Licence de Sciences et Technologies
S6 - mention Informatique

Université de Perpignan Via Domitia

TD2 : Analyse d'erreur et qualité numérique

Travaux dirigés notés

Les modalités de rendu se trouvent en dernière page.

Ce TD s'inspire d'un sujet de 2014.

1 Sommer n flottants

Cet exercice permet la mise en pratique de l'illustration des différents points vus en cours concernant l'arithmétique élémentaire de la norme IEEE-754[1], la perte de précision lors d'un enchaînement de calculs, la perte de propriétés arithmétiques sur la précision de la solution calculée avec un algorithme inverse stable. On utilisera MPFR pour plus de flexibilité dans la manipulation des formats et des calculs.

On s'intéresse au calcul de la somme s_n de n nombres flottants x_i : $s_n = \sum_{i=1}^{n} x_i$. Il s'agit d'observer la qualité numérique de différents algorithmes qui calculent cette somme pour des jeux de données $(x_i)_i$ de conditionnement variables. On choisit de fixer la longueur de la somme à une valeur arbitraire $n = 100$; la somme s_{100} sera notée plus simplement s.

Les données. Dans l'archive fournie sur l'ENT, vous trouverez un répertoire `data2018` qui contient des jeux de données pour des conditionnements d'ordre de grandeur 10^c avec $c \in \{3, 6, 9, 12, 15, 16, 18, 20, 24, 28, 32\}$. Quatre jeux de données sont disponibles pour chaque ordre de grandeur des conditionnements. Au total, on dispose donc de 44 jeux d'opérandes, chacun dans un fichier.
Chaque fichier comporte 101 lignes et stocke le jeu de données comme suit :

- ligne 1 : n,`cond` : le nombre d'opérandes à sommer (ici $n = 100$) et le conditionnement de leur somme s ;
- ligne 2-101 : les opérandes $x_i, i \in [1, n]$ en `binary64` ;

Les algorithmes. On considère les 9 algorithmes de sommation suivants (+ 2 facultatifs).

A1. dans l'ordre croissant des indices : $((x_0 + x_1) + x_2)$...,

A2. dans l'ordre décroissant des indices,

A3. opérandes positifs puis négatifs dans l'ordre croissant des indices,

A4. opérandes négatifs puis positifs dans l'ordre croissant des indices,

A5. somme partielle S_+ des opérandes positifs, puis somme partielle S_- des opérandes négatifs chacune dans l'ordre croissant des indices, puis somme de S_+ et S_- ,

A6. Ordre croissant des valeurs absolues des opérandes

A7. Ordre décroissant des valeurs absolues des opérandes

[1] Zuras, Dan, et al. "IEEE standard for floating-point arithmetic." IEEE Std 754-2008 (2008): 1-70.

A8. addition récursive par paire (*pairwise*) :
$((x_0 + x_1) + (x_2 + x_3)) + ((x_4 + x_5) + (x_6 + x_7))$ pour $n = 8$ par exemple.

A9. Sommation compensée

A10. [Facultatif : Sommation k-fois compensée]

A11. [Bonus : supprimer les deux opérandes de plus plus petite valeur absolue, les additionner, ajouter cette somme comme un nouvel opérande et recommencer (addition des deux plus petites valeurs absolues, ...) jusqu'à ce qu'il n'y ait plus d'opérande à additionner.]

Les mesures. L'erreur relative entre une somme calculée $\hat{s}$ et la somme exacte s mesure la précision de $\hat{s}$. La somme exacte s est en général inconnue en pratique. Ici, nous nous proposons d'en obtenir une approximation fiable en sommant avec une précision suffisante (`200` bits) à l'aide de MPFR. En arrondissant correctement cette somme en `b64`, nous obtenons ainsi la somme exacte s. On aura donc : `ErrRel`$(s, \hat{s}) = |s - \hat{s}|/|s|$.

L'étude.

1. Les sources

   (a) Afin de gagner du temps sur le développement des fonctions basiques qui seront utilisées dans la suite du TP, vous trouverez sur GitHub les sources d'un programme "vide" lisant les flottants des fichiers de données (qu'il contient déjà)[2].

   (b) Pour le télécharger

      i. Si vous avez git, tapez simplement dans une console
      `git clone https://github.com/christof337/AlgoEtCalculScientifique.git`

      ii. Sinon, vous pouvez télécharger le zip à cette adresse
      `https://github.com/christof337/AlgoEtCalculScientifique`, bouton (vert) "Clone or download", puis "Download zip".

