P. CARRÉ

Note sur quelques sous-programmes de précision étendue dite "améliorée".

Complément à "Travaux du mois de janvier 1968"

I . Rappels. Réprésentation des nombres en virgule flottante

Tout nombre x peut se mettre sous la forme

$$x = a \times 2^n \qquad \begin{cases} 0{,}5 \leqslant a < 1 \\ -1 < a \leqslant -0{,}5 \end{cases}$$

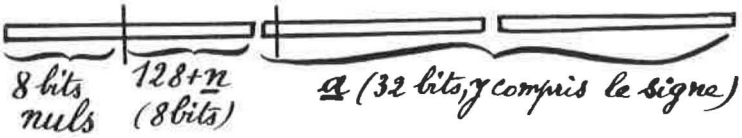
Dans les sous-programmes de calcul fournis par IBM on représente en mémoire a et n selon l'un ou l'autre des modèles ci-dessous

1) Précision normale

Le 1er bit de a donne le signe, le suivant a pour poids 2^{-1}, le 24e a pour poids 2^{-23}. On voit donc que a ne peut varier que par sauts de 2^{-23} soit en valeur relative 2^{-22} si $a \approx 0{,}5$ et 2^{-23} si $a \approx 1$; mais un arrondi correct du dernier bit gardé pourrait permettre une précision de représentation deux fois meilleure.

n a pour valeurs extrêmes - 128 et + 127

2) Précision étendue



On voit immédiatement que le 32e bit de a a une valeur relative comprise entre 2^{-30} (soit 10^{-9}) et 2^{-31} (soit $10^{-9,3}$)

La précision de représentation et, dans une certaine mesure, la précision de calcul pourraient être 2 fois meilleures au moyen d'un arrondi correct du dernier bit mais cela n'est pas garanti. On doit donc compter avec des erreurs relatives de 10^{-9} dans chaque multiplication ou division (L'erreur relative dans une addition ou une multiplication est, évidemment, fonction des données). De plus, certains sous-programmes peuvent donner localement des erreurs un peu plus fortes (par exemple racine carrée).

Les limites de variation de n ne posent pas de problème, le nombre maximal que l'on peut représenter étant $2^{127} \times ... ...(1 - 2^{-23}) \approx 10^{38,2}$ et le plus petit nombre positif non nul étant $2^{-128} \times \frac{1}{2} \approx 10^{-38,8}$

II . But et caractères généraux de la précision étendue améliorée

On voit donc, en gros, que l'on dispose de sous-programmes à 10^{-7} et de sous-programmes à 10^{-9}, les uns ou les autres pouvant être appelés par les ordres Fortran habituels (mais pas les deux, dans un même programme). Or, nous faisons des expériences à 10^{-8} près. Il n'est pas tolérable que des années d'effort expérimental soient anéanties par dix opérations consécutives, chacune introduisant une erreur de 10^{-9}.

IBM diffuse des sous-programmes à 10^{-19} mais

1°/ ils sont d'une lourdeur désespérante ;
2°/ ils ne comportent que les 4 opérations de l'école primaire ;
3°/ ils consomment beaucoup de place en mémoire (pour s'y loger et pour loger les variables) ;
4°/ ils sont utilisables en Fortran au moyen des ordres d'appel explicite des sous-programmes et non au moyen des formules de sorte qu'on peut programmer en Fortran (FORmula TRANslator !) tout sauf les formules ;
5°/ enfin leur précision est surabondante pour longtemps encore !

Nous les avons néanmoins commandés. Il n'est pas impossible qu'ils recèlent quelques astuces.

Il m'a donc semblé utile de faire des sous-programmes ayant les caractères suivants :

1°/ bonne rapidité
2°/ comportant au départ les 4 opérations, la racine carrée, le logarithme, l'exponentielle mais extensibles si besoin est.
3°/ ne prenant pas une place exagérée en mémoire
4°/ utilisables en Fortran sans modification des ordres si possible.
5°/ précision égale à la précision théorique.

La solution adoptée a consisté à récupérer les 8 bits perdus à gauche du 1er mot pour y mettre les bits 33 à 40 de a.

En arrondissant correctement lors de chaque opération (entrée comprise) le ~~premier~~ dernier bit ~~non~~ conservé, on doit pouvoir réduire l'erreur de représentation et l'erreur de calcul à une valeur au plus égale à 2^{-40}, soit une erreur relative comprise entre 2^{-39} (soit $10^{-11,7}$) et 2^{-40} (soit 10^{-12}).

