CODES CORRECTEURS

MICHEL RIGO

Ce texte sert de compagnon succinct à la présentation "PowerPoint" portant sur les codes correcteurs (Printemps des Sciences 2009).

Qu'on le veuille ou non, nous vivons dans un monde où les informations et les moyens de communication sont omniprésents. Les exemples où se mêlent informatique et télécommunications sont nombreux. Que ce soit l'Internet, la musique achetée en ligne ou gravée sur CD[1], les baladeurs de type `iPod` et autres appareils portables, les supports mémoire de tout type, la photographie numérique, la présence d'ordinateurs de bord sophistiqués dans nos voitures, *etc.* Sans code correcteur, le CD (implémentant un code correcteur de type Reed-Solomon) et de nombreux autres produits n'auraient probablement jamais connu le succès !

Sans nécessiter de pré-requis particulier, ce texte et l'exposé qui s'y rapporte développent quelques exemples montrant que des mathématiques parfois abstraites et ne présentant pas d'intérêt pratique immédiat — comme certains résultats d'algèbre linéaire ou générale — ont de réelles applications dans la vie de tout un chacun. Voir aussi [5, 6] pour d'autres exemples ayant le même leitmotiv.

Notons que les développements rencontrés ici peuvent fournir à l'enseignant une illustration de la notion de *vecteur* et du "*calcul en composantes*" (certains codes correcteurs sont en fait des sous-espaces vectoriels) ainsi que quelques applications de l'*analyse combinatoire* et du calcul élémentaire des *probabilités*. Ces derniers aspects ne seront abordés, dans l'exposé oral, que pour les élèves ayant vu ces notions en classe.

Modestement, cet exposé donne un embryon de réponse à la question si souvent posée : "*Les Maths ça sert à quoi ?* ". Les applications se nourrissent de la recherche fondamentale et réciproquement. Cependant la recherche peut et doit vivre pour elle-même, sans avoir en ligne de mire une quelconque application. Certains codes correcteurs évolués reposent par exemple sur des résultats non triviaux de géométrie algébrique développés de manière indépendante [3]. Ces codes n'auraient jamais vu le jour sans des développements en mathématiques pures. Quelques chiffres pour conclure cette introduction : près de 2400 articles scientifiques en théorie des codes[2] parus entre 1990 et 2000 et déjà plus de 2500 depuis 2000.

1. La base 2

Pour représenter et manipuler les nombres, nous utilisons depuis notre plus tendre enfance le système décimal. Un nombre entier est représenté par une suite $c_k \cdots c_1 c_0$ de chiffres compris entre 0 et 9 (ne commençant pas par zéro) et la convention veut que le nombre ainsi représenté soit

$$c_k \times 10^k + \cdots + c_1 \times 10 + c_0 \times \underbrace{10^0}_{1}.$$

On parle de *numération de position* car chaque chiffre de la représentation est multiplié par une puissance convenable de dix en fonction de la position qu'il occupe

[1] Bien que le bon usage voudrait qu'on écrivît cédé, je m'autoriserai le CD.

[2] AMS Classification `94B` : Theory of error-correcting codes and error-detecting codes.

au sein de cette représentation. On trouve ainsi de droite à gauche, le chiffre des unités, puis celui des dizaines, des centaines, des milliers, *etc.* Par exemple, $2762 = 2 \times 10^3 + 7 \times 10^2 + 6 \times 10 + 2$.

Que se passe-t-il si on remplace la base de numération, en l'occurrence 10, par un autre entier, disons $b \geq 2$? Dans ce cas, les chiffres utilisés pour écrire un nombre en base b sont pris dans $\{0, \ldots, b-1\}$ et chaque chiffre est multiplié par la puissance de b correspondant à sa position dans l'écriture considérée. Ainsi, avec des chiffres dans $\{0, \ldots, b-1\}$, $c_k \cdots c_1 c_0$ représente

$$[c_k \cdots c_1 c_0]_b = c_k \times b^k + \cdots + c_1 \times b + c_0.$$

En particulier, en base 2, on obtient le système binaire et les premiers entiers strictement positifs se représentent dans l'ordre par

$$1, 10, 11, 100, 101, 110, 111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111, 10000, \ldots.$$

