

Calcul Scientifique

Ecole Hassania des Travaux Publics

Salem Nafiri

Introduction à l'arithmétique de la machine et à l'analyse de l'erreur



Introduction à l'arithmétique de la machine

Introduction

Nous avons tous appris, à l'école primaire, à effectuer des additions, des multiplications et des divisions. Pour cette raison, la plupart des utilisateurs d'ordinateurs ne se sont jamais demandé comment leurs machines effectuaient les opérations arithmétiques, pensant que les mêmes méthodes (ou tout au moins de légères variantes, adaptées par exemple à l'emploi de la base) étaient utilisées, ce qui est nous le verrons souvent faux.

Introduction

En ce qui concerne l'évaluation de fonctions plus complexes (sinus, cosinus, logarithme, exponentielle. . .), nombre d'entre nous se sont demandés, au lycée, quelles étaient les méthodes utilisées par nos calculatrices de poche. Cette interrogation a en général pris fin lorsque nous avons appris ce qu'est un développement de Taylor . Nous avons alors cru savoir.

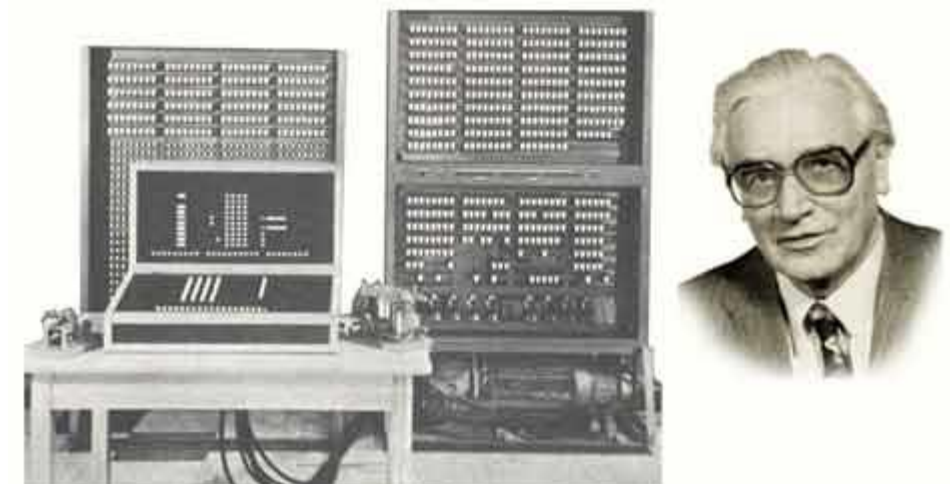
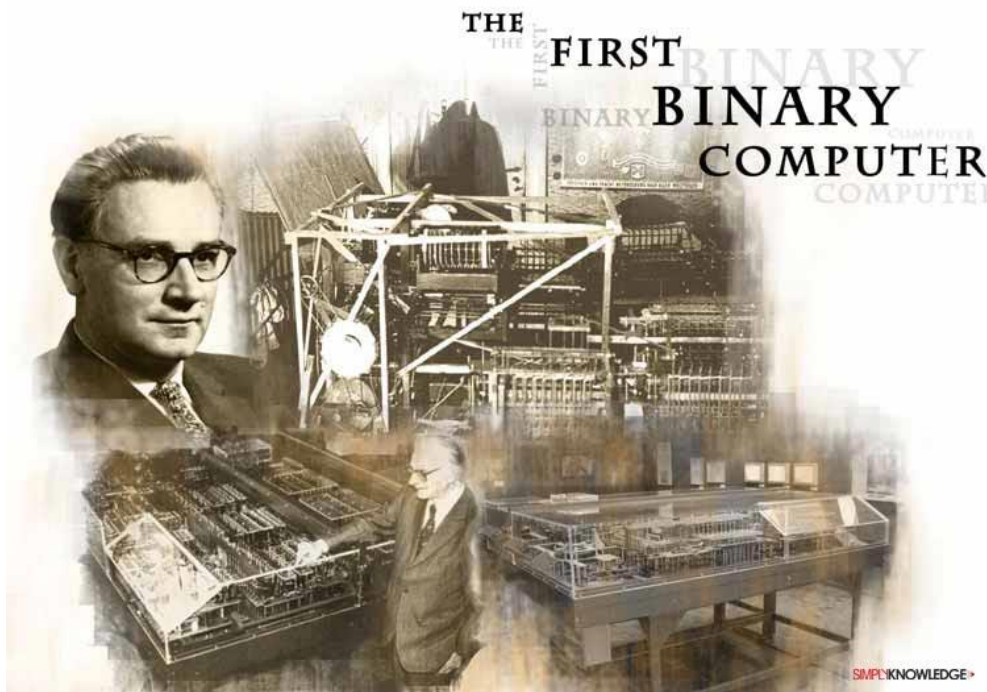
Un des buts de cet ouvrage est de montrer que les algorithmes utilisés en arithmétique des ordinateurs sont souvent plus complexes qu'on ne croit, et qu'ils sont en général très différents de ceux que l'on utilise pour faire des calculs « à la main ».

La machine arithmétique

- 1642: **Blaise Pascal** présente une machine à additionner, qui sera appelée plus tard la Pascaline.
- 1673: **Leibniz** reprend le flambeau et se donne pour objectif de réaliser une machine à multiplier qui libèrera les savants des tâches de calcul répétitives. Il travaillera toute sa vie à ce projet. Malgré les efforts importants de Leibniz, l'argent et le temps qu'il a consacrés à ce projet, la machine ne fonctionne pas correctement.
- 1820: **Thomas de Colmar** réalise une machine fiable sur la base des idées de Leibniz : son « arithmomètre » sera la première machine à calculer de fabrication industrielle.

Premier ordinateur flottant

Le premier calculateur électromécanique programmable binaire, **Z3**, avec une **arithmétique flottante** date de 1941 et a été créé par **Konrad Zuse (1910-1995, Allemand)**.



Représentation binaire de la machine

Le développement de l'informatique et plus généralement de ce qu'on appelle « **le numérique** », est étroitement lié à l'arithmétique. Lorsqu'on a besoin de traiter des informations, de faire fonctionner des documents multimédias (textes, sons, images) sur des machines, il est souvent nécessaire de les coder. Toute information peut être codée en utilisant des suites formées uniquement des deux symboles 0 et 1. On parle de **représentation binaire** ...

Calcul simple

vraiment simple...

Calcul de $\sum_{i=1}^n x$

Avec $n = 1000$ et

- $x = 0.5$
- $x = 0.25$
- $x = 0.1$
- $x = 0.7$

Un peu de C++

```
int main() {  
    float x;  
    cout<<"Entrez_la_valeur_de_x"<<endl;  
    cin>>x;  
    float res = 0.0;  
    for (int i=0;i<1000;i++){  
        res+=x;  
    }  
    cout<<"Somme_des_x_(1000_fois)_"<<setprecision(10)<<res<<endl;  
    return 0;  
}
```

Calcul en simple précision:

Calculons

$$\sum_{i=1}^{1000} 0.5 = 500$$

$$\sum_{i=1}^{1000} 0.25 = 250$$

$$\sum_{i=1}^{1000} 0.1 = 99.999046$$

$$\sum_{i=1}^{1000} 0.7 = 700.006958$$



Ne faites pas aveuglement confiance à votre ordinateur !!!

Et avec des double precision ?

On remplace les float par des double et ...

$$\sum_{i=1}^{1000} 0.1 = 99.99999999999985931253$$

On repousse simplement le problème !

