Projet 1
Méthodes de calcul numérique
Limites de la machine

Groupe 3 - Equipe 2

Responsable : Imad BOUDROUA
Secrétaire : Mohamed Faycal BOULLIT
Codeurs : Theo LELASSEUX,
Deborah PEREIRA,
Mohammed BOUDALI

Résumé : Ce projet a pour but d'évaluer les problèmes qui peuvent apparaître lors de l'utilisation d'opérations élémentaires et d'algorithmes sur des flottants. Il traite dans un premier temps de la précision insuffisante des opérations élémentaires dans des cas particuliers, et dans un second temps, d'algorithmes utilisés dans des conditions de calcul en basse précision, comme l'algorithme de CORDIC utilisé dans les calculatrices.

1 Représentation des nombres en machine

Cette partie consiste à aborder certains exemples et appliquer les notions vues en cours pour pouvoir déterminer les différents types d'erreur. En effet, chaque flottant sur la machine est approximé, et après chaque opération élémentaire, on a une erreur induite de calcul, ce qui est mis en évidence dans cette partie à travers l'implémentation de la fonction **rp(x,p)**.

1.1 Représentation décimale réduite rp(x,p)

Pour l'implémentation de l'algorithme de calcul de la représentation décimale réduite, nous avons utilisé la fonction **round(n,p)** qui arrondi avec un nombre flottant n à p décimales. Pour "tronquer" `x` à l'aide de `round(x,m)`, il faut donc connaître l'exposant k en notation scientifique de `x`. L'algorithme traite deux cas :

— $x < -1$ ou $x > 1$: diviser par 10 une variable temporaire t tant que $x*t >= 10$ ou $x*t <= -10$ en incrémentant le compteur k.

— $-1 \leq x \leq 1$: Tant que $-1 \leq x*t \leq 1$ t est multipliée par 10 à chaque fois en incrémentant k.

Dans les deux cas on cherche donc à représenter x par sa notation scientifique : $x = m * 10^k$ afin de pouvoir appliquer round sur m et obtenir ainsi la représentation en décimale réduite de x.

1.2 Opérations usuelles

Dans cette section nous allons effectuer les opérations usuelles addition et multiplication en représentation décimale réduite.

1.2.1 Addition et multiplication

L'algorithme adopté dans ces opérations est le suivant :

1. Les représentations décimales réduites de x et y sont récupérées.
2. L'addition ou la multiplication est effectuée.
3. La représentation décimale réduite du résultat est récupérée.

1.2.2 Erreurs relatives

On calcule l'erreur relative de l'addition et la multiplication sur x et y en représentation décimale réduite d'après la formule donnée dans le sujet. On obtient alors le graphique suivant :

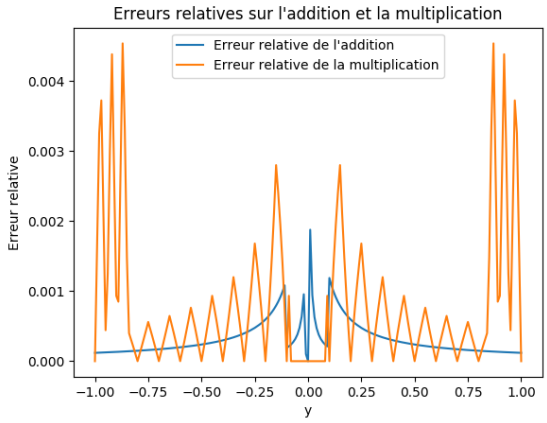

FIGURE 1 – Erreur relative sur l'addition et la multiplication de x et y avec x=0.000119

1.2.3 Calcul de log(2)

Pour calculer log(2) on calcule la somme de de $\frac{(-1)^{i+1}}{i}$ avec i allant de 1 à 100. Pour cela on utilise une boucle qui à chaque tour ajoute à notre somme s le terme $\frac{(-1)^{i+1}}{i}$. Une fois que i=100, on calcule la représentation décimale réduite à p décimales de cette valeur et on calcule l'erreur relative avec la valeur calculée par la fonction log de numpy. On constate que quelque soit p, l'erreur relative est nulle. L'algorithme que nous avons trouvé a donc la même précision que celui de numpy.

2 Algorithmes CORDIC

Dans cette partie, nous allons nous intéressé à l'implémentation de quelques fonctions en utilisant les algorithmes **CORDIC** utilisés dans des conditions où on ne possède pas de ressources importantes(mémoire, puissance de calcul) pour effectuer des calculs. C'est le cas pour les calculatrices de poche.

2.1 Représentation des nombres sur une calculatrice

Dans une calculatrice typique un nombre 'flottant' occupe 8 octets de mémoire et se décompose en :

- une mantisse constituée de 13 chiffres codés indépendamment sur 4 chiffres binaires (on parle de Binaire Code Decimal ou BCD)
- un exposant (puissance de 10) éventuellement signé

- le signe du nombre (et de l'exposant s'il n'est pas signé)

Cette représentation permet de représenter un très large intervalle de nombres réelles. Cependant, cette représentation occupe un peu plus de place, car il est nécessaire de coder la position de la virgule. De plus, pour de grand nombre on perd en précision, les chiffres de poids faible n'étant alors plus représentés.

