n = 20$, et vaut 4.30... 10^7. L'erreur absolue sur cette valeur est de l'ordre de 1, beaucoup plus grande que la valeur de la somme finale! En outre, l'ordre des calculs ne change rien à cet état de fait, l'erreur est due ici aux arrondis, et non à la non-associativité.

Il faudrait donc beaucoup plus de précision pour calculer de manière exacte cette série alternée. Il faudrait au moins 16 (9+7) chiffres pour obtenir un chiffre correct dans la somme.

6.7 Exemple d'élimination et d'absorption: calcul de (x+y-x)/y

Soit x et y deux réels et soit E l'expression

$$E \quad = \quad (x+y-x)/y$$

On cherche à calculer E avec un système flottant, dans le cas où y est petit devant x, et à comparer le résultat $\overline{E}$ avec la valeur exacte 1.

On note $\overline{x}$ et $\overline{y}$ les représentations flottantes (arrondis) de x et y.

On remarque que l'ordre des opérations a ici encore une grande importance. En effet, si on effectue le calcul

$$\overline{E} \quad = \quad [(\overline{x} \ominus \overline{x}) \oplus \overline{y}] \oslash \overline{y}$$

On obtient le résultat exact à l'arrondi machine près (c'est-à-dire $|\overline{E} - 1| \leq \epsilon/2$).

Considérons maintenant le cas où on effectue le calcul

$$\overline{E} \quad = \quad [(\overline{x} \oplus \overline{y}) \ominus \overline{x}] \oslash \overline{y}$$

Cette fois, le résultat peut être très différent de 1! Pour analyser les calculs, on va distinguer 3 cas:

- absorption de y par x ($|y| \leq |x|\ \epsilon/2$.
  On suppose en fait que:

  $$|\overline{y}| \quad \leq \quad b^{e_{\overline{x}}}\ \epsilon/2$$

  d'où

  $$\begin{aligned} \overline{x} \oplus \overline{y} \quad &= \quad \overline{x} \\ \text{donc } \overline{E} \quad &= \quad 0 \end{aligned}$$

```
  n         x
-----------------------
  0           1.000000
  1         -20.000000
  2         200.000000
  3       -1333.333370
  4        6666.666990
  5      -26666.668000
  6       88888.890600
  7     -253968.266000
  8      634920.688000
  9    -1410934.880000
 10     2821869.750000
 11    -5130672.500000
 12     8551121.000000
 13   -13155571.000000
 14    18793672.000000
 15   -25058230.000000
 16    31322788.000000
 17   -36850340.000000
 18    40944824.000000
 19   -43099816.000000
 20    43099816.000000
 21   -41047444.000000
 22    37315860.000000
 23   -32448574.000000
 24    27040478.000000
 25   -21632382.000000
 26    16640294.000000
 27   -12326144.000000
 28     8804389.000000
 29    -6071992.500000
 30     4047995.000000
 31    -2611609.750000
 32     1632256.130000
 33     -989246.125000
 34      581909.500000
 35     -332519.719000
 36      184733.172000
 37      -99855.765600
 38       52555.668000
 39      -26951.625000
 40       13475.812500
 41       -6573.566900
 42        3130.270020
 43       -1455.939580
 44         661.790711
 45        -294.129211
 46         127.882263
 47         -54.417984
 48          22.674160
 49          -9.254759
 50           3.701904
 51          -1.451727
 52           0.558357
 53          -0.210701
 54           0.078037
 55          -0.028377
 56           0.010135
 57          -0.003556
 58           0.001226
 59          -0.000416
 60           0.000139
 61          -0.000045
 62           0.000015
 63          -0.000005
 64           0.000001
-----------------------
exp(-20) =    0.109833
```

Figure 9 : Calcul de e^{-20} par une série alternée

- élimination de $x+y$ et de $-x$ ($|y| \leq (|x+y|+|x|)\ \epsilon/2$).
  On suppose en fait que:

$$b^{e_{\overline{x}}}\ \epsilon/2 < |\overline{y}| \quad < \quad b^{e_{\overline{x}}}\ \epsilon$$

  d'où

$$\begin{aligned} \overline{x} \oplus \overline{y} &= \overline{x} \pm b^{e_{\overline{x}}}\ \epsilon \\ (\overline{x} \oplus \overline{y}) \ominus \overline{x} &= \pm b^{e_{\overline{x}}}\ \epsilon \\ 1 < b^{e_{\overline{x}}}\ \epsilon/|\overline{y}| &< 2 \\ 1-\epsilon/2 < \overline{E} &< 2+\epsilon/2 \end{aligned}$$

- transition vers l'élimination de $x+y$ et de $-x$ ($|y| > |x|\ \epsilon$).
  On suppose en fait que

$$\begin{aligned} b^{e_{\overline{x}}}\ \epsilon \leq |\overline{y}| &< b^{e_{\overline{x}}}\ (b - m_{\overline{x}}) \\ \text{d'où } (\overline{x} \oplus \overline{y}) \ominus \overline{x} &= \overline{y} + \alpha \text{ avec } |\alpha| \leq b^{e_{\overline{x}}}\ \epsilon/2 \\ \overline{E} &= [(\overline{y}+\alpha)/\overline{y}]\ (1+\beta) \text{ avec } |\beta| \leq \epsilon/2 \\ \overline{E} &= 1+\gamma \text{ avec } |\gamma| \leq (0.5+\epsilon) \end{aligned}$$

La figure 10 montre la courbe donnant $\overline{E}$ en fonction de y pour $x = 1$, obtenue en calculant E en simple précision. On observe des discontinuités aux points $(2k+1)\epsilon/2$ ($k = 0, 1, \ldots$), qui proviennent des discontinuités de l'arrondi de $(1+y)$ en ces points.

6.8 Exemple d'élimination et d'absorption: dérivation approchée d'une fonction

Une idée naturelle pour calculer de manière approchée la dérivée d'une fonction est d'utiliser le taux d'accroissement. Mais a-t-on toujours la notion de limite sur ordinateur?

