La théorie classique de l'information.

1ère partie : le point de vue de Kolmogorov.

La suite de caractères comme outil de description des systèmes.

La science peut être vue comme l'art de compresser les données quelles qu'en soient la provenance et la signification du contenu. Ces données, issues le plus souvent d'un ensemble de mesures expérimentales, peuvent très généralement être encodées dans de grands fichiers assimilables à de longues suites de caractères. Voici quelques exemples choisis :

1) Suite de nucléotides, longue de 66495 bases (Adénine, Cytosine, Guanine, Thymine), au début de la séquence contenant les gènes d'hormone de croissance et d'hormone chorionique humaine (extrait de : O. Delgrange, La Compression Informatique)

```
GAATTCCTGGGCCTGGGGCTGTGGCAGCTGCCTCGTCCCTTCACCTCCTGGCTTATTCTCTCCCTCCATATCTTA
GCAATTTCTTCATGGGAATGTCCCCAATTAGAAATTTCTATTATACCATTATATTACCAACATATATATATATAC
CTGGCCGGGCGCGGTGGCTCATGCCTGTAATCCCAGCATTTTGGTAGGCCAAGGCGGGTCGGATCACCTGAGGTC
AGGAGTTCGAGGGCCAGCCTGATGACCATGGTGAAAC.....
```

2) Suite de tirages binaires (ex. pile = '0' ou face = '1') :

1001010110000110100111100101010010 0 …

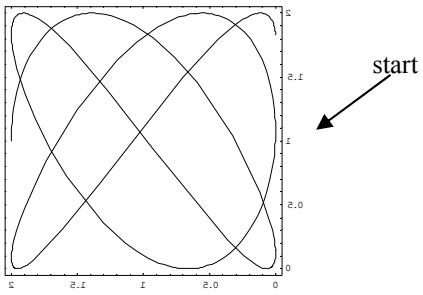
3) Suite de nombres, digitalisant sur 8 ou 16 bits un signal sonore ou graphique, un texte écrit dans n'importe quelle langue, une conversation téléphonique, … :

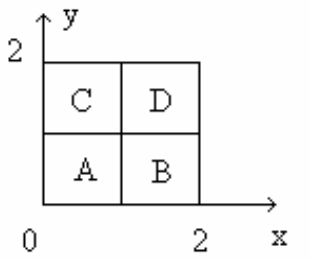
11011101 11011110 11011111 11011111 …

4) Suite de nombres, {x, y}, extraits d'un tableau de « mesures », peu importe pour l'instant ce qu'elles représentent mais on y reviendra à la fin de l'exposé :

```
{{3.00000, 0}, {2.89056, 0.987899}, {2.58063, 1.90851},
 {2.11504, 2.71611}, {1.54502, 3.39221}, {0.914311, 3.93810},
 {0.255055, 4.36503}, {-0.410933, 4.68749}, {-1.06946, 4.91983},
 {-1.71150, 5.07492}, {-2.33143, 5.16379}, {-2.92585, 5.19576},
 {-3.49274, 5.17860}, {-4.03101, 5.11886}, {-4.54017, 5.02204},
 {-5.02008, 4.89280}, {-5.47086, 4.73510}, {-5.89276, 4.55236},
 {-6.28612, 4.34754}, {-6.65132, 4.12321}, {-6.98876, 3.88166}}
```

5) Suite étiquetée des sites visités par un point matériel qui décrit une trajectoire dans l'espace des phases. Cet espace est préalablement partitionné en N cellules auxquelles on colle une étiquette codée dans un alphabet à N symboles. La dimension des cellules dépend de la précision visée. On capture l'essentiel de l'information concernant l'évolution du point matériel en écrivant la suite des codes des cellules visitées à intervalles de temps réguliers suffisamment rapprochés. Voici l'exemple d'un point qui décrit une courbe de Lissajous :





$$\begin{cases} x(t) = 1 + \cos(\sqrt{5}t) \\ y(t) = 1 + \sin(3t) \end{cases}$$

$$\{x(t) \in [0,2]\} \cup y(t) \in [0,2]\}$$

Echantillonnée aux temps discrets, t = 1/10, 2/10, … , la trajectoire peut être encodée par la suite suivante :

DDDDDDDDCCCAAAAAAAAAABDDDDDDDDDDBBBAAAAAACCCCCCCCD …

On peut naturellement améliorer la précision de l'encodage en partitionnant l'espace plus finement mais cela allonge l'alphabet.

L'étude des suites dans le cadre d'une théorie classique de l'information.

Les exemples qui précèdent ont en commun de consigner l'information dans des suites de caractères prélevés dans un alphabet qui en comporte C, $\{a_1, a_2, \ldots, a_C\}$. La dimension de l'alphabet n'a pas réellement d'importance et il est toujours possible de réencoder la suite sous forme binaire en n'utilisant que les caractères, '0' et '1', appelés bits dans ce cas précis.

Le fait que la suite soit l'outil de prédilection pour l'encodage des données amène à poser un ensemble de questions fondamentales parmi lesquelles nous en retiendrons quatre :

Comment encoder une suite sur un support en sorte qu'elle soit décodable sans ambiguïté ?
Comment effectuer cet encodage le plus économiquement possible ?
Comment transmettre économiquement le contenu d'une suite à un correspondant lointain ?
Comment formaliser et quantifier le contenu informationnel d'une suite ?

La théorie classique de l'information a notamment pour objet de répondre à ces questions. Appliquée à des systèmes physiques macroscopiques qui encodent et manipulent les caractères de l'alphabet, elle est à la base de la révolution informatique.

La théorie de l'information étudie bien d'autres questions : reconstruction d'un signal échantillonné, protection contre le bruit de l'encodage ou de la transmission d'un fichier, corrections d'erreurs, compression avec pertes, …, qui relèvent davantage du domaine de l'ingénieur et qui ne concernent pas directement la physique. Signalons encore l'existence d'une théorie quantique de l'information. La première différence se situe au niveau du support physique qui réalise l'encodage de l'information, support classique donc macroscopique dans le cas qui nous occupe ici et quantique donc à l'échelle atomique dans la théorie quantique. D'autres différences essentielles surgissent dues aux lois qui régissent le monde quantique, très différentes de celles qui nous sont familières. La théorie quantique de l'information fait l'objet d'exposés séparés.

Encodage et mémorisation d'une suite : le problème du délimiteur « endoffile ».

Il existe C^N suites distinctes, de longueur N, écrites dans un alphabet comprenant C caractères. Par exemple, il existe $2^4 = 16$ suites binaires de longueur, N=4 :

0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111.

La mémorisation d'une suite particulière à l'état brut, c'est-à-dire caractère après caractère, exige un emplacement en mémoire égal à sa longueur, soit, $\ell g_C(C^N) = N$, exprimé dans une unité qu'on pourrait baptiser C-bit (bit si C=2, trit si C=3, etc …). Il est d'usage de n'utiliser que le bit comme unité d'information, d'où toute suite occupe, $\ell g(C^N) = N\ell g(C)$ bits, en mémoire (ℓg , sans indice, désigne le logarithme en base 2). En particulier, si la suite est binaire, C=2, N bits sont nécessaires.

L'inventaire complet des suites, quelles qu'en soit la longueur, est facile à dresser dans un ordre canonique :

Λ={ε={},{0},{1},{00},{01},{10},{11},{000},{001},{010},{011},{100},{101},{110},{111},{0000},{0001},{0010},…}

Soit l'une quelconque de ces suites, de longueur, N, que l'on cherche à mémoriser au début d'un support quelconque, conventionnellement appelé disque dur (DD). Dans l'exemple qui suit, cette suite est binaire pour simplifier :

$s = 00110101010001000\ldots^N$

Sans précautions particulières, sa mémorisation à l'état brut, au début du DD, fait craindre qu'elle se trouve noyée dans l'ensemble des caractères déjà écrits sur le DD (éventuellement un paquet de '0') sans qu'il soit possible de reconnaître l'endroit exact où elle se termine. L'idée qui vient immédiatement à l'esprit d'utiliser un séparateur, une virgule ou n'importe quel signe distinctif, #, ne convient pas car elle signifie l'introduction d'un caractère intrus dans l'alphabet binaire. On peut régler ce problème de plusieurs façons plus ou moins élégantes en recourant à une technique de délimitation externe.

Réencodage double. Si on duplique chaque bit de la suite, conformément aux règles, $0 \Rightarrow 00$ et $1 \Rightarrow 11$, on récupère les doublets, 01 ou 10, pour l'encodage du symbole terminal, #, (en gras dans l'exemple) : $\# \Rightarrow 01$ (ou 10). Avec cette stratégie, la suite s^N se réécrirait :

$s \Rightarrow 0000111100110011001100000011000000\ldots\mathbf{01}^{2N+2}$

On voit que la suite est devenue parfaitement décodable grâce au doublet '01' terminal. Toutefois sa longueur a plus que doublé ce qui représente un gaspillage inacceptable.

