

Calcul en virgule flottante

Cours ENPC - Pratique du calcul scientifique

Motivations

Comment expliquer ce résultat ?

```
0.1 + 0.2 == 0.3  
false
```

Alors que

```
1 + 2 == 3  
true
```

Un indice

```
0.1 + 0.2  
0.30000000000000004
```

Effet d'arrondi

```
round(1.4)  
1.0  
  
round(1.4)+round(1.4)  
2.0  
  
round(1.4+1.4)  
3.0
```

Et celui-ci ?

```
10^19  
-8446744073709551616
```

Alors que

```
10^18  
1000000000000000000  
  
(10.)^19  
1.0e19
```

Comment expliquer l'importance de l'ordre de sommation ?

```
n = 1_000_000  
S_n = sum(1/k for k in 1:n) - log(n)  
  
0.5772161649007153  
  
S'_n = sum(1/k for k in n:-1:1) - log(n)  
  
0.5772161649014986  
  
S_n == S'_n  
false
```

- Dans le premier cas $1 + \frac{1}{2} + \dots + \frac{1}{999999} + \frac{1}{1000000} \Rightarrow$ on somme les gros d'abord
Dans le second cas $\frac{1}{1000000} + \frac{1}{999999} + \dots + \frac{1}{2} + 1 \Rightarrow$ on somme les petits d'abord
Quel calcul est le plus précis ?
- Si ε est la précision machine $1 + \varepsilon/2 \approx 1$ pour l'ordinateur mais $1 + \varepsilon > 1$
alors $(1 + \varepsilon/2) + \varepsilon/2 \approx 1 + \varepsilon/2 \approx 1$ tandis que $1 + (\varepsilon/2 + \varepsilon/2) = 1 + \varepsilon$

Représentation binaire des nombres réels

Définition de la décomposition en base β

Soit $x \in \mathbb{R}$ et $\beta \in \mathbb{N}^*$. On note

$$\pm(a_{-n}a_{-n+1} \dots a_{-1}a_0.a_1a_2 \dots)_\beta$$

la représentation de x en base β si

$$x = \pm \sum_{k=-n}^{+\infty} a_k \beta^{-k}, \quad a_k \in \{0, \dots, \beta - 1\}$$

- $\beta = 2 \rightarrow$ écriture **binaire**, $a_k \in \{0, 1\}$ est un **bit**
- $\beta = 10 \rightarrow$ écriture **décimale**, $a_k \in \{0, 1, \dots, 9\}$ est un **chiffre** (digit en anglais)
- $\beta = 16 \rightarrow$ écriture **hexadécimale**, $a_k \in \{0, 1, \dots, 9, A, B, \dots, F\}$

Exemples

$$(10)_\beta = 1 \times \beta^1 + 0 \times \beta^0 = \beta$$

$$(FF)_{16} = 15 \times 16^1 + 15 \times 16^0 = 15 \times 17 = 16^2 - 1 = (255)_{10}$$

Représentation périodique

$$(a_0.a_1a_2 \dots \overline{a_k \dots a_{k+p}})_\beta = (a_0.a_1a_2 \dots \underbrace{a_k \dots a_{k+p}}_{\text{période}} \underbrace{a_k \dots a_{k+p}}_{\text{période}} \underbrace{a_k \dots a_{k+p}}_{\text{période}} \dots)_\beta$$

Conversion binaire \rightarrow décimal

Application simple de la définition $x = \pm \sum_{k=-n}^{+\infty} a_k 2^{-k}$

Exercice $(0.\overline{10})_2 = (0.1010101010 \dots)_2$ en base 10

$$(0.\overline{10})_2 = \sum_{k=0}^{+\infty} 2^{-2k-1} = \frac{1}{2} \sum_{k=0}^{+\infty} (2^{-2})^k = \frac{1}{2} \frac{1}{1-\frac{1}{4}} = \frac{2}{3} = (0.\overline{6})_{10}$$

Arithmétique binaire

- Les réflexes appris au primaire fonctionnent $(1)_2 + (1)_2 = (10)_2$ (retenue)
- La multiplication par 2 est facile car $2 = (10)_2$

$$2 \times (a_{-n}a_{-n+1} \dots a_{-1}a_0.a_1a_2 \dots)_2 = (a_{-n}a_{-n+1} \dots a_{-1}a_0a_1.a_2 \dots)_2$$

