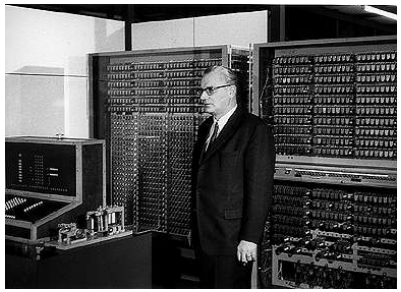
Arithmétique virgule flottante

Jean-Michel Muller
CNRS - Laboratoire LIP
mai 2006

http://perso.ens-lyon.fr/jean-michel.muller/

Virgule flottante ?

- babyloniens (absence de zéro, base 60, on n'écrit que les "mantisses") ;
- 1936-1938 : machine Z1 de Konrad Zuse (1910-1995) : base 2, mantisses de 16 bits et exposants de 7 bits. Mémoire de 16 nombres. Voir http://www.epemag.com/zuse/



Quelques chiffres

- besoins en dynamique ?

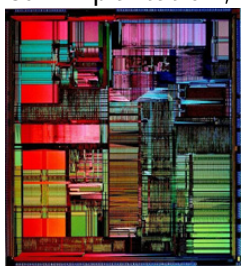
$$\frac{\text{Diamètre estimé de l'Univers}}{\text{Distance de Planck}} \approx 1.4 \times 10^{62}$$

- besoins en précision ? Certaines prédictions de la relativité générale et de la mécanique quantique vérifiées avec une précision relative d'environ 10^{-14}
- calculs intermédiaires : besoin de calculs en quadruple précision (J. Laskar, Observatoire de Paris) pour stabilité à très long terme du système solaire ;
- record actuel : 1241 milliards de chiffres décimaux de π (Kanada, 2002), en utilisant les deux formules

$$\begin{aligned}\pi &= 48\arctan\frac{1}{49} + 128\arctan\frac{1}{57} - 20\arctan\frac{1}{239} + 48\arctan\frac{1}{110443}\\ &= 176\arctan\frac{1}{57} + 28\arctan\frac{1}{239} - 48\arctan\frac{1}{682} + 96\arctan\frac{1}{12943}\end{aligned}$$

On sait faire du très mauvais travail...

- Division du Pentium 1 : résultat faux dans 1 cas sur 4×10^{10} (en simple précision). Le calcul de 8391667/12582905 donnait $0.666869\cdots$ au lieu de $0.666910\cdots$. Erreur dans l'algorithme lui-même, pas dans son implantation ;



- Excel, de la version 3.0 à la version 7.0, tapez 1.40737488355328 vous verrez apparaître 0.64
- Calculator de Windows 3.1 : 2.01 - 2.00 = 0.0 ;

- dans la version 7.0 de Maple, si l'on calcule

$$\frac{5001!}{5000!}$$

on obtient 1 au lieu de 5001 ;
- dans la version 6.0, si on entrait :

$$214748364804136478196437 94$$

la « quantité » affichée et mémorisée était `413647819643790)+'-.(-.(.`
Si on entrait 214748364810 on obtenait 10.
- avec la version 9, la commande `evalf(Sum(floor(n), n= 2..infinity)) ;`, qui demande à évaluer

$$\sum_{n=2}^{\infty} \lfloor n \rfloor$$

retourne -1.000000

$\frac{1}{10} = 0.099999904632568359375$

- Guerre du Golfe de 1991 : un missile US Patriot dont le programme tournait depuis 100 heures a raté l'interception d'un missile Irakien Scud → 28 morts ;
- Cause : 1/10 non représentable sur un nombre fini de chiffres en base 2 ;
- le Patriot incrémentait un compteur toutes les 0.1 secondes ;
- simple précision → 0.1 approché avec erreur $0.0000000953\cdots$;
- au bout de 100 heures, erreur cumulée ≈ 0.34 : dans un tel laps de temps le Scud parcourt 500 mètres.

Pire encore, avant la norme IEEE 754...

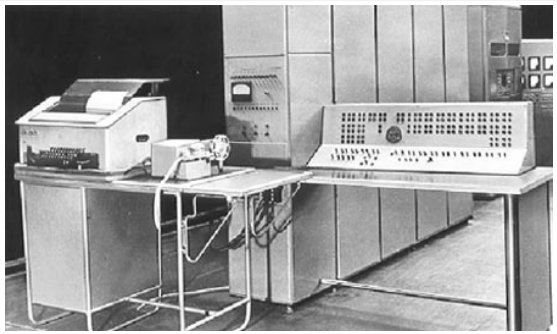
- sur certaines machines Cray, on avait

$$\text{parfois, } 1 \times x \Rightarrow \text{ overflow}$$
$$x + y \neq y + x$$
$$0.5 \times x \neq x/2.0$$

- IBM 370, en Fortran, on avait

$$\sqrt{-4} = 2$$
$$\texttt{I = 14.0/7.0} \rightarrow I = 1$$

Beaucoup de bizzareries



- Machine Setun, université de Moscou, 1958. 50 exemplaires ;
- base 3 et chiffres -1, 0 et 1. Nombres sur 18 « trits » ;
- idée : base β, nombre de chiffres n, + grand nombre représenté M. Mesure du « coût » : $\beta \times n$.
- minimiser $\beta \times n$ sachant que $\beta^n \approx M$. Si variables réelles, optimum $\beta = e = 2.718\ldots \approx 3$.

Etant donnés :

$$\begin{cases} \text{base} & \beta \geq 2 \\ \text{précision} & p \geq 1 \\ \text{plage d'exposants} & E_{\min} \cdots E_{\max} \end{cases}$$

Un nombre VF fini x est représenté par 2 entiers :

- mantisse entière : M, $|M| \leq \beta^p - 1$;
- exposant e, $E_{\min} \leq e \leq E_{\max}$.

tels que

$$x = M \times \beta^{e+1-p}.$$

On appelle mantisse réelle, ou mantisse de x le nombre

$$m = M \times \beta^{1-p},$$

de sorte que $x = m \times \beta^e$.

Buts :

- représentation unique ;
- a priori la plus précise (3.142×10^0 vs. 0.003×10^3).

La représentation normalisée de x, si elle existe, est celle pour laquelle $1 \leq m < \beta$. C'est celle qui minimise l'exposant.
En base 2 le premier chiffre de mantisse d'un nombre normalisé est un "1" $\rightarrow$ pas besoin de le mémoriser (convention du 1 implicite).
Un nombre sous-normal est un nombre de la forme

$$M \times \beta^{E_{\min}+1-p}.$$

avec $|M| \leq \beta^{p-1} - 1$. Un tel nombre n'a pas de représentation normalisée.
Correspond à $\pm 0.xxxxxxxx \times \beta^{E_{\min}}$.

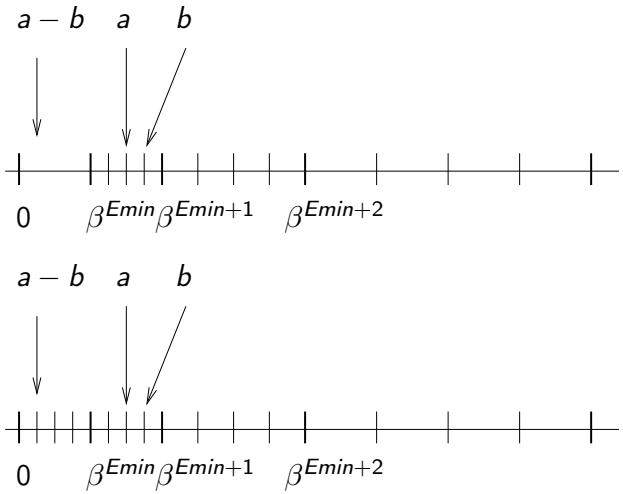


Fig.: *En haut : nombres VF normalisés. Dans cet ensemble, $a - b$ n'est pas représentable $\rightarrow$ le calcul $a - b$ donnera 0 en arrondi au plus près. En bas : on ajoute les sous-normaux.*

Système	β	p	$E_{\min}$	$E_{\max}$	+ grand représ.
DEC VAX	2	24	-128	126	$1.7\cdots \times 10^{38}$
(D format)	2	56	-128	126	$1.7\cdots \times 10^{38}$
HP 28, 48G	10	12	-500	498	$9.9\cdots \times 10^{498}$
IBM 370	16	6 (24 bits)	-65	62	$7.2\cdots \times 10^{75}$
et 3090	16	14 (56 bits)	-65	62	$7.2\cdots \times 10^{75}$
IEEE-754	2	23+1	-126	127	$3.4\cdots \times 10^{38}$
	2	52+1	-1022	1023	$1.8\cdots \times 10^{308}$
IEEE-754-R "binary 128"	2	112+1	-16382	16383	$1.2\cdots \times 10^{4932}$
IEEE-754-R "decimal 64"	10	16	-383	384	$9.999\cdots 9 \times 10^{384}$

Pour amuser vos élèves

Problème :

$$\boxed{I_n = \int_0^1 x^n e^{-x} dx, \text{ calculer } I_{50}}$$

Intégration par parties :

$$I_n = nI_{n-1} - 1/e \qquad (1)$$

On part de $I_0 = 1 - 1/e$, et on calcule successivement $I_1, I_2, I_3, \ldots, I_{50}$ à l'aide de (1). Calculs intermédiaires sur 50 chiffres décimaux (Maple), donne

$$-179030065581728.8725\cdots$$

pourtant I_{50} est trivialement entre 0 et 1.

L'âge du capitaine...

Hypothèse : le capitaine a 55 ans. En prenant comme valeur (très fausse) de I_{100} le nombre

$$\text{age du capitaine} \times \frac{\sqrt{\pi}}{512 + \sin(3)} \approx 0.190347851$$

et en utilisant la même formule $I_{n-1} = \frac{I_n + 1/e}{n}$ Pour calculer successivement $I_{99}, I_{98}, \ldots, I_{50}$, en faisant les calculs intermédiaires sur 50 chiffres décimaux, donne

$$0.00735470679580018886393671930987336729123712106726 29$$

qui a 49 chiffres en commun avec le résultat exact. Même propriété si le capitaine a entre 0 et 10^{43} ans.

Norme IEEE-754 (1985)

- mettre fin à un chaos (aucune portabilité, qualité très variable) ;
- moteur : W. Kahan (père de l'arithmétique de la HP35 et du coprocesseur Intel 8087) ;
- formats de représentation ;
- spécification des opérations, des conversions ;
- gestion des exceptions (max+1, 1/0, $\sqrt{-2}$, 0/0, etc.)