Codage préfixe variable. Un codage variable ne consomme pas le même nombre de bits pour l'encodage de tous les symboles du message d'origine. On comprend intuitivement qu'un codage variable puisse être intéressant pour encoder économiquement une suite toutes les fois que les symboles alphabétiques n'y apparaissent pas avec des fréquences égales : il suffit de réserver les codes courts aux symboles fréquents. Toutefois la variabilité du codage ne garantit pas qu'il soit uniquement décodable. De fait, le codage variable suivant est ambigu :

$0 \Rightarrow 0$, $1 \Rightarrow 01$, $\# \Rightarrow 10$.

On voit que '0' étant préfixe de '01', jamais le décodeur qui commence à lire '0010…' ne saura s'il doit comprendre '00#' ou '010…'. Pour être uniquement décodable, un codage variable doit obligatoirement être (libre de tout) préfixe.

Un codage est préfixe si aucun des codes qui le constituent n'est le préfixe d'un autre code. De plus, il est dit complet si aucune adjonction de code n'est possible sans briser le caractère préfixe de l'ensemble. Par exemple le codage suivant est préfixe complet :

$0 \Rightarrow 0$, $1 \Rightarrow 10$, $\# \Rightarrow 11$

Il est complet parce que tout symbole supplémentaire commencerait fatalement par 0, 10 ou 11. Ce codage préfixe permet l'encodage uniquement décodable de la suite, s, sous la forme :

s = 00110101010001000… $\Rightarrow$ 00101001001001000010000…**11**

Il semblerait que ce codage préfixe soit coûteux puisque dans le cas d'une suite aléatoire, on passe, en moyenne, de 1 bit/symbole (en abrégé 1 b/s) à : (1+2)/2 = 1.5 b/s (on néglige provisoirement les deux bits qui encodent le symbole terminal, #, qui n'apparaît qu'une seule fois en fin de suite). Toutefois, on améliorerait la situation en lisant la suite par blocs de deux caractères comme si elle avait été écrite dans un alphabet comprenant quatre caractères, les blocs, '00', '01', '10' et '11'. Voici un codage préfixe complet valable pour un alphabet à quatre symboles plus un séparateur :

$00 \Rightarrow 00$, $01 \Rightarrow 01$, $10 \Rightarrow 10$, $11 \Rightarrow 110$, $\# \Rightarrow 111$

Remarque : il conviendrait tout aussi bien à l'encodage d'une séquence ADN :
$A \Rightarrow 00$, $C \Rightarrow 01$, $G \Rightarrow 10$, $T \Rightarrow 110$, $\# \Rightarrow 111$.

La suite de départ s'encode cette fois comme suit :

s = 00110101010001000… $\Rightarrow$ 001100101010001000…**111**

Si la suite est aléatoire, l'encodage moyen ne consomme plus que :

(1/2) (3x2+3)/4 = 1.125 b/s.

On améliorerait encore la situation en lisant la suite par blocs successifs de trois bits et en utilisant le code :

$000 \Rightarrow 000$, $001 \Rightarrow 001$, $010 \Rightarrow 010$, $011 \Rightarrow 011$,
$100 \Rightarrow 100$, $101 \Rightarrow 101$, $110 \Rightarrow 110$, $111 \Rightarrow 1110$, $\# \Rightarrow 1111$,

pour une longueur moyenne d'encodage tombant à : (1/3) (7x3+4)/8 = 1.041 b/s.

Le codage préfixe moyen semble tendre vers 1 b/s à mesure que la dimension, n, des blocs augmente mais c'est sans compter qu'il faut incorporer la valeur de n quelque part dans l'encodage de la suite ainsi que le détail des codes particuliers utilisés. Sans cela le décodeur sera incapable d'effectuer son travail. Ces détails de décodabilité ont un prix que l'on peut

estimer avec précision : on trouve que l'encodage préfixe variable d'une suite quelconque de longueur, N, consomme, en moyenne un nombre de bits qui est de l'ordre de :

$$N + \ell g(N) + \ell g(\ell g(N)) + \ell g(\ell g(\ell g(N))) + \cdots \approx N + \ell g^*(N)$$

où on a introduit la notation condensée, $\ell g^*(n) = \sum_{k=1}^{\infty} \underbrace{\ell g \ell g \cdots \ell g}_{k \ fois}(n)$.

L'encodage variable est couramment utilisé en informatique mais il n'est pas le plus avantageux du point de vue théorique qui nous intéresse ici. En particulier, le découpage de la suite d'origine en blocs de longueurs constantes, n, introduit une difficulté qui réclame des aménagements toutes les fois que N n'est pas un multiple de n. Une méthode, théoriquement idéale, existe qui est basée sur l'auto délimitation.

Suites préfixes auto délimitées.

On peut se passer du symbole terminal, #, en procédant comme suit. On préfixe la suite binaire par une étiquette, en fait un entier binaire, qui dicte sa longueur. Comme il n'existe aucun moyen de savoir où l'étiquette se termine et où la suite commence, on fait pareil avec l'étiquette. On poursuit itérativement jusqu'à rencontrer l'une ou l'autre des étiquettes à deux bits, 10 ou 11, ce qui ne peut manquer de se produire. A ce stade, il ne reste plus qu'à préfixer le tout par le nombre total d'étiquettes utilisées, en notation unaire n'utilisant que le chiffre 0. Une économie est possible à tous les stades de l'itération du fait que toute étiquette binaire commençant nécessairement par le chiffre 1, celui-ci peut être omis (il est mis entre parenthèses dans l'exemple qui suit). Seules les étiquettes terminales, 10 ou 11, doivent être écrites in extenso. Nous illustrons la méthode en encodant la suite, *10001*, de longueur, 5. 5 se note '101' en binaire où le '1' initial peut être sous-entendu ce qui laisse subsister, '01'. '01' est de longueur, 2, soit '10' en binaire. '10' est donc l'étiquette terminale, écrite in extenso. On préfixe le tout par un double '0' puisque deux étiquettes ont été nécessaires au total.

$$10001 \Rightarrow 10001^{5=(1)01^{2=10}} \Rightarrow 00 \ \ 10 \ \ (1)01 \ \ 10001 \Rightarrow 00100110001$$

Le programme Mathematica suivant automatise l'encodage :

```
li={1,0,0,0,1};longeti=Length[li];i=0;While[longeti>3,{eti=Rest[IntegerDigi
ts[longeti,2]];li=Join[eti,li],longeti=Length[eti]};i++];Join[Table[0,{i+1}
],IntegerDigits[longeti,2],li]
```

```
{0,0,1,0,0,1,1,0,0,0,1}
```

Le décodage s'effectue comme suit. Soit à décoder, 00100110001… noyée à droite dans une mer de 0 et de 1 qui polluent éventuellement le DD. On commence par compter combien de 0 préfixent le message, en fait, 2. Deux étiquettes sont donc nécessaires. La première étiquette est toujours 10 ou 11, ici c'est 10. Elle impose la longueur de la deuxième étiquette à la valeur 2 (=10 en binaire). Celle-ci vaut '01', en fait '101' car le '1' initial a été omis à l'encodage, soit le nombre 5. Il convient donc de lire les 5 bits suivants qui constituent la suite cherchée, en l'occurrence, 10001.

Cette procédure échoue pour la suite vide et les suites à un seul élément, {0} et {1}. On les encode séparément sous les formes suivantes, {1,0}, {1,1,0} et {1,1,1}, ce qui ne mène à aucune ambiguïté puisqu'elles sont les seules à commencer par le chiffre 1. Voici le détail des encodages des premières suites, en commençant par la suite vide :

$\varepsilon \Rightarrow \{1,0\}$ $\{0\} \Rightarrow \{1,1,0\}$ $\{1\} \Rightarrow \{1,1,1\}$

$\{0,0\} \Rightarrow \{0,1,0,0,0\}$ $\{0,1\} \Rightarrow \{0,1,0,0,1\}$ $\{1,0\} \Rightarrow \{0,1,0,1,0\}$ $\{1,1\} \Rightarrow \{0,1,0,1,1\}$

$\{0,0,0\} \Rightarrow \{0,1,1,0,0,0\}$ $\{0,0,1\} \Rightarrow \{0,1,1,0,0,1\}$ $\{0,1,0\} \Rightarrow \{0,1,1,0,1,0\}$
$\{0,1,1\} \Rightarrow \{0,1,1,0,1,1\}$ $\{1,0,0\} \Rightarrow \{0,1,1,1,0,0\}$ $\{1,0,1\} \Rightarrow \{0,1,1,1,0,1\}$
$\{1,1,0\} \Rightarrow \{0,1,1,1,1,0\}$ $\{1,1,1\} \Rightarrow \{0,1,1,1,1,1\}$

$\{0,0,0,0\} \Rightarrow \{0,0,1,0,0,0,0,0,0,0\}$ $\{0,0,0,1\} \Rightarrow \{0,0,1,0,0,0,0,0,0,1\}$ …

Voici, en résumé, la liste des suites préfixes écrite dans l'ordre canonique :

$\Lambda_{pref}=\{\{10\},\{110\},\{111\},\{01000\},\{01001\},\{01010\},\{01011\},\{011000\},\{011001\},\{011010\},\{011011\},\{011100\},\{011101\},\{011110\},\{011111\},\{0010000000\},\{0010000001\},\ldots\}$

Toutes les suites de longueurs, N, sont encodées par une suite de longueur, $l(N)>N$, ou l'allongement, $l(N)-N$, est le prix que l'on paye pour la décodabilité. La fonction, $l(N)$, croît de façon plus ou moins régulière, on trouve pour, N =0,1, 2, 3, 4, … :

$l(N)=\{2,3,5,6,10,11,12,13,15,16,17,18,19,20,21,22,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,44,\ldots\}$

La suite, $l(N)$, se comporte asymptotiquement comme :

$$l(N) \approx N + \ell g(N) + \ell g(\ell g(N)) + \ell g(\ell g(\ell g(N))) + \cdots \approx N + \ell g^*(N).$$

On trouve ce comportement en observant que l'encodage consiste à préfixer itérativement la suite d'origine par des étiquettes de plus en plus courtes qui ont typiquement pour longueur le logarithme en base deux de la suite qu'elles étiquettent.