Conversion décimal → binaire

Cas des nombres $0 \leq x < 1$ soit $x = (0.b_1b_2 \dots b_k \dots)_2$

- $b_0 = 0$
- $2x = (b_1.b_2 \dots b_k \dots)_2 \Rightarrow \begin{cases} b_1 = 1 & \text{si } 2x \geq 1 \\ b_1 = 0 & \text{sinon} \end{cases}$
- $x \leftarrow 2x - b_1$ et on recommence pour $b_2 \dots$

Exercice : décomposer $(0.1)_{10}$, $(0.2)_{10}$ et $(0.3)_{10}$ en binaire

x	i	b_i
0.1	1	0
0.2	2	0
0.4	3	0
0.8	4	1
0.6	5	1
0.2	6	0
0.4	7	0
0.8	8	1
0.6	9	1

$(0.1)_{10} = (0.00011)_2$

$(0.2)_{10} = 2 \times (0.1)_{10} = (0.\overline{0011})_2$

x	i	b_i
0.3	1	0
0.6	2	1
0.2	3	0
0.4	4	0
0.8	5	1
0.6	6	1
0.2	7	0
0.4	8	0
0.8	9	1

$(0.3)_{10} = (0.010011)_2$

Exercices

- implémenter l'algorithme décimal → binaire en [Julia](#) pour des nombres $0 \leq x < 1$
- établir l'algorithme décimal → binaire pour des entiers $n \in \mathbb{N}$ et l'implémenter en [Julia](#)

Les ensembles de réels à virgule flottante

Norme (IEEE-754, 2008)

Ensemble des réels à virgule flottante défini par

$$\mathbf{F}(p, E_{\min}, E_{\max}) = \left\{ (-1)^s 2^E (b_0.b_1b_2 \dots b_{p-1})_2 : s \in \{0, 1\}, b_i \in \{0, 1\} \text{ and } E_{\min} \leq E \leq E_{\max} \right\}$$

où E est l'exposant et $(b_0.b_1b_2 \dots b_{p-1})_2$ la mantisse.

Cet ensemble est décomposé en

$$\mathbf{F}(p, E_{\min}, E_{\max}) = \left\{ (-1)^s 2^E (\mathbf{1}.b_1b_2 \dots b_{p-1})_2 : s \in \{0, 1\}, b_i \in \{0, 1\} \text{ et } E_{\min} \leq E \leq E_{\max} \right\} \cup \underbrace{\left\{ (-1)^s 2^{E_{\min}} (\mathbf{0}.b_1b_2 \dots b_{p-1})_2 : s \in \{0, 1\}, b_i \in \{0, 1\} \right\}}_{\text{nombres dénormalisés}}$$

Encodage sur n bits avec $n = 1 + (p - 1) + n_E$

- 1 bit pour le signe ($s = 0$ pour $+$ et $s = 1$ pour $-$)
- $p - 1$ bits pour la mantisse
- n_E bits pour l'exposant
- Codage : $\underbrace{se_0 \dots e_{n_E-1}}_{\text{exposant}} \underbrace{b_1 \dots b_{p-1}}_{\text{mantisse}}$

- On pose $E_{\max} = 2^{n_E-1} - 1$ et $E_{\min} = -2^{n_E-1} + 2$ soit un nombre possible d'exposants de $E_{\max} - E_{\min} + 1 = 2^{n_E} - 2$
- Les n_E bits déterminent un entier e tel que $0 \leq e \leq 2^{n_E} - 1$
 - $e = 0 \Rightarrow E = E_{\min}$ et (nombre dénormalisé) $x = (-1)^s 2^{E_{\min}} (0.b_1b_2 \dots b_{p-1})_2$ (y compris 0)
 - $1 \leq e \leq 2^{n_E} - 2 \Rightarrow E = E_{\min} + e - 1$ et (nombre normalisé) $x = (-1)^s 2^E (1.b_1b_2 \dots b_{p-1})_2$
 - $e = 2^{n_E} - 1 \Rightarrow \begin{cases} \text{Inf} & s = 0 \text{ et } b_1b_2 \dots b_{p-1} = 00 \dots 0 \\ -\text{Inf} & s = 1 \text{ et } b_1b_2 \dots b_{p-1} = 00 \dots 0 \\ \text{NaN} & \text{sinon} \end{cases}$