Arrondi correct

Definition (Arrondi correct)

L'utilisateur définit un *mode d'arrondi actif* parmi :

- arrondi au plus près (mode par défaut), càd vers le nombre VF le + proche. S'il y en a 2, on choisit celui dont la mantisse entière est paire.
- Arrondi vers $+\infty$, soit vers le plus petit nombre machine supérieur ou égal au résultat.
- Arrondi vers $-\infty$, soit vers le plus grand nombre machine inférieur ou égal au résultat.
- Arrondi vers zéro.

Une opération dont les entrées sont des nombres VF doit fournir ce qu'on obtiendrait si on avait d'abord calculé le résultat exactement, pour l'arrondir ensuite suivant le mode d'arrondi actif.

IEEE-754 (1985) : arrondi correct pour les 4 opérations arithmétiques ($+$, $-$, $\times$ et $\div$), $\sqrt{}$ et certaines conversions de format. Intérêts :

- si le résultat d'une opération portant sur 2 nombres VF est exactement représentable, c'est celui-ci qu'on obtient ;
- tant qu'on se limite aux 4 opérations et à $\sqrt{}$ l'arithmétique est déterministe : on peut élaborer des algorithmes et des preuves qui utilisent les propriétés de cette "arithmétique machine" ;
- la précision et la portabilité des algorithmes numériques se trouvent en général grandement améliorées ;
- en jouant sur les modes d'arrondi vers $+\infty$ et vers $-\infty$ (modes dirigés), on peut obtenir des minorants et/ou des majorants certains d'un résultat $\rightarrow$ arithmétique d'intervalles.

Approche "floue" $\rightarrow$ approche "exacte"

Changement d'approche

- Manière "classique" de raisonner : les opérations VF sont des approximations des opérations réelles $\rightarrow$ majorations successives ou approches probabilistes des erreurs d'arrondis.
  - arithmétique d'intervalles : `http://www.cs.utep.edu/interval-comp/` ;
  - arithmétique "stochastique" : `http://www-anp.lip6.fr/cadna/`
- Arrondi correct : l'arithmétique VF est une structure qu'on peut étudier, et pas seulement une approximation du corps $\mathbb{R}$. On peut prouver des propriétés fines et mettre au point des algorithmes utiles.

Un premier exemple

Lemme (Lemme de Sterbenz)

Soient a et b deux nombres VF positifs. Si

$$\frac{a}{2} \leq b \leq 2a$$

alors $a - b$ est exactement représentable en VF ($\rightarrow$ il est calculé sans erreur dans chacun des modes d'arrondi).

Preuve : élémentaire en se ramenant aux notations $x = M \times \beta^{e+1-p}$.

Erreur de l'addition flottante

Dans tout ce qui suit $\mathrm{RN}(x)$ désigne l'arrondi au plus près de x.

Lemme

Soient a et b deux nombres VF normalisés. On suppose que l'exposant de a est supérieur ou égal à celui de b. (est moins fort que d'imposer $|a| \geq |b|$). En l'absence de dépassement, si

$$s = RN(a + b)$$

et

$$r = (a + b) - s.$$

alors $|r| \leq |b|$ et r est un nombre VF.

Démonstration :

1. s est le nombre VF le plus proche de $a+b$. Donc en particulier il est plus proche de $a+b$ que ne l'est a. Donc

$$|(a+b)-s| \leq |(a+b)-a|$$

donc

$$|r| \leq |b|.$$

2. notons $a = M_a \times \beta^{e_a-p+1}$ et $b = M_b \times \beta^{e_b-p+1}$, avec $|M_a|, |M_b| \leq \beta^p - 1$, et $e_a \geq e_b$. On a : $a+b$ est un multiple de β^{e_b-p+1}, donc s aussi, donc r aussi. On peut donc écrire

$$r = R \times \beta^{e_b-p+1}$$

or, $|r| \leq |b| \Rightarrow |R| \leq |M_b| \leq \beta^p - 1 \Rightarrow r$ est représentable exactement.

Obtenir r : Algorithme fast2sum

Base $\beta = 2$. A partir de 2 nombres VF normalisés a et b, avec $|a| \geq |b|$, $2a \leq MAX$, on effectue les calculs :

$$\begin{aligned} s &= \mathrm{RN}(a+b) \\ z &= \mathrm{RN}(s-a) \\ t &= \mathrm{RN}(b-z) \end{aligned}$$

qui correspondent au programme C :

```
s = a+b;
z = s-a;
t = b-z;
```

En supposant que $s = \mathrm{RN}(a+b)$ ne provoque pas de dépassement. Bien entendu, s est le nombre VF le plus proche de $a+b$, mais de plus $s+t = a+b$ exactement. Donc $t = r$ contient l'erreur commise lors de l'addition de a et b.

Se méfier des compilateurs "optimisants".

$$\begin{aligned} s &= \mathrm{RN}(a+b) \\ z &= \mathrm{RN}(s-a) \\ t &= \mathrm{RN}(b-z) \end{aligned}$$

Preuve :

- si a et b sont de même signe, alors $|a| \leq |a+b| \leq |2a|$ donc (base $2 \rightarrow 2a$ est représentable, et l'arrondi est croissant) $|a| \leq |s| \leq |2a|$, donc (Lemme Sterbenz) $z = s-a$ exactement. Comme $r = (a+b)-s$ est représentable exactement et $b-z = r$, on trouve $\mathrm{RN}(b-z) = r$.
- si a et b sont de signes opposés, alors
  1. soit $|b| \geq \frac{1}{2}|a|$, auquel cas (lemme Sterbenz) $a+b$ est exact, donc $s = a+b$, $z = b$ et $t = 0$;
  2. soit $|b| < \frac{1}{2}|a|$, auquel cas $|a+b| > \frac{1}{2}|a|$, donc $s \geq \frac{1}{2}|a|$ (base $2 \rightarrow \frac{1}{2}a$ est représentable, et l'arrondi est croissant), donc (lemme Sterbenz) $z = \mathrm{RN}(s-a) = s-a = b-r$. Comme $r = (a+b)-s$ est représentable exactement et $b-z = r$, on trouve $\mathrm{RN}(b-z) = r$.

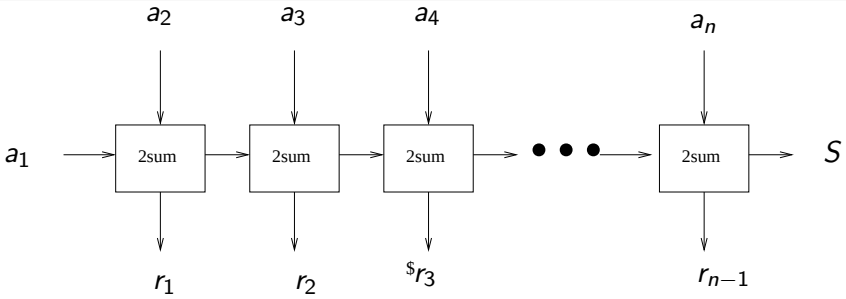
Généralisations

- on peut se contenter de supposer exposant(a) $\geq$ exposant(b) ;
- correct également pour $\beta = 3$ (démonstration + délicate). On trouve des contre-exemples en base ≥ 4.
- si on n'a pas l'hypothèse exposant(a) $\geq$ exposant(b) on peut utiliser l'algorithme 2sum :

```
s = a + b ;
b' = s - a ;
a' = s - b' ;
deltab = b - b' ;
deltaa = a - a' ;
r = deltaa + deltab ;
```

Même résultat que fast2sum. Trois opérations de plus, mais sur toutes les architectures modernes coûte moins cher que faire une comparaison.

Application : méthodes de "somme compensée"



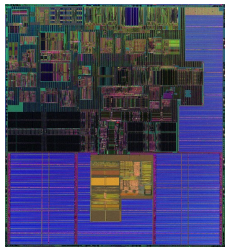
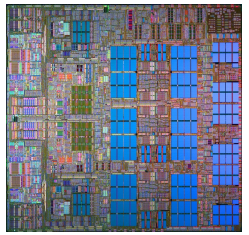
- remarque : $S + (r_1 + r_2 + \cdots + r_{n-1}) = a_1 + a_2 + \cdots + a_{n-1}$ exactement ;
- On calcule $S + (r_1 + r_2 + \cdots + r_{n-1})$ par additions VF usuelles. Si tous les a_i sont de même signe, et en double précision ($\beta = 2$, $p = 53$, arrondi au plus près) :

$$\text{erreur} \leq \left(\frac{1}{2} + n(n-1)2^{-54}\right) \text{ulp}(S)$$

On peut additionner $\sqrt{2} \times 2^{26}$ termes et garder une erreur $\leq$ poids du dernier bit.

Et avec des produits ?

- fma : *fused multiply-add*, calcule $\text{RN}(ab + c)$. Sur Itanium et PowerPC. Evaluation de polynômes (Horner), produits scalaires : + rapide et en général + précis qu'avec $\pm$ et $\times$;
- si a et b sont des nombres VF, alors $r = ab - \text{RN}(ab)$ est un nombre VF (élémentaire) ;
- s'obtient par $\begin{cases} p &= \text{RN}(ab) \\ r &= \text{RN}(ab - p) \end{cases}$
- sans fma, algorithme de Dekker : 17 opérations (7 $\times$, 10 $\pm$).

Itanium 2   PowerPC 5

Une autre propriété (Kahan)

$$z = \frac{x}{\sqrt{x^2 + y^2}}$$

- arrondi correct, arrondi au plus près, base 2 ;
- x et y sont des nombres VF ;
- pas d'overflow/underflow.

La valeur calculée de z est comprise au sens large entre -1 et $+1$. Propriété très importante car jugée (naïvement !) évidente par la plupart des programmeurs, qui pourront par exemple calculer une fonction de z définie seulement entre -1 et $+1$.

L'algorithme de Malcolm-Gentleman

Hypothèses : arithmétique VF, base β, arrondi correct, pas de dépassement.

```
A := 1.0;
B := 1.0;
while ((A+1.0)-A)-1.0 = 0.0 do A := 2*A;
while ((A+B)-A)-B <> 0.0 do B := B+1.0;
return(B)
```

Que fait ce petit programme ?

- en gros, $\mathrm{ulp}(x)$ = distance entre deux nombres VF consécutifs au voisinage de x ;
- ambigu au voisinage des puissances de $\beta \rightarrow$ plusieurs définitions légèrement différentes existent ;
- laquelle choisir ? Celle qui préserve dans ces "zones ambiguës" des propriétés vraies ailleurs.

Definition (Kahan)

$\mathrm{KahanUlp}(x)$ est la longueur de l'intervalle I dont les bornes sont les deux nombres VF les plus proches de x (même si $x \notin I$, même si x est une de ces bornes).