L'(in)égalité de Kraft.

Tous les codages préfixes partagent une propriété qui s'avère essentielle pour la suite : ils obéissent à l'(in)égalité de Kraft,

$$\sum_i 2^{-\ell_i} \leq 1$$

où les ℓ_i représentent les longueurs des codes et où le cas d'égalité à 1 correspond aux codages complets. Illustrons cette relation sur le codage fini préfixe et complet,

$000 \Rightarrow 000$, $001 \Rightarrow 001$, $010 \Rightarrow 010$, $011 \Rightarrow 011$,
$100 \Rightarrow 100$, $101 \Rightarrow 101$, $110 \Rightarrow 110$, $111 \Rightarrow 1110$, $\# \Rightarrow 1111$.

On y dénombre sept codes de longueur 3 et deux codes de longueur 4. On vérifie que l'on a effectivement,

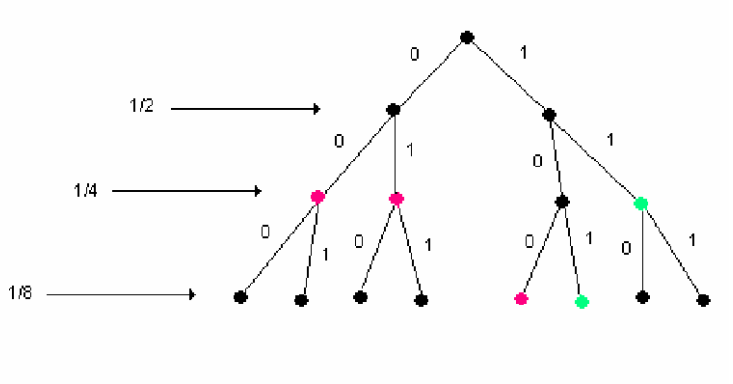
$$7\times 2^{-3}+2\times 2^{-4}=1$$

et que l'égalité se transforme en une inégalité si l'on ampute ce code complet,.

Contentons-nous de donner une « démonstration graphique » de cette inégalité importante. A partir d'un nœud initial situé au sommet, dessinons un arbre binaire descendant. A chaque dédoublement, les branches naissantes sont étiquetées '0' et '1'. Affectons les poids 2^{-k} aux générations successives (k=1, 2, ...) et sélectionnons un nombre fini de nœuds de telle manière qu'aucun nœud ne possède un nœud déjà choisi parmi ses ascendants. Ces nœuds définissent un code préfixe à condition de lire les symboles en partant du sommet et en descendant vers chaque nœud séparément. Le code obtenu est complet s'il n'y a plus moyen d'ajouter un nœud qui respecte la condition préfixe. On constate qu'un code complet est toujours tel qu'il satisfait l'égalité

$$\sum_i 2^{-\ell_i}=1$$

alors qu'un code incomplet satisfait l'inégalité correspondante.



Dans la figure ci-dessus, les nœuds roses définissent un code préfixe incomplet,'00', '01' et '100' , que les deux nœuds verts viennent compléter par '101' et '11'.

Compression des suites.

La compression des suites obéit certainement à des motivations utilitaires : en effet une suite compressée occupe un espace mémoire réduit sur un DD. De même, l'envoi d'une suite compressée à un correspondant lointain, via un canal quelconque, économise les ressources de transmission. Nous reviendrons sur ces aspects pratiques dans l'exposé consacré à la théorie de Shannon. Ici nous nous préoccupons uniquement de l'étude théorique relative à la compression des suites.

Il revient à Kolmogorov d'avoir compris en premier que compresser une suite c'est comprendre sa logique interne et par conséquent que c'est pénétrer l'organisation du système que cette suite décrit. L'idée de base apparaît clairement lorsqu'on considère la suite suivante, particulièrement régulière :

s1 = '0101010101010101010101010101010101010101010101010101010101010101…' ^{N}

qui répète N/2 fois le motif '01'. On est en droit de penser que consommer N bits pour la mémoriser sur un DD représente un fameux gaspillage. De même, si l'on veut communiquer son contenu à un correspondant lointain, il serait du plus haut ridicule de la transmettre telle quelle alors qu'il suffirait d'envoyer un programme nettement plus court, du style « Répéter N/2 fois le motif '01' ». Dans l'optique de Kolmogorov, compresser une suite revient à substituer la « recette » de la suite à la suite elle-même.

Par définition, on dit qu'une suite, s, de longueur, N, est compressible s'il existe un programme, p, de longueur, M, inférieure à N, qui est capable de la reconstituer sans perte, Output[p] = s. Si ce programme n'existe pas, la suite est incompressible.

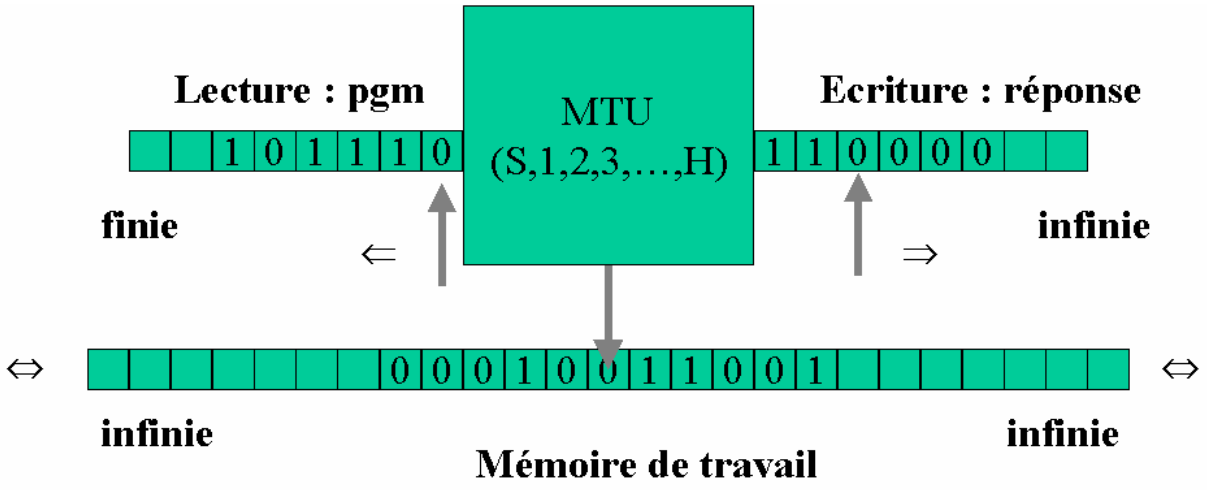
Dans cette définition, il faut comprendre, une fois pour toutes, que tant la suite que le programme qui la compresse doivent être uniquement décodables.

La suite s1 est compressible : en effet, le programme suivant la compresse sans perte d'information (peu importe le langage de programmation utilisé) :

p(s1) = Do[Print['01'],{k,1,N/2}]

La longueur de p(s1) correspond en gros, c'est-à-dire à des termes constants inessentiels près, à l'encodage de la valeur de N soit, ℓgN bits, ce qui est bien inférieur à N. Les termes constants sont inessentiels dans la mesure où l'on ne s'intéresse qu'à de très longues suites (N>>1) d'où il résulte que seuls les termes asymptotiquement croissants comptent.

On peut remplacer la notion de programme par celle du plus petit numéro, dans la notation de Wolfram, de la machine de Turing qui, travaillant sur une bande initialement vierge, s'arrête après avoir imprimé la suite. Mieux, on peut se mettre d'accord sur une machine de Turing universelle et rechercher le plus court fichier auto délimité à inscrire préalablement sur la bande de lecture de cette MTU afin qu'elle s'arrête après avoir écrit la suite donnée sur la bande d'écriture. Rappelons que l'auto délimitation est importante car on doit pouvoir distinguer, sur la bande de lecture de la MTU, où se termine le programme qui provoque le calcul de la suite considérée. L'étude des MTU est détaillée par ailleurs.



Dans cet exposé, nous utilisons une variante de MTU à trois bandes. La bande de lecture est finie et héberge le programme décodable. La bande d'écriture, potentiellement semi infinie, reçoit les résultats, c'est-à-dire la suite visée, également décodable. Enfin, la bande de travail, potentiellement doublement infinie, sert aux calculs intermédiaires. Comme dans toute MTU, l'unité centrale peut se trouver dans un nombre fini d'état dont l'un, S, est l'état de départ et l'autre, H, est l'état sur lequel la MTU arrête son calcul pour autant qu'elle y parvienne un jour.

