Analyse numérique : Introduction au calcul approché

Pagora 1A

Chapitre 1

14 janvier 2013

Plan

Plan

Exemple introductif

On cherche à évaluer de manière numérique l'exponentielle $z = \exp(x)$.
Trouver $\tilde{z}$ approchant numériquement z.

Une méthode consiste à utiliser les séries de Taylor

$$\forall\, x \in \mathbb{R} \qquad \exp(x) = \sum_{k=0}^{+\infty} \frac{x^k}{k!}$$

En considérant des ressources de calcul limitées, on en est réduit à déterminer

$$\tilde{z} = \sum_{k=0}^{K} \frac{x^k}{k!}$$

$\Longrightarrow$ Pour le calcul de l'exponentielle, on commet donc une erreur.

Erreurs relatives aux données d'entrée

1. **Erreur de mesure** générée par la différence entre la valeur exacte x et la valeur mesurée $\tilde{x}$. Elle vaut

$$\Delta x = |x - \tilde{x}|$$

2. **Erreur d'arrondi** générée par la différence entre la valeur exacte x et la valeur arrondie $fl(x) = \hat{x}$. Elle vaut

$$\Delta x = x - \hat{x}$$

Exemple : Soit un objet valant 100 F, son vrai prix en euro est donc de

$$x = \frac{100}{6.55957} \approx 15,24490172374\,€$$

mais on ne le trouvera uniquement au prix de $\hat{x} = 15,24\,€$.
L'erreur commise est donc de $\Delta x = 4,901723741038 \times 10^{-2}\,€$

Erreurs résultantes d'un calcul

1. **Erreur de troncature**

Exemple : dans le calcul numérique de l'exponentielle, l'erreur de troncature vaut

$$T = z - \tilde{z} = \sum_{k=K+1}^{+\infty} \frac{x^k}{k!}$$

2. **Erreur d'arrondi** dans les étapes d'un calcul (algorithme, programme)

Exemple : on veut calculer avec une calculatrice la somme entre

$$x_1 = 6,2317 \times 10^7 \qquad x_2 = 2,179 \times 10^2$$

La valeur exacte du calcul est $x = 6,23172179 \times 10^7$ *et la valeur calculée est* $\hat{x} = 6,2317 \times 10^7$.

Erreurs

Soit z un résultat exact et $\tilde{x}$ un résultat approché. On définit alors

- **Erreur absolue**

$$\Delta z = z - \tilde{z}$$

- **Erreur relative**

$$\delta z = \frac{z - \tilde{z}}{z}$$

Exemple avec l'exponentielle

On rappelle

$$z = \exp(x) = \sum_{k=0}^{+\infty} \frac{x^k}{k!} \qquad \tilde{z} = \sum_{k=0}^{K} \frac{x^k}{k!}$$

Quelques résultats numériques sur l'erreur absolue et l'erreur relative

x	K	erreur absolue	erreur relative
1	3	$5,162 \times 10^{-2}$	1,90 %
1	4	$9,848 \times 10^{-3}$	0,37 %
1	10	$2,731 \times 10^{-10}$	$1,00 \times 10^{-8}$
5	3	$1,091 \times 10^{2}$	73,5 %
5	4	$8,304 \times 10^{1}$	55,9 %
5	10	$2,033 \times 10^{0}$	1,37%

Plan

Représentation des nombres

On cherche une manière simple de représenter les entiers dans un ordinateur.

La façon la plus simple de représenter un nombre est d'utiliser un symbole unique (représentation unaire)

$$\mathrm{I} = 1 \qquad \mathrm{II} = 2 \qquad \mathrm{IIIIIIIII} = 10$$

Le calcul est facile

$$\mathrm{I} + \mathrm{III} = \mathrm{IIII} \qquad \mathrm{II} \times \mathrm{III} = \mathrm{II\,II\,II} = \mathrm{IIIIII}$$

mais ça devient vite incompréhensible. De plus, le 0 n'existe pas dans cette représentation !

Base binaire

Dans le système décimal, un entier s'écrit

$$9325 = 9 \times 10^3 + 3 \times 10^2 + 2 \times 10^1 + 5 \times 10^0$$

Le même nombre en binaire s'écrit

$$\begin{aligned}(10010001101101)_2 = {} & 1 \times 2^{13} + 0 \times 2^{12} + 0 \times 2^{11} + 1 \times 2^{10} \\ & + 0 \times 2^9 + 0 \times 2^8 + 0 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 \\ & + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0\end{aligned}$$

Exercice : montrer que 9325 s'écrit bien $(10010001101101)_2$ *en base 2 puis reconvertir* $(10010001101101)_2$ *en base 10.*

Presque tous les ordinateurs travaillent en binaire.