Les formats de réels à virgule flottante

Erreur relative et epsilon machine

- $\mathbf{F}(p, E_{\min}, E_{\max}) = \left\{ (-1)^s 2^E (\textcolor{red}{1}.b_1 b_2 \dots b_{p-1})_2 : s \in \{0, 1\}, b_i \in \{0, 1\} \text{ et } E_{\min} \leq E \leq E_{\max} \right\} \cup \left\{ (-1)^s 2^{\textcolor{red}{E}_{\min}} (\textcolor{red}{0}.b_1 b_2 \dots b_{p-1})_2 : s \in \{0, 1\}, b_i \in \{0, 1\} \right\}$
- On définit le epsilon machine relatif à l'ensemble $\mathbf{F}(p, E_{\min}, E_{\max})$ par

$\varepsilon_M = 2^{-(p-1)}$
- Si $|x| \in [2^{E_{\min}}, 2^{E_{\max}} (2 - \varepsilon_M)]$, alors $\min_{\hat{x} \in \mathbf{F}(p, E_{\min}, E_{\max})} \frac{|x - \hat{x}|}{|x|} \leq \frac{1}{2} 2^{-(p-1)} = \frac{\varepsilon_M}{2}$ i.e. on peut toujours trouver un représentant de x dans $\mathbf{F}(p, E_{\min}, E_{\max})$ à $\frac{\varepsilon_M}{2}$ près en relatif.
- On définit alors l'arrondi de x comme le \hat{x} vérifiant ce minimum et on note $\text{fl} : \mathbb{R} \rightarrow \mathbf{F}(p, E_{\min}, E_{\max}) ; x \mapsto \text{fl}(x) = \arg \min_{\hat{x} \in \mathbf{F}(p, E_{\min}, E_{\max})} \frac{|x - \hat{x}|}{|x|}$

Formats de réels selon la norme (IEEE-754, 2008) et correspondances Julia

	Demi-précision	Simple précision	Double précision
p	11	24	53
n_E	5	8	11
E_{\min}	-14	-126	-1022
E_{\max}	15	127	1023
ε_M	$2^{-10} = 0.000977$	$2^{-23} = 1.192092910^{-7}$	$2^{-52} = 2.22044604925031310^{-16}$
type Julia	Float16	Float32	Float64

Sous Julia, `typeof(x)` renvoie le type de `x` et `bitstring(x)` renvoie “ $\underbrace{se_0 \dots e_{n_E-1}}_{\text{exposant}} \underbrace{b_1 \dots b_{p-1}}_{\text{mantisse}}$ ”

Décomposition des réels à virgule flottante

- Soit $x \in [2^{E_{\min}}, 2^{E_{\max}} (2 - \varepsilon_M)]$ (quitte à travailler avec $-x$ si $x < 0$)
- $E = \lfloor \log_2(x) \rfloor \Rightarrow x = 2^E y$ avec $y \in [1, 2[$
- Décomposition de $y - 1 = (0.b_1 b_2 \dots b_{p-1})_2$
- $x = 2^E (1.b_1 b_2 \dots b_{p-1})_2$

Opérations sur $\mathbf{F}(p, E_{\min}, E_{\max})$

Définition des opérations élémentaires

- Opérations dans $\mathbf{F} = \mathbf{F}(p, E_{\min}, E_{\max})$
 $\forall \circ \in \{+, -, \times, /\}, \quad \hat{\circ} : \mathbf{F} \times \mathbf{F} \rightarrow \mathbf{F} ; (x, y) \mapsto \text{fl}(x \circ y)$
- Extension à \mathbb{R}
 $\forall \circ \in \{+, -, \times, /\}, \quad \hat{\circ} : \mathbb{R} \times \mathbb{R} \rightarrow \mathbf{F} ; (x, y) \mapsto \text{fl}(\text{fl}(x) \circ \text{fl}(y))$
- Remarque importante : $\hat{\circ}$ n'est pas associative
$$(x \hat{+} y) \hat{+} z \neq x \hat{+} (y \hat{+} z)$$

Exemples

```
ε = eps(Float64)
(1+ε/2)+ε/2
1.0

1+(ε/2+ε/2)
1.0000000000000002

2 + ε
2.0

2 + 1.00001ε
2.0000000000000004
```

Exercice : montrer que $0.1 \hat{+} 0.2 \neq 0.3$ dans **Float16**

Rappel : $(0.1)_{10} = (0.00011)_2, (0.2)_{10} = 2 \times (0.1)_{10} = (0.0011)_2$ et $(0.3)_{10} = (0.010011)_2$

