Méthodes de calcul numérique, limites de la machine

Algorithmique numérique

Dao Emilie (coord)
Parpaite Thibault (secr)
Drault Fabien (prog)
Damiani Thomas (prog)
Seignette Thomas (prog)

Projet n°1
ENSEIRB-MATMECA
M. Renault David
Semestre 6, G4 - E6
8 février 2017

Présentation du travail

Résumé

L'objectif de ce projet consiste à identifier et évaluer les problèmes pouvant apparaître lors de l'utilisation de nombres flottants en machine.

Dans un premier temps, nous allons introduire une représentation machine [1] sur les flottants à l'aide de Python pour mettre en évidence la perte de précision sur les opérations élémentaires (addition et multiplication).

La seconde partie se concentrera sur des algorithmes plus complexes et la manière dont on peut les raffiner afin de garantir leur stabilité.

Répartition des tâches

Nous avons travaillé en utilisant la méthode de *pair programming*. Chaque tandem a eu pour rôle de se concentrer sur une partie spécifique (avec un pilote et un copilote), et Emilie s'est chargée de coordonner le déroulement du projet (création d'un dépot github, suivi de l'avancement chaque semaine).

Le rapport a été rédigé à l'aide de la plateforme collaborative sharelatex.

Dépot github : https ://github.com/emiliedao/algonum-project1

Conclusion, axes d'améliorations

Le rapport aurait pu être plus synthétisé afin de tenir sur cinq pages. Ce sont notamment les figures qui ont pris de la place [2].

1. Représentation Décimale Réduite
2. Compromis taille/lisibilité

1 Représentation des nombres en machine

Dans cette partie, nous allons exhiber des exemples pour lesquels les opérations élémentaires telles que l'addition et la multiplication sont insuffisamment précises.

1.1 Représentation décimale réduite

Dans un premier temps, nous avons implémenté une fonction `rp` 1 permettant de simuler la représentation machine des nombres manipulés. Cette représentation que nous appellerons **représentation décimale réduite** consiste à considérer l'arrondi à p décimales significatives (au sens IEEE 754) [1].

Algorithm 1: `rp(x, p)`

Param : x : le flottant à convertir, p (entier) : le nombre de décimales significatives
Retour: la représentation décimale réduite de x
Comp. : Temps : $\Theta(log_{10}(x)), Espace : \Theta(1)$

La subtilité de la fonction réside dans le fait qu'on veut afficher p décimales **significatives** et non pas p décimales après la virgule. Pour cela nous avons utilisé la fonction `round` de la bibliothèque standard Python dont voici le prototype :

Algorithm 2: `round(x, n)`

Param : x : le flottant à convertir, n : un entier
Retour: x arrondi à n chiffres à partir de la virgule

On constate que cette fonction round se contente d'arrondir à un certain nombre de chiffres k à partir de la virgule [2] . Le point crucial de notre algorithme va être de déterminer k de manière à ce que l'appel de la fonction `round` avec ce k corresponde à la représentation décimale réduite recherchée. On distingue trois cas :

— Si $n = 0$, il n'y a pas d'arrondi à effectuer.

Sinon, nous calculons val, valeur à retirer à k pour obtenir l'arrondi voulu. val est calculé à partir de la partie entière du logarithme en base 10 de $|n|$.

1. Le terme "décimales" désigné ici ne doit pas être confondu avec le nombre de chiffres après la virgule.
2. k peut être négatif

— Si $log_{10}(|n|) \geq 0$, nous choisissons $val = \log_{10}(|n|) + 1$, qui représente le nombre de chiffres significatifs avant la virgule.
— Sinon, nous choisissons $val = log_{10}(|n|)$, qui correspond à l'opposé du nombre de 0 non significatifs d'un flottant.

Dans tous les cas, nous obtenons la représentation décimale réduite de n à p décimales en retournant la valeur calculée par `round`$(n,\ p - val)$.

1.2 Stabilité des opérations élémentaires

Erreur relative

Par la suite, nous avons également implémenté les fonctions permettant de simuler les opérations usuelles en représentation décimale réduite (`addition_rp` et `multiplication_rp`).

