TELECOM Nancy 1A 2025-26
Mathématiques Numériques MN1

Feuille 1 : arithmétique flottante

Remarque. Les exercices "étoilés" * sont généralement un peu plus difficiles ou simplement un peu redondants. Ils ne seront pas nécessairement traités en séance TD.

1. Nombres flottants, représentations, approximations

Exercice 1. *Un premier ensemble de flottants*

On se place dans l'ensemble de flottants $\mathcal{F} = \mathbb{F}(10, 4, -4, 4)$.

1. Calculer les nombres caractéristiques M, m, μ de cet ensemble.
2. Donner la valeur de $fl(1,234 \cdot 10^{-3})$, $fl(1,0015)$, $fl(1,234 \cdot 10^{-5})$, $fl(1,234 \cdot 10^{5})$, $fl(9,9995 \cdot 10^{4})$ et $fl(9,99949 \cdot 10^{4})$

Exercice 2*NOUVEAU.* *En base 2*

1. Soit $x = (c_0, c_1 \dots c_{p-1})_2 \cdot 2^e \in \mathcal{F} = \mathbb{F}(2, p, e_{min}, e_{max})$ avec $x < M$. On considère $x' \in \mathcal{F}$ le *premier flottant suivant* x. Montrer que le milieu de x et x' vaut $\frac{x+x'}{2} = (c_0, c_1 \dots c_{p-1}1)_2 \cdot 2^e$.
2. Dans $\mathbb{F}(2, 4, -2, 2)$, calculer les valeurs $fl(1, 101011)$, $fl(1, 01110111)$, $fl(1, 0101)$.

Exercice 3. *Changements de base*

1. Calculer $0,2$ en base 2 (aide : partir de $0,2 = \sum_{i>=1} d_i 2^{-i}$ avec $d_i = 0$ ou 1)
2. Même question pour $0,1$, $0,4$, $0,8$, etc. . . !
3. Il s'ensuit qu'en introduisant ces nombres dans un ordinateur fonctionnant en base 2 on commet nécessairement une (petite) erreur. Montrer que :

$$fl(0.2) = (\underbrace{1,100}_{1}\ \underbrace{1100}_{2} \dots \underbrace{1100}_{12}\ \underbrace{1101}_{13}\ 0)_2\ 2^{-3}$$

dans $\mathbb{F}(2, 53, -1022, 1023)$ (cad en double précision), aide : $53 = 13 \times 4 + 1$. En base 10 ce nombre s'écrit exactement 0.2000000000000000011102230246251565404236316680908203125. C'est donc ce nombre que vous devez obtenir lorsque vous demandez une sortie avec suffisamment de chiffres (le format adéquat sera 56.54f).

Exercice 4. *Unicité de l'arrondi*

Dans l'approximation d'un nombre réel par un flottant avec l'arrondi au plus proche, en cas d'existence de 2 flottants à égale distance du nombre réel, on choisit celui dont le dernier chiffre de la mantisse est *pair*. Le but de l'exercice est de montrer que parmi 2 nombres flottants *consécutifs*, il y en a un qui a le dernier chiffre de la mantisse *pair* et l'autre *impair*.

On considère le système de flottants $\mathcal{F} = \mathbb{F}(\beta, p, e_{min}, e_{max})$ avec une base β **paire**. Soient deux nombres flottants de $\mathcal{F}$, positifs (pour simplifier) avec les écritures de position suivantes :

$$x = \left(c_0, c_1 \dots c_{p-1}\right)_\beta \beta^e$$
$$x' = \left(c'_0, c'_1 \dots c'_{p-1}\right)_\beta \beta^{e'}$$

On suppose que x' est le *premier flottant suivant* x (i.e. le plus petit flottant x' tel que $x' > x$).

Donner c'_{p-1} en fonction de c_{p-1} et e' en fonction de e dans les deux cas suivants :

1. Cas où $0 \leq c_{p-1} < \beta - 1$.
2. Cas où $c_{p-1} = \beta - 1$.

Dans chacun des cas, montrer que si l'un des derniers chiffres c_{p-1} ou c'_{p-1} est pair, l'autre est nécessairement impair.

Exercice 5. *Epsilon machine*

Soit le système flottant $\mathcal{F} := \mathbb{F}(\beta, p, e_{min}, e_{max})$ et un nombre réel non-nul, positif (pour simplifier) $x \in [m, \tilde{M}[$. La précision machine (epsilon machine) est définie à partir de l'erreur relative entre x et son approximation flottante $fl(x)$ dans $\mathcal{F}$.