Definition (Harrison)

$\mathrm{HarrisonUlp}(x)$ est la distance entre les nombres VF les plus proches a et b tels que $a \leq x \leq b$ et $a \neq b$.

Definition (Goldberg)

Si $x \in [\beta^e, \beta^{e+1})$ alors $\mathrm{GoldbergUlp}(x) = \beta^{e-p+1}$.

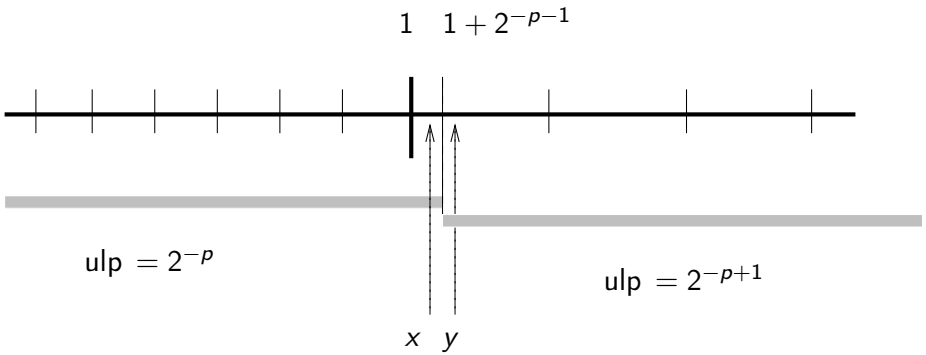


Fig.: *Les valeurs de* $\mathit{KahanUlp}(x)$ *au voisinage de* 1, *en supposant un système de base* 2 *et précision* p. *Noter l'effet étrange :* 1 *semble être une meilleure approximation de* y *que de* x.

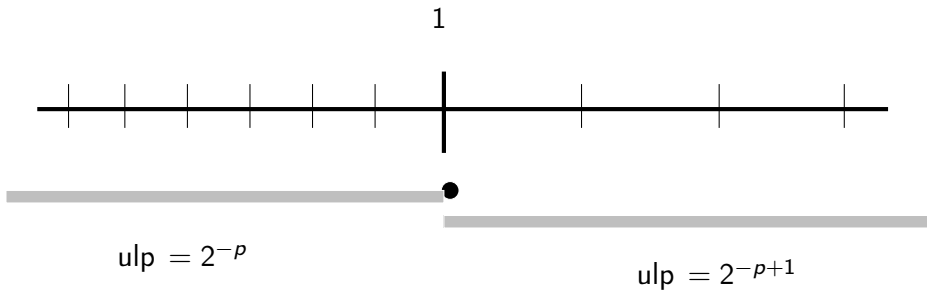


Fig.: *Les valeurs de* $\mathit{HarrisonUlp}(x)$ *au voisinage de* 1, *en supposant un système de base* 2 *et précision* p.

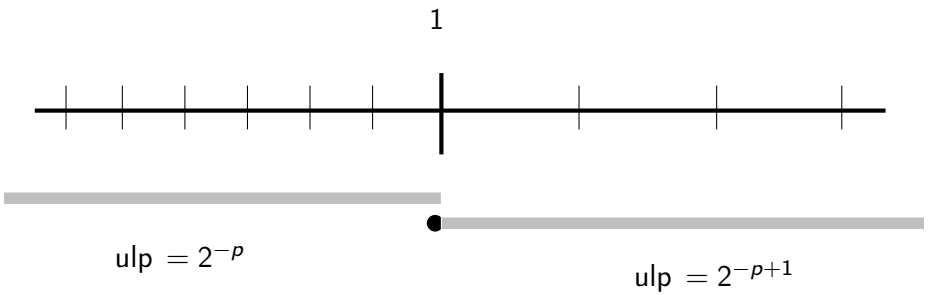


Fig.: *Les valeurs de* $GoldbergUlp(x)$ *au voisinage de* 1, *en supposant un système de base* 2 *et précision* p.

En arrondi au plus près

Majuscules : nombres VF, et minuscules : nombres réels.

Propriété (1)

En base 2,

$$|X - x| < \frac{1}{2}\, HarrisonUlp(x) \Rightarrow X = RN(x)$$

Cette propriété est fausse en base ≥ 3.

Propriété (2)

Pour toute valeur de la base,

$$X = RN(x) \Rightarrow |X - x| \leq \frac{1}{2}\, HarrisonUlp(x)$$

Avec l'ulp de Kahan, c'est la première propriété qui est vraie pour toute base, et la deuxième qui n'est vraie qu'en base 2.

Contre-exemple en base 3

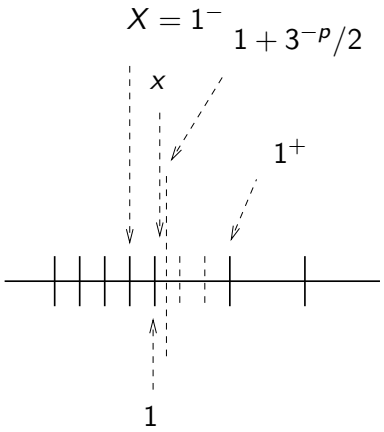


Fig.: La propriété 1 devient fausse en base 3. Ici, x satisfait $1 < x < 1 + \frac{1}{2}3^{-p}$, et $X = 1^- = 1 - 3^{-p}$. Nous avons $\mathrm{HarrisonUlp}\,(x) = 3^{-p+1}$, et $|x - X| < 3^{-p+1}/2$, de sorte que $|x - X| < \frac{1}{2}\,\mathrm{HarrisonUlp}\,(x)$. Pourtant, $X \neq \mathrm{RN}\,(x)$.

Avec les modes d'arrondi "dirigés"

Propriété

Pour toute valeur de la base,

$$X \in \{\, RD(x),\, RU(x)\} \Rightarrow |X - x| < 1\, HarrisonUlp(x)$$

Mais la réciproque est fausse. On trouve des valeurs X et x pour lesquelles $|X - x| < 1\,\mathrm{HarrisonUlp}\,(x)$, et pourtant $X \notin \{\,\mathrm{RD}\,(x),\,\mathrm{RU}\,(x)\}$ (cas où x est légèrement au dessus de 1 et $X = 1^-$, le prédécesseur de 1).

- Aucune propriété avec l'ulp de Kahan → on choisira en général l'ulp de Harrison ;
- plus d'information : `ftp://ftp.inria.fr/INRIA/publication/publi-pdf/RR/RR-5504.pdf`

Division utilisant la méthode de Newton-Raphson et un fma

Ici : version simplifiée d'une méthode utilisée sur Itanium. Précision p, base 2. On veut calculer a/b. On suppose $1 \leq a < 2$ et $1 \leq b < 2$ (mantisses de nombres VF normalisés). Calculons déjà $1/b$.

- Itération de Newton-Raphson pour calculer $1/b$:

$$y_{n+1} = y_n(2 - by_n)$$

- On va chercher $y_0 \approx 1/b$ dans une table adressée par les premiers (typiquement 6 à 10) bits de b ;
- on décompose l'itération en deux fmas :

$$\begin{cases} e_n & = & \mathrm{RN}(1 - by_n) \\ y_{n+1} & = & \mathrm{RN}(y_n + e_n y_n) \end{cases}$$

où $\mathrm{RN}(x)$ est l'arrondi au plus près de x.

Propriété

Si

$$\left|\frac{1}{b} - y_n\right| < \alpha 2^{-k},$$

où $1/2 < \alpha \leq 1$ *et* $k \geq 1$, *alors*

$$\begin{aligned} \left|\frac{1}{b} - y_{n+1}\right| &< b\left(\frac{1}{b} - y_n\right)^2 + 2^{-k-p} + 2^{-p-1} \\ &< 2^{-2k+1}\alpha^2 + 2^{-k-p} + 2^{-p-1} \end{aligned}$$

$\Rightarrow$ il semble qu'on peut s'approcher arbitrairement près de 2^{-p-1} (i.e., $1/2\,\mathrm{ulp}(1/b)$), sans jamais toutefois l'atteindre.

Exemple : double précision de la norme IEEE-754

Supposons $p = 53$ et $|y_0 - \frac{1}{b}| < 2^{-8}$ (petite table), on trouve

- $|y_1 - 1/b| < 0.501 \times 2^{-14}$
- $|y_2 - 1/b| < 0.51 \times 2^{-28}$
- $|y_3 - 1/b| < 0.57 \times 2^{-53} = 0.57\,\mathrm{ulp}(1/b)$

Comment aller plus loin ?

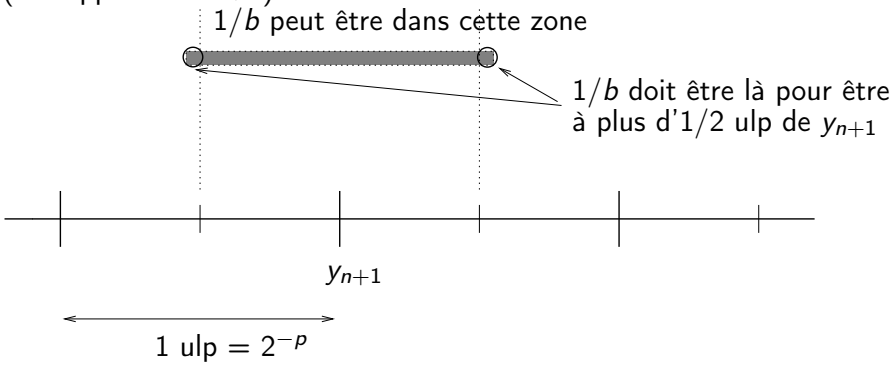
Propriété

Lorsque y_n *approche* $1/b$ *avec une erreur* $< 1\,ulp(1/b) = 2^{-p}$, *alors, comme* b *est multiple de* 2^{-p+1} *et* y_n *est multiple de* 2^{-p}, $1 - by_n$ *est multiple de* 2^{-2p+1}.
Or $|1 - by_n| < 2^{-p+1} \rightarrow 1 - by_n$ *est exactement représentable en virgule flottante avec une précision de* p *bits* $\rightarrow$ *il est calculé exactement.*

$$\Rightarrow \left|\frac{1}{b} - y_{n+1}\right| < b\left(\frac{1}{b} - y_n\right)^2 + 2^{-p-1}.$$

$$\left|y_n - \frac{1}{b}\right| < \alpha 2^{-p} \Rightarrow \left|y_{n+1} - \frac{1}{b}\right| < b\alpha^2 2^{-2p} + 2^{-p-1}$$

(en supposant $\alpha < 1$)



Qu'en déduit-on ?