Il est possible de comparer les résultats obtenus dans le "monde réel" avec la représentation machine, on parle d'**erreur relative** ou encore de **précision machine**. Nous avons donc implémenté deux fonctions (`delta_add` et `delta_mul`) qui permettent de calculer cette erreur relative définie par les deux formules ci-dessous.

$$\delta_s(x,y) = \frac{\left|(x \underset{\text{réel}}{+} y) - (x \underset{\text{machine}}{+} y)\right|}{\left|(x \underset{\text{machine}}{+} y)\right|}$$

(a) Erreur relative sur la somme

$$\delta_p(x,y) = \frac{\left|(x \underset{\text{réel}}{\times} y) - (x \underset{\text{machine}}{\times} y)\right|}{\left|(x \underset{\text{machine}}{\times} y)\right|}$$

(b) Erreur relative sur le produit

FIGURE 1.1 – Formules de calcul de l'erreur relative

Limites de la précision machine

Le graphique 1.2 est représentatif des erreurs relatives que l'on retrouve pour la plupart des flottants : c'est-à-dire une courbe avec de fortes variations de l'ordre de 10^{-16}.

En revanche le graphique 1.2 a été choisi spécifiquement pour maximiser l'erreur relative et montrer la limite de la précision machine. L'ordre de grandeur est 10^{-4}. La valeur de x n'a pas été choisie au hasard et elle correspond à $\frac{10^{-p+1}}{2}$ avec p correspondant à la représentation décimale réduite sur p décimales.