Mathématiquement, on a, pour une fonction f dérivable en a:

$$\begin{aligned} f'(a) &= \lim_{h \to 0} f_h(a) \\ \text{où } f_h(a) &= \frac{f(a+h)-f(a)}{h} \end{aligned}$$

L'erreur globale sur le résultat $f'(a)$ est la combinaison de l'erreur d'approximation $|f_h(a) - f'(a)|$ et de l'erreur de calcul $|\overline{f}_h(a) - f_h(a)|$, où $\overline{f}_h(a)$ est la valeur calculée.

$$|\overline{f}_h(a) - f'(a)| \leq |f_h(a) - f'(a)| + |\overline{f}_h(a) - f_h(a)|$$

L'erreur d'approximation tend vers 0 quand h tend vers 0. Mais comment se comporte l'erreur de calcul? En fait, cette expression est du même type que l'expression E de l'exemple précédent. Quand h tend vers 0, on a donc 4 cas:

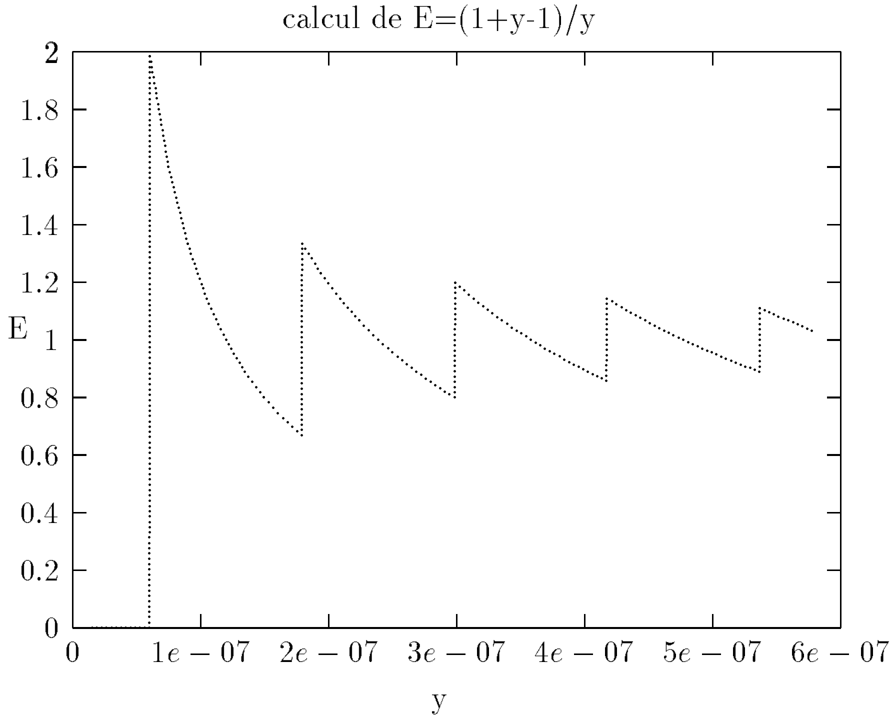


Figure 10 : Calcul de E en simple précision

- calcul assez précis: l'erreur de calcul est faible, plus petite que l'erreur d'approximation. La valeur calculée oscille autour de la dérivée exacte, dans un intervalle de faible amplitude. On a un résultat exact à la précision machine.

- transition vers l'élimination: l'erreur de calcul devient de plus en plus grande, les oscillations sont dans un intervalle d'amplitude croissante.

- élimination: $f(a+h)$ voisin de $f(a)$. L'erreur de calcul sur $f_h(a)$ risque alors d'être très grande, supérieure à l'erreur d'approximation. Le résultat calculé risque d'être faux.

- absorption: $f(a+h)$ négligeable devant $f(a)$.
  On obtient alors une dérivée calculée nulle, c'est-à-dire en général un résultat complètement faux.

On constate donc qu'il existe une valeur optimale h_0 de h pour laquelle les erreurs d'approximation et de calcul sont minimales. Pour h plus grand que h_0, l'erreur d'approximation est prépondérante, tandis que pour h plus petit que h_0, l'erreur de calcul devient prépondérante.

La notion de limite n'a donc plus de sens sur ordinateur. On peut toutefois dire que la dérivée approchée tend vers la dérivée exacte à la précision machine quand h tend vers h_0, et qu'il diverge pour h plus petit que h_0.