En contre-partie, on remplace des opérations portant sur deux mots-machine par des opérations portant sur trois mais parmi lesquels un peu plus de deux et demi seulement sont significatifs. L'expérience montre que ce petit demi-mot libre contenant initialement l'exposant donne beaucoup de souplesse à la programmation.

III . Caractères particuliers et principes mathématiques des divers sous-programmes.

1°/ Entrées et Sorties

. Pour mettre au point des programmes de calcul il faut pouvoir fournir des nombres à la machine et il est commode qu'elle puisse imprimer les résultats. Il faut donc commencer par les entrées et les sorties.

. Les programmes actuels sont provisoires, ils ne permettent pas l'utilisation des ordres Fortran de lecture et d'écriture.

. Lorsqu'il y avait conflit entre la longueur et le temps d'exécution j'ai sacrifié la vitesse à l'encombrement.

- Entrée par cartes perforées. On spécifie combien il y a de nombres sur la carte. Le nombre de chiffres à la partie entière est limité seulement par la plus grande valeur absolue représentable (donc 37 ou 38). Il n'y a aucune limitation au nombre de décimales. Evidemment, le programme ne conservera qu'un nombre de chiffres significatifs correspondant à la précision de représentation (en général 12 mais le 13^{e} peut agir sur l'arrondi en binaire). Les nombres sont cadrés de façon absolument arbitraire sur la carte mais ils doivent être séparés au moins par un blanc. Les nombres décimaux possèdent un point ou une virgule (indifféremment). Pour les nombres négatifs, les blancs sont permis entre le signe - et le premier chiffre. Pour les nombres positifs le signe + est facultatif.

- Sortie par "l'imprimante de pupitre", autrement dit la machine à écrire.
On choisit l'impression en rouge ou en noir.
On spécifie le nombre de positions réservées à la partie entière, le nombre de décimales et, si on le désire, le "caractère" à sortir après le nombre (par exemple "retour chariot", "tabulation" ....).

Le nombre de positions pour la partie entière est limité à 15 mais la plus grande valeur absolue que l'on puisse imprimer est $2^{31} - 1 = 2\ 147\ 483\ 647$. Si le nombre de chiffres de la partie entière du nombre (augmenté de 1 si le nombre est négatif) est supérieur au nombre de positions spécifiées, le nombre sera imprimé à partir de la position actuelle de la tête, sans être tronqué. Son cadrage ne correspondra évidemment pas aux prévisions du

programmeur (imprévoyant). Si le nombre est $\geqslant 2^{31}$, la machine imprime "INFINI".

Le nombre de décimales est limité à 9. Si on spécifie davantage, la machine n'en imprime que 9. La dernière décimale est arrondie au plus proche quel que soit le nombre de décimales imprimé. Si ce nombre est non nul, partie entière et partie fractionnaire sont séparées par une virgule.

Il peut être nécessaire de multiplier ou diviser un résultat par une certaine puissance de dix avant de demander l'impression, c'est-à-dire choisir une unité adaptée aux limitations exposées ci-dessus.

2°/ Sous-programmes mathématiques

Pour tous ces sous-programmes, j'ai cherché la rapidité d'exécution à condition qu'elle n'exige pas un accroissement trop important de la longueur.

∘ addition. Le principe est le suivant, en supposant $n' \leqslant n$

$$a \times 2^n + a' \times 2^{n'} = (a + a'/2^{n-n'}) \times 2^n$$

d'où les opérations à effectuer :

- séparation des exposants des deux nombres
- division par $2^{n-n'}$, c'est-à-dire décalage à droite de $n-n'$ positions, du nombre qui n'a pas le plus grand exposant.
- addition (avec reports éventuels)
- contrôle de dépassement de capacité,
- normalisation
- réunion de l'exposant avec arrondi binaire.

∘ soustraction . On se ramène à l'addition par un changement de signe après séparation de l'exposant.

∘ multiplication . Principe: $a \times 2^n \times a' \times 2^{n'} = aa' \times 2^{n+n}$

Le produit aa' fait intervenir 9 produits partiels dont 3 sont négligeables. Ces produits partiels étant en double longueur, alors que le produit est en triple longueur, il faut bien faire attention dans le cas de facteurs négatifs. Voici les principales étapes, sans insister autrement sur le détail des tests :

- séparation des exposants
- calcul ~~du nouvel exposant~~ et sommation des 6 produits partiels
- calcul du nouvel exposant
- réunion de l'exposant avec arrondi binaire.