Imaginons un observateur qui désire envoyer une suite, s, à un correspondant lointain et qui tente, pour ce faire, de la compresser par un programme plus court qu'elle. Pour le receveur tout est simple : il reçoit la suite compressée sous la forme d'un programme qu'il n'a qu'à inscrire sur la bande de lecture de sa MTU. Une fois le calcul lancé, l'arrêt de la MTU est certain car l'encodeur ne promet que des programmes bien formés et le résultat sera la suite, s, inscrite in extenso dans un format uniquement décodable sur la bande d'écriture.

La situation de l'expéditeur est beaucoup moins confortable car c'est à lui qu'incombe la tâche de trouver un programme court qui est capable de reconstruire la suite sans perte. Or il n'existe pas de procédure algorithmique donc de programme qui effectuerait à tout coup la tâche de lire n'importe quelle suite, s, et de s'arrêter en écrivant un de ses programmes compresseurs s'il en existe au moins un ou de répondre « impossible » s'il n'en n'existe pas. En bref, il n'est pas garanti de pouvoir trouver un programme compresseur d'une suite quelconque et encore moins de trouver le plus court de tous.

Complexité algorithmique d'une suite.

Si on connaît un programme compresseur d'une suite, on sait que celle-ci est compressible et rien n'empêche a priori qu'on puisse en trouver d'autres. Parmi l'ensemble des programmes compresseurs, il y en a forcément un qui est le plus court de tous. On définit précisément la complexité algorithmique d'une suite comme la longueur du plus court programme (auto délimité) capable de la reconstituer sans perte :

$$K(s) = \min_{p:MTU(p)=s} \ell(p)$$

On peut montrer que le choix de la MTU n'a pas vraiment d'importance. Certes, deux MTU distinctes livreront des complexités différentes mais la différence ne portera que sur un nombre fini et constant de bits, précisément la longueur du programme capable d'émuler une MTU à partir de l'autre. Cette constante additive ne joue aucun rôle significatif pour les très longues suites considérées en pratique du fait que $K(s^N)$ croît au minimum comme ℓgN.

Reprenons l'exemple de la suite s1 :

s1 = '0101010101010101010101010101010101010101010101010101010101010101…'.

Le programme, Print[s1], qui imprime la suite in extenso requiert, en gros, N bits : une machine à écrire ne ferait pas plus mal !

Nous connaissons une compression possible de s1 qui consomme ℓgN bits :

Do[Print['01'],{k,1,N/2}]

En voici une autre, équivalente en ressources consommées, qui calcule, en base deux, le développement fractionnaire de 1/3 :

Print[Frac[1/3],base$_2$,N]

Ces deux programmes sont optimaux car il n'y a pas moyen de faire mieux. En effet, tout programme compresseur doit au minimum spécifier la longueur de la suite qu'il entend compresser d'où il résulte que, à une constante inessentielle près, sa complexité ne peut descendre en dessous de ℓgN bits. Puisque à l'opposé, la complexité ne peut excéder la longueur, N, de l'encodage in extenso, on a que la complexité d'une suite est, en gros, comprise entre deux limites extrêmes :

$$\ell gN \quad \leq K(s^N)+c \leq \quad N$$

Compression maximum

Incompressibilité = Suite aléatoire

Ramenée à un symbole binaire, l'entropie algorithmique d'une suite est donc comprise entre $\ell g(N)/N$ et 1, soit entre 0 et 1 bit/symb, à la limite des suites infinies. Etudions de plus près les deux extrêmes.

Incompressible ⇔ aléatoire !

Une suite incompressible n'obéit à aucune logique interne qui pourrait être mise à profit pour la compresser : il se fait qu'il n'existe pas de meilleure définition du caractère aléatoire d'une suite. Les fondements modernes du calcul des probabilités, dus à Kolmogorov, posent précisément que la suite incompressible définit la suite aléatoire et inversement donc que ces deux notions se recouvrent exactement.

Il est intéressant de confronter cette définition à l'intuition. En supposant que nous voulions transmettre à un correspondant lointain la suite des N tirages successifs d'un processus binaire complètement aléatoire,

'10111001011111000101011100111011000110010111101101101100011101011110…',

on ne voit, de fait, pas comment on pourrait travailler autrement que bit par bit, sans espoir de compression. A cet égard, il est satisfaisant qu'une suite aléatoire soit incompressible.

Exemples de systèmes fortement compressibles.

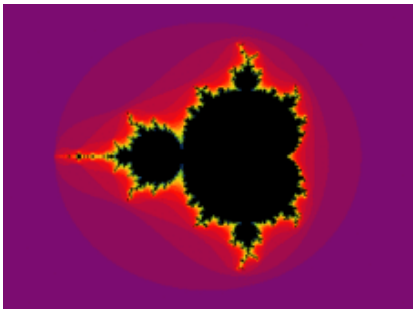
La suite des chiffres dans l'écriture binaire du nombre π-3 est fortement compressible car il existe de nombreux algorithmes courts qui calculent ce nombre :

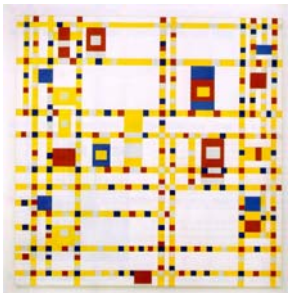
00100100001111110110101010001000100001011010001100001000110100110001001100011001100010100001011000

De même la suite des chiffres du dix milliardième nombre premier,

1110101011001000110011001010011010111 1,

l'ensemble fractal de Mandelbrot ou … un tableau de Mondrian :





L'immense majorité des suites de longueur N sont incompressibles tout simplement parce qu'il est impossible qu'il en soit autrement. Déjà, il existe 2^N suites binaires de longueur N alors qu'il n'existe que $\sum_{k=0}^{N-1} 2^k = 2^N - 1$ programmes binaires distincts plus courts. On voit qu'une suite au moins ne sera pas comprimée. Plus généralement, suivant le même argument, seule une petite fraction, $1/2^n$, pourrait être compressée de n bits, 1/1000 si n=10 !

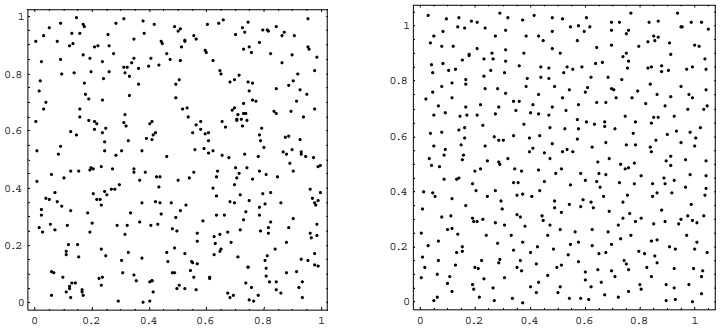
La réalité est bien pire que cela car la plupart des suites (courtes ou non) ne constituent pas un programme générique de suite. En effet, tout programme de compression, p(s), d'une suite, s, de longueur, N, peut être vu comme la suite binaire qui alimente la bande de lecture d'une MTU afin qu'après exécution, celle-ci s'arrête en imprimant la suite donnée s. Si on alimente cette MTU successivement par chacune des 2^N-1 suites courtes, on constate rapidement que la plupart du temps, la MTU boucle ou se retrouve bloquée du fait que le programme correspondant est syntaxiquement incorrect. Même lorsqu'il n'y a aucune faute de programmation, on sait que la MTU peut calculer sans répit sans qu'on soit capable de prédire son arrêt éventuel. Au bilan, il n'y a qu'un très petit nombre de programmes qui répondent à la question posée sans parler de ceux qui impriment une suite déjà trouvée.

On peut comprendre autrement l'extrême rareté des suites compressibles. Le fait qu'un programme valable est nécessairement de longueur finie et que l'ensemble des suites finies est dénombrable a pour conséquence que l'ensemble des suites compressibles est, lui aussi, dénombrable. Par contre, l'ensemble de toutes les suites existantes, finies ou infinies, n'est pas dénombrable. On en conclut que l'immense majorité des suites sont incompressibles donc aléatoires !

Dit encore autrement, toute suite binaire est équivalente à la limite infinie à un réel compris entre 0 et 1 et on sait que l'ensemble de ces réels n'est pas dénombrable. Or comme l'ensemble des machines de Turing est dénombrable, on en conclut à nouveau qu'il y a infiniment plus de suites que de programmes compresseurs.

Génération des suites aléatoires.

Les suites aléatoires étant incompressibles, elles sont infiniment plus nombreuses que les suites régulières essentiellement parce que, rappelons-le, il y a beaucoup plus de suites que d'algorithmes compresseurs. Paradoxalement, il est difficile de produire une suite aléatoire. C'est même, par définition, une tâche quasiment impossible pour le cerveau humain qui tôt ou tard se trahit par un recours souterrain à un algorithme inconscient. Le test classique consistant à demander à un sujet de placer 400 points au hasard dans un carré révèle invariablement une compressibilité indésirable.



Test : quel motif est aléatoire ?

