Introduction à l'informatique

pour les mathématiques, la physique et les sciences computationnelles

Yann Thorimbert

Chapitre 2 - Partie 2
Codage des réels

Yann Thorimbert

UNIVERSITÉ DE GENÈVE
CENTRE UNIVERSITAIRE D'INFORMATIQUE

Chapitres du cours

1. Origines des ordinateurs et des réseaux informatiques
2. **Codage des nombres** ←
3. Codage des médias
4. Circuits logiques
5. Architecture des ordinateurs
6. Conception et exécution de programmes
7. Algorithmique, programmation et structures de données

Rappel

- Les entiers naturels sont composés d’une information fondamentale :
  - Leur norme.
- Les entiers relatifs sont composés de deux informations fondamentales :
  - Leur signe.
  - Leur norme.
- Les nombres réels sont composés de trois informations fondamentales :

Représentation des réels

- Les entiers naturels sont composés d'une information fondamentale :
  - Leur norme.
- Les entiers relatifs sont composés de deux informations fondamentales :
  - Leur signe.
  - Leur norme.
- Les nombres réels sont composés de trois informations fondamentales :
  - Leur signe.
  - Leur partie entière.
  - Leur partie fractionnaire.

Représentation des réels

- Autrement dit : le réel inclut une information supplémentaire : la position de la virgule.
- Exemple pour le nombre -36.4, qui contient 5 informations pour 3 chiffres.

Signe	**a_2**	**a_1**	**a_0**	**Position virgule**
−1	3	6	4	2

Rappel | **Systèmes positionnels**

- Un nombre peut avoir un **développement fractionnaire** fini dans une base mais infini dans une autre.
- Développement fini si $n = x / b^y$, où x et y sont des entiers positifs.
- Exemples :

  $0.5 = 5 / 10^1 = 1 / 2^1$ a un développement fini en bases 10 et 2.

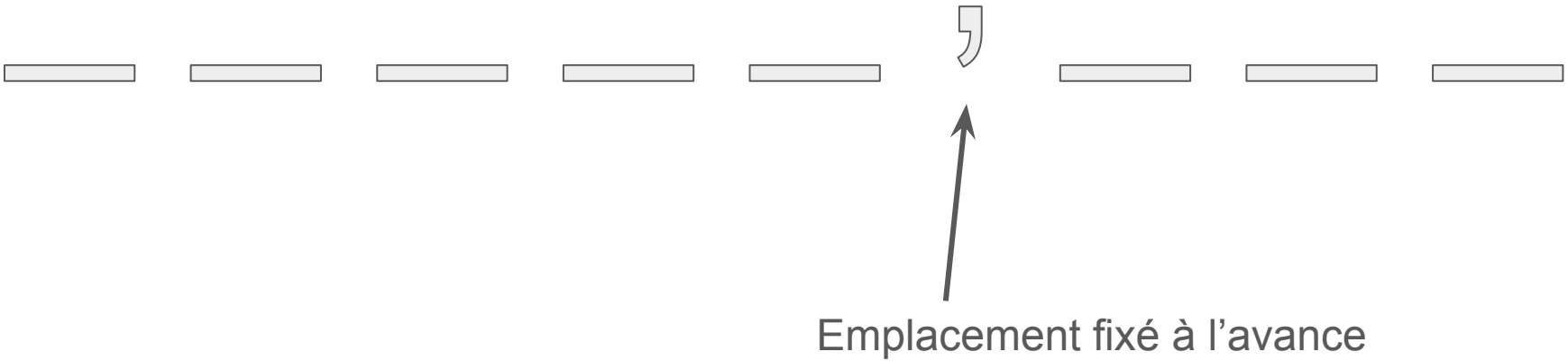
  $0.1 = 1 / 10^1$ n'a pas de développement fini en base 2.

  $1 / 3$ n'a de développement fini dans aucune base entière.

⇒ Une **précision arbitraire** doit être choisie pour approximer un réel quelconque sur un nombre fini de bits.

Représentation de réels en virgule fixe

Exemple sur 8 bits, dont trois après la virgule :



Virgule fixe | **Principe**

- On **fixe** à l'avance, et pour **tous** les nombres représentés, le nombre de chiffres de chaque partie (entière et fractionnaire).