1. Soit deux nombres flottants $f_1 < f_2$ *consécutifs* dans $\mathcal{F}$ [1] avec $f_1 = \mathfrak{m}\beta^e$ et la mantisse $\mathfrak{m} \geq 1$ (écriture normalisée). Calculer la différence $ulp = f_2 - f_1$.
2. En déduire une majoration de l'erreur absolue $|x - fl(x)|$.
3. Montrer que $\dfrac{|x - fl(x)|}{|x|} \leq \mathbf{u} := \dfrac{\beta^{1-p}}{2}$. Le nombre $\mathbf{u}$ est l'epsilon machine.

2. Opérations sur les flottants, analyse d'erreurs

Exercice 6. *Non associativité de l'addition machine*

On se place dans $\mathbb{F}(10, 8, -\infty, +\infty)$ avec la règle de la norme IEEE :

Lors d'un calcul machine, tout se passe comme si chaque opération était effectuée exactement avant d'appliquer la fonction d'arrondi fl qui donne le nombre flottant le plus proche.

1. Calculer le epsilon machine $\mathbf{u}$ de cet ensemble de nombres flottants.
2. Montrer que :
$$(11111113 \oplus -11111111) \oplus 7,5111111 = 9,5111111$$
(le résultat obtenu étant exact) puis que :
$$11111113 \oplus (-11111111 \oplus 7,5111111) = 10$$
ce qui donne une erreur relative beaucoup plus importante que $\mathbf{u}$ (malgré le fait que les deux calculs consécutifs sont effectués chacun avec une précision relative inférieure à $\mathbf{u}$).

Exercice 7. *Arrondi au plus proche avec parité et stabilité*

Dans l'arrondi d'un nombre réel par un nombre flottant au plus proche, en cas d'égale distance entre deux flottants on choisit usuellement celui dont le dernier bit de la mantisse est *pair*. Ce choix de parité permet de stabiliser certains calculs. L'objectif de cet exercice est d'illustrer cette propriété de stabilité. Pour cela, on considère le système flottant $\mathcal{F} = \mathbb{F}(10, 8, -\infty, +\infty)$ avec les deux flottants suivants :
$$u = 1,000\,000\,0 \qquad v = 0,555\,555\,55$$
On va effectuer des calculs machine entre u et v, en utilisant deux critères différents pour choisir le flottant en cas d'égale distance entre deux flottants.

1. *Premier critère* : en cas d'égale distance entre deux flottants, on choisit toujours le flottant *le plus grand*. Calculer, en appliquant la règle de la norme IEEE rappelée dans l'exercice précédent, les quantités $u_1 = (u \oplus v) \ominus v$, puis $u_2 = (u_1 \oplus v) \ominus v$ et $u_3 = (u_2 \oplus v) \ominus v$.

   En déduire que $u < u_1 < u_2 < u_3 < \ldots$.

1. éventuellement dans $\mathbb{F}(\beta, p, e_{min}, +\infty)$ pour gérer le cas $x > M$.

2. *Deuxième critère* : en cas d'égale distance entre deux flottants, on choisit le flottant dont le dernier chiffre de la mantisse est *pair*. Calculer à nouveau u_1. Conclure.

Exercice 8. *Une façon de noter l'erreur relative*

Soit $x \in \mathbb{R}$, $x \neq 0$ et y une approximation de x telle que l'erreur relative entre x et y soit inférieure à E :

$$\left|\frac{y-x}{x}\right| \leq E$$

Montrer qu'il est équivalent de dire qu'il existe un réel e (que l'on peut appeler erreur relative signée) avec $|e| \leq E$ pour lequel $y = x(1+e)$.

Exercice 9. *Analyses d'erreur*

Dans cet exercice, on suppose que les calculs effectués par la machine ne provoquent ni overflow ni underflow et que sa F.P.U. respecte la norme IEEE, et donc si x et y sont deux nombres flottants alors :

$$x \odot y = fl(x \cdot y) = (x \cdot y)(1+\epsilon) \text{ avec } |\epsilon| \leq \mathbf{u} \tag{1}$$

D'autre part on pourra utiliser le fait que la multiplication ou la division par une puissance de 2 est exacte (sauf overflow ou underflow bien sûr).