- pour être à plus d'1/2 ulp de y_{n+1}, $1/b$ doit être à moins de $b\alpha^2 2^{-2p} < b2^{-2p}$ du milieu de deux nombres VF consécutifs ;
- implique que la distance entre y_n et $1/b$ est de la forme $2^{-p-1} + \epsilon$, avec $|\epsilon| < b2^{-2p}$;
- implique $\alpha < \frac{1}{2} + b2^{-p}$ et donc en supposant $p \geq 4$,

$$\left| y_{n+1} - \frac{1}{b} \right| < \frac{25}{64} b 2^{-2p} + 2^{-p-1}$$

- donc pour être à plus d'1/2 ulp de y_{n+1}, $1/b$ doit être à moins de $\frac{25}{64} b 2^{-2p}$ du milieu de deux nombres VF consécutifs.

- b est un nombre VF entre 1 et 2 (strictement) $\Rightarrow$ il est de la forme $B/2^{p-1}$ où B est un entier, $2^{p-1} < B \leq 2^p - 1$;
- le milieu de de nombres VF au voisinage de $1/b$ est de la forme $g = (2G+1)/2^{p+1}$ où G est un entier, $2^{p-1} \leq G < 2^p - 1$;
- on en déduit

$$\left| g - \frac{1}{b} \right| = \left| \frac{2BG + B - 2^{2p}}{B.2^{p+1}} \right|$$

- la distance entre $1/b$ et le milieu de 2 nombres machines est un multiple de $1/(B.2^{p+1}) = 2^{-2p}/b$. Elle n'est pas nulle (sauf dans le cas trivial où $b = 1$). Si $1/b$ est à moins de $\frac{25}{64} b 2^{-2p}$ de g, alors nécessairement

$$2BG + B - 2^{2p} = \pm 1.$$

Comment procède-t-on ?

- décomposition en facteurs premiers de $2^{2p}+1$ et $2^{2p}-1$
- on construit tous les couples (B, G) possibles tels que

$$\begin{aligned} 2^{p-1} < B \leq 2^p - 1 \\ 2^{p-1} \leq G \leq 2^p - 1 \\ B(2G+1) = 2^{2p} \pm 1 \end{aligned}$$

  En pratique il y en a très peu. Il y en a au moins un : $B = 2^p - 1$ et $G = 2^{p-1}$: vient de $2^{2p} - 1 = (2^p - 1)(2^p + 1)$;
- pour tous les B (et donc les $b = B/2^{p-1}$) qui ne sont pas dans la liste, on est certain que $y_{n+1} = \mathrm{RN}(1/b)$;
- autres valeurs : on essaie l'algorithme avec les 2 valeurs possibles de y_n à moins d'un ulp de $1/b$ (i.e. les 2 nombres VF qui encadrent $1/b$). La plupart du temps, ça marche (si une seule marche, essayer de "bricoler" la table qui donne y_0).

Application : double précision ($p = 53$)

On part de y_0 tel que $|y_0 - \frac{1}{b}| < 2^{-8}$. On calcule :

$$\begin{aligned} e_0 &= \mathrm{RN}(1 - b y_0) \\ y_1 &= \mathrm{RN}(y_0 + e_0 y_0) \\ e_1 &= \mathrm{RN}(1 - b y_1) \\ y_2 &= \mathrm{RN}(y_1 + e_1 y_1) \\ e_2 &= \mathrm{RN}(1 - b y_1) \\ y_3 &= \mathrm{RN}(y_2 + e_2 y_2) \quad \text{erreur} \leq 0.57 \text{ ulps} \\ e_3 &= \mathrm{RN}(1 - b y_2) \\ y_4 &= \mathrm{RN}(y_3 + e_3 y_3) \quad 1/b \text{ arrondi au plus près} \end{aligned}$$

. . . mais on n'a fait que la moitié du travail !

On a $1/b$, il reste à calculer a/b

- on part de $y_{n+1} = \mathrm{RN}(1/b)$ que l'on vient d'obtenir, et on calcule $q_0 = \mathrm{RN}(a y_{n+1})$;
- dans le cas $a > b$, donnera $|q_0 - a/b| < 1\,\mathrm{ulp}(a/b)$;
- sinon, en effectuant

$$\begin{cases} r_0 &= \mathrm{RN}(a - bq_0) \\ q_1 &= \mathrm{RN}(q_0 + r_0 y) \end{cases}$$

on obtient $|q_1 - a/b| < 1\,\mathrm{ulp}(a/b)$;

Théorème (Markstein)

Soit $\circ$ un des 4 modes d'arrondi. Si y est égal à $1/b$ arrondi au plus près, et si q est à au plus un ulp de a/b, alors en effectuant

$$\begin{cases} r &= RN(a - bq) \\ q' &= \circ(q + ry) \end{cases}$$

On obtient $q' = \circ(a/b)$.

Multiplication correctement arrondie par des constantes de précision arbitraire

- On veut calculer $\mathrm{RN}(Cx)$, où x est un nombre VF, et C une constante (i.e., connue à la compilation) réelle (qui n'est pas un nombre VF, sinon élémentaire).
- Valeurs typiques qui apparaissent dans des programmes numériques : π, $1/\pi$, $\ln(2)$, $\ln(10)$, e, $1/k!$, $B_k/k!$ (Euler-McLaurin), $\cos(k\pi/N)$ and $\sin(k\pi/N)$ (FFT), ...
- système de base 2 avec arrondi correct, précision p, disponibilité d'un fma

L'algorithme naïf

- on remplace C par $C_h = \mathrm{RN}(C)$;
- on calcule $\mathrm{RN}(C_h x)$ (instruction `y = Ch * x`).

p	Prop. de résultats correctement arrondis
5	0.93750
6	0.78125
7	0.59375
...	...
16	0.86765
17	0.73558
...	...
24	0.66805

Proportion de valeurs x pour lesquelles $\mathrm{RN}(C_h x) = \mathrm{RN}(Cx)$ pour $C = \pi$ et différentes valeurs de p.

L'algorithme proposé

On suppose pré-calculés les deux nombres VF

$$\begin{cases} C_h &= \mathrm{RN}(C), \\ C_\ell &= \mathrm{RN}(C - C_h), \end{cases} \tag{2}$$

Algorithme (MultC)

(Multiplication par C à l'aide d'un opérateur fma). *On calcule*

$$\begin{cases} u_1 &= RN(C_\ell x), \\ u_2 &= RN(C_h x + u_1). \end{cases} \tag{3}$$

Le résultat retourné est u_2.

Pour C et p donnés, on va chercher si l'algorithme MultC donne $\mathrm{RN}(Cx)$ pour tout nombre VF x.

Pour un C donné, l'algorithme est-il correct ?

Quelques remarques :

- Si l'algorithme MultC donne un résultat correct pour une constante C et un nombre VF x, il en sera de même pour $2^m C$ et $2^n x$.
- Si x est une puissance 2 ou si C est représentable exactement, ou si $C - C_h$ est une puissance de 2, alors $u_2 = \mathrm{RN}(Cx)$

$\Rightarrow$ Sans perte de généralité, on suppose que $1 < x < 2$ et $1 < C < 2$, que C n'est pas un nombre VF, et que $C - C_h$ n'est pas une puissance de 2.

Distance entre Cx et u_2

Propriété

Soient $x_{cut} = 2/C$ *et*

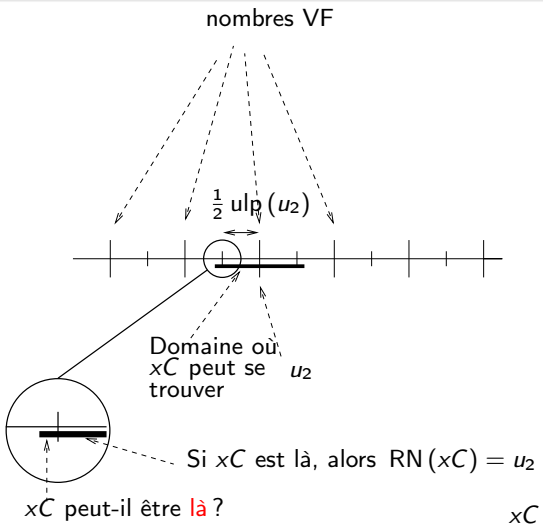
$$\epsilon_1 = |C - (C_h + C_\ell)| \tag{4}$$

- *Si* $x < x_{cut}$ *alors* $|u_2 - Cx| < 1/2\, ulp(u_2) + \eta$,
- *Si* $x \geq x_{cut}$ *alors* $|u_2 - Cx| < 1/2\, ulp(u_2) + \eta'$,

où

$$\begin{cases} \eta &= \frac{1}{2}\, ulp(C_\ell x_{cut}) + \epsilon_1 x_{cut}, \\ \eta' &= ulp(C_\ell) + 2\epsilon_1. \end{cases}$$

Analysons l'algorithme MultC



xC est à une distance $\leq 1/2\, ulp(u_2) + \eta$ (resp. η') de u_2. Si on montre que xC ne peut être à une distance $\leq \eta$ (resp. η') du milieu de 2 nombres VF consécutifs, alors $u_2 = RN(xC)$.

Remarques

- Si $x < x_{\mathrm{cut}}$ alors $xC < 2$, donc le milieu de deux nombres VF consécutifs au voisinage de xC est de la forme $A/2^p$, où A est un entier impair compris entre $2^p + 1$ et $2^{p+1} - 1$.
- Si $x \geq x_{\mathrm{cut}}$, alors le milieu de deux nombres VF consécutifs au voisinage de xC est de la forme $A/2^{p-1}$.

Petit rappel sur les fractions continues

Soit $\alpha \in \mathbb{R}$. On construit 2 suites (a_i) et (r_i) définies par :

$$\begin{cases} r_0 &= \alpha, \\ a_i &= \lfloor r_i \rfloor, \\ r_{i+1} &= 1/(r_i - a_i). \end{cases} \quad (5)$$

Si $\alpha \notin \mathbb{Q}$, ces suites sont définies $\forall i$, et le rationnel

$$\frac{p_i}{q_i} = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{\ddots + \cfrac{1}{a_i}}}}}$$

est la $i$ème réduite de α. Si $\alpha \in \mathbb{Q}$, s'arrête pour un i, et $p_i/q_i = \alpha$ exactement.

