Université Batna 2
Faculté des Mathématiques et de l'Informatique
Département d'Informatique

Méthodes Numériques

Oussama HARKATI

Cours, 2`eme année licence Informatique

2021-2022

Contenu de la matière

CHAPITRE I

Chapitre 1

Généralités sur l'analyse numérique et le calcul scientifique.

1.1 Motivation

Un algorithme est un ensemble de règles qui permettent d'effectuer un certain travail ou résoudre un certain problème. Si les données et les résultats sont des nombres on dit que cet algorithme est un algorithme numérique, l'étude de cette sorte d'algorithme forme le calcul numérique. En général le calcul numérique ou scientifique se compose de deux types de méthodes ; les méthodes exactes et les méthodes approchées.

Certains problèmes peuvent se résoudre de façon exacte avec un nombre fini d'opérations arithmétiques élémentaires on appelle ça une méthode exactes, mais il est bien connu que l'analyse mathématique classique est incapable de résoudre tous les problèmes. Dans un tel cas on remplace la résolution mathématique des problèmes par sa résolution numérique, c'est à cause de cela que sont apparues les méthodes numériques qu'on l'appelle des méthodes approchées. C'est méthode numérique sont étudiées par une branche des Mathématiques dite L'ANALYSE NUMERIQUE.
On peut dire encore que l'analyse numérique étudie la théorie des méthodes constructive de l'analyse mathématique c'est-à-dire des méthodes qui permettent d'obtenir la solution numérique approchée d'un problème mathématique, avec une précision désirée, après un nombre fini d'opération arithmétiques élémentaires.

Puisque les méthodes de l'analyse numérique sont des méthodes approchées, l'analyse numérique devra étudier l'erreur d'approximation de la méthode. De son côté l'algorithmique traitera des erreurs numériques engendrées par le fait qu'un ordinateur ne travaille qu'avec un nombre fini de chiffres significatifs.

En résumé l'analyse numérique étudie la mathématique des algorithmes tandis que l'algorithmique étudie les problèmes posés par leur mise en œuvre sur ordinateur.

1.2 Arithmétique en virgule flottante et erreurs d'arrondis.

De nos jours, de nombreux calculs sont effectués à l'aide d'ordinateurs. Nombreux sont les gens qu'effectuent des calculs sur l'ordinateur sans attention particulière mais il est important de savoir la manière dont sont représentés et manipulés les nombres dans une machine car un nombre est représenté par un nombre fini de caractères, fixé à l'avance, qui dépend de l'architecture de la machine. Ainsi tous les nombres entiers ou réels ne peuvent pas être représentés. Les conséquences en sont très importantes, en particulier dans la précision des résultats lors de calculs ou même le plus simple des programmes peut avoir un comportement surprenant.

1.2.1 Représentation des nombres en machine.

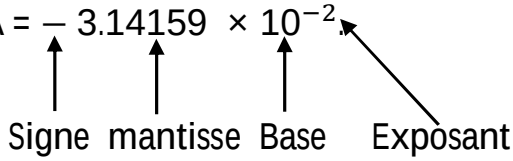
De nombreux domaines de l'informatique nécessitent une manipulation efficace des nombres réels. Or l'ensemble des nombres réel est à la fois infini et continu. Une représentation complète de cet ensemble des nombres en machine est donc impossible. Dans ce cas on distingue trois types des nombres :

1. Les nombres entiers dont la représentation et la manipulation sont celles de l'arithmétique usuelle.
2. Les entiers relatifs codés sur n bits ont pour valeur dans $[-2^{n-1}, 2^{n-1}-1]$.
3. Les nombres flottants qui représentent les nombres réels ou les nombres décimaux.

Pour approximer les nombres réels, plusieurs approches ont été utilisées, comme la représentation en virgule fixe, les nombres rationnels... . La représentation en virgule flottante reste la plus utilisée pour approximer les nombres réels.

1.2.2 Arithmétique en virgule flottante.

Le nombre réel A = -314.159 peut être représenté par plusieurs représentation comme A = -3.14159×10^2 ou $A = -314159 \times 10^{-3}$ ou par une représentation scientifique comme suit A = -3.14159×10^{-2}.



Les ordinateurs utilisent une représentation similaire dite représentation en virgule flottante.