Question 1

- Pourquoi les résultats avec $x = 0.5$ et $x = 0.25$ sont-ils justes et ceux avec $x = 0.7$ et $x = 0.1$ faux ?
- 0.7 et 0.1 sont-ils plus dangereux que 0.5 et 0.25 ?



Retour sur la question 1

- Pourquoi le calcul de $\sum_{i=1}^{100} 0.5$ est-il **juste** ?
- Pourquoi le calcul de $\sum_{i=1}^{100} 0.1$ est-il **faux** ?

1^{ers} éléments de réponse

Tous les nombres ne sont pas représentables

Réponse:

- 0.5 est parfaitement représentable (de même pour 0.25).
- 0.1 n'est pas parfaitement représentable (de même pour 0.7).

Arithmétique de la machine

La capacité mémoire d'un ordinateur est par construction finie. Il est donc nécessaire de représenter les nombres réels sous forme approchée.

Deux types de représentations existent:

- Représentation en virgule fixe
- Représentation en virgule flottante

Représentation des nombres réels

Codage en virgule fixe

Ce codage peut s'écrire sous la forme :

[partie entière en binaire , partie décimale en binaire]

- la partie entière d'un nombre se traduit par des puissances positives de 2.

$$25 = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

- La partie décimale va se traduire par des puissances négatives de 2.


$$0.375 = 0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3}$$

Exemple : $25,375 \Rightarrow 11001,011$

Exemple

Exemple : 0.625_{10}

0.625	×	2	=	1.250	poids binaire	$1(\times 2^{-1})$
0.250	×	2	=	0.500	poids binaire	$0(\times 2^{-2})$
0.500	×	2	=	1.000	poids binaire	$1(\times 2^{-3})$



Quand il ne reste plus de partie fractionnaire on s'arrête.

Ainsi : $0.625_{10} = .101_2$

Cette méthode a l'inconvénient d'utiliser beaucoup de bits après la virgule si le nombre à coder comporte beaucoup de chiffre après la virgule.

Exercice

Exercice : Convertir en binaire 307.18_{10}

$307/2$	$=$	153	reste	1
$153/2$	$=$	76	reste	1
$76/2$	$=$	38	reste	0
$38/2$	$=$	19	reste	0
$19/2$	$=$	9	reste	1
$9/2$	$=$	4	reste	1
$4/2$	$=$	2	reste	0
$2/2$	$=$	1	reste	0
$1/2$	$=$	0	reste	1



0.18×2	$=$	0.36	\rightarrow	0
0.36×2	$=$	0.72	\rightarrow	0
0.72×2	$=$	1.44	\rightarrow	1
0.44×2	$=$	0.88	\rightarrow	0
0.88×2	$=$	1.76	\rightarrow	1
0.76×2	$=$	1.52	\rightarrow	1



$$307.18_{10} = 100110011.001011..._2$$

Nombre binaire à virgule flottante

Afin de représenter les très grands ou très petits nombres, nous utilisons généralement une méthode assez pratique appelée **notation scientifique**.

Rappel : $125.68 = 1.2568 \times 10^2$

$$(-1)^{\text{signe}} . \textit{Mantisse} . 10^{\pm \textit{Exposant}}$$

En binaire:

$$(-1)^{\text{signe}} . \textit{Mantisse} . 2^{\pm \textit{Exposant}}$$

La norme IEEE 754 normalise cette représentation. Elle distingue 2 représentations, celle en simple précision (sur 32 bits), et celle en double précision (sur 64 bits).

Virgule fixe, virgule flottante

Représentation des réels en virgule fixe

Lorsque le nombre de chiffres après la virgule est **fixe** quelque soit le réel.

Peu pratique

- Masse d'un électron $0, \overbrace{0 \dots 0}^{28 \text{ décimales}} 9$ grammes
- Masse du soleil : $20, \overbrace{0 \dots 0}^{33 \text{ zéros}} 0$ grammes

Virgule fixe, virgule flottante

Représentation des réels en virgule flottante

Exemple : 37.5 peut s'écrire (en base 10) :

- 37500.10^{-3}
- $0.0375.10^3$
- ...
- $0.375.10^2$ (représentation normalisée)

Représentation normalisée (base 10)

- La partie entière est systématiquement nulle
- Tous les chiffres significatifs sont à droite de la virgule
- La virgule n'a pas besoin d'être représentée
- La mantisse est inférieure à 1
- L'exposant est un entier (positif ou négatif)

Virgule flottante

Et en base 2 ?

Notation normalisée :

Exemple : $11011 = (0, 11011).2^{101}$

- La partie entière est systématiquement nulle
- Tous les chiffres significatifs sont à droite de la virgule
- La virgule n'a pas besoin d'être représentée
- La mantisse est inférieure à 1
- L'exposant est un entier (positif ou négatif)

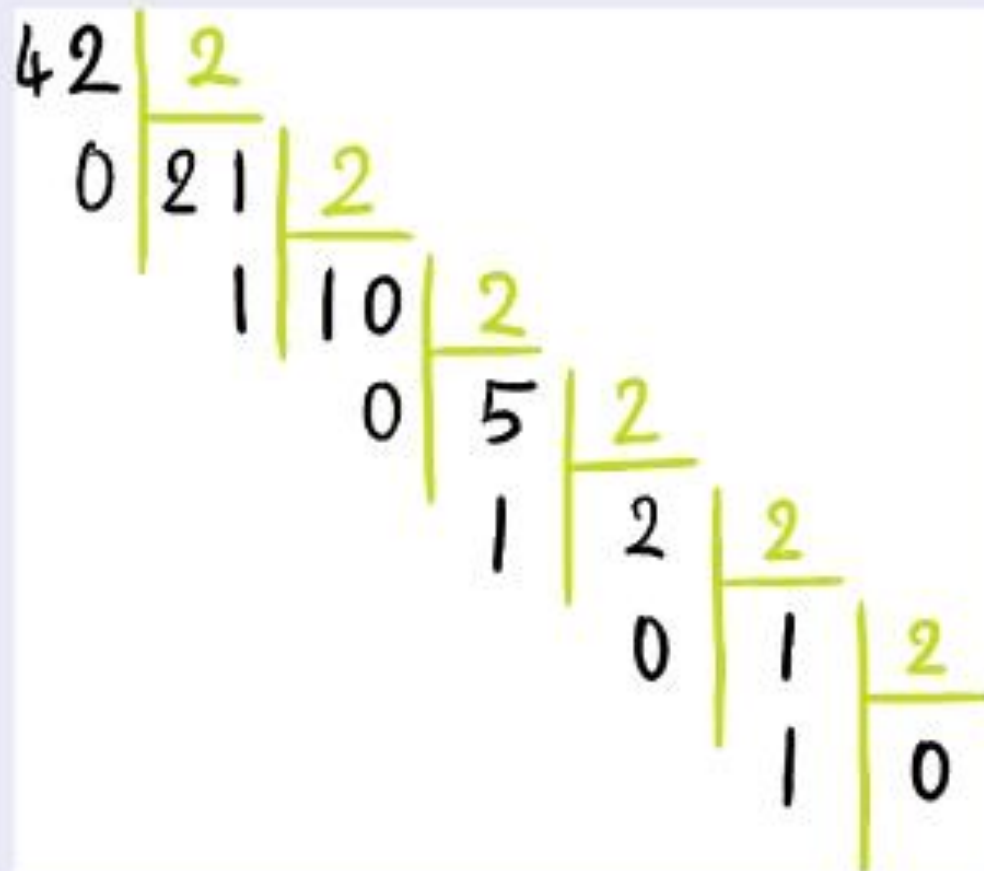
De la base 10 à la base 2

Un entier

- $9_{10} = 1001_2$ car $9 = 8 + 1 = 1.2^3 + 0.2^2 + 0.2^1 + 1.2^0$
- $176_{10} = 10110000_2$ car $176 = 128 + 32 + 16 = 1.2^7 + 1.2^5 + 1.2^4$

