

# CI 1 : ARCHITECTURE MATÉRIELLE ET LOGICIELLE

## CHAPITRE 5 – CONSÉQUENCES DE LA REPRÉSENTATION LIMITÉE DES NOMBRES RÉELS EN MACHINE

Savoir

### Savoirs

- Capacité Dec - C3 : Initier un sens critique au sujet de la qualité et de la précision des résultats de calculs numériques sur ordinateur
- Principe de la représentation des nombres entiers en mémoire
- Principe de la représentation des nombres réels en mémoire

1	Problèmes de précision	1
2	Précision machine [2]	2
	2.1 Mise en évidence de l'influence du codage sur la précision	2
3	... et Python	4
	3.1 Premiers exemples	4
	3.2 Dépassement de capacité	5
	3.3 Erreurs de comparaisons	5
	3.4 Problèmes liés à la conversion de décimal en flottant	6

## 1 Problèmes de précision

A cause de la base utilisée, il est impossible de représenter exactement la plupart des nombres décimaux. Des nombres qui habituellement ne posent pas de problème dans les calculs en mathématiques deviennent ainsi une source d'erreurs multiples.

*On désire représenter le nombre 0,4 en virgule flottante au format simple précision.*

1 – Convertir en binaire la partie fractionnaire du nombre

$$\begin{array}{rclclcl}
 0,4 & \times & 2 & = & 0,8 & = & 0 & + & 0,8 \\
 0,8 & \times & 2 & = & 1,6 & = & 1 & + & 0,6 \\
 0,6 & \times & 2 & = & 1,2 & = & 1 & + & 0,2 \\
 0,2 & \times & 2 & = & 0,4 & = & 0 & + & 0,4 \\
 0,4 & \times & 2 & = & 0,8 & = & 0 & + & 0,8 \\
 & & & & & & \dots & & 
 \end{array}$$

Exemple

Par conséquent :  $0,4_{(10)} = 0,0110\ 0110\ 0110\ 0110\dots_{(2)}$ .

Le nombre 0,4 admet pour développement en base 2 un développement infini périodique.

2 – Décaler la virgule vers la gauche pour le mettre sous la forme normalisée (IEEE 754) :

$0,4_{(10)} = 1,1001100110\dots_{(2)} \times 2^{-2}$  : -2 : décalage de 2 bits vers la droite

3 – Codage du nombre réel avec les conventions suivantes :

- signe = 0 : nombre positif;
- Exposant décalé :  $-2 + 127 = 125_{(10)} = 0111\ 1101_{(2)}$

« 1 » pour avoir la  
valeur représentable  
la plus proche

S	Exposant								Mantisse																						
0	0	1	1	1	1	1	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	1
	3			E			C		C			C			C			C			D										

La représentation en virgule flottante sera donc forcément une valeur approchée de ce nombre.

D'après ce codage, la valeur approchée choisie pour 0,4 est :

0,4000000059604644775390625

Si la dernière valeur de la mantisse prenait la valeur 0, comme le suggérerait l'item 1, on aurait :

0,39999997615814208984375

Exemple

Remarque

Pour vérifier vos calculs, on peut se reporter aux sites :

- <http://babbage.cs.qc.cuny.edu/IEEE-754/>
- [http://www.binaryconvert.com/result\\_float.html](http://www.binaryconvert.com/result_float.html)

## 2 Précision machine [2]

### 2.1 Mise en évidence de l'influence du codage sur la précision

Soit  $F$  l'ensemble des nombres en virgule flottante que l'on peut représenter. Si l'on représente la mantisse avec un mot de  $t$  bits, les éléments  $f \in F$  sont définis par

$$f = \pm \boxed{1 \mid d_2 \mid \dots \mid d_t} \times 2^{-t} \times 2^e = \pm n \times 2^{e-t}$$

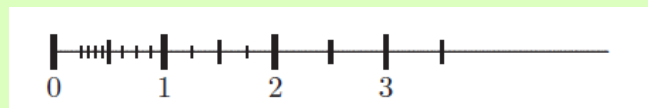
Si l'exposant peut prendre les valeurs entières de l'intervalle  $[L, U]$  on a :

$$m \leq |f| \leq M \quad \text{avec} \quad m = 2^{L-1} \quad \text{et} \quad M = 2^U (1 - 2^{-t})$$

Construction de l'ensemble  $F$  pour  $t = 3$  (nombre de bits de la mantisse) et  $e \in [-1, 2]$  (exposant).

$f = n \times 2^{e-t}$							
$n$			$2^{e-t}$				
			$e = -1$	$e = 0$	$e = 1$	$e = 2$	
			1/16	1/8	1/4	1/2	
1	0	0	$=2^2=4$	1/4	1/2	1	2
1	0	1	$=2^2+2^0=5$	5/16	5/8	5/4	5/2
1	1	0	$=2^2+2^1=6$	3/8	3/4	3/2	3
1	1	1	$=2^2+2^1+2^0=7$	7/16	7/8	7/4	7/2

En reportant ces nombres sur une droite on observe que les nombres en virgule flottante ne sont pas également espacés.



