Mathématiques discrètes

Marc-André Désautels

Table des matières

Préface

Ce document est un livre Quarto.

Pour en apprendre davantage sur les livres Quarto, visitez https://quarto.org/docs/books.

1 Systèmes de numération positionnelle

Un système de numération est un ensemble de règles qui permettent de représenter des nombres. Le plus ancien est probablement le système unaire où le symbole | représente l'entier un, || représente l'entier deux, ||| pour trois, |||| pour quatre et ainsi de suite. Ce système atteint vite ses limites, mais il permet de mettre en évidence le fait qu'il existe plusieurs façons de représenter les entiers.

Nom français	Système unaire	Système décimal	Chiffres romains
Zéro		0	
Un	|	1	I
Deux	||	2	II
Trois	|||	3	III
⋮	⋮	⋮	⋮
Douze	|||| |||| ||||	12	XII
⋮	⋮	⋮	⋮

Dans la table ci-dessus, on remarque que sur une ligne donnée, on retrouve quatre manières différentes de représenter le même entier. Pour le reste de cette section, il sera important de dissocier la **représentation** d'un nombre et sa **valeur**.

Définition 1.1 (Système de numération)**.** Un **système de numération** permet de compter des objets et de les représenter par des nombres. Un système de numération **positionnel** possède trois éléments:

- Base b (un entier supérieur à 1)
- Symboles (digits): 0, 1, 2, ..., b-1
- Poids des symboles selon la position et la base, où poids=base$^{\text{position}}$

> **Note**
>
> Lorsque plusieurs bases interviennent dans un même contexte, on écrit $(a_n \dots a_1 a_0)_b$ pour indiquer que le nombre représenté en base b.

Définition 1.2 (Représentation polynomiale)**.** Le système positionnel utilise la **représentation polynomiale**. Celle-ci est donnée par:

$$(a_n a_{n-1} \dots a_1 a_0, a_{-1} a_{-2} \dots a_{-m})_b = a_n b^n + a_{n-1} b^{n-1} + \dots + a_1 b^1 + a_0 b^0 + \dots \\ \dots + a_{-1} b^{-1} + \dots + a_{-m} b^{-m}$$

où b est la **base** et les a_i sont des **coefficients** (les symboles de votre système de numération).

1.1 Système décimal

Il s'agit du système de numération le plus utilisé dans notre société. On peut le résumer avec les trois règles suivantes.

- Base = 10
- Symboles ordonnés qu'on nomme les *chiffres* : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
- Le poids des symboles est donné par 10^{position}

Ainsi, l'écriture "197 281" signifie:

$$197\ 281 = 1 \cdot 10^5 + 9 \cdot 10^4 + 7 \cdot 10^3 + 2 \cdot 10^2 + 8 \cdot 10^1 + 1 \cdot 10^0$$

Exemple 1.1. Représentez le nombre 3482 sous une forme de numération positionnelle.

1.2 Système binaire

Ce concept est essentiel en informatique, puisque les processeurs des ordinateurs sont composés de transistors ne gérant que deux états chacun (0 ou 1). Un calcul informatique n'est donc qu'une suite d'opérations sur des paquets de 0 et de 1, appelés **bits**.

- Base = 2
- Symboles ordonnés qu'on nomme les *bits*: 0, 1
- Le poids des symboles est donné par 2^{position}

> **Important**
>
> En base 2, le *chiffre* 2 n'existe pas (c'est un **nombre**); tout comme le *chiffre* 10 n'existe pas en base 10 (c'est un **nombre**).

> **Nombres binaires en `Python`**
>
> Pour indiquer qu'un nombre est en binaire dans `Python`, il faut le faire précéder par `0b`. Pour convertir un nombre en binaire, on utilise la command `bin`.

Exemple 1.2. Quels sont les nombres qui, dans la base deux, succèdent à $(0)_2$?

```
depart = 0b0
for i in range(6):
    depart = depart + 1
    print(bin(depart))
```

```
0b1
0b10
0b11
0b100
0b101
0b110
```

Exemple 1.3. Quels sont les nombres qui, dans la base deux, succèdent à $(1110)_2$?

```
depart = 0b1110
for i in range(6):
    depart = depart + 1
    print(bin(depart))
```

```
0b1111
0b10000
0b10001
0b10010
0b10011
0b10100
```

Exemple 1.4. Convertissez le nombre $(11001)_2$ en décimal.

Exemple 1.5. Convertissez les nombres suivants en décimal.

(a) $(110)_2 =$
(b) $(101101)_2 =$
(c) $(0{,}1011)_2 =$
(d) $(110{,}101)_2 =$

1.3 Système octal

Le système de numération octal est le système de numération de base 8, et utilise les chiffres de 0 à 7. D'après l'ouvrage de Donald Knuth's, *The Art of Computer Programming*, il fut inventé par le roi Charles XII de Suède.