Pour engendrer une suite véritablement aléatoire, une expérience quantique devrait pouvoir faire l'affaire. Un dispositif envisageable consiste à réaliser de façon répétée l'expérience des deux fentes d'Young avec un seul photon. Deux détecteurs placés en positions symétriques se voient affecter les symboles '0' et '1' respectivement. Chaque fois qu'un détecteur enregistre l'arrivée d'un photon, la suite est complétée du symbole correspondant.

Dans la pratique courante, on se contente souvent du lancer d'une pièce de monnaie non biaisée ou de dispositifs équivalents telle la roulette de casino. On pourrait penser que de telles procédures ne devraient pas pouvoir livrer de suites aléatoires du fait que la trajectoire de la pièce ou de la roulette est en principe prédictible à partir des lois de la mécanique et des conditions initiales qui, ensemble, fonctionnent comme un algorithme plus ou moins court. Ce qui sauve la méthode, c'est le fait que la dynamique en question est chaotique. Elle fonctionne donc à condition d'agiter la pièce dans un cornet un temps suffisant en sorte qu'on puisse être sûr que l'encodage de sa position initiale consomme un nombre de chiffres binaires significatifs égal à la longueur de la suite que l'on veut construire.

Non calculabilité de la complexité algorithmique.

Si séduisante et naturelle que puisse paraître la définition de la complexité d'une suite, elle souffre d'un handicap profond et irrémédiable : il n'existe pas d'algorithme général permettant de répondre, dans tous les cas, par oui ou par non, à la question de savoir si une suite donnée est compressible ou si elle ne l'est pas. A fortiori, il est impossible de proposer un algorithme universel capable de compresser au maximum une suite si elle est compressible et de la restituer telle quelle sinon, en un mot de calculer K(s) : ces problèmes sont

indécidables au sens de Turing. Il en résulte que la définition de l'entropie algorithmique n'est pas effective ou si l'on préfère que la fonction K(s) n'est pas calculable.

Ce résultat négatif doit être tempéré par l'observation que rien n'empêche le problème de la compression algorithmique d'être soluble dans un très grand nombre de cas particuliers. En général, plus un observateur est sagace, plus il a de chance d'améliorer la compression d'une suite compressible et de se rapprocher de la valeur optimale de sa complexité algorithmique, K(s). Sauf cas particulier plutôt exceptionnel, tel celui de la suite s1, qui est compressible à l'optimum, $K(s1) \approx \ell gN$, de façon particulièrement évidente, cet observateur n'aura cependant pas souvent la certitude d'avoir atteint l'optimum. Dans cette optique, il y a lieu de penser que la théorie ultime qui serait capable de compresser au maximum théorique l'ensemble des données expérimentales de la physique a toutes les chances d'être une quête sans fin : ce problème serait lui aussi indécidable au sens de Turing.

Arguments Kolmogoroviens.

La double inégalité qui caractérise la complexité algorithmique d'une suite fournit un ensemble d'arguments heuristiques qui éclairent quelques problèmes classiques de la théorie des nombres. Nous en avons retenu quatre.

1) Il existe une infinité de nombres premiers.
Procédons par l'absurde et posons qu'un nombre premier, p_k, serait plus grand que tous les autres. On sait que tout entier, n, admet la décomposition en facteurs premiers :

$n = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$ (où $p_1=2$, $p_2=3$, … est la suite des nombres premiers)

Certains exposants peuvent éventuellement être nuls signe que le facteur correspondant est absent. La suite des exposants, $\{e_1, e_2, e_3, \ldots, e_k\}$ = code(n) fournit un codage possible de l'entier n. Par exemple le nombre $n=63=3^27$ serait codé par la suite {0, 2, 0, 1}. On a de façon évidente que :

$$\ell gn = e_1 \ell gp_1 + e_2 \ell gp_2 + \cdots + e_k \ell gp_k$$

Il résulte de cette égalité que : $e_i \leq \ell gn$.

Cette inégalité signifie que chaque exposant peut s'écrire en binaire avec un maximum de $k\ell g(\ell gn))$ chiffres. On aurait dès lors trouvé le moyen d'encoder n'importe quel entier, n, si grand soit-il, avec un maximum de $k\ell g(\ell gn))$ chiffres. Or cela est impossible car il faut en général ℓgn chiffres binaires pour encoder un entier quelconque.

2) Comment trouver simplement une borne supérieure pour le $i^{ème}$ nombre premier noté, p_i ?
Tout nombre, n, qui est multiple de p_i peut trivialement s'écrire :

$$n = p_i \frac{n}{p_i}$$

Un codage possible pour n serait donc de coder l'indice i sous forme préfixe (car un séparateur est nécessaire) puis la valeur du quotient n/p_i. Il n'y a aucune raison pour que ce codage coïncide avec la complexité du nombre mais quoi qu'il en soit, on aura toujours :

$$K(n) \le K(i, n/p_i)$$

$$\ell gn \le K(n) \le K(i, n/p_i) \le \ell gi + \ell g(\ell gi)) + \cdots + \ell gn - \ell gp_i$$

$$\ell gp_i \le \ell gi + \ell g(\ell gi)) + \cdots \qquad \Rightarrow \qquad p_i \le i \cdot \ell gi$$

On retrouve un résultat bien connu de la théorie des nombres.

3) A combien de '0' (ou de '1') consécutifs peut-on s'attendre dans une suite aléatoire de longueur, N ?

Précisons la question posée. Si un grand nombre de lanceurs de pièces de monnaie construisent chacun une suite binaire aléatoire de longueur, N, et qu'ils notent la dimension de la plus longue suite de bits consécutifs identiques quel sera la valeur la plus souvent observée ?

La réponse, $2\ell gN$, serait déraisonnable car elle signifierait qu'on observerait fréquemment une suite du type,

$$s = u00\cdots0^{2\ell gN}v$$

Dans ce cas, on devrait pouvoir mettre à profit l'abondance de '0' consécutifs pour compresser cette suite ce qui contredirait le fait qu'elle est aléatoire. En effet, on, aurait successivement :

$$N \le K(s) \le (K(u) + K(v)) + \ell g|u| + \ell g(\ell g|u|) + \cdots$$
$$\le (N - 2\ell gN) + \ell gN + \ell g\ell gN + \cdots$$

On remarque que la contradiction disparaît dès qu'on se montre moins gourmand quant au nombre de '0' consécutifs, en l'occurrence, ℓgN. Lors de $N=10^9$ lancers consécutifs, on observera très probablement quelque part une trentaine de '0' (ou de '1') qui se suivent.

4) Complexité d'un système formel.

C'est en suivant la même démarche que l'on trouve que la complexité d'un théorème énoncé dans le cadre d'un système formel, ne peut dépasser celles combinées des axiomes de base de ce système, des règles de productions logiques et de l'archivage du chemin le plus court qui permet de déduire le théorème des axiomes. Selon une formule chère à Chaitin, dans un système formel, un kilo d'axiomes ne donnera jamais une tonne de théorèmes puisque les énoncés de ceux-ci sont largement compris dans les prémices. Nous reviendrons plus en détail sur cet argument dans l'exposé consacré à l'universalité au sens de Gödel.

L'inférence d'un modèle à partir des données.

La compression des données n'est pas qu'une préoccupation utilitaire visant à alléger le stockage et la transmission des données. C'est aussi l'affaire de tous les scientifiques à la recherche du meilleur modèle qui rend compte de leurs données expérimentales.

Souvenons-nous des premiers astronomes qui consignaient, jours après jours, dans de grands registres, les positions des astres dans le ciel. Des années d'observations représentaient une somme d'informations qui se serait rapidement avérée ingérable sans la découverte d'un fil conducteur révélant que cette somme est, en fait, largement redondante. C'est précisément un des objets de la science de découvrir des procédés algorithmiques courts capables de condenser l'information expérimentale dans un modèle cohérent. Inversement, celui-ci doit pouvoir reconstruire le fichier des valeurs expérimentales voire idéalement l'extrapoler.

Etudions le problème suivant et demandons-nous s'il est bien posé. Compléter la suite : 1,2,4,8,16, ? Au vu de ce qui suit, il semblerait que la réponse soit négative. On trouve, sans grande surprise, autant de solutions que l'on veut !

```
Table[2^(k-1), {k, 6}]
```

```
{1, 2, 4, 8, 16, 32}
```

```
Table[1/24 (24 - 18 k + 23 k^2 - 6 k^3 + k^4), {k, 6}]
```

```
{1, 2, 4, 8, 16, 31}
```

```
Table[384/(384 + 348 k - 544 k^2 + 237 k^3 - 44 k^4 + 3 k^5), {k, 6}]
```

```
{1, 2, 4, 8, 16, 1}
```

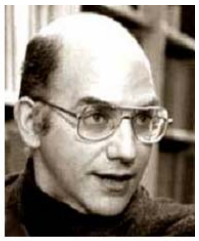
On pourrait être tenté de rejeter ce type de problème en prétextant qu'il est mal formulé mais ce serait ignorer que la démarche scientifique procède fréquemment de la sorte : devant un tableau de mesures, il serait faux de prétendre qu'il n'existe qu'un seul modèle qui les explique. C'est tout le problème de l'inférence du meilleur modèle qui rend compte des données. Plusieurs philosophies ont vu le jour à cet égard :

Le principe d'indifférence prescrit de n'écarter aucune hypothèse et d'attendre que les mesures ultérieures s'affinent et tranchent en défaveur d'un ensemble de modèles que l'on écartera pour ne garder que ceux qui restent.