- Élimine le besoin de coder explicitement la position de la virgule.

$$n = s \left(\sum_{i=0}^{k-1} a_i 2^i + \sum_{i=1}^{\ell} f_i 2^{-i} \right)$$

Virgule fixe | **Principe**

- On **fixe** à l'avance, et pour **tous** les nombres représentés, le nombre de chiffres de chaque partie (entière et fractionnaire)

$$n = s\left(\sum_{i=0}^{k-1} a_i 2^i + \sum_{i=1}^{\ell} f_i 2^{-i}\right)$$

- Exemple sur un octet, où $k = 3$ et $\ell = 4$:

s	a_2	a_1	a_0	f_1	f_2	f_3	f_4

Virgule fixe | **Principe**

Exemple sur un octet, où $k = 3$ et $\ell = 4$

codage	s	a_2	a_1	a_0	f_1	f_2	f_3	f_4
valeur	1	1	0	1	0	0	1	0

$$n = (-1)^1 (2^2 + 2^0 + 2^{-3}) = (-1)(4.125) = -4.125$$

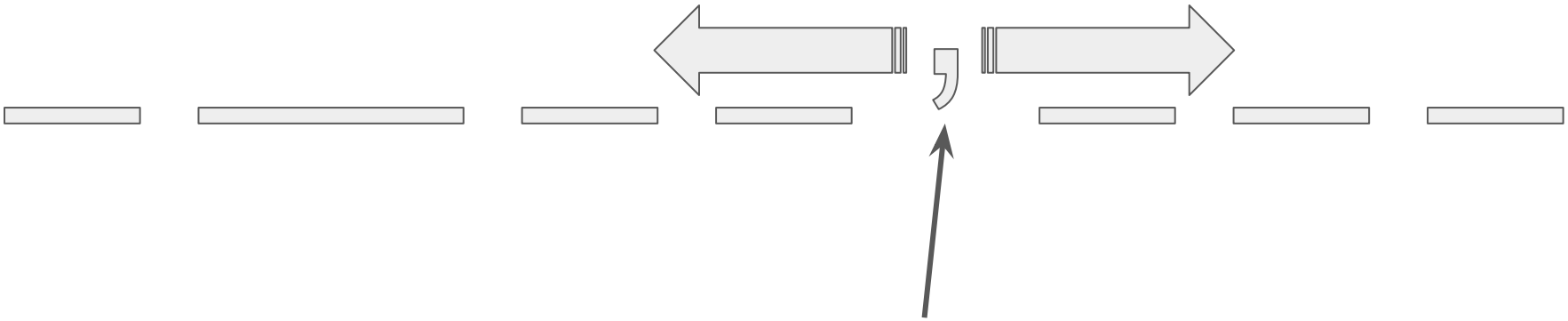
Virgule fixe | **Avantages et désavantages**

✔ Simplicité.

✔ Si l’on connaît la plage de valeurs à représenter, permet d’exploiter au mieux la mémoire disponible.

❌ Ne permet pas de représenter efficacement des valeurs avec des précisions ou des ordres de grandeurs très différents (comme par exemple des valeurs issues de deux appareils de mesure différents en physique).

Représentations de réels en virgule flottante



Virgule flottante | **Principe**

- La **position** de la virgule est **codée** au sein de la séquence de bits.
- Correspond réellement à l'exemple plus haut pour le nombre -36.4, qui contient 5 informations pour 3 chiffres.

Signe	**a_2**	**a_1**	**a_0**	**Position virgule**
−1	3	6	4	2

Virgule flottante | **Principe**

Signe	a_2	a_1	a_0	**Position virgule**
−1	3	6	4	2

- Revient à donner un signe, une norme et un facteur de **mise à l'échelle**.
- Équivalent à …

Virgule flottante | **Principe**

Signe	a_2	a_1	a_0	**Position virgule**
−1	3	6	4	2

- Revient à donner un signe, une norme et un facteur de **mise à l'échelle**.
- Équivalent à la **notation scientifique**.

$$n = s \cdot m \cdot b^E$$

Virgule flottante | **Notation scientifique**

- $n = s \cdot m \cdot b^E$, où s est le **signe**, m la **mantisse**, b la base et E **l'exposant**.
- L'exposant spécifie la position de la virgule.
- La mantisse spécifie la séquence des chiffres au sein desquels placer la virgule.
- Permet de séparer **l'ordre de grandeur** d'un nombre et ses **chiffres significatifs**.