1. Soient a, b, c, d et e des nombres flottants, on cherche à calculer $x = abc/(de)$. On suppose que ce calcul sera effectué selon le parenthésage : $x_c = ((a \otimes b) \otimes c) \oslash (d \otimes e)$. Donner une majoration de l'erreur relative entre x_c et le résultat exact x (utiliser le lemme de l'exercice 11*). On en déduira que, sauf problèmes d'overflow ou d'underflow, une séquence raisonnable de multiplications/divisions ne pose pas de problème de précision en arithmétique flottante.
2. Soit des nombres flottants $x_1, x_2, \ldots, x_n$. On cherche à calculer $S^{(n)} = \sum_{k=1}^{n} x_k$. Dans la suite, on suppose qu'on ne rencontre pas de problèmes d'overflow ou d'underflow lors des calculs. L'algorithme usuel consiste à sommer les nombres de la gauche vers la droite, soit selon le parenthésage :

   $$S_c^{(n)} = ((((((x_1 \oplus x_2) \oplus x_3) \oplus x_4) \cdots \oplus x_n)$$

   Ou de manière équivalente : $S_c^{(1)} = x_1$ puis pour $k > 1$, $S_c^{(k)} = S_c^{(k-1)} \oplus x_k$.

   (a) Montrer (par une récurrence facile) que :

   $$\begin{array}{rll} S_c^{(n)} = & (x_1+x_2)(1+\epsilon_1)(1+\epsilon_2)(1+\epsilon_3)\ldots(1+\epsilon_{n-1}) & \\ + & x_3(1+\epsilon_2)(1+\epsilon_3)\ldots(1+\epsilon_{n-1}) & \\ + & x_4(1+\epsilon_3)\ldots(1+\epsilon_{n-1}) & \text{avec } |\epsilon_k| \leq \mathbf{u}, \forall k \\ + & \ldots & \\ + & x_n(1+\epsilon_{n-1}) & \end{array}$$

   où ϵ_i est l'erreur relative signée lors de la i ème addition [2].

   (b) En déduire, en utilisant le lemme de simplification que :

   $$S_c^{(n)} - S^{(n)} = x_1\delta_{n-1} + x_2\delta_{n-1} + \sum_{k=3}^{n} x_k\delta_{n+1-k} \text{ avec } |\delta_k| \leq \frac{k\mathbf{u}}{1-k\mathbf{u}}, \forall k$$

   puis (si $S^{(n)} \neq 0$) que :

   $$\frac{|S_c^{(n)} - S^{(n)}|}{|S^{(n)}|} \leq \kappa_x \, \frac{(n-1)}{1-(n-1)\mathbf{u}} \, \mathbf{u}, \quad \text{avec } \kappa_x = \frac{\sum_k |x_k|}{|\sum_k x_k|}$$

2. D'après les hypothèses (absence d'overflow et d'underflow) on a $S_c^{(i+1)} = S_c^{(i)} \oplus x_{i+1} = fl(S_c^{(i)} + x_{i+1}) = (S_c^{(i)} + x_{i+1})(1+\epsilon_i)$.

Ainsi quand les x_k sont tous de même signe $\kappa_x = 1$, l'erreur relative est alors "quasi" bornée par $(n-1)\mathbf{u}$ (et beaucoup plus petite en pratique). Dans le cas contraire un coefficient d'amplification (κ_x appelé nombre de conditionnement pour la somme des x_k est alors supérieur à 1) apparait dans la majoration et l'erreur relative peut donc être plus grande en particulier lorsque $|\sum_k x_k| \ll \sum_k |x_k|$.

3. Un problème important avec l'arithmétique flottante est la soustraction de deux nombres flottants de même signe et de magnitude voisine, ce qui semble contradictoire puisque ce calcul sera en général exact (le théorème de *Sterbenz* nous dit que si x et y sont 2 flottants de même signe tels que $|x|/2 \leq |y| \leq 2|x|$ alors $x \ominus y = x - y$). Le problème est l'amplification des éventuelles erreurs contenues dans les 2 nombres que l'on soustrait. Soit deux nombres a et b résultats d'un calcul. Suite aux erreurs d'arithmétique flottante (et éventuellement aussi de l'introduction de données) les nombres obtenus par l'ordinateur sont entachés d'erreur, soit $A = a + \delta a$ et $B = b + \delta b$. On notera $r_a = a/(a-b)$, $r_b = b/(a-b)$ et e_a, e_b les erreurs relatives signées sur a et b. Montrer que l'erreur relative signée sur le résultat, c'est-à-dire $\frac{(A \ominus B)-(a-b)}{a-b}$ est égale à :