Conversion décimal vers binaire

$$
\begin{aligned}
9325 &= 2 \times 4662 + 1 \\
4662 &= 2 \times 2331 + 0 \\
2331 &= 2 \times 1165 + 1 \\
1165 &= 2 \times 582 + 1 \\
582 &= 2 \times 291 + 0 \\
291 &= 2 \times 145 + 1 \\
145 &= 2 \times 72 + 1 \\
72 &= 2 \times 36 + 0 \\
36 &= 2 \times 18 + 0 \\
18 &= 2 \times 9 + 0 \\
9 &= 2 \times 4 + 1 \\
4 &= 2 \times 2 + 0 \\
2 &= 2 \times 1 + 0 \\
1 &= 2 \times 0 + 1
\end{aligned}
$$

Conversion binaire vers décimal

						0	1001000110110**1**
0	×	2	+	1	=	*1*	001000110110**1**
1	×	2	+	0	=	**2**	01000110110**1**
2	×	2	+	0	=	*4*	1000110110**1**
4	×	2	+	1	=	**9**	000110110**1**
9	×	2	+	0	=	*18*	00110110**1**
18	×	2	+	0	=	**36**	0110110**1**
36	×	2	+	0	=	*72*	110110**1**
72	×	2	+	1	=	**145**	10110**1**
145	×	2	+	1	=	*291*	0110**1**
291	×	2	+	0	=	**582**	110**1**
582	×	2	+	1	=	*1165*	10**1**
1165	×	2	+	1	=	**2331**	0**1**
2331	×	2	+	0	=	*4662*	**1**
4662	×	2	+	1	=	9325	

Opération sur des binaires : l'addition

On passe d'un nombre binaire au suivant en ajoutant 1, comme en décimal, sans oublier les retenues et en utilisant la table ordinaire :

- $0 + 0 = 0$
- $0 + 1 = 1$
- $1 + 0 = 1$
- $1 + 1 = (1)0$ (1) étant la retenue

Exemple : 11 + 1 = 100 en base 2.

Exercice : écrire $(34)_{10}$ *et* $(27)_{10}$ *en binaire puis effectuer l'opération en binaire* $(34)_{10} + (27)_{10}$ *et vérifier que le résultat obtenu soit le bon.*

Correction exercice

Exercice : écrire $(34)_{10}$ *et* $(27)_{10}$ *en binaire puis effectuer l'opération en binaire* $(34)_{10} + (27)_{10}$ *et vérifier que le résultat obtenu soit le bon.*

- $(34)_{10} = (100010)_2$
- $(27)_{10} = (11011)_2$
- $(100010)_2 + (11011)_2 = (111101)_2$
- $(111101)_2 = (61)_{10}$
- $(34)_{10} + (27)_{10} = (61)_{10}$

Opération sur des binaires : la multiplication

Elle s'effectue comme en décimal, la table ordinaire se résume à :

- $0 \times 0 = 0$
- $0 \times 1 = 0$
- $1 \times 0 = 0$
- $1 \times 1 = 1$

Exemple : $10 \times 11 = 110$ en base 2.

Exercice : écrire $(90)_{10}$ et $(97)_{10}$ en binaire puis effectuer l'opération en binaire $(90)_{10} \times (97)_{10}$ et vérifier que le résultat obtenu soit le bon.

Correction exercice

Exercice : écrire $(90)_{10}$ *et* $(97)_{10}$ *en binaire puis effectuer l'opération en binaire* $(90)_{10} \times (97)_{10}$ *et vérifier que le résultat obtenu soit le bon.*

- $(90)_{10} = (1011010)_2$
- $(97)_{10} = (1100001)_2$
- $(1011010)_2 \times (1100001)_2 = (10001000011010)_2$
- $(10001000011010)_2 = (8730)_{10}$
- $(90)_{10} \times (97)_{10} = (8730)_{10}$

Les entiers dans un ordinateur

- Dans un ordinateur, les entiers sont représentés à l'aide de bits (1 bit = un caractère 0 ou 1)
- Typiquement, on utilise 32 ou 64 bits pour les représenter
- 1 des bits est uniquement dedié au signe de l'entier (0 = +, 1 = -)

Exemple : sur 32 bits, les entiers peuvent prendre une valeur comprise entre - 2 147 483 648 et 2 147 483 647.

Exercice : si on dispose de 4 bits (bit de signe compris), quelles valeurs peuvent prendre les entiers ?

Correction exercice

Exercice : si on dispose de 4 bits (bit de signe compris), quelles valeurs peuvent prendre les entiers ?

- Si on dispose de de 4 bits (dont bit de signe), cela laisse trois bits pour compter les entiers naturels vont donc varier de 0 à $2^3 - 1 = 7$.
- En prenant compte du bit de signe, les entiers devraient varier entre -7 et 7.
- Mais 2 combinaisons ont la même valeur : 1 000 et 0 000 (en bleu le bit de signe) pour 0.
- Pour éviter cette redondance, on pose 1 000 = -8 (classiquement le bit de signe vaut 1 indique un signe négatif).
- sur 4 bits, les entiers prennent une valeur comprise entre - 8 et 7.