Définition 1.1 : Les nombres à virgule flottante souvent appelés nombres flottants, ou encore flottants sont une représentation d'un sous-ensemble fini des nombres réels. Ils sont

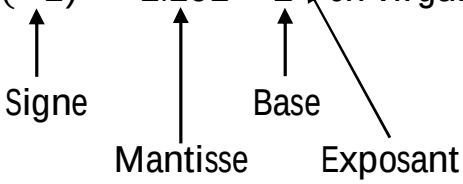
utilisés pour représentés les nombres réels de façon approximative en mémoire, avec la convention standardisée de la forme suivante : $x = (-1)^s \times m \times \beta^e$ où :

- s représente le signe = (0 ou 1).
- m est la mantisse ou la fraction, représentée par un nombre fixé de chiffres $d_0 d_1 d_2 \dots d_{p-1}$ avec $0 \leq d_i \leq \beta - 1$, $\forall\, i = \overline{0, p-1}$ où p est le nombre de chiffres qui composent la mantisse on dit alors que p est la précision de x d'où $0 \leq m < \beta^p$.
- β est la base de calculs (usuellement 2 ou 10).
- e est l'exposant qui est un entier signé compris entre deux bornes $e_{min} \leq e \leq e_{max}$.

Cette représentation n'est pas unique : par exemple, le flottant 3.1416 en base 10 et précision 5 peut s'écrire aussi 0.31416×10^1 ou encore 31416×10^{-4}.
La notion d'exposant n'est donc pas intrinsèque et est relative à la convention choisie pour la mantisse. Une fois choisie une telle, on peut calculer l'intervalle de nombres flottants correspondant à $[e_{min};\ e_{max}]$.

Exemple 1.1

Soit le nombre 3.25, ce nombre peut être écrit en binaire comme suit :
3 = 11 et 0.25 = 0.01
D'où 3.25 = $(11.01)_2 = (-1)^0 \times 0.1101 \times 2^2$ ou $(-1)^0 \times 1.101 \times 2^1$ en virgule flottante.



Exercice d'application 1.1

Ecrire les nombres suivants en binaire, puis représenter les en virgule flottante.
238 et 61.625 et 181.3.

1.2.2.1 La norme IEEE 754-2008.

Comme on a vu, il y a plusieurs manières de représentation des nombres réels en virgule flottante selon les valeurs de m et e. Donc il faut avoir une manière standard pour une représentation unique des réels dans la machine.

Représentation 1.1 : La norme IEEE 754 définit 4 formats de flottants ; deux formats de nombres en virgule flottante (et deux formats étendus optionnels) en base 2.
Dans ce cours on s'intéresse aux deux représentations de nombres en virgule flottante ; la représentation en simple précision et en double précision.

- Représentation en simple précision : Dans cette représentation on utilise 32 bits pour la représentation des réels. La répartition des bits est la suivant : 1 bit de signe et 8 bits pour un exposant biaisé avec 23 bits pour les bits de m comme suit :

1	8	23
Signe	Exposant Biaisé	Mantisse

Cette représentation est dite aussi Binary32.

- Représentation en double précision : Dans cette représentation on utilise 64 bits pour la représentation des réels. La répartition des bits est la suivant : 1 bit de signe et 11 bits pour un exposant biaisé avec 52 bits pour les bits de m comme suit :

1	11	52
Signe	Exposant Biaisé	Mantisse

Cette représentation est dite aussi Binary64.

Les deux représentations sont résumées dans ce tableau.

Précision	Simple	Double
Encodage	32 Bits	64 Bits
Signe	1 Bit	1 Bits
Exposant	8 Bits	11 Bits
Mantisse	23 Bits	52 Bits
Valeur d'un nombre	$(-1)^s \times m \times 2^{Eb-127}$	$(-1)^s \times m \times 2^{Eb-1023}$
Précision	24 Bits	53 Bits
e_{min}	-126	-1022
e_{max}	+127	+1023
Chiffres significatifs	Environ 7	Environ 16

Remarque 1.1

Eb est dit l'exposant biaisé.

1.2.2.1.1 Nombres normalisés et dénormalisés.

Même en fixant la place de la virgule dans la mantisse, le même nombre peut avoir plusieurs représentations. Ainsi en base β = 10, avec une précision p = 4, les deux flottants 3.140 et 0.314×10^1 sont identiques.

Définition 1.2 : On parle de nombre normalisé lorsque le chiffre de poids fort de la mantisse est non nul et pour tout réel représentable non nul, la représentation devient alors unique.
Une conséquence intéressante en base $\beta = 2$ est que pour un nombre normalisé, le premier chiffre (bit) de la mantisse est toujours 1 c'est-à-dire une mantisse normalisée est constituée d'un chiffre 1, de la virgule, et du reste de la mantisse. On peut alors décider de ne pas le stocker physiquement en mémoire, et on parle alors de bit de poids fort implicite.