Le principe du rasoir d'Occam prescrit d'opter pour l'explication la plus simple c'est-à-dire celle dont la complexité est la plus petite. Dans l'exemple présenté, ce serait la première qui correspond au programme le plus court.

La thèse SKC (Solomonoff-Kolmogorov-Chaitin), adopte un point de vue intermédiaire : elle n'écarte aucune hypothèse mais elle lui attribue une probabilité de vraisemblance, $2^{-\ell(p^*)}$, où $\ell(p^*)$ est la longueur du programme préfixe qui reproduit les données sans pertes.





Solomonoff **Kolmogorov** **Chaitin**

Thèse SKC.
En sciences, toutes les hypothèses compatibles avec les données ne sont pas équiprobables : elles ont une probabilité de vraisemblance a priori égale à $2^{-\ell(p^*)}$.

Probabilité universelle d'une suite.

Pour comprendre le contenu de la thèse SKC, il convient de se poser une question préalable. Quel sens peut-on donner un sens à l'interrogation suivante : quelle est la probabilité de tirer au hasard une suite binaire particulière?

Si on se restreignait aux suites de longueur, N, la réponse serait claire. Comme elles sont en nombre fini, soit 2^N, on pourrait les mettre toutes dans un chapeau et on aurait une chance sur 2^N d'en tirer une. La somme des probabilités vaudrait bien 1. Par contre, la question se complique si on autorise toutes les suites sans restriction de longueur. Il est clair qu'il n'est plus possible de les mettre toutes dans un chapeau afin d'en extraire une au hasard ! Il faut donc trouver autre chose et, en particulier, il faut abandonner l'idée que toutes les suites puissent être équiprobables puisque la somme des probabilités serait infinie.

C'est précisément ce que fait la thèse SKC lorsqu'elle associe à toute suite une densité de probabilité proportionnelle à (1/2) exposant la longueur de son encodage préfixe. La viabilité de cette procédure repose sur ce fait que tirer progressivement à pile ou face une suite de '0' et de '1' engendre tôt ou tard une suite finie qui fait partie une et une seule fois de l'ensemble préfixe canonique,

Λ_{pref} =
{{10},{110},{111},{01000},{01001},{01010},{01011},{011000},{011001},{011010},{011011},{011100},{011101},{011110},{011111},{0010000000},{0010000001},…}

La thèse SKC attribue donc une probabilité d'occurrence $2^{-l(N)}$ à toute suite comportant N bits. Par exemple, la suite, {0,0,0,1}, est encodée par, {0,0,1,0,0,0,0,0,0,1}, d'où une probabilité d'occurrence égale à 2^{-10}.

Il est essentiel de prendre l'encodage préfixe comme base de calcul. Poser la probabilité de la suite, {0,0,0,1}, proportionnelle à 2^{-4}, sous le prétexte qu'elle ne consomme que 4 bits ne fonctionne pas. Il existe, en effet, une suite vide, deux suites à 1 bit, 4 suites à deux bits, etc, et la somme des probabilités affectées à chacune divergerait :

$$1\times 2^{-0}+2\times 2^{-1}+4\times 2^{-2}+8\times 2^{-3}+\cdots=1+1+1+1+\cdots=\infty.$$

Cette méthode d'attribution des probabilités a priori peut paraître bien compliquée mais c'est la seule qui n'introduit aucun biais. On pourrait penser procéder plus simplement pour sélectionner aléatoirement une suite parmi l'ensemble, Λ,

Λ = {ε, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111, 0000, …}.

La procédure suivante semblerait équitable. On lancerait une pièce de monnaie en faisant la distinction suivante entre lancers impairs et pairs. Les lancers impairs serviraient uniquement à permettre une interruption de la procédure selon le code suivant : un tirage, "pile"='1', signifierait 'continue' et un tirage 'face'='0' signifierait "arrête". Les lancers pairs feraient quant à eux partie intégrante de la suite cherchée aussi longtemps qu'aucune instruction d'arrêt n'a pas interrompu la procédure. De cette manière, la suite vide, ε, aurait une chance sur deux de se produire, les 2 séquences à un bit auraient une chance sur quatre de se produire et plus généralement, les 2^N séquences à N bits auraient chacune une probabilité d'occurrence égale à 2^{-2N}. Au bilan on aurait que la somme des probabilités vaudrait 1.

Ce que la thèse SKC reproche à cette méthode c'est de ne pas être impartiale, elle favorise trop les suites courtes. De fait, elle correspond en gros à un schéma de délimitation des suites basé sur le doublement de leur longueur ce que nous savons excessif. L'encodage préfixe lui est impartial : toute suite originellement de longueur, N, est n'allongée que d'un facteur, *l(N)*, asymptotiquement égal à $\ell g^*(N)$ et cet allongement est précisément tout juste ce qu'il faut pour faire converger la somme des probabilités. De plus, la convergence se fait automatiquement vers 1, en vertu de l'égalité de Kraft, puisque l'ensemble, Λ_{pref}, est complet. Voici les probabilités universelles des suites les plus simples :

$$P(\varepsilon)=\frac{1}{2^2},P(0)=P(1)=\frac{1}{2^3},P(00)=P(01)=P(10)=P(11)=\frac{1}{2^5},$$
$$P(000)=P(001)=\cdots=\frac{1}{2^6},P(0000)=P(0001)=\cdots=\frac{1}{2^{10}},\cdots$$

On vérifie aisément que la somme de ces probabilités vaut 1 comme il se doit. On a en effet, en regroupant toutes les suites de longueurs, N :

$$\sum P=\sum_{N=0}^{\infty}2^N 2^{-\ell(N)}=\frac{1}{4}+\frac{1}{4}+\frac{1}{8}+\frac{1}{8}+\frac{1}{64}+\frac{1}{64}+\frac{1}{64}+\frac{1}{64}+\frac{1}{128}+\ldots=1$$

Toutefois, la convergence de cette suite est extrêmement lente du fait du comportement asymptotique de ses termes. La série,

$$\sum_N 2^{-(l(N)-N)}=\sum_N 2^{-\ell g^*(N)}=\sum_N\frac{1}{N(\ell gN)(\ell g\ell gN)(\ell g\ell g\ell gN)\cdots}$$

se situe, en effet, très exactement à la frontière entre la convergence et la divergence.

Le critère de convergence de l'intégrale associée permet de démontrer que les séries suivantes divergent toutes de plus en plus lentement :

$$\sum_n \frac{1}{n} \qquad \sum_n \frac{1}{n(\ell gn)} \qquad \sum_n \frac{1}{n(\ell gn)(\ell g\ell gn)} \qquad \sum_n \frac{1}{n(\ell gn)(\ell g\ell gn)(\ell g\ell g\ell gn)}$$

Par contre, les séries suivantes convergent de plus en plus lentement :

$$\sum_n \frac{1}{n^2} \qquad \sum_n \frac{1}{n(\ell gn)^2} \qquad \sum_n \frac{1}{n(\ell gn)(\ell g\ell gn)^2} \qquad \sum_n \frac{1}{n(\ell gn)(\ell g\ell gn)(\ell g\ell g\ell gn)^2}$$

On peut d'ailleurs se faire une idée de la lenteur de la convergence en observant comment la série tend effectivement vers $1 = 0.111\cdots_{bin}$. On calcule les premiers termes de cette série en regroupant comme il a été dit les contributions provenant des suites non préfixes de longueurs égales.

Il existe 2^k programmes de longueurs, k (=0,1,2, …), dont l'encodage préfixe consomme respectivement, {2,3,5,6,10,11,12,13,15,16,17,18,19,20,21,22,…} bits. Le programme vide, ε, de longueur préfixe, 2, contribue pour, $1/2^2 = 0.01_{bin}$, à la valeur de S. Les deux programmes de longueur, 1, de longueur préfixe, 3, contribuent ensemble pour, $2 \times 1/8 = 0.01$, soit au total partiel, $1/4 + 1/4 = 1/2 = 0.1_{bin}$. A ce stade, $3 = 2^2 - 1$, programmes ont été comptabilisés et le premier chiffre 1 de S après la virgule est stabilisé à sa valeur définitive. On montre de la même manière que pour stabiliser le deuxième 1 après la virgule, il faut comptabiliser les, $15 = 2^{2^2} - 1$, premiers programmes puis qu'il en faut, $2^{2^{2^2}} - 1 = 65535$, pour stabiliser le troisième, etc. On réalise la lenteur de la convergence.

Cette lenteur est essentielle dans la philosophie SKC car c'est elle qui garantit que l'on ne triche pas dans l'inventaire des modèles possibles. Procéder autrement reviendrait à privilégier indûment les programmes courts c'est-à-dire reviendrait à coller de trop près et sans justification à la théorie d'Occam.