   (c) Afin de prendre en main le programme, placez vous dans la fonction `main` du fichier `TD2.c` et remplissez le tableau (déjà initialisé) `arraySum` avec la valeur de la somme s des éléments de chaque fichier. On rappelle que cette somme (simple) devra être calculée avec une précision de `200` (`PRECISION_LARGE`). Elle sera automatiquement arrondie à 53 bits lorsque copiée dans le tableau `arraySum` (à l'aide de `mpfr_set`). Elle servira de référence s dans la suite du TD. **Le format flottant mpfr de précision 53 bits sera utilisé dans tous les calculs suivants.**

      i. À titre indicatif et à des fins de correction, la valeur de la somme attendue pour le fichier `TD2-File01-N100-C10e3.txt` est `-9.2509937555610233e-01`

2. Les algorithmes

   (a) Comprendre que les 9 algorithmes de sommation considérés ne diffèrent entre eux que par l'ordre des sommations partielles, c'est-à-dire par les parenthésages de l'évaluation de $\hat{s}$ de s. Imaginer une proposition de vérification expérimentale de cette propriété (code ou pseudo-code).

   (b) Justifier que ces algorithmes sont inverse-stables.

3. Mesure de la perte de précision d'une somme calculée.

   (a) Expliciter et coder `ErrRelBits`, l'erreur relative dans la somme calculée $\hat{s}$ mesurée comme le nombre de bits significatifs de $\hat{s}$.

   (b) Proposer et coder une vérification expérimentale de cette mesure appliquée à l'erreur d'arrondi de la représentation de précision 53 bits de MPFR (mantisse aussi grande qu'en `binary64`).

[2] Vous êtes libres de vous en servir ou non, mais étant donné le temps imparti il est recommandé de réutiliser les fonctions implémentées

(c) Expliciter et coder `NbBitsPerdus`, le nombre de bits non significatifs dans $\hat{s}$. **Cette mesure sera utilisée dans toutes les questions suivantes.**

4. Dans les questions suivantes, il s'agit de comparer la précision perdue par les différents algorithmes A1, A2, ..., et l'effet du conditionnement sur cette perte de précision. On procède en deux temps.

   (a) À l'aide des différents jeux de données fournis, comparer pour chaque algorithme l'évolution de la perte de précision, en nombre de bits significatifs perdus, à son nombre de conditionnement fixé. Qu'en conclure?

   (b) Générer deux fichiers `moy.data` et `max.data` qui contiennent respectivement la moyenne et le maximum de cette perte de précision pour *chaque* algorithme et *chaque* ordre de grandeur du conditionnement. On pourra utiliser la méthode `writeMatrix` de `inputOutput`.

   (c) Écrire un fichier de commandes `gnuplot`[3] pour tracer les deux graphiques suivants à partir des fichiers `moy.data` et `max.data`.

      i. Le nombre moyen de bits perdus (ordonnée) comme fonction du conditionnement (abscisse),

      ii. Le nombre maximum de bits perdus comme fonction du conditionnement.

      Le conditionnement sera exprimé sous la forme d'une puissance de 2. Une échelle logarithmique est adaptée à son affichage. Chaque graphique regroupera les résultats de l'ensemble des algorithmes.

   (d) Commenter les résultats ainsi obtenus.

Le rendu On rappelle que **2** séances de TP seront accordées à ce travail. Le rendu se fera soit par mail à christophe.pont@univ-perp.fr, soit via l'ENT avant le **lundi 26/03 à 23h**. L'attendu est :

1. Un fichier de réponses aux questions en pdf[4]. Il devra contenir les graphiques demandées à la question 4.(c).

2. Votre code source zippé, avec, lorsque c'est possible, les numéros de questions en en-tête de méthodes. Vous ne serez pas noté sur la qualité du code, simplement sur la précision à laquelle il répond à la problématique.

Si une question vous pose problème et que vous n'êtes pas parvenu à obtenir de l'aide à son sujet, expliquez quel a été votre raisonnement et les pistes explorées (par exemple, un algorithme en pseudo-code répondant au problème peut être accepté).

[3] `http://gnuplot.sourceforge.net/`

[4] Éventuellement au format papier, auquel cas le rendu se fera le lundi 26 à 17h à la fin de la séance au plus tard.