$(0.1)_{10} = 2^{-4}(\underbrace{1.10011001100110011\dots}_\text{10 bits})_2 \Rightarrow \text{fl}_{16}(0.1) = 2^{-4}(1.1001100110)_2$

```
x = Float16(0.1) ; exponent(x), bitstring(x)[7:end]
(-4, "1001100110")
```

$(0.2)_{10} = 2^{-3}(\underbrace{1.10011001100110011\dots}_\text{10 bits})_2 \Rightarrow \text{fl}_{16}(0.2) = 2^{-3}(1.1001100110)_2$

```
x = Float16(0.2) ; exponent(x), bitstring(x)[7:end]
(-3, "1001100110")
```

$(0.3)_{10} = 2^{-2}(\underbrace{1.00110011001100110\dots}_\text{10 bits})_2 \Rightarrow \text{fl}_{16}(0.3) = 2^{-2}(1.0011001101)_2$

```
x = Float16(0.3) ; exponent(x), bitstring(x)[7:end]
(-2, "0011001101")
```

$0.1 \hat{+} 0.2 = 2^{-4} ((1.1001100110)_2 + (11.0011001100)_2)$

$(0.3)_{10} = 2^{-4}(100.1100110100)_2$

$$\begin{array}{r} 1.1001100110 \\ + 11.0011001100 \\ \hline = 100.1100110010 \\ \neq 100.1100110100 \end{array}$$

Encodage des entiers signés

- Les types principaux d'entiers sous **Julia** sont **Int16**, **Int32**, **Int64** (**Int64** par défaut)
- Un entier n est codé sous la forme de p bits : $b_{p-1} b_{p-2} \dots b_0$
- L'algorithme de correspondance sous **Julia** est le **complément à deux**

$$n = -b_{p-1}2^{p-1} + \sum_{i=0}^{p-2} b_i 2^i$$

- Les $p - 1$ premiers bits b_0, \dots, b_{p-2} , déterminent de manière unique un entier entre 0 et $N_{\max} = 2^{p-1} - 1$
- Le dernier bit b_{p-1} opère ou non une translation dans les négatifs de -2^{p-1} si bien que $N_{\min} = -2^{p-1}$

Examples

$$1 = -0 \times 2^{p-1} + \sum_{i=1}^{p-2} 0 \times 2^i + 1 \times 2^0$$

```
bitstring(1)
```

[illegible]

$$-1 = -2^{p-1} + 2^{p-1} - 1 = -1 \times 2^{p-1} + \sum_{i=0}^{p-2} 1 \times 2^i$$

```
bitstring(-1)
```

[illegible]

Pas de Inf ou NaN mais comportement cyclique $N_{\max} + 1 = 2^{p-1} \rightarrow N_{\min}$ et $N_{\min} - 1 = -2^{p-1} - 1 \rightarrow N_{\max}$

 2^{63}
$$-2^{63}-1$$

-9223372036854775808

9223372036854775807

$$2(N_{\max} + 1) = 2^p \rightarrow N_{\min} + N_{\max} + 1 = 0$$

 2^{64}

0

Algorithme de sommation de Kahan

Présentation de l'algorithme de sommation de Kahan

```
function KAHANSUM(x)
    Σ = 0. // somme, on veut calculer  $\sum_{i=1}^n x_i$ 
    c = 0. // variable de compensation
    for i = 1 to LENGTH(x) do
        y = x[i] - c // incrément compensé de l'erreur précédente
        Σ' = Σ + y //  $\Sigma + y \Rightarrow \exists$  erreur pour  $|y| \ll |\Sigma|$ 
        δΣ = Σ' - Σ // évalue la part de y correctement intégrée
        c = δΣ - y // évalue la petite part de y non intégrée
        Σ = Σ' // met à jour la somme
    end for
    return Σ // retourne la somme
end function
```

Exercice

1. Implémenter une fonction “naïve” `MySum` faisant la somme des composantes d'un vecteur `X`
2. Implémenter l'algorithme `KahanSum`
3. Choisir un type `T`, un entier `n` et construire le vecteur `X=[one(T),eps(T).2,...,eps(T)/2]` où `eps(T)/2` est répété `n - 1` fois
4. Comparer les sommes obtenues en utilisant les fonctions `MySum`, `sum` et `KahanSum` sur `X` ainsi que sur `view(X,n:-1:1)`
5. Construire un vecteur aléatoire `Y=rand(T,n)` ainsi qu'une version ordonnée `Ỹ=sort(Y)`
6. Comparer les sommes obtenues sur `Y` et `Ỹ` avec les différents algorithmes. On pourra notamment se servir de la bibliothèque `Xsum.jl` comme référence.