En pratique :

$$42_{10} = 101010_2$$



De la base 10 à la base binaire

On considère l'exemple suivant:

$$1993_{10} = ?_2$$

[illegible]

De la base 10 à la base 2

Un décimal

Le principe est le même sauf qu'après la virgule, on divise par $\frac{1}{2}$ (ou on multiplie par 2)

- $0.375_{10} = 0.011_2$ car

- ▶ $0.375 \times 2 = 0.75$

- ▶ $0.75 \times 2 = 1.5$

- ▶ $0.5 \times 2 = 1.0$

- $0.1_{10} = 0.000[1100]_2$ car

- ▶ $0.1 \times 2 = 0.2$

- ▶ $0.2 \times 2 = 0.4$

- ▶ $0.4 \times 2 = 0.8$

- ▶ $0.8 \times 2 = 1.6$

- ▶ $0.6 \times 2 = 1.2$

- ▶ $0.2 \times 2 = 0.4$

- ▶ $0.4 \times 2 = 0.8$

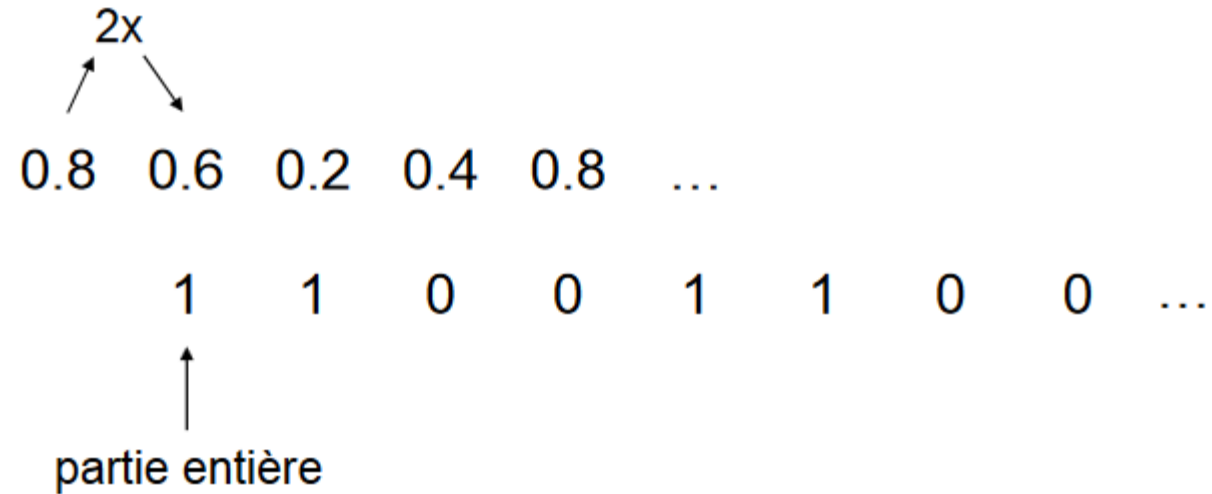
- ▶ ...

- $9.375_{10} = 1001.011_2$

De la base 10 à la base binaire

On considère l'exemple suivant:

$$1993.8_{10} = ?_2$$



D'où :

$$1993.8_{10} = 1111100100, \overline{11001100}_2$$

Contraintes

- En base b , on ne peut représenter exactement que les nombres fractionnaires de la forme $\frac{X}{b^k}$ où X et k sont des entiers
- En base 10 : $\frac{X}{10^k}$
- En base 2 : $\frac{X}{2^k}$

Exemples :

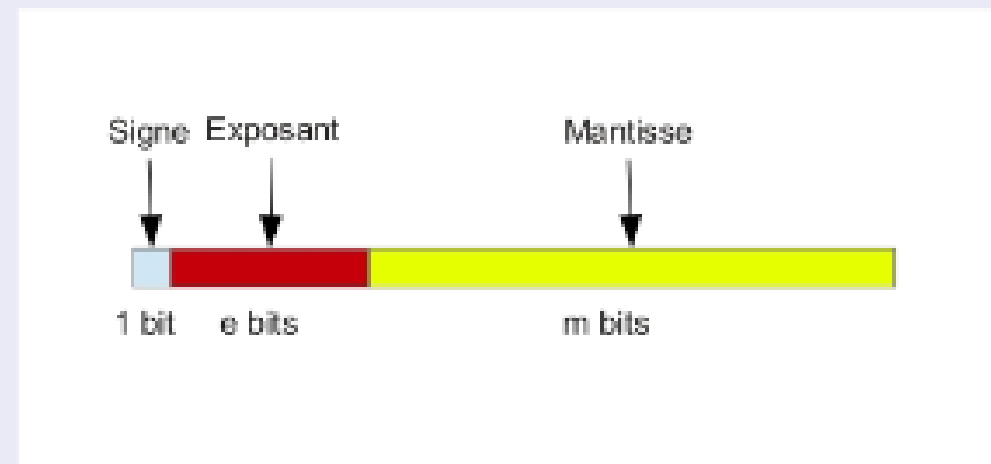
- En base 10, $\frac{1}{3}$ ne peut être représenté : $0.[33]_{10}$
- En base 2, $\frac{1}{10}$ ne peut être représenté : $0.000[1100]_2$

Ecriture des nombres réels (base 2)

Mantisse, exposant, signe

Un nombre réel est représenté par un triplet :

- le signe (1 bit)
Négatif si bit=1, positif sinon
- l'exposant *decidé* (e bits)
Nombre entier compris entre 0 et $2^e - 1$
- la mantisse (m bits)
Représente un nombre réel compris entre 0 et 1



Norme IEEE-754 - nombre de bits

Type	Signe	Exposant	Mantisse
float (simple précision)	1	8	23
double (double précision)	1	11	52

Codes vraiment équivalents ?

$$f(x) = x^2 + \frac{1}{10} \sin(x)$$

$$f(x) = x * x + 0.1 \sin(x)$$

$$g(n) = \sum_{i=1}^n \frac{1}{i}$$

$$g(n) = \sum_{i=n}^1 \frac{1}{i}$$

donnent elles toujours les mêmes résultats ?

NON !!!

2 codes mathématiquement équivalents peuvent mener
à des résultats différents

Impact de l'ordre des opérations

Somme des inverses de i (de 1 à N)

En simple précision, on obtient :

N	10^5	10^6	10^7	10^8
valeur exacte	12.09015	14.39273	16.69531	18.99790
$1 \rightarrow N$	12.09085	14.35736	15.40368	15.40368
$N \rightarrow 1$	12.09015	14.39265	16.68603	18.80792

- $\sum_{i=1}^n \frac{1}{i}$: Pour n grand, on finira par additionner des réels d'ordres de grandeur très différents : **absorptions**
- $\sum_{i=n}^1 \frac{1}{i}$: Les ordres de grandeur dans les additions seront semblables

L'ensemble des nombres flottants

On définit l'ensemble $F \subset \mathbb{R}$ par :

$$F = \{y \in \mathbb{R} \mid y = \pm \beta^e \left(\frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \cdots + \frac{d_t}{\beta^t} \right), e_{\min} \leq e \leq e_{\max}\},$$

ou encore

$$F = \{y \in \mathbb{R} \mid y = \pm m \beta^{e-t}, e_{\min} \leq e \leq e_{\max}\}.$$

Ceci correspond aux deux écritures $0,0038 = +10^{-2} \left(\frac{3}{10} + \frac{8}{10^2} \right) = +38.10^{-4}$ avec $e = -2$, $t = 2$, $e - t = -4$. Le nombre m s'appelle *la mantisse* et on utilise la notation $m = \overline{d_1 d_2 \dots d_t}^\beta$.