Plan

Aspect fini des ordinateur

Soit l'exemple suivant :

- $a = 10^{20}$, $b = -10^{20}$, $c = 1$
- Lorsque l'ordinateur effectue $(a + b) + c$, on obtient le résultat suivant

$$1$$

- Lorsque l'ordinateur effectue $a + (b + c)$, on obtient le résultat suivant

$$0$$

Pourquoi ?

Ceci est dû au fait que le stockage de a, b et c sur l'ordinateur ne permet pas d'avoir une assez grande précision pour éviter dans le second cas que $b + c = -10^{20}$.

Nombre flottant

Définition

Un nombre flottant est composé de 3 éléments :

- la base : notée b, elle donne la base arithmétique utilisée pour la représentation.
- la mantisse : elle contient la valeur normalisée du nombre que l'on veut représenter. Sa taille maximale est spécifiée par par un entier positif m.
- l'exposant : il définit le décalage par rapport à la normalisation. Sa taille maximale est spécifiée par un entier positif e.

On note $F[b, m, e]$ le système.

Exemple

Soit le système $F[10, 5, 3]$.

On souhaite représenter le nombre $x = 0,00123456789$.

La valeur normalisée est donnée par $x = 0,123456789 \times 10^{-3}$.

La valeur arrondie est donnée par $\hat{x} = 0,12346 \times 10^{-003}$.

La valeur tronquée est donnée par $\hat{x} = 0,12345 \times 10^{-003}$.

Remarque

- Dans ce système, la mantisse est bornée par 99999 et l'exposant par 999.
- Le plus grand nombre que l'on puisse représenter est $0,99999 \times 10^{999}$.
- Le plus petit nombre positif que l'on puisse représenter est $0,00001 \times 10^{-999}$.

Flottant binaire

Dans le système décimal, un nombre s'écrit

$$9,90625 = 9 \times 10^0 + 9 \times 10^{-1} + 0 \times 10^{-2} + 6 \times 10^{-3} + 2 \times 10^{-4} + 5 \times 10^{-5}$$

Le même nombre en binaire s'écrit

$$(1001,11101)_2 = 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3} + 0 \times 2^{-4} + 1 \times 2^{-5}$$

Exercice : vérifier l'égalité des 2 nombres

La conversion et les calculs se font de la même manière que pour les entiers.

Correction exercice

Exercice : vérifier l'égalité entre $(9,90625)_{10}$ *et* $(1001,11101)_2$

- Pour la partie entière, on a évidemment que $(9)_{10} = (1001)_2$.
- La partie décimale se traite de la manière suivante :

$$
\begin{array}{rcccccccl}
0,90625 & \times & 2 & = & 1,\mathbf{8125} & \geq & 1 & \Longrightarrow & 1001,1 \\
0,\mathbf{8125} & \times & 2 & = & 1,\mathit{625} & \geq & 1 & \Longrightarrow & 1001,11 \\
0,\mathit{625} & \times & 2 & = & 1,\mathbf{25} & \geq & 1 & \Longrightarrow & 1001,111 \\
0,\mathbf{25} & \times & 2 & = & 0,\mathit{5} & < & 1 & \Longrightarrow & 1001,1110 \\
0,\mathit{5} & \times & 2 & = & 1 & \geq & 1 & \Longrightarrow & 1001,11101
\end{array}
$$

Système standard : simple précision

- C'est l'une des représentations standard des nombre flottants les plus utilisés (norme IEEE 754)
- Les nombres sont représentés sur des blocs de mémoire de taille 32 bits.
- Chaque nombre se décompose de la manière suivante :
  - 24 bits pour la mantisse (dont 1 bit de signe)
  - 8 bits pour l'exposant (dont 1 bit de signe)
- Le plus grand nombre que l'on puisse représenter est $x_{max} = 1,7 \times 10^{38}$
- Le plus petit nombre positif que l'on puisse représenter est $x_{min} = 7,0 \times 10^{-46}$
- Ce standard correspond à $F[2, 23, 7]$.

Système standard : double précision

- C'est l'une des représentations standard des nombre flottants les plus utilisés (norme IEEE 754)
- Les nombres sont représentés sur des blocs de mémoire de taille 64 bits.
- Chaque nombre se décompose de la manière suivante :
  - 53 bits pour la mantisse (dont 1 bit de signe)
  - 11 bits pour l'exposant (dont 1 bit de signe)
- Le plus grand nombre que l'on puisse représenter est $x_{max} = 9 \times 10^{307}$
- Le plus petit nombre positif que l'on puisse représenter est $x_{min} = 2,5 \times 10^{-324}$
- Ce standard correspond à $F[2, 52, 10]$.