RAIRO. INFORMATIQUE THÉORIQUE

JACQUES DUMONTET

À propos de l'optimalité des algorithmes numériques : étude du calcul de χ^n

RAIRO. Informatique théorique, tome 12, n° 2 (1978), p. 109-123

<http://www.numdam.org/item?id=ITA_1978__12_2_109_0>

R.A.I.R.O. Informatique théorique/Theoretical Computer Science
(vol. 12, n° 2, 1978, p. 109 à 123)

A PROPOS DE L'OPTIMALITÉ DES ALGORITHMES NUMÉRIQUES : ÉTUDE DU CALCUL DE x^n (*)

par Jacques DUMONTET (1)
Communiqué par J.-F. PERROT

Résumé. — *Un algorithme est souvent qualifié d'optimal lorsqu'il nécessite « un minimum d'opérations ». Encore faut-il, dans le cas d'un problème numérique, que ce ne soit pas au détriment de la précision du résultat. Nous appliquons ce double critère au calcul de* x^n.

I. INTRODUCTION

Le calcul de x^n, pour x réel et n entier, par l'algorithme dichotomique classique, est décrit par l'organigramme 1 ci-après. Les parties entre crochets, sur cet organigramme, sont les assertions qui prouvent la validité de cet algorithme : elles illustrent la méthode de *preuve des programmes par assertions inductives*, imaginée par Floyd [2], axiomatisée par Hoare [4], puis exposée par Wirth [10], Leroy [7], Arsac [1].

Le but du programme est d'affecter à la variable y la valeur de x^n.

L'assertion-clé est $[x^n = y \bigstar z^p]$, qui devient $[x^n = y]$ lorsque $p = 0$, ce qui prouve la validité du programme.

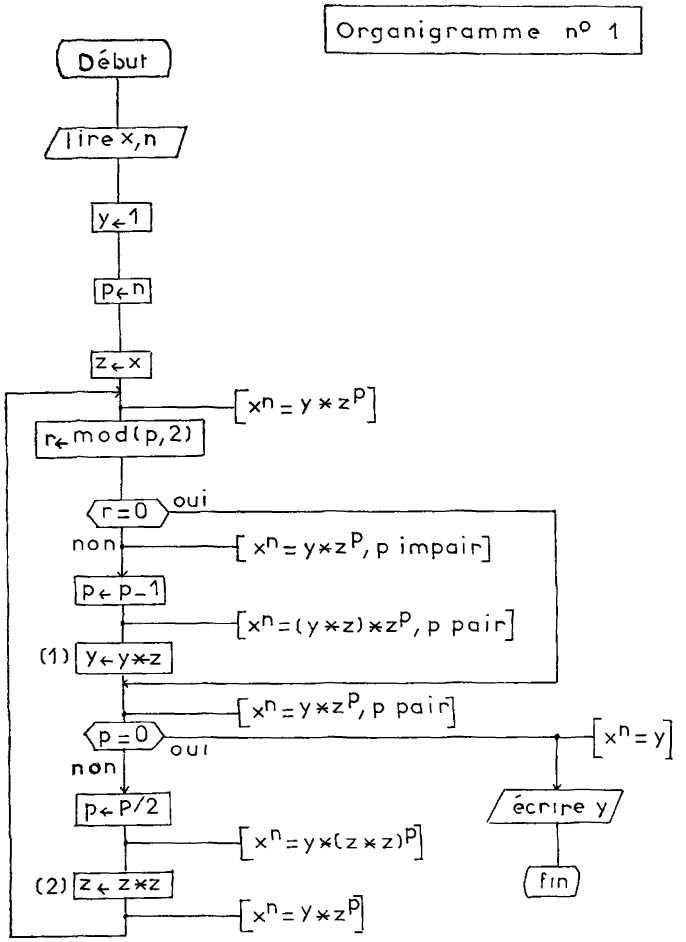
Voici l'organigramme du calcul de x^n, accompagné de sa preuve (*voir* organigramme 1).

Les erreurs de calcul

Malheureusement, l'arithmétique à précision limitée des ordinateurs entraîne des erreurs successives au cours des calculs; des auteurs comme La Porte [6] et Vignes [9] se sont spécialisés dans l'étude de ces phénomènes (méthode de permutation-perturbation).