Virgule flottante | **Format de la mantisse**

- $n = s \cdot m \cdot b^E$
- $s \in \{-1;1\}$ et $E \in \mathbb{Z}$
- En base 10, on a pour coutume de choisir $1 \leq m < 10$.

Virgule flottante | **Notation scientifique en base 2**

- En base 2, on choisit $1 \leq m < 2$.

- Ce choix ouvre la porte à une optimisation…

Virgule flottante | **Codage de la mantisse et du signe**

- En base 2, on choisit $1 \leq m < 2$.

- Le bit de poids fort vaudra nécessairement 1 dans ce cas ! On **omet** donc le bit de poids fort de la mantisse. Le codage de la mantisse est alors tel que $\text{cod}(m) = \text{cod}_{\mathbb{N}}(m - 1)$ et $\text{dec}(b) = \text{dec}_{\mathbb{N}}(b) + 1$.

- Si l'on exprime le signe comme $s = (-1)^{cod(s)}$, alors $cod(s) = 1$ correspond à un nombre négatif comme pour les entiers signés.

Virgule flottante | **Format**

- Finalement : $n = s \cdot m \cdot 2^E$, avec une règle de codage pour *m et s*… et bientôt pour E.
- Que vaut le nombre ci-dessous ?

Grandeur codée	***s***	m_2	m_1	m_0	E_1	E_0
Valeur du bit codé	1	1	0	0	0	1

Virgule flottante | **Format**

- Finalement : $n = s \cdot m \cdot 2^E$
- Que vaut le nombre ci-dessous ?

Grandeur codée	s	m_2	m_1	m_0	E_1	E_0
Valeur du bit codé	1	1	0	0	0	1

- $n = (-1)^1 \cdot (1+100_2) \cdot 2^1 = -1 \cdot 5 \cdot 2^1 = -10$.

- Pour exprimer des nombres à virgule, il faut que *E* puisse être un entier relatif.

Virgule flottante | **Format**

- Pour exprimer des nombres à virgule, il faut que *E* puisse être négatif.
- On pourrait utiliser la méthode du complément à 2 pour coder *E*…
- … Mais pour des raisons pratiques, c'est la méthode du biais qui est utilisée.
- On dit que cod(*E*) est l'**exposant biaisé**.

Virgule flottante | **Norme IEEE 754**

En résumé :

- $n = s \cdot m \cdot 2^E$
- cod(s) = 0 si $n \geq 0$, sinon cod(s) = 1
- cod(m) = $\text{cod}_{\mathbb{N}}(m - 1)$
- cod(E) = $\text{cod}_{\mathbb{Z}B+(k')}(E) = \text{cod}_{\mathbb{N}(k')}(E + 2^{k'-1} - 1)$, avec k' le nombre de bits de E.

Virgule flottante | **Norme IEEE 754**

- $n = s \cdot m \cdot 2^E$
- $cod(s) = 0$ si $n \geq 0$, sinon $cod(s) = 1$
- $cod(m) = cod_{\mathbb{N}}(m - 1)$
- $cod(E) = cod_{\mathbb{Z}B+(k')}(E) = cod_{\mathbb{N}(k')}(E + 2^{k'-1} - 1)$
- Formats courants :
  - Sur 32 bits : "**simple précision**". Souvent associé à l'alias `float`.

    1 bit pour *s*, 8 bits pour *E*, 23 bits pour *m* (environ 7 chiffres significatifs en base 10).
  - Sur 64 bits : "**double précision**". Souvent associé à l'alias `double`.

    1 bit pour *s*, 11 bits pour *E*, 52 bits pour *m* (environ 15 chiffres significatifs en base 10).

