

Information, Calcul et Communication

Représentation de l'information

R. Boulic

Objectif du cours d'aujourd'hui

Répondre aux questions suivantes :

- ▶ Existe-t-il une représentation universelle de l'information ?
- ▶ Par quels moyens peut on représenter des symboles et des nombres ?
- ▶ Est-il possible de construire une représentation exacte du monde réel ?

Plan du cours d'aujourd'hui

Lien avec les Leçons précédentes :

- ▶ Rappel des domaines d'applications
- ▶ Une représentation est une convention
- ▶ Vers l'unité élémentaire d'information (exercices)

Manipulation sur les nombres entiers :

- ▶ Opérations et domaine couvert

La virgule flottante : Pourquoi ? Comment ?

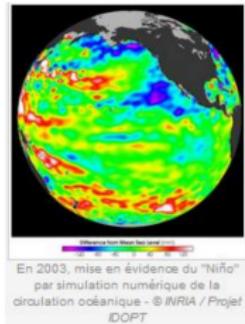
- ▶ Un exemple qui pose problème

Retour à la représentation des symboles

- ▶ De l'alphabet aux idéogrammes

Lien avec les leçons précédentes

Domaines d'application



Calcul scientifique /Simulation

-> *nombres*

Conduite de processus

-> *signaux
(mesures,
contrôle...)*



Gestion d'information

-> *texte*



Google datacenter

Une représentation est une convention

- ▶ pour faciliter l'activité d'un groupe d'utilisateurs
- ▶ correspondance entre un ensemble de signes et leur signification.



☞ Fragilité de cette convention : langues mortes, codes perdus,etc...

Il n'existe pas de représentation universelle :

- ▶ standard de facto = porté par le marché, l'usage (ex : pdf)
- ▶ standard de jure = normalisation (IEEE, ACM, ISO...).

Exemples : alphabet romain, chiffres arabes, code de la route, papier monnaie

Vers l'unité élémentaire d'information



214 motifs graphiques, appelés des clefs, ont été utilisés pour construire ~100.000 idéogrammes chinois

Les **26 lettres** de l'alphabet latin ont été utilisés pour construire ~1.000.000 mots des langues occidentales

A B C ...

0 1 2 3... Les **10 chiffres** indo-arabes permettent de construire une infinité de nombres (et même d'encrypter tous les mots!)

Question: quel est le système de signes le plus simple permettant de conserver la même richesse d'expression que les 10 chiffres ?

Réponse: un système de **2 symboles**

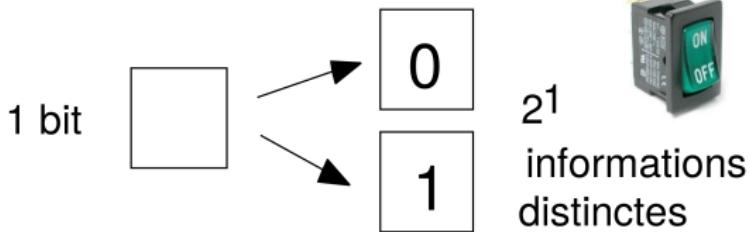
Unité élémentaire d'information : le bit

Toute information peut être représentée à l'aide d'un ensemble **d'éléments binaires**

Par convention, un **élément binaire** vaut **0** ou **1**.

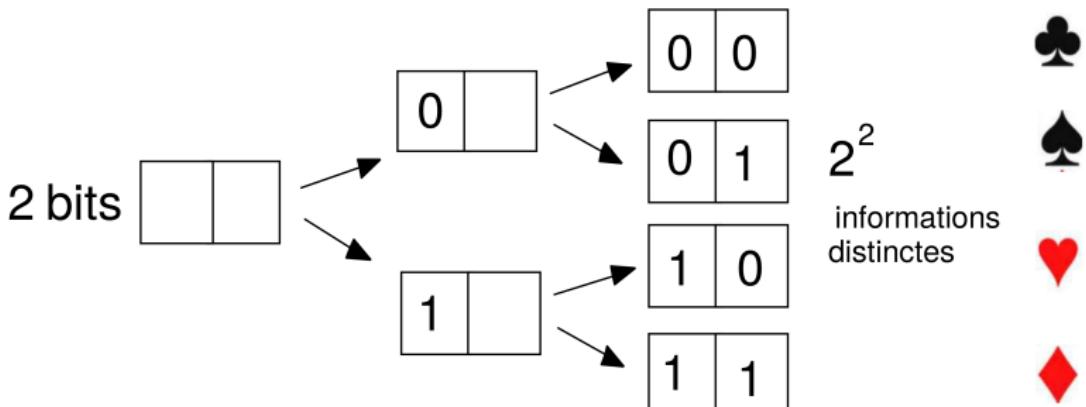
Par la suite on utilise l'expression anglaise “**binary digit**” ou **bit** en abrégé

Dans cette leçon nous faisons abstraction de la manière dont les éléments binaires sont réalisés (états magnétiques, tensions, courants, etc..). Cela sera abordé dans le Module3.



Capteurs à seuil

Comment représenter plus d'informations ?



n bits permettent de représenter 2^n informations distinctes

n	2^n
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
10	1024
20	1048576
30	1073741824
32	4294967296

Bonne pratique pour estimation rapide:

$$2^{10} = \text{kibi (Ki)} \approx 10^3 = \text{kilo (k)}$$

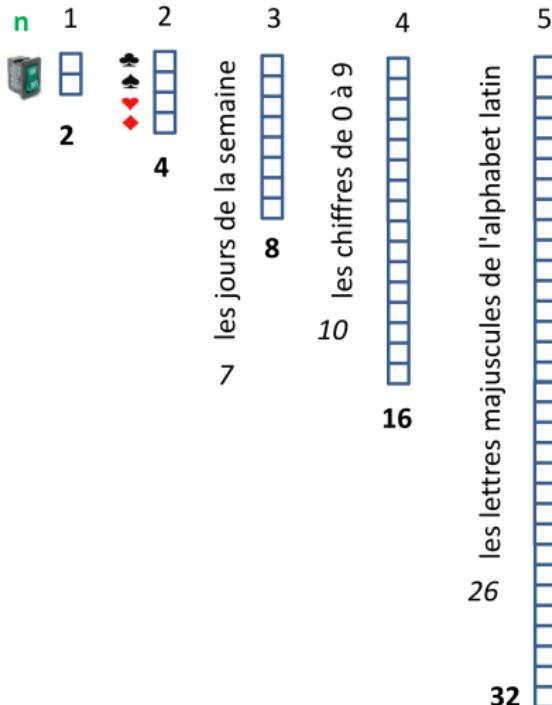
$$2^{20} = \text{mébi (Mi)} \approx 10^6 = \text{méga (M)}$$

