Projet n 1
Méthodes de calcul numérique/Limites de la machine

Groupe n 1 - Equipe n 1

Responsable : ESTUPINA
Secrétaire : BENKIRANE
Codeurs : EL MOUMNI, PATRY, RIVERO

Résumé : Le but de ce projet consiste à évaluer les problèmes qui peuvent apparaître lors de l'utilisation d'opérations élémentaires, voire d'algorithmes plus poussés, sur des nombres flottants. la première partie s'intéresse à trouver des exemples dans lesquels les opérations élémentaires sont insuffisamment précises. la seconde partie donne des exemples d'algorithmes utilisés dans des conditions de calcul en basse précision.

1 Représentation des nombres en machine

1.1 Représentation décimale réduite

La représentation décimale réduite consiste à représenter les nombres en machine avec une précision de p décimales. Le terme "décimales" désigne ici le nombre de chiffres significatifs et pas le nombre de chiffres après la virgule. Afin de simuler cette représentation, nous allons nous appuyer sur les calculs effectués par python qui seront supposés réels. Nous appellerons r_p la fonction qui étant donné un réel x et un entier p retourne la représentation décimale réduite de x sur p décimales.
Soit x un réel et p un entier, le calcul de $r_p(x, y)$ se fait en quatre étapes : tout d'abord il faut déplacer la virgule de x de façon à avoir p décimales avant la virgule, soit x' le nouveau nombre obtenu et n le nombre de déplacements. Ensuite, on calcul $y = x' - z$, où $z = E(x')$. y représente forcément un nombre entre 0 et 1. Ainsi si $y >= 0.5$ on ajoute 1 à z. Finalement, on déplace n fois la virgule de z vers la gauche et on retourne le nombre obtenu. Pour la mise en œuvre de cet algorithme, nous avons utilisé les fonctions partie entière et log_{10} prédéfinies dans python.

Notons que pour avoir une idée sur les imprécisions des calculs machines, il est nécessaires de simuler les opérations usuelles que sont l'addition et la multiplication en représentation décimale réduite. Pour cela, il suffit d'appeler la fonction r_p sur la somme ou le produit de deux nombres.

1.2 Erreur relative

Dans cette partie nous allons mesurer les erreurs de calculs effectués par la machine. Plus précisément nous exploiterons deux fonctions calculant l'erreur relative obtenue sur la somme et le produit. Ces deux fonctions sont définies comme suit :

$$\delta_s(x, y) = \frac{|(x+y) - r_p(x+y)|}{r_p(x+y)}$$
$$\delta_p(x, y) = \frac{|(x*y) - r_p(x*y)|}{r_p(x*y)}$$

$(x + y)$ et $(x * y)$ font référence aux opérations réelles effectuées par python. r_p prend un second paramètre qui est la précision que nous supposerons fixe.
Notre but était de trouver les valeurs de x et y maximisant les erreurs. Pour cela nous avons fixé x, puis nous avons tracé les fonctions des erreurs relatives en fonction de y. Les résultats obtenus avec une précision sur 3 chiffres sont représentés dans les figures présentées sur les dernières pages.

Dans la figure 1 on voit que la perte d'une décimale engendre une erreur sur la somme maximale $\epsilon = 5 * 10^{-3}$. Dans la figure 2, la somme de deux quantités équivalentes opposées augmente l'erreur : ce sont les pics remarqués au niveau des $y = -3, -2, -1$. L'erreur sur le produit quant à elle augmente si on perd une décimale (cf figure 3) ou si le produit se rapproche de 0 .

1.3 Exemple de calcul : $log(2)$

Dans cette partie nous allons nous intéresser au calcul de $log(2)$ avec une précision sur p décimales grâce à la formule :

$$ln(2) = \sum_{i=1}^{+\infty} \frac{(-1)^{i+1}}{i}$$

Pour cela nous avons implémenté deux algorithmes : le premier effectue un calcul classique de $log(2)$ en faisant alternativement la somme de nombres positifs et négatifs. Le second fait la somme des nombres positifs et négatifs indépendamment. Le temps de calcul est très long pour les deux algorithmes (plus de $20s$ à partir d'une précision de 4 décimales). Cependant pour le deuxième algorithme l'erreur cumulée est améliorée car on évite la somme de quantités équivalentes opposées. Ci-dessous se trouve le tableau comparant les deux erreurs et les deux algorithmes de calcul de $log(2)$.

Nb décimales	log2_1	log2_2
1	13,689113324482935	3,6545676014904322
2	4,949446294246405	2,314315072337656
3	13 ;345624437472104	1,592946875856434
4	10,614248843386585	8,2933003000029668

2 Algorithmes CORDIC

2.1 Représentation des nombres en calculatrice

Dans une calculatrice un nombre flottant est représenté sur 8 octets par :
- m : une mantisse constituée de 13 chiffres codés indépendamment sur 4 bits, soit 52 bits au total ;
- e : un exposant (en puissance de 10) éventuellement signé sur 11 bits ;
- s : le signe du nombre (et de l'exposant s'il n'est pas signé) sur 1 bits.

