TD sur l'arithmétique flottante 16 janvier 2008

Prérequis : Représentation d'un nombre en virgule fottante, définition formelle d'un systeme fini de représentation en virgule flottante, approximation d'un réel par un nombre flottant, flottants IEEE, conséquences sur les calculs, calculs d'erreurs, overflow et underflow parasites.
Durée : 1 h 50

TD 1 – Arithmétique flottante

Exercice 1

/ Changements de base */*

1. Calculer $0,2$ en base 2 (aide : partir de $0,2 = \sum_{i\geq 1} d_i 2^{-i}$ avec $d_i = 0$ ou 1)
2. Même question pour $0,1$, $0,4$, $0,8$, etc ...

Exercice 2

/ Analyses d'erreur */*

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

$$u \odot v = fl(u \cdot v) = (u \cdot v)(1+\epsilon) \text{ avec } |\epsilon| \leq \epsilon_m \tag{1}$$

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). Lors des calculs suivants, on aura intérêt à simplifier des termes que celui de gauche ci-dessous par le terme de droite :

$$\frac{\prod_{i=1}^{k}(1+\epsilon_i)}{\prod_{i=k+1}^{n}(1+\epsilon_i)} = 1+\delta \tag{2}$$

Pour cela on pourra utiliser le résultat suivant :
Lemme de simplification : sous les conditions $n\epsilon \leq 0,1$, $\epsilon < 5\ 10^{-3}$, et $|\epsilon_i| < \epsilon$, la valeur de δ dans l'équation (2) vérifie :

$$|\delta| < 1,06\ n\epsilon.$$

Rmq : en simple précision $\epsilon \simeq 6\ 10^{-8}$ la condition $n\epsilon \leq 0,1$ est respectée jusqu'à des valeurs de n de l'ordre de $1,6\ 10^{6}$.

1. Soient a, b, c, d et e des nombres flottants, on cherche à calculer $x = abc/(de)$ qui, écrit sous la forme `x = (a*b*c)/(d*e)` ou encore `x := (a*b*c)/(d*e)` selon le langage de programmation utilisé, sera effectué selon la séquence :

   $$x_c = ((a \otimes b) \otimes c) \oslash (d \otimes e)$$

   Donner une majoration de l'erreur relative entre x_c et le résultat exact x. 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. Même question avec x_1, x_2, x_3 des nombres flottants de même signe avec le calcul $S = x_1 + x_2 + x_3$. Généraliser à une somme à n termes. Refaire la question en supposant que les nombres n'ont pas forcément le même signe.

3. Un des problèmes importants avec l'arithmétique flottante est la soustraction de deux nombres flottants proches, ce qui semble contradictoire puisque ce calcul sera en général exact : le problème est l'amplification des éventuelles erreurs contenues dans les 2 nombres que l'on soustrait. Soient deux nombres u et v résultats d'un calcul. Ces nombres sont entachés d'erreur et l'ordinateur a obtenu en fait $U = u + \delta u$ et $V = v + \delta v$. On notera $R_u = |u|/|u - v|$, $R_v = |v|/|u - v|$ et e_u, e_v les erreurs relatives sur u et v. Montrer que l'erreur relative sur le résultat est égale à :
$$|R_u e_u \pm R_v e_v|$$
(on suppose ici que U et V sont tels que $U \ominus V = U - V$). Donner des exemples numériques concrets de perte de précision.
4. On cherche à calculer la fonction :
$$\phi(x) = \frac{1}{1+x} - \frac{1}{1-x} = \frac{-2x}{(1+x)(1-x)}$$
pour un nombre flottant x tel que $|x|$ est assez petit. Analyser l'erreur obtenue par les deux « algorithmes » :
   (a) $y1 := (1 \oslash (1 \oplus x)) \ominus (1 \oslash (1 \ominus x))$
   (b) $y2 := (-2 \otimes x) \oslash ((1 \oplus x) \otimes (1 \ominus x))$
5. Même question avec la fonction $f(x) = \sqrt{1+x} - \sqrt{1-x}$. On analysera 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 $sqrt(u) = \sqrt{u}(1+\epsilon)$). Puis on cherchera à réécrire f différemment pour trouver le bon algorithme (lorsque $|x|$ est petit). Rappel : $\sqrt{1+x} \approx 1 + x/2 + O(x^2)$.

Exercice 3

/ 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 $[m, M]$ [1] 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 4 octets ;
2. trouver un algorithme simple qui permet d'éviter ce problème.

Exercice 4

/ 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 $|q| < m$ on obtient alors une perte de précision via une troncature à 0 ou l'utilisation d'un nombre dénormalisé et dans le cas où $|q| > M$ on obtient un Inf.

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

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

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

Exercice 5

/ Exemple de problèmes avec les float en JAVA */*

On considère le fichier JAVA de la page suivante. Compte-tenu du fait que le type *float* de JAVA représente le système flottant $\mathbb{F}(2, 24, -126, 127)$, donnez et expliquez le résultat de l'exécution[2].