Ici, les erreurs sont dues aux multiplications figurant dans les instructions (1) et (2). En effet, sur un ordinateur donné, les mantisses des représentations des nombres réels sont limitées à co chiffres, où co est une constante de l'ordinateur. Lorsqu'on multiplie entre elles les représentations de 2 nombres réels, la mantisse du résultat a 2 co-1 ou 2 co chiffres, et il faut n'en garder que co. Il y a donc, selon l'arithmétique de l'ordinateur utilisé, arrondi ou troncature,

(*) Reçu juin 1977, révisé janvier 1978.
(1) Université de Poitiers.



avec, dans l'un ou l'autre cas, introduction d'une erreur. Pour alléger l'exposé, nous envisagerons uniquement le cas des troncatures.

La preuve de la validité du calcul de x^n ne tenant pas compte de ces erreurs de calcul, nous nous fixons les objectifs suivants :

OBJECTIF 1 : construire une preuve, adaptée à l'arithmétique des ordinateurs courants, qui prenne en compte les erreurs de calcul successives. Cette preuve doit donc :

1° établir que y est une approximation de x^n;

2° fournir un majorant *explicite* de l'erreur commise.

Cette erreur peut être très élevée. Ainsi, sur l'ordinateur Iris 45 (CII), c'est

l'algorithme dichotomique classique qui est utilisé pour calculer x^n, et, en simple précision, il peut arriver que *tous* les chiffres du résultat soient faux.

Exemple :

$$\left.\begin{array}{l} x = (1.003\mathrm{FF})_{16} \\ n = 170\,000 \end{array}\right\} \Rightarrow \quad y = 85. \bigstar 10^{70},$$

alors que

$$x^n = 99. \bigstar 10^{70}.$$

Nous nous fixons donc un second objectif :

OBJECTIF 2 : on se fixe préalablement le nombre e de chiffres *exacts* désirés pour l'approximation y de x^n. Il faut donc imaginer une arithmétique originale permettant d'atteindre ce but, et effectuer les multiplications figurant dans l'algorithme dichotomique au moyen de cette arithmétique.

Il faudra construire une preuve, qui devra :

1° établir que y est une approximation de x^n;

2° montrer que les e premiers chiffres de y sont exacts.

Arithmétique proposée

Afin d'obtenir la précision souhaitée pour le résultat, on représente les mantisses des nombres réels avec un nombre de chiffres variable, à la discrétion du programmeur. Ce type d'arithmétique existait sur l'ordinateur IBM 1620. En écrivant les sous-programmes nécessaires, nous l'avons simulée sur l'ordinateur Iris 45.

Soient en particulier c un entier supérieur ou égal à 1, et y et z, les représentations, en virgule flottante normalisée, de deux nombres réels. La taille des mantisses de y et z dépend des calculs précédents.

Nous noterons $y \underset{c}{\bigstar} z$ le résultat obtenu en tronquant les mantisses de y et z à partir du $(c+1)$-ième chiffre, et en multipliant entre eux les 2 nombres ainsi obtenus. On ne tronque pas la mantisse du résultat qui a donc, au plus, 2 c chiffres ; (mais elle pourra être tronquée par la suite à l'occasion d'une nouvelle multiplication...).

Il nous suffit de chercher en fonction des données x, n, e une constante c permettant, en utilisant l'arithmétique proposée, d'obtenir une approximation y de x^n ayant e chiffres exacts. (Les mantisses des nombres à multiplier seront donc uniformément tronquées à c chiffres.)

Mais nous mettrons en évidence le rôle non uniforme des itérations des instructions (1) et (2) vis-à-vis de l'erreur finale. Ainsi, la seule première ité-

ration de l'instruction (2) peut entraîner à elle seule la moitié de l'erreur globale. D'où l'idée d'*optimiser* le programme en faisant varier le nombre de chiffres des mantisses au cours des calculs, ce que permet l'arithmétique proposée.

Nous nous fixons donc un troisième objectif :

OBJECTIF 3 : le programme proposé doit être optimal.

Recherche d'un critère d'optimalité

DÉFINITION : Nous appelons multiplication *complète* la multiplication de 2 *nombres* entre eux, et multiplication *élémentaire*, la multiplication de 2 *chiffres* entre eux.

Exemple : la multiplication complète, selon la méthode manuelle traditionnelle, de 27 par 356 nécessite 6 multiplications élémentaires.