- Base = 8
- Symboles ordonnés qu'on nomme les *chiffres*: 0, 1, 2, 3, 4, 5, 6, 7
- Le poids des symboles est donné par 8^{position}

> **Nombres octaux en `Python`**
>
> Pour indiquer qu'un nombre est en octal dans `Python`, il faut le faire précéder par `0o`.
> Pour convertir un nombre en octal, on utilise la commande `oct`.

Exemple 1.6. Quels sont les nombres qui, dans la base 8, succèdent à $(65)_8$?

```
depart = 0o65
for i in range(12):
    depart = depart + 1
    print(oct(depart))
```

```
0o66
0o67
0o70
0o71
0o72
0o73
0o74
```

```
0o75
0o76
0o77
0o100
0o101
```

1.4 Système hexadécimal

Le système hexadécimal est utilisé notamment en électronique numérique et en informatique car il est particulièrement commode et permet un compromis entre le code binaire des machines et une base de numération pratique à utiliser pour les ingénieurs. En effet, chaque chiffre hexadécimal correspond exactement à quatre chiffres binaires (ou bits), rendant les conversions très simples et fournissant une écriture plus compacte. L'hexadécimal a été utilisé la première fois en 1956 par les ingénieurs de l'ordinateur Bendix G-15.

- Base = 16
- Symboles ordonnés qu'on nomme les *chiffres*: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
- Le poids des symboles est donné par 16^{position}

On remarque qu'en base 16, les dix chiffres de 0 à 9 ne suffisent pas. Il faut donc se doter de 6 symboles additionnels. On utilise les lettres de A à F avec la signification suivante:

$$(A)_{16} = (10)_{10}, \quad (B)_{16} = (11)_{10}, \quad (C)_{16} = (12)_{10}$$

$$(D)_{16} = (13)_{10} \quad (E)_{16} = (14)_{10}, \quad (F)_{16} = (15)_{10}$$

> Nombres hexadécimaux en `Python`
>
> Pour indiquer qu'un nombre est en hexadécimal dans `Python`, il faut le faire précéder par `0x`.
> Pour convertir un nombre en hexadécimal, on utilise la commande `hex`.

Exemple 1.7. Quels sont les nombres qui, dans la base 16, succèdent à $(\text{AAA})_{16}$?

```python
depart = 0xAAA
for i in range(12):
    depart = depart + 1
    print(hex(depart))
```

```
0xaab
0xaac
0xaad
0xaae
0xaaf
0xab0
0xab1
0xab2
0xab3
0xab4
0xab5
0xab6
```

Exemple 1.8. Trouvez la représentation en base 10 de:

a) $(\text{AB0})_{16}$
b) $(\text{214,EA})_{16}$

> **!** Important
>
> Pour convertir un nombre de la base b vers la base 10 (décimal), on trouve sa représentation polynomiale.

> Conversion des nombres entiers vers décimal en `Python`
>
> Pour convertir un nombre entier nb représenté dans la base b en `Python` en décimal, on utilise la commande `int(nb, b)`. Le nombre entier nb doit être représenté comme une chaîne de caractères (**string**).
> Par exemple, si vous avez le nombre hexadécimal $A0F$, vous le convertissez de la manière suivante:

```
int('A0F', 16)
```

```
2575
```

1.5 Division entière

Définition 1.3 (Divisibilité)**.** Si $a \in \mathbb{Z}$, $b \in \mathbb{Z}$ et $a \neq 0$, on dit que a **divise** b s'il existe un entier c tel que $b = ac$. L'entier a est alors appelé **facteur** de b.

Si a **divise** b, nous le notons $a \mid b$.

Théorème 1.1 (Divisibilité)**.** *Soit a, b et c des nombres entiers quelconques, avec $a \neq 0$.*

1. *Si $a \mid b$ et $a \mid c$ alors $a \mid (b + c)$ et $a \mid (b - c)$.*
2. *Si $a \mid b$ alors $a \mid (bc)$.*
3. *Si $a \mid b$ et $b \mid c$ alors $a \mid c$.*

Exemple 1.9. Vrai ou faux? Justifiez en invoquant une définition, un théorème, en donnant une preuve ou un contre-exemple.

a) $7 \mid 10$
b) $-5 \mid 10$
c) $100 \mid 10$
d) $5 \mid -10$

Théorème 1.2. *Soit a et d des entiers, avec $d > 0$. Il existe une seule paire d'entiers q et r satisfaisant*

$$0 \leq r < d \quad et \quad a = dq + r$$

Définition 1.4 (Diviseur, dividende, quotient, reste)**.** Considérons a et d des entiers, avec $d > 0$. Le Théorème **??** stipule qu'il existe une seule paire d'entiers q et r satisfaisant

$$a = dq + r \quad \text{et} \quad 0 \leq r < d$$

Par exemple, si $a = 17$ et $d = 3$, on a

$$17 = 3 \cdot 5 + 2 \quad \text{et} \quad 0 \leq 2 < 3$$

- L'entier $d = 3$ est appelé **diviseur**.
- L'entier $a = 17$ est appelé le **dividende**.
- L'entier $q = 5$ est appeléle **quotient** (notation: $q = a$ **div** d).
- L'entier $r = 2$ est appelé le **reste**.

