Le signal et sa représentation

Sebastien.Kramm@univ-rouen.fr

IUT de Rouen, dept. SRC

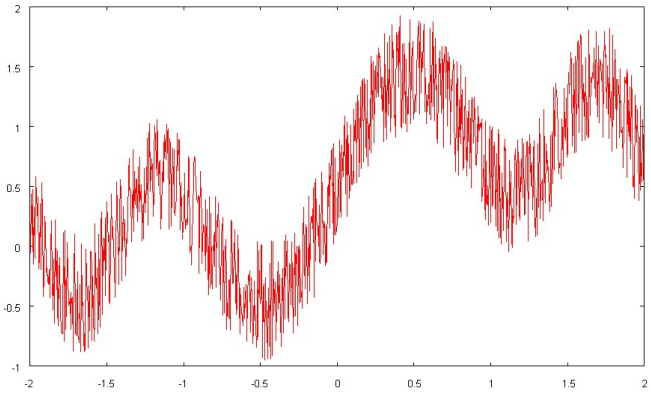
2012-2013

Sommaire

Signal ?

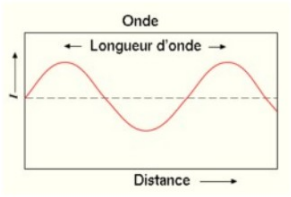
Définition

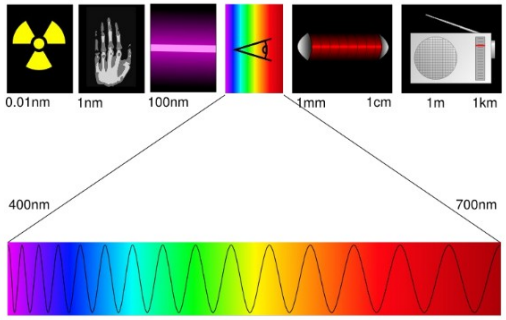
Variation d'une grandeur physique de nature quelconque porteuse d'**information**.



Support physique du signal

- Un signal est **porté** par une grandeur physique.
- Le plus souvent, une grandeur électrique (tension), qu'on pourra observer à l'oscilloscope.
- Mais aussi :
  - fibre optique : lumière (onde électromagnétique),
  - ondes radio (onde électromagnétique),
  - ondes sonores,
  - ondes sismiques,
  - etc.





Types de signaux

Deux types de classification dans les signaux

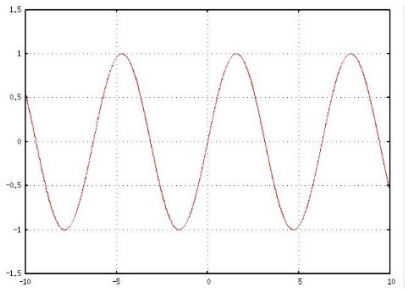
- Signal déterministe ou aléatoire.
- Signal numérique ou analogique.

- Remarque : on peut avoir un signal déterministe analogique ou numérique, idem avec un signal aléatoire.

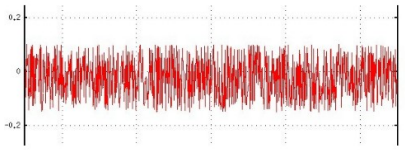
Classification 1

Deux familles de signaux :

- Signal déterministe : on connait parfaitement son comportement à tout instant.
- Signal aléatoire : la valeur à un instant 't' est imprévisible, on ne peut que l'estimer par des outils statistiques (lois de probabilités, grandeurs statistiques : moyenne, écart-type, ...).



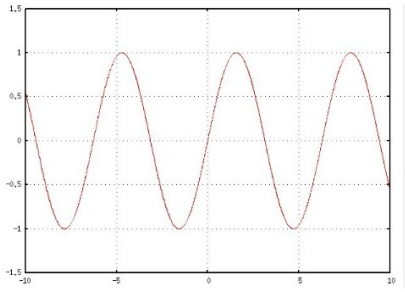
Signal déterministe



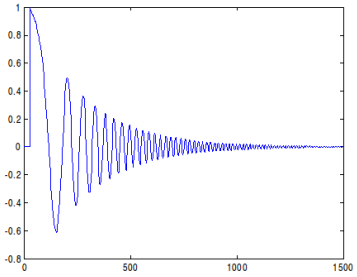
Signal aléatoire

Caractéristiques d'un signal déterministe

- Parmi les signaux déterministes, on a les signaux périodiques et non-périodiques.
- Pour les les signaux périodiques :
  - Période T : durée d'un cycle complet du signal, exprimé en s.
  - Dual : on peut lui associer la notion de **fréquence** $f = \frac{1}{T}$, exprimée en Hertz (Hz).



Signal périodique



Signal non périodique

Caractéristiques d'un signal aléatoire

- Un signal aléatoire est dit **stationnaire** si ses propriétés statistiques d'ensemble ne dépendent pas de l'instant choisi.
  - Stationnarité au premier ordre : les moyennes calculées à des instants différents sont identiques.
  - Stationnarité au deuxième ordre : les variances (ou écarts-type) calculées à des instants différents sont identiques.

Variance et écart-type d'une variable aléatoire x

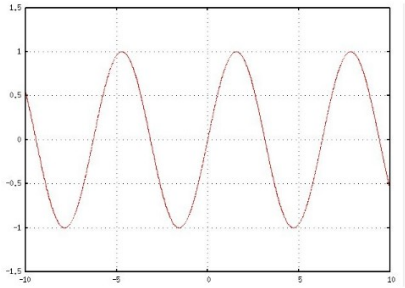
- La variance mesure la dispersion des valeurs par rapport à la moyenne $\bar{x}$
- L'écart type σ est la racine de la variance : $\sigma(x) = \sqrt{Var(x)}$

$$Var(x) = \sum_i \left[(x_i - \bar{x})^2\right]$$

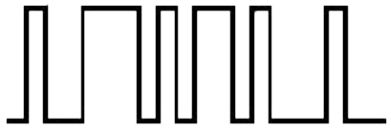
Classification 2

Deux familles de signaux :

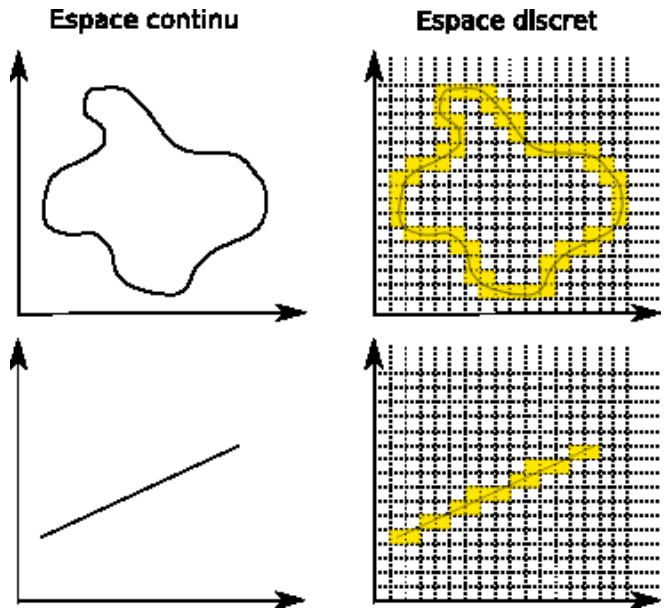
- Signal analogique : la grandeur varie de façon **continue** dans le temps, pas de discontinuités.
  Il est impossible de dénombrer les valeurs possibles.
- Signal numérique : le signal ne porte qu'un nombre finis de valeurs (alphabet), on est dans un espace **discret**.