$$2^{30} = \text{gibi (Gi)} \approx 10^9 = \text{giga (G)}$$

$$2^{32} = 2^{30+2} = 2^{30} \cdot 2^2 \approx 4 \text{ G}$$

n bits permettent de représenter 2^n informations distinctes

réciproquement, 2^n informations distinctes sont représentables par $\log_2(2^n) = n \log_2(2) = n$ bits



Exercice:

Combien de bits suffisent pour représenter :

- les jours de la semaine :
- les chiffres de 0 à 9 :
- les lettres de l'alphabet:
 - Majuscules
 - Minuscules + Majuscules
 - Min + Maj + chiffres + signes ...

règle générale:

Pour K informations distinctes, le **nombre de bits n** suffisant pour représenter ces informations est l'entier supérieur ou égal à $\log_2 K$

Organisation de l'information

Convention: on appelle **byte** un groupe de 8 bits (**octet**).

Les représentations les plus courantes pour l'information exploitent un octet ou une séquence d'octets.



Plan

Lien avec les Leçons précédentes :

- ▶ Rappel des domaines d'applications
- ▶ Une représentation est une convention
- ▶ Vers l'unité élémentaire d'information (exercices)

☞ Manipulation sur les nombres entiers :

- ▶ Opérations et domaine couvert

La virgule flottante : Pourquoi ? Comment ?

- ▶ Un exemple qui pose problème

Retour à la représentation des symboles

- ▶ De l'alphabet aux idéogrammes

Comment représenter un nombre entier ?

Commençons par les entiers naturels (=positifs et nul)

Rappel: tout nombre peut être représenté à l'aide d'un ensemble d'éléments binaires.

Définition: une suite de 0 et de 1 est appelé un **motif binaire**

*un motif binaire isolé est insuffisant
pour comprendre ce qui est codé*

- Il faut en plus une méthode d'interprétation du motif binaire en tant que **donnée (data)**
- Une solution: la notation positionnelle des nombres

Notation positionnelles des nombres

Exemple d'un nombre entier en base 10 :

le nombre **703** est la notation abrégée de l'expression :

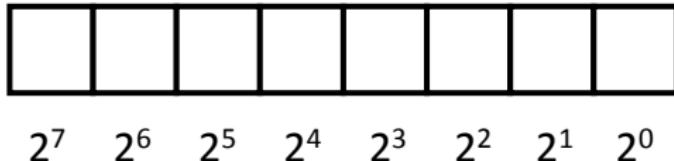
$$7 \cdot 10^2 + 0 \cdot 10^1 + 3 \cdot 10^0$$

- ▶ Le chiffre de droite est toujours associé à la puissance 0 de la base 10
- ▶ La puissance de la base augmente d'une unité de chiffre en chiffre, en allant de la droite vers la gauche

Cette convention de **notation positionnelle** peut être exploitée dans n'importe quelle base

Représentation *positionnelle* en base 2

repose sur les mêmes conventions qu'en base dix (décimal)

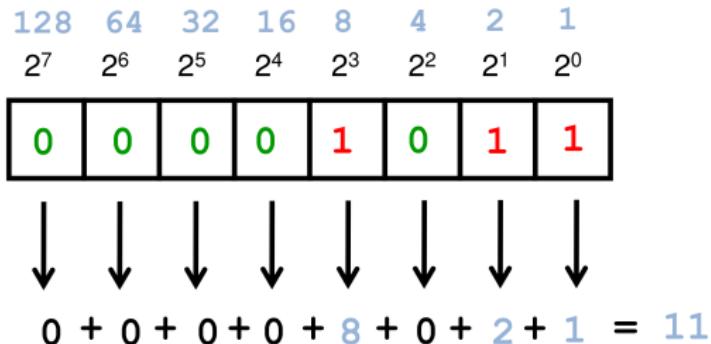


poids forts à gauche,

poids faibles à droite

Pratique: Conversions

Du **binnaire** vers le **décimal**:
additionner les puissances de 2 présentes dans le motif binaire



Du **décimal** vers le **binnaire** :
décomposer un nombre entier X en une somme de puissances de 2 par division entières successives tant que le quotient ≥ 2

$$\begin{aligned} 11 &= 2 \cdot 5 + 1 \\ &= 2 \cdot (2 \cdot 2 + 1) + 1 \\ &= 2 \cdot (2 \cdot (2 \cdot 1 + 0) + 1) + 1 \\ &= 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 \\ &= 1011 \end{aligned}$$

Entiers: domaine couvert

Une représentation destinée à une machine est associée à une capacité fixe exprimée en nombre de bits (d'octets).

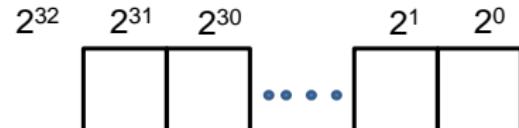
Ex: entier manipulé par une machine « **32 bits** ». Cette machine dispose d'instructions pour réaliser très rapidement les opérations de base (addition, multiplication, etc) pour des entiers représentés sur 4 octets (max).