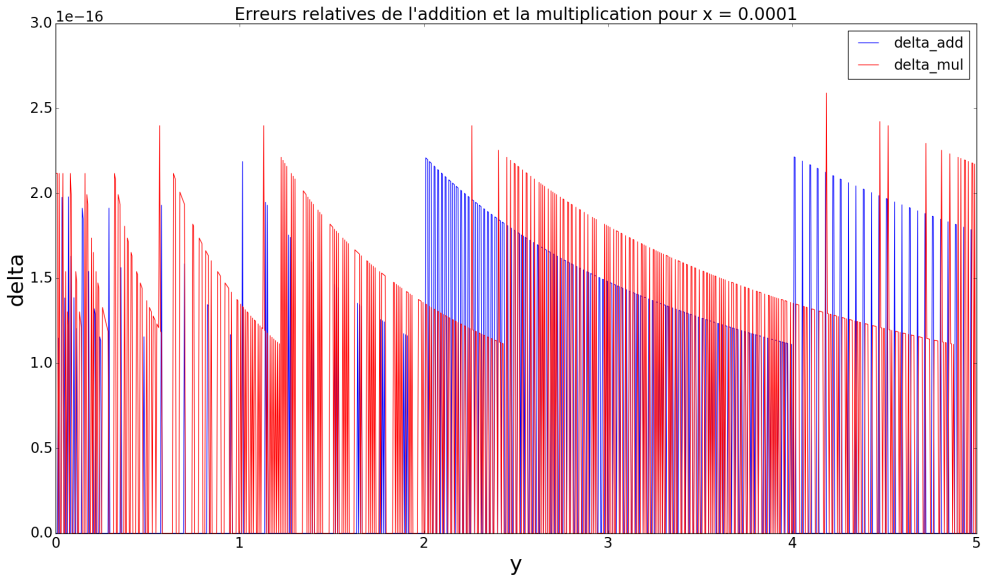


FIGURE 1.2 – Exemple d'erreur relative classique

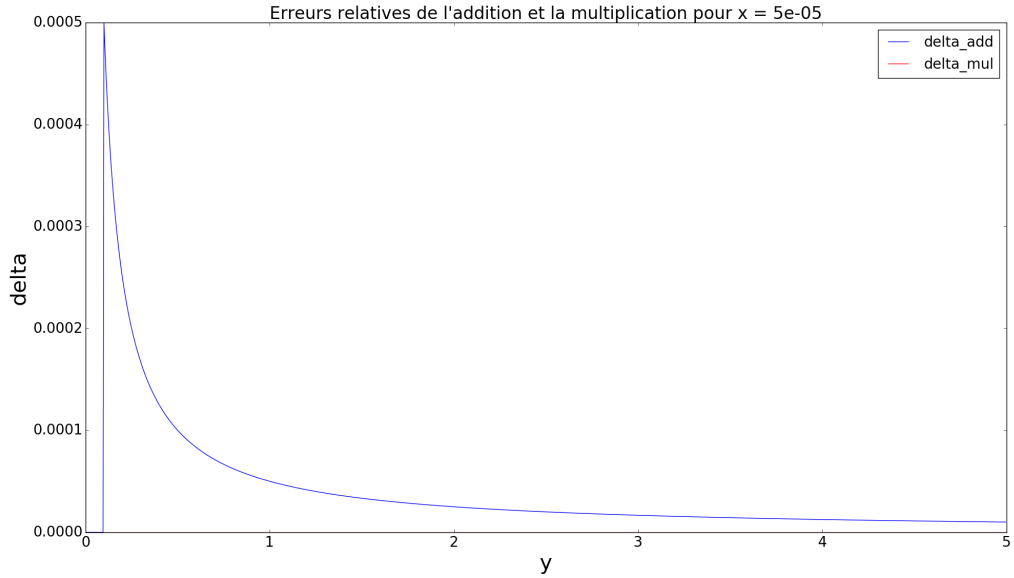


FIGURE 1.3 – Exemple d'erreur relative maximale

1.3 Algorithme simple : calcul de log(2)

$$\log(2) = \sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n}$$

FIGURE 1.4 – Formule de calcul de $log(2)$

La perte de précision de cet algorithme de calcul de $log(2)$ réside principalement dans la somme qui est réalisée à chaque itération. Nous avons donc implémenté cette fonction en prenant soin de calculer en même temps l'erreur relative qui s'accumule à chaque tour de boucle. Plus précisemment la fonction retourne un couple (res, erreur) [3].

Exemple de l'exécution d'un test :

```
log_2_v1(8) : (0.693197, 0.00036008655508652484)
log_2_v2(8) : (0.6930971, 0.0018818414665731384)
np.log(2) (reel) : 0.69314718056
Erreur relative de v1 par rapport au res final : 7.18690935689e-05
Erreur relative de v2 par rapport au res final : 7.22561960586e-05
```

Remarque : on constate que les résultats fournis par nos algorithmes utilisant la représentation décimale réduite sont plus précis en pratique (comparaison du résultat final par rapport au résultat réel) qu'en théorie (l'erreur relative "cumulée" calculée au cours de l'itération de la somme).

Conclusion : cela veut dire que de manière globale, les erreurs relatives engendrées par notre représentation décimale réduite se compensent lorsqu'on les somme. C'est d'ailleurs un comportement que l'on retrouve dans la plupart des implementations de flottant (ie. python en abuse pour retomber sur des résultats "justes").

3. Nous avons implémenté deux algorithmes équivalent, un additionnant à partir du premier terme de la somme et l'autre à partir du dernier

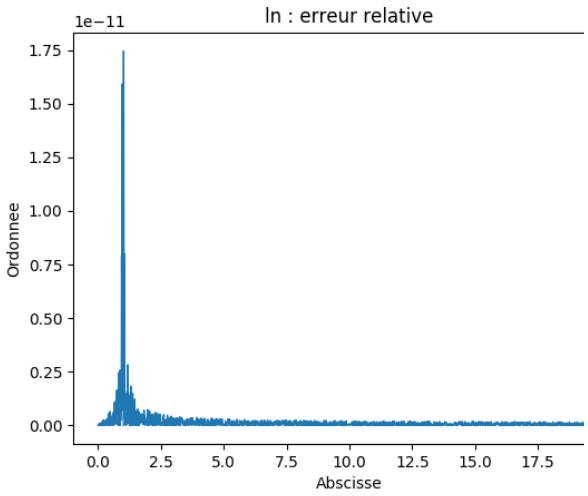
2 Algorithmes CORDIC

Dans cette partie, nous allons nous mettre dans le cas où l'on dispose de ressources limitées en mémoire et puissance de calcul pour effectuer des calculs. Les algorithmes utilisés devront donc s'adapter à ces contraintes. On rencontre ce genre de situation dans les calculatrices de poche par exemple.

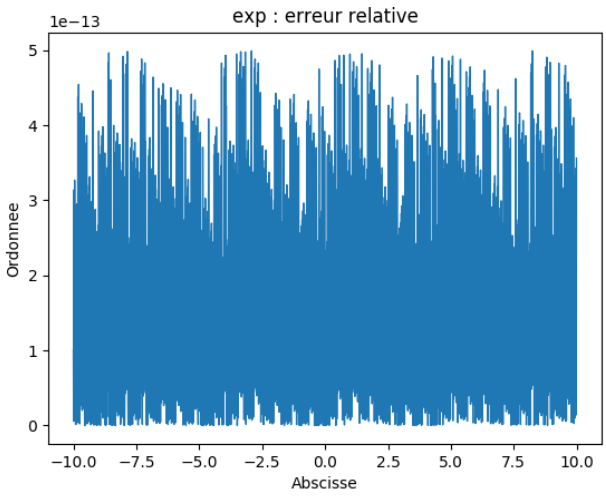
Question 2 Pour représenter les nombres, une calculatrice utilise la représentation normalisée à virgule flottante. L'avantage de cette représentation est qu'elle permet de représenter un intervalle de nombres très important. Un inconvénient de cette représentation est qu'il peut exister plusieurs représentations pour un même nombre. Des normes ont cependant été réalisées de manière à éviter ce genre de problème.

