TD1 - Erreurs numériques

Alexandra Bac

Méthodes numériques
Polytech Marseille - IRM 3A

Dans ce TP, on va explorer la représentation des réels et des exemples de ses conséquences sur le calcul numérique. Le TP sera fait en Matlab et permettra une première prise en main du langage.

La commande `format long` permet d'avoir un affichage avec 6 chiffres significatifs en Matlab.

1 Représentation des réels

La représentation d'un nombre réel x est basée sur le décomposition suivante :

$$x = \underbrace{\pm}_{\text{bit de signe}} \left(\underbrace{b_1 2^{-1} + b_2 2^{-2} + \cdots + b_n 2^{-n}}_{\text{mantisse } m} \right) \cdot 2^l \qquad l : \text{exposant}$$

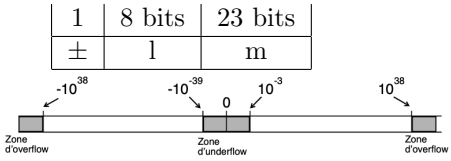
avec :

- $b_i \in \{0, 1\}$
- En représentation normalisée : $b_1 = 1$

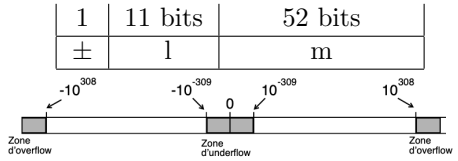
Exposant et mantice sont stockés sur un nombre de bits dépendant du type de données (*float* ou *double)*. Et puisque $b_1 = 1$, ce **premier bit est implicite** et n'est pas stocké dans le codage de la mantisse.

En norme IEEE 754 (2008), les *float* et *double* sont donc respectivement représentés de la manière suivante :

Représentation des `float` (ou simple précision)



Représentation des `double` (ou double précision)



Une particularité : l'exposant est représenté en décalage, donc sur l bits, 0 code $-(2^{l-1})-1$ et 2^l-1 code 2^{l-1}. Par exemple sur 8 bits ($2^8=256$, $2^7=128$), on obtient :

0	127	255
↓	↓	↓
-127	0	128

Donc un exposant sur l bits code des valeurs dans l'intervalle $[-2^{l-1}-1, 2^{l-1}]$.

Représentation des `float` (ou simple précision)

- $l \in [-127, 128]$
- $m \in [2^{-1}, 2^{-1}+2^{-2}+\cdots+2^{-24}]$, ie $m \in [2^{-1}, 1-2^{-24}]$ $m \in [0.5, 0.9999]$

Représentation des `double` (ou double précision)

- $l \in [-1023, 1024]$
- $m \in [2^{-1}, 2^{-1}+2^{-2}+\cdots+2^{-53}]$, ie $m \in [2^{-1}, 1-2^{-53}]$ $m \in [0.5, 0.9999]$

La représentation est approchée Contrairement aux entiers, la représentation des flottants n'est pas exacte (car elle implique une conversion en base 2). Ainsi, par exemple :

$$0,25 = \frac{1}{2^2} = 2^{-2} = (\underbrace{1}_{b_1} \cdot 2^{-1}) \cdot 2^{\overbrace{-1}^{l}} \qquad \text{Représentation exacte}$$

$$0,75 = 3 \cdot 2^{-2} = (1+2^1) \cdot 2^{-2} = (\underbrace{1}_{b_1} \cdot 2^{-1} + \underbrace{1}_{b_2} \cdot 2^{-2}) \cdot 2^{\overbrace{0}^{l}} \qquad \text{Représentation exacte}$$

Décomposition d'un flottant $x \in]0,1[$ en base 2 :

- Si $x = b_1 2^{-1} + b_2 2^{-2} + \cdots$
- Alors $2x = b_1 + (b_2 2^{-1} + b_3 2^{-2} + \cdots)$, donc b_1 est la partie de $2x$ supérieure à 1 et $b_2 2^{-1} + b_3 2^{-2} + \cdots$ le reste.

Décomposition de $0.1 = b_1 2^{-1} + b_2 2^{-2} + \cdots$:

$$\begin{array}{llll} 0,1*2=0,2 & \rightarrow & b_1=0 & \text{reste } 0,2 \\ 0,2*2=0,4 & \rightarrow & b_2=0 & \text{reste } 0,4 \\ 0,4*2=0,8 & \rightarrow & b_3=0 & \text{reste } 0,8 \\ 0,8*2=1,6 & \rightarrow & b_4=1 & \text{reste } 0,6 \\ 0,6*2=1,2 & \rightarrow & b_5=1 & \text{reste } 0,2 \\ \cdots & & & \end{array}$$

La représentation étant normalisée, le premier bit (associé à 2^{-1}) doit être égal à 1, ici, il correspond donc à b_4 et l'exposant est donc de -3 (car $1 \cdot 2^{-4} = (1 \cdot 2^{-1})2^{-3}$. En représentation normalisée, on obtient donc la décomposition de $0,1$:

- Exposant : -3
- Mantisse : $1100110011001100\ldots$
- Cette suite est infinie et la représentation de $0,1$ est donc **inexacte et approchée**

L'erreur est relative ! Nous l'avons vu, la représentation est inexacte. Donc le dernier bit est (potentiellement) inexact. Quelle erreur cela représente-t-il (on considère ici que l'on est en simple précision) ?