Signal analogique

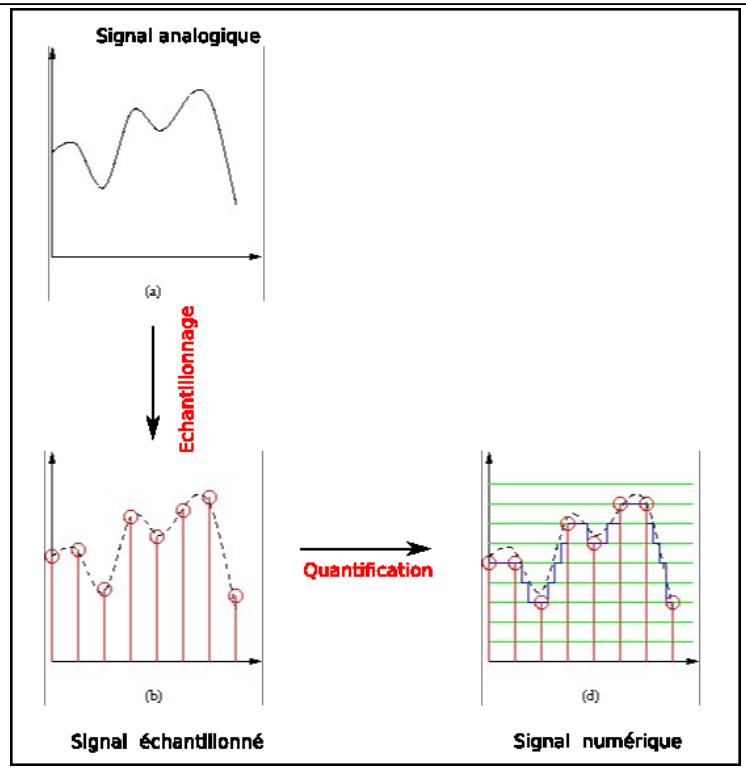


Signal numérique
Rem : souvent l'information est **encodée** de façon temporelle.



Passage du continu au discret

- Le monde physique est analogique (pression, température, tension, . . .)
- Nos calculateurs sont numériques.
- Il faut pouvoir passer du domaine continu au domaine discret.
- Deux étapes :
  1. Echantillonnage
  2. Quantification
- Ces deux étapes constituent la **numérisation**.
- Tout signal est numérisable, quelque soit l'information véhiculée : une température, du son, une image, etc.



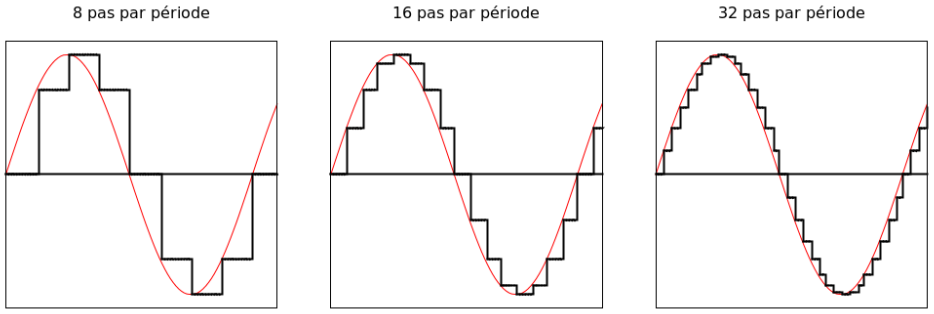
Choix pratique

- En numérisant, on veut **conserver** l'information portée par le signal.
- Comment répondre aux questions suivantes :
  - Combien de niveaux de quantification choisir ?
  - Quelle fréquence d'échantillonnage ?
- Pour la quantification, c'est fonction de la complexité du signal. Un signal simple se contentera d'un faible nombre de niveaux, un signal complexe demandera un nombre de niveaux important.
- Plus formellement, la quantification introduit du bruit, il faut définir le **rapport signal sur bruit** acceptable, et en déduire le nombre minimal de niveaux.
- Pour la fréquence d'échantillonnage, le **théorème de Shannon** indique qu'il faut une fréquence supérieure à 2 fois la fréquence maximale du signal.

→ On reviendra la-dessus plus tard...

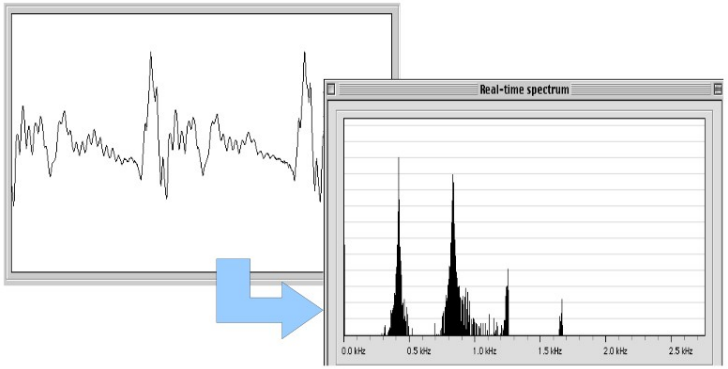
Influence des paramètres de numérisation

- Influence du pas d'échantillonnage



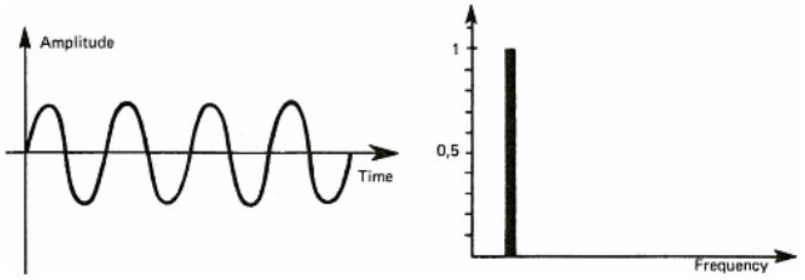
Représentation d'un signal

- Un signal peut être représenté :
  - par son évolution temporelle : axe 0x → temps.
  - par son **spectre** (répartition des fréquences) : axe 0x → fréquence.
- Ces deux représentations sont équivalentes, on parle de **dualité temps-fréquence**.
- Outil mathématique permettant de passer de l'une à l'autre de ces deux représentation : **Transformée de Fourier**.



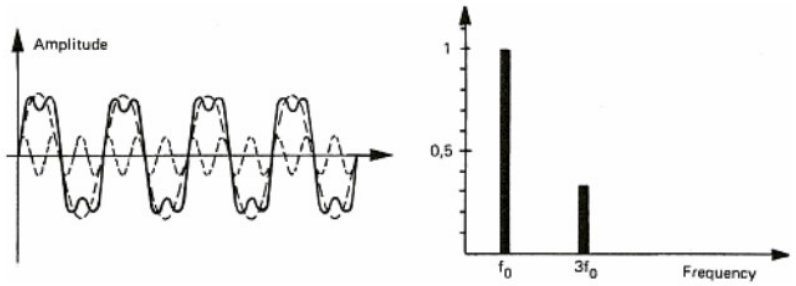
En pratique - 1

- Pour un signal pur (sinusoide), il n'y a qu'une seule raie dans le spectre.