Exemple 1.2

Soit le nombre A = 5.5
Ce nombre en binaire est : 5 = 101 et 0.5 = 1
Alors 5.5 en virgule fixe est : 101.1 mais c'est dénormalisé, pour le normalisé il faut décaler la virgule vers la gauche, de façon à ne laisser qu'un 1 sur sa gauche d'où 5.5 en sa forme normalisé sera 1.011×10^2

Remarque 1.2

Si N est un nombre réel qu'on veut l'écrire avec la norme IEEE 754, il y'a 5 étapes à suivre :

1. Détecter le signe du N.
2. Ecrire N en binaire $(N)_{10} \Leftrightarrow (N)_2$.
3. Normaliser le nombre $(N)_2$.
4. Calculer l'exposant biaisé Eb = 127 + e ou Eb = 1023 + e selon la précision utilisée.
5. Faire remplir les champs selon la précision utilisée, simple ou double.

Remarque 1.3

Si B est un nombre représenté avec la norme IEEE 754 et qu'on veut l'écrire sous la forme décimale il faut suivre les étapes suivantes :

1. Diviser le nombre en binaire selon la méthode utilisée simple ou double.
2. Détecter le signe de B.
3. Calculer l'exposant e = Eb -127 ou e = Eb – 1023.
4. Ecrire la mantisse en base 10.
5. Réécrire B sous la forme $(-1)^s \times m \times 2^e$ et faire les calculs nécessaires.

Exemple 1.3

Représenter le nombre N = 181.3 on utilisant la norme IEEE 754 à simple précision.

- Etape 01 Détecter le signe de N.

181.3 est un nombre positif donc $s = 0$.

- Etape 02 Ecrire N en binaire.
  181.3 = $(10110101.0100110011\ldots)_2$
- Etape 03 Normaliser le nombre $(N)_2$.
  181.3 = $1.01101010100110011 \ldots \times 2^7$.
- Etape 04 Calculer l'exposant biaisé Eb.
  On a $e = 7$ d'où Eb = 127 + e = 127 + 7 = 134.
  134 = $(10000110)_2$.
- Etape 05 Faire remplir les champs.

0	10000110	01101010100110011001100
Signe	Exposant Biaisé	Mantisse

Remarque 1.4

Avant d'écrire la mantisse il faut enlever le bit le plus fort car c'est un bit implicite.

Exemple 1.4

Convertir le nombre suivant B = 00111111010000000000000000000000 en décimal.

- Etape 01 Diviser les champs de nombre.
  0 / 01111110 / 10000000000000000000000
- Etape 02 Détecter le signe de B.
  Le premier bit est 0 donc B est positive.
- Etape 03 Calcule de l'exposant e.
  On a Eb = 01111110 = $(126)_{10}$ donc e = 126 – 127 = -1.
- Etape 04 Ecrire la mantisse en base 10.
  m = 10000000000000000000000 = 2^{-1} = 0.5.
  N'oublie pas d'ajouter le bit implicite d'où m sera 1.5.
- Etape 05 Réécrire B sous la forme $(-1)^s \times m \times 2^e$ et faire les calculs nécessaire.
  B = $(-1)^0 \times 1.5 \times 2^{-1}$ = 0.75.

Exemple 1.5

Convertir le nombre suivant B = 01000011001101010100110011001100 en décimal.

- Etape 01 Diviser les champs de nombre
  0 / 10000110 / 01101010100110011001100
- Etape 02 Détecter le signe de B.
  Le premier bit est 0 donc B est positive.

- Etape 03 Calcule de l'exposant e.
  On a Eb = 10000110 = $(134)_{10}$ donc e = 134 – 127 = 7.
- Etape 04 Ecrire la mantisse en base 10.
  m = 01101010100110011001100 = $2^{-2} + 2^{-3} + 2^{-5} + 2^{-7} + 2^{-9} + 2^{-12} + 2^{-13} + 2^{-16} + 2^{-17} + 2^{-20} + 2^{-21}$.
  N'oublie pas d'ajouter le bit implicite d'où m sera $1 + 2^{-2} + 2^{-3} + 2^{-5} + 2^{-7} + 2^{-9} + 2^{-12} + 2^{-13} + 2^{-16} + 2^{-17} + 2^{-20} + 2^{-21}$.
- Etape 05 Réécrire B sous la forme $(-1)^s \times m \times 2^e$ et faire les calculs nécessaire.
  $B = (-1)^0 \times (1 + 2^{-2} + 2^{-3} + 2^{-5} + 2^{-7} + 2^{-9} + 2^{-12} + 2^{-13} + 2^{-16} + 2^{-17} + 2^{-20} + 2^{-21}) \times 2^7 = 2^7 + 2^5 + 2^4 + 2^2 + 2^0 + 2^{-2} + 2^{-5} + 2^{-6} + 2^{-9} + 2^{-10} + 2^{-13} + 2^{-14} = 181.29998779296875 \approx 181.3$