Virgule flottante | **Norme IEEE 754 - Exemples**

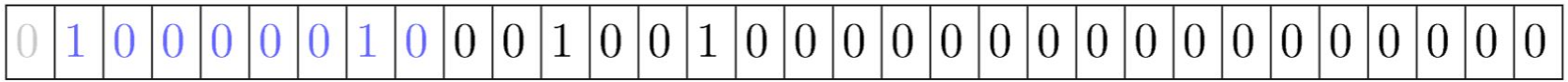
Exemple de bout de code C :

```
double monNombre = 9.125;
```

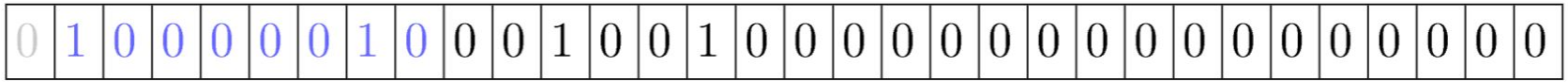
Ou en Matlab :

```
mon_nombre = single(9.125)
```

Dans tous les cas, séquence de bits stockée dans la mémoire de la machine :

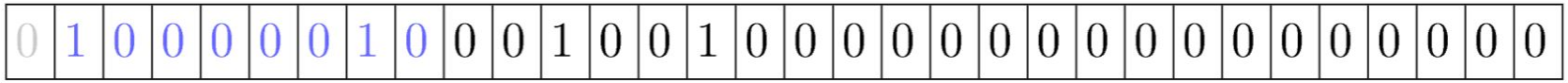


Virgule flottante | **Norme IEEE 754 - Exemples**



- Vérification ! Rappel pour la simple précision :
  $dec(b_E) = dec_{\mathbb{N}(k')}(b_E) - 2^{k'-1} + 1 = dec_{\mathbb{N}(k')}(b_E) - 127$

Virgule flottante | **Norme IEEE 754 - Exemples**



- cod(s) = 0 ⇒ s = $(-1)^0$ = 1.
- cod(E) = 10000010_2 = 130 ⇒ E = 130 - 127 = 3.
- cod(m) = $2^{-3} + 2^{-6}$ = 0.140625 ⇒ m = 1.140625.

⇒ n = s · m · 2^E = 1 · 1.140625 · 2^3 = 9.125

Virgule flottante | **IEEE 754 - Représentation du zéro ?**

- Essayons d’écrire le nombre le plus proche de zéro :

Virgule flottante | **IEEE 754 - Représentation du zéro ?**

- Essayons le nombre positif le plus proche de zéro en précision simple :
  - cod(s) = 0
  - cod(m) = 0 ⇒ m = 1
  - cod(E) = 0 ⇒ E = -127
  - $n = 2^{-127} \neq 0$

Virgule flottante | **IEEE 754 - Représentation du zéro ?**

- Essayons le nombre positif le plus proche de zéro en précision simple :
  - cod(s) = 0
  - cod(m) = 0 ⇒ m = 1
  - cod(E) = 0 ⇒ E = -127
  - $n = 2^{-127} \neq 0$
- On veut tout de même pouvoir représenter le zéro. On le **définit** donc comme correspondant à l'état binaire rempli de zéros.

Virgule flottante | **IEEE 754 - Nombres dénormalisés**

- Plage de valeurs qui rompt avec la convention $n = s \cdot m \cdot 2^E$.
- Concerne tous les nombres avec $E = 0$ et $m \neq 0$.
- Passage à une technique similaire à la virgule fixe dans ce cas :
  - E = constante implicite (-126 pour simple précision).
  - m codée sans le bit implicite.
- Permet de représenter des quantités très petites.
- Pas utilisé dans ce cours.

Virgule flottante | **IEEE 754 - Nombres spéciaux**

Valeurs spécifiques de *E* et *m* réservées. Exemple en simple précision :

cod(*E*)	cod(*m*)	Signification
00000000	0 . . . 0	???
00000000	≠ 0 . . . 0	Nombre dénormalisé
11111111	0 . . . 0	???
11111111	≠ 0 . . . 0	???

N.B : le bit de signe n'intervient pas ici.