Exemple

Pour décrire l'ensemble des nombres réels qui trouvent une représentation dans  $F$  on définit l'ensemble  $G$  tel que

$$G = \{x \in \mathbb{R} \text{ tel que } m \leq |x| \leq M\} \cup \{0\}$$

Pour arrondir un nombre on utilise la fonction  $\text{float} : G \rightarrow F$  qui peut avoir deux expressions différentes :

$$\text{float}(x) = \text{plus proche } f \in F \text{ à } x \text{ qui satisfait } |f| \leq |x| \text{ (chopping)}$$

ou

$$\text{float}(x) = \text{plus proche } f \in F \text{ à } x \text{ (perfect rounding)}.$$

**Exemple** Soit l'ensemble  $F$  défini pour  $t = 3$  et  $e \in [-1, 2]$  dans l'exemple précédent, alors avec chopping on a  $\text{float}(3.4) = 3$  et  $\text{float}(0.94) = 0.875$  et avec perfect rounding le résultat est  $\text{float}(3.4) = 3.5$  et  $\text{float}(0.94) = 1$ .

**Définition** La précision machine d'une arithmétique en virgule flottante est définie comme le plus petit nombre positif  $\epsilon$  tel que  $\text{float}(1 + \epsilon) > 1$ .

Regardons quelle est la valeur de  $\epsilon$  pour une machine avec des mots dont la mantisse comporte  $t$  bits et lorsqu'on arrondit selon la technique dite chopping. Représentons le nombre 1 et le plus proche nombre suivant :

$$\underbrace{\begin{array}{|c|c|c|c|} \hline 1 & 0 & \dots & 0 \\ \hline \end{array}}_{2^{t-1}} 2^{-t} \times 2^1 = 1 \quad \underbrace{\begin{array}{|c|c|c|c|} \hline 1 & 0 & \dots & 1 \\ \hline \end{array}}_{2^{t-1}+1} 2^{-t} \times 2^1 = 1 + 2^{1-t}$$

La distance qui sépare le nombre 1 du plus proche nombre suivant est égale à  $2^{1-t}$ . Ainsi pour une mantisse comportant  $t$  bits on a :

$$\epsilon = 2^{1-t}$$

On peut montrer que pour une mantisse comportant  $t$  bits et si on arrondit avec la fonction  $\text{float}()$  selon la technique dite "perfect rounding" on a  $\epsilon = 2^{-t}$ . Si on utilise des mots en double précision, qui utilise  $t = 52 \text{ bits}$  pour la mantisse et perfect rounding, la précision machine est alors  $\epsilon = 2^{-52} \approx 2.2210^{-16}$ .

Pour déterminer directement la précision machine en utilisant python on peut utiliser "np.finfo" :



```
>>> np.finfo(np.float64).eps
2.2204460492503131e-16
>>> np.finfo(np.float32).eps
1.1920929e-07
```

Au final l'épsilon machine est ici de :

Simple précision	Double précision
$\epsilon \approx 1,2 \cdot 10^{-7}$	$\epsilon \approx 2,2 \cdot 10^{-16}$

(On retrouve bien la valeur déterminée par le calcul.)

## 3 ... et Python

### 3.1 Premiers exemples

On donne les lignes de codes suivantes :

python

```
a,b,c=3,2,1
while c<20:
    print ("Iter . ",c, " : ",a)
    a,b,c=a*b,a,c+1
```

python

```
a,b,c=3.,2.,1
while c<20:
    print ("Iter . ",c, " : ",a)
    a,b,c=a*b,a,c+1
```

python

```
>>> a=0.1
>>> a=3*a
>>> if (a==0.3) :
    print ("Gagne ! ")
else :
    print ("Perdu ! ")
```

Comment peut-on expliquer ce comportement ?

Dans python, il est possible d'accéder aux informations suivantes :

python

```
>>> import sys
>>> sys.float_info.max
>>> sys.float_info.min
>>> 2.0**(1023)
>>> 2.0**(1024)
>>> 2**(1024)
```

### 3.2 Dépassement de capacité

python

```
>>> a=9
>>> a=a**a
>>> a=a**a
```

python

```
>>> a=9.
>>> a=a**a
>>> a=a**a
```

En python, le dépassement de la capacité maximale peut se manifester sous les aspects *infou* *OverflowError*. Les nombres flottants sont limités à  $10^{308}$ . A l'inverse, les nombres très petits peuvent être arrondis à zéro.

python

```
>>> 10.**309
```

python

```
>>> a=10.**(-323)
>>> a==0, a/10==0
```

### 3.3 Erreurs de comparaisons

La comparaison de deux nombres de type float peut générer des erreurs.

python

```
>>> a,b=1.,1e-16
>>> b>0,a+b>a
```

### 3.4 Problèmes liés à la conversion de décimal en flottant



```
>>> 0.1+0.1+0.1+0.1+0.1+0.1+0.1+0.1
>>> a,b=1+10**(-15),1
>>> (a-b)*10**15
```

## Références

- [1] Christophe François, Représentation de l'information, représentation des nombres.
- [2] Manfred GILLI, Méthodes numériques, Département d'économétrie Université de Genève, 2006.
- [3] Cours d'informatique de PCSI. Lycée Bertran de Born.