Mathématiques Numériques
MN1

Bruno Pinçon /J.-F. Scheid

Télécom Nancy 1A

2025-2026

Version du 18/1/2026

Modules MN

☛ 2 modules de 7 CM (1h) + 7 TD (2h) + 1 examen (2h) chacun.

- **MN1** (obligatoire)
  1. Arithmétique flottante : représentation des nombres et calculs sur ordinateur.
  2. Simuler le hasard
  3. Factorisations de matrice (LU, Cholesky, QR, SVD).
  4. Normes vectorielles et matricielles ; conditionnement de systèmes (linéaires).
  5. Classification non-supervisée : k-means.
- **MN2** (option)
  1. Approximation par interpolation.
  2. Approximation par moindres carrés ; rappels et compléments de calcul différentiel.
  3. Optimisation non-linéaire (cas sans contrainte et avec contraintes).
  4. Introduction aux réseaux de neurones (perceptron multi-couches, rétropropagation du gradient)

Chapitre 1 : Arithmétique flottante

1. Introduction

La représentation en *virgule flottante* (appelée aussi *notation scientifique* et *floating point representation* en anglais) est naturelle pour écrire un nombre grand ou petit en valeur absolue. Exemples :

$A = 6,02252 \cdot 10^{23}$ (le nombre d'Avogadro)
$h = 6,625 \cdot 10^{-34}$ [*joule seconde*] (la constante de Planck)

Elle comporte deux parties :

- la mantisse (ici $6,02252$ et $6,625$) ;
- la partie exposant (23 et -34), le 10 est ici obligatoire car c'est la base choisie pour écrire la mantisse et l'exposant.

La représentation est *normalisée* si [a] : $c_0, c_1 c_2 c_2 c_3 \ldots$ avec $c_0 \neq 0$

a. Autre convention $0, c_1 c_2 c_2 c_3 \ldots$ avec $c_1 \neq 0$

Remarques :

1. Tout nombre réel (sauf zéro) peut s'écrire avec cette notation en virgule flottante normalisée avec en général un nombre de chiffres infini pour la mantisse ; on peut aussi remarquer que certains nombres peuvent s'écrire de deux façons, par exemple $0,999\cdots = 1$.
2. Un changement de base peut introduire quelques bizarreries dans l'écriture d'un nombre alors que celle-ci est anodine dans la base initiale, par exemple (cf TD) :

$$(0,2)_{10} = (0,0011\ 0011\ 0011\ \ldots)_2.$$

3. La mantisse d'un nombre rationnel écrit en virgule flottante est soit finie, soit infinie mais dans ce cas avec un pattern périodique de chiffres, cf exemple précédent ou encore :

$$\frac{1}{3} = (0,33333...)_{10},\ \frac{1}{7} = (0,\underline{142857}\,142857\,142857...)_{10}$$

Quelques exemples de l'importance de l'AF.

- **L'explosion d'Ariane 5 en 1996.**

  Bug logiciel dû à une erreur d'arithmétique flottante de dépassement de capacité (overflow) lors d'une conversion de type.

  - Le logiciel de navigation d'Ariane 5 réutilisait du code d'Ariane 4, sans aucun changement.
  - Une variable contenant la vitesse horizontale de la fusée était stockée sous forme de nombre flottant 64 bits (Ariane 5) et sur 16 bits dans le logiciel d'Ariane 4.
  - Conversion de la valeur de 64 bits à 16 bits.
  - Or, Ariane 5 était plus rapide qu'Ariane 4. La vitesse horizontale était supérieur à 32 767 le plus grand entier signé représentable avec 16 bits (15 bits avec le signe). La conversion a échoué...

- **De l'importance de la précision.**
  Echec du missile patriot en 1991.

  Temps mesuré par une horloge interne en dixième de secondes : doit être multiplié par $1/10$ pour avoir le temps en secondes. Or

  $$(1/10)_{10} = (0.0001100110011001100110011001100....)_2$$

  Avec une mantisse de 24 bits, on obtient le nombre flottant :

  $$(0.00011001100110011001100)_2$$

  avec une erreur de

  $$(0.0000000000000000000011001100...)_2 \simeq 9.5 \cdot 10^{-8} s.$$

  Au bout de 100 heures de fonctionnement, on obtient $9.5 \cdot 10^{-8} \times 100 \times 60 \times 60 \times 10 = 0.342s$.

  Or, un missile se déplace à la vitesse de $1676m/s$, donc pendant $0.342s$ il parcourt $573m$...

2. Ensembles de nombres flottants

Dans un ordinateur, on est obligé de restreindre le nombre de chiffres pour les mantisses et de limiter l'étendue des exposants. On obtient alors des ensembles de nombres flottants.

Ensemble des nombres flottants

On définit l'ensemble des nombres flottants $\mathbb{F}(\beta, p, e_{min}, e_{max})$ à partir de 4 entiers :

- β l'entier définissant la base, $\beta \geq 2$ (*radix* en anglais) ;
- p le nombre de chiffres de la mantisse (*significand* en anglais) ;
- e_{min} l'exposant minimum et e_{max} l'exposant maximum ;

correspondant à tous les nombres réels x s'écrivant :

$$x = s \left(\sum_{i=0}^{p-1} c_i \beta^{-i} \right) \beta^e, \text{ où } \begin{cases} s = \pm 1 \text{ le signe} \\ 0 \leq c_i \leq \beta - 1, \forall i \\ e_{min} \leq e \leq e_{max} \end{cases}$$

La représentation est *normalisée* si $c_0 \neq 0$. Sinon (avec $c_0 = 0$), on dira qu'on a une représentation *dénormalisée* et que les nombres correspondant sont dénormalisés (*subnormal numbers*).

Remarques.