L'ensemble des nombres flottants

Notons que $0 \notin F$.

Pour $y \neq 0$, on a $m \beta^{e-t} = \beta^e \left(\frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \cdots + \frac{d_t}{\beta^t} \right) \geq \beta^e \frac{1}{\beta}$ car $d_1 \geq 1$. D'où $m \geq \beta^{t-1}$.
D'autre part, $m = \overline{d_1 d_2 \dots d_t} \beta^e = d_1 \beta^{t-1} + \cdots + d_{t-k} \beta^k + \cdots + d_{t-1} \beta + d_t < \beta^t$. On a donc montré que

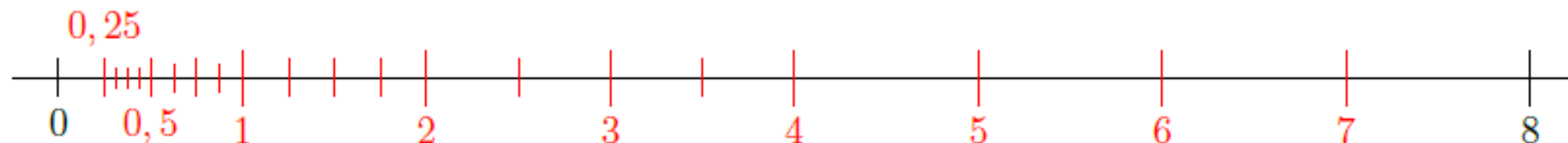
$$\beta^{t-1} \leq m < \beta^t.$$

L'ensemble des nombres flottants

F est un *système de nombres à virgule flottante* (floating point number system) noté $F(\beta, t, e_{\min}, e_{\max})$. Il dépend de quatre paramètres :

1. la *base* β (chiffres utilisés $0, 1, \dots, \beta - 1$),
2. la *précision* t (nombre de chiffres utilisés pour représenter la mantisse),
3. e_{\min} et e_{\max} qui définissent *le domaine des exposants*.

Par exemple, pour $F(2, 3, -1, 3)$, on obtient les nombres représentés en rouge sur la figure ci-dessous :



On constate que l'écart entre deux nombres consécutifs est multiplié par 2 à chaque puissance de 2.

Propriétés algébriques dans \mathbb{F}

Addition et multiplication

- La **commutativité est respectée** pour l'addition et la multiplication
 - ▶ $a + b = b + a$
 - ▶ $a * b = b * a$
- L'**associativité n'est pas respectée** (en général) ni pour l'addition, ni pour la multiplication
 - ▶ $(a + b) + c \stackrel{?}{=} a + (b + c)$
 - ▶ $(a * b) * c \stackrel{?}{=} a * (b * c)$
- La **distributivité n'est pas respectée** (en général) entre la multiplication et l'addition
 - ▶ $a(b + c) \stackrel{?}{=} ab + ac$

Exemple:

Soit $x = 8,22$, $y = 0,00317$ et $z = 0,00432$.

- $x \boxed{+} y = 8,22$ donc $(x \boxed{+} y) \boxed{+} z = 8,22$,
- $y \boxed{+} z = 0,01$ donc $x \boxed{+} (y \boxed{+} z) = 8,23$.

Analyse de l'erreur

Nombre approché

- Un *nombre approché* \hat{x} est un nombre légèrement différent du nombre exact x et qui dans le calcul remplace ce dernier.
- Si l'on sait que $\hat{x} < x$, \hat{x} est dit valeur approchée du nombre x *par défaut*;
- si $\hat{x} > x$, \hat{x} est une valeur approchée *par excès*.
- Soit $x = \sqrt{2}$. Le nombre $\hat{x} = 1.41$ est une valeur approchée par défaut, alors que le nombre $\hat{x} = 1.42$ est une valeur approchée par excès.
- Si \hat{x} est une valeur approchée de x on note

$$\hat{x} \approx x$$

Erreur absolue

Définition (Erreur absolue). *On appelle erreur absolue δ_x d'un nombre approché \hat{x} la valeur absolue de la différence entre le nombre exact x correspondant et le nombre approché donné*

$$\delta_x = |\hat{x} - x|.$$

Définition (Écart relatif). *L'écart relatif d'un nombre approché \hat{x} est le rapport*

$$\rho_x = \frac{\hat{x} - x}{x}.$$

Cette relation peut aussi être écrite sous la forme

$$\hat{x} = x(1 + \rho_x).$$

Erreur relative

Définition (Erreur relative). *L'erreur relative ε_x d'un nombre approché \hat{x} est la valeur absolue de l'écart relatif, c.-à-d. le rapport de l'erreur absolue δ_x de ce nombre et du module du nombre exact correspondant (si $x \neq 0$)*

$$\varepsilon_x = |\rho_x| = \left| \frac{\hat{x} - x}{x} \right| = \frac{\delta_x}{|x|}$$

L'erreur relative fournit une information plus pertinente sur la grandeur réelle de l'erreur. Cependant, elle n'est définie que pour $x \neq 0$.

Définition (Borne supérieure relative.). *La borne supérieure d'erreur relative u d'un nombre approché \hat{x} donné est un nombre quelconque supérieur ou égal à l'erreur relative de ce nombre*

$$\varepsilon_x = |\rho_x| \leq u$$

Sources d'erreurs: erreurs de modélisation

Les erreurs commises dans les problème mathématiques peuvent être en principe classées en cinq catégories.

Les deux premiers types d'erreur sont regroupés sous le nom d' erreurs de modélisation tandis que les trois derniers sont appelés erreurs numériques.

Erreurs de modèle: ces erreurs sont dues au fait que les modèles mathématiques sont plus ou moins idéalisés, ce qui donne lieu à plusieurs erreurs. Un exemple est l'erreur du modèle du pendule qui ne tient pas en considération la force de friction.

Erreurs de mesure: ces erreurs sont dues à la présence dans le modèle mathématique de paramètres numériques dont les valeurs ne peuvent être déterminées qu'approximativement suite à des mesures expérimentales. Telles sont toutes les constantes physiques, comme, par exemple, la longueur l dans le modèle du pendule.

Sources d'erreurs: erreurs numériques

Erreurs d'approximation ou de troncature: ces sont les erreurs associées aux processus infinis en analyse mathématique (par exemple les séries numériques).

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \dots$$

Erreurs d'arrondi: ce sont les erreurs associées au système de numération. Elles sont dues au fait qu'un ordinateur ne peut prendre en considération qu'un nombre fini de chiffres.

Erreurs de propagation et génération: ces sont les erreurs qui apparaissent dans le résultat d'une opération comme conséquence des erreurs des opérandes.

Dans ce qui suit nous allons nous intéresser aux deux derniers types d'erreur.