Knuth [5], pour l'optimisation du calcul de x^n, retient comme critère le nombre de multiplications complètes. Mais il ne tient pas compte des pertes de précision dues aux troncatures des résultats partiels successifs. Au contraire, MacCarthy [8], en s'appuyant sur des résultats établis par Graham, Yao et Yao [3], prend pour critère le nombre de multiplications *élémentaires* (*chiffre à chiffre*) et garde *tous les chiffres*, d'où un résultat *rigoureusement exact*, mais un *temps de calcul généralement prohibitif.*

Par souci d'efficacité, nous proposons donc un critère « intermédiaire » entre celui de Knuth et celui de MacCarthy :

Critère d'optimalité

Un algorithme qui fournit une approximation y de x^n *avec e chiffres exacts* est optimal s'il nécessite un minimum de multiplications *élémentaires.*

THÉORÈMES : *A chacun des trois objectifs définis plus haut, correspond un théorème, que nous allons énoncer :*

soient b la base de l'ordinateur utilisé, l le nombre total d'itérations des instructions (1) *et* (2).

THÉORÈME 1 : *Soit, dans une arithmétique standard, co le nombre de chiffres de toutes les mantisses, les chiffres suivants étant tronqués. L'algorithme dichotomique fournit une approximation y de* x^n. *La déviation introduite, définie par :* $d = x^n/y$, *vérifie la propriété :* $d \in [1, (1+b^{1-co})^{2n}[$.

THÉORÈME 2 : *Pour obtenir, en utilisant l'algorithme dichotomique, une approximation y de x^n ayant e chiffres exacts, il suffit de remplacer, grâce à l'arithmétique proposée l'instruction* (1) : $y \leftarrow y \bigstar z$ *par* : $y \leftarrow y \underset{c}{\bigstar} z$, *et l'instruction* (2) : $z \leftarrow z \bigstar z$ *par* : $z \leftarrow z \underset{c}{\bigstar} z$ *avec* $c \geqq e+1+\log_b(2n)$.

THÉORÈME 3 : *Pour obtenir, en utilisant l'algorithme dichotomique, une approximation y de x^n ayant e chiffres exacts, il suffit de remplacer, grâce à l'arithmétique proposée, l'instruction* (1) : $y \leftarrow y \bigstar z$ *par* : $y \leftarrow y \underset{c}{\bigstar} z$ *avec* : $c \geqq e+1+\log_b(2l)$ *et l'instruction* (2) : $z \leftarrow z \bigstar z$ *par* $= z \leftarrow z \underset{c'}{\bigstar} z$ *avec* $c' \geqq e+1+\log_b(2l \bigstar p)$ [*c est une constante, et c' une variable qu'il faut recalculer, en fonction de p, avant chaque nouvelle exécution de* (2)].

Cette solution n'est pas optimale, mais elle est meilleure que la précédente.

L'objet de cet article est de démontrer ces trois théorèmes.

La démarche qui sera suivie est intéressante en elle-même, du fait de sa nouveauté, et de ses possibilités d'extension à d'autres algorithmes.

II. PRÉSENTATION DE L'ALGORITHME MODIFIÉ

Nous avons présenté dans l'introduction l'algorithme dichotomique classique, accompagné de sa preuve (organigramme 1). Cette preuve ne tient pas compte des erreurs de calcul [instructions (1) et (2)]. Nous resterons volontairement dans le cadre de l'algorithme dichotomique. Mais nous allons :

– introduire des notations nouvelles, afin de construire une preuve tenant compte des erreurs de calcul;

– effectuer les multiplications figurant dans les instructions (1) et (2) dans la nouvelle arithmétique que nous proposons, afin d'obtenir *y* avec *e* chiffres exacts :

Les modifications

Nous introduisons dans le programme précédent un indice d'itération *i* : Initialisé à 0, *i* sera incrémenté avant chaque multiplication. Si *l* est le nombre de multiplications, *i* varie donc de 0 à *l*.

Nous noterons I_1 (resp I_2) l'ensemble des valeurs que peut avoir *i* lorsque s'exécute l'instruction (1) [resp. (2)].

A chaque valeur de *i*, nous faisons correspondre un entier c_i, de façon à remplacer l'opérateur $\bigstar$ par l'opérateur $\underset{c_i}{\bigstar}$ défini dans l'introduction.

Un des problèmes fondamentaux sera, par la suite, de déterminer des valeurs adéquates pour les c_i, afin d'obtenir y avec e chiffres exacts.

A chaque nouvelle valeur de i correspond un nouveau calcul de p; en effet, p est modifié avant chaque multiplication (*voir* programme précédent). On notera p_i la valeur de p calculée juste après avoir incrémenté i.

Soit, pour une valeur i de l'indice d'itération, une expression algébrique A_i contenant la sous-expression $(y \bigstar z)$. Soit A'_i l'expression obtenue à partir de A_i en remplaçant $(y \bigstar z)$ par $(y \underset{c_i}{\bigstar} z)$. Nous appelons *déviation sur* A_i, *due à* $(y \underset{c_i}{\bigstar} z)$, la quantité : $d_i = A_i/A'_i$.