Théorème (2)

Soient $(p_j/q_j)_{j\geq 1}$ les réduites de α. Pour tout (p, q), t.q. $q < q_{n+1}$, on a

$$|p - \alpha q| \geq |p_n - \alpha q_n|.$$

□

Théorème (3)

Soient p, q des entiers non nuls et premiers entre eux. Si

$$\left|\frac{p}{q} - \alpha\right| < \frac{1}{2q^2}$$

alors p/q est une réduite de α.

□

Méthode 1 : utilisation du théorème 2

Soient $X = 2^{p-1}x$ et $X_{\text{cut}} = \lfloor 2^{p-1}x_{\text{cut}} \rfloor$.
Cas $x < x_{\text{cut}}$: on cherche s'il existe A entre $2^p + 1$ et $2^{p+1} - 1$ t.q.

$$\left|Cx - \frac{A}{2^p}\right| < \eta.$$

Ceci équivaut à

$$|2CX - A| < 2^p\eta$$

Soient $(p_i/q_i)_{i\geq 1}$ les réduites de $2C$, et k le + petit entier t.q. $q_{k+1} > X_{\text{cut}}$, et soit $\delta = |p_k - 2Cq_k|$.
Théorème 2 $\Rightarrow \forall A, X \in \mathbb{Z}$, avec $0 < X \leq X_{\text{cut}}$, $|2CX - A| \geq \delta$.

Par conséquent

- Si $\delta \geq 2^p\eta$ alors $|Cx - A/2^p| < \eta$ est impossible $\Rightarrow$ l'algorithme MultC donne le bon résultat pour tout $x < x_{\text{cut}}$;
- si $\delta < 2^p\eta$, on essaie MultC avec $x = q_k 2^{-p+1} \rightarrow$ soit on obtient un contre-exemple, soit on ne peut conclure

Cas $x > x_{\text{cut}}$: étude similaire (avec les réduites de C)

Exemple 1 : C égal à π

```
> method1(Pi/2,53);
Ch = 884279719003555/562949953421312 = 1.570796327...
Cl = 4967757600021511/81129638414606681695789005144064
= .6123233996e-16...
xcut = 1.2732395447351626862, Xcut = 5734161139222658
eta = .8069505497e-32
pk/qk = 6134899525417045/1952799169684491
delta = .9495905771e-16
OK for X < 5734161139222658
etaprime = .1532072145e-31
pkprime/qkprime = 12055686754159438/7674888557167847
deltaprime = .6943873667e-16
OK for 5734161139222658 <= X < 9007199254740992
```

$\Rightarrow$ On obtient toujours un résultat correct pour $C = 2^k\pi$, avec $C_h = 2^{k-48} \times 884279719003555$ et $C_\ell = 2^{k-105} \times 4967757600021511$.
Multiplier avec arrondi correct par π : une multiplication et un fma.

C	p	méthode 1	méthode 2	méthode 3
π	24	?	?	OK
π	53	OK	?	OK
π	64	?	OK	OK (c)
π	113	OK	OK	OK (c)
$1/\pi$	24	?	?	OK
$1/\pi$	53	contre-exemple 6081371451248382	?	seul contre-exemple $X =$ 6081371451248382
$1/\pi$	64	OK	OK	OK (c)
$1/\pi$	113	?	?	OK
$\ln 2$	24	OK	OK	OK (c)
$\ln 2$	53	OK	?	OK (c)
$\ln 2$	64	OK	?	OK (c)
$\ln 2$	113	OK	OK	OK (c)
$\frac{1}{\ln 2}$	24	?	OK	OK (c)
$\frac{1}{\ln 2}$	53	OK	OK	OK (c)
$\frac{1}{\ln 2}$	64	?	?	OK
$\frac{1}{\ln 2}$	113	?	?	OK

Autres exemples (double précision : $p = 53$)

Toujours en utilisant la même méthode :

- pour $C = 2^k/\pi$, on trouve des contre-exemples, les nombres x de la forme
$$6081371451248382 \times 2^m.$$
- pour $C = \frac{2^k}{\ln 10}$, on ne sait pas conclure (une autre méthode permet de montrer que l'algorithme donne toujours un résultat correct) ;
- pour $C = 2^k \ln(2)$, l'algorithme donne toujours un résultat correct.

Programmes Maple et Pari implantant ces méthodes :
`http://perso.ens-lyon.fr/jean-michel.muller/MultConstant.html`

Gestion des exceptions

- Nov. 1998, navire lance-missiles américain USS Yorktown, on a par erreur tapé un «zéro» sur un clavier $\rightarrow$ division par 0. Le programmeur n'avait pas songé que ce problème pourrait arriver $\rightarrow$ cascade d'erreurs $\rightarrow$ arrêt du système de propulsion.
- premier envol... et premier plongeon d'Ariane 5 (500 M$)

Vitesse horizontale de la fusée calculée sur des flottants 64 bits. Dans le programme du calculateur de bord, conversion vers un entier 16 bits. Il n'avait pas été envisagé que cette conversion puisse faire un dépassement. Fonctionnait très bien sur Ariane 4.

Juste quelques mots sur les exceptions dans la norme

- l'utilisateur peut (difficilement, et c'est un euphémisme) définir le comportement du programme dans les cas exceptionnels ;
- philosophie par défaut : le calcul doit toujours continuer ;
- le format comporte deux infinis, et deux zéros. Règles intuitives : $1/(+0) = +\infty$, $5 + (-\infty) = -\infty \ldots$;
- tout de même un truc bizarre : $\sqrt{-0} = -0$;
- Not a Number (NaN) : résultat de $\sqrt{-5}$, $(\pm 0)/(\pm 0)$, $(\pm\infty)/(\pm\infty)$, $(\pm 0) \times (\pm\infty)$, NaN $+3$, etc.
- parfois, des étrangetés. Comportement pour x grand de

$$\frac{x^2}{\sqrt{x^3+1}}$$

Quelques propriétés des NaN

- se propagent : $x+$ NaN $=$ NaN. Nécessaire (que faire d'autre ?) mais dangereux (embarqué) ;
- comparaisons : toute comparaison impliquant un NaN retourne faux (sauf une, voir suite) $\rightarrow$ $x \geq y$ n'est pas exactement le contraire de $x < y$. Les concepteurs de compilateurs doivent en tenir compte.
- si x est un NaN, le test "$x = x$" retourne "faux" et le test "$x \neq x$" retourne "vrai"
  $\rightarrow$ fournit un moyen simple de tester qu'une variable est un NaN ;

Les drapeaux correspondant aux exceptions

Aucune exception ne doit arrêter le calcul. Un mécanisme de drapeaux informe le système sur ce qui s'est produit. Les 5 drapeaux sont :

- opération invalide : le résultat par défaut est un NaN ;
- division par zéro : le résultat par défaut est un infini exact (p.ex., $1/0$, ou $\log(+0)$) ;
- dépassement de capacité vers $\pm\infty$: suivant le mode d'arrondi, le résultat est soit $\pm\infty$, soit (en valeur absolue) le + grand nombre VF fini ;
- dépassement vers 0 : le résultat est soit ± 0 soit un dénormalisé (important : dénormalisés $\rightarrow$ dégradation de la précision $\rightarrow$ doit être signalé) ;
- résultat inexact : le résultat d'une opération est inexact (permet par exemple de faire de l'arithmétique entière avec le format VF).

Une fois qu'un drapeau est levé, il le reste jusqu'à une remise a zéro volontaire.

Les codages internes de la norme IEEE-754

- simple précision : 32 bits (1 de signe, 8 d'exposant, $23 + 1$ de mantisse) ;
- double précision : 64 bits (1 de signe, 11 d'exposant, $52 + 1$ de mantisse) ;
- le premier bit de mantisse des normalisés est forcément un "1" $\rightarrow$ on ne le mémorise pas (convention du 1 implicite) ;
- l'exposant est biaisé : on représente l'exposant e par l'entier positif $e + b$ (simple précision : $b = 127$, double précision : $b = 1023$) ;
- écriture dans l'ordre (signe, exposant biaisé, mantisse) $\rightarrow$ comparaison lexicographique, comme pour des entiers.

Exposants réservés

- la plage d'exposants "normaux" utilisables est de -126 à $+127$ (biaisé : 1 à 254) en simple précision et de -1022 à $+1023$ (biaisé : 1 à 2046) en double précision ;
- 8 bits (resp. 11 bits) $\rightarrow$ on peut représenter les entiers de 0 à 255 (resp. 0 à 1023). Les valeurs extrêmes sont réservées :
  - l'exposant biaisé 0 code 0 et les dénormalisés. Pas de "1" implicite dans ce cas.
  - l'exposant maximum (255 ou 1023 $\rightarrow$ rien que des "1") code les valeurs spéciales $\pm\infty$ et `NaN`.

-0	1	00000000	00000000000000000000000
$+0$	0	00000000	00000000000000000000000
$-\infty$	1	11111111	00000000000000000000000
$+\infty$	0	11111111	00000000000000000000000
NaN	0	11111111	chaîne non nulle
5	0	10000001	01000000000000000000000

D'autres formats

format	taille totale	taille exposant	taille mantisse
double étendu IEEE-754	≥ 80	≥ 15	≥ 64
double étendu Intel	80	15	64
quad (SUN et 754-R)	128	15	113
Itanium "register format"	82	17	64

- formats "étendu" et "register format" de l'Itanium : pas de "1" implicite ;
- idée des formats étendus : calculs intermédiaires, fonctions élémentaires (n'était pas destiné à l'utilisateur de base) ;
- format des registres de l'Itanium : permet d'exécuter l'algorithme de division de nombres du format double étendu sans risque de dépassement. Permet aussi de calculer $\sqrt{x^2+y^2}$ sans "faux dépassement".

Quelques valeurs extrêmes, conversions

format	+ petit sous-normal	+ petit normalisé	+ grand fini
simple préc.	1.401×10^{-45}	1.175×10^{-38}	3.403×10^{38}
double préc.	4.941×10^{-324}	2.225×10^{-308}	1.798×10^{308}
préc. étend.	3.645×10^{-4951}	3.362×10^{-4932}	1.190×10^{4932}
quad préc.	6.475×10^{-4966}	3.362×10^{-4932}	1.190×10^{4932}

- nombre simple précision converti vers une chaîne décimale d'au moins 9 chiffres de mantisse, puis reconverti en simple précision $\rightarrow$ même résultat (idem double préc. avec 17 chiffres, PE avec 21 chiffres) ;
- chaîne décimale d'au plus 6 chiffres de mantisse convertie en SP, puis reconvertie en décimal avec le même nb de chiffres $\rightarrow$ même résultat (idem DP avec 15 chiffres, PE avec 18 chiffres)

But : l'écriture dans un fichier puis la relecture ne doivent pas entraîner d'erreur.