1. La partie mantisse pourra s'écrire $(c_0, c_1 \dots c_{p-1})_\beta$: numération de position.
2. Pour représenter 0, on utilise une écriture spéciale en ne mettant que des 0 dans la mantisse (ce qui est logique) et un exposant de $e_{min} - 1$:

$$0 = (0, 0 \dots 0)_\beta \ \beta^{e_{min}-1}$$

Exercice. L'ensemble de flottants "jouets" $\mathbb{F}(2, 3, -1, 2)$.

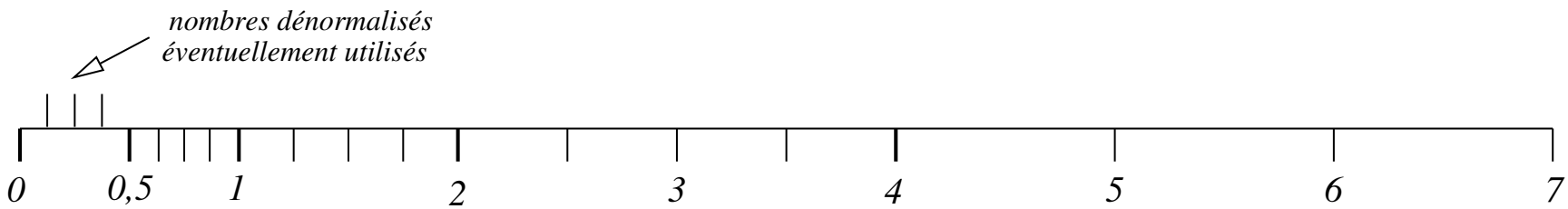
1. Trouver tous les nombres normalisés positifs (plus zéro) de $\mathbb{F}(2, 3, -1, 2)$
2. Les dessiner comme les traits d'une règle graduée.
3. Rajouter les 3 nombres dénormalisés positifs **non redondants**.

   Exemple d'un nombre dénormalisé redondant :

   $(0, 10)_2 \times 2^1$ est une écriture dénormalisée de 1 ; avec notre convention, l'écriture normalisée de 1 est $(1, 00)_2 \times 2^0$.

Solution :

0		
$1,00 \cdot 2^{-1}$	$=$	$(0,5)_{10}$
$1,01 \cdot 2^{-1}$	$=$	$(0,625)_{10}$
$1,10 \cdot 2^{-1}$	$=$	$(0,750)_{10}$
$1,11 \cdot 2^{-1}$	$=$	$(0.875)_{10}$
$1,00 \cdot 2^{0}$	$=$	1
$1,01 \cdot 2^{0}$	$=$	$(1,25)_{10}$
$\vdots$	$\vdots$	$\vdots$
$1,11 \cdot 2^{2}$	$=$	$(7)_{10}$



Nombres dénormalisés non redondants : $(0,11)_2 \times 2^{-1} = (0,375)_{10}$
$(0,10)_2 \times 2^{-1} = (0,25)_{10}$
$(0,01)_2 \times 2^{-1} = (0,125)_{10}$

Remarques.

- Dans les intervalles $[\beta^e, \beta^{e+1}]$ et $[-\beta^{e+1}, -\beta^e]$ l'incrément entre deux nombres flottants est constant et égal à β^{1-p+e}.

  En effet, pour obtenir le nombre suivant ou précédent, on ajoute/retranche

$$(0,0\ldots01)_\beta \times \beta^e = \beta^{1-p} \times \beta^e = \beta^{1-p+e}$$

- Pendant assez longtemps, on a utilisé uniquement les nombres normalisés de ces ensembles (plus le zéro). Cependant, comme la figure précédente le montre, il en résulte un "vide" entre le plus petit nombre normalisé et le zéro. L'utilisation des nombres dénormalisés non redondants permet d'aller vers zéro *plus graduellement*.

Notations.

1. *Nombres flottants remarquables.*
   - M le plus grand nombre positif de $\mathbb{F}(\beta, p, e_{min}, e_{max})$:

$$M = \left(\sum_{i=0}^{p-1} (\beta - 1)\beta^{-i} \right) \beta^{e_{max}} = (1 - \beta^{-p})\beta^{e_{max}+1}$$

   - m le plus petit nombre *normalisé* positif :

$$m = (1, 00...0)_\beta \ \beta^{e_{min}} = \beta^{e_{min}}$$

   - μ le plus petit nombre *dénormalisé* (> 0) :

$$\mu = (0, 00...1)_\beta \ \beta^{e_{min}} = \beta^{1-p+e_{min}}$$

2. *Opérations machines.* On notera $\oplus$, $\ominus$, $\otimes$, $\oslash$ les opérations d'addition, de soustraction, de multiplication et de division, effectuées par l'ordinateur.

3. *Nombres flottants spéciaux.* Dans les systèmes flottants actuels, on rajoute des nombres spéciaux

$$+inf\,,\ -inf \text{ et } NaN$$

(inf pour *infini*, NaN pour *Not a Number*) qui ont une représentation spéciale utilisant en particulier un exposant de $e_{max} + 1$.

Remarque. Du fait du bit de signe, le zéro a deux représentations [1] qui conduisent à des résultats différents sur quelques calculs (par exemple $1 \oslash +0$ donnera $+inf$ alors que $1 \oslash -0$ donnera $-inf$).

1. que l'on notera $+0$ et -0 : si mathématiquement c'est le même nombre, informatiquement non !

Arithmétique des nombres spéciaux.

$$
\begin{aligned}
inf \oslash inf &= NaN \\
inf \otimes 0 &= NaN \\
0 \oslash 0 &= NaN \text{ et/ou exception "division par zéro"}
\end{aligned}
$$

$$
\begin{aligned}
+inf \oplus x &= +inf && \text{si } x \neq -inf, x \neq NaN \\
inf \otimes x &= inf && \text{si } x \neq 0, x \neq NaN \\
inf \oslash x &= inf && \text{si } x \neq inf, x \neq 0, x \neq NaN \\
x \oslash 0 &= inf && \text{si } x \neq 0, x \neq NaN \\
x \oslash inf &= 0 && \text{si } x \neq inf, x \neq NaN
\end{aligned}
$$

3. Approximation d'un réel par un nombre flottant

A partir de maintenant on suppose que la base β est **paire**.