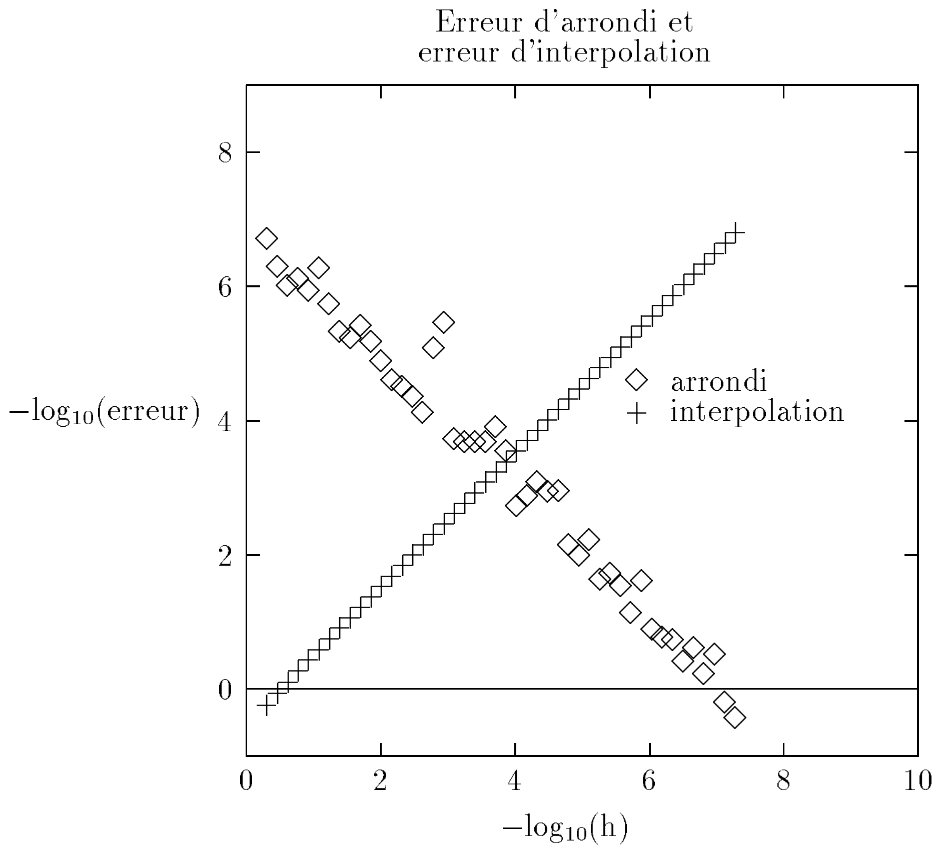


Figure 11 : Dérivation approchée de x^3 en 1

Considérons par exemple la dérivée de

$$f(x) = x^3$$

au point a=1. On calcule une valeur approchée de la dérivée par la formule

$$f_h(a) \;=\; \frac{f(a+h) - f(a)}{h}$$

On commet alors une erreur d'approximation égale à

$$|f'(a) - f_h(a)| \;=\; |3h + h^2|$$

La figure 11 montre d'une part l'erreur d'approximation, et d'autre part l'erreur d'arrondi (ou erreur de calcul) sur le taux d'accroissement $f_h(a)$, pour différentes valeurs de h, pour des calculs effectués en simple précision IEEE sur SUN. On constate comme prévu que lorsque h devient très petit, l'erreur de calcul devient plus grande que l'erreur d'approximation. Sur cet exemple, la valeur optimale h_0 est d'environ 10^{-4}.

7 Stabilité des problèmes et des algorithmes

7.1 Stabilité et conditionnement d'un problème

Un problème stable est un problème qui n'est pas trop sensible aux perturbations sur les données initiales. Autrement dit, le résultat est perturbé avec un facteur d'amplification compatible avec la précision requise. Pour formaliser cette notion, on introduit les définitions suivantes.

Considérons un problème (P), s'écrivant pour simplifier: Etant donné d dans $\mathcal{R}^n$, trouver x dans $\mathcal{R}^n$, tel que $x = F(d)$. Soit $\|.\|$ une norme de $\mathcal{R}^n$.

Définition: on dit que le problème (P) est *bien posé* au voisinage de d_0 si F est continue en d_0.

Remarque: pour un problème mal posé, le résultat devient complètement différent dès qu'on perturbe un tant soit peu les données initiales.

Définition: on dit que le problème (P) est *stable* au voisinage $\mathcal{V}$ de d_0 si on a la condition suivante:

$$\exists K \in \mathcal{R} \quad \forall d \in \mathcal{V} \quad \|x - x_0\| \leq K * \|d - d_0\|$$

Définition: soit (P) un problème stable au voisinage de d_0. Le *conditionnement absolu* C_a du problème (P) dans le voisinage de d_0 est la plus petite constante K possible:

$$C_a = \lim_{d \in \mathcal{V}} \inf \frac{\|x - x_0\|}{\|d - d_0\|}$$

Définition: soit (P) un problème stable au voisinage de d_0, avec d_0 et x_0 non nuls. Le *conditionnement relatif* C_r du problème (P) est défini par:

$$C_r = \lim_{d \in \mathcal{V}} \inf \frac{\|x - x_0\| / \|x_0\|}{\|d - d_0\| / \|d_0\|}$$

Remarque : si F est de classe C^1, un majorant du conditionnement absolu est donné par $\sup_{d \in \mathcal{V}} \|DF(d)\|$, où $DF(d)$ est la différentielle de F en d.
Remarque : Un problème instable correspond à un conditionnement (absolu ou relatif) infini. Le conditionnement d'un problème est en quelque sorte le facteur d'amplification des erreurs sur les données. En pratique, lorsque le conditionnement est grand, on dit que le problème est *mal conditionné*, et lorsqu'il est petit que le problème est *bien conditionné*.

7.2 Exemple de problème mal posé: limite d'une suite récurrente

Considérons la suite récurrente définie de la manière suivante (exemple de J.M. Muller):

$$\begin{array}{l} x_{n+1} = 111 - \frac{1130}{x_n} + \frac{3000}{x_n * x_{n-1}} \\ x_0 = 11/2 \\ x_1 = 61/11 \end{array} \qquad (2)$$

Cette suite converge vers 6. Mais si on calcule les termes de cette suite sur n'importe quelle machine, on trouve qu'elle converge... vers 100 ... Comment expliquer ce phénomène mystérieux?

Pour cela, nous allons nous ramener à une récurrence linéaire d'ordre 3: Supposons que

$$x_n = \frac{a_{n+1}}{a_n}$$

Alors a_n vérifie la récurrence:

$$\begin{array}{l} a_{n+2} = 111 a_{n+1} - 1130 a_n + 3000 a_{n-1} \\ a_1 = a_0 \ x_0 \\ a_2 = a_1 \ x_1 \end{array} \qquad (3)$$

Réciproquement, si a_n vérifie (3), avec a_0 quelconque, alors x_n vérifie (2).

On sait que les solutions génériques d'une récurrence linéaire sont de la forme q^n. On cherche donc q solution de l'équation du 3^e degré:

$$q^3 - 111 q^2 + 1130 q - 3000 = 0$$

Les racines de cette équation sont 5,6, et 100. Les suites a_n solutions de (3) ont donc la forme générale:

$$a_n = \alpha \ 5^n + \beta \ 6^n + \gamma \ 100^n$$

α,β,et γ sont déterminés par les données initiales a_0,a_1, et a_2.

Etudions maintenant la suite x_n, en distinguant 3 cas:

- $\gamma \neq 0$: dans ce cas, la suite converge vers 100. En effet:

$$\begin{aligned} x_n &= \frac{\alpha \ 5^{n+1} + \beta \ 6^{n+1} + \gamma \ 100^{n+1}}{\alpha \ 5^n + \beta \ 6^n + \gamma \ 100^n} \\ &= \frac{\gamma \ 100^n \ [5 \ \alpha/\gamma \ (5/100)^n + 6 \ \beta/\gamma \ (6/100)^n + 100]}{\gamma \ 100^n \ [\alpha/\gamma \ (5/100)^n + \beta/\gamma \ (6/100)^n + 1]} \\ &= \frac{100 + u_n}{1 + v_n} \end{aligned}$$

$$\begin{array}{ll} \text{avec} & \lim_{n \to \infty} u_n = \lim_{n \to \infty} v_n = 0 \\ \text{d'où} & \lim_{n \to \infty} x_n = 100 \end{array}$$

- $\gamma = 0$ et $\beta \neq 0$: dans ce cas, la suite converge vers 6. Il suffit d'appliquer la même méthode que précédemment.