Quelques temps (en nb de cycles)

Processor	FP Add	FP Mult	FP Div
Pentium	3	3	39
Pentium Pro	3	5	18-38
Pentium III	3	5	32
Pentium IV	5	7	38
PowerPC 601	4	4	31
PowerPC 750	3	4	31
MIPS R10000	2-3	2-3	11-18
UltraSPARC III	4	4	24
Cyrix 5x86 and 6x86	4-9	4-9	24-34
Alpha21264	4	4	15
Athlon K6-III	3	3	20

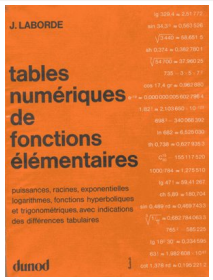
La norme IEEE 754 est en cours de révision

- les grandes lignes restent les mêmes ;
- ajouts fma, base 10, quelques considérations sur les fonctions élémentaires (sin, cos, exp, log, etc.) et leur arrondi correct ;
- voir `http://754r.ucbtest.org/`
- propositions de spécification de ces fonctions : `http://perso.ens-lyon.fr/jean-michel.muller/NumericalAlgorithms2004.pdf`

Les humains n'ont pas besoin des machines pour faire des bêtises

La sonde Mars Climate Orbiter s'est écrasée sur Mars en 1999. Une partie des concepteurs des logiciels supposait que l'unité de mesure était le mètre, et l'autre partie que c'était le pied.

Le dilemme du fabricant de tables



Considérons le nombre VF double précision ($p = 53$)

$$x = 1.11100100010110010110010100100110101111111001010010011 \times 2^{-10}$$

Nous avons

$$2^x = \overbrace{1.00000000010100111111110000101110\cdots0011}^{53 \text{ bits}}$$
$$0\underbrace{1111111111111111\cdots1111111111111111}_{59 \text{ “1”}}0100\cdots$$

Pas de standardisation des fonctions élémentaires (Ng)

Système	$\sin 10^{22}$
résultat exact	$-0.8522008497671888017727\cdots$
HP 48 GX	-0.852200849762
HP 700	0.0
HP 375, 425t (4.3 BSD)	$-0.65365288\cdots$
matlab V.4.2 c.1 pour Macintosh	0.8740
matlab V.4.2 c.1 pour SPARC	-0.8522
Silicon Graphics Indy	$0.87402806\cdots$
SPARC	-0.85220084976718879
IBM RS/6000 AIX 3005	$-0.852200849\cdots$
IBM 3090/600S-VF AIX 370	0.0
DECstation 3100	NaN
Casio fx-8100, fx180p, fx 6910 G	Error
TI 89	Trig. arg. too large

Arrondir correctement les fonctions élémentaires

- dans tout ce qui suit : base 2 ;
- Nombre VF x et entier m (avec $m > p$) $\rightarrow$ on peut calculer une approximation de $f(x)$ dont l'erreur sur la mantisse y est $\leq 2^{-m}$.
- Ce calcul peut être fait en employant un format plus large, ou en utilisant des algorithmes tels que Fast2Sum, Dekker, etc.
- obtenir un arrondi correct de $f(x)$ à partir de y ne sera pas possible si y est trop proche d'un point où l'arrondi change.

- en arrondi au plus près,

$$\overbrace{1.\underbrace{xxxxx\cdots xxx}_{p \text{ bits}}1000000\cdots000000}^{m \text{ bits}}\,xxx\cdots$$

ou

$$\overbrace{1.\underbrace{xxxxx\cdots xxx}_{p \text{ bits}}0111111\cdots111111}^{m \text{ bits}}\,xxx\cdots;$$

- avec les modes d'arrondi "dirigés",

$$\overbrace{1.\underbrace{xxxxx\cdots xxx}_{p \text{ bits}}0000000\cdots000000}^{m \text{ bits}}\,xxx\cdots$$

ou

$$\overbrace{1.\underbrace{xxxxx\cdots xxx}_{p \text{ bits}}1111111\cdots111111}^{m \text{ bits}}\,xxx\cdots.$$

Trouver une valeur de m au-delà de laquelle le problème ne se pose pas ?

- fonction f : sin, cos, arcsin, arccos, tg, arctg, exp, log, sh, ch,
- Théorème de Lindemann (x algébrique non nul $\Rightarrow e^x$ transcendant) $\rightarrow$ sauf pour les cas triviaux (e^0, ln(1), sin(0), arccos(1), etc.), lorsqu'on calcule $f(x)$ où x est un nombre VF, il existe une valeur de m, disons m_x, au delà de laquelle l'écriture binaire de $f(x)$ ne peut plus commencer comme au transparent précédent ;
- pour un format donné, nombre fini de nombres VF $\rightarrow$ il existe un $m_{\max} = \max_x(m_x)$ pour lequel, pour tout x, arrondir l'approximation de $f(x)$ sur $m_{\max}$ bits est équivalent à arrondir $f(x)$;
- problème : ce raisonnement ne nous donne aucune idée de la valeur de $m_{\max}$. Pourrait être énorme, ce qui n'est pas le cas en pratique.

Une borne de Baker (1975)

- $\alpha = i/j$, $\beta = r/s$, avec $i, j, r, s < 2^p$;
- $C = 16^{200}$;

$$|\alpha - \log(\beta)| > (p2^p)^{-Cp\log p}$$

Application : Pour calculer ln et exp en double précision ($p = 53$), il suffit de calculer une approximation intermédiaire bonne sur environ

$$10^{244} \text{ bits}$$

Soit $i/j \in \mathbb{Q}$ avec $\gcd(i,j) = 1$. On définit $H(i/j) = \max\{|i|, |j|\}$.

Théorème (Y. Nesterenko et M. Waldschmidt (1995))

Soit $(\alpha, \beta) \in \mathbb{Q}^2$. Soient A, B, et E des réels positifs, avec $E \geq e$, satisfaisant $A \geq \max(H(\alpha), e)$, $B \geq H(\beta)$. On a

$$|e^{\beta} - \alpha| \geq \exp\Big(-211 \times (\ln B + \ln\ln A + 2\ln(E|\beta|_+) + 10) \times (\ln A + 2E|\beta| + 6\ln E) \times (3.3\ln(2) + \ln E) \times (\ln E)^{-2}\Big),$$

where $|\beta|_+ = \max(1, |\beta|)$.

Application : Pour calculer ln et exp en double précision ($p = 53$), il faut faire les calculs intermédiaires sur environ 10^6 bits.

impossible → très cher

- la mantisse réelle y de $f(x)$ est de la forme :

$$y = y_0.y_1y_2\cdots y_{p-1}\overbrace{01111111\cdots11}^{kbits}xxxxx\cdots$$

ou

$$y = y_0.y_1y_2\cdots y_{p-1}\overbrace{10000000\cdots00}^{kbits}xxxxx\cdots$$

avec $k \geq 1$.

- En supposant qu'après la $p^{\text{ème}}$ position les "1" et les "0" apparaissent de manière équiprobable, la "probabilité" d'avoir $k \geq k_0$ est 2^{1-k_0} ;
- si on a N nombres virgule flottante en entrée, on observera environ $N \times 2^{1-k_0}$ valeurs pour lesquelles $k \geq k_0$;
- en pratique, le phénomène ne se produit donc plus dès que k_0 est significativement plus grand que $\log_2(N)$ (pour une seule valeur de l'exposant, dès que $k_0 >> p$).

	k										
p	1	2	3	4	5	6	7	8	9	10	11
1	1	0	0	0	0	0	0	0	0	0	0
2	1	1	0	0	0	0	0	0	0	0	0
3	1	2	0	1	0	0	0	0	0	0	0
4	3	3	1	1	0	0	0	0	0	0	0
5	9	4	0	2	1	0	0	0	0	0	0
6	16	7	6	2	1	0	0	0	0	0	0
7	33	14	8	3	2	2	2	0	0	0	0
8	67	27	16	11	5	1	0	0	0	0	1
9	132	57	34	20	8	1	1	1	0	2	0
10	255	128	71	27	15	7	4	2	2	0	1
11	506	265	113	62	35	18	15	4	3	1	2
12	1012	528	232	153	54	29	21	7	8	4	0
13	2049	1046	449	267	131	79	39	16	14	3	2
14	4080	2072	966	517	287	146	65	24	18	7	2
15	8160	4087	2152	994	502	247	133	53	28	13	8
16	16397	8151	4191	2043	1010	463	255	131	62	35	16

Tab.: Nombre d'occurrences des différentes valeur de k pour $\sin(x)$

Tab.: *Résultats pour de petites valeurs d'entrée, en double précision.*

Cette fonction	peut être remplacée par	lorsque
$\exp(\epsilon), \epsilon \geq 0$	1	$\epsilon < 2^{-53}$
$\exp(\epsilon), \epsilon \leq 0$	1	$\lvert\epsilon\rvert \leq 2^{-54}$
$\ln(1+\epsilon)$	ϵ	$\lvert\epsilon\rvert < \sqrt{2} \times 2^{-53}$
$2^{\epsilon}, \epsilon \geq 0$	1	$\epsilon < 1.4426\cdots \times 2^{-53}$
$2^{\epsilon}, \epsilon \leq 0$	1	$\lvert\epsilon\rvert < 1.4426\cdots \times 2^{-54}$
$\sin(\epsilon), \arcsin(\epsilon), \sinh(\epsilon), \sinh^{-1}(\epsilon)$	ϵ	$\lvert\epsilon\rvert \leq 1.4422\cdots \times 2^{-26}$
$\cos(\epsilon)$	1	$\lvert\epsilon\rvert < \sqrt{2} \times 2^{-27}$
$\cosh(\epsilon)$	1	$\lvert\epsilon\rvert < 2^{-26}$
$\tan(\epsilon), \tanh(\epsilon), \arctan(\epsilon), \tanh^{-1}(\epsilon)$	ϵ	$\lvert\epsilon\rvert \leq 1.817\cdots \times 2^{-27}$

- algorithmes de V. Lefèvre : basés sur une linéarisation des fonctions pour un préfiltrage ne laissant qu'un nombre raisonnable de cas à tester avec grande précision ;
- préfiltrage : variante de l'algorithme d'Euclide et utilisation du théorème des 3 distances ;
- double précision :
  - la plus utilisée ;
  - calculer tous les sinus des 2^{32} nombres simple-précision ne demande que quelques heures ;
  - plus hautes précisions : le nombre de cas à traiter (2^{128} en quadruple précision) est inatteignable dans un futur prévisible.