∘ <u>division</u>. Principe $\frac{a \times 2^n}{a' \times 2^{n'}} = \frac{a}{a'} \times 2^{n-n'}$

Malheureusement, le quotient ne peut s'exprimer simplement en fonction de "quotients partiels".

Toutefois, les 3 mots représentant a et a' ayant des poids proportionnels à 1, 2^{-16}, 2^{-32}, on peut appliquer une technique de développement limité et utiliser les opérations câblées (multiplication et division). Les difficultés proviennent des risques de dépassement de capacité et aussi des signes. Mais les opérations câblées sont algébriques, cette méthode <u>doit</u> pouvoir marcher.

Néanmoins on peut s'attendre à des difficultés de mise au point.

J'ai donc écrit un autre programme, très simple, (mais plus long en ce qui concerne le nombre d'instructions) opérant par soustractions successives du diviseur décalé chaque fois de une position à droite. Si la soustraction est possible on marque un 1 au quotient, sinon, c'est-à-dire si le dividende change de signe, on rajoute le diviseur et on marque un zéro au quotient ; on décale le diviseur, on recommence.

Ce programme a fonctionné pour ainsi dire au premier essai. Il effectue la division en 13 ms.

Mais l'autre fonctionne aussi et il effectue la division en 2,8 ms.

Le choix de celui qu'il faut garder est donc évident. (A titre documentaire, le programme IBM, 500 fois moins précis dure 2,4 ms).

∘ <u>racine carrée</u> . Principe: $\sqrt{a \times 2^n} = \sqrt{a' \times 2^{2p}} = \sqrt{a'} \times 2^p$

avec $\begin{cases} a' = a \\ 2p = n \end{cases}$ si n est pair

$\begin{cases} a' = a/2 \\ 2p = n + 1 \end{cases}$ si n est impair

On voit donc que $0{,}25 \leqslant a' < 1$

D'autre part, si $\alpha = \sqrt{a'}\ (1 + \varepsilon)$ est une valeur approchée de la racine carrée,

$\frac{1}{2}\left(\alpha + \frac{a'}{\alpha}\right) \approx \sqrt{a'}\left(1 + \frac{\varepsilon^2}{2}\right)$ est une bien meilleure approximation.

C'est la méthode de Newton bien connue.

Dans notre cas, le plus économique semble être d'appli-

quer deux fois la méthode de Newton. Si ε est l'erreur relative de la valeur de départ, $\frac{\varepsilon^4}{8}$ sera l'erreur après 2 applications de la méthode de Newton.

Pour avoir $\frac{\varepsilon^4}{8} < 10^{-12}$ il faut $\varepsilon < \sqrt[4]{8} \times 10^{-3} \approx 1{,}7 \times 10^{-3}$

Dans chacun des deux sous-intervalles $0{,}25 \leqslant a' < 0{,}5$ et $0{,}5 \leqslant a' < 1$ une fonction homographique donne une précision bien suffisante moyennant 2 additions et une division, un polynôme du second degré donne une précision moins bonne mais encore meilleure que 10^{-3}, donc suffisante, moyennant 2 additions et 2 multiplications. La division (câblée) dure plus longtemps que deux multiplications ; j'ai donc adopté le polynôme du second degré. Son calcul est fait en simple longueur. La 1ère application de la méthode de Newton donne un résultat en double longueur. La 2e application donne le résultat cherché. De plus, on remplace les moyennes par des sommes, grâce à un cadrage approprié et ces sommes sont simplifiées du fait que le mot de droite de l'un des deux termes est nul. Le résultat est obtenu automatiquement normalisé. Je serais déçu si la durée dépassait 5 ms.

Si le nombre donné est négatif, le programme le remplace par son opposé mais il le signale.

- <u>logarithme népérien</u>

Principe : $\mathrm{Ln}\ (\underline{a} \times 2^n) = \mathrm{Ln}\ \underline{a} + \underline{n}\ \mathrm{Ln}\ 2$

On a toujours $0{,}5 \leqslant \underline{a} < 1$

On réduit l'intervalle de variation de $\underline{a}$ et on le centre (géométriquement) par rapport à 1 en multipliant $\underline{a}$ par l'un des coefficients

$2^{7/8}\ \ 2^{5/8}\ \ 2^{3/8}\ \ 2^{1/8}$ selon sa position par rapport aux nombres $2^{-1}\ \ 2^{-3/4}\ \ 2^{-1/2}\ \ 2^{-1/4}\ \ 1$.