Virgule flottante | **IEEE 754 - Nombres spéciaux**

Valeurs spécifiques de *E* et *m* réservées. Exemple en simple précision :

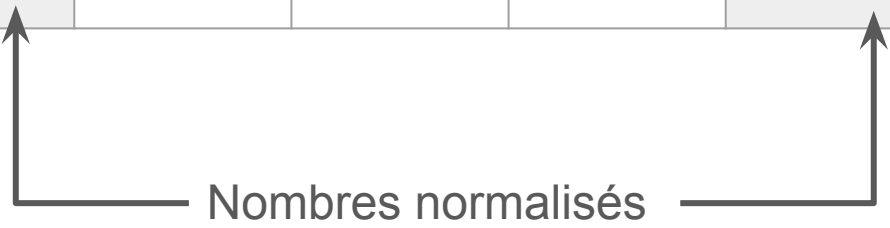
cod(*E*)	**cod(*m*)**	**Signification**
`00000000`	`0 . . . 0`	Zéro exact
`00000000`	`≠ 0 . . . 0`	Nombre dénormalisé
`11111111`	`0 . . . 0`	$s \cdot \infty$
`11111111`	`≠ 0 . . . 0`	NaN (**N**ot **a** **N**umber)

Valeur minimale normalisée : cod(*m*) = 0…0 et cod(*E*) = 00000001 donne environ $\pm 10^{-38}$. Valeur max donne environ $\pm 10^{38}$.

Virgule flottante | **IEEE 754 - Plage de valeurs**

Valeurs spécifiques de *E* et *m* réservées. Exemple en simple précision :

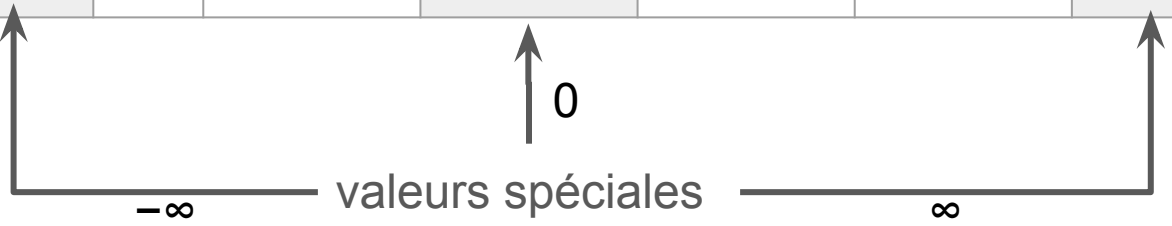
$-\infty$ à -10^{38}		-10^{-38} à 0	0	0 à 10^{-38}		10^{38} à ∞
overflow		underflow		underflow		overflow



Virgule flottante | **IEEE 754 - Plage de valeurs**

Valeurs spécifiques de *E* et *m* réservées. Exemple en simple précision :

$-\infty$ à -10^{38}		-10^{-38} à 0	0	0 à 10^{-38}		10^{38} à ∞
overflow		underflow		underflow		overflow



Virgule flottante | **IEEE 754 - Plage de valeurs**

Valeurs spécifiques de *E* et *m* réservées. Exemple en simple précision :

$-\infty$ à -10^{38}		-10^{-38} à 0	0	0 à 10^{-38}		10^{38} à ∞
overflow		underflow		underflow		overflow

Nombre dénormalisés

(Importance secondaire pour nous)

Virgule flottante | **Précision finie et erreurs d'arrondi**

- Partie fractionnaire pas forcément représentable par une suite finie de bits.
- Partie fractionnaire pas forcément représentable par une suite finie de bits plus courte que **l'espace dont on dispose** !
- Par ailleurs : non seulement on représente un sous-ensemble de $\mathbb{R}$, mais encore au sein de ce sous-ensemble on fait des "sauts".

Virgule flottante | **Précision finie et erreurs d'arrondi**

- Partie fractionnaire pas forcément représentable par une suite finie de bits.
- Partie fractionnaire pas forcément représentable par une suite finie de bits plus courte que **l'espace dont on dispose** !
- Deux problèmes de natures différentes :
  - Un problème mathématique (expression du nombre dans la base utilisée)
  - Un problème informatique (taille de la séquence de bits codant le nombre)

Virgule flottante | **Précision finie et erreurs d'arrondi**

Exemple en simple précision :

- $0.1 = (1.100110011001100\ldots)_2 \cdot 2^{-4}$

- Le codage sur un nombre fini de bits nécessite de **tronquer** la série.