1.6 Conversions de la base 10 vers une base b

Pour convertir un nombre entier de la base 10 vers une base b, il faut effectuer de façon successive des divisions en utilisant la Définition **??**. Les restes des divisions successives correspondent aux coefficients de la représentation polynomiale (**lire de base en haut**).

1.6.1 Conversions vers binaire

Exemple 1.10. Convertissez les nombres suivants en binaire.

a) 115
b) 71

Nous pouvons utiliser la commande `bin` de `Python` pour convertir des **entiers** décimaux en binaire.

```
print(bin(115))
print(bin(71))
```

```
0b1110011
0b1000111
```

Pour convertir un nombre fractionnaire en binaire, il suffit de multiplier (plutôt que de diviser) la partie fractionnaire en notant les parties entières et fractionnaires obtenues. Il faut ensuite répéter ces étapes avec la nouvelle partie fractionnaire et poursuivre le processus jusqu'à ce que la partie fractionnaire soit nulle. Les parties entières des résultats de ces produits correspondent aux coefficients de la représentation polynomiale (**lire de haut en bas**).

Exemple 1.11. Convertissez les nombres suivants en binaire.

a) $(0{,}8125)_{10}$
b) $(0{,}15)_{10}$

> **!** Important
>
> La conversion en binaire ou en n'importe quelle base ne donne pas toujours une suite finie. Si c'est un nombre rationnel, la conversion donnera toujours une suite finie ou périodique.

Exemple 1.12. Convertissez en binaire les nombres suivants, en ne conservant que 6 chiffres pour la partie fractionnaire, au besoin.

a) $(51{,}375)_{10}$
b) $(564{,}32)_{10}$

1.6.2 Conversions vers octal

Nous pouvons utiliser la command `oct` de `Python` pour convertir des **entiers** décimaux en octal.

```
print(oct(115))
print(oct(71))
```

```
0o163
0o107
```

1.6.3 Conversions vers hexadécimal

Exemple 1.13. Convertissez les nombres décimaux suivants en hexadécimal.

a) $(176{,}47)_{10}$
b) $(69{,}28)_{10}$

Nous pouvons utiliser la command `hex` de `Python` pour convertir des **entiers** décimaux en hexadécimal.

```
print(hex(115))
print(hex(71))
```

```
0x73
0x47
```

1.6.4 Conversions binaire - hexadécimal

Une des raisons pour lesquelles le format hexadécimal a été inventé est qu'il est particulièrement simple de convertir un nombre binaire en nombre hexadécimal et inversement.

	Hexa	0	1	2	3	4	5	6	7
Binaire	0000	0001	0010	0011	0100	0101	0110	0111
Hexa	8	9	A	B	C	D	E	F
Binaire	1000	1001	1010	1011	1100	1101	1110	1111

Pour convertir un nombre binaire, on regroupe par *paquets* de 4 chiffres à partir de la virgule (pour la partie entière et la partie fractionnaire).

Exemple 1.14. Convertissez les nombres binaires suivants en hexadécimal.

a) $(111001,1101)_2$
b) $(1110001,11\overline{001})_2$

Exemple 1.15. Convertissez les nombres hexadécimaux suivants en binaire.

a) $(537,14)_{16}$
b) $(45B,1\overline{DE})_{16}$

2 Représentation des nombres dans l'ordinateur

Lorsque nous voulons représenter des nombres dans un ordinateur, il faut distinguer deux cas biens différents; la représentation des nombres **entiers** et la représentation des nombres **fractionnaires**.

2.1 Représentation des entiers

En `Python`, contrairement à la plupart des langages informatiques, les entiers sont représentés avec une précision **infinie**. C'est-à-dire que la seule limite correspond à la mémoire interne de la machine que vous utilisez. Cependant, dans la majorité des langages informatiques, la précision de la représentation des entiers est **finie**, c'est-à-dire qu'un certain nombre de bits est alloué en mémoire pous stocker votre nombre et vous ne pouvez pas le dépasser.