Cette représentation, qui est la virgule flottante, nous permet de gérer un intervalle de nombres réels plus important que la représentation à virgule fixe. En revanche, le format à virgule flottante occupe un peu plus de place, car il est nécessaire d'encoder la position de la virgule.
De plus cette représentation ne nous permet pas de gérer tous les nombres réels, ce qui nous pousse à faire des approximations qui peuvent générer des erreurs importantes dans les calculs.

2.2 Calcul des fonctions trigonométriques et exponentielles

Sur calculatrice on tente d'éviter les opérations 'lourdes' que sont la multiplication et la division. L'évaluation des fonctions ln, exp, tan, et arctan se fait selon les algorithmes CORDIC dont le principe est d'effectuer une série de transformations simples (addition/soustraction et décalage) réduisant la valeur de x à une valeur très faible en même temps qu'élaborant le résultat. Chacune de ces transformations nécessite une valeur précalculée de ces fonctions. Le résultat est obtenu par une simple interpolation linéaire. Pour les 4 fonctions précédentes les tableaux valeurs suivants suffiront pour obtenir plus de 12 chiffres de précision :

$$\begin{cases} \ln(2), \ln(1.1), \ln(1.01), \dots \ln(1.000001) \\ \arctan(1), \arctan(0.1), \arctan(0.01), \dots \arctan(0.0001) \end{cases}$$

On procède en deux étapes :
Ramener la valeur à évaluer dans l'intervalle où la méthode est applicable grâce aux transformations :

$$\begin{cases} \ln(a \times b) = \ln(a) + \ln(b) \\ \exp(a + b) = \exp(a) \times \exp(b) \\ \arctan(\ \frac{1}{a}\ ) + \arctan(a) = \frac{\pi}{2} \\ \tan(a + b) = \frac{tan(a)+tan(b)}{1-tan(a)\times tan(b)} \end{cases}$$

Appliquer l'algorithme en itérant sur l'indice k de l'élément précalculé .L'erreur commise dans chaque cas est inférieure à 10^{-12}. Par exemple pour ln : on représente x par $x = (1+1)^{n0} * (1+1/10)^{n1} * ... * (1+1/10^6)^{n6} * (1+eps)$. Ainsi le calcul de $ln(x)$ se réduit au calcul simple suivant : $ln(x) = n0 * ln(1+1) + n1 * ln(1+1/10) + ... + ln(1+eps)$.
Cette méthode est particulièrement adaptée aux calculatrices parce qu'elle fait appel à la fois à des calculs simples réduisant les erreurs et à des ressources mémoires faibles. Le tableau ci-dessous donne quelques exemples de calculs avec des fonctions précédentes.

	1.347	10.427	13.23
ln	0.297880	2.344399	2.582487
exp	3.845871	33758.921429	556821.499575
tan	4.393500	1.565046	0.781937
arctan	0.932183	1.475184	1.495354

2.3 Problèmes et solutions

Dans cette partie nous tenterons d'élucider trois problèmes liés à l'évaluation de fonctions usuelles en machines. Nous présenterons en même temps les solutions pour y remédier.
L'une des méthodes d'évaluation des fonctions usuelles est la formule de Taylor-Lagrange :

.

$$f(x) = \sum_{i=0}^{+\infty} f^{(i)}(x_0) \frac{(x-x_0)}{i!}$$

Ne pouvant effectuer de somme infinie, quand s'arrêter ? En pratique, on somme les termes un a un jusqu'a ce que le terme ajoutée soit plus petit qu'un petit fois l'ordre de grandeur total de la somme.

Il faut de plus que la convergence soit suffisamment rapide. La solution est donc d'accéeléerer cette vitesse. Suivant la nature de la série, il existe différentes méthodes.
Si on dispose d'une serie presque geometrique, on peut extrapoler les sommes partielles successives avec une formule simple :

$$S'_n = S_{n+1} - \frac{(S_{n+1}-S_n)^2}{S_{n+1}-2S_n+S_{n-1}}$$

Si la série est alternée, on peut appliquer la transformation d'Euler mais on aboutit à un algorithme complexe. Toutefois, il existe un algorithme simple selon Wyngaariden qui ne nécessite qu'un vecteur qui stocke les différences partielles successives.

