# NSI 1ère - Données - Flottants

QK

# Nombre à virgule flottantes

Comment représenter un nombre à virgule en machine ?

**Brisons le mythe :**

- Machine : base 2,
- Humains : base 10

Les *décimaux* sont exprimés en *base 10* ($2 \times 5$).
ils n'ont généralement pas de représentation **exacte** en machine.

# 0.30000000000000004

Les ordinateurs savent manipuler les “nombres à virgules”

```
>>> 1.255465 * 753156.45
945561.5624992499
```

mais les résultats sont parfois surprenants :

```
>>> 0.1 + 0.2
0.30000000000000004
```

## Nombres à virgule flottante

Dans les machines, on utilise les **les nombres à virgule flottante**

Les nombres sont alors appelés des *flottants* (*floats* en anglais)

**L'égalité de deux flottants n'a aucun sens**

# Décimaux

Nos machines travaillent en base 2 et les nombres à virgules flottantes sont représentés de la même manière.

Dans le système décimal on utilise les puissances de 10 avant et après la virgule :
Par exemple 325,47 s'écrit

| | | | | | | |
|---|---|---|---|---|---|---|
| Position | 100 | 10 | 1 | virgule | 1/10 | 1/100... |
| chiffres | 3 | 2 | 5 | . | 4 | 7 |

# Nombres *dyadiques*

Dans la machine on utilise le même principe mais avec des puissances de 2.

On parle de nombres *dyadiques*

Par exemple : $4 + 2 + 1 + 1/2 + 1/8$ et s'écrit en dyadique :

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| `Position` | `4` | `2` | `1` | `virgule` | `1/2` | `1/4` | `1/8` |
| `chiffres` | `1` | `1` | `0` | `.` | `1` | `0` | `1` |

$4 + 2 + 1 + 1/2 + 1/8 = 7.625$

# Revenons sur 0,1 + 0,2

0,1 et 0,2 ont des notations décimales *finies* (ce sont des *décimaux*)

Leur notiation *dyadique* n'est pas finie !

$$0,1 = (0,00011001100110011001100110011001100110011\cdots)_2$$

En machine elle est tronquée (mais sera très proche de 0,1)

Ce n'est *généralement* pas génant : a-t-on souvent besoin d'une telle précision ?

# Une approche naïve

Cette approche est intéressante et naïvement, on pourrait penser que la machine stocke ainsi ses nombres.

**Problème :**
**comment manipuler des nombres très grands et des nombres très petits en même temps ?**
La taille de l'unvivers d'un côté, la taille d'un atome de l'autre !

- Il faudrait des milliers de chiffres...
- Les calculs sont compliqués...

# Contourner la difficulté : la notation scientifique

Pour s'en convaincre :

$A = 300000000 \times 0.00000015$

Clairement la notation décimale *n'est pas adaptée*

On préfère la *notation scientifique* :

$A = 300000000 \times 0.00000015$

$A = (3 \times 10^8) \times (1.5 \times 10^{-7})$

Souvenons nous

- on *multiplie 3 et 1,5*
- on *ajoute les exposants 8 et -7*

$A = (3 \times 1.5) \times 10^{8-7}$

$A = 4.5 \times 10^1$

$A = 45$

La machine fait la même chose... mais en base 2.

## Nombre à virgules flottante

Un **nombre dyadique** est représenté par :

$$\pm(\ 1, b1 \cdots bk\ )_2 \times 2^e$$

où $b1, \ldots, bk$ sont des bits et $e$ est un entier relatif.

La suite de bits $b1 \ldots bk$ est la *mantisse* du nombre,
La puissance de 2 est *l'exposant* du nombre.

# Exemple

$6,25 = (110,01)_2 = (1,1001)_2 \times 2^2$

- La mantisse est la suite `1 0 0 1`
- L'exposant est 2

# Stockage en mémoire

Dans cette norme, les nombres dyadiques sont codés sur 64 bits en réservant :

- 1 bit pour le signe ;
- 11 bits pour l'exposant ;
- 52 bits pour la mantisse.

| | | |
|---|---|---|
| $s$ | $e$ | $m$ |

# Amplitude

Sans entrer dans les détails, en codant sur 64 bits on peut représenter des nombres entre :

- $2^{-1022} \approx 2,23 \times 10^{-308}$ pour le plus petit et
- $2^{1024} - 2^{971} \approx 1,80 \times 10^{308}$ pour le plus grand

Des améliorations sont faites pour les nombres très proches de 0.

Quand un flottant dépasse le plus grand nombre possible il est considéré comme *infini*

```
>>> 2.0 * 10**308 # dépasse le plus grand
inf
```

# Quelques surprises avec inf

```
>>> a = float('inf')     # pour définir inf
>>> a
inf
>>> -a
-inf                     # - inifini
>>> a + a
inf
>>> a - a                # opération interdite
nan                      # not a number
>>> a + a == a
True
>>> b = 2.0 * 10 ** 309  # b = inf
>>> c = 2   * 10 ** 1000 # un integer
>>> c > b                 # inf est plus grand que tous les
False
```

# Deux problèmes dans les calculs avec les flottants

**Absorption**

```
>>> (1. + 2.**53) - 2.**53  # = 1
0.0                         # 1 a été absorbé par les gros
>>> 2.**53 - 2.**53 + 1     # on change l'ordre...
1                           # et ça fonctionne
```

**Annulation**

Soustraire deux nombres proches fait perdre de la précision

```
>>> a = 2.**53 + 1
>>> b = 2.**53
>>> a - b
0.0
```

## Il peut y avoir des conséquences

- Le 25 février 1991, à Dharan en Arabie Saoudite, un missile Patriot américain a raté l'interception d'un missile Scud irakien, ce dernier provoquant la mort de 28 personnes. L'enquête a mis en évidence le défaut suivant :
- L'horloge interne du missile mesure le temps en 1/10s. Ce nombre *n'est pas dyadique* et est converti avec une erreur d'environ 0,000000095s
- Le missile a été mis en route 100h avant son lancement, ce qui entraine un décalage de

$$0,000000095 \times 100 \times 3600 \times 10 \approx 0,34s.$$

- C'est assez pour qu'il rate sa cible. Source