2.2 Fonctions trigonométriques et exponentielles

Le principe des algorithmes **CORDIC** 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'en élaborant le résultat. Chacune de ces transformations nécessite une valeur pré-calculée de f (ou de son inverse). Le résultat est obtenu par une simple interpolation linéaire (ou un développement en série si davantage de chiffres sont requis).

Tout d'abord, il est nécessaire de ramener la valeur à calculer ,soit x, dans un intervalle où on peut appliquer l'algorithme, tout en effectuant des transformations simples (addition/soustraction et décalage)

Pour l'exponentielle par exemple, il faut ramener la valeur à calculer x dans l'intervalle $[0, ln(10)[$ en trouvant un entier n réalisant : $exp(x) = exp(x - n * ln(10)) * 10^n$

Et donc il suffit de calculer $exp(x - n * ln(10))$ par l'algorithme de **CORDIC** puis multiplier par 10^n

Cette technique est efficace lorsqu'elle est ramenée à une calculatrice pour les raisons suivantes :

- L'utilisation exclusive des deux opérations : addition et soustraction (opérations de faible coût), ce qui permet de minimiser le temps de calcul.

- L'utilisation de valeurs pré-calculées : en effet, la recherche d'une valeur dans un tableau prend moins de temps que de réaliser le calcul, généralement, lourd de cette valeur.

Pour tester les fonctions implémenter on a tracé les graphes dans la figure **??** qui correspondent à l'erreur relative entre les valeurs calculées par l'algorithme **CORDIC** et ceux des fonctions de la bibliothèque MATH en python. Les graphiques correspondent successivement à l'erreur relative de la fonction logarithme, exponentielle, arctangente et tangente.

On remarque que pour la fonction logarithme népérien pour $x \in [0, 20]$ l'erreur relative maximale est de $5 * 10^{-9}\%$. Pour la fonction exponentielle pour $x \in [-40, 40]$ l'erreur relative maximale est de 0,001%. Pour la fonction arctangente pour $x \in [-20, 20]$ l'erreur relative maximale est de 0,008%. Pour ces trois fonctions, aucun comportement particulier de l'erreur relative n'est notable. Pour la fonction tangente pour $x \in [-1.5, 1.5]$ l'erreur relative maximale est de $4,6 * 10^{-10}\%$. On constate que plus x est proche de la valeur $\frac{\pi}{2}$ ou $-\frac{\pi}{2}$, plus l'erreur relative est importante. Cela est dû au fait qu'on cherche à calculer une valeur très grande, la fonction tangente tendant vers l'infini.