- $\gamma = \beta = 0$ et $\alpha \neq 0$: dans ce cas, la suite est stationnaire de valeur 5.

- $\gamma = \beta = \alpha = 0$: la suite est stationnaire de valeur 0.

Le cas $x_0 = 11/2$ et $x_1 = 61/11$ correspond à:

$$\begin{array}{l} a_0 = 2\alpha \quad a_1 = 11\alpha \quad a_2 = 61\alpha \text{ avec } \alpha \neq 0 \\ \text{d'où } \alpha = \beta \text{ et } \gamma = 0 \end{array}$$

La suite x_n définie en (2) converge donc vers 6.

Pourquoi obtient-on une convergence vers 100? Ceci s'explique par les erreurs d'arrondi sur les données. Il suffit de perturber les données initiales x_0 et x_1 pour obtenir une composante γ non nulle. Lorsqu'on convertit 61/11 en base 2, on commet une erreur d'arrondi qui suffit à faire basculer la suite vers le cas $\gamma \neq 0$. Même si l'on pouvait représenter exactement x_0 et x_1, les erreurs d'arrondi en cours de calcul introduiraient une perturbation qui se manifesterait par l'apparition d'une composante γ non nulle. Même si les calculs sont effectuées en précision finie très grande, il apparaît à partir d'un certain rang une composante γ non nulle suite aux erreurs d'arrondi. Il faut vraiment calculer avec une précision infinie (calcul exact) pour obtenir le bon résultat, c'est-à-dire la convergence vers 6.

Cette sensibilité aux arrondis provient du fait que le problème est ici mal posé: la limite de la suite x_n n'est pas continue par rapport aux données initiales, c'est-à-dire finalement par rapport aux composantes (α, β, γ). En effet, dans tout voisinage de $(\alpha, \beta, 0)$, cette limite vaut 100, sauf au point $(\alpha, \beta, 0)$. Donc toute perturbation des données initiales, et toute erreur d'arrondi se traduisent par un résultat faux. Il faut noter toutefois que ce résultat n'est pas quelconque, on trouve toujours une limite égale à 100.

7.3 Exemple de problème mal posé : limite d'une suite récurrente

L'exemple précédent peut sembler compliqué ou construit de toutes pièces. Considérons alors la suite de Fibonacci, récurrence linéaire du second ordre, qui est connue pour d'autres utilisations :

$$\begin{array}{l} x_n = x_{n-1} + x_{n-2} \\ x_0 \text{ et } x_1 \text{ donnés.} \end{array} \tag{4}$$

Les suites (x_n) solutions de (4) ont la forme générale :

$$x_n \quad = \quad \alpha \; {q_1}^n + \beta \; {q_2}^n$$

où α et β sont déterminés par les données initiales x_0 et x_1, et où q_1 et q_2 sont les solutions de l'équation du second degré :

$$\begin{array}{l} q^2 - q - 1 = 0 \\ q_1 = (\sqrt{5} - 1)/2 \\ q_2 = (\sqrt{5} + 1)/2 \end{array}$$

De même que dans l'exemple précédent, la convergence de la suite x_n dépend des valeurs de α et β : si β est non nul, la suite diverge (tend vers l'infini), alors que si β est nul, la suite converge vers 0.

On n'obtiendra jamais sur ordinateur une convergence vers 0. En effet, les erreurs d'arrondi sur les données initiales ou en cours de calcul introduisent une composante β non nulle qui très vite conduit la suite à exploser, c'est-à-dire à tendre rapidement vers l'infini. Là encore, le problème de la convergence de la suite n'est pas continu par rapport aux données initiales.

7.4 Exemple de problème mal conditionné: résolution d'un système linéaire

Considérons le système linéaire $A\ u\ =\ b$ (exemple dû à R.S. Wilson, et extrait du livre de P.G. Ciarlet):

$$\begin{pmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{pmatrix} = \begin{pmatrix} 32 \\ 23 \\ 33 \\ 31 \end{pmatrix}$$

de solution $(1, 1, 1, 1)^t$,
et considérons le système perturbé, où les seconds membres ont été légèrement modifiés, la matrice restant inchangée:

$$\begin{pmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{pmatrix} \begin{pmatrix} u_1 + \delta u_1 \\ u_2 + \delta u_2 \\ u_3 + \delta u_3 \\ u_4 + \delta u_4 \end{pmatrix} = \begin{pmatrix} 32.1 \\ 22.9 \\ 33.1 \\ 30.9 \end{pmatrix}$$

de solution $(9.2, -12.6, 4.5, -1.1)^t$.
Autrement dit, une erreur relative de 1/200 sur les données (ici les composantes du second membre) entraîne une erreur de 10/1 sur le résultat (la solution du système linéaire), soit un rapport d'amplification des erreurs relatives de l'ordre de 2000. Le conditionnement relatif de ce problème est donc de l'ordre de 2000.

Considérons également le système perturbé, où, cette fois, ce sont les éléments de la matrice qui ont été légèrement modifiés:

$$\begin{pmatrix} 10 & 7 & 8.1 & 7.2 \\ 7.08 & 5.04 & 6 & 5 \\ 8 & 5.98 & 9.89 & 9 \\ 6.99 & 4.99 & 9 & 9.98 \end{pmatrix} \begin{pmatrix} u_1 + \Delta u_1 \\ u_2 + \Delta u_2 \\ u_3 + \Delta u_3 \\ u_4 + \Delta u_4 \end{pmatrix} = \begin{pmatrix} 32 \\ 23 \\ 33 \\ 31 \end{pmatrix}$$

de solution $(-81, 137, -34, 22)^t$.