$$\sum_{i=0}^{+\infty}(-1)^i u_i = u_0 + u_1 + \ldots + u_{n-1} + \sum_{i=0}^{+\infty}\frac{(-1)^i}{2^{i+1}}[\Delta^i u_i] \text{ avec}$$

$$\begin{cases} \Delta^0 u_k = u_n \\ \Delta^1 u_n = u_{n+1} - u_n \\ \Delta^2 u_n = u_{n+2} - 2u_{n+1} + u_n \\ \Delta^3 u_n = u_{n+3} - 3u_{n+2} + 3u_{n+1} - u_n \\ \ldots \end{cases}$$

Il existe certaines séries qu'on peut transformer en série alternée par la relation suivante :

$$\sum_{n=1}^{+\infty} u_n = \sum_{n=1}^{+\infty}(-1)^{n-1} w_n \text{ avec } w_n = u_n + 2u_{2n} + 4u_{4n} + 8u_{8n} + \ldots$$

Cependant, il demeure des fonctions qui ne satisfont pas aux critères précédents, telle que la fonction sinus :

$$sin(x) = \sum_{k=0}^{+\infty}\frac{(-1)^k}{(2k+1)!}\ x^{2k+1}$$

Un autre problème récurent est le problème d'overfow qui survient lors de la manipulation de nombres complexes.Par exemple pour évaluer le module d'un nombre complexe on calcule :

$$|a+ib| = \begin{cases} |a|\sqrt{1+(\frac{b}{a})^2} & \text{si } |b| \leq |a| \\ |b|\sqrt{1+(\frac{a}{b})^2} & \text{si } |a| < |b| \end{cases}$$

De même, dans le cas du quotient de deux nombres complexes, on calcule la quantité suivante :

$$\frac{a+ib}{c+ib} = \begin{cases} \frac{(a+b\frac{d}{c})+i(b-a\frac{d}{c})}{c+d\frac{d}{c}} & \text{si } |d| \leq |c| \\ \frac{(b+a\frac{c}{d})+i(-a+b\frac{c}{d})}{d+c\frac{c}{d}} & \text{si } |c| < |d| \end{cases}$$

Finalement, l'évaluation de la dérivée d'une fonction peut aboutir à une imprécision assez importante lorsqu'on utilise la formule du taux d'accroissement :

$$f'(x) = \frac{f(x+h)-f(x)}{h}$$

En effet, les termes de plus haut degré posent problème dans le développement en série de Taylor , ce qui peut aboutir à une erreur de troncature. De plus, des arrondis peuvent apparaitre à cause d'origines diverses.

FIGURE 1 – Erreur relative sur la somme : perte d'une décimale

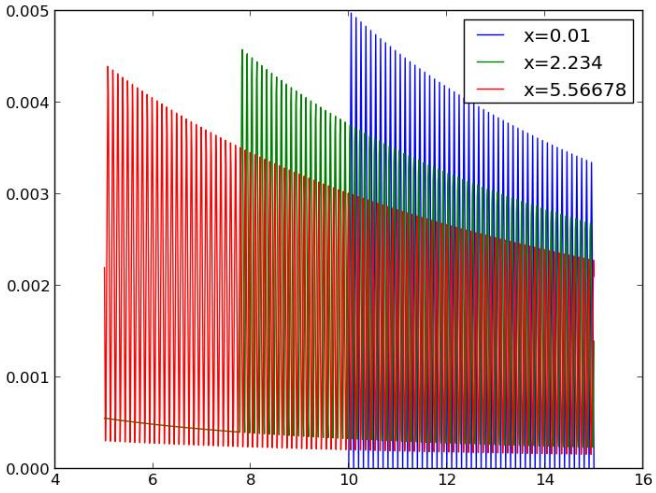

FIGURE 2 – Erreur relative sur la somme : somme de quantités équivalentes opposées

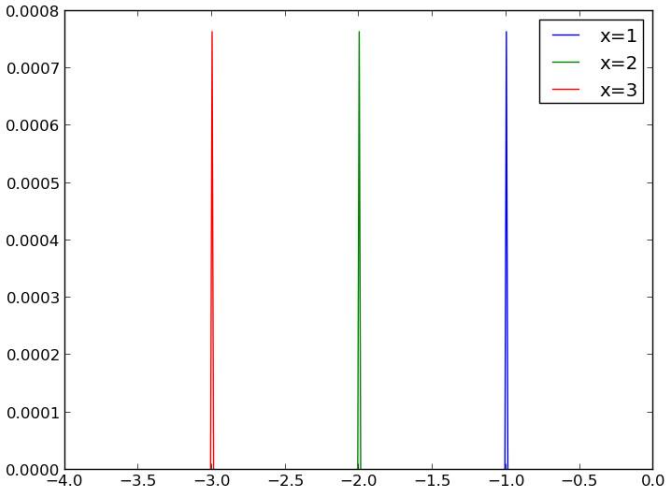

FIGURE 3 – Erreur relative sur le produit : perte d'une décimale