On a la propriété évidente : $A_i = A'_i \bigstar d_i$.

Cette propriété sera utilisée, dans l'organigramme n° 2, pour écrire l'assertion (f) :

— $(y \bigstar z)$ est remplacé par $(y \bigstar z) \bigstar d_i$ et pour écrire l'assertion (i) :

— $(z \bigstar z)^{p_i}$ est remplacé par $(z \underset{c_i}{\bigstar} z)^{p_i} \bigstar d_i$ (le rôle donné à y dans la définition de la déviation est joué ici par z).

Compte tenu de ces différentes modifications, le nouveau programme se présente ainsi (*voir* organigramme 2).

De notre système d'assertions, il résulte qu'à la sortie du programme, on a

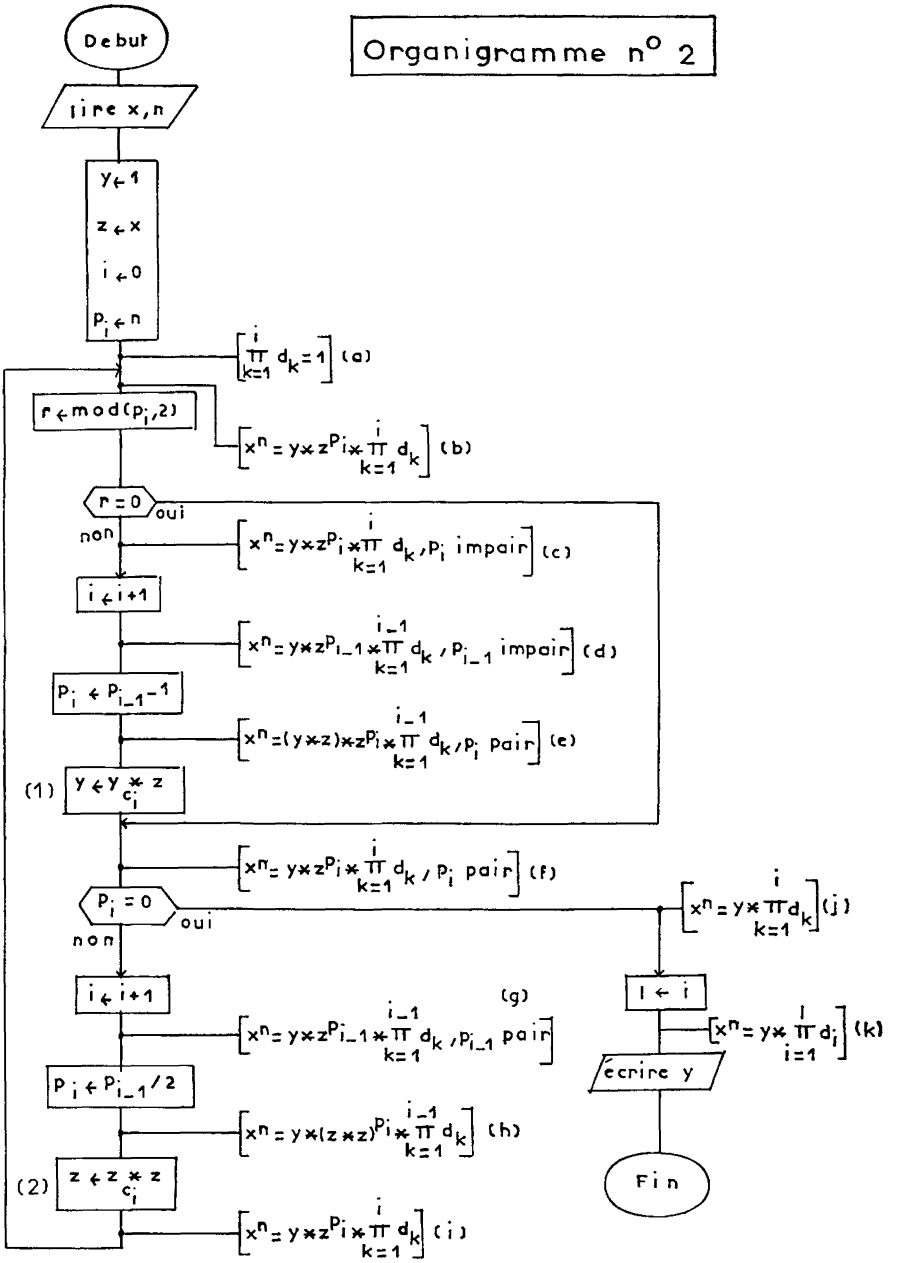
$$x^n = y \bigstar \prod_{i=1}^{l} d_i .$$

L'étude de la quantité $\prod_{i=1}^{l} d_i$ va maintenant nous permettre de démontrer les théorèmes annoncés.