Étant donné $x \in \mathbb{R}$, en général $x \notin \mathbb{F}(\beta, p, e_{min}, e_{max})$ et on associe à x une approximation $fl(x) \in \mathbb{F}(\beta, p, e_{min}, e_{max})$ avec le critère suivant :

Approximation flottante : arrondi au plus proche

$fl(x)$ est le flottant le plus proche de x ; si x est à égale distance de deux flottants, on choisit celui dont le dernier chiffre de la mantisse est *pair*[a].

a. en base 2, le dernier chiffre à 0.

Remarque. La base β étant paire, parmi deux nombres flottants consécutifs, il y en a toujours un dont le dernier chiffre de la mantisse est *pair* et l'autre dont le dernier chiffre de la mantisse est *impair*. Dans ce cas, l'arrondi au plus proche est déterminé de façon *unique*.

Attention cependant, la règle d'approximation ne fonctionne pas pour les nombres de magnitude supérieure à M : en toute rigueur il faut considérer d'abord $\widetilde{fl}(x)$ comme étant la même opération mais à valeur dans

$$\mathbb{F}(\beta, p, e_{min}, +\infty).$$

Dans ces conditions,

- Si $|\widetilde{fl}(x)| > M$ alors $fl(x)$ est le nombre flottant spécial $sign(x)inf$: on dit qu'il y a **"overflow"** (débordement vers l'infini ou dépassement de capacité)
- Sinon $fl(x) = \widetilde{fl}(x)$.

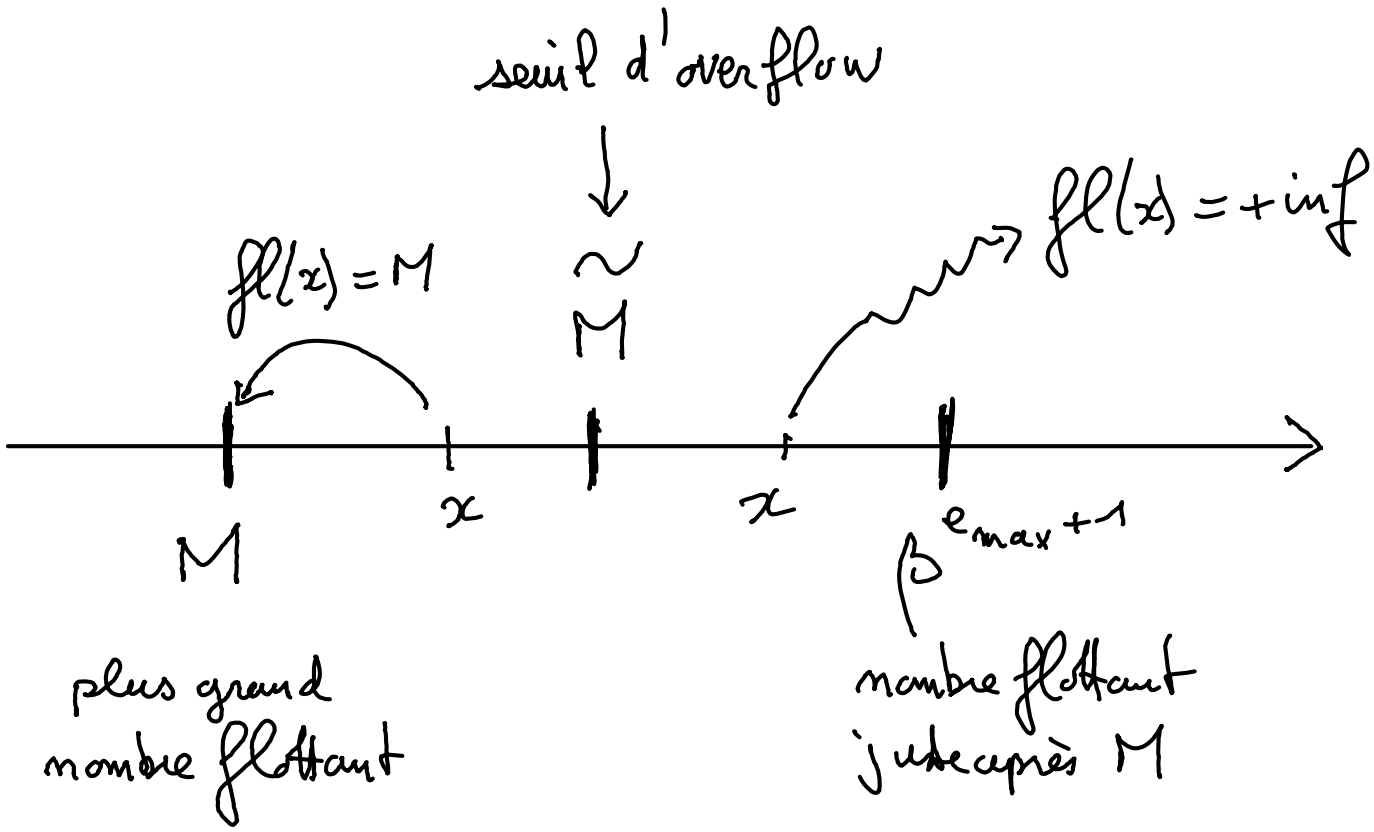
Seuil d'overflow. Avec cette règle d'approximation, M ne correspond pas exactement au seuil d'overflow. Dans l'ensemble $\mathbb{F}(\beta, p, e_{min}, +\infty)$ le nombre flottant *juste après M* est $\beta^{e_{max}+1}$, dont le dernier chiffre de la mantisse est *pair*.