Là encore, de très petites variations des données (ici, les éléments de la matrice) modifient complètement le résultat (la solution du système linéaire). Le système linéaire est très mal conditionné. Pourtant, la matrice d'origine A est symétrique, son déterminant vaut 1, et la matrice inverse A^{-1} a une allure normale:

$$A^{-1} = \begin{pmatrix} 25 & -41 & 10 & -6 \\ -41 & 68 & -17 & 10 \\ 10 & -17 & 5 & -3 \\ -6 & 10 & -3 & 2 \end{pmatrix}$$

Alors comment mesurer a priori le conditionnement d'un système linéaire? Analysons ce phénomène: il s'agit de comparer les solutions exactes u, $u+\delta u$, et $u+\Delta u$ des systèmes:

$$\begin{aligned} A\,u &= b \\ A\,(u+\delta u) &= b+\delta b \\ (A+\Delta A)(u+\Delta u) &= b \end{aligned}$$

Soit $\|.\|$ une norme vectorielle, et $\|.\|$ la norme matricielle subordonnée. On a les inégalités:

$$\begin{aligned} \|b\| &\leq \|A\|\,\|u\| \\ \delta u &= A^{-1}\,\delta b \\ \text{d'où } \|\delta u\|/\|u\| &\leq \|A\|\|A^{-1}\|\ \|\delta b\|/\|b\| \end{aligned}$$

et:

$$\begin{aligned} \Delta u &= -A^{-1}\,\Delta A\,(u+\Delta u) \\ \text{d'où } \|\Delta u\|/\|u+\Delta u\| &\leq \|A\|\|A^{-1}\|\ \|\Delta A\|/\|A\| \end{aligned}$$

On peut montrer que, si ΔA est petit, alors Δu est petit, donc $\|u+\Delta u\|$ est proche de $\|u\|$. On peut montrer que cette inégalité est la meilleure possible. On a donc trouvé le conditionnement relatif du système linéaire $Au=b$:

$$cond(A) = \|A\|\,\|A^{-1}\|$$

Remarque: ce conditionnement dépend du choix de la norme. Le conditionnement *spectral* correspond à la norme $\|.\|_2$. Pour les matrices normales, on peut calculer le conditionnement en fonction des valeurs propres:

$$cond_2(A) = \frac{\max_i |\lambda_i|}{\min_i |\lambda_i|}$$

Revenons à l'exemple précédent. Les valeurs propres de la matrice ont pour valeurs approchées

$$\begin{aligned} \lambda_1 &\simeq 0.01015 \\ \lambda_2 &\simeq 0.8431 \\ \lambda_3 &\simeq 3.858 \\ \lambda_4 &\simeq 30.2887 \end{aligned}$$

de sorte que

$$cond_2(A) \;=\; \lambda_4/\lambda_1 \;\;\simeq\;\; 2984$$

et on retrouve le facteur d'amplification voisin de 2000.

Ce problème mal conditionné sera très mal résolu sur ordinateur: en effet, une simple conversion des données due au changement de base modifiera sensiblement la solution.

7.5 Stabilité et analyse d'erreur d'un algorithme

On a vu des exemples de sensibilité aux erreurs sur les données initiales. Mais à ces erreurs s'ajoutent les erreurs de calcul. Pour résoudre un problème sur ordinateur, on utilise un *algorithme*, et on fait les calculs avec l'arithmétique flottante. L'erreur sur le résultat final dépend alors non seulement des erreurs sur les données, mais aussi des erreurs sur les résultats intermédiaires.

Il existe deux types d'analyse des erreurs d'arrondi d'un algorithme : *analyse de sensibilité*, et *analyse régressive*. Pour définir ces notions, il faut se donner un modèle de l'arithmétique flottante utilisée. En général, on suppose qu'il existe un arrondi avec une précision u et que les opérations arithmétiques sont correctes. Pour l'arrondi au plus près, u vaut $\epsilon/2$, tandis que pour les autres modes d'arrondi, u vaut ϵ.

Considérons un problème (P), s'écrivant pour simplifier: Etant donné d dans $\mathcal{R}^n$, trouver x dans $\mathcal{R}^n$, tel que $x = F(d)$. Soit $\|.\|$ une norme de $\mathcal{R}^n$. Considérons également un algorithme de résolution (A). Soit x la solution de (P) et $\overline{x}$ le résultat obtenu par (A).

Définition : l'analyse de sensibilité consiste à trouver un majorant de l'erreur $\|x - \overline{x}\|$ de la forme :

$$\|x - \overline{x}\| \leq K_1 \; u$$

Définition : l'analyse régressive consiste à trouver un problème approché dont x est solution. On cherche $\overline{d}$, tel que :

$$\overline{x} = F(\overline{d})$$
$$\|d - \overline{d}\| \leq K_2 \; u$$

Remarque : si on connaît le conditionnement C du problème (P), l'analyse régressive permet d'obtenir une majoration de l'erreur $\|x - \overline{x}\|$.

$$\|x - \overline{x}\| \leq C \; K_2 \; u$$

7.6 Exemple d'analyse de sensibilité: calcul approché d'une intégrale

On cherche à calculer l'intégrale (exemple extrait du livre de J.M. Muller):

$$I_n = \int_0^1 x^n \, e^{-x} \, dx$$

pour n=50. En intégrant par parties, on obtient une formule de récurrence:

$$\begin{aligned} I_n &= n \, I_{n-1} - 1/e \\ I_0 &= 1 - 1/e \end{aligned}$$

Si l'on applique cette méthode sur ordinateur, on obtient un résultat complètement faux! Pourquoi?

On commet une erreur sur la valeur initiale I_0, puisqu'on ne peut pas représenter exactement le nombre e. Même si l'on ne commettait plus d'erreur d'arrondi dans la suite des calculs, on obtiendrait un résultat faux uniquement par amplification de l'erreur sur I_0. Evaluons l'erreur sur I_n en fonction de celle sur I_0, en supposant les autres calculs exacts.

$$\begin{aligned} \overline{I_0} &= I_0 + \Delta I_0 \\ \overline{I_n} &= I_n + \Delta I_n \\ \overline{I_n} &= n(I_{n-1} + \Delta I_{n-1}) - 1/e \\ \text{donc } \Delta I_n &= n \, \Delta I_n \\ \text{et } \Delta I_n &= n! \, \Delta I_0 \end{aligned}$$

On constate donc qu'il est impossible d'obtenir un résultat correct par cet algorithme. On pourrait penser que les erreurs d'arrondi se compensent mutuellement, mais une simple analyse montre qu'au contraire elles s'ajoutent.