III. DÉMONSTRATION DES THÉORÈMES

Nous majorerons d'abord la quantité $d = \prod_{i=1}^{l} d_i$, qui représente ce que nous appelons la « déviation globale », en fonction des « coefficients de troncature » c_i.

Nous aborderons ensuite la détermination de ces coefficients, en vue d'obtenir l'approximation y de x^n avec e chiffres exacts, puis de rendre optimal de calcul défini par l'organigramme 2 : Le cas *actuellement* le plus fréquent est celui des mantisses de longueur fixe, déterminée une fois pour toutes sur un ordinateur donné. Nous l'étudierons tout d'abord, en parallèle avec le cas des mantisses de longueur fixe, mais déterminée par notre programme lui-même; enfin, nous traiterons le cas où on peut choisir des longueurs de mantisses variables suivant les opérations effectuées.


Organigramme n° 2
Debut
lire x,n
y ← 1
z ← x
i ← 0
$p_i \leftarrow n$
$\prod_{k=1}^{i} d_k = 1$ (a)
r ← mod(p_i,2)
$x^n = y * z^{p_i} * \prod_{k=1}^{i} d_k$ (b)
r = 0
oui
non
$x^n = y * z^{p_i} * \prod_{k=1}^{i} d_k, p_i$ impair (c)
i ← i+1
$x^n = y * z^{p_{i-1}} * \prod_{k=1}^{i-1} d_k, p_{i-1}$ impair (d)
$p_i \leftarrow p_{i-1} - 1$
$x^n = (y*z) * z^{p_i} * \prod_{k=1}^{i-1} d_k, p_i$ pair (e)
(1) $y \leftarrow y *_{c_i} z$
$x^n = y * z^{p_i} * \prod_{k=1}^{i} d_k, p_i$ pair (f)
$p_i = 0$
oui
non
$x^n = y * \prod_{k=1}^{i} d_k$ (j)
i ← i+1
$x^n = y * z^{p_{i-1}} * \prod_{k=1}^{i-1} d_k, p_{i-1}$ pair (g)
$p_i \leftarrow p_{i-1}/2$
$x^n = y * (z*z)^{p_i} * \prod_{k=1}^{i-1} d_k$ (h)
(2) $z \leftarrow z *_{c_i} z$
$x^n = y * z^{p_i} * \prod_{k=1}^{i} d_k$ (i)
I ← i
$x^n = y * \prod_{i=1}^{I} d_i$ (k)
écrire y
Fin

1. Étude des déviations

a) *Déviation sur une multiplication*

Nous définissons la déviation δ_i *sur la multiplication* $y \underset{c_i}{\bigstar} z$ par

$$\delta_i = \frac{y \bigstar z}{y \underset{c_i}{\bigstar} z}.$$

REMARQUE : il ne faut pas confondre cette quantité avec $d_i = A_i/A_i'$, qui est la déviation *sur* A_i due à la multiplication $y \underset{c_i}{\bigstar} z$ [*voir* II)].

PROPRIÉTÉ : Pour chaque valeur de i, on a : $1 \leqq \delta_i < (1+b^{1-c_i})^2$.

Démonstration : Posons $y = y_1 + y_2$ et $z = z_1 + z_2$, où y_1 et z_1 sont les valeurs de y et z tronqués à c_i chiffres.

On a

$$y \bigstar z = (y_1+y_2) \bigstar (z_1+z_2) = y_1 \bigstar z_1 + y_2 \bigstar z_1 + y_1 \bigstar z_2 + y_2 \bigstar z_2.$$

D'où

$$\delta_i = \frac{y \bigstar z}{y \underset{c_i}{\bigstar} z} = \frac{y \bigstar z}{y_1 \bigstar z_1} = 1 + \frac{y_2}{y_1} + \frac{z_2}{z_1} + \frac{y_2 \bigstar z_2}{y_1 \bigstar z_1}.$$

Mais

$$\frac{y_2}{y_1}, \quad \frac{z_2}{z_1} \in [0,\ b^{1-c_i}[.$$

D'où le résultat annoncé.

b) *Déviation* d_i *due à* (1)

Pour passer de l'assertion (*e*) à l'assertion (*f*), on a remplacé $y \bigstar z$ par $y \underset{c_i}{\bigstar} z \bigstar d_i$, pour tout $i \in I_1$.