La notation $[\cdot]_b$ permet de rappeler la base utilisée. Par exemple, on a $[1001]_2 = 9$ mais $[1001]_3 = 28$. Calculer dans une autre base que la base 10 requiert un peu d'entraînement, mais les règles de calcul de l'école primaire (comme les "calculs écrits") fonctionnent toujours très bien. Pour une addition faite par écrit en base 2, lorsqu'on se trouve face à deux 1 alignés dans une même colonne, pour effectuer ce "$1 + 1$", on aura la règle orale : "*j'écris* 0 *et je reporte* 1" car $1 + 1 = [10]_2$.

$$\begin{array}{cccccccc} & & {\scriptstyle 1} & & & & & \\ & 1 & 0 & 1 & 0 & 0 & 1 \\ + & & & 1 & 0 & 1 & 0 \\ \hline & 1 & 1 & 0 & 0 & 1 & 1 \end{array}$$

Voici quelques petites questions sur le système binaire :

- Que vaut le nombre $[111111]_2 + 1$? Quelle en est la représentation en base 2 correspondante ? Pour l'obtenir, réalisez le calcul écrit suivant et observez les reports.

$$\begin{array}{cccccccc} & 1 & 1 & 1 & 1 & 1 & 1 \\ + & & & & & & 1 \\ \hline & & & & & & ? \end{array}$$

- Quels entiers ont une écriture en base 2 de la forme 1, 10, 100, ..., $1\overbrace{0\cdots0}^{i}$, avec $i \geq 0$?
- Même question pour ceux de la forme 1, 11, $\overbrace{1\cdots1}^{i}$?
- Que se passe-t-il sur l'écriture d'un entier n en base 2, lorsqu'on multiplie celui-ci par 2 ou encore, par 4 ? Même question lorsqu'on remplace n par $2n + 1$?
- Comment décider si un nombre est pair ou impair, lorsqu'on connaît sa représentation en base 2 ?

2. Coder des données

Avec des règles bien précises, des informations de types variés peuvent être représentées par une suite de `0` et de `1`. La manière dont est représenté l'objet d'intérêt ne relève alors que de *conventions* connues de tous les intervenants ayant à manipuler l'information. Par exemple, on peut convenir de coder les lettres de l'alphabet par des successions de cinq éléments pris dans $\{0, 1\}$ comme suit :

a=`00001`, b=`00010`, c=`00011`, d=`00100`, e=`00101`, ..., z=10110.

Cela correspond simplement à représenter chaque lettre (minuscule) par sa position en base 2 au sein de l'alphabet. De plus, on ajoute, si nécessaire, des zéros de tête pour que chaque mot soit de longueur 5. Avec de telles conventions, la suite

`00010 01111 01110 01010 01111 10101 10011`

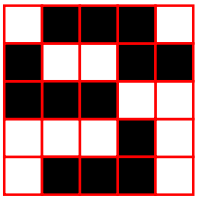
représente le mot "bonjour". Les espaces ne sont présents ici que pour faciliter la lecture. Le code transmis est `00010011110111001010011111010110011`. Que se passerait-il sans la convention d'ajouter des zéros de tête ? Le code

`10111111101010111111010110011`

est-il encore intelligible ? Comment savoir où s'arrête le code d'une lettre et où commence le code de la lettre suivante ?

Sans nous attarder sur de lourds détails techniques, nous voulons convaincre le lecteur qu'il est aisé de coder des textes, du son ou des images. Plusieurs standards utilisés internationalement permettent de coder des **textes**. A la manière de l'exemple décrit ci-dessus, les standards ASCII ou unicode attribuent à chaque caractère (majuscule, minuscule, signe de ponctuation) un code, *i.e.*, un entier. Un texte est dès lors formé d'une suite de tels codes représentés en base 2. Un fichier `.doc` ou `.rtf` contient non seulement un texte mais aussi de nombreuses informations de mise en page (police de caractère, couleur, taille, *etc*) codées de manière appropriée.