Ainsi le *seuil d'overflow réel* est exactement :

$$\boxed{\tilde{M} := \frac{M + \beta^{e_{max}+1}}{2}}$$

De plus, du fait de la règle d'arrondi, on a $\tilde{fl}(\tilde{M}) = \beta^{e_{max}+1}$ (et non pas M) et donc

$$fl(\tilde{M}) = +inf\,.$$



Seuil d'overflow

Soit x nombre réel.

- Si $|x| < \tilde{M}$, alors x doit être codé par un flottant $fl(x)$ "usuel" (cad qui n'est pas un nombre flottant spécial comme $\pm inf$ ou Nan).
- si $|x| \geq \tilde{M}$ alors $fl(x) = \pm inf$

Exemple. On considère l'ensemble $\mathbb{F}(10, 3, -1, +2)$ pour lequel on a $M = (9, 99)_{10} \cdot 10^2 = 999$ et $\tilde{M} = 999, 5$.

Calculs de $fl(x)$:

- Soit $x = 999, 4548 > M$. On a $\tilde{fl}(x) = M$ et donc $fl(x) = \tilde{fl}(x) = M$ et non pas $+inf$ (on a bien $x < \tilde{M}$).
- Soit $x = 999, 78 > M$. On a $\tilde{fl}(x) = 1000 > M$ et donc $fl(x) = +inf$ (on a bien $x \geq \tilde{M}$).
- Soit $x = 999, 5 = \tilde{M}$. On a donc $fl(x) = +inf$.

Exercice.

On se place dans l'ensemble de flottants $\mathcal{F} = \mathbb{F}(10, 4, -2, 2)$.

1. Calculer les nombres caractéristiques M, m, μ de cet ensemble ainsi que le seuil d'overflow $\tilde{M}$.
2. Donner la valeur des quantités suivantes : $fl(1, 2346)$, $fl(1, 2345)$, $fl(999, 95)$, $fl(999, 94999999)$.

Remarque. Pour calculer le seuil d'overflow $\tilde{M}$ sur une machine, on peut utiliser les *entiers longs*. En python, un entier `int` (par défaut) est représenté par un entier long. Un entier long est uniquement limité par la mémoire de l'ordinateur [2], contrairement aux nombres flottants `float`.

On a le petit script python suivant.

```
beta = 2
p = 53
emax = 1023
M = beta**(emax+1) - beta**(emax+1-p) # M est un entier long
Mtilde = (M+temp)//2        # Mtilde est aussi un entier long
```

Attention : ne pas utiliser la formule $M = (1 - \beta^{-p})\beta^{emax+1}$ car alors M est un `float` et on obtient un *overflow*. De même, utiliser l'opérateur `//` de division euclidienne car avec `/` on aurait aussi un float...

2. Python utilise une base 2^{30} pour représenter un entier par mots. Par ex., pour $x = 10^{20}$, on a $x = 976371285 + 453776706 * 2^{30} + 86 * 2^{60}$: il y a 3 mots utilisés $m[0] = 976371285, m[1] = 453776706, m[2] = 86$

On obtient :

$M =$ 17976931348623157081452742373170435679807056752584499
65989174768031572607800285387605895586327668781715404589535143824642343213268894641827684675467035375169860499105765512820762454900903893289440758685084551339423045832369032229481658085593321233482747978262041447231687381771809192998812504040261841248583 68

$Mtilde =$ 17976931348623158079372897140530341507993413271003782693617377898044496829276475094664901797758720709633028641669288791094655554785194040263065748867150582068190890200070838367627385484581771153176447573027006985557136695962284291481986083493647529271907416844436551070434271155969950809304288017790417449779 2

Erreur relative de l'approximation flottante : l'espilon machine.

Epsilon machine.

On montre que si $|x| \in [m, \tilde{M}[$ alors l'erreur relative est bornée par un nombre **u** appelé *epsilon machine* :

$$\frac{|fl(x) - x|}{|x|} \leq \mathbf{u} := \frac{\beta^{1-p}}{2}.$$

Preuve : Soit $x \in \mathbb{R}$ tel que $|x| \in [m, \tilde{M}[$. Il existe $e \in [\![e_{min}, e_{max}]\!]$ tel que $|x| \in [\beta^e, \beta^{e+1}[$. Nous avons vu que l'incrément entre deux flottants dans la plage $[\beta^e, \beta^{e+1}]$ est égal à β^{1-p+e}, ainsi l'erreur absolue est bornée par la moitié de cette quantité (approximation par le flottant le plus proche) :

$$ea = |x - fl(x)| \leq \frac{1}{2}\beta^{1-p+e}$$

On peut alors borner l'erreur relative en divisant par le plus petit nombre de la plage, d'où :

$$er = \frac{|x - fl(x)|}{|x|} \leq \frac{ea}{\beta^e} = \frac{1}{2}\beta^{1-p} := \mathbf{u}$$

□

Lien avec `ulp` (Unit in last place), le poids du plus petit chiffre de la mantisse d'un flottant : $\texttt{ulp} := \beta^{1-p+e} = 2\mathbf{u}\,\beta^e$

Une façon plus pratique de noter cette erreur relative est d'écrire que (cf TD) :

Approximation flottante et erreur machine

$$fl(x) = x(1+\epsilon), \text{ avec } |\epsilon| \leq \mathbf{u} := \frac{1}{2}\beta^{1-p}$$

Underflow (débordement vers 0).

Lorsque $|x| \in [0, m[$ avec $fl(x) \neq x$, on dit qu'il y a **"underflow"**.