On a donc

$$d_i = \frac{y \bigstar z}{y \underset{c_i}{\bigstar} z} = \delta_i,$$

c'est la déviation sur une multiplication.

c) *Déviation* d_i *due à* (2)

Pour passer de l'assertion (*h*) à l'assertion (*i*), on a remplacé $(z \underset{c_i}{\bigstar} z)^{p_i}$ par $(z \underset{c_i}{\bigstar} z)^{p_i} \bigstar d_i$, pour tout $i \in I_2$.

On a donc

$$d_i = \left(\frac{z \bigstar z}{\underset{c_i}{z \bigstar z}} \right)^{p_i} = \delta_i^{p_i},$$

où δ_i est la déviation sur une multiplication.

d) *Déviation globale*

Nous la définissons par

$$d = \frac{x^n}{y}.$$

De l'assertion de sortie de notre programme, il résulte que

$$d = \prod_{i=1}^{l} d_i = \prod_{i \in I_1} d_i \bigstar \prod_{i \in I_2} d_i.$$

D'où en appliquant (a), (b) et (c) :

$$d < \prod_{i \in I_1} (1 + b^{1-c_i})^2 \bigstar \prod_{i \in I_2} (1 + b^{1-c}\, i)^{2p_i}.$$

2. Cas des mantisses de longueur fixe

Supposons que tous les c_i sont égaux : $c_i = c$, $\forall\, i \in [1, l]$.

Sous cette hypothèse, on peut établir un majorant facilement utilisable de la déviation globale :

Soit l_1 le nombre d'itérations de l'instruction (1). On a la propriété :

$$l_1 + \sum_{i \in I_2} p_i = n.$$

La formule établie en 1 (d) devient donc :

$$d < (1 + b^{1-c})^{2n}.$$

REMARQUE: Dans la première itération de l'instruction (1), y n'est pas tronqué, car $y = 1$.

D'où finalement : $d < (1 + b^{1-c})^{2n-1}$.

REMARQUE : En faisant $c = \mathrm{co}$, dans ce qui précède, on tombe dans le cas d'une arithmétique standard, où toutes les mantisses sont tronquées à co chiffres, à une légère différence près : la mantisse du résultat y a $2c-1$ ou

$2c$ chiffres, dans l'arithmétique que nous avons proposée. En arithmétique standard, cette mantisse sera tronquée, passant de $2\,co-1$ ou $2\,co$ chiffres à co chiffres, ce qui ajoute une nouvelle déviation, d' : $1 \leqq d' < 1+b^{1-co}$.

La prise en compte de ces deux remarques donne le

THÉORÈME 1 :

$$d \in [1, (1+b^{1-co})^{2n}[.$$

Recherche de la précision désirée

Si la mantisse de y a e chiffres, il suffit, pour que ces e chiffres soient exacts, d'avoir :

$$d = \frac{x^n}{y} \leqq 1+b^{-e}+b^{-2e} \qquad \text{(P)}$$

En fait, la mantisse de y a $2c-1$ ou $2c$ chiffres. N'en garder que e entraînerait une déviation supplémentaire. Nous excluons cette troncature superflue, et gardons comme critère, pour s'assurer que y a e chiffres exacts, la propriété (P)

Soit

$$c \geqq e+1+\log_b(2n)$$

c est la longueur uniforme des mantisses que nous allons multiplier.

Alors

$$d = (1+b^{1-c})^{2n} \leqq (1+b^{-e-\log_b(2n)})^{2n}$$

mais

$$(1+\varepsilon)^k < 1+k\varepsilon+k^2\varepsilon^2 \qquad \text{si} \quad k\varepsilon < 3/2$$

or

$$2nb^{-e-\log_b(2n)} = b^{-e} < 3/2$$

donc

$$d < 1+b^{-e}+b^{-2e}$$

y a donc e chiffres exacts.

Nous avons donc démontré le :

THÉORÈME 2 : *Pour obtenir e chiffres exacts, il suffit de multiplier, dans l'arithmétique que nous proposons, des mantisses de longueur constante c, avec*

$$\boxed{c \geqq e+1+\log_b(2n).}$$

COROLLAIRE : *Si le nombre* co *de chiffres des mantisses, en arithmétique standard, vérifie :* $co \geqq e+1+\log_b(2n)$, *il suffira d'utiliser l'arithmétique standard pour obtenir au moins e chiffres exacts.*

3. Cas des mantisses de longueur variable

L'observation des résultats (b) et (c) du 1 montre que la première itération de l'instruction (2) peut, à elle seule, entraîner la moitié de l'erreur globale. En effet on a $d_i = \delta_i^{p_i}$, et p_i est divisé approximativement par 2 à chaque étape. Nous proposons donc de calculer les c_i (nombres de chiffres à conserver dans les mantisses successives), de telle sorte que les d_i (déviations successives sur x^n) soient aussi uniformes que possible. Notre objectif est d'obtenir y avec e chiffres exacts en effectuant moins de multiplications élémentaires qu'en appliquant la méthode du théorème 2.