Donc limitation du nombre d'entiers représentables = 2^{32}

Si la représentation est totalement destinée au nombres positifs, son domaine couvert est alors, pour 32 bits :

Min = motif binaire avec des 0 partout = **zéro**

Max = motif binaire avec des 1 partout = $2^{32} - 1$



Entiers: domaine couvert (2)

Les calculs sur des entiers sont exacts si le résultat désiré est un entier et appartient au domaine couvert

La représentation choisie doit tenir compte de l'ensemble des résultats possibles

Plusieurs causes possibles de dépassement de capacité:

- division entière: perte de partie fractionnaire
- multiplication, addition, soustraction: propagation de retenue au delà de 2^{31}

Exemples de dépassement de capacité

Exemple 1 : addition de 2 entiers sur **un seul bit** ➡ 4 cas :

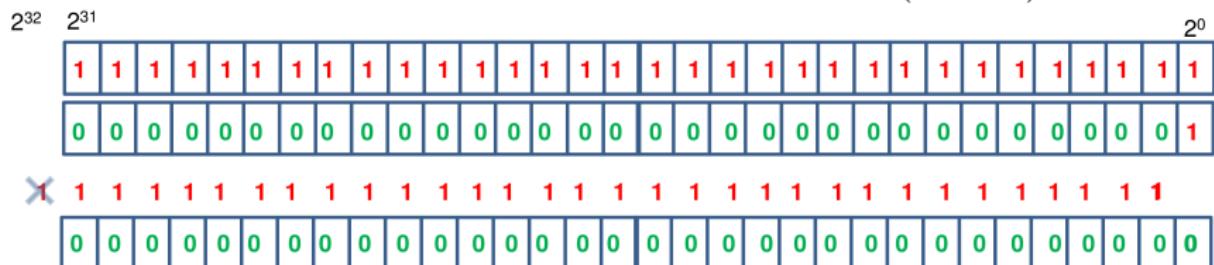
The diagram illustrates four distinct paths, each starting and ending at different values, with intermediate points indicated by vertical lines and labels:

- Path 1: Starts at 0 (green), ends at 0 (green).
- Path 2: Starts at 0 (green), goes up to +1 (red), down to - (dashed), and ends at 1 (red).
- Path 3: Starts at 1 (red), goes down to 0 (green), and ends at 1 (red).
- Path 4: Starts at 1 (red), goes up to +1 (red), down to - (dashed), and ends at 10 (blue-green).

la retenue est perdue

- ☞ la retenue est perdue dans le dernier cas : dépassement de capacité

Exemple 2 : addition sur **32 bits** : dépassement sur $(2^{32} - 1) + 1$:



Entiers négatifs

représentation par signe et valeur absolue

Le **signe** d'un nombre est une information binaire + ou -
il suffit d'un bit pour le représenter
par convention: **0** pour + , **1** pour -

Exercice: quelles sont les conséquences de la représentation suivante d'un entier signé sur 8 bits, avec **signe** et **valeur absolue** :

signe	2^6	2^5	2^4	2^3	2^2	2^1	2^0

- > symétrie parfaite du domaine couvert mais 2 représentations pour 0
- > la soustraction ne peut pas être effectuée en additionnant l'opposé d'un nombre

Entiers : dépassement de capacité et représentation des nombres négatifs

Question: comment tirer partie d'une capacité limitée de n bits pour en déduire une représentation des entiers négatifs permettant de remplacer la soustraction par l'addition de l'opposé ?

Rappel: n bits permettent de représenter 2^n nombres entiers positifs de 0 à $(2^n - 1)$ quand tous les bits sont à 1.

La valeur 2^n elle-même n'est pas représentable sur n bits, on a:

$$(2^n - 1) + 1 = 2^n \quad (\text{en théorie})$$

Mais $(2^n - 1) + 1 = 0 \quad (\text{sur } n \text{ bits})$

Conséquence: le motif binaire de $(2^n - 1)$ est une bonne représentation de -1 car on obtient 0 quand il est ajouté à 1

Représentation des entiers signés

Propriétés à vérifier: si a et b sont deux nombres opposés

Alors :

$$a + b = 0$$

de plus $-(-a) = a$

Avec n bits de capacité, on pose que l'opposé d'un nombre x est donné par l'expression $2^n - x$ appelée le Complément à 2 de x .

donc $a + b = (2^n - b) + b = 2^n = 0 \quad (\text{sur } n \text{ bits})$

$$-(-a) = 2^n - (2^n - a) = a$$

Pratique: comment calculer l'opposé ($2^n - x$) d'un entier x ?

Une transformation est nécessaire pour :

- faire apparaître des quantité représentables sur n bits
- les manipuler à l'aide d'opérations simples

$$\begin{aligned} & 2^n - x \\ &= 2^n \underline{-1 + 1} - x \\ &= (\underline{(2^n - 1)} - x) + 1 \\ &\quad \downarrow \\ &\boxed{ \begin{array}{|c|c|c|c|c|c|c|c|} \hline 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \hline \end{array}} \end{aligned}$$

Cas particulier: $((2^n - 1) - x)$ est très facile à obtenir !

Il suffit d'inverser chaque bit de x : $0 \rightarrow 1$ ou $1 \rightarrow 0$

Cette valeur est appelée le Complément à 1 de x .

Complément à 2 de x = Complément à 1 de x + 1

Exemple1: Complément à 1 du nombre onze en binaire = (2^n-1) - onze

0	0	0	0	1	0	1	1
---	---	---	---	---	---	---	---

motif binaire de onze

1	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---

2^n-1

0	0	0	0	1	0	1	1
---	---	---	---	---	---	---	---

soustraction de onze

1	1	1	1	0	1	0	0
---	---	---	---	---	---	---	---

complément à 1 de onze

Exemple2: calcul de l'opposé de onze avec son complément à 2=(complément à 1)+1

<table border="1"><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td></tr></table>	0	0	0	0	1	0	1	1	motif binaire de onze								
0	0	0	0	1	0	1	1										
<table border="1"><tr><td>↓</td><td>↓</td><td>↓</td><td>↓</td><td>↓</td><td>↓</td><td>↓</td><td>↓</td></tr><tr><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td></tr></table>	↓	↓	↓	↓	↓	↓	↓	↓	1	1	1	1	0	1	0	0	complément à 1 de onze
↓	↓	↓	↓	↓	↓	↓	↓										
1	1	1	1	0	1	0	0										
<table border="1"><tr><td>+</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td></tr></table>	+	0	0	0	0	0	0	0	1	addition de + 1							
+	0	0	0	0	0	0	0	1									
<hr/>																	
<table border="1"><tr><td>=</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td></tr></table>	=	1	1	1	1	0	1	0	1	complément à 2 de onze = opposé de onze							
=	1	1	1	1	0	1	0	1									
<table border="1"><tr><td>+</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td></tr></table>	+	0	0	0	0	1	0	1	1	addition de onze							
+	0	0	0	0	1	0	1	1									
<hr/>																	
<table border="1"><tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0 sur n bits
1	1	1	1	1	1	1	1										
0	0	0	0	0	0	0	0										