Mais peut-on faire autrement? Oui, il suffit d'utiliser la récurrence rétrograde. Supposons que l'on connaisse I_p pour $p > n$. On utilise la récurrence

$$\begin{aligned} &I_{n-1} = 1/n \; (I_n + 1/e) \\ &I_p \; (p > n) \text{ connu} \end{aligned}$$

Cette fois, la sensibilité à l'erreur sur I_p est beaucoup plus faible (toujours en supposant les autres calculs exacts).

$$\Delta I_n = n!/p! \, \Delta I_p$$

Mais on ne connaît pas la valeur de I_p. Heureusement, il suffit d'une valeur approchée pour obtenir une valeur de I_n exacte à la précision machine!

Par exemple, si on choisit $p = 60$, et $I_p = 1$, on commet une erreur $|\Delta I_p|$ inférieure à 1, ce qui aboutit à une erreur sur I_n (n=50) inférieure à $50!/60! \simeq 3 \; 10^{-18}$.

De plus, une analyse détaillée des erreurs d'arrondi montre que cet algorithme est stable.

Effectivement, le calcul sur ordinateur de I_{50} par cette méthode donne un résultat exact (à la précision machine).

Cet exemple met en évidence l'importance de l'analyse de la stabilité pour choisir un algorithme.

7.7 Exemple d'analyse régressive: résolution d'un système linéaire par la méthode de Gauss

Ces exemples sont extraits du livre de P.G. Ciarlet.

Pour résoudre numériquement un système linéaire, la méthode de Cramer nécessiterait beaucoup trop d'opérations. On utilise la *méthode de Gauss*, qui comporte deux phases:

- une phase *d'élimination*: on élimine les inconnues dans les équations de façon à obtenir un système équivalent qui soit triangulaire supérieur.
- une phase *de remontée*: on résout le système triangulaire supérieur, en commançant par la dernière équation, et en remontant jusqu'à la première.

Considérons le système linéaire

$$\begin{array}{rcrcrcl} 5u_1 & + & 2u_2 & + & u_3 & = & 12 \\ 5u_1 & - & 6u_2 & + & 2u_3 & = & -1 \\ -4u_1 & + & 2u_2 & + & u_3 & = & 3 \end{array}$$

Après élimination de u_1, on a

$$\begin{array}{rcrcrcl} 5u_1 & + & 2u_2 & + & u_3 & = & 12 \\ & & -8u_2 & + & u_3 & = & -13 \\ & & 18/5u_2 & + & 9/5u_3 & = & 63/5 \end{array}$$

Puis après élimination de u_2

$$\begin{array}{rcrcrcl} 5u_1 & + & 2u_2 & + & u_3 & = & 12 \\ & & -8u_2 & + & u_3 & = & -13 \\ & & & & 9/4u_3 & = & 27/4 \end{array}$$

Il suffit alors de remonter à partir de u_3

$$\begin{array}{rcl} u_3 & = & 3 \\ u_2 & = & 2 \\ u_1 & = & 1 \end{array}$$

Cette résolution ne pose pas de problème d'arrondi, on obtient le bon résultat sur la plupart des machines.

Considérons maintenant le système d'ordre 2

$$\begin{aligned} 10^{-4}u_1 + u_2 &= 1 \\ u_1 + u_2 &= 2 \end{aligned}$$

de solution

$$\begin{aligned} u_1 &= 1.00010\ldots \\ u_2 &= 0.99990 \end{aligned}$$

Ce système est bien conditionné, le conditionnement spectral est de l'ordre de 2.7.

Supposons maintenant qu'on applique l'algorithme de Gauss sur un système flottant en base 10, avec 3 chiffres de mantisse.

Si on choisit d'éliminer d'abord u_1, puis u_2, on obtient, après arrondis

$$\begin{aligned} 10^{-4}u_1 + u_2 &= 1 \\ -10^{4}u_2 &= -10^{4} \end{aligned}$$

donc la solution calculée vaut

$$\begin{aligned} u_1 &= 0 \\ u_2 &= 1 \end{aligned}$$

On obtient un résultat faux pour u_1!

Il y a en effet absorption de 1 et 2 par 10^4, lors de l'élimination de u_1. Cette absorption est due à la division par 10^{-4}, dont le résultat est un nombre très grand devant les autres coefficients.
Si on élimine maintenant d'abord u_2, puis u_1, on obtient, toujours après arrondis

$$\begin{aligned} u_1 + u_2 &= 2 \\ u_1 &= 1 \end{aligned}$$

donc la solution calculée vaut

$$\begin{aligned} u_1 &= 1 \\ u_2 &= 1 \end{aligned}$$

On obtient un résultat correct à la précision machine.

On constate donc que l'algorithme de Gauss est sensible à l'ordre d'élimination. Lorsque pour éliminer une inconnue, on divise par un nombre trop petit (dans l'exemple 10^{-4}), on commet des erreurs d'arrondi désastreuses, dues au phénomène d'absorption. Les diviseurs qui apparaissent lors de l'élimination d'une inconnue sont appelés des *pivots*. Il faut donc choisir des pivots aussi grands que possibles. On notera que l'ordre de grandeur des pivots n'est pas lié directement au conditionnement du système linéaire. L'algorithme de Gauss n'est stable que si l'on applique une *stratégie de pivot* pour déterminer l'ordre d'élimination des inconnues.

Il est possible d'effectuer une analyse régressive de l'algorithme de Gauss. On introduit les notations suivantes : soit A et B deux matrices de même ordre. On définit $|A|$ par

$$|A|_{ij} \quad = \quad |A_{ij}|$$

et la relation d'ordre (partiel) $\leq$ par

$$A \leq B \quad \Leftrightarrow \quad \forall i, j \quad A_{ij} \leq B_{ij}$$

Soit à résoudre le système linéaire d'ordre n

$$A\, x \quad = \quad b$$

et soit $\overline{x}$ la solution calculée par l'algorithme de Gauss. On montre (voir livre de Golub et Van Loan), qu'il existe une matrice E et une matrice K telles que :

$$\begin{aligned} (A+E)\, x &= b \\ |E| &\leq K\, u + O(u^2) \end{aligned}$$

De plus, on montre que les coefficients de K sont d'autant plus grands que les pivots de l'élimination sont petits.

