Introduction

Nous avons tous appris, à l'école primaire, à effectuer des additions, des multiplications et des divisions. Pour cette raison, la plupart des utilisateurs d'ordinateurs ne se sont jamais demandé comment leurs machines effectuaient les opérations arithmétiques, pensant que les mêmes méthodes (ou tout au moins de légères variantes, adaptées par exemple à l'emploi de la base 2) étaient utilisées, ce qui est nous le verrons souvent faux.

En ce qui concerne l'évaluation de fonctions plus complexes (sinus, cosinus, logarithme, exponentielle...), nombre d'entre nous se sont demandés, au lycée, quelles étaient les méthodes utilisées par nos calculatrices de poche. Cette interrogation a en général pris fin lorsque nous avons appris ce qu'est un développement de Taylor. Nous avons alors cru savoir.

Un des buts de cet ouvrage est de montrer que les algorithmes utilisés en arithmétique des ordinateurs sont souvent plus complexes qu'on ne croit, et qu'ils sont en général très différents de ceux que l'on utilise pour faire des calculs « à la main ».

Un autre point important concerne la fiabilité des calculs effectués sur ordinateur. Tout d'abord parce que les circuits et programmes arithmétiques peuvent, comme les autres, comporter des erreurs. Ensuite parce que même avec des circuits et programmes arithmétiques corrects, le résultat d'une opération arithmétique n'est pas forcément exactement représentable en machine : il faut l'arrondir au « nombre machine » le plus proche. On commet alors une toute petite *erreur d'arrondi*. En général, l'influence de ces erreurs d'arrondi sur le résultat final d'un calcul est faible, mais ce n'est pas toujours le cas. Donnons tout d'abord deux exemples d'erreur de conception dans des circuits ou programmes arithmétiques :

– le diviseur de la première version du processeur Pentium d'Intel donnait un résultat faux dans environ un cas sur 4×10^{10} (en simple précision). Par exemple, le calcul de

$$8391667/12582905$$

donnait $0.666869\cdots$ au lieu de $0.666910\cdots$. Il est d'ailleurs amusant de constater que l'erreur réside dans l'algorithme lui-même, et non dans son implantation [Mul95a] ;

– dans la version 7.0 du système de calcul formel Maple, si l'on calcule

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

on obtient 1 au lieu de 5001. Dans la version précédente (6.0) du même système, si on entrait :

$$214748364804136478196437 94$$

la « quantité » affichée et mémorisée était `413647819643790)+'- .( -..(`

Il est très facile[1] de construire des exemples numériques pour lesquels l'accumulation des erreurs d'arrondi finit par conduire à un résultat inacceptable. Considérons l'exemple suivant, construit par l'un d'entre nous [Mul95b] : en partant d'une valeur initiale

$$u_0 = e - 1 = 1.7182818284590452353\,602874\,7\cdots$$

où e est la base des logarithmes naturels, on construit la suite u_n définie par

$$u_n = nu_{n-1} - 1.$$

Le problème est de calculer, par exemple, u_{25}. C'est *a priori* facile : on n'effectue que des multiplications par de tous petits entiers, et des soustractions du nombre 1. Pourtant, selon le système utilisé, on obtiendra des résultats radicalement différents. Le tableau 1 donne quelques exemples obtenus sur une arithmétique de base 10, en faisant varier le nombre de décimales utilisées lors des calculs. On voit que sur cet exemple, il faut une très grande précision des calculs intermédiaires pour obtenir un résultat final correct.

Le présent ouvrage a été conçu pour donner au lecteur une idée de ce qu'est l'arithmétique des ordinateurs ainsi qu'un inventaire de propriétés et de solutions dans lequel il pourra venir piocher pour résoudre certains problèmes qu'il peut rencontrer.