Démonstration du théorème 3 :

Cas de l'instruction (1) :

Soit $c_i \geqq e+1+\log_b(2\,l)$ pour tout $i \in I_1$.

On a :

$$d_i^l < (1+b^{1-c_i})^{2l} \quad (cf.\ 1\ (a) \text{ et } (b)).$$

Donc

$$d_i^l < (1+b^{-e-\log_b(2l)})^{2l}.$$

Un calcul analogue à celui du théorème 2 donne :

$$d_i^l < 1+b^{-e}+b^{-2e}$$

Cas de l'instruction (2) :

Soit $c_i \geqq e+1+\log_b(2\,lp_i)$ pour tout $i \in I_2$.

On a :

$$d_i^l < (1+b^{1-c_i})^{2lp_i} \quad (cf.\ 1\ (a) \text{ et } (c)).$$

Le même calcul que ci-dessus donne :

$$d_i^l < 1+b^{-e}+b^{-2e}.$$

Majorons la déviation globale :

$$d = \prod_{i=1}^{l} d_i.$$

donc

$$d < 1+b^{-e}+b^{-2e}$$

y a donc e chiffres exacts.

Nous avons donc démontré le :

THÉORÈME 3 : *Pour obtenir e chiffres exacts, il suffit d'effectuer, dans l'organigramme* n° 2, *l'instruction* (1) *avec* $c_i \geqq e+1+\log_b(2\,l)$ *pour tout* $i \in I_1$ *et*

l'instruction (2) *avec* $c_i \geqq e+1+\log_b (2\, l \bigstar p_i)$ *pour tout* $i \in I_2$ (*l est le nombre de multiplications*).

IV. EXEMPLE NUMÉRIQUE ET DISCUSSION

Soient

$$b = 10, \qquad x = 1.00009999, \qquad n = 2095000.$$

On applique à cet exemple :

a) l'algorithme dichotomique avec mantisse de longueur fixe; le nombre c de chiffres multipliés est constant; le nombre de chiffres exacts désirés est $e = 1$. Le théorème 2 donne $c = 9$;

b) L'algorithme dichotomique avec mantisses de longueur variable; le nombre de chiffres multipliés est variable. Le nombre de chiffres exacts désirés pour le résultat est :

1° $e = 1$, le théorème 3 donne $c_i \in [4, 10]$,

2° $e = 3$, le théorème 3 donne $c_i \in [6, 12]$.

Le tableau I récapitule les résultats.

NOTATIONS : e, nombre de chiffres exacts du résultat; c_i, nombre de chiffres pris en compte dans un nombre pour une multiplication; m, nombre total de multiplications élémentaires.

TABLEAU I

	e	c_i	m	$x^n = .93550791789\ldots E91$ y
a longueur fixe.....	1	9	2835	.917...E91
b_1 longueur variable.....	1	4 à 10	1413	.919...E91
b_2 longueur variable.....	3	6 à 12	2389	.93538...E91

REMARQUES : 1° on obtient juste le nombre de chiffres exacts désirés, ce qui confirme les résultats des théorèmes 2 et 3;

2° la perte de précision est importante, ce qui confirme l'insuffisance de la preuve des programmes lorsqu'elle néglige les erreurs dues aux nombres réels;

3° avec deux fois moins de multiplications élémentaires qu'en longueur fixe on obtient, en longueur variable, un résultat aussi bon. Avec moins de multiplications élémentaires qu'en longueur fixe, on obtient, en longueur variable, un résultat meilleur ($e = 3$).

Le tableau II donne le détail des c_i (cas où $e = 1$), dans le déroulement de l'algorithme proposé.