Erreur de Cancellation:

In this example we compare the following two equivalent expressions:

$$f_1(x) = \frac{(1+x) - 1}{x}$$

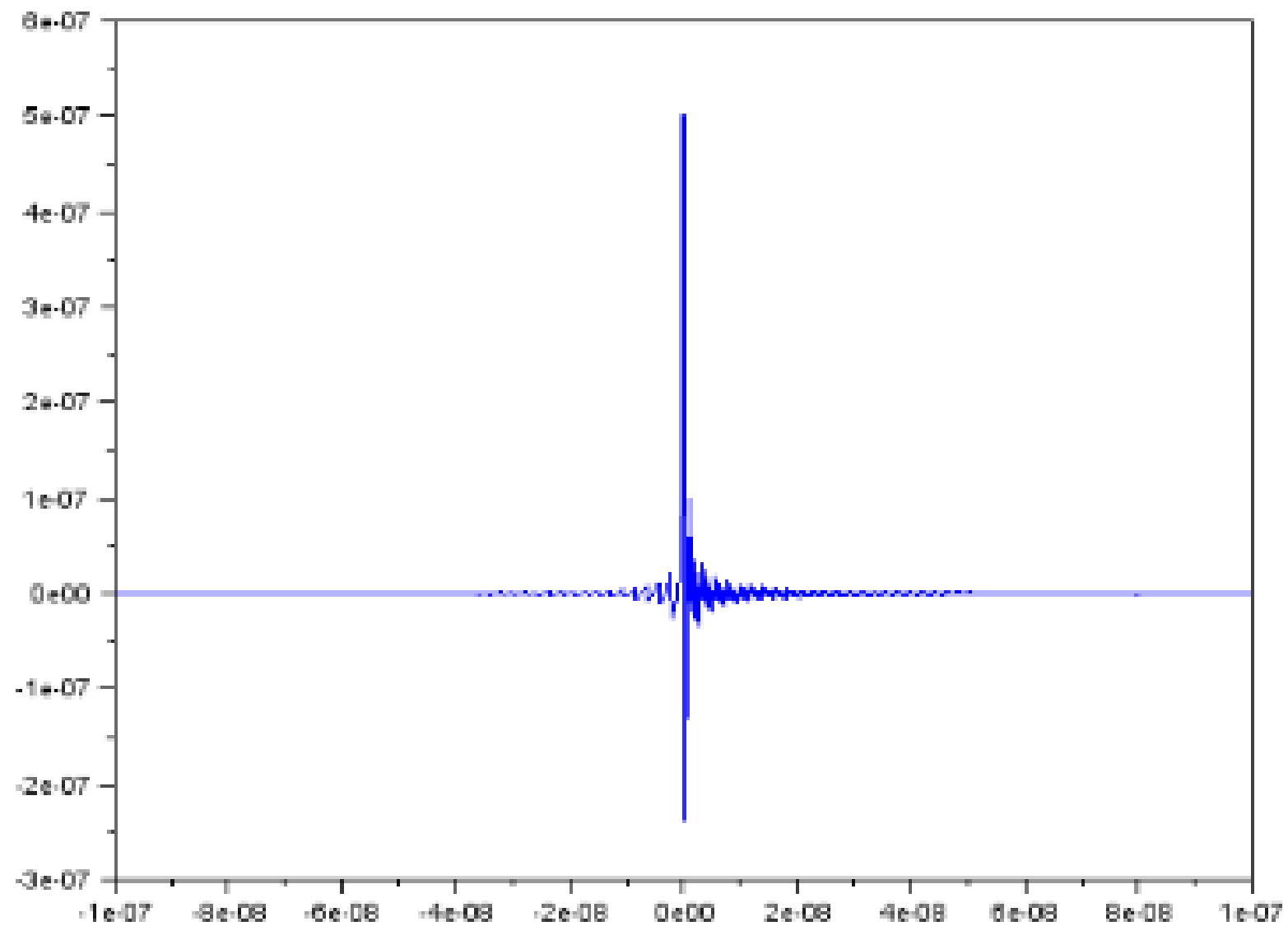
$$f_2(x) = 1$$

These two expressions are equivalent from an algebraic point of view; the second expression is simply obtained from the first one through simplification.

The first function suffers of the cancellation error when x goes to zero. This phenomenon has been explained in the previous step.

In the reported figure we have plotted the error $err = f_1(x) - f_2(x)$. We may see that the cancellation error is huge when x is close to zero. This phenomenon produces a loss of significant digits.

$$err(x) = f_1(x) - f_2(x)$$



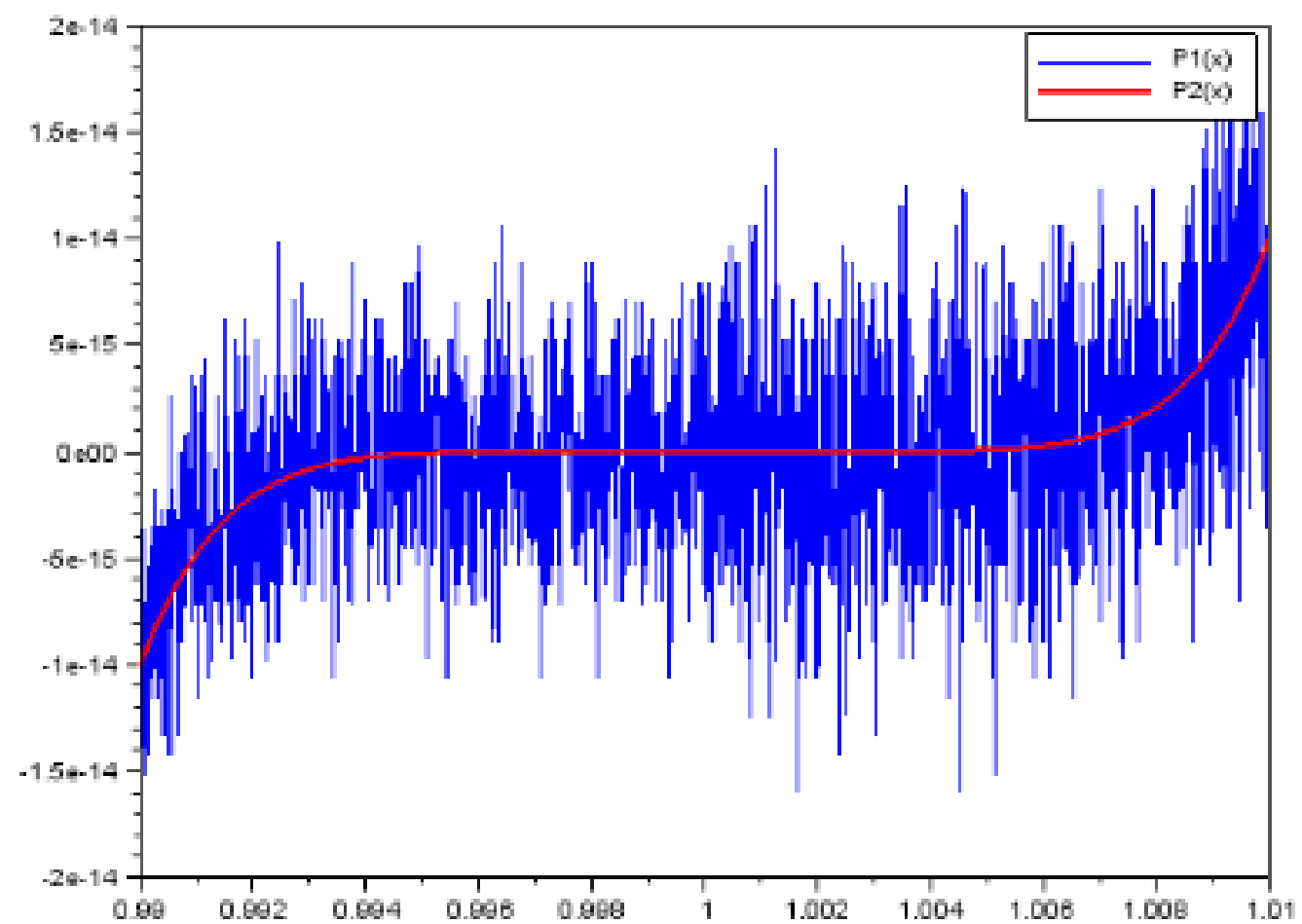
In this example, we compare the two following polynomial expressions:

$$P_1(x) = x^7 - 7x^6 + 21x^5 - 35x^4 + 35x^3 - 21x^2 + 7x - 1$$

$$P_2(x) = (x - 1)^7$$

where the first expression is obtained from the second by expansion.