En résumé, pour en revenir à la question posée au début de ce paragraphe, le tirage aléatoire d'une suite quelconque se conçoit comme suit. On lance une pièce de monnaie un nombre suffisant de fois et on compte combien de fois on commence par tirer "face" = "0". Si ce nombre vaut zéro, c'est qu'on est dans l'un des trois cas suivants, faciles à distinguer au vu des tirages suivants,

{1,0}⇒ ε {1,1,0}⇒ {0} {1,1,1}⇒{1}

Si ce nombre vaut 1, c'est qu'on est dans l'un des dix cas suivants, faciles à distinguer,

{0,1,0,0,0}⇒{0,0} {0,1,0,0,1}⇒{0,1} {0,1,0,1,0}⇒{1,0} {0,1,0,1,1}⇒{1,1}

{0,1,1,0,0,0}⇒{0,0,0} {0,1,1,0,0,1}⇒{0,0,1} {0,1,1,0,1,0}⇒{0,1,0}
{0,1,1,0,1,1}⇒{0,1,1} {0,1,1,1,0,0}⇒{1,0,0} {0,1,1,1,0,1}⇒{1,0,1}
{0,1,1,1,1,0}⇒{1,1,0} {0,1,1,1,1,1}⇒{1,1,1}

et ainsi de suite. On arrête les lancers lorsque la suite de bits obtenue coïncide avec un code de l'ensemble préfixe, Λ_{pref}, ce qui ne peut manquer de se produire puisque c'est la définition d'un ensemble préfixe de ne pas tolérer qu'un code soit le préfixe d'un autre.

L'inférence du modèle dans le cadre du modèle SKC.

On appelle modèle d'une suite, s, de données tout programme préfixe, p*, qui préinscrit sur la bande d'une MTU provoque l'arrêt de celle-ci en imprimant cette suite s. le programme qui alimente la MTU doit être préfixe afin qu'on sache où il se termine sur la bande. De plus, il doit incorporer les données éventuelles, ce qui ne pose jamais de problème de principe. Il peut exister plusieurs voire une infinité de programmes distincts qui effectuent cette tâche particulière et tous sont recevables au titre de modèle des données, s. Tous les programmes n'ont cependant pas le même degré de vraisemblance et c'est précisément l'objet de la thèse SKC que de quantifier cette propension à la vraisemblance, P(s).

La quantité, P(s), peut être vue comme la probabilité universelle « a priori » d'occurrence de la suite s. Les programmes courts contribuent majoritairement à P(s) d'où il en résulte qu'une suite qui n'est engendrée par aucun programme court n'a que très peu de chances d'apparaître un jour à un expérimentateur.

Si le programme le plus court fournissait, à lui seul, le terme dominant de P(s), on écrirait, $P(s) \approx 2^{-K(s)}$. En réalité et afin de tenir compte de programmes concurrents, on peut démontrer (c'est loin d'être trivial !) que l'on doit écrire qu'il existe une constante, c, indépendante de s, telle que l'on a :

$$2^{-K(s)} \le P(s) \le c2^{-K(s)} \qquad \Rightarrow \qquad K(s) - c' \le \ell g \frac{1}{P(s)} \le K(s)$$

Il y a donc un rapport direct entre la complexité d'une suite et sa probabilité universelle d'occurrence. Tout se passe comme si la nature favorisait l'occurrence des suites de faibles complexités. Inversement les programmes courts garantissent des suites non aléatoires qui structurent nos mesures. Dans un sens il est heureux qu'on parvienne à cette conclusion sans laquelle la quête scientifique serait sans espoir éternellement confrontée à des données aléatoires impossibles à exploiter.

Il est amusant de comparer la probabilité pour qu'un bonobo imprime une version exploitable des 10^6 bits d'une œuvre de Shakespeare selon qu'il tapote le clavier d'une machine à écrire ou celui d'un ordinateur universel : dans le premier cas, la probabilité est de l'ordre de $2^{-N} = 2^{-1000000}$ tandis que dans le second cas, elle est de l'ordre de $2^{-K(N)} = 2^{-330000}$, si l'on adopte l'idée d'un taux de compression de la langue anglaise approximativement égal à 3 ! Il n'y a rien de mystérieux là-dessous : simplement, pour répondre à la question qui lui est posée, le bonobo n'a pas besoin de taper le texte immédiatement lisible, sa compression suffit dans le cadre du modèle reconnu optimal pour ce qui concerne la langue anglaise.

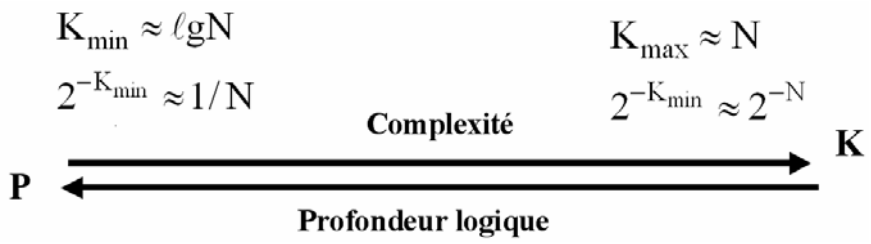
Notion de complexité calculatoire.

La complexité calculatoire d'un programme est une notion relativement informelle quantifiée par le « temps » que met le programme décompresseur à calculer les termes de la suite avant de les imprimer. Pour des raisons évidentes d'indépendance vis-à-vis d'une machine particulière, ce temps se mesure en nombre de transitions subies par la MTU plutôt qu'en secondes. La complexité calculatoire du programme de compression maximale d'une suite s'appelle la profondeur logique de cette suite.

Les suites aléatoires sont incompressibles et de complexité maximale : elles sont caractérisées par une profondeur logique nulle puisque le programme minimum qui les décrit

se note « Print[s] », qui se contente de mémoriser la suite et de l'imprimer telle quelle sans effectuer le moindre calcul.

A l'opposé, les suites de faible complexité sont habituellement compressées en mettant en jeu les ressources du calcul arithmétique. C'est, par exemple, ce qui se passe dans le cas des deux programmes élémentaires, « Do[Print['01'],{k,1,N/2}] » et « Print[N[1/3,10^N] », le premier recourant à un compteur de boucle et le deuxième effectuant une division binaire. Leur décompression implique le détricotage correspondant ce qui augmente leur profondeur logique. Complexité et profondeur logique évoluent généralement en sens contraires.



La thèse SKC enseigne que les suites peu compressibles donc de profondeur logique très faible sont peu probables au contraire des suites complexes qui possèdent une profondeur logique importante. On peut montrer, sur cette base, que la profondeur calculatoire moyenne d'un algorithme appliqué à une suite, s, ne s'écarte pas de sa valeur maximum !

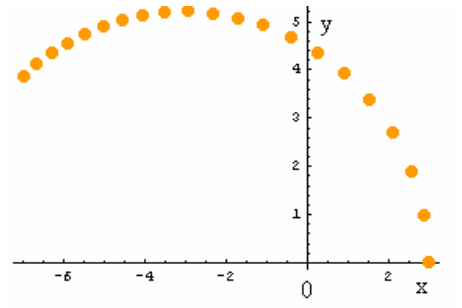
Cette conclusion est familière aux usagers des algorithmes de tri. Il est connu qu'un tri ordinaire appliqué à un fichier de longueur, n, requiert un nombre d'étapes de l'ordre de n^2. Il existe une version rapide (QuickSort) qui n'en n'exige généralement que $n\ell gn$ mais elle échoue toutefois lorsque le fichier est déjà presque trié, ce qui est hélas la majorité des cas rencontrés dans une pratique ordinaire, en conformité avec la thèse SKC. La parade est bien connue des informaticiens : ils commencent par brouiller le fichier avant de le trier, cette manœuvre ne coûtant que n étapes (Randomized QuickSort).

L'inférence du modèle : une lubie sans intérêt ni application ?

Si les problèmes d'inférence sont le pain quotidien des statisticiens, l'exemple qui suit montre qu'ils concernent également le physicien. Reprenons le tableau de « mesures » évoqué un peu mystérieusement au début de cet exposé : il présente les relevés des positions d'une planète autour de son soleil, dans le plan de la trajectoire.

```
{{3.00000, 0}, {2.89056, 0.987899}, {2.58063, 1.90851},
 {2.11504, 2.71611}, {1.54502, 3.39221}, {0.914311, 3.93810},
 {0.255055, 4.36503}, {-0.410933, 4.68749}, {-1.06946, 4.91983},
 {-1.71150, 5.07492}, {-2.33143, 5.16379}, {-2.92585, 5.19576},
 {-3.49274, 5.17860}, {-4.03101, 5.11886}, {-4.54017, 5.02204},
 {-5.02008, 4.89280}, {-5.47086, 4.73510}, {-5.89276, 4.55236},
 {-6.28612, 4.34754}, {-6.65132, 4.12321}, {-6.98876, 3.88166}}
```

La représentation graphique, dans le plan, (x, y), est bien telle qu'on l'imagine et il est bien connu que c'est Newton, le premier, qui a trouvé un programme court capable de restituer ces données sans pertes apparentes, aux erreurs de mesure près : ce programme fait appel à un ensemble d'équations différentielles.



$$\partial_t^2(x,y) = -2\cdot 10^{-8}\frac{(x,y)}{(x^2+y^2)^{3/2}}$$

$$(x_0 = 3; y_0 = 0; x'_0 = 0; y'_0 = 10^{-4}; m = 1)$$

Ce tour de force a tellement impressionné ses contemporains qu'il a conditionné l'évolution de la physique en termes de forces et de calcul différentiel. Il n'est pas question de dénier au modèle newtonien le mérite qu'il a d'expliquer ces données et beaucoup d'autres mais simplement de faire remarquer que l'attitude est dangereuse de croire qu'un modèle qui marche « si bien du premier coup » ne peut qu'être « vrai ». On sait ce qu'il est advenu de la notion de vérité en mathématique et les mêmes réserves s'appliquent ici sur un terrain différent.