Une **image** dont les dimensions sont connues, n'est en fin de compte qu'une succession de pixels (ou points lumineux) alignés. Pour une image en noir et blanc, chaque pixel est soit éteint soit allumé ce qui, par convention, peut une fois encore être codé par deux nombres : `0` et `1`.



```
1 0 0 0 1
0 1 1 0 0
0 0 0 1 1
1 1 1 0 1
1 0 0 0 1
```

Pour une image en dégradés de gris (par exemple, 16 dégradés allant du noir au blanc), on peut convenir de coder chaque teinte par les nombres $0 = 0/15$, $1/15$, $2/15$,..., $14/15$, $1 = 15/15$, ces fractions correspondant au niveau d'intensité lumineuse du pixel considéré. Les numérateurs compris entre 0 et 15 peuvent être codés en base 2 par 4 chiffres. Avec $[0000]_2 = 0$ représentant le noir et $[1111]_2 = 15$ représentant le blanc. Un peu comme dans l'exemple précédent où chaque lettre était codée par 5 chiffres, chaque nombre est codé par un nombre constant de chiffres.

Les systèmes plus évolués utilisent jusqu'à $2^8 = 256$ teintes de gris. On utilise alors des intensités variant par pas de $1/255$. Celles-ci vont du noir codé par huit zéros, au blanc représenté par $[1111111]_2 = 255$. Pour obtenir des images en couleurs, il faut savoir que chaque couleur peut être obtenue par addition des couleurs primaires : rouge, vert et bleu. Imaginer une feuille blanche éclairée par un spot bleu, la feuille paraît bleu. Eclairée par deux spots, un bleu et un vert, la page paraît turquoise. Enfin éclairée par trois spots, un rouge, un vert et un bleu, les couleurs s'additionnent pour retrouver une page blanche. Ainsi, en superposant trois images

distinctes (une pour chaque couleur primaire), on obtient l'image en couleurs désirée. Chacune de ces trois images peut être codée comme une image formée de dégradés d'une teinte de base semblable aux images en dégradés de gris. Avec ce modèle, il est aisé de définir jusqu'à $2^{3\times 8} = 16.777.216$ couleurs.

Un **son** n'est autre qu'une onde produite par la vibration mécanique d'un support (par exemple, les cordes vocales) et propagée grâce à l'élasticité du milieu (en l'occurrence l'air) sous forme d'onde longitudinale. L'air étant un milieu compressible, le son se propage sous forme de variations de pression. Ces vibrations sont captées par la membrane d'un micro qui les transforme en impulsions électriques. Le signal électrique est alors *échantillonné*[3]. On transforme un signal continu (penser au graphe d'une fonction continue) en un signal discret (penser à une approximation polygonale du graphe en question), en enregistrant des valeurs prises par ce signal à intervalles de temps réguliers. Ces valeurs qui ne sont que des nombres, sont alors codées de façon *ad hoc* par une suite de 0 et de 1.

Signalons encore que l'ADN renferme les informations nécessaires au développement et au fonctionnement d'un organisme. Il est aussi le support de l'hérédité. Ainsi, nous portons également le codage de nombreuses informations. Les zéros et les uns sont simplement remplacés par des paires Adenine-Thymine, Guanine-Cytosine.

3. Pourquoi avoir des codes *correcteurs*

On pourrait penser vivre dans un monde parfait dans lequel aucune erreur n'est jamais commise. Il n'en est hélas rien. Pour de nombreuses raisons, des erreurs peuvent se produire. Reprenons l'exemple où a=00001, ..., z=10110. Ainsi, si le message

00010 01111 01110 01010 01111 10101 10011

a été mal retranscrit (par exemple, il a été dicté et la personne recevant les informations ne les a pas bien comprises), on peut avoir gardé uniquement trace d'un message comme

00011 01111 01010 01010 01111 10101 10100

qui, avec nos conventions, sera décodé par "cojjous" devenu incompréhensible ! Par exemple, une communication peut être altérée par des interférences, un orage, la proximité d'une ligne à haute tension, un CD ou un DVD peut présenter des grattes mêmes microscopiques, des empreintes de doigts ou des poussières en surface. On peut aussi, sur un support magnétique, imaginer des erreurs de lecture ou d'écriture (dans un disque dur, les têtes de lecture et d'écriture se déplacent à la surface d'une fine couche magnétique utilisée pour sauvegarder les informations).