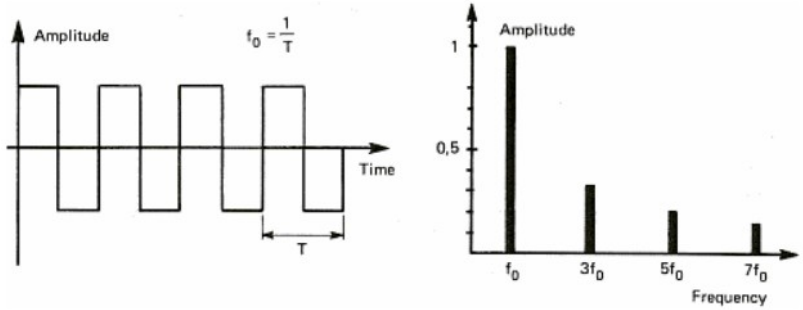
En pratique - 2

- Pour une combinaison de deux sinusoïdes, on trouve deux fréquences dans le spectre, avec une hauteur correspondant à leur amplitudes relatives.



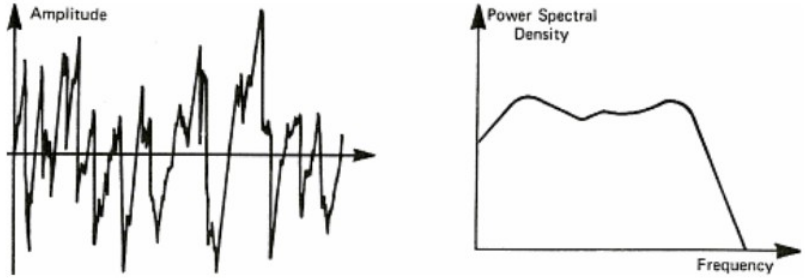
En pratique - 3

- Pour un signal carré, on a des raies impaires avec une amplitude décroissante.



En pratique - 4

- Pour un signal quelconque (bruit aléatoire), on a un spectre continu représentant toutes les fréquences contenues dans le signal.



Transformée de Fourier

- Joseph Fourier (1768-1830), mathématicien et physicien français.
- Dans le domaine continu, la transformation de Fourier associe à une fonction $f(t)$ sa transformée $\mathcal{F}(f)$, notée $\hat{f}(\nu)$
- Avec t en s. et ν la fréquence (s^{-1}), elle s'exprime :

$$\mathcal{F}(f(t)) = \hat{f}(\nu) = \int_{-\infty}^{+\infty} f(t)\, \mathrm{e}^{-i2\pi\nu t}\, \mathrm{d}t$$

- Cette transformation fait appel à l'ensemble $\mathbb{C}$ des nombres complexes.
- On a : $\mathrm{e}^{it} = \cos(t) + i\sin(t)$

Transformée de Fourier inverse

- La transformation de Fourier est inversible : connaissant $\hat{f} = \mathcal{F}(f)$, on peut retrouver $f(t)$:

$$f(t) = \int_{-\infty}^{+\infty} \hat{f}(\nu)\, \mathrm{e}^{+i2\pi\nu t}\, \mathrm{d}\nu$$

Calcul des coefficients de la série

- Les coefficients a_n et b_n se calculent avec les expressions suivantes :

$$a_n = \frac{2}{T} \int_{t_0}^{t_0+T} f(t).\cos(n\frac{2\pi}{T}t)dt$$

$$b_n = \frac{2}{T} \int_{t_0}^{t_0+T} f(t).\sin(n\frac{2\pi}{T}t)dt$$

Signaux périodiques

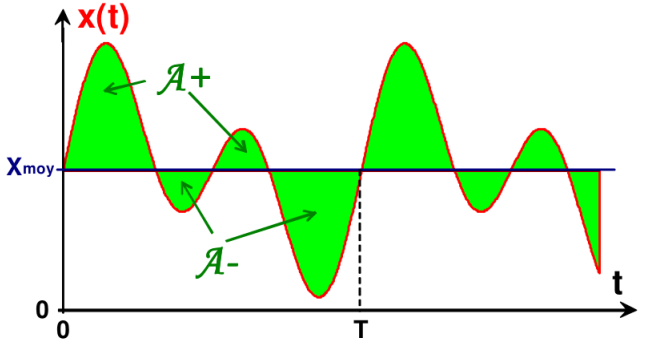
- Pour les signaux périodiques, la transformation de Fourier sera discrète : on aura un ensemble de raies.
- Tout signal périodique peut-être reconstruit à partir d'une somme de fonctions sinus et cosinus.

$$f(t) = a_0 + \sum_{n=1}^{\infty} \left[a_n \cdot \cos\left(n\, t \frac{2\pi}{T}\right) + b_n \cdot \sin\left(n\, t \frac{2\pi}{T}\right) \right]$$

- Les valeurs (a_n, b_n) constituent les **coefficients** de la série de Fourier. Leur valeur relative fixe la **phase** de l'harmonique de rang n.
- La valeur a_0 représente la **valeur moyenne du signal**.

Calcul de la valeur moyenne

- La valeur moyenne du signal représente sa **composante continue**.
- Elle est définie comme la valeur délimitant la surface du signal en deux parties égales.



$\Rightarrow \mathcal{A}^+ = \mathcal{A}^-$

Source : C. Bissieres

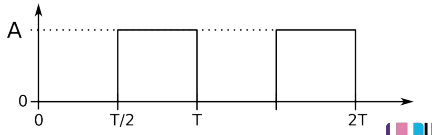
Calcul de la valeur moyenne

- La valeur moyenne d'une fonction sinusoidale (sin ou cos) est **nulle**.
- Cas général d'une fonction périodique quelconque : on intègre la fonction sur sa période.

$$a_0 = \frac{1}{T}\int_0^T f(t)\mathrm{d}t$$

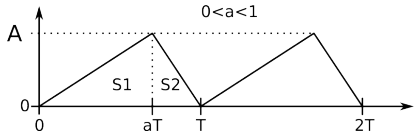
- Aspect intuitif : correspond à la surface non-nulle d'une période du signal, divisé par la période T.

- Pour un signal carré : la moyenne vaut A/2.



Valeur moyenne : signal triangulaire

- Pour un signal triangulaire : on calcule séparemment les deux surfaces : $S = S_1 + S_2$



- La surface d'une portion triangulaire de signal est égale à la surface du rectangle englobant $(h \times l)$, divisée par 2
  - pente ascendante : $S_1 = \frac{aT \times A}{2} = \frac{1}{2}a.T.A$
  - pente descendante : $S_2 = \frac{(T-aT)\times A}{2} = \frac{1}{2}(1-a).T.A$
- D'ou : $S = \frac{1}{2}.A.T.[a + (1-a)] = \frac{A.T}{2}$
- et $a_0 = S/T = A/2$
  $\Rightarrow$ La valeur moyenne est ici indépendante de la valeur de a.