- Sans nombres dénormalisés, en cas d'underflow l'arrondi sera soit 0, soit $\pm m$, selon la règle du flottant le plus proche (avec un seuil valant$\pm m/2$).
- Avec les nombres dénormalisés, l'arrondi sera soit 0, soit le nombre flottant dénormalisé adéquat (on parle alors d'*underflow graduel*). En particulier, avec l'utilisation des nombres dénormalisés, on a

$$fl(x) = \begin{cases} 0 & \text{si } |x| \leq \mu/2, \\ \mu & \text{si } \mu/2 < |x| \leq \mu. \end{cases}$$

Exercice.

On considère à nouveau l'ensemble de flottants $\mathcal{F} = \mathbb{F}(10, 4, -2, 2)$.

Donner les valeurs de $fl(10,2444 \cdot 10^{-1})$, $fl(1,1558 \cdot 10^{-3})$, $fl(5,8912 \cdot 10^{-6})$, $fl(0,4847 \cdot 10^{-5})$

4. Règles des 4 opérations arithmétiques usuelles

Les 4 opérations usuelles sur les flottants doivent respecter le critère suivant (norme IEEE754, cf. plus loin) : *tout se passe comme si le calcul était exact puis arrondi au flottant le plus proche.*

Propriété d'arrondi correct

Soit $\cdot$ l'une des 4 opérations élémentaires $(+,-,\times,/)$ et $\odot$ l'opération machine correspondante. Si x et y sont deux nombres flottants alors :

$$x \odot y = fl(x \cdot y)$$

Par conséquent, en l'absence d'overflow et d'underflow on a :

$$x \odot y = (x \cdot y)(1+\epsilon), \text{ avec } |\epsilon| \leq \mathbf{u}$$

Remarque : la norme sur les flottants impose aussi cette même règle pour certaines fonctions comme par exemple la racine carrée : si x est un flottant et *sqrt* la racine carrée en machine, on doit obtenir :

$$sqrt(x) = fl(\sqrt{x}) \quad \text{d'où : } sqrt(x) = \sqrt{x}(1+\epsilon) \text{ avec } |\epsilon| \leq \mathbf{u}$$

Le théorème de Sterbenz

On peut montrer que la soustraction entre deux flottants x et y de même signe est **exacte** s'ils sont de magnitude voisine.

Théorème de Sterbenz

Soit $\mathbb{F}(\beta, p, e_{min}, e_{max})$ un ensemble de flottants et $x \in \mathbb{F}$, $y \in \mathbb{F}$ de même signe tels que $x/2 \leq y \leq 2x$ (s'ils sont positifs) ou tels que $2x \leq y \leq x/2$ (s'ils sont négatifs) alors [a] :

$$x \ominus y = x - y$$

a. En toute rigueur cette relation est vraie si les nombres dénormalisés non redondants sont utilisés sinon il faut rajouter l'hypothèse qu'il n'y a pas d'underflow.

Preuve. En utilisant le fait que certaines propriétés restent vraies en arithmétique flottante :

- l'addition $\oplus$ est toujours commutative ($x \oplus y = y \oplus x$)
- le moins "unaire" exact : $\ominus x = -x$,
- $\ominus(\ominus y) = \oplus y = y$,

on peut se restreindre (exercice) au cas où les deux flottants x et y sont positifs et tels que $x/2 \leq y \leq x$.

Soient x et y des flottants qui s'écrivent $x = (x_0, x_1 \ldots x_{p-1})_\beta \beta^e$ et $y = (y_0, y_1 \ldots y_{p-1})_\beta \beta^{e'}$ avec $e' \leq e$ (puisque $y \leq x$).

- Si les exposants e et e' sont égaux, il est clair que $x \ominus y = x - y$ puisque le resultat de la soustraction exacte tient sur p chiffres (ou moins).
- Si x n'est pas un flottant normalisé alors $e = e' = e_{min}$ et de même on se trouve dans le cas où les exposants sont égaux.

- Si x est un flottant normalisé alors nécessairement $e' = e$ ou $e' = e - 1$. En effet si $e' < e - 1$ alors :

$$y < \beta^{e-1} = \frac{\beta^e}{\beta} \leq \frac{\beta^e}{2} \text{ car } \beta \geq 2$$

  or on a $x \geq \beta^e$ ce qui contredit $y \geq x/2$. Le seul cas à examiner est donc lorsque $e' = e - 1$ (ce qui implique $y < \beta^e$).

- On suppose donc que $e' = e - 1$. On a, dans ce cas

$$x = (\underbrace{x_0, x_1 \cdots x_{p-1} 0}_{p+1 \text{ chiffres}})_\beta \beta^e \text{ et } y = (\underbrace{0, y_0 \cdots y_{p-1}}_{p+1 \text{ chiffres}})_\beta \beta^e,$$

  de sorte que

$$x - y = (\underbrace{d_0, d_1 \cdots d_p}_{p+1 \text{ chiffres}})_\beta \beta^e$$

Montrons que la mantisse de $x - y$ tient sur p chiffres et donc que $x \ominus y = fl(x - y) = x - y$.

Par l'absurde : on suppose que le résultat de $x - y$ tient sur $p + 1$ chiffres i.e. $d_0 \neq 0$ et $d_p \neq 0$. On a alors

$$x - y \geq (\underbrace{1, 0 \ldots 01}_{p+1 \text{ chiffres}})_\beta \beta^e = \beta^e + \beta^{e-p}$$

Comme $x/2 \leq y$, il vient :

$$x \geq y + \beta^e + \beta^{e-p} \Rightarrow x \geq \tfrac{x}{2} + \beta^e + \beta^{e-p} \iff \tfrac{x}{2} \geq \beta^e + \beta^{e-p}$$

soit finalement $y \geq \beta^e + \beta^{e-p}$ contradiction avec $y < \beta^e$ $\square$.

Remarque importante pour la soustraction $\ominus$.