Tab.: *Pires cas pour l'exponentielle de nombres VF double précision.*

Intervalle	pire cas (binaire)
$[-\infty, -2^{-30}]$	$\exp(-1.11101101001100011000111011111011011000100111111101010 \times 2^{-27})$ $= 1.11111111111111111111111111100\cdots0111000100\quad 1\quad 1^{59}0001\ldots \times 2^{-1}$
$[-2^{-30}, 0)$	$\exp(-1.00000000000000000000000000000000000000000000000000000001 \times 2^{-51})$ $= 1.11111111111111\cdots1111111111111100\quad 0\quad 0^{100}1010\ldots \times 2^{-1}$
$(0, +2^{-30}]$	$\exp(1.11111111111111111111111111111111111111111111111111111111 \times 2^{-53})$ $= 1.00000000000000000000000000000000000000000000000000000000\quad 1\quad 1^{104}0101\ldots$
$[2^{-30}, +\infty]$	$\exp(1.01111111111111100111111111111111011000000000000100100 \times 2^{-32})$ $= 1.00000000000000000000000000000001011111111111111101000\quad 0\quad 0^{57}1101\ldots$ $\exp(1.10000000000000010111111111111110110111111111111011100 \times 2^{-32})$ $= 1.00000000000000000000000000000011000000000000000010111\quad 1\quad 1^{57}0010\ldots$ $\exp(1.10011110100111001011101111111110101100000100000001011 \times 2^{-31})$ $= 1.00000000000000000000000000000011001111010011100101111\quad 1\quad 0^{57}1010\ldots$ $\exp(110.00001111010100101111001101111010111011001111110100)$ $= 110101100.01010000101101000000100111001000101011101110\quad 0\quad 0^{57}1000\ldots$

Tab.: *Pires cas pour le logarithme népérien de nombres VF double précision.*

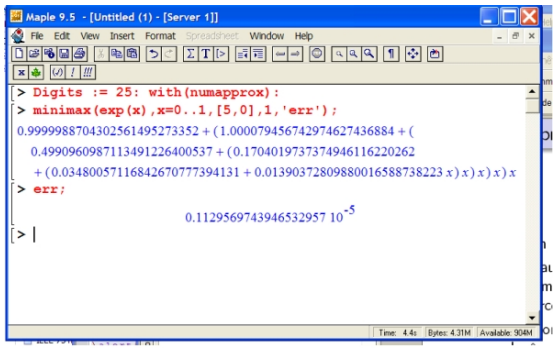
Interval	worst case (binary)
$[2^{-1074}, 1)$	$\log(1.11101010011100011101100001011100111011100000000100000 \times 2^{-509})$ $= -101100000.00101001011010100110011010110100001011111111\quad 1\quad 1^{60}0000\ldots$
	$\log(1.10010100011101101110001100000100110011010111110000111 \times 2^{-384})$ $= -100001001.10110110000011001010111101000111101100110101\quad 1\quad 0^{60}1010\ldots$
	$\log(1.00100110111010011100010011010011001001111100101100000 \times 2^{-232})$ $= -10100000.10101011001011000010010111100110100001000000100\quad 0\quad 0^{60}1001\ldots$
	$\log(1.01100001001110010101010111011100100000000010111110000 \times 2^{-35})$ $= -10111.11110000001011111001101110101111011000000000110101\quad 0\quad 1^{60}0011\ldots$
$(1, 2^{1024}]$	$\log(1.01100010101010001000011000010011011000101001101101110 \times 2^{678})$ $= 111010110.01000111100111101011101001111100100101110001\quad 0\quad 0^{64}1110\ldots$

Tab.: *Pire cas pour le cosinus de nombres VF double précision pris dans* $[1/64, 12867/8192]$. $12867/8192$ *est légèrement inférieur à* $\pi/2$.

Interval	worst case (binary)
$[1/64, 1]$	$\cos(1.10010111110011001101001111010010110001000011100011111 \times 2^{-6})$ $= 0.11111111111010111011001\cdots111000010101000\quad 1\quad 1^{55}0111\ldots$
$[1, \frac{12867}{8192}]$	$\cos(1.01101011100010100110001001110011110101111110000100001)$ $= 1.00110011011111111100010\cdots10110001010010\quad 1\quad 0^{54}1011\ldots \times 2^{-3}$

Meilleurs approximations polynomiales "tronquées"

- fonction continue f à évaluer en machine ;
- $\pm$, $\times$, $\div$ et comparaisons $\rightarrow$ approximation par des polynômes ou des fractions rationnelles par morceaux ;
- division bien plus lente que $+$ et $\times$ $\rightarrow$ polynômes ;
- norme L_∞, intervalle de la forme $[0, a]$.



En pratique

- théorie "continue" établie depuis longtemps (existence : Weierstrass, caractérisation : Tchebycheff, algorithme de calcul : Remès en 1934) ;
- oui mais... lorsqu'on va faire un programme ou un circuit de calcul, on a besoin de coefficients de taille finie dans un format donné ;
- arrondir chaque coefficient du meilleur approximant ?

But : meilleur approximant polynomial de f sur $[0, a]$ pour L_∞ parmi ceux dont les coefficients satisfont à des contraintes de taille données. Ici on cherchera un polynôme dont le coefficient de degré i est un multiple de 2^{-m_i}.

Polynômes de Tchebycheff

Définitions équivalentes :

$$\begin{cases} T_0(x) &= 1, \\ T_1(x) &= x, \\ T_n(x) &= 2xT_{n-1}(x) - T_{n-2}(x); \end{cases}$$

ou

$$T_n(x) = \begin{cases} \cos\left(n\cos^{-1}x\right) & \text{for } |x| \le 1, \\ \cosh\left(n\cosh^{-1}x\right) & \text{for } x > 1. \end{cases}$$

Propriété

Pour $n \ge 0$, on a

$$T_n(x) = \frac{n}{2}\sum_{k=0}^{\lfloor n/2\rfloor}(-1)^k\frac{(n-k-1)!}{k!(n-2k)!}(2x)^{n-2k}.$$

$\rightarrow$ T_n est de degré n et de coefficient dominant 2^{n-1}. Il a n racines réelles, toutes strictement entre -1 et 1.

Propriété (Polynômes unitaires de + petite norme)

$a, b \in \mathbb{R}$, $a < b$. Le polynôme unitaire de degré n ayant la + petite norme $||\cdot||_{[a,b]}$ est

$$\frac{(b-a)^n}{2^{2n-1}}T_n\left(\frac{2x-b-a}{b-a}\right).$$

Propriété (généralisation)

Soit $a \in (0, +\infty)$, soit $\alpha_0 + \alpha_1 x + \alpha_2 x^2 + \cdots + \alpha_n x^n = T_n^*\left(\frac{x}{a}\right)$. Soit $k \in \mathbb{N}$, $0 \le k \le n$, le polynôme

$$\frac{1}{\alpha_k}T_n^*\left(\frac{x}{a}\right)$$

a la plus petite norme $||\cdot||_{[0,a]}$ parmi ceux de degré $\le n$ dont le coefficient de degré k vaut 1. Cette norme vaut $|1/\alpha_k|$.

- $\mathcal{P}_n^{[m_0,m_1,\ldots,m_n]} = \left\{ \frac{a_0}{2^{m_0}} + \frac{a_1}{2^{m_1}}x + \cdots + \frac{a_n}{2^{m_n}}x^n,\ a_0,\ldots,a_n \in \mathbb{Z} \right\}$.
- p : meilleure approximation de f sur $[0,a]$ pour la norme infinie (problème "continu") ;
- $\hat{p}$ polynôme obtenu en arrondissant le coefficient de degré i de p au multiple de 2^{-m_i} le plus proche pour tout i ;
- $\epsilon = ||f-p||_{[a,b]}$;
- $\hat{\epsilon} = ||f-\hat{p}||_{[a,b]}$;
- on cherche $p^\star$ tel que

$$||f-p^\star||_{[a,b]} = \min_{q\in\mathcal{P}_n^{[m_0,m_1,\ldots,m_n]}} ||f-q||_{[a,b]}.$$

On suppose $\hat{\epsilon} > \epsilon$, sinon $\hat{p} = p^\star$.

Soit K vérifiant $\epsilon < K \leq \hat{\epsilon}$. On cherche $p^\star \in \mathcal{P}_n^{[m_0,m_1,\ldots,m_n]}$ tel que

$$||f-p^\star||_{[a,b]} = \min_{q\in\mathcal{P}_n^{[m_0,m_1,\ldots,m_n]}} ||f-q||_{[a,b]}$$

et

$$||f-p^\star||_{[a,b]} \leq K. \quad (6)$$

Lorsque $K = \hat{\epsilon}$, ce problème a une solution puisque $\hat{p}$ satisfait (6).
Important : $p^\star$ n'est en général pas égal à $\hat{p}$.

Trouver $p^\star$ en se ramenant à un nombre fini de cas

Rappel :

Propriété

Soit $a \in (0,+\infty)$, soit $\alpha_0+\alpha_1 x+\alpha_2 x^2+\cdots+\alpha_n x^n = T_n^*\left(\frac{x}{a}\right)$. Soit $k \in \mathbb{N}$, $0 \leq k \leq n$, le polynôme $\frac{1}{\alpha_k} T_n^*\left(\frac{x}{a}\right)$ a la plus petite norme $||\cdot||_{[0,a]}$ parmi ceux de degré $\leq n$ dont le coefficient de degré k vaut 1. Cette norme vaut $|1/\alpha_k|$.

Soit q un polynôme dont le coefficient de degré i est $p_i+\delta_i$. On a :

$$||q-p||_{[0,a]} \geq \frac{|\delta_i|}{|\alpha_i|},$$

Si pour un i donné, $|\delta_i| > (\epsilon+K)|\alpha_i|$, alors $||q-p||_{[0,a]} > \epsilon+K$ donc $q \neq p^\star$ puisque

$$||q-f||_{[0,a]} \geq ||q-p||_{[0,a]} - ||p-f||_{[0,a]} > K.$$

Puisque $2^{m_i}p_i^\star$ est entier, on a donc

$$\underbrace{\lceil 2^{m_i}(p_i-(\epsilon+K)|\alpha_i|)\rceil}_{c_i} \leq 2^{m_i}p_i^\star \leq \underbrace{\lfloor 2^{m_i}(p_i+(\epsilon+K)|\alpha_i|)\rfloor}_{d_i}, \quad (7)$$

Pour $2^{m_i}p_i^\star$, on a (d_i-c_i+1) valeurs possibles.

$\Rightarrow \prod_{i=0}^n (d_i-c_i+1)$ polynômes possibles à tester. Pour chacun d'eux, on calcule la distance à f pour la norme infinie.