Exercice d'application 1.2

Ecrire les nombres suivants en utilisant la norme IEEE 754.
A = -0.625 et B = 433.75 et C =-0.005 et D = $\frac{567.25}{5}$.

Remarque 1.5

En plus des nombres usuels, la norme IEEE 754 définit trois nombres spéciaux : $+\infty$ et $-\infty$ et aussi NaN (Not-a-Number). De plus, le zéro est signe (il y a un +0 et un -0, qui sont égaux).

1.2.2.1.2 Notion d'arrondi.

Comme on a vu dans l'exemple 1.5 le résultat de la conversion de la norme IEEE 754 en décimal est une valeur approchée de 181.3 il faut faire un arrondi pour avoir le résultat exact.

Représentation 1.2 : La norme IEEE 754 définit 4 modes d'arrondi : vers $-\infty$, vers $+\infty$, vers 0, et au plus proche. Un autre mode intéressant est l'arrondi à l'opposé de 0 qui est le symétrique de l'arrondi vers 0, comme l'arrondi vers $+\infty$ est le symétrique de celui vers $-\infty$. Les arrondis vers $\pm\infty$ et vers 0 sont appelés arrondis dirigés car ils sont dirigés dans une direction donnée.

La conséquence la plus importante est que, une fois un mode d'arrondi choisi, le résultat d'une opération est parfaitement spécifié, en particulier il y a un seul résultat possible. On parle alors d'arrondi correct.

1.2.2.1.2.1 Arrondi correct

La norme IEEE 754 impose l'arrondi correct pour les 4 opérations de base (addition, soustraction, multiplication, division) ainsi que pour la racine carrée. Par contre, la norme IEEE 754 n'impose rien pour les autres opérations mathématiques (puissance, exponentielle, logarithme, sinus, cosinus, etc.) pour lesquelles aucune exigence n'est imposée.

1.2.3 Erreurs d'arrondis.

Définissons un numéro de machine, comme un nombre qui peut être exactement représenté dans le système flottant à l'étude. En général, la somme, le produit et le quotient de deux numéros de machine n'est pas un numéro de machine et le résultat d'une telle opération arithmétique doit être arrondi.

1.2.3.1 L'erreur absolue et l'erreur relative

Soit x un nombre exact et x^* une valeur approchée de x, pour dire que x^* est une valeur approchée de x , ou x^* presque égal à x on écrit $x \approx x^*$ ou $x \simeq x^*$.
Si $x^* > x$ on dit que x^* est une valeur approchée par excès sinon si $x^* < x$ on dit que x^* est une valeur approchée par défaut.

Exemple 1.6

1- Nombres exacts comme 1, 3, π, e, $\sqrt{7}$ …
2- Valeurs approché comme $\pi \approx 3.14$, $\sqrt{5} \approx 2.23$ sont des approximations par défaut mais le nombre $e \approx 2.72$ est une approximation par excès.

1.2.3.1.1 L'erreur absolue

Définition 1.3 : On appelle erreur absolue du nombre approché x^* de x la quantité réelle positive, notée $\Delta(x)$, définie par $\Delta(x) = |x - x^*|$ plus qu'il est petite, plus x^* est précis.
L'erreur absolue (Δ) est la distance absolue entre le résultat flottant et la valeur réelle. Cette mesure peut être utilisée pour estimer l'erreur d'arrondi d'une formule lorsqu'on connait une majoration pour les valeurs des variables.