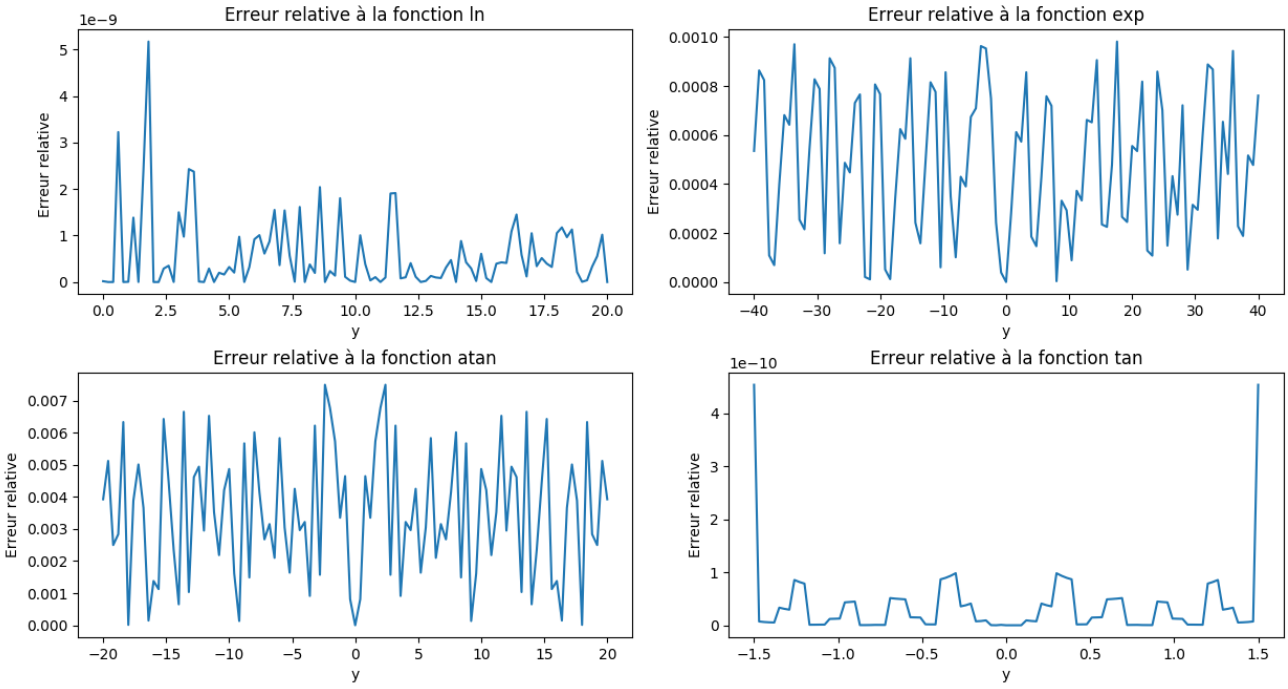

FIGURE 2 – Erreurs relatives pour les fonctions implémentées par les algorithmes CORDIC

2.3 Problèmes soulevés dans *Numerical Recipes in C*

2.3.1 Arithmétique sur les complexes

Afin d'évaluer une multiplication entre deux complexes, il est possible d'utiliser l'expression suivante :

$$(a+ib)(c+id) = (ac-bd) + i(bc+ad) \tag{1}$$

On utilise alors 4 multiplications, 1 addition et 1 soustraction. Il est possible d'utiliser une autre expression :

$$(a+ib)(c+id) = (ac-bd) + i[(a+b)(c+d) - ac - bd] \tag{2}$$

On utilise cette fois 3 multiplications, 2 additions et 3 soustractions. On utilise plus d'opérations au total, mais 1 multiplication de moins ce qui est rentable au vu du coût d'une multiplication par rapport à une addition ou soustraction.

Dans le cas du calcul du module d'un complexe, il est fortement déconseillé d'utiliser l'expression classique du module :

$$|a+ib| = \sqrt{a^2+b^2} \tag{3}$$

En effet, si a ou b est aussi grand que la racine carrée du plus grand nombre représentable, un

overflow a lieu. Il est donc préférable d'utiliser l'expression suivante :

$$|a+ib| = |a| * \sqrt{1+(\frac{b}{a})^2} si |a| >= |b| \tag{4}$$

$$|a+ib| = |b| * \sqrt{1+(\frac{a}{b})^2} si |a| < |b| \tag{5}$$

2.3.2 Fractions continues

Les fractions continues sont un outil puissant pour évaluer des fonctions. Une fraction continue est une expression de la forme :

$$b_0 + \cfrac{a_1}{b_1 + \cfrac{a_2}{b_2 + \cfrac{a_3}{b_3 + \cfrac{a_4}{\ddots}}}} \tag{6}$$

comportant un nombre fini ou infini d'étages.

Le problème posé par les fractions continues est le suivant : **comment savoir jusqu'où aller lorsqu'on évalue une fraction continue ?**

Contrairement à une série, on ne peut pas simplement évaluer l'équation de gauche à droite jusqu'à ce que le terme suivant à ajouter n'apporte qu'un faible changement. Écrite sous la forme de (**??**), la seule façon d'évaluer la fraction continue est de droite à gauche, ce qui nous obligerait à choisir aveuglément un point de départ. Pour éviter cela, il faut utiliser un résultat qui relie les fractions continues aux approximations rationnelles, et qui donne un moyen d'évaluer notre fraction continue de gauche à droite. Notons f_n le résultat de l'évaluation de (**??**) au rang n :

$$f_n = \frac{A_n}{B_n} \tag{7}$$

Avec :

$$A_{-1} = 1, \quad A_0 = b_0, \quad et \quad A_j = b_j A_{j-1} + a_j A_{j-2} \quad j = 1, 2, .., n \tag{8}$$

$$B_{-1} = 0, \quad B_0 = 1, \quad et \quad B_j = b_j B_{j-1} + a_j B_{j-2} \quad j = 1, 2, .., n \tag{9}$$

Notre fraction continue, ainsi écrite sous la forme (**??**) peut être évaluée de gauche à droite comme pour une série.

2.3.3 Evaluation de polynômes

Un polynôme de degré N peut être représenté par un tableau de coefficients tel que c[j] est le coefficient de x^j.

Pour évaluer un polynôme de degré 5 on peut simplement calculer :

$$P(x) = c[0] + c[1] * x + c[2] * x * x + c[3] * x * x * x + c[4] * x * x * x * x + c[5] * x * x * x * x * x \tag{10}$$

Cependant cette méthode d'évaluation n'est pas du tout optimale. En effet ici on a 15 multiplications et 5 additions. On préférera donc faire :

$$P(x) = c[0] + x * (c[1] + x * (c[2] + x * (c[3] + x * (c[4] + x * c[5])))) \tag{11}$$

Ce qui nous permet de réduire le nombre de multiplication à 5, avec toujours 5 additions. Cette astuce qui n'est rien d'autre que de la factorisation nous permet d'évaluer un polynôme de degré N en N multiplications.