Phénomène de *cancellation*

Si la soustraction machine $\ominus$ de 2 flottants voisins est *exacte* (théorème de Sterbenz), elle est cependant à l'origine de certains problèmes de perte de précision.

☛ En effet dans l'opération $x \ominus y$, les nombres flottants x et y (de même signe) résultent le plus souvent de calculs comportant des erreurs et **la soustraction amplifiera ces erreurs** (*cancellation*).

Dans le paragraphe "Analyse d'erreurs", on illustrera plus précisément le comportement de la soustraction $\ominus$ et l'analyse de ce phénomène de *cancellation* sera faite en TD.

5. Flottants utilisés sur ordinateurs

La plupart des ordinateurs utilisent la base 2[a]. La base 2 (avec la convention du bit de poids fort *implicite*[b]) donne une précision moyenne meilleure que les autres bases.

Les deux ensembles de flottants les plus utilisés sur ordinateurs sont :

1. $\mathbb{F}(2, 24, -126, 127)$ appelés flottants "simple précision" dont le codage tient sur 4 octets $= 4 \times 8 = 32$ bits, avec :
   - mantisse : 24 bits dont un bit de signe.
   - exposant : 8 bits (dont un bit de signe) ;
     $e_{max} = 2^0 + \cdots + 2^6 = 2^7 - 1 = 127$.
     A priori $e_{min} = -127$ mais le zéro est codé avec des 0 dans la mantisse et un exposant $e_{min} - 1 = -127$, donc $e_{min} = -126$.

a. La base 16 a été parfois utilisée (IBM 360/370 dans les années 70) ou la base 10 (calculettes). Le SETUN fabriqué à Moscou dans les années 1960 utilisait la base 3 avec les digits $-1, 0, +1$.

b. bit=1 car nombre normalisé, donc non codé.

2. $\mathbb{F}(2, 53, -1022, 1023)$ appelés flottants "double précision" tenant sur 8 octets $= 8 \times 8 = 64$ bits (53 bits pour la mantisse, 11 bits pour l'exposant).

Voici leurs nombres caractéristiques (valeurs approchées) :

ensembles de flottants	$\mathbb{F}(2, 24, -126, 127)$	$\mathbb{F}(2, 53, -1022, 1023)$
$\mathbf{u} \simeq$	$5.96 \cdot 10^{-8}$	$1.11 \cdot 10^{-16}$
$m \simeq$	$1.17 \cdot 10^{-38}$	$2.225 \cdot 10^{-308}$
$M \simeq$	$3.40 \cdot 10^{38}$	$1.79 \cdot 10^{308}$

Remarque. Sauf problème d'overflow ou d'underflow toute multiplication/division par une puissance de 2 est exacte avec ces nombres flottants.

Norme IEEE754 (Institute of Electrical and Electronics Engineers). Il s'agit d'une norme pour l'arithmétique flottante (en base 2 et 10) créée en 1985 avec des révisions majeures en 2008 et 2019. Elle définit :

- les formats des nombres flottants : signe, mantisse, exposant, utilisation des nombres dénormalisés ;
- les nombres spéciaux (inf et NaN) ;
- un ensemble d'opérations machine sur les nombres flottants ainsi que le comportement de certaines fonctions (cf. propriétés d'arrondi correct) ;
- 5 modes d'arrondi (arrondi au plus proche par défaut, la plupart du temps [1]).

---

1. Autres arrondis existants : arrondi au plus proche avec variante en cas d'égale distance : vers le plus loin de 0 (vers le haut en valeur absolue) ; arrondi vers $+\infty$ (le plus grand flottant le plus proche) ; arrondi vers $-\infty$ (le plus petit flottant le plus proche) ; arrondi vers 0 (arrondi vers $+\infty$ ou vers$-\infty$ en fonction du signe).

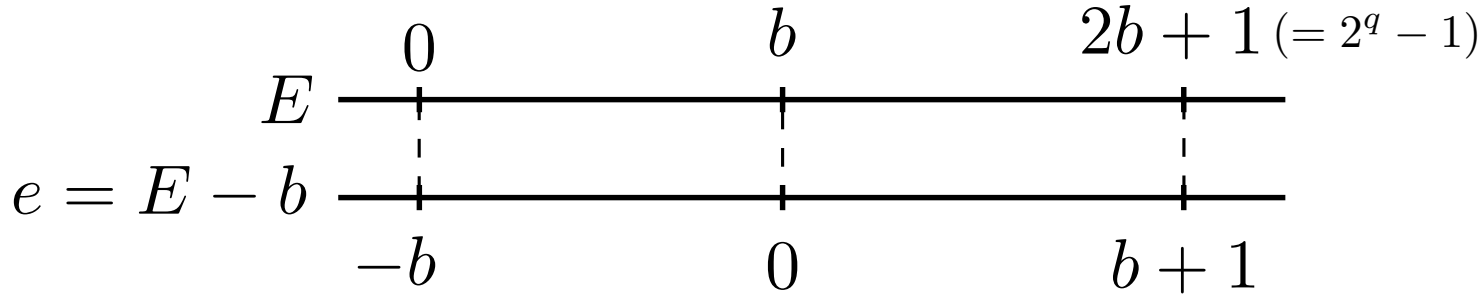
Norme IEEE et exposant biaisé.

La norme IEEE754 fonctionne un peu différemment pour le stockage de l'exposant. Avec la norme IEE754, l'exposant E codé sur q bits, est toujours **positif** et on introduit un biais (translation) b pour retrouver l'exposant e du nombre flottant par

$$e = E - b.$$

On parle alors d'*exposant biaisé* pour E.