Domaine couvert avec le complément à 2

Bit de poids fort = signe

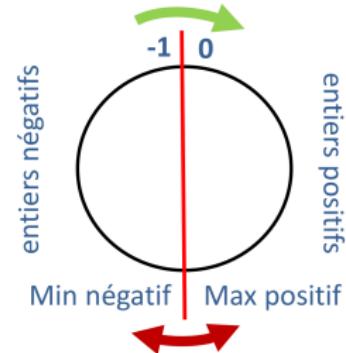
Min positif = 00000...0000 = 0

Max positif = 01111...1111

Min négatif = 10000...0000

Max négatif = 11111...1111 = -1

retenure ignorée: $1 + (-1) = 0$



violation du domaine couvert

Dépassemement de capacité: aucun problème quand l'addition d'un entier positif et d'un entier négatif donne un résultat positif

Violation du domaine couvert: changement incorrect du bit de signe quand on additionne 2 entier positifs ou 2 entiers négatifs

25 / 50

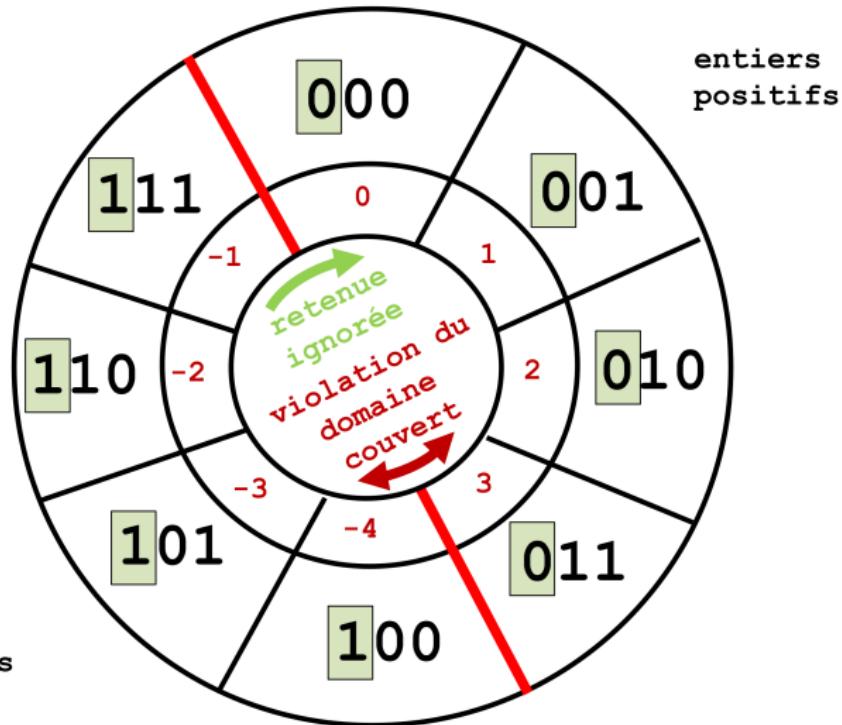
Entiers signés : domaine couvert

Exemple:
sur 3 bits



signe

entiers
négatifs



entiers
positifs

Plan

Lien avec les Leçons précédentes :

- ▶ Rappel des domaines d'applications
- ▶ Une représentation est une convention
- ▶ Vers l'unité élémentaire d'information (exercices)

Manipulation sur les nombres entiers :

- ▶ Opérations et domaine couvert

☞ **La virgule flottante : Pourquoi ? Comment ?**

- ▶ Un exemple qui pose problème

Retour à la représentation des symboles

- ▶ De l'alphabet aux idéogrammes

Un exemple qui pose problème

La mise en œuvre d'un algorithme doit aussi tenir compte du choix de représentation des nombres.

Exemple : $a \cdot x^2 + b \cdot x + c$

avec les valeurs *décimales* $a = 0.25$, $b = 0.1$, $c = 0.01$

Discriminant : $\Delta = b^2 - 4ac$

en théorie Δ est nul pour cet exemple

Mais sur machine (représentation et calculs sur 64 bits) :

$$\Delta = 9.02 \cdot 10^{-19}$$

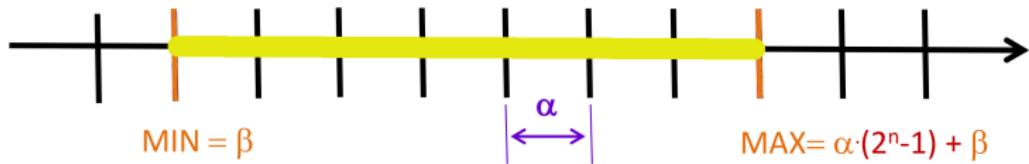
☞ Pourquoi ?

Représentation à virgule fixe

Le domaine couvert des entiers positifs avec n bits est $[0, 2^{n-1}]$

Ce domaine peut être mis à une autre échelle α et décalé de β pour représenter des nombres à virgule.

En représentation **à virgule fixe** sur n bits, les 2^n valeurs représentées sont *uniformément réparties* dans le nouvel intervalle $[\text{MIN}, \text{MAX}]$ et séparées par la quantité α :



Problème : il existe un nombre infini de nombres à virgule à l'intérieur de $[\text{MIN}, \text{MAX}]$ qui sont donc représentés de façon approchée par l'une des 2^n valeurs représentées.