$$r_a e_a - r_b e_b$$

(on supposera que a et b sont suffisamment proches et que les erreurs commises sont suffisamment petites pour que $A \ominus B = A - B$). Donner des exemples numériques concrets de perte de précision (on pourra choisir par exemple $a = 10^p + 1$ et $b = 10^p$ de sorte que $a - b = 1$ avec p de plus en plus grand). Revoir aussi la question 2 à la lumière de ce résultat.

4. On cherche à calculer la fonction :

$$\phi(x) = \frac{1}{1+x} - \frac{1}{1-x} = \frac{-2x}{(1+x)(1-x)} \tag{2}$$

pour un nombre flottant x tel que $|x|$ est assez petit. Analyser l'erreur obtenue par les deux « algorithmes » :

(a) $y2 := (-2 \otimes x) \oslash ((1 \oplus x) \otimes (1 \ominus x))$

(b) $y1 := (1 \oslash (1 \oplus x)) \ominus (1 \oslash (1 \ominus x))$ (aide : utilisez le résultat de la question précédente en posant $A = 1 \oslash (1 \oplus x)$ et $B = 1 \oslash (1 \ominus x)$).

Exercice 10 *. *Analyse d'erreurs : un peu de* `python` *pour vérifier la théorie*

On reprend l'analyse d'erreurs de la fonction ϕ de l'exercice précédent, définie par (2). L'objectif de cet exercice est de réaliser une étude numérique pour mettre en évidence les défauts de la première formule (en se servant de la deuxième comme référence).

Pour cela vous allez écrire un script/module python :

1. qui va balayer des petits nombres grâce à la fonction `logspace` du module numpy [3] :

```
n = 2000
x = logspace(-17,-1,n)
```

Ceci permet d'obtenir un vecteur x dont les composantes vont de 10^{-17} à 10^{-1} avec n composantes en tout et une répartition logarithmique (on a $x_i/x_{i+1} = Cte$).

2. qui calcule ensuite les ordonnées correspondantes aux deux formules. Avec y_2 supposée "exacte", calculer l'erreur absolue (`ea = abs(y1 - y2)`) puis l'erreur relative.

3. Avec spyder et en exécutant le fichier via l'interface, vous disposez directement des fonctionnalités des modules numpy et matplotlib (sans avoir à rajouter de préfixe) il est donc en fait inutile de rajouter l'instruction `from pylab import *` dans votre fichier.

3. et finalement qui affiche l'erreur relative en échelle `log`.
   Attention, dans certains cas les deux formules donneront le même résultat (c-a-d que la première fonction marche bien sur certains arguments) et l'erreur relative obtenue (0 donc) ne peut pas s'afficher en échelle log (pourquoi ?). Lors de l'affichage de la courbe vous pouvez sélectionner les composantes des vecteurs qui correspondent à une erreur absolue non nulle grâce à un indiçage booléen obtenu avec l'expression `ea>0`. La courbe en échelle log-log s'obtient avec `loglog(x[ea>0], er[ea>0], ...)`

   Dans l'exercice précédent, vous avez dû montrer qu'une "quasi" borne pour l'erreur relative est :

$$|e_r| \lesssim \frac{2\mathbf{u}}{|x|}$$

   Visualiser cette borne dans le même graphe.

   Rmq : pour obtenir un graphe en échelle loglog, il suffit de remplacer `plot` par `loglog`. Comme python travaille en double précision, on a $\epsilon_m = 2^{-53}$.

Question : pourquoi la partie gauche de la courbe est constante et égale à 1 lorsque $|x|$ est suffisamment petit (vous pouvez mieux mettre ce phénomène en évidence en partant de 10^{-18} plutôt que de 10^{-17}) ?

Exercice 11*. *Encore une analyse d'erreurs*

On cherche à calculer la fonction :

$$f(x) = \sqrt{1+x} - \sqrt{1-x}$$

pour un nombre flottant x assez petit (en valeur absolue).