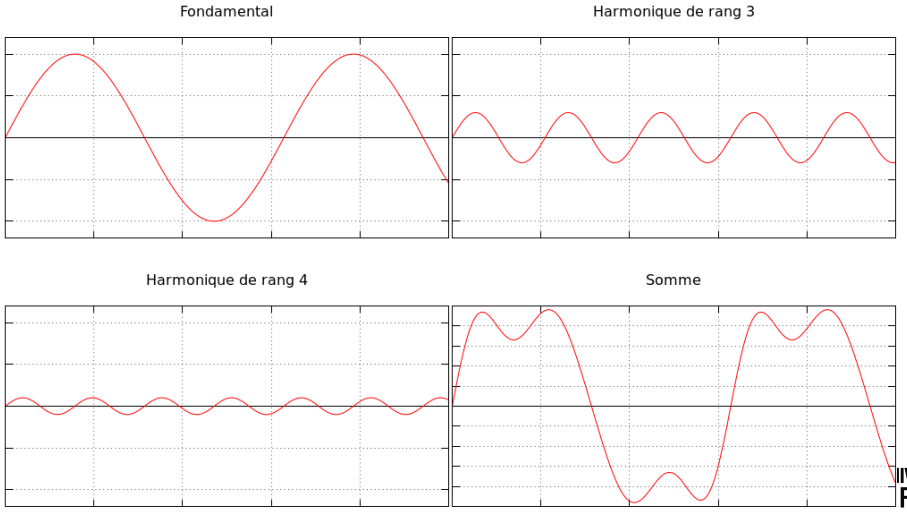
Signaux périodiques : en pratique

A retenir

- Toute fonction périodique est une somme de sinusoides de fréquence f_0, $2f_0$, $3f_0$, $4f_0$,...
- Le signal de fréquence f_0 est appelée **fondamental**.
- Les signaux de fréquences supérieures sont appelées **harmoniques**.

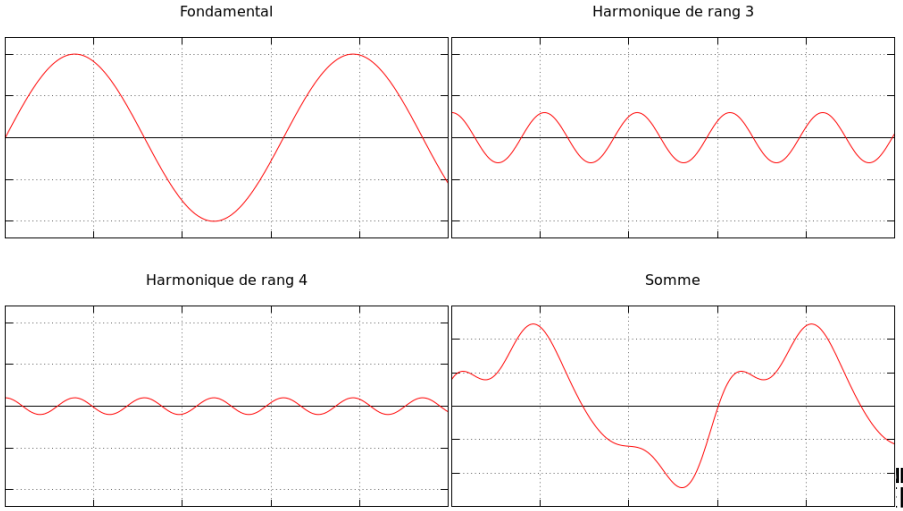
Exemple de synthèse - 1

- Somme d'une fondamentale et d'harmoniques de rang 3 et 4 de même phase.
  $s(t) = \sin(t) + 0,3 \cdot \sin(3t) + 0,1 \cdot \sin(4t)$



Exemple de synthèse - 2

- Somme d'une fondamentale et d'harmoniques de rang 3 et 4 de phase décalée de 90 ° (fonction cos).
  $s(t) = \sin(t) + 0,3 \cdot \cos(3t) + 0,1 \cdot \cos(4t)$



Signaux périodiques : en pratique

- L'amplitude de l'harmonique de rang n vaut $c_n = \sqrt{a_n^2 + b_n^2}$
- Pour un signal carré, les termes b_n sont nuls, on a une somme de fonction sinus impaire (rang 3,5,7,...), avec une amplitude relative de 1/3, 1/5, 1/7, ...
- D'un point de vue humain, plus un signal sera riche en harmonique, plus il donnera l'impression d'être "large".
- Plus un signal est de durée brève par rapport à sa période, plus son spectre fréquentiel est étendu.
- Plus un signal périodique varie brutalement, plus les harmoniques de rang élevé seront importantes.

Sommaire

Base de numération

- Un signal va être représenté à un instant t par une valeur numérique (un nombre) N s'exprime dans une base de numération.
- Par exemple, la base 10 est celle utilisée par l'humain.
- Mais le nombre en lui même est **indépendant** de sa base de numération.
- D'un point de vue mathématique, on peut convertir un nombre dans une **autre** base de numération, puis faire l'opération inverse : le nombre est inchangé.
- En pratique, les ordinateurs représentent le nombre de façon **finie**, donc :
  - En math : $a + b - a = b$
  - Sur une machine : ... pas toujours.
  - Exemple : $a = 10^{100}$, $b = 1$

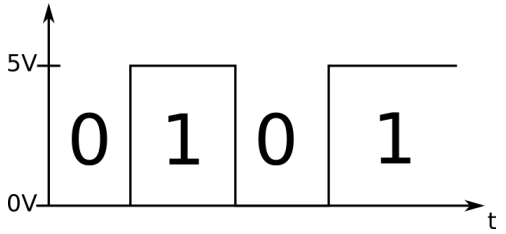
Base de numération, symboles et alphabets

- Le mot "2387" désigne un nombre exprimé en base 10.
- Avec **un** symbole d'une base b, on pourra coder b valeurs.
- Avec n symboles d'une base b, on pourra coder b^n valeurs.
  Exemple : avec 3 symboles de l'alphabet $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, je peux coder $10^3 = 1000$ valeurs différentes (0 à 999)
- Un entier naturel a une écriture unique (si on interdit les '0' à gauche) :
  $2387 = 2.10^3 + 3.10^2 + 8.10^1 + 7.10^0$
- On peut généraliser cette notation à n'importe quelle base b :
  (avec $b > 0$)

$$n = \sum_{i=0}^{p} a_i b^i = a_p b^p + ... + a_2 b^2 + a_1 b^1 + a_0 b^0$$

Codage binaire

- Les ordinateurs travaillent en interne en **binaire** : l'élément de base est le chiffre binaire, le bit (BInary digiT).
  $\Rightarrow$ alphabet à deux symboles : a= {'0' ; '1' }
- Au niveau interne (électronique), ceci correspond à des niveaux de tensions :
  - '0' : 0V
  - '1' : niveau haut (en général, la tension d'alimentation des circuits intégrés, 5V ou moins)