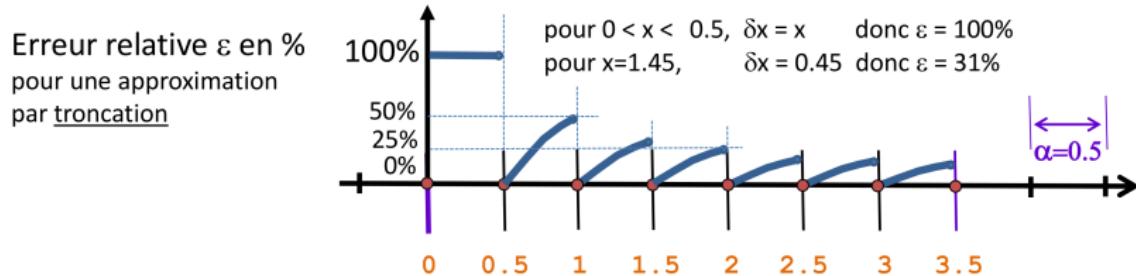
- ☞ production d'une **erreur absolue de quantification** (ou « de discréttisation ») au maximum égale à α .

Erreurs relatives sur le domaine couvert

L'erreur relative décrit l'importance relative de l'erreur de discréétisation δx par rapport au nombre à représenter x .
 $|\delta x|/|x|$ n'est pas uniforme sur le domaine couvert.

Exemple: avec 3 bits et $\alpha = 0.5$, le domaine couvert est $[0, 3.5]$. Seules 8 valeurs sont représentées exactement: $0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5$.

L'erreur relative (en %) est importante lorsque δx est grand par rapport à x .



Cette répartition hétérogène de l'erreur relative n'est pas acceptable pour de nombreux problèmes

Erreurs relatives uniformes ?

Le compromis retenu: garantir une erreur relative uniforme veut dire accepter une croissance de l'erreur de discréétisation au sein du domaine couvert !

Inspiration: notation scientifique en base 10 avec un nombre fixe de chiffres significatifs.

Exemples avec 3 chiffres significatifs :

3.1415 s'écrit $3.14 \cdot 10^0$

0.0125 s'écrit $1.25 \cdot 10^{-2}$

7354 s'écrit $7.35 \cdot 10^3$

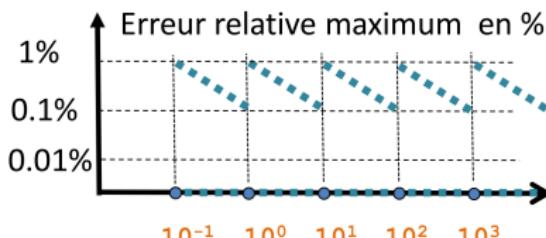
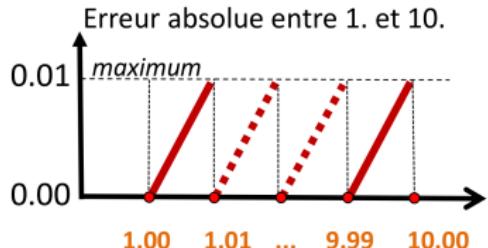
l'erreur relative est (presque) uniforme.

le pire des cas est en début d'intervalle

$[10^k, 10^{k+1}[$ lorsque $\delta x = 0.01 \cdot 10^k$

pour $x = 1.00999.. \cdot 10^k$

ce qui donne $\varepsilon = \delta x / x \approx 0.01 = 1\%$



La représentation en virgule flottante

= une notation scientifique en base 2

Représentation flottante en base 2: comporte 3 parties qui se partagent le nombre de bits à disposition: le **signe**, l'**exposant** de la base 2 et le **nombre normalisé** en base 2. La partie fractionnaire du nombre normalisé est appelée la **mantisso**.

Particularité de la base 2: le chiffre le plus significatif du nombre normalisé est constant et toujours égal à 1. Il est donc implicite.

$$\text{signe} \cdot 2^{\text{exposant}} \cdot 1, \text{mantisso}$$

Comme pour la notation scientifique, l'erreur relative maximum est définie par la puissance la plus faible de la mantisse

La représentation en virgule flottante

Exemple avec 2 bits pour l'exposant et 2 bits pour la mantisse. On a la forme: $2^{\square\Box} \cdot 1.\square\square$. L'erreur relative est au plus de $2^{-2} = \frac{1}{4}$.

exp : 00

01

10

11

mantisse

00

00

00

00

01

01

01

01

10

10

10

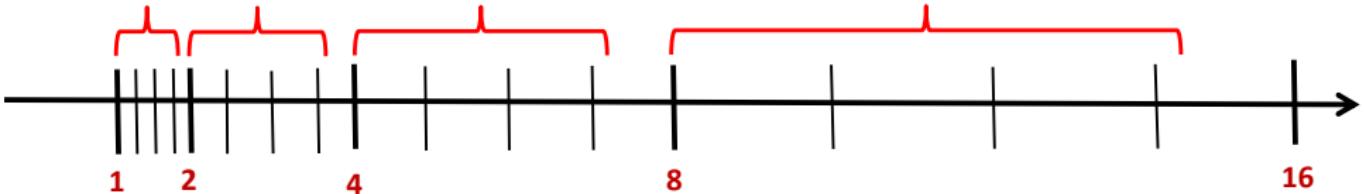
10

11

11

11

11



La représentation de 0 en virgule flottante

Même exemple mais pour inclure 0 on traite *la plus petite puissance de la base*, notée P , comme un cas particulier qui permet de couvrir l'intervalle $[0, 2^{P+1}]$ avec la formule: $2^{P+1} \cdot 0$, \square

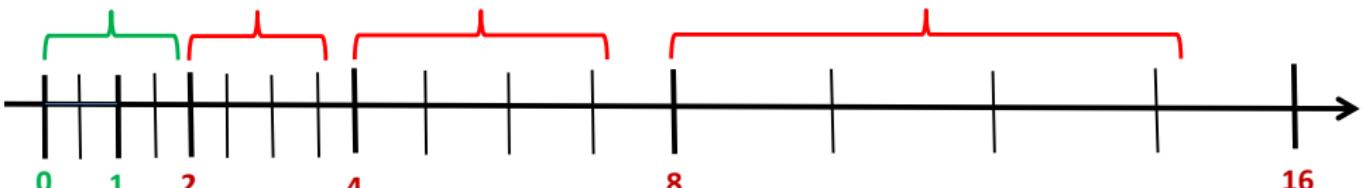
exp : 00 01 10

11

mantisse
00 00 00
01 01 01
10 10 10
11 11 11

00 01 10
11

Note : $P = 0$ ici;
mais en réalité P est
négatif, typiquement
 $-(2^{|exp|-1} - 1)$.