Exemple 1.7

Pour la valeur exacte $= \frac{2}{3}$, la valeur approchée $x_1^* = 0.666667$ est 1000 fois plus précise que la valeur approchée$x_2^* = 0.667$ En effet, nous avons :

$$\Delta_1(x) = |x - x_1^*| = \left|\frac{2}{3} - 0.666667\right| = \left|\frac{2}{3} - \frac{666667}{10^6}\right| = \frac{1}{3}\,10^{-6}$$
$$\Delta_2(x) = |x - x_2^*| = \left|\frac{2}{3} - 0.667\right| = \left|\frac{2}{3} - \frac{667}{10^3}\right| = \frac{1}{3}\,10^{-3}$$

Exercice d'application 1.3

Calculer l'erreur absolue correspondant aux mesures suivantes :
$x = \frac{1}{18} = 0.556$ et $y = \frac{22}{7} = 3.14285$ et aussi $z = \frac{9}{14} = 0.643$

1.2.3.1.2 L'erreur relative

Définition 1.4 : On appelle erreur relative du nombre approché x^* de x la quantité réelle positive, notée $\varepsilon(x)$, définie par $\varepsilon(x) = \frac{|x-x^*|}{|x|} = \frac{\Delta(x)}{|x|}$, l'erreur relative est souvent exprimée en pourcentage (précision relative) par $\varepsilon\,\% = \varepsilon(x) \times 100$.
L'erreur relative (ε) est le rapport entre l'erreur absolue et la valeur réelle.

Exemple 1.8

Pour les valeurs exactes $x = \frac{2}{3}$ et $y = \frac{1}{15}$ on considère les valeurs approchées $x^* = 0.67$ et $y^* = 0.667$ respectivement. Les erreurs absolues correspondantes sont :

$$\Delta(x) = |x - x^*| = \left|\frac{2}{3} - 0.67\right| = \left|\frac{200-201}{3.10^2}\right| = \frac{1}{3}\,10^{-2}$$
$$\Delta(y) = |y - y^*| = \left|\frac{1}{15} - 0.07\right| = \left|\frac{100-105}{10^3}\right| = \frac{1}{3}\,10^{-2}$$

Les erreurs relatives correspondantes sont :

$$\varepsilon(x) = \frac{|x-x^*|}{|x|} = \frac{\Delta(x)}{|x|} = 0.5 \times 10^{-2} = 0.5\%$$
$$\varepsilon(y) = \frac{|y-y^*|}{|y|} = \frac{\Delta(y)}{|y|} = 5 \times 10^{-2} = 5\%$$

Ainsi, bien que les erreurs absolues soient égales, x^*est une approximation 10 fois plus précise pour x que y^* l'est pour y.

Exercice d'application 1.4

Calculer l'erreur relative correspondant aux mesures suivantes :
$x = \frac{1}{18} = 0.556$ et $y = \frac{22}{7} = 3.14285$ et aussi $z = \frac{9}{14} = 0.643$

1.2.3.1.3 Majorants des erreurs absolue et relative

Si la valeur exacte est connue on peut déterminer les erreurs absolue et relative. Mais dans la majorité des cas, elle ne l'est pas, et pour les estimer on introduit la notion de majorant de l'erreur absolue et de l'erreur relative.

Définition 1.5 : On appelle majorant de l'erreur absolue d'une valeur approchée x^*de x tout nombre réel positif noté Δx vérifiant : $\Delta(x) = |x - x^*| \leq \Delta x$.
Ou de manière équivalente : $x^* - \Delta x \leq x \leq x^* + \Delta x$, et on écrit : $x = x^* \pm \Delta x$ qui veut dire : $x \in [x^* - \Delta x;\ x^* + \Delta x]$.

Remarques 1.6

1. Plus Δx est petit, plus l'approximation x^* est précise. D'où, en pratique, on prend le plus petit Δx possible.
2. Comme $x \simeq x^*$, en pratique on prend $\varepsilon(x) \simeq \frac{\Delta x}{x^*}$ qui est un majorant de l'erreur relative de x^* et on écrit $x = x^* \pm |x^*|\,\varepsilon x$.

Bibliographie

1. Cours de licence de Mathématiques d'Analyse Numérique de Karima MEBARKI (Université Abderrahmane Mira, Béjaia).
2. Cours de licence d'Informatique de Méthode Numérique de Sarah OTSMANE (Université Batna 2, Batna).
3. Cours d'Analyse Numérique de Mazen SAAD (Ecole Centrale de Nantes).
4. Cours Arithmétique Flottante de Vincent Lefèvre, Paul Zimmermann (INRIA 2004)
5. Cours Architecture des ordinateurs : Codage binaire et hexadécimal Arithmétique des processeurs de F. Pellegrini (Université de Bordeaux).
6. Livre Algorithme Numérique C.BREZINSKI (ellipses 1988).
7. Livre Applied Numerical Linear Algebra de James W. DEMMEL (SIAM 1997).