Code

```
n = 10_000_000 ; T = Float64 ; ε = eps(T)
X = fill(ε/2,n-1) ; pushfirst!(X,one(T)) ;

1 + ((n - 1) * ε) / 2 = 1.000000001110223
MySum(X) = 1.0

MySum(view(X, n:-1:1)) = 1.000000001110223
sum(X) = 1.0000000011102188
sum(view(X, n:-1:1)) = 1.0000000011102228
KahanSum(X) = 1.000000001110223

KahanSum(view(X, n:-1:1)) = 1.000000001110223

using Xsum
Y = rand(T,n) ; Ỹ = sort(Y)
Σref = xsum(Y) ;

xsum(Ỹ) - Σref = 0.0
MySum(Y) - Σref = 1.9837170839309692e-7
MySum(Ỹ) - Σref = -1.862645149230957e-9
sum(Y) - Σref = 0.0
sum(Ỹ) - Σref = 0.0
KahanSum(Y) - Σref = 0.0

KahanSum(Ỹ) - Σref = 0.0
```

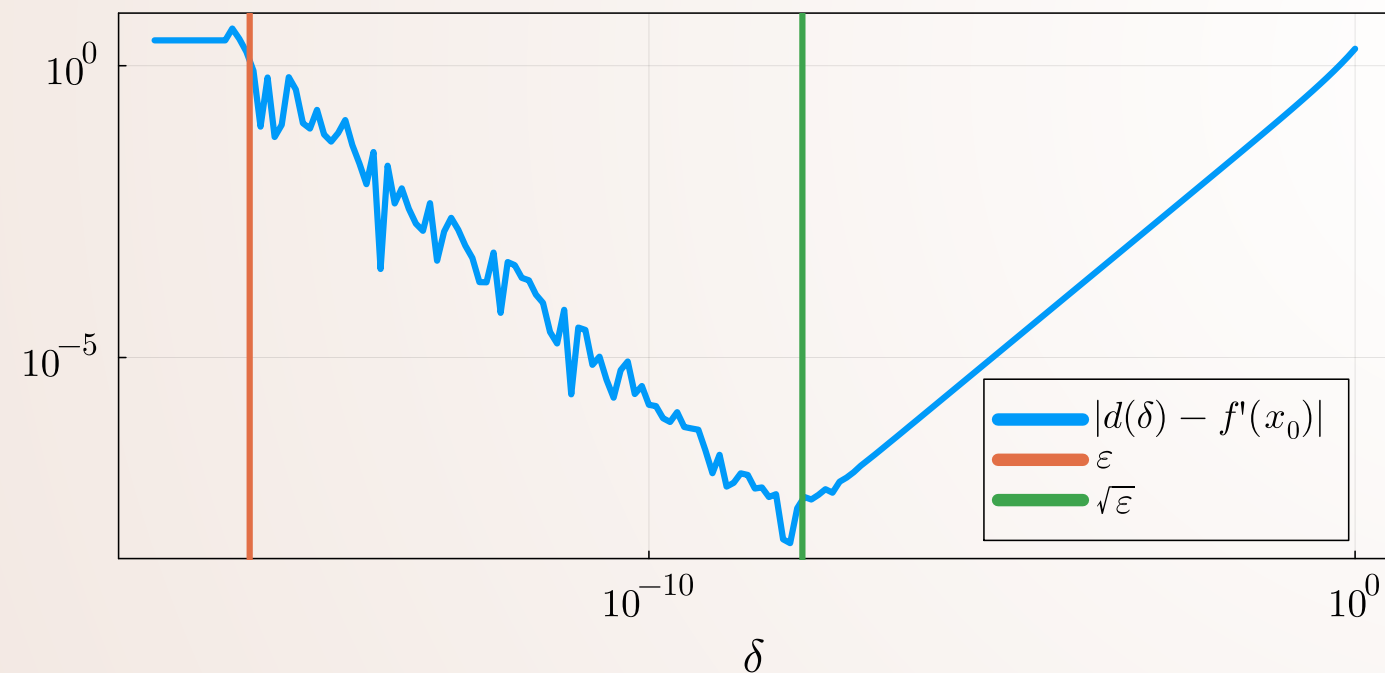
Différentiation numérique

Exercice

On pose $f(x) = \exp x$, $x_0 = 1$ et $d(\delta) = \frac{f(x_0+\delta)-f(x_0)}{\delta}$.