L'erreur d'arrondi est la règle

On appelle aussi **erreur d'arrondi** l'écart entre un nombre x et sa représentation approchée en virgule flottante.

Exercice : quel est l'écriture binaire de « un dixième » ?

0.000110011001100110011001100110011...

- ☞ « un dixième » ne peut pas être représenté de manière exacte avec un nombre fini de bits

Remarques :

- ▶ Ce problème existe dans toutes les bases, pensez à « un tiers » en base 10.
- ▶ Même si l'erreur d'arrondi est inévitable pour la majorité des nombres, on peut **garantir qu'elle se situe en dessous d'un seuil défini** par les besoins du problème traité, en choisissant une représentation adaptée.

Un exemple qui pose problème (le retour)

Exemple: $a \cdot x^2 + b \cdot x + c$

avec les valeurs décimales $a=0.25$, $b=0.1$, $c=0.01$

Discriminant $\Delta = b^2 - 4ac$

Mais $\Delta = 9.02 \cdot 10^{-19}$ (représentation et calculs sur 64 bits)

- 4 et 0.25 sont représentés exactement, leur produit donne 1.
- Par contre, approximations faites sur les décimaux 0.1 et 0.01
- Conséquence sur les calculs en binaire : $0.1 * 0.1 \neq 0.01$

Conséquence des erreurs d'arrondi

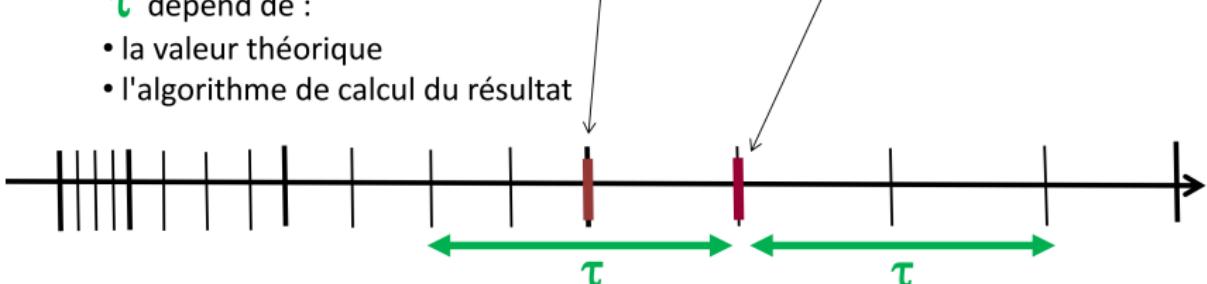
Tester l'EGALITE d'un résultat en virgule flottante vis-à-vis d'UNE SEULE valeur théorique est une absurdité. Le résultat est en général différent de la valeur théorique.

Le test d'égalité doit être redéfini à l'aide d'une tolérance variable τ AUTOUR de la valeur_théorique. Il y a égalité si le résultat appartient à cet intervalle:

$$| \text{résultat} - \text{valeur_théorique} | < \tau$$

τ dépend de :

- la valeur théorique
- l'algorithme de calcul du résultat



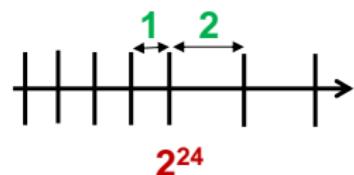
Conséquence des erreurs d'arrondi (2)

Le résultat est différent selon l'ordre des opérations. l'addition n'est plus associative:

Il existe des valeurs a , b , c telles que $(a + b) + c \neq a + (b + c)$

Exemple avec un standard sur 32 bits possédant 23 bits de mantisse:

$$(2^{24} + 1) + 1 \rightarrow 2^{24} + 1 \rightarrow 2^{24}$$
$$2^{24} + (1 + 1) = 2^{24} + 2 \text{ qui est représenté}$$



Bonne pratique: d'abord additionner les petits nombres entre eux avant de les additionner aux plus grands

La maîtrise de la précision est possible

Pour un problème donné et son algorithme de résolution, il est important de se poser les questions suivantes:

- De quelle précision ai-je besoin pour mes résultats ?
- Quelle est l'influence de l'algorithme sur la précision des résultats ?
- Quelle est la *précision maximum* disponible sur la machine cible ?

En cas de précision insuffisante, il faut reconSIDéRer l'algorithme de résolution et/ou adapter la représentation pour obtenir une *précision désirée*.

Compromis Précision / Coûts calcul et mémoire

Plan

Lien avec les Leçons précédentes :

- ▶ Rappel des domaines d'applications
- ▶ Une représentation est une convention
- ▶ Vers l'unité élémentaire d'information (exercices)

Manipulation sur les nombres entiers :

- ▶ Opérations et domaine couvert

La virgule flottante : Pourquoi ? Comment ?

- ▶ Un exemple qui pose problème

☞ Retour à la représentation des symboles

- ▶ De l'alphabet aux idéogrammes

Comment représenter un alphabet ?

Ensemble fini de signes

Considéré avec des variantes : Majuscule / minuscule

Associé aux symboles des chiffres, de ponctuation

Convention la plus large possible est nécessaire:

Table ASCII de base codifie 2^7 caractères

American Standard Code for Information Interchange

<http://www.asciitable.com/>

Code ASCII

0 0 000 NUL (null)	32 20 040 SPACE	64 40 100 @	96 60 140 `
1 1 001 SOH (start of heading)	33 21 041 !	65 41 101 A	97 61 141 a
2 2 002 STX (start of text)	34 22 042 "	66 42 102 B	98 62 142 b
3 3 003 ETX (end of text)	35 23 043 #	67 43 103 C	99 63 143 c
4 4 004 EOT (end of transmission)	36 24 044 \$	68 44 104 D	100 64 144 d
5 5 005 ENQ (enquiry)	37 25 045 %	69 45 105 E	101 65 145 e
6 6 006 ACK (acknowledge)	38 26 046 &	70 46 106 F	102 66 146 f
7 7 007 BEL (bell)	39 27 047 '	71 47 107 G	103 67 147 g
8 8 010 BS (backspace)	40 28 050 (72 48 110 H	104 68 150 h
9 9 011 TAB (horizontal tab)	41 29 051)	73 49 111 I	105 69 151 i
10 A 012 LF (NL line feed, new line)	42 2A 052 *	74 4A 112 J	106 6A 152 j
11 B 013 VT (vertical tab)	43 2B 053 +	75 4B 113 K	107 6B 153 k
12 C 014 FF (NP form feed, new page)	44 2C 054 ,	76 4C 114 L	108 6C 154 l
13 D 015 CR (carriage return)	45 2D 055 -	77 4D 115 M	109 6D 155 m
14 E 016 SO (shift out)	46 2E 056 .	78 4E 116 N	110 6E 156 n
15 F 017 SI (shift in)	47 2F 057 /	79 4F 117 O	111 6F 157 o
16 10 020 DLE (data link escape)	48 30 060 0	80 50 120 P	112 70 160 p
17 11 021 DC1 (device control 1)	49 31 061 1	81 51 121 Q	113 71 161 q
18 12 022 DC2 (device control 2)	50 32 062 2	82 52 122 R	114 72 162 r
19 13 023 DC3 (device control 3)	51 33 063 3	83 53 123 S	115 73 163 s
20 14 024 DC4 (device control 4)	52 34 064 4	84 54 124 T	116 74 164 t
21 15 025 NAK (negative acknowledgement)	53 35 065 5	85 55 125 U	117 75 165 u
22 16 026 SYN (synchronous idle)	54 36 066 6	86 56 126 V	118 76 166 v
23 17 027 ETB (end of trans. block)	55 37 067 7	87 57 127 W	119 77 167 w
24 18 030 CAN (cancel)	56 38 070 8	88 58 130 X	120 78 170 x
25 19 031 EM (end of medium)	57 39 071 9	89 59 131 Y	121 79 171 y
26 1A 032 SUB (substitute)	58 3A 072 :	90 5A 132 Z	122 7A 172 z
27 1B 033 ESC (escape)	59 3B 073 ;	91 5B 133 [123 7B 173 {
28 1C 034 FS (file separator)	60 3C 074 <	92 5C 134 \	124 7C 174
29 1D 035 GS (group separator)	61 3D 075 =	93 5D 135]	125 7D 175 }
30 1F 036 RS (record separator)	62 3F 076 >	94 5F 136 ^	126 7F 176 ~

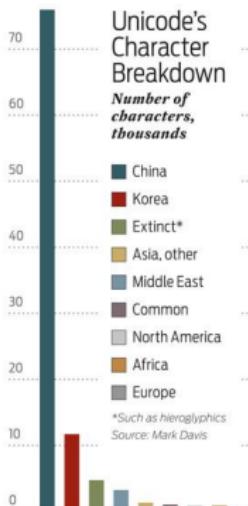
Au-delà du code ASCII de base

ASCII étendu sur 8 bits: Codes 0x80 à 0xFF

Le code étendu ISO 8859 Latin1 offre les caractères accentués minuscules et majuscules des langues occidentales: é è ê à ä ö ü ...

La norme **UNICODE** permet d'intégrer les autres langues, > 109'000 caractères pour 93 écritures dont le chinois.

- les 256 codes d' ISO 8859 sont au début de l'UNICODE
- **UTF-8** est un codage des caractères UNICODE comprenant de 1 à 4 octets. Il est recommandé mais son usage n'est pas encore généralisé.



70

Encoding Systems Behind Web Pages

Usage in percentages



UTF-8 (red) shows a sharp increase starting around 2006, surpassing ASCII (dark blue) by 2009.

ASCII (dark blue) shows a steady decline from 2001 to 2009.

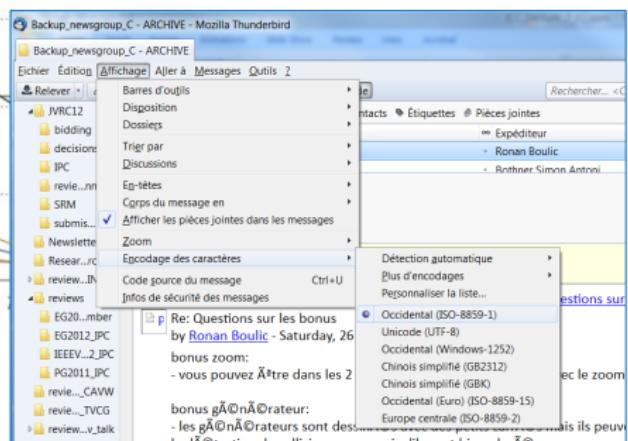
Other encoding systems (yellow, light blue, purple, grey, orange) show relatively low and stable usage throughout the period.

GB2312 (yellow) and 1252 (green) show a slight increase in usage between 2005 and 2009.

SJIS (light blue) and 1251 (purple) show a slight decrease in usage between 2005 and 2009.

EUC-KR (grey) and Other (orange) show very low and stable usage throughout the period.

Exemple: texte d'un message interprété avec ISO 8859 au lieu de UTF-8



Taux d'utilisation de UTF-8 pour l'écriture de pages web [IEEE Spectrum 7/12]

Du code multi-byte (idéogramme) à l'image



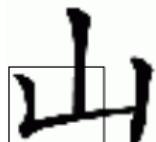
shan = montagne

le **symbole** peut être codé par 1 à 4 octets en UTF-8

MAIS la **représentation du symbole = son image** demande plus d'information.
Plusieurs approches sont possibles, du plus haut vers le plus bas niveau :



- 1) Préciser la **police de caractères** = « *style classique* ».
- 2) Caractériser les **contours** de la forme par un ensemble de **courbes** mathématiques paramétrées (silhouette). C'est la manière dont les polices de caractères sont construites.
- 3) Décomposer le plan de l'image en un **ensemble de cellules** qui indiquent la quantité d'encre (pixel).