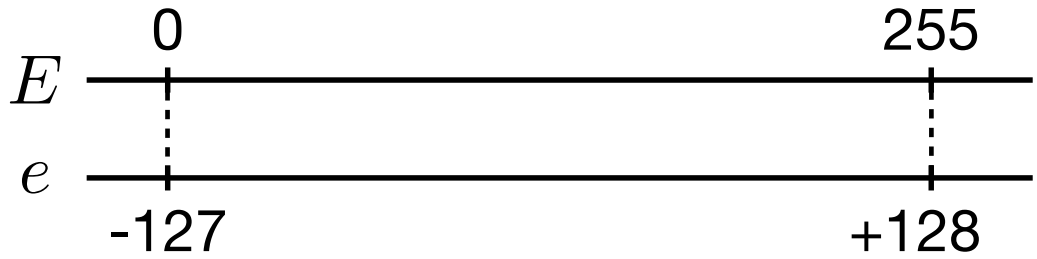
- L'exposant biaisé E varie de 0 à $r = 2^0 + \cdots + 2^{q-1} = 2^q - 1$ avec $r = 2b + 1$ (car r est impair).
- L'exposant réel e varie alors de $e_{min} = -b$ à $e_{max} = b + 1$.



Si q est le nombre de bits dans l'exposant, le biais vaut

$$\boxed{b = 2^{q-1} - 1}$$

Ainsi, en simple précision ($q = 8$), l'exposant biaisé E varie de 0 à $2^8 - 1 = 255$ avec un biais $b = 2^7 - 1 = 127$. On a donc l'exposant réel (non-biaisé) $e = E - 127$ qui varie de -127 à $+128$.



- Le zéro est codé avec un exposant $e_{min} - 1 = -127$ (et des 0 dans la mantisse), donc on a bien $e_{min} = -126$.
- De même, les nombres spéciaux $\pm$ inf et NaN ont un exposant de $e_{max} + 1 = 128$, donc on a bien $e_{max} = 127$. Le format de la norme IEEE permet ainsi de coder les nombres spéciaux inf et NaN.

Remarque. Avec la norme IEEE754, on a toujours $e_{min} = 1 - e_{max}$.

Détermination "pratique" du epsilon machine.

$x \leftarrow 1$
$p \leftarrow 0$
Tant que $1 \oplus x > 1$
    $x \leftarrow x \oslash 2$
    $p \leftarrow p \oplus 1$
Fin Tant que

On montre que (cf. Examen 2020-21) :

$$\boxed{x = \mathbf{u} = 2^{-p}}$$

☛ **Autre définition possible du epsilon machine**.

u est le plus petit nombre flottant qui, ajouté à 1, donne un résultat différent de 1.

6. Propagation et analyse d'erreurs.

Pour faire une analyse de précision de calculs menés avec les flottants, on fait l'hypothèse *qu'il n'y a pas eu d'overflow ni d'underflow*.

a) Calcul d'erreur : un exemple.

On considère a, b et c des flottants et on veut analyser la précision de $x_c := (a \otimes b) \oslash (c \otimes d)$. Avec l'hypothèse faite ci-dessus, on a :

$$\begin{aligned} a \otimes b &= (a \times b)(1+\epsilon_1), \quad |\epsilon_1| \leq \mathbf{u} \\ c \otimes d &= (c \times d)(1+\epsilon_2), \quad |\epsilon_2| \leq \mathbf{u} \\ (a \otimes b) \oslash (c \otimes d) &= \frac{(a \otimes b)}{(c \otimes d)}(1+\epsilon_3), \quad |\epsilon_3| \leq \mathbf{u} \end{aligned}$$

soit finalement :

$$x_c = \underbrace{\frac{a \times b}{c \times d}}_{x} \underbrace{\frac{(1+\epsilon_1)(1+\epsilon_3)}{(1+\epsilon_2)}}_{(1+\delta)} = x(1+\delta).$$

Comme $\mathbf{u} \ll 1$, l'erreur relative $|\delta|$ est "quasi-bornée" par $3\mathbf{u}$.

Une manière élégante de traiter ce problème est d'utiliser le résultat suivant :

Lemme de simplification

Si $|\epsilon_k| \leq \mathbf{u}$, $s_k = \pm 1$ pour $k = 1, \ldots n$ et si $n\mathbf{u} < 1$ alors :

$$\prod_{k=1}^{n} (1 + \epsilon_k)^{s_k} = 1 + \delta_n \quad \text{avec} \quad |\delta_n| \leq \frac{n\mathbf{u}}{1 - n\mathbf{u}}$$

Preuve : cf TD 1.

En utilisant ce lemme dans l'exemple précédent, on obtient donc que :

$$|\delta| \leq \frac{3\mathbf{u}}{1 - 3\mathbf{u}} \simeq 3\mathbf{u}$$

L'erreur relative est donc très petite.

Remarque. En général les bornes obtenues par analyse d'erreurs sont (bien) plus grandes que l'erreur relative "exacte" (il faut pour s'en approcher que chaque opération soit réalisée avec l'erreur maximale et qu'il n'y ait aucune compensation dans les termes en ϵ_k).

De façon générale :

- *En l'absence d'overflow et d'underflow, les opérations machines* $\otimes$ *et* $\oslash$ *se comportent bien au sens où les erreurs commises sur les flottants ne sont pas amplifiées.*
- *De même, l'opération* $\oplus$ *entre flottants de* **même signes** *n'amplifie pas les erreurs.*
- Il n'en est pas de même pour la soustraction $\ominus$ (de deux flottants de *même signe*) qui, bien qu'exacte pour des flottants très proches (Théorème de Sterbenz), amplifie généralement les erreurs commises sur ces flottants quand ceux-ci résultent d'un calcul (cf. TD et exemple suivant).

b) Un exemple de phénomène de *cancellation*.