As depicted on the right, the second expression is numerically stable while the first one is more unstable.



In this step, we study the computation of a derivative from a numerical point of view. To approximate the first derivate, we use here the difference quotient (a.k.a. Newton's quotient):

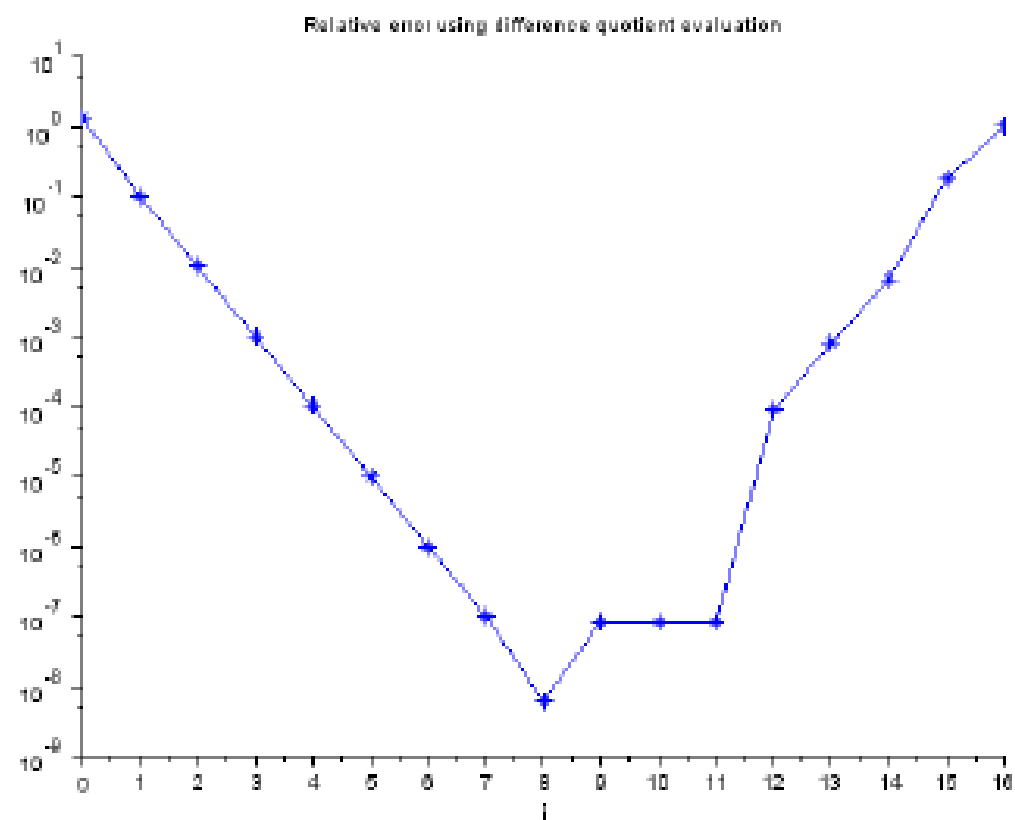
$$f'(x) = \frac{f(x+h) - f(x)}{h}$$

For example, we may analyze the formula for the function $f(x) = x^3 + 1$ at point $x = 1$ varying h in the following way:

$$h = 10^0, 10^{-1}, 10^{-2}, \dots, 10^{-16}$$

The plot on the right visualizes the results.

From a mathematical point of view, when h goes to zero, the formula should assume the value of the first derivative. Intuitively, we may expect the smaller the h , the better the approximation. Unfortunately, because of cancellation errors, it happens that the most reliable value for the derivate is reached in $h = 10^{-8}$ (for $h = 10^{-16}$, the value of the derivate is the same as $h = 10$).



i	h	Der. exact	Der. approx	Abs. error.	Rel. error.
0	1.0e+000	3.000000e+000	7.0000000000e+000	4.00000e+000	1.33333e+000
1	1.0e-001	3.000000e+000	3.3100000000e+000	3.10000e-001	1.03333e-001
2	1.0e-002	3.000000e+000	3.0301000000e+000	3.01000e-002	1.00333e-002
3	1.0e-003	3.000000e+000	3.0030010000e+000	3.00100e-003	1.00033e-003
4	1.0e-004	3.000000e+000	3.0003000100e+000	3.00010e-004	1.00003e-004
5	1.0e-005	3.000000e+000	3.0000300001e+000	3.00001e-005	1.00000e-005
6	1.0e-006	3.000000e+000	3.00000029998e+000	2.99980e-006	9.99933e-007
7	1.0e-007	3.000000e+000	3.00000003015e+000	3.01512e-007	1.00504e-007
8	1.0e-008	3.000000e+000	2.9999999818e+000	1.82324e-008	6.07747e-009
9	1.0e-009	3.000000e+000	3.00000002482e+000	2.48221e-007	8.27404e-008
10	1.0e-010	3.000000e+000	3.00000002482e+000	2.48221e-007	8.27404e-008
11	1.0e-011	3.000000e+000	3.00000002482e+000	2.48221e-007	8.27404e-008
12	1.0e-012	3.000000e+000	3.0002667017e+000	2.66702e-004	8.89006e-005
13	1.0e-013	3.000000e+000	2.9976021665e+000	2.39783e-003	7.99278e-004
14	1.0e-014	3.000000e+000	3.0198066270e+000	1.98066e-002	6.60221e-003
15	1.0e-015	3.000000e+000	3.5527136788e+000	5.52714e-001	1.84238e-001
16	1.0e-016	3.000000e+000	0.0000000000e+000	3.00000e+000	1.00000e+000

It is possible to compute the second derivatives of a function using the following approximation

$$f''(x) \approx \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}$$

Apply the above formula to compute the second derivative of the function $f(x) = \cos(x)$ at a given point $x = \frac{1}{2}$.

Moreover, prove the following estimate for the total error:

$$\left| f''(x) - \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} \right| \leq \frac{3\bar{\epsilon}}{h^2} M_1 + \frac{h^2}{12} M_2$$