Regroupement de bits

- Pour encoder des alphabets avec plus de symboles (=des valeurs supérieures à 2), on **associe** plusieurs bits en parallèle.
  $\Rightarrow$ Avec n bits, on peut coder 2^n valeurs différentes.
  - 2 bits $\rightarrow 2^2 = 4$ valeurs différentes (0 à 3)
  - 3 bits $\rightarrow 2^3 = 8$ valeurs différentes (0 à 7)
  - 8 bits $\rightarrow 2^8 = 256$ valeurs différentes (0 à 255)
  - 16 bits $\rightarrow 2^{16} = 65.536$ valeurs différentes
  - 32 bits $\rightarrow 2^{32} = 4.294.967.296$ valeurs différentes

Nombre de bits	8	16	32
Terme anglais	byte	word	double word
Terme français	octet	mot	double mot

Arithmétique binaire

- Les ordinateurs sont dotés d'unités de calcul binaire.
- L'algèbre binaire fonctionne de façon similaire à l'algèbre en base 10.
  - $0 + 0 = 0$
  - $0 + 1 = 1$
  - $1 + 1 = 10$ (=2 en base 10)
  - $1 + 1 + 1 = 11$ (=3 en base 10)
- Multiplier et diviser par 2 revient à **décaler les bits**.
  - Division par 2 : 0100.0010 /2 = 0010.0001 $\leftrightarrow$ 66/2 = 33
  - Multiplication par 2 : 0001.1000 $\times$ 2 = 0011.0000 $\leftrightarrow$ 24 $\times$ 2 = 48

Blague d'informaticien

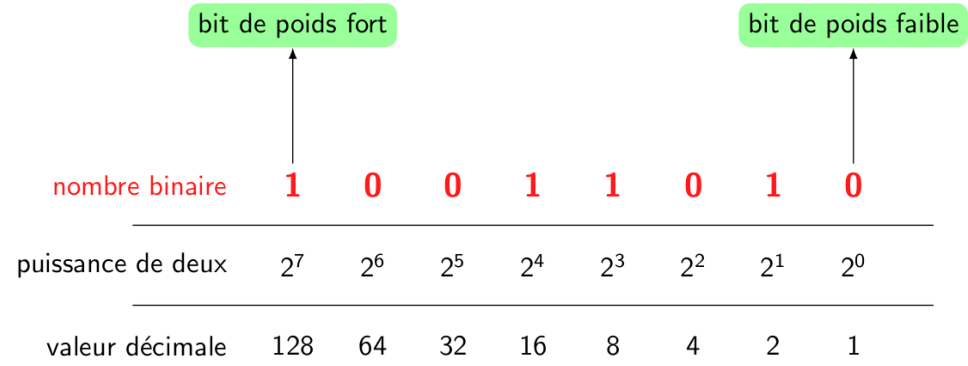
Il n'y a que 10 sortes de personnes dans le monde :
celles qui comprennent le binaire et celles qui ne le comprennent pas.

Codage binaire

- On parle de "codage en binaire" :

Valeur (base 10)	Codage en binaire
0	000
1	001
2	010
3	011
4	100
...	...

- Dans un système informatique, les bits sont regroupés par 8
- Un groupe de 8 bits s'appelle un **octet**.
  $2^8 = 256$ valeurs possibles, numérotées de 0 à 255.



source : J. Landré, IUT Troyes

- MSB (*Most Significant Bit*) : "bit le plus signifiant", on dit "bit de poids fort"
- LSB (*Least Significant Bit*) : "bit le moins signifiant", on dit "bit de poids faible".

Hexadécimal : base 16

- Problème du binaire : difficile à lire par l'humain...
- Exemple : le nombre 1010101111001101 est-il plus grand ou plus petit que le nombre 1010101111101101 ?
- Pour "montrer" une valeur binaire à un humain, on a donc adopté une représentation en base 16 : l'**hexadécimal**.
- Les 16 symboles sont les 10 du système décimal, plus les lettres A,B,C,D,E,F

Symbole	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Valeur associée	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

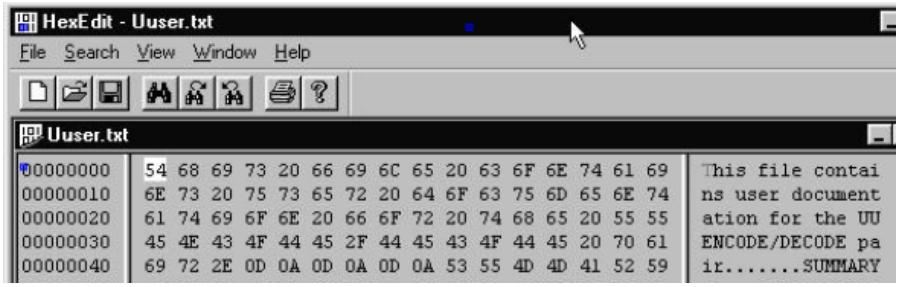
- Motivation : **un** symbole hexa correspond à **quatre** bits.
  $(2^4 = 16)$

Intérêt de l'hexadécimal

- Le codage hexadécimal permet une représentation plus compacte du binaire.
- La conversion entre binaire et hexadécimal est très simple !
  - binaire ⇒ hexa : on regroupe les bits par groupe de 4, en partant de la droite.
  - hexa ⇒ binaire : on utilise la table précédente.
- Exemple :
  - 1010101111001101 = 1010.1011.1100.1101 = `ABCD`
  - 1010101111101101 = 1010.1011.1110.1101 = `ABED`
- Notation : pour signifier la base hexadécimale, plusieurs notations peuvent être rencontrées.
  `0xABCD` ; `$ABCD` ; `ABCDh` ; ...

Quand rencontre-t-on de l'hexadécimal ?

- Quand on s'interesse (de près) au contenu d'un fichier sur un ordinateur.
- Un fichier est une suite d'octets, dont le sens dépend de ce qui y est stocké (texte, image, son, vidéo, ...)
- Un **éditeur hexadécimal** montre le contenu du fichier sous forme brute (binaire), mais représenté en hexa.



Conversion en base 2 ou 16

- Pour convertir un nombre n exprimé en base 10 dans une base b, il faut faire une succession de divisions par b, jusqu'à ce que le résultat de la division soit inférieur à b.
- Exemple : soit la valeur 135, à convertir en base 2 :

135	2
$a_0 = 1$	67
	2
$a_1 = 1$	33
	2
$a_2 = 1$	16
	2
$a_3 = 0$	8
	2
$a_4 = 0$	4
	2
$a_5 = 0$	2
	2
$a_6 = 0$	$1 < 2$

- On relève le résultat de la **dernière** division, et l'on ajoute les **restes** des divisions précédentes : $(135)_{10} = (10000111)_2$
- Remarque : pour le binaire, le premier chiffre sera toujours un 1

Conversion d'une base 2 ou 16 en base 10

- Le passage d'une base b en base 10 se fait toujours de la même façon, par un développement de la série des puissances.
- Exemple en base 2 :

$$\begin{aligned}(10010001)_2 &= 1.2^7 + 0.2^6 + 0.2^5 + 1.2^4 + 0.2^3 + 0.2^2 + 0.2^1 + 1.2^0 \\ &= 2^7 + 2^4 + 2^0 \\ &= 128 + 16 + 1 \\ &= 145\end{aligned}$$

- Exemple en base 16 :

$$\begin{aligned}(C3E)_{16} &= 12 \cdot 16^2 + 3 \cdot 16^1 + 14 \cdot 16^0 \\ &= 12 \cdot 256 + 3 \cdot 16 + 14 \cdot 1 \\ &= 3072 + 48 + 14 \\ &= 3134\end{aligned}$$

Comment coder des décimales ?