Question 3 La méthode générale consiste d'abord à réduire la valeur de x à une valeur plus petite comprise dans un intervalle où les algorithmes vont être plus efficaces. Ensuite, des valeurs précalculées de la fonction voulue nous permettent de calculer la valeur au point x. Cette méthode est efficace sur une calculatrice puisqu'on fait à chaque fois des opérations simples (addition ou soustraction). Les multiplications ou divisions par 10^n sont simplement un décalage de $4 * n$ chiffres binaires, c'est la représentation des nombres qui le permet.

Question 5 Pour tester nos fonctions, nous avons tracé les graphes des erreurs relatives. Les calculs sont faits à chaque fois sur 2000 points sur un intervalle de taille 20. On obtient à chaque fois des erreurs relatives de l'ordre de 10^{-13}, excepté en quelques points spéciaux, $ln(0)$ par exemple.

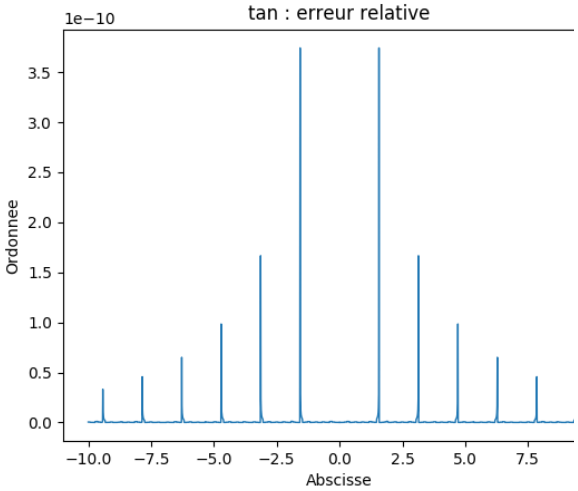


(a) Erreur relative : ln

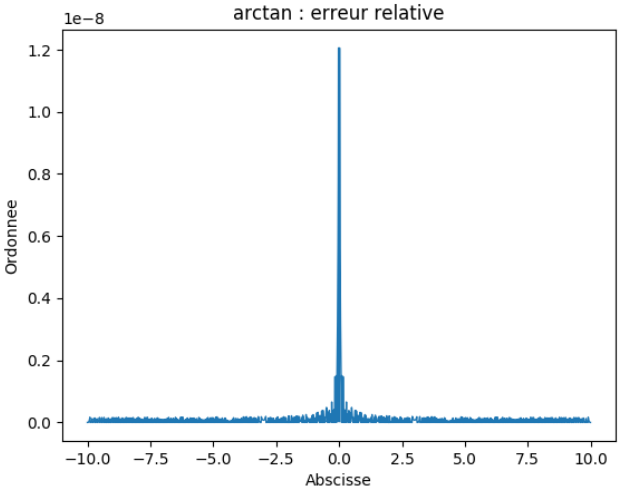


(b) Erreur relative : exp

FIGURE 2.1 – Erreurs relatives ln et exp



(a) Erreur relative : ln



(b) Erreur relative : exp

FIGURE 2.2 – Erreurs relatives tan et arc

Question 6 Calculer une fonction analytique grâce à une série entière est rarement applicable pour une utilisation informatique. Prenons l'exemple du sinus :

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

Cette série ne s'approche pas de $\sin(x)$ tant que $\|x\|$ n'est pas négligeable devant k, ce qui contraint à calculer bien souvent un grand nombre de termes, d'autre part la série est alternée. Ceci pose deux grands problèmes ; les erreurs s'accumulent avec le grand nombre d'additions effectuées, et l'alternance des signes induisent des erreurs qui s'amplifient, et qui influe donc sur la précision du résultat.

Calculer des fonctions trigonométrique en temps réel n'est pas une bonne façon de procéder, le calcul est en effet lourd et il existe des manières plus simples de procéder. En remarquant que :
$\cos(n\theta) = 2\cos(\theta)\cos((n-1)\theta) - \cos((n-2)\theta)$
$\sin(n\theta) = 2\cos(\theta)\sin((n-1)\theta) - \sin((n-2)\theta)$
Il suffit de calculer $\cos(\theta)$ et $\sin(\theta)$ et de les garder en mémoire pour pouvoir ensuite calculer par la relation de récurrence en temps réel cette fois tous les $\cos(n\theta)$ et $\sin(n\theta)$ qui en découlent.

Pour calculer la dérivée d'une fonction, utiliser l'approximation du premier ordre est classique : $f'(x) \sim \frac{f(x+h)-f(x)}{h}$
Cependant, il est expliqué dans le chapitre 5.7 que selon certaines valeurs de h, $f(x+h)$ pouvait être proche de $f(x)$, il est décrit que calculer : $f'(x) \sim \frac{f(x+h)-f(x-h)}{2h}$ était souvent plus adapté.