ALGORITHMIQUE · CHAPITRE 7 · FONDATIONS

Comprendre les limites des nombres en machine

Découvrez bits, octets, entiers bornés et complément à deux. Expérimentez les arrondis binaires et comparez des flottants avec une tolérance.

Par Ali ZAINOUL · Cours gratuit

Ce que vous allez apprendre

- Calculer les bornes d'un entier codé sur un nombre fixé de bits.
- Distinguer entier, écriture décimale et flottant binaire.
- Choisir une tolérance justifiée pour un calcul approché.

Prérequis. Savoir lire les puissances de deux et une comparaison.

Pourquoi 0,1 + 0,2 ne vaut-il pas exactement 0,3 en machine ?

En arithmétique exacte, cette égalité est vraie. Pourtant, les flottants binaires usuels stockent des approximations de ces fractions décimales. Pour interpréter un résultat, il faut distinguer la valeur mathématique, le codage en mémoire et le texte affiché.

Bits, octets et entiers bornés

Un bit prend deux valeurs, 0 ou 1. Un **octet** contient huit bits, donc $2^8 = 256$ configurations. Avec n bits, un entier **non signé** peut représenter les valeurs de 0 à $2^n - 1$. Sur huit bits, 11111111_2 vaut 255.

Pour coder aussi des entiers négatifs, il faut choisir une convention. En **complément à deux** sur n bits, le bit de gauche a le poids -2^{n-1} ; les autres gardent leurs poids positifs. La plage va de -2^{n-1} à $2^{n-1} - 1$. Sur huit bits : 11111111 représente -128 + 127 = -1 ; 10000000 représente -128 ; 01111111 représente 127.

Plages sur huit bits

Convention	Minimum	Maximum
Non signé	0	255
Signé en complément à deux	-128	127

Un **dépassement de capacité** survient si le résultat mathématique sort de la plage choisie. Il n'existe pas une réaction universelle : elle dépend du type et du langage. Les entiers `int` Python peuvent grandir dans les limites de la mémoire ; MATLAB stocke par défaut les nombres montrés ici en `double`, un type flottant, et possède aussi des types entiers bornés. Le codage sur huit bits de cette section est un modèle explicite, pas le type de toutes nos variables.

Comprendre une approximation binaire

Un **flottant** représente un nombre avec une précision finie, à l'aide d'un signe, de chiffres significatifs et d'un exposant. En base 10, 1/3 exige une infinité de chiffres ; en base 2, c'est aussi le cas de 1/10. À l'inverse, 0,5 = 1/2 a une écriture binaire finie. La base d'affichage ne détermine pas la précision du stockage.

Les `float` Python usuels et les `double` MATLAB utilisent le format binaire double précision. Tous les entiers jusqu'à 2^{53} inclus y sont exactement représentables, mais pas tous les entiers au-delà : $2^{53} + 1$ est déjà arrondi. Un `int` Python reste distinct d'un flottant.

Comparer avec une tolérance choisie

L'**erreur absolue** est la valeur absolue de la différence entre résultat et référence ; `abs` enlève le signe. Pour cette seule somme, nous acceptons une erreur au plus égale à 10^{-12}, écrit `1e-12`. Cette tolérance n'est pas une constante universelle : elle dépend de la précision requise et des unités du problème. Le pseudo-code décrit ici un calcul approché, puis compare l'erreur aux deux bornes.

Pseudo-code

```
// example.pseudo
// approximate_sum
// Approximate binary arithmetic, tolerance chosen for this example.
result <- 0.1 + 0.2
error <- result - 0.3
close <- (error >= -0.000000000001) AND (error <= 0.000000000001)
PRINT close
```

Python

```python
# approximate_sum.py
result = 0.1 + 0.2
error = result - 0.3
close = abs(error) <= 1e-12
print(result == 0.3, close)
```

MATLAB

```
% approximate_sum.m
result = 0.1 + 0.2;
error = result - 0.3;
close = abs(error) <= 1e-12;
disp([result == 0.3, close]);
```

L’égalité stricte est fausse ; la comparaison approchée est vraie. Python affiche False puis True ; MATLAB affiche 0 puis 1. L’affichage arrondi peut masquer l’écart sans le supprimer. Pour des échelles très différentes, il faut aussi envisager une tolérance relative proportionnelle à la valeur de référence.

Distinguer exactitude et affichage

Remplacez la somme par `0.5 + 0.25` et la référence par `0.75` : l’égalité est vraie, ces fractions ont des écritures binaires finies. Pour des montants à un nombre fixé de décimales, travailler en centimes entiers évite cette approximation dans les additions ; les pourcentages et arrondis nécessitent encore une règle explicite.

Relire le codage avant le résultat

▼ Pourquoi 11111111 peut-il valoir 255 ou -1 ?

Les bits seuls ne précisent pas leur interprétation : 255 en entier non signé sur huit bits ; -1 en complément à deux sur huit bits. Il faut annoncer type et largeur.

▼ La tolérance 10^{-12} convient-elle à toute mesure ?

Non. Une erreur acceptable dépend des unités, de la taille des valeurs et du besoin. La comparaison approchée doit exprimer une exigence du problème.

Référence Python : arithmétique flottante et limites.

À retenir - fiche mémo

Octet

Huit bits offrent 256 configurations ; leur interprétation dépend du type.

Entier borné

Sur n bits : 0 à 2^n - 1 non signé ; -2^{n-1} à 2^{n-1} - 1 en complément à deux.

Flottant

Un nombre binaire de précision finie peut seulement approcher une fraction décimale.

Pseudo-code

```
// example.pseudo
// approximate_comparison
error <- result - reference
close <- (error >= -tolerance) AND (error <= tolerance)
```

Point de vigilance. Une tolérance dépend du problème et de ses unités ; l'affichage arrondi ne rend pas un calcul exact.

Consulter la cheatsheet : Fondations · **Télécharger la fiche du thème en PDF**

Version web : https://www.clearcode.fr/courses/programming/algorithmic/foundations/machine-numbers