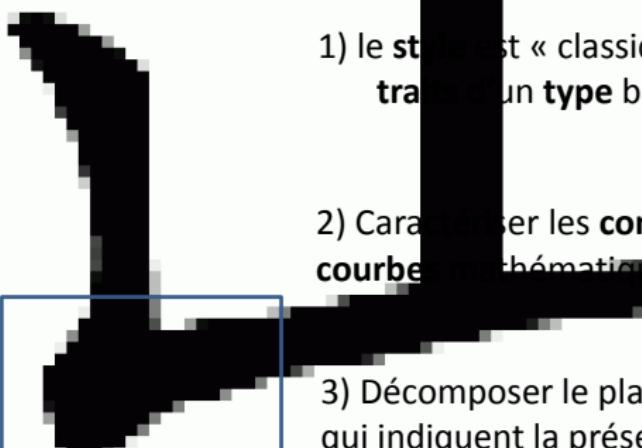
Du code multi-byte (idéogramme) à l'image



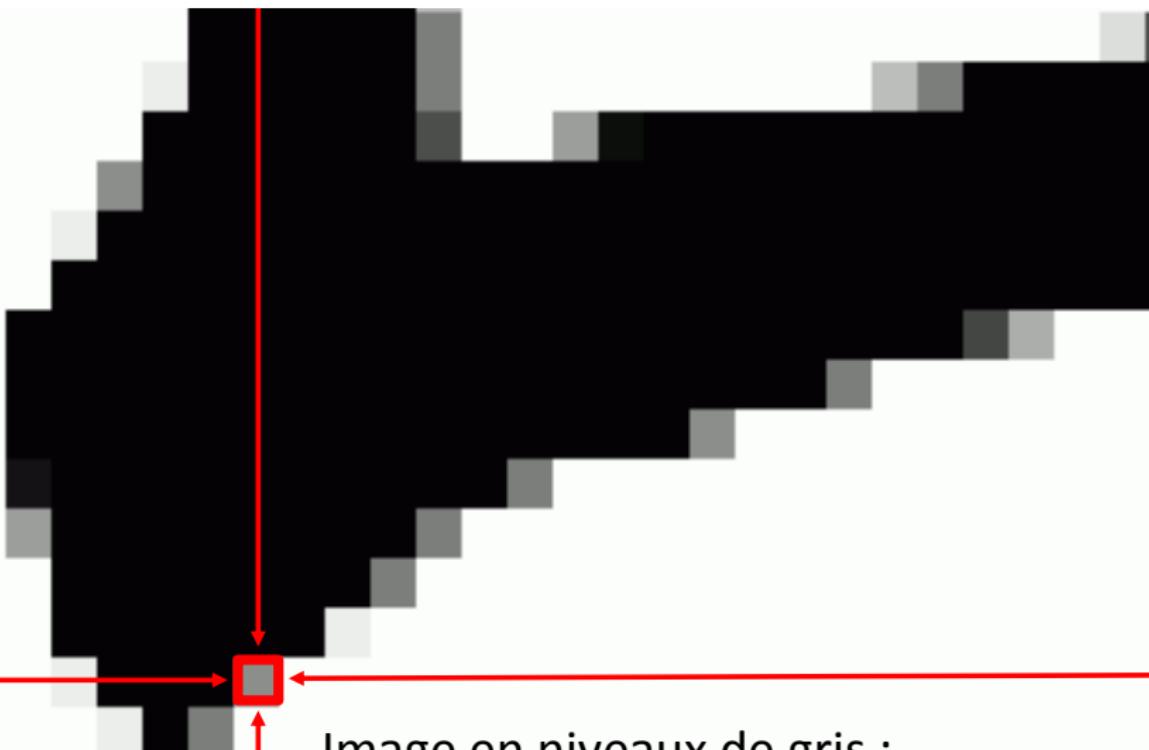
shan = montagne

le symbole peut être codé par 1 à 4 octets en UTF-8

MAIS **l'image du symbole** demande plus d'information pour la représenter telle quelle. Plusieurs approches:



- 1) le **stroke** est « classique ». Il comporte plusieurs traits d'un **type** bien précis parmi 37
- 2) Caractériser les **contours** de la forme par un ensemble de **courbes** mathématiques paramétrées (silhouette).
- 3) Décomposer le plan de l'image en un **ensemble de cellules** qui indiquent la présence d'encre (pixel).



bitmap

Image en niveaux de gris :

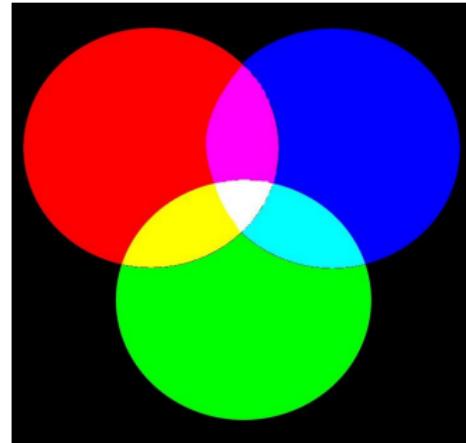
chaque pixel mémorise une intensité entre 0 (noir) et le maximum défini par le nombre de bits retenus (blanc)

Image en couleur:

chaque pixel mémorise l'intensité de 3 composantes primaires dont la combinaison permet de restituer un espace de couleurs.

Le codage RGB (Red, Green, Blue)

- synthèse additive des couleurs: Rouge + Vert donne
- niveaux de gris lorsque les trois composantes sont égales noir(0,0,0) et le maximum définit par le nombre de bits retenu (blanc)
- parfois complété d'une 4ième composante Alpha (transparence) pour les applications graphiques.



Taille d'une image UXGA 1600x1200 x 3octets/pixel = 5'760'000 octets

Résumé

Existe-t-il une représentation universelle de l'information ?

Une représentation est une **convention** humaine d'interprétation d'un ensemble de signes. Sa force est directement liée au nombre de personnes qui la partage, d'où l'importance des **standards** (ex : code ASCII, UTF).

Par quels moyens peut on représenter des symboles et des nombres ?

La **représentation binaire** suffit pour représenter de façon efficace (log) un nombre arbitrairement grand de signes.

Par convention nous utilisons les symboles 0 et 1.

Est il possible de construire une représentation exacte du monde réel ?

Les calculs avec la **représentation positionnelle entière** donnent des résultats exacts pour autant qu'on reste dans le **domaine couvert**.

Pour la **représentation à virgule flottante**, il faut se poser la question de la précision dont on a besoin. La représentation peut être **adaptée pour garantir une précision relative désirée**.