- Principe général : identique à celui utilisé pour les entiers, en étendant aux puissances négatives :

$$\begin{aligned}n &= \sum_{i=-q}^{p} a_i b^i \\ &= a_p b^p + \ldots + a_2 b^2 + a_1 b^1 + a_0 b^0 + a_{-1} b^{-1} + a_{-2} b^{-2} + \ldots + a_{-q} b^{-q}\end{aligned}$$

avec :
  - p : plus grande puissance positive du nombre,
  - q : plus grande puissance négative du nombre.
- Exemple en base 10 :
$(3,1415)_{10} = 3.10^0 + 1.10^{-1} + 4.10^{-2} + 1.10^{-3} + 5.10^{-4}$

Puissance négatives en base 2

- On étend ce principe en base 2 aux puissances de 2 négatives :

10,101

$1 \times 2^{-3} = 1 \times 0{,}125 = 0{,}125$

$0 \times 2^{-2} = 0 \times 0{,}25 = 0$

$1 \times 2^{-1} = 1 \times 0{,}5 = 0{,}5$

$0 \times 2^{0} = 0 \times 1 = 0$

$1 \times 2^{1} = 1 \times 2 = 2$

2,625

Réels : conversion de base 10 en base 2

- On traite **séparemment** partie entière et partie fractionnaire
- Pour la partie fractionnaire :
  - On effectue une suite de **multiplications** par 2, jusqu'à obtenir un 1.
  - A chaque étape, on ne garde que la partie fractionnaire du résultat.
  - Puis on regroupe les bits dans l'ordre d'apparition.
- Exemple : partie fractionnaire : 0,125

Etape			Résultat	Bit	poids
1	0,125	x 2 =	0,25	0	2^{-1}
2	0,25	x 2 =	0,5	0	2^{-2}
3	0,5	x 2 =	1	1	2^{-3}

$\Rightarrow (0,125)_{10} = (0,001)_2$

Remarque

En général, on arrive jamais à 1. On s'arrête à un nombre de bits donné.

Caractère irrationel de la conversion

- Si un nombre N a un nombre de décimales **fini** dans une base $b1$, il peut avoir un nombre de décimales **infini** lorsqu'il est exprimé dans une autre base $b2$.
- Exemple : soit le nombre $N = (0,2)_{10}$ à convertir en base 2 :

Etape			Résultat	Bit	poids
1	0,2	x 2 =	0,4	0	2^{-1}
2	0,4	x 2 =	0,8	0	2^{-2}
3	0,8	x 2 =	1,6	1	2^{-3}
4	0,6	x 2 =	1,2	1	2^{-4}
5	0,2	x 2 =	0,4	0	2^{-5}
6	0,4	x 2 =	etc.		

La valeur 0,2 est codée en binaire par 0,0011 0011 0011 00...
$\Rightarrow$ **infinité** de décimales.

Corollaire

- Conséquence : une représentation avec un nombre de symboles **fini** dans une base $b1$ n'est qu'une **approximation** du même nombre exprimé dans une autre base $b2$.
- Exemple : soit le nombre $N = (0,2)_{10}$, qu'on représente en binaire sur un nombre de bits n :

n	N (base 2)	N (base 10)
4	0,0011	0,1875
7	0,0011001	0.1953125
8	0,00110011	0.19921875
12	0,001100110011	0.199951172

Représentation en virgule flottante

- Tout nombre N peut être représenté en **virgule flottante**, dans une base de numération quelconque b sous la forme : $N = M \cdot b^E$
  - M : **Mantisse** signée
  - b : base de numération
  - E : **Exposant** (entier relatif)
- Exemples :

N		M	b	E
$3,1415 \cdot 10^0$	$\Rightarrow$	3,1415	10	0
$1011,11 \cdot 2^{-4}$	$\Rightarrow$	1011,11	2	-4
$2345,67 \cdot 8^9$	$\Rightarrow$	2345,67	8	9

- Problème : plusieurs représentations possibles pour un même nombre.
  $3,1415 \cdot 10^0 = 31,415 \cdot 10^{-1} = 0,31415 \cdot 10^1$
- Pour comparer des nombres, on ne peut pas ainsi comparer les mantisses et les exposants.

Normalisation de la représentation

- Afin d'avoir une représentation **unique** d'un nombre, on définit le concept de **normalisation** :

Normalisation
Décalage de la virgule jusqu'à avoir **un seul** chiffre à gauche de la virgule, et ajustement de l'exposant en fonction du nombre de décalages.
  - Décalage vers la gauche $\leftrightarrow$ incrémentation de l'exposant.
  - Décalage vers la droite $\leftrightarrow$ décrémentation de l'exposant.

- Par exemple (avec $b = 10$) :
  $250.000 = 2,5 \cdot 10^5 \rightarrow M = 2,5,\ E = 5$
  $0,000.004.59 = 4,59 \cdot 10^{-6} \rightarrow M = 4,59,\ E = -6$

Exercices

Donner l'exposant et la mantisse normalisée (base 10) des valeurs physiques suivantes (en unités SI) :

N	M	E
2,718		
42,195 km		
330 μF		
10 GW		
0,15 nF		

Sommaire

Pourquoi coder ?

- Pour pouvoir être traité par un ordinateur, chaque **élément d'information** va devoir être représenté par une **valeur binaire**.
- Le codage est fonction du contexte :
  - transmission directe à un humain : on choisira une représentation utilisant les lettres de l'alphabet usuel, dans une langue commune (comme par exemple le français ou l'anglais) ;
  - stockage : on essaiera éventuellement de minimiser la place nécessaire à ce stockage ;
  - transmission entre ordinateurs : on essaiera alors de minimiser les risques de perte d'information lors de cette transmission, en introduisant des **codes correcteurs** via de la **redondance**.

Exemples de codes

- 1837 : code Morse pour le télégraphe ; alphabet A={ - ; . ; silence}
  A : .- ; B : - ... C : -.-. D : - . . E : . ; etc.
- 1963 : code ASCII (*American Standard Code for Information Interchange*), pour coder les caractères alphabétiques.
- 1991 : UNICODE
- De façon formelle, l'opération de codage consiste à associer chaque symbole d'un alphabet source à un mot d'un alphabet cible.

A retenir

Un flot brut de bits sans information sur le code associé est inutilisable !
1000100110110110001000011110101101 $\Rightarrow$? ? ?

Codage des entiers

- Pour coder des entiers, on utilise le codage binaire naturel, en spécifiant par des moyens annexes :
  - le nombre de bits utilisés pour chaque symbole,
  - l'ordre dans lequel on envoie poids fort / poids faible (notion d'*endianess*).
- Par exemple, le message suivant :
  0001.0000.0000.0010.0000.0100.0000.0001
  pourra représenter :
  - Si les valeurs sont codées sur 4 bits, le message aura une longueur de 8 mots et sera : 1 ;0 ;0 ;2 ;0 ;4 ;0 ;1
  - Si les valeurs sont codées sur 8 bits, le message aura une longueur de 4 mots et sera : 16 ;2 ;4 ;1
  - Si les valeurs sont codées sur 16 bits, le message aura une longueur de 2 mots et sera : 4098 ;1025

Comment représenter en binaire un nombre relatif ?