Rappels de cours

/ Au cas où ... */*

→ Pour représenter informatiquement (coder) les nombres, on se donne un système fini de représentation en virgule flottante : $\mathbb{F} = \mathbb{F}(\beta, p, e_{min}, e_{max})$ où

- β = entier ($\beta > 2$) définissant la base
- p = nombre de chiffres de la mantisse
- e_{min} = exposant minimum
- e_{max} = exposant maximum

Cet ensemble $\mathbb{F}$ correspond à tous les nombres réels x qui s'écrivent :

$$x = \pm \left(\sum_{i=0}^{p-1} c_i \beta^{-i} \right) \beta^e \text{ où } \left\{ \begin{array}{l} \forall i, 0 \leq c_i \leq \beta - 1 \\ e_{min} \leq e \leq e_{max} \end{array} \right.$$

→ L'opérateur fl associe à tout réel x le flottant le plus proche de $\mathbb{F}$ et, dans le cas où deux flottants sont possibles, on choisit celui dont le chiffre de poids faible est pair (tout comme le mode arrondi IEEE). De même, quelque soient x et $y \in \mathbb{F}$, on a : $x \odot y = fl(x.y)$, où $\cdot$ est l'une des 4 opérations arithmétiques.

→ Le plus grand nombre positif représentable dans le système $\mathbb{F} = \mathbb{F}(\beta, p, e_{min}, e_{max})$ est : $M = (1 - \beta^{-p})\, \beta^{e_{max}+1}$ et le plus petit est : $m = \beta^{e_{min}}$

→ Etant donné $x \in \mathbb{R}$, il faut associer à x une approximation $fl(x) \in \mathbb{F}(\beta, p, e_{min}, e_{max})$:

- lorsque $|x| > M$: $\left\{ \begin{array}{l} fl(x) = +\infty \text{ si } x > M \\ fl(x) = -\infty \text{ si } x < -M \end{array} \right.$
- lorsque $|x| < m$: $fl(x) = \pm 0$ selon le signe de x
- lorsque $m \leq |x| \leq M$, l'approximation $fl(x)$ est le nombre flottant le plus proche de x, c'est à dire que si

$$x = \pm \left(\sum_{i=0}^{+\infty} x_i \beta^{-i} \right) \beta^e$$

alors :

- si $x_p < \beta/2$, on arrondit "en dessous" : $fl(x) = \pm \left(\sum_{i=0}^{p-1} x_i \beta^{-i} \right) \beta^e$
- si $x_p \geq \beta/2$ avec au moins un indice $j > p$ tel que $x_j \neq 0$, on arrondit "au dessus" : $fl(x) = \pm \left(\sum_{i=0}^{p-2} x_i \beta^{-i} + (x_{p-1} + 1) \beta^{-(p-1)} \right) \beta^e$
- si $x_p = \beta/2$, il y a plusieurs façon d'arrondir. Dans cet exercice, on retient le chiffre dont le dernier chiffre de la représentation soit pair càd :
  - si x_{p-1} est pair, alors on arrondi "en dessous"
  - si x_{p-1} est impair, alors on arrondi "au dessus"

→ Les opérations $\pm inf \oslash \pm inf$, $\pm inf \otimes \pm 0$, $\pm 0 \oslash \pm 0, \ldots$ produisent NaN (Not a Number).

[2] `puis(x,a)` $= x^a$, `val_abs(x)` $= |x|$ et `racine(x)` $= \sqrt{x}$

```
public class Tests {

   /* 1ere méthode pour calculer racine_carrée(x^2 + y^2) */
   public static float methode_1(float x, float y){
       return racine(puis(x,2)+puis(y,2));
   }

   /* 2nde méthode pour calculer racine_carrée(x^2 + y^2) */
   public static float methode_2(float x, float y){
        if(val_abs(x)>val_abs(y))
              return val_abs(x)*racine(1+puis(y/x,2));
        else
              return val_abs(y)*racine(1+puis(x/y,2));
   }

   /* 1ère méthode pour calculer tanh(x) */
   public static float tanh_1(float x){
        return (exponentielle(x) - exponentielle(-x))
                 /(exponentielle(x) + exponentielle(-x));}

   /* 2nde méthode pour calculer tanh(x) */
   public static float tanh_2(float x){
       if(x >= 0)
           return (1 - exponentielle(-2*x))/(1 + exponentielle(-2*x));
       else
           return (exponentielle(2*x) - 1)/(exponentielle(2*x) + 1);
   }

    public static void main(String[] args){
       float x = 2*puis(10,19);
       float y = x;
             System.out.printf("Test 1 : sqrt(x2+y2) = %f \n",methode_1(x,y));
             System.out.printf("test 2 : sqrt(x2+y2) = %f \n",methode_2(x,y));
       /*************************************************/
       x = 2*puis(10,-20);
       y = x;
             System.out.printf("Test 3 : x/x = %f \n",x/x);
       /*************************************************/
       x = 2*puis(10,-50);
       y = x;
            System.out.printf("Test 4 : x/x = %f \n",x/x);
       /*************************************************/
       x = 89;
            System.out.printf("Test 5 : tanh(x) %f \n",tanh_1(x));
       /*************************************************/
       x = 89;
            System.out.printf("Test 6 : tanh(x) %f \n",tanh_2(x));
       /*************************************************/
       x = 3;
       y = 3 + puis(10,-20);
             System.out.printf("Test 7 : (x-y).10^20 = %f \n",(x-y)*puis(10,20));
   }
}
```