TABLEAU II

	Instruction (1)		Instruction (2)			
p_i	c_i	c_i^2	p_i	c_i	c_i^2	
			1047500	10	100	
			523750	10	100	
			261875	10	100	
261874	4	16	130937	9	81	
130936	4	16	65468	9	81	
			32734	9	81	
			16367	9	81	
16366	4	16	8183	8	64	
8182	4	16	4091	8	64	
4090	4	16	2045	8	64	
2044	4	16	1022	7	49	
			511	7	49	
510	4	16	255	7	49	
254	4	16	127	6	36	
126	4	16	63	6	36	
62	4	16	31	6	36	
30	4	16	15	6	36	
14	4	16	7	5	25	
6	4	16	3	5	25	
2	4	16	1	4	16	
0	4	16				
		$\sum_{i \in I_1} c_i^2 = 240$			$\sum_{i \in I_2} c_i^2 = 1173$	$\sum_{i=1}^{l} c_i^2 = 1413$

Discussion sur l'optimalité de l'algorithme proposé (th. 3)

Rappelons que nous considérons comme optimal l'algorithme qui fournira y avec e chiffres exacts en nécessitant un minimum de multiplications élémentaires. L'exemple numérique vient de montrer clairement la supériorité de la deuxième méthode (mantisses de longueur variable, déviations successives « uniformes ») sur la première méthode (mantisses de longueur uniforme, déviations successives variables).

La deuxième méthode n'est pas pour autant optimale. En effet, plusieurs améliorations sont possibles :

1° arithmétique d'arrondi;

2° méthode non dichotomique;

3° recherche d'une famille $\{ c_i \}$ plus performante (déviations non uniformes);

4° après chaque multiplication, possibilité de réajuster l'erreur maximale encore autorisée (les c_i sont discrets, la déviation atteint rarement son majorant).

Etc.

Par exemple, en restant dans le cadre de la deuxième méthode, examinons le point 3° : La famille c_i optimale de ce point de vue est donnée par les relations

$$\begin{cases} \prod_{i \in I_1} (1+b^{1-c_i})^2 \bigstar \prod_{i \in I_2} (1+b^{1-c_i})^{2p_i} \leqq 1+b^{-e}+b^{-2e} \text{ [cf. 1 (d)]}, \\ \sum_{i=1}^{l} c_i^2 \text{ minimale.} \end{cases}$$

De telles améliorations (à part le passage, évident, à une arithmétique d'arrondi) sont très complexes. Dans les cas 3° et 4° le calcul des c_i effectivement optimaux, si on l'incorporait à l'algorithme proprement dit, compromettrait définitivement l'optimalité du temps de calcul. Le théorème 3 que nous avons proposé semble donc un compromis simple et efficace.

CONCLUSION

La méthode que nous proposons nécessite, pour une précision équivalente du résultat, moins de multiplications élémentaires qu'une programmation traditionnelle dans une arithmétique classique. De plus, le résultat fourni possède au moins autant de chiffres exacts qu'on en a demandé.

La notion d'algorithme optimal se trouve élargie, grâce aux instruments que sont les assertions prouvant le programme, l'arithmétique à longueur variable, le critère des déviations successives uniformes. De nombreux algorithmes numériques sont susceptibles d'être étudiés sous cet angle.

Le cas échéant, l'application simultanée de la méthode « permutation-perturbation », de La Porte [5] et Vignes [8], après adaptation à cette arithmétique, pourra fournir une estimation moyenne de la déviation globale, inférieure au majorant préalablement fixé.

BIBLIOGRAPHIE

1. J. ARSAC, *Nouvelles leçons de programmation*, Dunod, Paris, 1977.
2. R. FLOYD, *Assigning Meanings to Programs*. Proceedings of a Symposium in Applied Mathematics, vol. 19, Mathematical Aspects of Computer Science, Providence Rhode Island, American Mathematical Society, 1967.
3. R. L. GRAHAM, A. C. C. YAO et F. F. YAO, *Addition Chains with Multiplicative Cost*, Stanford University, 1976.
4. C. A. R. HOARE, *An Axiomatic Basis for Computer Programming*, Comm. A.C.M., vol. 12, 1969, p. 576-583.

5. D. E. KNUTH, *The Art of Computer Programming*, vol. 2, 1969, p. 398-422, Addison Wesley.
6. M. LA PORTE et J. VIGNES, *Algorithmes numériques, analyse et mise en œuvre*, Technip, 1974.
7. H. LEROY, *La fiabilité des programmes*, Presses universitaires de Namur, 1975.
8. D. P. MACCARTHY, *The Optimal Algorithm to Evaluate x^n Using Elementary Multiplication Methods*, Math. comp., vol. 31, janvier 1977, p. 251-256.
9. J. VIGNES et M. LA PORTE, *Error Analysis in Computing*. Proceedings of I.F.I.P. Congress, Stockholm, 1974, p. 610-614.
10. N. WIRTH, *Systematic Programming. An introduction*, Prentice Hall, 1973.