On effectue de même une analyse régressive pour l'algorithme de Gauss avec une stratégie de pivot (dite pivot partiel). On montre l'inégalité suivante :

$$\|E\|_\infty \leq 8\, n^3\, \rho\, \|A\|_\infty\, u + O(u^2)$$

où ρ est appelé *facteur de croissance*.

En pratique, ρ est petit, de l'ordre de 10, mais on peut trouver des exemples où ρ est aussi grand que 2^{n-1}.

7.8 Exemple d'analyse de sensibilité : orthonormalisation d'un système de vecteurs

Soit $(x_1, x_2, \ldots, x_n)$ un système libre de n vecteurs dans $\mathcal{R}^m$. On veut construire une base orthonormée $(v_1, v_2, \ldots, v_n)$, qui engendre le même sous-espace que ce système.

L'idée du principe d'orthogonalisation de Gram-Schmidt est d'utiliser les projections orthogonales sur l'orthogonal d'un sous-espace. Avant d'expliquer l'algorithme de Gram-Schmidt, rappelons quelques notions sur les projections orthogonales.

Soit $V = [v_1, v_2, \ldots, v_n]$ la matrice (m, n) dont les colonnes sont les vecteurs v_i. On note V^t la matrice transposée. Ces vecteurs v_i forment un système orthonormé, noté par extension V si et seulement si:

$$V^t\ V\ =\ I_n$$

où I_n désigne la matrice unité d'ordre n.

On note $\sigma(V)$ le sous-espace engendré par V. On suppose V orthonormé. Alors la projection orthogonale sur $\sigma(V)$ est définie par la matrice P

$$P\ =\ V\ V^t$$

et la projection orthogonale sur $\sigma(V)^\perp$ a pour matrice

$$Q_V\ =\ I\ -\ V\ V^t$$

Le cas particulier où $V = [v]$ est réduit à un seul vecteur normé se traduit

$$Q_v\ x\ =\ x\ -\ (v^t\ x)v$$

où $(v^t\ x)$ est le produit scalaire de v et x.

Il faut noter la propriété suivante, qui va donner lieu à deux formulations de l'algorithme de Gram-Schmidt

$$Q_V\ = Q_{v_n}\ Q_{v_{n-1}}\ \ldots\ Q_{v_1}$$

L'algorithme de Gram-Schmidt construit pas à pas la base orthonormée V_n. Le premier vecteur v_1 s'obtient en normalisant le vecteur x_1. Puis v_2 est la projection orthogonale normalisée de x_2 sur l'orthogonal de v_1.

Si $V_k = (v_1, v_2, \ldots, v_k)$ est construit, on calcule w_{k+1}, projection orthogonale de x_{k+1} sur l'orthogonal de V_k, que l'on normalise pour déterminer v_{k+1}. On obtient ainsi un système orthonormé V_{k+1} qui engendre le même sous-espace que $(x_1, \ldots, x_{k+1})$.

L'algorithme de Gram-Schmidt s'écrit donc

$v_1 := x_1/\|x_1\|$
pour $k = 1$ à $n - 1$ faire
  $w_{k+1} := Q_{V_k}\ x_{k+1}$
  $v_{k+1} := w_{k+1}/\|w_{k+1}\|$
fin faire

Selon l'expression que l'on choisit pour calculer Q_{V_k}, on aboutit à deux versions de l'algorithme:

- Gram-Schmidt *classique*: on utilise

  $$Q_{V_k}\ \ =\ \ I - V_k\ V_k^t$$

  soit

  $$Q_{V_k}\ x\ \ =\ \ x - (v_1^t\ x)v_1 - (v_2^t\ x)v_2 - \ldots - (v_k^t\ x)v_k$$

σ	$\|I - V^t\ V\|$	
	modifié	classique
10^{-4}	3.10^{-13}	2.10^{-9}
10^{-5}	7.10^{-13}	5.10^{-8}
10^{-6}	7.10^{-11}	5.10^{-5}
10^{-7}	2.10^{-9}	1.10^{-2}

Table 13 : Résidu pour Gram-Schmidt modifié et classique

- Gram-Schmidt *modifié*: on utilise

$$Q_{V_k} \;=\; Q_{v_k}\ \ldots\ Q_{v_2}\ Q_{v_1}$$

soit

$$\begin{aligned}
w_k^1 &= Q_{v_1}\ x = (x - v_1^t\ x)\ v_1 \\
w_k^2 &= Q_{v_2}\ w_k^1 = (w_k^1 - v_2^t\ w_k^1)\ v_2 \\
\ldots & \\
w_k^k &= Q_{v_k}\ w_k^{k-1} = (w_k^{k-1} - v_k^t\ w_k^{k-1})\ v_k \\
Q_{V_k} &= w_k^k
\end{aligned}$$

Laquelle de ces deux méthodes choisir? En fait, si elles sont équivalentes mathématiquement, elles ne le sont plus du tout numériquement. On peut montrer que la version modifiée est beaucoup plus stable que la version classique. Soit $\overline{V_n}$ la "base" calculée par l'algorithme de Gram-Schmidt. Pour comparer les deux méthodes, on introduit la notion de résidu défini par

$$\|I - \overline{V_n^t}\ \overline{V_n}\|$$

et qui mesure la perte d'orthogonalité de $\overline{V_n}$ par rapport à V_n, ($\|.\|$ est une norme matricielle quelconque).

Considérons dans $\mathcal{R}^4$ le système de 3 vecteurs

$$X \;=\; \begin{pmatrix} 1 & 1 & 1 \\ \sigma & 0 & 0 \\ 0 & \sigma & 0 \\ 0 & 0 & \sigma \end{pmatrix}$$

Le résidu est reporté dans la table 13 pour les versions classique et modifiée, pour différentes valeurs de σ. Les calculs ont été effectués en double précision IEEE, avec $\epsilon = 2^{-52} \simeq 2.10^{-16}$.