1. Analyser d'abord l'erreur obtenue par "l'algorithme" immédiat :

$$y1 := sqrt(1 \oplus x) \ominus sqrt(1 \ominus x)$$

   où $sqrt(u)$ désigne la racine calculée en machine (pour cette opération la norme IEEE impose aussi que $sqrt(u) = fl(\sqrt{u})$, on a donc $sqrt(u) = \sqrt{u}(1+\epsilon)$ avec $|\epsilon| \leq \mathbf{u}$ pour tout flottant $u \geq 0$ qui n'est pas un nombre spécial (car avec la racine carrée $[\sqrt{\mu}, \sqrt{M}] \subset [m, M]$)).
2. Réécrire f différemment pour trouver le bon algorithme (lorsque $|x|$ est petit). Rappel : $\sqrt{1+x} \approx 1 + x/2 + O(x^2)$.

Exercice 12*. *Un lemme de simplification*

Lorsque l'on conduit une étude de la précision d'une formule en arithmétique flottante on se retrouve souvent avec des quantités de la forme suivante :

$$Q = \frac{(1+\epsilon_1)(1+\epsilon_2)\dots(1+\epsilon_k)}{(1+\epsilon_{k+1})\dots(1+\epsilon_n)}$$

où les ϵ_i sont très petits devant 1. En développant en série les termes en $1/(1+\epsilon)$, on obtient :

$$Q = 1 + \delta, \text{ avec } \delta = \epsilon_1 + \dots + \epsilon_k - \epsilon_{k+1} - \dots - \epsilon_n + \text{ termes croisés}$$

Ainsi si on néglige les termes croisés, il vient $|\delta| \leq \sum_{i=1}^{n} |\epsilon_i|$ soit $|\delta| \leq n\mathbf{u}$ si $|\epsilon_i| \leq \mathbf{u}$ pour tout i. Cependant cette borne n'est pas exacte du fait que l'on a négligé les termes croisés.

Une manière élégante de traiter ce problème est d'utiliser le résultat suivant : si $|\epsilon_k| \leq \mathbf{u}$, $s_k = \pm 1$ pour $k = 1, \dots n$ et si $n\mathbf{u} < 1$ alors :

$$\prod_{k=1}^{n} (1+\epsilon_k)^{s_k} = 1 + \delta \quad \text{avec} \quad |\delta| \leq \frac{n\mathbf{u}}{1-n\mathbf{u}}$$

Démontrer ce résultat par récurrence sur n.

Exercice 13. *Problème d'overflow et d'underflow parasites*

Parfois au cours d'un calcul, la magnitude d'une quantité intermédiaire q peut ne pas tomber dans l'intervalle $[m, M]$ [4] alors que le résultat final pourrait être correctement approché dans le système flottant. On parle dans ce cas d'overflow ou d'underflow parasite, l'exemple le plus simple étant celui du calcul de la norme d'un vecteur (ici en dimension 2 mais c'est encore plus vrai en dimension supérieure) : $\sqrt{x^2 + y^2}$.

1. donner des cas d'overflow et d'underflow parasites pour les flottants 8 octets ;
2. trouver un algorithme simple qui permet d'éviter ce problème puis obtenir une majoration de l'erreur relative de ce dernier.

Exercice 14 *. *La bonne façon de résoudre l'équation du second degré*

Soit l'équation du second degré $ax^2 + bx + c = 0$. On se place dans le cas où $a \neq 0$ et $\Delta = b^2 - 4ac > 0$.

1. Dans le cas où $|4ac| \ll b^2$ quel problème peut-on rencontrer dans le calcul des racines ?
2. Comment peut-on remédier à ce problème (aide : $x_1 x_2 = c/a$) ?

4. Plus précisément si $|q| < m$ avec $fl(q) \neq q$ on obtient alors une perte de précision (erreur relative plus forcément bornée par $\mathbf{u}$) et dans le cas où $|q| \geq \tilde{M}$ on obtient un `Inf`.

ensembles de flottants	$\mathbb{F}(2, 24, -126, 127)$	$\mathbb{F}(2, 53, -1022, 1023)$
$\mathbf{u}$	$5.9604645\ 10^{-8}$	$1.1102230246251565\ 10^{-16}$
m	$1.1754944\ 10^{-38}$	$2.2250738585072014\ 10^{-308}$
M	$3.4028235\ 10^{38}$	$1.7976931348623157\ 10^{308}$

TABLE 1 – valeurs caractéristiques (approchées) des deux jeux de flottants usuels