Le calcul de la fonction

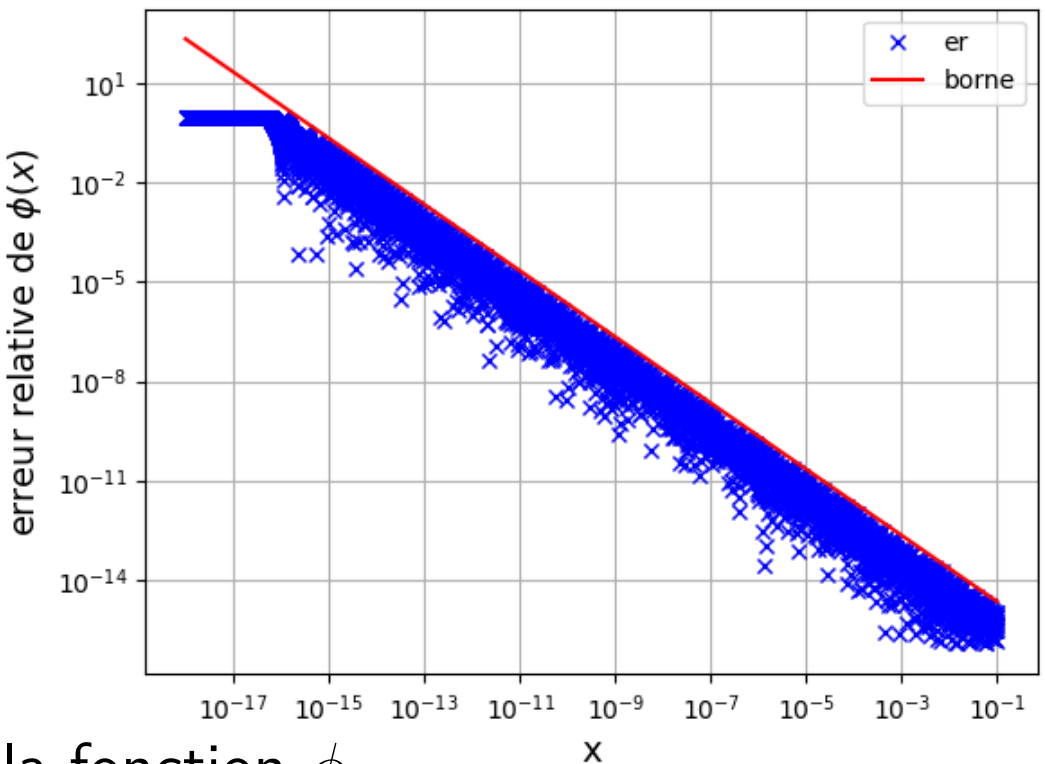
$$\phi(x) = \frac{1}{1+x} - \frac{1}{1-x} \tag{1}$$

avec la formule

$$y1 := (1 \oslash (1 \oplus x)) \ominus (1 \oslash (1 \ominus x)) \tag{2}$$

pour des $|x|$ petits, va conduire à des amplifications d'erreurs importantes.

En effet, pour $|x| \ll 1$, on a $1/(1+x) \simeq 1/(1-x) \simeq 1$ et on se retrouve dans la formule (1) (ou plutôt dans la formule (2)) avec une soustraction de deux quantités flottantes proches.



Remède : écrire différemment la fonction ϕ.

c) Un autre exemple. On considère la suite $(u_n)_{n\geq 0}$ définie par

$$\begin{cases} u_0 = 2,\ u_1 = -4, \\ u_n = 111 - \dfrac{1130}{u_{n-1}} + \dfrac{3000}{u_{n-1}u_{n-2}},\ n \geq 2 \end{cases}$$

Code python :

```
n=30
u=2
v=-4
print("Computation from 2 to %d"%n)

for i in range(2,n+1):
   w = 111. - 1130./v + 3000./(v*u)
   u = v
   v = w
   print("u%d = %1.18g"%(i, v))
```

Computation from 2 to 30	Exact value
u2 = **18.5**	18.5
u3 = **9.37837837837837**95	9.378378378378378 3. . .
u4 = **7.80115273775216**878	7.801152737752161 3833. . .
u5 = **7.154414480975**33339	7.154414480975249 3535. . .
u10 = **6.274438**6627281159	6.274438598216327 9138. . .
u11 = **6.21869**67685821628	6.218695739802397 7883. . .
u15 = **6**.16608655959809937	6.094739439333681 1283. . .
u16 = 7.23502116553493124	6.077722304847242 7363. . .
u17 = 22.062078463525 7934	6.063940322499808 7553. . .
u18 = 78.5755748878722358	6.052721761016152 1934. . .
u19 = 98.349503122165 3591	6.043552110189268 8678. . .
u20 = 99.898569266182 9034	6.036031881081856 7800. . .
u21 = 99.993870988902 7848	6.029847325023901 8567. . .
u22 = 99.99963038728634 5	6.024749652366847 8987. . .
u29 = 99.999999999989342	6.006786093031205 7585. . .
u30 = 99.999999999999289	6.005648688771420 2679. . .

Quelques explications. On a

$$u_n = \frac{\alpha\, 100^{n+1} + \beta\, 6^{n+1} + \gamma\, 5^{n+1}}{\alpha\, 100^{n} + \beta\, 6^{n} + \gamma\, 5^{n}}$$

où α, β, γ dépendent de u_0, u_1.

- Si $\alpha \neq 0$, alors la limite de u_n est 100, sinon (avec $\beta \neq 0$) la limite est 6.
- Avec $u_0 = 2$, $u_1 = -4$, on a $\alpha = 0$ et $\beta/\gamma = -3/4$, donc la limite théorique de u_n est 6.

La valeur de α calculée est très petite mais non-nulle. Cela suffit pour faire converger la suite vers 100 et non pas 6.

L'analyse d'erreur de la (mauvaise) convergence de cette suite est assez compliquée. Il s'agit encore d'un phénomène de propagation et d'amplification des erreurs (cancellation) essentiellement due à la soustraction de 2 flottants proches (et de même signe).