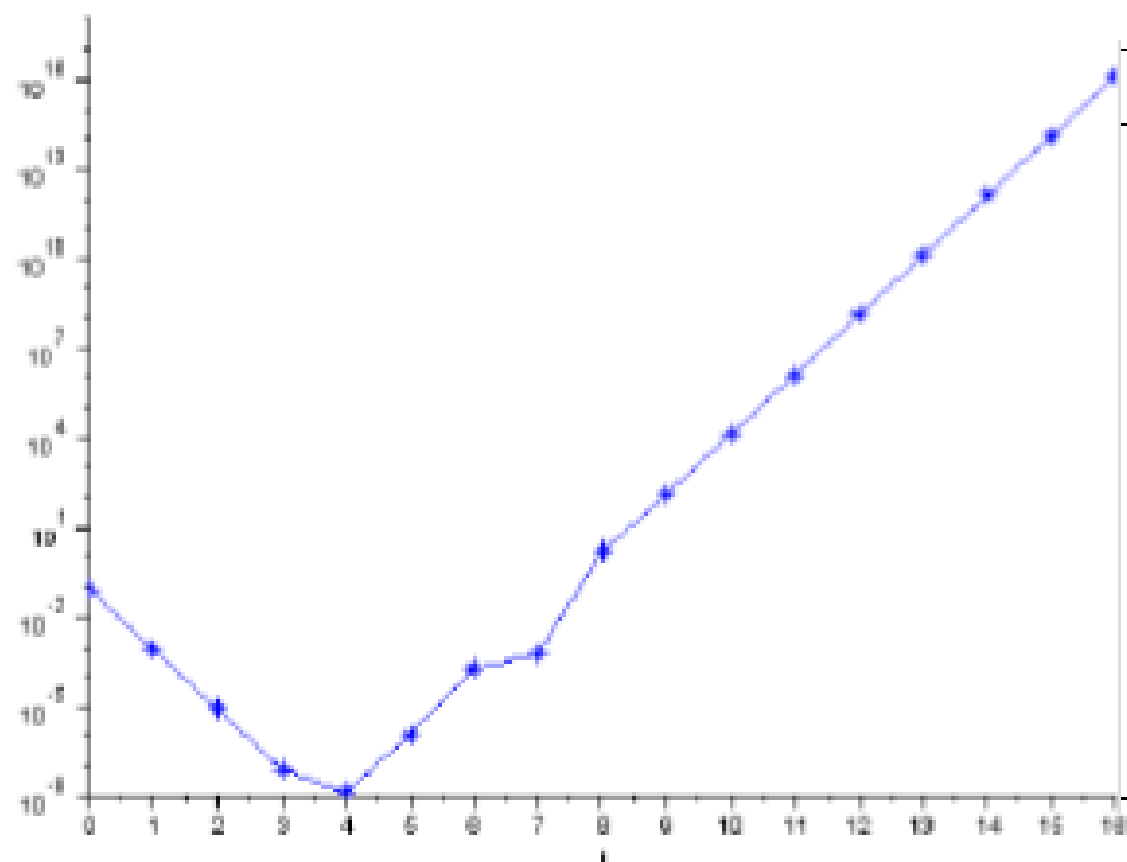
where $M_1 = \max_{x \in [x-h, x+h]} |f'(x)|$ and $M_2 = \max_{x \in [x-h, x+h]} |f^{(iv)}(x)|$.

The upper bound error is minimized when h has the value

$$h^* = \sqrt[4]{\frac{36\bar{\epsilon}M_1}{M_2}}$$

Hints: Using Taylor expansion for the function $f(x+h)$ and $f(x-h)$ to the fourth order paying attention to the signs.

Relative error



i	h	Der. exact	Der. approx	Abs. error.	Rel. error.
0	1.0e+000	-8.775826e-001	-8.0684536022e-001	7.07372e-002	8.06046e-002
1	1.0e-001	-8.775826e-001	-8.7685148682e-001	7.31075e-004	8.33056e-004
2	1.0e-002	-8.775826e-001	-8.7757524873e-001	7.31316e-006	8.33330e-006
3	1.0e-003	-8.775826e-001	-8.7758248890e-001	7.29897e-008	8.31713e-008
4	1.0e-004	-8.775826e-001	-8.7758257328e-001	1.13873e-008	1.29757e-008
5	1.0e-005	-8.775826e-001	-8.7758356138e-001	9.99486e-007	1.13891e-006
6	1.0e-006	-8.775826e-001	-8.7774232327e-001	1.59761e-004	1.82047e-004
7	1.0e-007	-8.775826e-001	-8.7707618945e-001	5.06372e-004	5.77008e-004
8	1.0e-008	-8.775826e-001	-2.2204460493e+000	1.34286e+000	1.53018e+000
9	1.0e-009	-8.775826e-001	-1.1102230246e+002	1.10145e+002	1.25509e+002
10	1.0e-010	-8.775826e-001	-1.1102230246e+004	1.11014e+004	1.26499e+004
11	1.0e-011	-8.775826e-001	-1.1102230246e+006	1.11022e+006	1.26509e+006
12	1.0e-012	-8.775826e-001	-1.1102230246e+008	1.11022e+008	1.26509e+008
13	1.0e-013	-8.775826e-001	-1.1102230246e+010	1.11022e+010	1.26509e+010
14	1.0e-014	-8.775826e-001	-1.1102230246e+012	1.11022e+012	1.26509e+012
15	1.0e-015	-8.775826e-001	-1.1102230246e+014	1.11022e+014	1.26509e+014
16	1.0e-016	-8.775826e-001	-1.1102230246e+016	1.11022e+016	1.26509e+016

La virgule flottante

La **virgule flottante** est une méthode d'écriture de **nombres réels** fréquemment utilisée dans les **ordinateurs**.

Quelques catastrophes dues à l'arithmétique flottante

Il y a un petit nombre “connu” de catastrophes dans la vie réelle qui sont due à une mauvaise gestion de l'arithmétique des ordinateurs (erreurs d'arrondis, d'annulation). Dans le premier exemple ci-dessous cela c'est payé en vies humaines.

Voir: <http://www5.in.tum.de/~huckle/bugse.htm>

Exemple concret d'une erreur numérique

- **Guerre du Golfe de 1991** : un anti-missile US Patriot dont le programme tournait depuis 100 heures a raté l'interception d'un missile Irakien Scud - **28 morts!!**
- **Explication** : l'anti **missile Patriot** incrémentait un compteur toutes les 0.1 secondes; 0.1 approché avec erreur 0.0000000953 (codé sur 24 bits) ; au bout de 100 heures, erreur cumulée 0.34s; dans ce laps de temps le Scud parcourt 500 mètres.

Explosion d'Ariane 5:

Le 4 juin 1996, une fusée Ariane 5, a son premier lancement, a explosé 40 secondes après l'allumage. La fusée et son chargement avaient coûté 500 millions de dollars. La commission d'enquête a rendu son rapport au bout de deux semaines. Il s'agissait d'une erreur de programmation dans le système inertiel de référence. À un moment donné, un nombre codé en virgule flottante sur 64 bits (qui représentait la vitesse horizontale de la fusée par rapport à la plate-forme de tir) était converti en un entier sur 16 bits. Malheureusement, le nombre en question était plus grand que 32768 (overflow), le plus grand entier que l'on peut coder sur 16 bits, et la conversion a été incorrecte.

Ariane 5 un lanceur de l'Agence spatiale européenne (ESA), développé pour placer des satellites sur orbite géostationnaire et des charges lourdes en orbite basse.

Bourse de Vancouver

Un autre exemple où les erreurs de calcul ont conduit à une erreur notable est le cas de l'indice de la Bourse de Vancouver. En 1982, elle a créé un nouvel indice avec une valeur nominale de 1000. Après chaque transaction boursière, cet indice était recalculé et tronqué après le troisième chiffre décimal et, au bout de 22 mois, la valeur obtenue était 524,881, alors que la valeur correcte était 1098.811. Cette différence s'explique par le fait que toutes les erreurs d'arrondi étaient dans le même sens : l'opération de troncature diminuait à chaque fois la valeur de l'indice.

Peut-on vraiment calculer avec un ordinateur ?

Conclusion

Les risques sont présents.



Mieux vaut les connaître.



TP avec Scilab

Exercices

On cherche à évaluer sur l'ordinateur de façon précise, pour de petites valeurs de x , les deux fonctions suivantes

$$f(x) = \frac{1}{1 - \sqrt{1 - x^2}}.$$

$$f(x) = \frac{1 + \sqrt{1 - x^2}}{x^2}$$

1. Pour $x = \sqrt{\text{\%eps}}/2$. Calculer $f(x)$ dans les deux cas.
2. Conclure.

Exercice:

Calcul des racines de $x^2 - 2px + 1$, quand $p \gg 1$ (ex : $p = 10^7$)

Algorithme 1

$$x^+ = p + \sqrt{p^2 - 1}$$

$$x^- = p - \sqrt{p^2 - 1}$$

Algorithme 2

$$x^+ = p + \sqrt{p^2 - 1}$$

$$x^- = 1/(p + \sqrt{p^2 - 1})$$

1. Calculer les deux racines pour les deux algorithmes.
2. Conclure.

Exercice

On considère la suite $u_{n+1} = \alpha u_n + \beta$, $n = 0, 1, \dots$, u_0 donné

1. Trouver l'expression de u_n en fonction de n .
2. Quand est ce que u_n est constante ?
3. Faites un programme scilab qui calcule u_n pour $n=0\dots30$.
4. Pour $u_0=1/3*(1-\text{delta})$, avec $\text{delta}=\text{eps_machine}$. Montrer que u_n converge vers $-\text{Inf}$.