Pour pallier à ces désagréments, les chercheurs et les ingénieurs ont développé de nombreux codes permettant de détecter ou même de corriger certaines erreurs. Dans une conversation normale entre deux personnes, lorsqu'un des interlocuteurs ne comprend pas un message, il demande en général qu'on lui répète une phrase ou un mot. Ainsi, pour être compris par un plus grand nombre, un discours est souvent plus long que nécessaire. Les idées principales sont reprises et répétées. On ajoute donc une certaine redondance par rapport au message initial.

[3]Voir l'un des théorèmes de C. E. Shannon (1916–2001), père fondateur de la théorie de l'information [7].

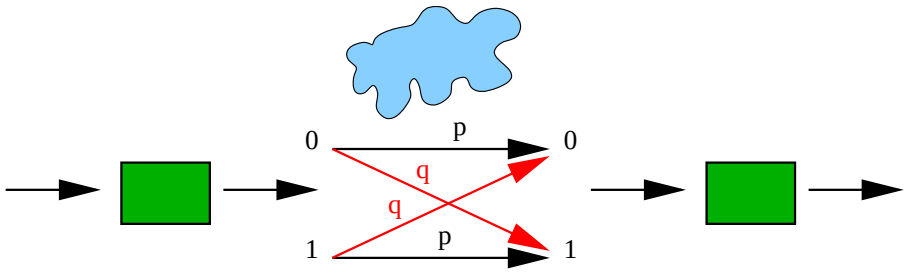
3.1. **Doubler chaque chiffre.** On pourrait décider dans un premier temps de répéter chaque chiffre. Ainsi, à la place des premiers éléments de notre exemple : `0001001111`, on pourrait stocker ou transmettre le message

`00 00 00 11 00 00 11 11 11 11`.

Si une erreur se produit et que l'on reçoit le message ci-dessous

`00 00 00 11 01 00 11 11 11 11`,

on peut facilement voir qu'une erreur s'est produite. En effet, si on découpe le message en blocs consécutifs de longueur deux, un des blocs n'est pas de la forme `00` ou `11`. Par contre, le bloc `01` peut provenir, avec la même probabilité, aussi bien d'un bloc `00` que d'un bloc `11`. Il n'y a donc pas moyen de corriger l'erreur qui a été détectée. Si cela est possible, on pourra demander de réexpédier le message une seconde fois (cela n'est pas toujours possible, pensez à une gratte sur un CD définitivement présente ou aux images envoyées par une sonde spatiale passant une seule fois à proximité d'une comète). Pire, si les deux chiffres d'un bloc ont été mal transmis, alors on recevra par exemple `11` à la place du bloc `00` sans détecter la présence d'une erreur. Pour pouvoir estimer efficacement le pourcentage d'erreurs détectées par un code donné, nous sommes amenés à formaliser notre modèle de communication (canal symétrique de communication) représenté schématiquement ci-dessous. Ce modèle convient pour des erreurs survenant de manière aléatoire. Cependant, dans certains cas pratiques, les erreurs arrivent plutôt par paquets.



Soit p la probabilité qu'un chiffre `0` ou `1` transmis sur le canal bruité soit correctement transmis. Posons $q = 1 - p$. Ainsi, on a

$$\mathbb{P}(\texttt{0 reçu}|\texttt{0 transmis}) = \mathbb{P}(\texttt{1 reçu}|\texttt{1 transmis}) = p,$$

$$\mathbb{P}(\texttt{0 reçu}|\texttt{1 transmis}) = \mathbb{P}(\texttt{1 reçu}|\texttt{0 transmis}) = 1 - p = q.$$

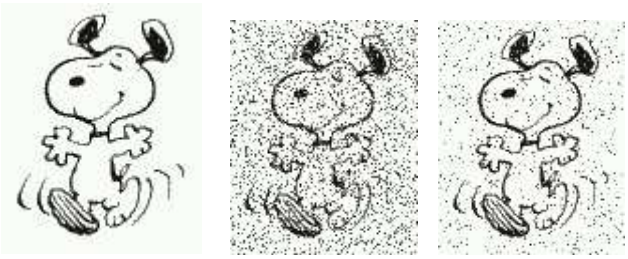
Avec ce modèle, imaginons disposer de l'information représentée par `01`. D'abord, celle-ci est codée en `0011` avec notre convention de doubler chaque chiffre. Ensuite, ce message de longueur 4 est envoyé sur le canal. La probabilité qu'un message de longueur 4 soit transmis sans erreur est de p^4 et celle qu'il soit transmis avec $i \in \{1, \ldots, 4\}$ erreurs est de $q^i p^{4-i}$.