Lorsque $\prod_{i=0}^n (d_i-c_i+1)$ est trop grand, il y a d'autres méthodes, soit exactes (parcours de polytopes), soit approchées (algorithme LLL).

Quelques exemples

	f	$[a, b]$	m	ϵ	$\hat{\epsilon}$	K
1	cos	$\left[0, \frac{\pi}{4}\right]$	[12, 10, 6, 4]	$1.13\ldots10^{-4}$	$6.93\ldots10^{-4}$	$\hat{\epsilon}/2$
2	exp	$\left[0, \frac{1}{2}\right]$	[15, 14, 12, 10]	$2.62\ldots10^{-5}$	$3.96\ldots10^{-5}$	$< \hat{\epsilon}$
3	exp	$\left[0, \log\left(1+\frac{1}{2048}\right)\right]$	[56, 45, 33, 23]	$1.18\ldots10^{-17}$	$2.36\ldots10^{-17}$	$< \hat{\epsilon}$
4	$\arctan(1+x)$	$\left[0, \frac{1}{4}\right]$	[24, 21, 18, 17, 16]	$2.38\ldots10^{-8}$	$3.77\ldots10^{-8}$	$< \hat{\epsilon}$
5	exp	$\left[-\frac{\log(2)}{256}, \frac{\log(2)}{256}\right]$	[25, 17, 9]	$8.27\ldots10^{-10}$	$3.31\ldots10^{-9}$	$< \hat{\epsilon}$
6	exp	$\left[-\frac{\log(2)}{256}, -\frac{\log(2)}{256}\right]$	[28, 19, 9]	$8.27\ldots10^{-10}$	$3.31\ldots10^{-9}$	$< \hat{\epsilon}$
7	$\frac{\log(3/4+x)}{\log(2)}$	$[-1/4, 1/4]$	[12, 9, 7, 5]	$6.37\ldots10^{-4}$	$7.73\ldots10^{-4}$	$< \hat{\epsilon}$
8	$\frac{\log(\sqrt{2}/2+x)}{\log(2)}$	$\left[\frac{1-\sqrt{2}}{2}, \frac{2-\sqrt{2}}{2}\right]$	[12, 9, 7, 5]	$6.37\ldots10^{-4}$	$9.34\ldots10^{-4}$	$< \hat{\epsilon}$

	Cette méthode	Polytope (d)	T_1	T_2	Gain en bits
1	330	1 (4)	0.62	0.26	≈ 1.5
2	84357	9 (20)	0.51	2.18	≈ 0.375
3	9346920	15 (20)	0.99	1.77	≈ 0.22
4	192346275	1 (20)	0.15	0.55	≈ 0.08
5	1	0 (4)	0.05	0	0
6	4	1 (4)	0.03	0.10	≈ 0.41
7	38016	2 (15)	0.13	0.69	≈ 0.06
8		12 (20)	0.59	5.16	≈ 0.26

Un exemple "grandeur nature" dans la bibliothèque CRLIBM

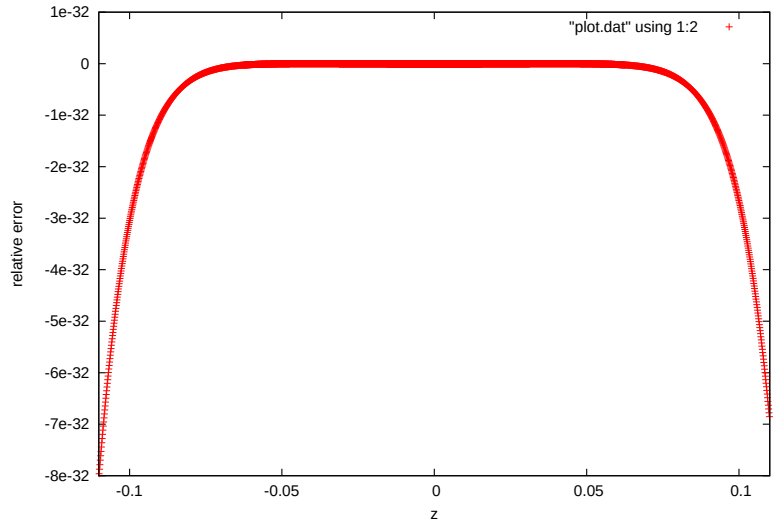
- implantation de la fonction arcsin au voisinage de 1 avec arrondi correct ;
- on se ramène au calcul de

$$g(z) = \frac{\arcsin(1-(z+m)) - \frac{\pi}{2}}{\sqrt{2\cdot(z+m)}}$$

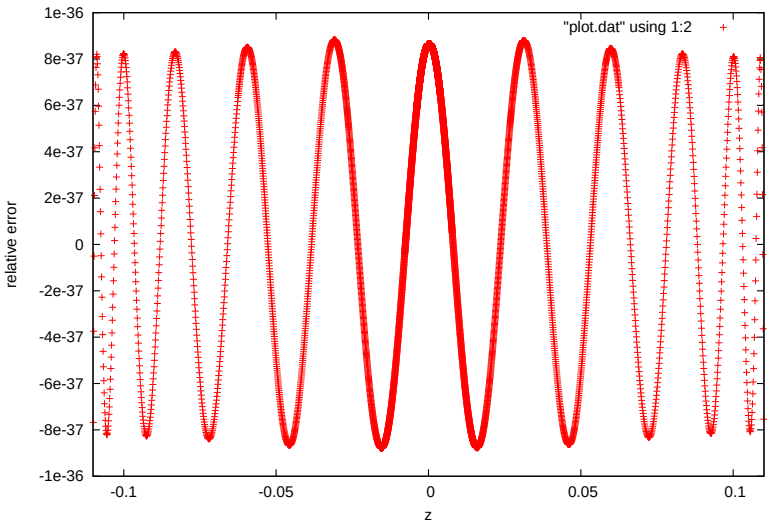
où $\mathrm{0xBFBC28F800009107} \leq z \leq \mathrm{0x3FBC28F7FFFF6EF1}$ (soit environ $-0.110 \leq z \leq 0.110$) et $m = \mathrm{0x3FBC28F80000910F} \simeq 0.110$;

- approximation polynomiale de degré 21.

En "tronquant" le meilleur polynôme "classique"



En utilisant notre approche



Un peu de multi-précision

Itération arithmético-géométrique (Gauss-Legendre)

$$\begin{cases} a_{n+1} &= \dfrac{a_n + b_n}{2} \\ b_{n+1} &= \sqrt{a_n b_n}. \end{cases}$$

convergence quadratique vers limite commune $A(a_0, b_0) =$ moyenne arithmético-géométrique de a_0 et b_0.
Gauss :

$$A(1, x) = \frac{\pi}{2F(x)},$$

où

$$F(x) = \int_0^{\pi/2} \frac{d\theta}{\sqrt{1 - (1 - x^2)\sin^2\theta}}.$$

A quoi ça sert ?

Donne un algorithme de calcul rapide de

$$F(x) = \int_0^{\pi/2} \frac{d\theta}{\sqrt{1 - (1 - x^2)\sin^2\theta}}.$$

or,

$$\begin{aligned} F(4/s) &= \ln(s) + \frac{4\ln(s) - 4}{s^2} \\ &\quad + \frac{36\ln(s) - 42}{s^4} + \frac{1200\ln(s) - 1480}{3s^6} \\ &\quad + O\left(\frac{1}{s^8}\right), \end{aligned}$$

si s est suffisament grand, $F(4/s)$ est une bonne approximation de $\ln(s) \rightarrow$ pour les très grandes précisions (milliers de chiffres), donne algorithme rapide de calcul du logarithme. Exponentielle : méthode de Newton. Contraire de ce qu'on fait pour les précisions moyennes (centaines de chiffres).

Quelques logiciels utiles

- CRLIBM : fonctions mathématiques avec arrondi correct.
  `http://lipforge.ens-lyon.fr/projects/crlibm/`
  La documentation qui explique les méthodes utilisées est à
  `http://lipforge.ens-lyon.fr/frs/download.php/41/crlibm-0.10.pdf`
- GAPPA : outil de vérifications de propriétés VF (p.ex. bornes) et de génération de preuves formelles.
  `http://lipforge.ens-lyon.fr/www/gappa/`
- MPFR : arithmétique multi-précision avec arrondi correct.
  `http://www.mpfr.org/`
- MPFI (basé sur MPFR) : arith. d'intervalles multi-précision.
  `http://perso.ens-lyon.fr/nathalie.revol/software.html`
- PARI/GP : calculs rapides en arithmétique (factorisations, théorie algébrique des nombres, courbes elliptiques...).
  `http://pari.math.u-bordeaux.fr/`

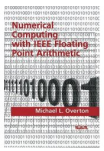
Tester votre environnement virgule flottante

- PARANOIA (W. Kahan et ses élèves) :
  `http://www.netlib.org/paranoia/`
- UCB Test (plus récent, plus complet) :
  `http://www.netlib.org/fp/ucbtest.tgz`
- MPCHECK : test des fonctions mathématiques :
  `http://www.loria.fr/~zimmerma/free/`
- méthodes de recherches de "cas difficiles" pour les opérateurs arithmétiques : John Harrison (factorisation), Michael Parks (remontée de Hensel) ;

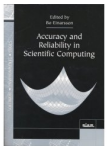
La virgule flottante sur le web

- le site de W. Kahan (père de la norme IEEE 754, de l'arithmétique du 8087 et de la HP35) :
  `http://http.cs.berkeley.edu/~wkahan/`
- le site de D. Hough sur la révision de la norme
  `http://www.validlab.com/754R/`
- l'article de Goldberg "What every computer scientist should know about Floating-Point arithmetic"
  `http://www.validlab.com/goldberg/paper.pdf`
- l'équipe Arénaire du LIP (ENS Lyon)
  `http://www.ens-lyon.fr/LIP/Arenaire/`
- l'équipe Algorithmique numérique et parallélisme du LIP6 (Paris 6)
  `http://www-anp.lip6.fr/`
- l'équipe SPACES du Loria (Nancy)
  `http://www.loria.fr/equipes/spaces/`
- ma propre page
  `http://perso.ens-lyon.fr/jean-michel.muller/`

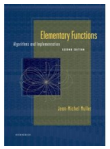
Quelques livres sur la virgule flottante



Michael Overton
Numerical Computing with IEEE Floating Point Arithmetic
Siam, 2001



Bo Einarsson
Accuracy and Reliability in Scientific Computing
Siam, 2005



Jean-Michel Muller
Elementary Functions, algorithms and implementation, 2ème édition
Birkhauser, 2006

Marc Daumas et al.
Qualité des calculs sur ordinateur
Masson, 1997.