Exercices

On considère la suite récurrente suivante

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

1. Montrer que u_n converge vers 6.

2. Montrer que

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

3. Ecrire un programme Scilab qui affiche u_n et la valeur exacte de u_n .

n	Computed value	Exact value
3	18.5	18.5
4	9.378378378378379	9.3783783783783784
5	7.8011527377521679	7.8011527377521613833
6	7.1544144809753334	7.1544144809752493535
11	6.2744386627644761	6.2744385982163279138
12	6.2186967691620172	6.2186957398023977883
16	6.1661267427176769	6.0947394393336811283
17	7.2356654170119432	6.0777223048472427363
18	22.069559154531031	6.0639403224998087553
19	78.58489258126825	6.0527217610161521934
20	98.350416551346285	6.0435521101892688678
21	99.898626342184102	6.0360318810818567800
22	99.993874441253126	6.0298473250239018567
23	99.999630595494608	6.0247496523668478987
30	99.999999999998948	6.0067860930312057585
31	99.99999999999943	6.0056486887714202679

Exercice: Calcul approché de π

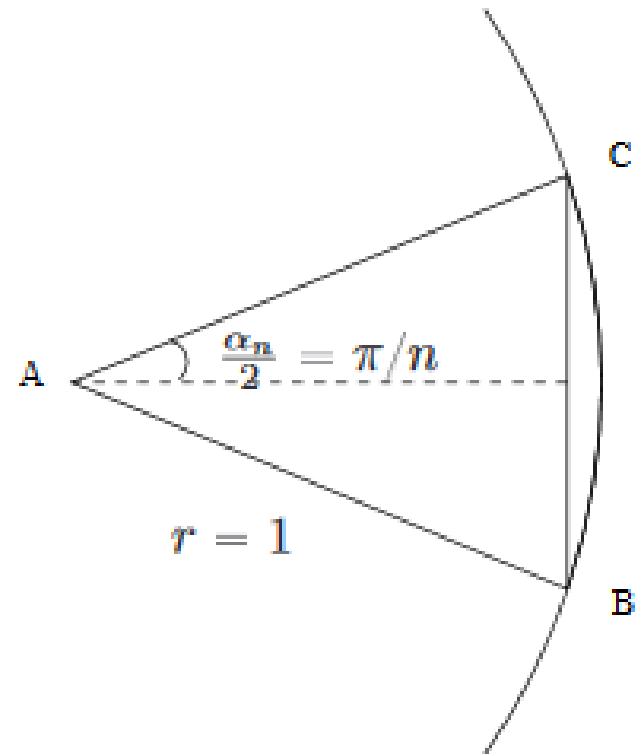
Regardons l'algorithme de calcul par les polygones inscrits. On considère un cercle de rayon $r = 1$ et on note A_n l'aire associée au polygone inscrit à n côtés. En notant $\alpha_n = \frac{2\pi}{n}$, A_n est égale à n fois l'aire du triangle ABC représenté sur la figure 1.1, c'est-à-dire

$$A_n = n \cos \frac{\alpha_n}{2} \sin \frac{\alpha_n}{2},$$

que l'on peut réécrire

$$A_n = \frac{n}{2} \left(2 \cos \frac{\alpha_n}{2} \sin \frac{\alpha_n}{2} \right) = \frac{n}{2} \sin \alpha_n = \frac{n}{2} \sin \left(\frac{2\pi}{n} \right).$$

Exercice: Calcul approché de π



Exercice: Calcul approché de π

Comme on cherche à calculer π à l'aide de A_n , on ne peut pas utiliser l'expression ci-dessus pour calculer A_n , mais on peut exprimer A_{2n} en fonction de A_n en utilisant la relation

$$\sin \frac{\alpha_n}{2} = \sqrt{\frac{1 - \cos \alpha_n}{2}} = \sqrt{\frac{1 - \sqrt{1 - \sin^2 \alpha_n}}{2}}.$$

Ainsi, en prenant $n = 2^k$, on définit l'approximation de π par récurrence

$$x_k = A_{2^k} = \frac{2^k}{2} s_k, \quad \text{avec } s_k = \sin\left(\frac{2\pi}{2^k}\right) = \sqrt{\frac{1 - \sqrt{1 - s_{k-1}^2}}{2}}$$

En partant de $k = 2$ (i.e. $n = 4$ et $s = 1$) on obtient l'algorithme suivant :

Algorithm 1.1 Algorithme de calcul de π , version naïve

1: $s \leftarrow 1, n \leftarrow 4$	▷ Initialisations
2: Tantque $s > 1e - 10$ faire	▷ Arrêt si $s = \sin(\alpha)$ est petit
3: $s \leftarrow \text{sqrt}((1 - \text{sqrt}(1 - s * s)))/2$	▷ nouvelle valeur de $\sin(\alpha/2)$
4: $n \leftarrow 2 * n$	▷ nouvelle valeur de n
5: $A \leftarrow (n/2) * s$	▷ nouvelle valeur de l'aire du polygône
6: fin Tantque	

Exercice: Calcul approché de π

1. Montrer que x_n converge vers π .
2. Ecrire un code Scilab qui approche la valeur de π .
3. Conclure.
4. Modifier votre code, en considérant la formule ci-après. Conclure.

$$\sin \frac{\alpha_n}{2} = \sqrt{\frac{1 - \sqrt{1 - \sin^2 \alpha_n}}{2}} = \sqrt{\frac{1 - (1 - \sin^2 \alpha_n)}{2(1 + \sqrt{1 - \sin^2 \alpha_n})}} = \frac{\sin \alpha_n}{\sqrt{2(1 + \sqrt{1 + \sin^2 \alpha_n})}}.$$

Bibliographie

1. P. Demailly, Analyse Numérique et Equations Différentielles, PUG, 1994.
2. M.J. Gander and W. Gander, Scientific Computing. An introduction using MAPLE and MATLAB, 2014.
3. DAVID GOLDBERG, What every computer scientist should know about Floating-Point Arithmetic, ACM Computing Surveys, Vol 23, No 1, March 1991.
4. Michael Overton, Numerical Computing with IEEE Floating Point Arithmetic, Siam, 2001.
5. Bo Einarsson, Accuracy and Reliability in Scientific Computing, Siam, 2005.
6. Jean-Michel Muller, Elementary Functions, algorithms and implementation, 2ème édition Birkhauser Boston, 2006.
7. Brisebarre, de Dinechin, Jeannerod, Lefèvre, Melquiond, Muller (coordinator), Revol, Stehlé and Torres, A Handbook of Floating-Point Arithmetic, Birkhauser Boston, 2010.
8. <http://www.computerhistory.org/timeline/computers/#169ebbe2ad45559efbc6eb35720d57b7>
9. Jean-Michel M ULLER et Jean-Claude BAJARD , Calcul et arithmétique des ordinateurs, version 3, 2004.
10. Donald Knuth, The art of computer programming, Addison-Wesley, 1997.
11. J. Higham, accuracy and Stability of Numerical Algorithms, SIAM.
12. Atkinson, An Introduction to Numerical Analysis, Wiley