Tracer l'erreur commise entre $d(\delta)$ et $f'(x_0) = \exp x_0$ en fonction de $\delta \in [10^{-17}, 10^0]$ sur un graphe log-log en repérant les droites verticales d'équations $\delta = \varepsilon_M$ et $\delta = \sqrt{\varepsilon_M}$.

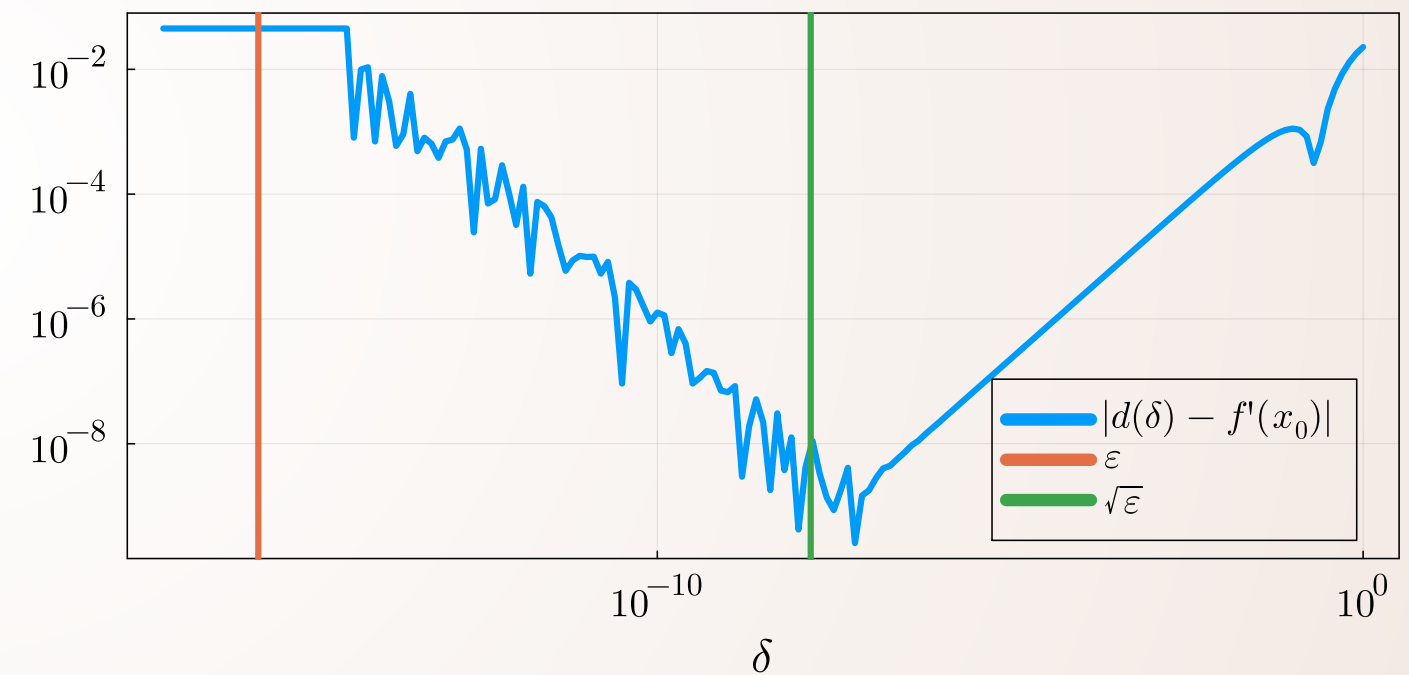
► Code



Exercice

Tester avec d'autres fonctions en utilisant la bibliothèque [Zygote.jl](#) pour la différentiation automatique.

► Code



References

IEEE-754, 2008. IEEE Std 754 (Revision of IEEE Std 754-1985), IEEE Standard for Floating-Point Arithmetic (American {{National Standard}} No. 754-2008). The Institute of Electrical and Electronics Engineers, Inc, 345 East 47th Street, New York, NY 10017, USA.