Nous pouvons connaître le nombre de bits utilisés par `Python` dans la représentation d'un entier en utilisant la fonction `getsizeof` du module `sys`.

```
from sys import getsizeof

n1 = 2**32
n2 = 2**128
print(getsizeof(n1), getsizeof(n2))
```

```
32 44
```

Pour étudier le comportement d'entiers ayant une taille fixe, on peut utiliser le module `numpy`. Ce module possède plusieurs classes d'entiers à taille fixe.

2.1.1 Entiers non signés

Définition 2.1 (Entiers non signés (nombres positifs))**.** Un nombre **entier non signé** (positif) est représenté par un nombre de bits préalablement fixé. Au besoin, on complète le nombre par des zéros à gauche fin d'avoir le nombre total de bits choisi.

Les entiers non signés à taille fixe en `Python`

- `numpy.ubyte`: entier non signé sur 8 bits
- `numpy.ushort`: entier non signé sur 16 bits
- `numpy.uintc`: entier non signé sur 32 bits
- `numpy.uint`: entier non signé sur 64 bits

Exemple 2.1. Transformez les entiers décimaux suivants en entiers non signés sur un octet (huit bits).

a) 143

b) 15
c) 30

```
import numpy as np

print(bin(np.ubyte(143)), bin(np.ubyte(15)), bin(np.ubyte(30)))
```

```
0b10001111 0b1111 0b11110
```

Soyez prudents!

Si on tente d'écrire un nombre entier qui dépasse la capacité du format, nous n'obtenons pas nécessairement un message d'erreur, il faut donc être très prudents. Par exemple, le format `numpy.byte` peut représenter les entiers de 0 à 255. Si nous tentons de représenter 256, nous obtenons:

```
import numpy as np

print(np.uint8(256))
```

```
0
```

Ce genre d'erreur est appelée un dépassement d'entier. Un dépassement d'entier (*integer overflow*) est, en informatique, une condition qui se produit lorsqu'une opération mathématique produit une valeur numérique supérieure à celle représentable dans l'espace de stockage disponible. Par exemple, l'ajout d'une unité au plus grand nombre pouvant être représenté entraîne un dépassement d'entier.

Le dépassement d'entier le plus célèbre de ces dernières années est très probablement celui qui causa la destruction de la fusée Ariane 5, lors de son vol inaugural, le 4 juin 1996.

Exemple 2.2. Quel est le plus grand entier non signé pouvant être représenté avec:

a) 8 bits?
b) 32 bits?
c) n bits?

2.1.2 Entiers signés

Pour travailler avec des entiers qui peuvent être positifs ou négatifs, il faut inclure le signe du nombre dans sa représentation, et l'on parle alors d'entiers signés.

Définition 2.2 (Entiers signés (représentation signe et module))**.** Un nombre **entier signé** (généralement représenté dans un octet) est un nombre où le 1er bit (à gauche) est réservé au signe, et les autres bits permettent d'indiquer la valeur absolue du nombre. Pour indiquer qu'un nombre est positif (+), le 1er bit est `0`, et pour un nombre négatif (-), le 1er bit est `1`.

Les entiers signés à taille fixe en `Python`

- `numpy.byte`: entier signé sur 8 bits
- `numpy.short`: entier signé sur 16 bits

- `numpy.intc`: entier signé sur 32 bits
- `numpy.int_`: entier signé sur 64 bits

Exemple 2.3. Complétez les tableaux suivants qui indiquent la représentation signe et module sur 4 bits.

Base 2	Base 10
0000	
0001	
0010	
0011	
0100	
0101	
0110	
0111	

Base 2	Base 10
1000	
1001	
1010	
1011	
1100	
1101	
1110	
1111	

En utilisant les nombres entiers signés:

- On peut écrire autant de nombres positifs que de négatifs.
- Pour un nombre exprimé avec n bits, les valeurs extrèmes sont $\pm(2^{n-1}-1)$

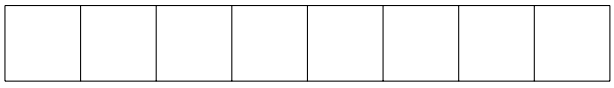
Exemple 2.4. Quelles sont les valeurs extrèmes pour des entiers signés représentés sur 4 bits?

⚠ Inconvénients de la représentation signe et module

- Il y a deux zéros! Un *zéro* positif (0000 0000) et un *zéro* négatif (1000 0000).
- Les opérations arithmétiques ne se font pas de la même manière qu'habituellement. Par exemple, sur 4 bits:
  - **Base 2**: 0100 + 1011 = 1111
  - **Base 10**: +4 + -3 = -7! (**FAUX!**)

Exemple 2.5. Écrivez la représentation signe et module sur 8 bits de:

a) 15



a) -15