Nous pensons, mais cela n'engage que nous, que la probabilité pour que la physique se soit développée comme la nôtre, sur une planète lointaine, est proche de zéro. Non que les données du problème soient fondamentalement différentes ici et là-bas, personne ne souscrirait à cette idée, mais parce que d'autres modèles existent qui asymptotiquement se valent pour expliquer ces données. Par asymptotiquement, on veut dire que divers modèles existent dont les prédictions se recouvrent tant qu'on ne les pousse pas dans leurs derniers retranchements. Mais précisément c'est quand on pousse aux extrêmes la théorie de Newton ou d'ailleurs n'importe quelle autre théorie que les limites de validité apparaissent.

Voici un modèle discret qui restitue tout aussi fidèlement les données de base. Par fidèlement, nous voulons dire que les valeurs calculées ne diffèrent pas des mesures entachées de l'erreur expérimentale. Les valeurs prédites par le modèle discret ne diffèrent d'ailleurs des valeurs du modèle Newtonien que vers la 7ème décimale : ils sont donc bien asymptotiquement équivalents.

$$\vec{p}_i = m\sqrt{\frac{r_{i+1}}{2r_i}}\frac{\hat{r}_{i+1}+\hat{r}_i}{\sqrt{1+\hat{r}_{i+1}\cdot\hat{r}_i}} - \hat{r}_i\sqrt{m^2 - 2m\frac{\lambda}{r_i} + p_i^2}$$

$$\vec{p}_{i+1} = -m\sqrt{\frac{r_i}{2r_{i+1}}}\frac{\hat{r}_{i+1}+\hat{r}_i}{\sqrt{1+\hat{r}_{i+1}\cdot\hat{r}_i}} + \hat{r}_{i+1}\sqrt{m^2 - 2m\frac{\lambda}{r_i} + p_i^2}$$

$$t_{i+1} - t_i = r_{i+1} + r_i$$

$$(x_0 = 3; y_0 = 0; \delta_0 = -2\cdot 10^{-8}/3; p_{x,0} = 0; p_{y,0} = 10^{-4})(\lambda = 2\cdot 10^{-8}; m = 1)$$

Aucun argument expérimental ne permet de trancher entre les deux modèles. Peut-être la thèse SKC le pourrait-elle ? Sur le papier le système des équations différentielles paraît plus concis que le système discret mais c'est probablement une illusion d'optique : imaginons que nous voulions envoyer le fichier complet des mesures à un correspondant lointain, il ne suffit pas de lui faire parvenir le système différentiel, il faut, en réalité, lui envoyer un programme entièrement prêt à l'emploi, y compris les routines qui calculent la solution par une méthode de Runge-Kutta par exemple. En comparaison le système discret semble très économique !

Imaginons, à présent, que ce système ait vu le jour avant 1687, date de parution du traité de Newton : on n'aurait pas manqué de trouver miraculeux d'expliquer ainsi des données qui semblaient résister à toute analyse antérieure. L'évolution de la physique aurait-elle été différente, privilégiant la notion d'écart de masse (défaut de masse dans le cas attractif et excès de masse dans le cas répulsif) au lieu de la notion de force ? Personne ne peut répondre à une telle question et le fait est que personne ne se la pose sérieusement parce que cela n'aurait tout simplement plus de sens de se la poser. La science a clairement opté pour un développement qui ne résout pas tous les problèmes mais qui donne satisfaction. Cependant, le piège est à éviter d'écarter dédaigneusement toute alternative sous le prétexte que nous « savons bien », depuis Newton, que les choses ne se passent pas autrement : on ne juge pas une théorie à la lumière d'une autre mais plutôt à la lumière des faits expérimentaux

Pour la petite histoire, on montre comment on peut habiller ce modèle discret en lui écrivant un scénario plausible. Le soleil est fixé au centre des axes de coordonnées (on pourrait le faire bouger comme il sied au problème à deux corps mais c'est inutile à ce stade élémentaire de l'exposé où l'on considère que le soleil est infiniment lourd). La planète, de masse propre, m, est située initialement en (x_0, y_0) et sa quantité de mouvement vaut $(p_{x,0}, p_{y,0})$. Le principe du modèle exploré repose sur quelques hypothèses simples :

- la masse de la planète, en t = 0, diffère la masse propre d'une quantité égale à δ_0/c^2 où δ_0 (<0) représente un « défaut de masse » dû à l'interaction,

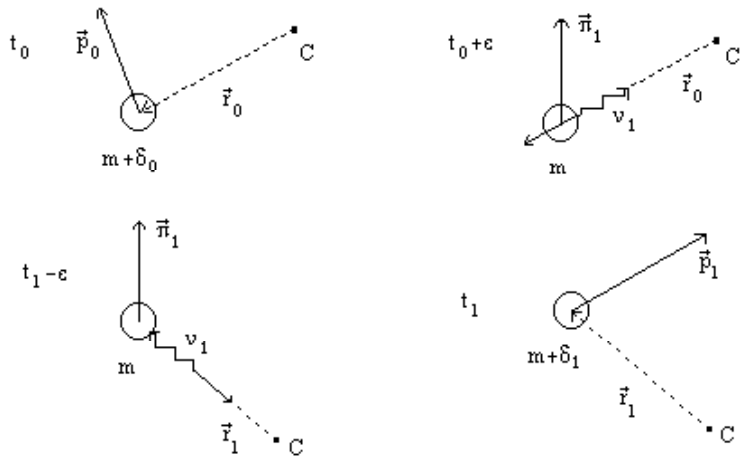
- l'énergie initiale de la planète est donnée par l'expression, $\sqrt{(m+\delta_0)^2 c^4 + c^2 p_0^2}$,

- la planète se débarrasse, en t = 0, de ce défaut de masse en émettant, dans la direction du centre, un « boson intermédiaire » d'énergie négative, $h\nu_1$, et de vitesse, c, tout en respectant les lois de conservation de l'énergie et de la quantité de mouvement,

- le soleil, supposé infiniment lourd dans ce cas simple, réfléchit le boson et le renvoie vers la position de la planète où celle-ci se trouvera prête à le réabsorber. Le cycle recommence sans fin.

La figure détaille les premières étapes de l'évolution d'un tel système. A l'instant, $t = t_0$, la particule possède un défaut de masse δ_0. Sa quantité de mouvement est, $\vec{p}_0$, et sa position, par rapport au centre attracteur, C, est $\vec{r}_0$. A l'instant, $t_0+\varepsilon$, la particule change de quantité de mouvement, $\vec{p}_0 \rightarrow \vec{\pi}_1$, en émettant un boson d'énergie négative, $h\nu_1$, dans la direction du centre. Celui-ci le renvoie aussitôt afin qu'il soit réabsorbé à l'instant ultérieur, t_1, dans la nouvelle position, $\vec{r}_1$, de la planète. La particule possède alors un nouveau défaut de masse, δ_1, une nouvelle quantité de mouvement, $\vec{p}_1$, et le processus recommence sans fin.

Ce scénario n'a strictement rien d'obligatoire : il n'est qu'une interprétation sémantique possible du modèle syntaxique que nous livrons ci-après. Nous reviendrons sur cette distinction essentielle dans l'exposé consacré à l'étude Gödélienne des systèmes formels.



Les équations de conservation et d'évolution revêtent la forme suivante (i=0,1,2,...) :

$$\begin{cases} \left[(m+\delta_i)^2 + p_i^2\right]^{1/2} = \left[m^2 + \pi_{i+1}^2\right]^{1/2} + \nu_{i+1} \\ \vec{p}_i = \vec{\pi}_{i+1} - \nu_{i+1}\hat{r}_i \end{cases}$$

$$\begin{cases} \left[m^2 + \pi_{i+1}^2\right]^{1/2} + \nu_{i+1} = \left[(m+\delta_{i+1})^2 + p_{i+1}^2\right]^{1/2} \\ \vec{p}_{i+1} = \vec{\pi}_{i+1} + \nu_{i+1}\hat{r}_{i+1} \end{cases}$$

$$\begin{cases} \vec{r}_{i+1} = \vec{r}_i + \vec{\pi}_{i+1}(t_{i+1} - t_i)/\left[(m^2 + \pi_{i+1}^2)\right]^{1/2} \\ r_i + r_{i+1} = t_{i+1} - t_i \end{cases}$$

où â est unitaire et où on a posé, c=1.

Un peu d'algèbre permet d'isoler les seules variables pertinentes, de position et de quantité de mouvement et d'écrire le système récurrent annoncé (on y a posé c=1). Ce n'est pas l'objet de cet exposé d'étudier plus en détails les implications d'un tel modèle qui renonce à la notion de force et supprime l'interaction instantanée à distance.