En désignant par 2^c celui des coefficients ci-dessus qui nous intéresse, la formule devient

$$\mathrm{Ln}(\underline{a} \times 2^n) = \mathrm{Ln}(2^c \times \underline{a}) + (\underline{n} - \underline{c})\ \mathrm{Ln}\ 2$$

On passe alors à la variable z définie par

$$2^c \times a = \frac{1 + z}{1 - z}$$

On constate que l'intervalle de variation de z est environ $-0{,}043 \leqslant z \leqslant 0{,}043$. On utilise un développement de $\mathrm{Ln}\ \frac{1 + z}{1 - z}$ (qui ne contient que les puissances impaires de z, donc rapidement convergent) en fraction continue. Pour obtenir les 12 chiffres significatifs désirés j'utilise la 4e réduite qui s'écrit :

$$\operatorname{Ln}\frac{1+z}{1-z} \approx \cfrac{2z}{1-\cfrac{z^2}{3-\cfrac{4z^2}{15-\cfrac{9z^2}{35}}}}$$

Remarque : le repérage de l'intervalle contenant a se fait en simple précision. Pour la fraction continue, on commence en précision simple puis double, puis triple ; n - c est calculé comme différence de nombres entiers en prenant comme unité 1/8. Les autres opérations utilisent la précision maximale.

- exponentielle

Principe $e^x = 2^{x/\operatorname{Ln} 2}$

posons E = partie entière de $\frac{x}{\operatorname{Ln}2}$

$F = \frac{x}{\operatorname{Ln}2} - E$ $\qquad 0 \leqslant F < 1$

$F' = F\operatorname{Ln}2 = x - \operatorname{Ln}2 \times E$ $\qquad 0 \leqslant F' < \operatorname{Ln}2$

ou $x = F' + \operatorname{Ln} 2 \times E$

On a donc : $e^x = 2^{F'/\operatorname{Ln}2} \times 2^E = 2^E \times e^{F'}$

Considérons alors les 4 intervalles définis par les bornes

$$0 ; \tfrac{1}{4}\operatorname{Ln}2 ; \tfrac{1}{2}\operatorname{Ln}2 ; \tfrac{3}{4}\operatorname{Ln}2 ; \operatorname{Ln}2$$

auquels nous associons les valeurs $i = 0 \quad i = 1 \quad i = 2 \quad i = 3$

Posons $z = F' - (i + \frac{1}{2}) \times \frac{1}{4}\operatorname{Ln}2$ alors $-\frac{1}{8}\operatorname{Ln}2 \leqslant z < \frac{1}{8}\operatorname{Ln}2$

soit $-0{,}086 \leqslant z < 0{,}086$

On a finalement

$$e^x = 2^E \times 2^{\frac{2i+1}{8}} \times e^z$$

Une fois E et i déterminés, il faut calculer F' puis z avec toute la précision possible. Le terme 2^E est très simplement représenté sous la forme habituelle $a \times 2^n$ avec $a = 0{,}5 \quad n = E + 1$; les 4 valeurs de

$2^{\frac{2i+1}{8}}$, soit $2^{1/8} \quad 2^{3/8} \quad 2^{5/8} \quad 2^{7/8}$ sont conservées en mémoire, d'ailleurs on les utilise déjà pour le logarithme. Il reste à calculer e^z.

La relation $e^z = \frac{1}{e^{-z}}$ conduit à utiliser un développement de la forme $\frac{P(z)}{P(-z)}$.

On pourrait penser prendre pour P(z) le développement de $e^{z/2}$ en série de Mac-Laurin ; mais il y a des polynômes beaucoup plus économiques .

Ainsi $P(z) \equiv z^3 - 12\ z^2 + 60\ z - 120$ doit donner 12 chiffres exacts.

Mis sous une forme analogue à une fraction continue, le quotient $\frac{P(z)}{P(-z)}$ s'écrit $e^z \approx \frac{P(z)}{P(-z)} = -1 + \cfrac{24}{-z + 12 - \cfrac{50}{z + \cfrac{10}{z}}}$

28.1.68