- Pour encoder le signe, on réserve un bit, appelé le bit de signe, et qui sera **toujours** en position de poids fort (MSB).
  - 0 : nombre positif
  - 1 : nombre négatif
- Plusieurs solutions sont envisageables :
  1. bit de signe + binaire naturel
     4 = 0000.0100 $\rightarrow$ -4 = 1000.0100
  2. bit de signe + complément à un
     4 = 0000.0100 $\rightarrow$ -4 = 1111.1011
  3. complément à deux
- Problème des méthodes 1 et 2 : deux représentations possible pour 0 :
  1. bit de signe + binaire naturel
     $+0_{10} = 0000.0000 \leftrightarrow -0_{10} = 1000.0000$
  2. bit de signe + complément à un
     $+0_{10} = 0000.0000 \leftrightarrow -0_{10} = 1111.1111$
- De plus, seul le complément à deux permet de faire de l'arithmétique correcte avec la même unité de calcul.

Notation en complément à deux

- Cette représentation s'entend pour un **nombre de bits donné**.
- Pour une valeur sur n bits, on dispose de $n-1$ bits pour coder la valeur (MSB : bit de signe).
- Nombre positif : codage binaire classique.
- Nombre négatif : complément à deux :
  1. Complémentation de tous les bits
  2. Addition de 1 à la valeur obtenue
- Exemple : codage de la valeur -14 sur 8 bits :
  $14_{10} = 8 + 4 + 2 = (0000.1110)_2$
  1. Complément à 1 : $\overline{0000.1110} = 1111.0001$
  2. Ajout de 1 : $1111.0001 + 1 = 1111.0010 = \texttt{0xF2}$
- Exemple : codage de la valeur -14 sur 16 bits :
  $14_{10} = (0000.0000.0000.1110)_2$
  1. Complément à 1 : 1111.1111.1111.0001
  2. Ajout de 1 : $1111.1111.1111.0010 = \texttt{0xFFF2}$

Intérêt du complément à deux

- Opération : $5 + 4$, sur 8 bits

	binaire	décimal
	0000.0101	5
+	0000.0100	4
	0000.1001	9

- Opération : $5 - 4$, sur 8 bits
  Le processeur va en fait exécuter l'opération : 5 + (-4)

	binaire	décimal
	0000.0101	5
+	1111.1100	-4
	1.0000.0001	1

- Remarque : Le bit 1 en position 2^9 est la **retenue**, dont il ne faut pas tenir compte ici (calcul sur 8 bits).

Plage de valeurs en complément à deux

- Rappel : en binaire naturel, pour n bits, on peut encoder 2^n valeurs, qui représentent les valeurs de 0 à $2^n - 1$.

Par exemple, $n = 8 \Rightarrow$ 256 valeurs, de 0 à 255.

- En complément à deux, on ne dispose plus que de $n-1$ bits...
  ... mais on a toujours 2^n valeurs possibles !
- Ces 2^n valeurs sont pour moitié positives et négative.

Par exemple, $n = 8 \Rightarrow$ 128 valeurs positives et 128 valeurs négatives.

- Comme la valeur 0 est (arbitrairement) positive, l'étendue de chaque zone est donc différente.
  Par exemple, si $n = 8 \rightarrow -128 < N < +127$

Généralisation

En complément à deux sur n bits, la valeur numérique aura une étendue de -2^{n-1} à $+2^{n-1} - 1$

Encodage de nombres fractionnaires

- Soit le nombre (base 2)
  1.0010.0111.0001.0000.1010,1110.0010.1011.101
- Question : comment encoder cette valeur dans un ordinateur ?
- Deux approches envisageables :
  - Codage en virgule fixe : on impose un format, n bits pour la partie entière, m bits pour la partie fractionnaire.
  - Codage en virgule flottante : on utilise la notation $N = M \cdot b^E$ vue précedemment.

Représentation en virgule fixe

- Principe : on décide (arbitrairement ou en fonction d'un contexte) d'une position fixe pour la virgule.
- Exemple (en binaire et sur 8 bits) : XXXXX,XXX
- calcul :

	base 2	base 10
	01000,100	8,5
+	00100,001	4,25
	01100,101	12,75

Inconvénients

- Erreur d'arrondi importante.
  Exemple : soit l'opération $4,25/2 \times 2$
  $\rightarrow$ 00100,001 / 2 = 00010,000
  $\rightarrow 00010,000 \times 2 = 00100,000 = 4,0$
- Impossible de représenter à la fois des très grand nombres et des très petits.

Représentation en virgule flottante

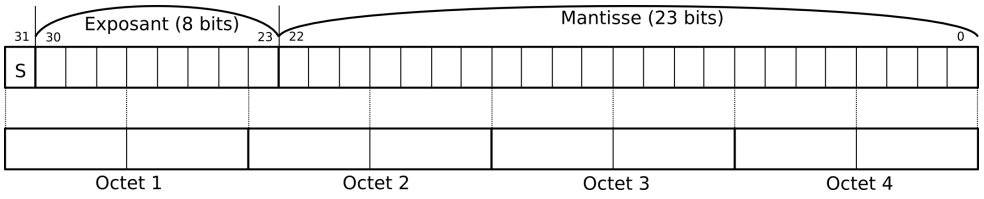
- Pour eviter les erreurs d'arrondi inhérent au codage en virgule fixe, la plupart des machines proposent un codage en virgule flottante.
- Un nombre sera codé sous la forme $N = M \cdot 2^E$, avec :
  - M : mantisse **normalisée**,
  - E : exposant
- On pourra ainsi facilement comparer des nombres entre eux.

En pratique

La grande majorité des machines modernes utilisent la norme **IEEE 754** pour le codage des nombres.

Norme IEEE 754

- Cette norme définit (principalement) deux formats de stockage des nombres représentés en virgule flottante.
  - *Simple précision* : 32 bits (4 octets)
    - 1 bit de signe : $N >= 0 \rightarrow S = 0$, $N < 0 \rightarrow S = 1$
    - 8 bits d'exposant (décalé) : -128 à +127
    - 23 bits de mantisse : 0 à $2^{23} - 1$
  - *Double précision* : 64 bits (8 octets)
    - 1 bit de signe
    - 11 bits d'exposant (décalé) : - 1024 à +1023
    - 52 bits de mantisse 0 à $2^{52} - 1$



Codage en IEEE 754 - Mantisse

- Mantisse : on code la mantisse normalisée **sans** le premier 1.
  En effet, comme **tout** nombre normalisé aura un '1' comme premier chiffre [1], on le considère implicite (on ne le mémorise pas), et on gagne un bit !
- Exemple : soit à encoder le nombre 1,25
  En binaire : $1,25 = 1 + 1/4 = 2^0 + 2^{-2} = (1,01)_2$
  La mantisse à coder sera : ,01
  La mantisse stockée sur 23 bits sera : 0100.0000.0000.0000.0000.000

Au décodage...

... Penser à **ajouter** ce '1' !