On constate sur cet exemple que la version modifiée est bien plus stable que la version classique. Le résidu est en effet beaucoup plus petit, surtout pour de petites valeurs de σ.

De manière générale, on peut majorer le résidu de l'algorithme modifié, à l'aide du conditionnement de la matrice (m,n) X, noté $cond(X)$. Ce conditionnement est défini par

$$cond(X) \quad = \quad \sqrt{cond_2(X^t\ X)}$$

On montre l'inégalité

$$\|I - \overline{V_n^t}\ \overline{V_n}\| \quad \leq \quad K\ cond(X)\ u$$

où K est une constante qui ne dépend que de n et m.

Dans l'exemple précédent, le conditionnement vaut

$$cond(X) \quad = \quad \sqrt{3+\sigma^2}/\sigma$$

ce qui explique que le résidu devient grand lorsque σ devient petit. L'algorithme de Gram-Schmidt est en effet de plus en plus instable.

Ces algorithmes montrent comme l'algorithme de Gauss précédent que deux versions mathématiquement équivalentes ont des stabilités numériques très différentes. L'analyse d'erreur de l'algorithme modifié montre l'importance de la notion de résidu. On notera de plus que sur cet exemple, le résidu est proportionnel au conditionnement du système de vecteurs à orthonormaliser.

8 Conclusion

Tous ces exemples montrent que les résultats d'un calcul sur ordinateur ne sont qu'approchés, et surtout peuvent être faux. Alors que peut-on faire? Tout d'abord, il faut avoir conscience de ces phénomènes, et ne pas avoir une confiance aveugle dans l'ordinateur.

Il faut choisir une formulation stable du problème, dont on puisse évaluer le conditionnement. Les exemples précédents montrent des sensibilités très variables aux erreurs sur les données. Ensuite, il faut choisir un algorithme de résolution stable, dont on puisse analyser l'effet des erreurs d'arrondi. Les exemples précédents montrent des comportements numériques très différents d'algorithmes mathématiquement équivalents.

En algèbre linéaire, les problèmes et les algorithmes ont été bien étudiés, et on dispose à la fois de formules de conditionnement et d'analyses régressives. Par contre, dans le cas général, ce n'est pas toujours aussi facile. On peut alors avoir recours à des méthodes analytiques, qui effectuent un développement limité à l'ordre un du résultat. Mais ces méthodes sont très couteuses en temps calcul, et ne s'appliquent pas à tous les programmes. Les estimations statistiques semblent très prometteuses, elles permettent d'estimer la stabilité d'un algorithme quelconque à partir d'un échantillon de résultats provenant de calculs où l'on a introduit des perturbations sur les opérations. Mais la méthodologie pour interpréter les résultats reste actuellement du domaine de la recherche.

L'arithmétique d'intervalles permet en théorie de déterminer un encadrement de la solution, donc de mesurer la précision du résultat. Cette arithmétique consiste à remplacer les flottants par des intervalles dont les bornes sont deux flottants consécutifs. On définit alors des opérations sur les intervalles, de façon à ce que l'intervalle résultat contienne tous les résultats possibles entre deux réels des deux intervalles opérandes. On utilise pour calculer les bornes les arrondis par défaut et par excés. Mais outre la durée d'exécution, l'inconvénient est une amplification de la largeur des intervalles, qui aboutit à un intervalle résultat beaucoup trop grand pour être d'une quelconque utilité. Il est inutile de savoir que le résultat est dans $(-\infty, +\infty)$!

On peut néanmoins utiliser cette arithmétique d'intervalles astucieusement. D'une part, on effectue, par simulation logicielle, les produits scalaires de manière exacte, c'est-à-dire qu'on arrondit uniquement le résultat final. En effet, la majoration de l'erreur sur le produit scalaire est proportionnelle à leur longueur, ce qui conduit très vite à des intervalles très grands. D'autre part, on exprime le problème sous forme d'un point fixe, et on le résout par raffinement itératif, après avoir calculé une solution approchée par l'algorithme habituel, avec l'arithmétique flottante standard. Dans beaucoup de cas, il suffit alors de quelques itérations (une ou deux) pour obtenir un encadrement de la solution assez précis. Cependant, cette méthode n'est pas universelle, et son application n'est pas encore industrialisée.

Lorsque le problème est mal conditionné, il faut effectuer les calculs avec une grande précision. Toutefois, l'utilisateur ne dispose que de deux précisions, simple et double, et parfois d'une troisième, dite quadruple (128 bits). Sinon, il faut avoir recours à des outils logiciels qui simulent une arithmétique de plus grande précision, et qui sont très couteux en temps calcul. Il faut noter de plus qu'augmenter la précision ne garantit pas toujours l'obtention de résultats corrects, comme le prouvent les exemples précédents.

Evidemment, l'idéal serait d'effectuer les calculs avec une précision infinie. C'est ce que l'on peut faire avec le calcul formel. On manipule alors des symboles, comme en mathématiques, et on effectue les calculs seulement sur la dernière formule par exemple, et avec la précision voulue. Les logiciels de calcul formel sont des outils très puissants, qui offrent beaucoup de fonctionnalités, mais la durée d'exécution est encore prohibitive pour les gros calculs scientifiques.

Ce panorama est bien sûr incomplet, le scientifique utilise aussi et surtout sa connaissance physique du problème pour juger de la qualité d'un résultat de calcul. L'essentiel est de garder un oeil critique et de savoir interpréter les résultats, en estimant la précision obtenue grâce notamment aux outils mentionnés.

Bibliographie

[1] American National Standard. *IEEE Standard for Binary Floating-Point Arithmetic.* ANSI/IEEE Std 754-1985, publié par IEEE, New-York.

[2] P.G. Ciarlet. *Introduction à l'analyse numérique matricielle et à l'optimisation.* Masson, Paris, 1982.

[3] G.H. Golub, C.F. Van Loan. *Matrix computations.* John Hopkins, 1989.

[4] U.W. Kulisch, W.L. Miranker. *The arithmetic of the digital computer: a new approach.* SIAM Review 28, pages 1-40, 1986.

[5] J.M. Muller. *Arithmétique des ordinateurs.* Masson, Paris, 1989.