- $dec_{\mathbb{R}32}(cod_{\mathbb{R}32}(0.1)) = 0.100000001490116119384765625$

Virgule flottante | **Précision finie (exemples de code)**

En C :

```
#include <stdio.h>
int main()
{
    float a = 0.1f;
    printf("Valeur : %.27f\n", a);
    return 0;
}
```

En Matlab:

```
a = single(0.1);
fprintf("%.27f\n", a);
```

(Diapositive hors champ)

Virgule flottante | **Comparaisons**

```
#include <stdio.h>
int main(){
    printf("Le programme commence.\n");
    float val = 1.0f;
    for (int i = 0; i < 10; i++)
    {
        val -= 0.2; //on soustrait 0.2 à val
        if(val == 0) //ceci devrait se produire après 5 étapes (ou pas?)
        {
            printf("La valeur est nulle\n");
        }
    }
    printf("Le programme est terminé.\n");
    return 0;
}
```

(Diapositive hors champ)

Virgule flottante | **Accumulation des erreurs (C)**

L'erreur d'arrondi peut se propager à travers des calculs répétés.

```
#include <stdio.h>
int main()
{
    const float delta = 0.2; // increment (essayer 0.25f aussi!)
    const long N = 10000000; // nb d'additions
    float a = 0.; // compteur
    const float theo_result = delta * N;
    printf("delta * N = %f * %ld = %f\n", delta, N, theo_result);
    for(long i=0; i<N; i++)
        a = a + delta;
    printf("delta + delta + ... (N fois) = %f\n", a);
}
```

(Diapositive hors champ)

Virgule flottante | **Accumulation des erreurs (MATLAB)**

L’erreur d’arrondi peut se propager à travers des calculs répétés.

```
d = single(0.2);
N = 10000000;
a = single(0);
theo = d * N;
disp(theo); % affiche 2'000'000
for i=1:N
    a = a + d;
end
disp(a) % affiche 2'175'874
```

(Diapositive hors champ)

Virgule flottante | **Annulation catastrophique**

- Exemple frappant lié aux erreurs d'arrondi en général (pas forcément en programmation).
- On chronomètre deux coureurs : A = 9.44 s et B = 9.56 s.
  $\Rightarrow \Delta = 0.12$ s.
- Si on utilise un chronomètre précis au dixième, alors $A_c = 9.4$ s et $B_c = 9.6$ s.
  $\Rightarrow \Delta_c = 0.2$ s.

(Diapositive hors champ)

Virgule flottante | **Annulation catastrophique**

- On chronomètre deux coureurs : A = 9.44 s et B = 9.56 s.

  $\Rightarrow \Delta = B - A = 0.12$ s.
- Si on utilise un chronomètre précis au dixième, alors $A_c = 9.4$ s et $B_c = 9.6$ s.

  $\Rightarrow \Delta_c = 0.2$ s et alors $\Delta_c / \Delta \approx 167$ % !
- Pourtant $A_c / A \approx 99.6\%$ et $B_c / B \approx 100.4\%$.
- Soustraire deux **bonnes** approximations de nombres proches entre eux peut mener à une **mauvaise** approximation du résultat.

(Diapositive hors champ)

Virgule flottante | **Annulation catastrophique**

- Deux nombres proches entre eux donnent un résultat proche de zéro lorsqu'on les soustrait.
- Par conséquent, les bits issus de l'arrondi endossent une importance relative plus grande que pour des nombres plus éloignés de zéro.
- Dans certains cas de programmation scientifique, il est important de réfléchir aux potentielles erreurs d'arrondi et, si besoin, trouver des solutions pour les contourner.
- Exemple : Beaucoup d'équations physiques ont la forme d'une relaxation vers un équilibre : $x_{t+1} \sim k(x_t - x_0)$.

(Diapositive hors champ)

Exemple de code avec annulation catastrophique

```
// NB : essayez la version double également !
#include <stdio.h>
int main() {
    float a = 1.000001;
    float b = 1.000000;
    float c = 0.000001;

    float result = (a - b) / c;
    printf("Ceci devrait valoir 1: %.12f\n", result);

    return 0;
}
```

(Diapositive hors champ)