3.2. **Tripler chaque chiffre.** Imaginons à présent répéter chaque chiffre non pas deux mais trois fois. Bien sûr, la taille du message augmente en conséquence, mais au point de vue des erreurs détectées et corrigées, cela change-t-il quelque chose ? Les chiffres `0` et `1` sont à présent codés respectivement par `000` et `111`. Lorsque le triplet `000` est transmis sur le canal de communication, on peut recevoir

triplet reçu	probabilité d'apparition	si $p = 0,9$
`000`	p^3	$0,729$
`001, 010, 100`	$3p^2q$	$0,243$
`011, 101, 110`	$3pq^2$	$0,027$
`111`	q^3	$0,001$

Lorsqu'on reçoit un triplet comme 000 ou 001, le plus vraisemblable (au sens des probabilités calculées ci-dessus[4]) est que le triplet réellement émis soit effectivement 000 provenant donc du chiffre 0. Ainsi, on convient de décoder un triplet à la majorité des chiffres présents. Le code ainsi défini permet de détecter une erreur et de la corriger. La probabilité qu'un chiffre significatif, codé par un triplet transmis sur le canal, soit au final bien décodé est donc $p^3 + 3p^2q = p^2(3 - 2p)$ ce qui pour $p = 0,9$ vaut $0,972$. Donc, si sur le canal on enregistre un taux de 10% d'erreurs, on est à même de transmettre des informations avec moins de 3% d'erreurs. Cela a un prix : on a triplé la taille du message initial.

Dans les images ci-dessous, on a utilisé une image codée par une suite de 0 et de 1. Quand elle est transmise sur un canal bruité ($p = 0,9$) sans aucun code correcteur, on obtient l'image centrale. L'image de droite donne le décodage après avoir transmis sur le canal chaque chiffre significatif trois fois.



Regardons à présent sur un exemple ce que permet ce code lorsque l'on transmet plus d'un chiffre significatif. Imaginons vouloir envoyer les quatre chiffres significatifs 0110. Ceux-ci sont d'abord remplacés par la suite de 12 chiffres

$$000\,111\,111\,000.$$

Ce message est transmis sur un canal bruité. Nous savons déjà qu'il est possible de détecter et corriger correctement une erreur. Imaginons maintenant avoir deux erreurs. Si celles-ci se trouvent dans des triplets distincts, elles seront bien corrigées

$$000\,11\underline{0}\,111\,0\underline{1}0 \longrightarrow 0110.$$

Par contre, si elles se présentent dans un même triplet, on procédera à un décodage erroné

$$000\,111\,111\,0\underline{11} \longrightarrow 0111.$$

La probabilité que les 4 chiffres codés par une suite de longueur 12 soient correctement décodés après transmission sur le canal bruité peut se calculer directement à partir du calcul précédent (pour un triplet). On obtient $(p^3+3p^2q)^4 = p^8(3-2p)^4$ exprimant que chacun des quatre triplets doit être correctement décodé. Pour $p = 0,9$, cette quantité vaut $0,892617$.

On peut aussi raisonner comme suit. Pour avoir un décodage correct, les erreurs éventuelles doivent apparaître dans des triplets différents. Ainsi, l'événement "*observer* $i \in \{0, 1, 2, 3, 4\}$ *erreur(s) se produisant des triplets distincts*" a une probabilité

$$3^i \ \mathrm{C}_4^i \ p^{12-i} \ q^i.$$

En effet, on ne doit pas sélectionner i positions arbitraires parmi 12. On sélectionne d'abord i triplets parmi les 4 disponibles et dans chacun de ces triplets, l'erreur qui y survient peut occuper l'une des 3 positions. Si plus de 4 erreurs surviennent, au moins deux apparaissent dans un même triplet qui sera dès lors mal décodé.