- Example : on code la mantisse suivante (23 bits) :
  1111.0000.1111.0000.0000.000
  $\Rightarrow$ la mantisse réelle du nombre sera :
  1,1111.0000.1111

1. Sauf la valeur 0, évidemment...

Codage en IEEE 754 - Exposant

- L'exposant peut être positif ou négatif, $\forall$ la base.
  (Exemple en base 10 : 5 mm $= 5 \cdot 10^{-3}$ m. ; 5 km $= 5 \cdot 10^{3}$ m.)
- Afin d'éviter d'avoir à coder ce signe, la norme IEEE-754 prévoit d'ajouter un *offset* (décalage) à l'exposant avec une valeur fixe.
- Au décodage, il faudra bien sur **retrancher** cette même valeur.
- Les valeurs de ce décalage sont :
  - Simple précision : d=127
  - Double précision : d= 1023
- Exemple 1 (simple précision) : E=3 $\Rightarrow$ on encode la valeur $127 + 3 = 130 = (1000.0010)_2$
- Exemple 2 (simple précision) : E=-19 $\Rightarrow$ on encode la valeur $127 - 19 = 108 = (0110.1100)_2$

IEEE 754 - nombres particuliers

- Certaines valeurs donnent lieu à un codage particulier :
  (X : valeur quelconque)

Valeur	S	Mantisse	Exposant
Zéro	0	0...0	0...0
NAN	X	X...X, sauf 0...0	1...1
$+\infty$	0	0...0	1...1
$-\infty$	1	0...0	1...1

- NAN (*Not A Number*) est utilisé pour signaler des erreurs de calcul. Par exemple, c'est la valeur renvoyée en cas de tentative de calcul d'une racine carrée d'un nombre négatif.
- ∞ est la valeur renvoyée par exemple en cas de division par 0.

Code ASCII

- 1963, codage des caractères latins non accentués uniquement.
- Codage sur 7 bits : `00h` à `7Fh`.
- Les caractères de numéro 0 à 31 (0 à `1Fh`) et le 127 (`7Fh`) ne sont pas affichables ; ils correspondent à des commandes de contrôle de terminal informatique.

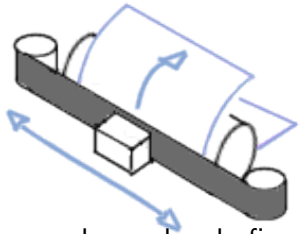
En hexadécimal

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
10	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
20	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Exemple : 'A' = 65 = $41

Codes de contrôle de l'ASCII

- DEL (*Delete*) : effacement. Correspond sur un clavier contemporain à la touche du même nom. (retour arrière en effaçant).
- Fin de ligne : deux codes sont utilisés
  - LF (*Line Feed*), saut de ligne, code 10 (0x0A)
  - CR (*Carriage Return*), retour chariot, code 13 (0x0D)
- Historiquement, ceci correspondait aux deux opérations nécéssaires sur une imprimante à rouleaux :
  - faire tourner le rouleau d'un cran,
  - ramener la tête d'impression à gauche.



- Ceci perdure dans l'informatique moderne, à travers les codes de fin de ligne dans les fichiers texte :
  - Certains OS (Windows) utilisent les 2 codes (CR suivi de LF).
  - D'autres (Linux, Mac OS X, etc.) n'en utilisent qu'un seul (LF).
- D'où des problèmes parfois lors de transferts de fichiers entre machines.

Extensions aux code ASCII

- Pour pouvoir coder les caractères de chaque langue, on a d'abord utilisé les 127 autres valeurs (0x80 à 0xFF).
- Problème : insuffisant pour tout coder.
- Solution : notion de **page de code** : on spécifie en amont quel jeu de caractère est utilisé pour les valeurs entre 128 et 255.

Page de code 850 (DOS Latin 1) — mode standard[1]

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
8x	Ç	ü	é	â	ä	à	å	ç	ê	ë	è	ï	î	ì	Ä	Å
9x	É	æ	Æ	ô	ö	ò	û	ù	ÿ	Ö	Ü	ø	£	Ø	×	ƒ
Ax	á	í	ó	ú	ñ	Ñ	ª	º	¿	®	¬	½	¼	¡	«	»
Bx	░	▒	▓	│	┤	Á	Â	À	©	╣	║	╗	╝	¢	¥	┐
Cx	└	┴	┬	├	─	┼	ã	Ã	╚	╔	╩	╦	╠	═	╬	¤
Dx	ð	Ð	Ê	Ë	È	ı	Í	Î	Ï	┘	┌	█	▄	¦	Ì	▀
Ex	Ó	ß	Ô	Ò	õ	Õ	µ	þ	Þ	Ú	Û	Ù	ý	Ý	¯	´
Fx	SHY	±	‗	¾	¶	§	÷	¸	°	¨	·	¹	³	²	■	NBSP

Extensions aux code ASCII

- Quelques pages de codes usuelles :
  - CP 850
  - CP 1252
  - ISO 8859-1 (Latin-1)
  - ISO 8859-15 (Latin-9)
  - Windows-1252
  - etc.
- Problème : multiples pages de codes, et interopérabilité limitée.
- Impossible d'avoir dans un même document des symboles spécifiques à deux pages de code différentes !

Unicode

- Standard informatique qui permet des échanges de textes dans différentes langues, à un niveau mondial.
- Extension de la norme ISO-10646, 109 000 caractères couvrant 93 écritures.
  - Chaque caractère abstrait est identifié par un nom unique (un en anglais et un en français) et associé à un nombre entier positif appelé son **point de code** (différent de son codage !)
  - L'espace de codage est divisé en 17 zones de 65 536 points de codes. Ces zones sont appelées **plans**.
  - Le point de code est noté U+xxxx où xxxx est en hexadécimal, et comporte 4 à 6 chiffres :
    - 4 chiffres pour le premier plan, appelé plan multilingue de base (donc entre U+0000 et U+FFFF) ;
    - 5 chiffres pour les 15 plans suivants (entre U+10000 et U+FFFFF) ;
    - 6 chiffres pour le dernier plan (entre U+100000 et U+10FFFF).

Unicode : encodage

- La transformation d'un point de code passe par une **représentation** en UTF-8, UTF-16 ou UTF-32.
- Le nombre après "UTF" spécifie le nombre d'octets **minimal** avec lequel un caractère est codé.
  - Windows utilise en interne UTF-16 : chaque caractère est codé sur 2 octets.
  - Linux utilise par défaut l'UTF-8.

UTF-8

- Code de taille variable, compatibilité ascendante avec l'ASCII 7 bits.

Représentation binaire UTF-8	Signification
0xxxxxxx	1 octet codant 1 à 7 bits
110xxxxx 10xxxxxx	2 octets codant 8 à 11 bits
1110xxxx 10xxxxxx 10xxxxxx	3 octets codant 12 à 16 bits
11110xxx 10xxxxxx 10xxxxxx 10xxxxxx	4 octets codant 17 à 21 bits

- Remarques
  - Tout octet de bit de poids fort **nul** désigne un point de code assigné à un caractère ASCII, codé sur ce seul octet ;
  - Tout octet de bits de poids fort valant 11 est le premier octet d'une séquence représentant un point de code codé sur plusieurs octets ;
  - Tout octet de bits de poids fort valant 10 est un des octets suivants d'une séquence unique représentant un point de code codé sur plusieurs octets ;