- Pour $l = -127$, les nombres représentables appartiennent au segment $[2^{-1}2^{-127}, (1-2^{-24})2^{-127}[= [2.9\cdots 10^{-39}, 5.8\cdot 10^{-39}[$
  - L'erreur porte sur le dernier bit et est donc de l'ordre de $2^{-24}2^{-127} = 3.5\cdot 10^{-46}$
- Pour $l = 1$, les nombres représentables appartiennent au segment $[2^{-1}2, (1-2^{24})2[= [1, 2[$
  - L'erreur porte sur le dernier bit et est donc de l'ordre de $2^{-24}2 = 1.2\cdot 10^{-7}$
- Pour $l = 128$, les nombres représentables appartiennent au segment $[2^{-1}2^{128}, (1-2^{24})2^{128}[= [1.7\cdot 10^{38}, 3.4\cdot 10^{38}[$
  - L'erreur porte sur le dernier bit et est donc de l'ordre de $2^{-24}2^{128} = 2\cdot 10^{31}$

Donc l'erreur est relative !

2 TP

Exercice 1 (Erreur et représentation des réels)**.** On cherche à savoir quelle est la plus petite valeur que l'on puisse ajouter à un réel x pour modifier sa valeur (dans le cas des `float` et des `double`). On pourra partir d'une variable $\delta = 1$ et itérer ensuite la division de δ par 2 jusqu'à ce que x et $x+\delta$ soient identiques. Quelle est alors la plus petite valeur de δ ne modifiant pas x ?

Vous ferez cet exercice pour $x = 1$ puis pour des valeurs grandes et petites de x. Vous tracerez la coube de δ en fonction de x.

Enfin vous interprétez en fonction de la section 1.

Exercice 2. Dans cet exercice, on souhaite tester l'associativité (ou non) des opérations numériques. On s'intéresse à la somme. On sait bien que la somme est associative dans $\mathbb{R}$, qu'en est-il de la somme numérique de réels. On considèrera des `single`.

Calculer $(x+y)+z$ et $x+(y+z)$ pour :

- $x = 1, y = 1, z = 1$

- $x = 1.23456e^{-3}, y = 1.0, z = -y$

D'une manière générale, il est préférable de commencer par sommer les chiffre de plus grande valeur absolue, pourquoi ?

Exercice 3. On souhaite comparer $f_1(n) = n \times \frac{1}{n}$, $f_2(n) = \underbrace{\frac{1}{n} + \cdots + \frac{1}{n}}_{n \text{ fois}}$ et le résultat théorique de ce calcul : 1.

Vous ferez cette comparaison pour des valeurs de n de plus en plus grandes tout d'abord en `float`. Pour cela :

1. Vous construirez un vecteur N contenant les valeurs $[1, 10^1, 10^2, \ldots, 10^9]$. Vous écrirez une fonction renvoyant deux vecteurs x_1 et x_2 contenant les images des éléments de N par rapport à f_1 et f_2 respectivement. Vous calculerez ensuite les vecteurs booléen des différences entre x_1, x_2 et 1, puis les valeurs des différences entre 1 et x_2. Vous utiliserez le moins possible de `for` mais plutôt, quand c'est possible, les opérations matricielles de Matlab.

2. Interprétez ce résultat.

3. Faites de même en `double`. Interprétez votre résultat. A quelle puissance de 10^{-1} faut-il monter pour voir une différence ? Testez.

4. Que se passe-t-il si vous remplacez les puissances négatives de 10 par des puissances négatives de 8 ? Interprétez.

5. Pour tirer parti des résultats de l'exercice 2, il est préférable de réaliser le calcul $\frac{1}{n} + \cdots + \frac{1}{n}$ en parenthésant pour avoir les chiffres les plus similaires possibles à sommer. Par exemple :

$$((\frac{1}{8} + \frac{1}{8}) + (\frac{1}{8} + \frac{1}{8})) + ((\frac{1}{8} + \frac{1}{8}) + (\frac{1}{8} + \frac{1}{8}))$$

Ecrire une fonction récusive réalisant le calcul $\underbrace{x + \cdots + x}_{n \text{ fois}}$ (en `single`) "équilibré" et comparer avec le résultat obtenu en `single` pour la somme itérative basique.

Exercice 4. Ecrivez un programme comparant (en `float`) les résultats des deux calculs suivants :

$$\left(\sum_{i=1}^{n} i \times h\right) / \ k$$

et

$$\sum_{i=1}^{n} (i \times (h/k))$$

On pourra commencer par prendre $h = 0.3333333$ et $k = 0.333333$ (attention au nombre de chiffres après la virgule !) et prendre $n = 10, 100, 1000, 10000, 100000$. Vous comparerez avec la valeur théorique (évidente à calculer) et interprétez.

Exercice 5. On s'intéresse à la suite numérique :

$$u_{n+1} = \frac{a}{c + c \cdot u_n}$$

pour différentes valeurs de a, b, c et u_0.

1. La limite est donnée par le point fixe d'une fonction évidente. Calculez la limite théorique de la suite.
2. Ecrivez une fonction calculant (en `double`) la limite de la suite à ε près.
3. Testez cette fonction en prenant tout d'abord $a = b = c = u_0 = 1$.
4. Testez ensuite la fonction en prenant $a = 5 \cdot 10^{10}$. Que vaut la limite théorique ? Quelle est la limite numérique sur votre machine ?