[4]Principe du maximum de vraisemblance.

Au total, la probabilité que les 4 chiffres codés par une suite de longueur 12 soit correctement décodés après transmission sur le canal bruité est

$$\sum_{i=0}^{4} 3^i \ \mathrm{C}_4^i \ p^{12-i} \ q^i.$$

On pourra vérifier que ce résultat est exactement $(p^3 + 3p^2q)^4$.

Enfin, on pourrait aussi s'interroger sur la proportion des erreurs que l'on sait corriger. Si $i = 2, 3, 4$, le nombre de cas possibles pour avoir exactement i erreurs sur les 12 chiffres transmis est de C_{12}^i et le nombre de cas que l'on sait effectivement corriger est de $3^i \ \mathrm{C}_4^i$. Pour $i = 2, 3, 4$, on obtient $3^i \ \mathrm{C}_4^i/\mathrm{C}_{12}^i$ qui vaut, pour $p = 0,9$, respectivement $0,818182$; $0,490909$; $0,163636$.

3.3. **Un code de Hamming.** Nous avons vu que le codage ci-dessus permettait de corriger une erreur mais avec un prix certain : les chiffres significatifs contenant véritablement l'information ne constituent qu'un tiers du message transmis. Pour limiter la quantité d'informations à stocker ou pour ne pas saturer le réseau de communication, il serait bien évidemment souhaitable de transmettre un message moins long tout en conservant des propriétés permettant la correction d'erreurs éventuelles. Idéalement, on voudrait pouvoir corriger un maximum d'erreurs, en limitant les redondances, *i.e.*, en n'allongeant pas le message original contenant l'information plus que nécessaire. C'est là l'un des challenges de la théorie des codes correcteurs. On peut aussi vouloir obtenir des codes permettant de prendre en compte des situations particulières (par exemple, si les erreurs n'apparaissent pas de manière aléatoire, mais plutôt par paquets).

Le code que nous allons présenter maintenant est un code *linéaire*. On peut faire le parallèle avec les vecteurs introduits en quatrième année et le calcul en composantes dans une base fixée. Ce code permet d'avoir à sa disposition une application autre que le calcul de forces en physique. Dans $\mathbb{Z}_2 = \{0, 1\}$, on a $1 + 1 = 0$ (autrement dit, on compte modulo 2, *cf.* par exemple [5]). On considère l'ensemble $(\mathbb{Z}_2)^7$ des 7-uples d'éléments de $\mathbb{Z}_2$ et on y définit une addition qui s'effectue composante à composante (à la manière de $\mathbb{R}^2$ ou $\mathbb{R}^7$). Par exemple, on a

$$(1, 0, 0, 0, 1, 1, 1) + (0, 1, 1, 0, 0, 1, 0) = (1, 1, 1, 0, 1, 0, 1).$$

En fait, $(\mathbb{Z}_2)^7$ possède une structure d'espace vectoriel (les scalaires sont ici les éléments de $\mathbb{Z}_2$) et on va en définir un sous-espace vectoriel (sous-ensemble non vide contenant les combinaisons linéaires de ses éléments).

Pour définir le code de Hamming $H[7, 4]$, on considère les 4 "vecteurs" (on peut effectivement parler de vecteurs puisqu'il s'agit d'éléments d'un espace vectoriel) suivants

$$\begin{array}{rcl} a & = & (1, 0, 0, 0, 0, 1, 1) \\ b & = & (0, 1, 0, 0, 1, 0, 1) \\ c & = & (0, 0, 1, 0, 1, 1, 0) \\ d & = & (0, 0, 0, 1, 1, 1, 1). \end{array}$$

L'enveloppe linéaire de ces 4 éléments est un sous-espace vectoriel qui contient toutes les combinaisons possibles de ceux-ci :

$$\begin{aligned}
a &= (1,0,0,0,0,1,1)\\
b &= (0,1,0,0,1,0,1)\\
c &= (0,0,1,0,1,1,0)\\
d &= (0,0,0,1,1,1,1)\\
a+b &= (1,1,0,0,1,1,0)\\
a+c &= (1,0,1,0,1,0,1)\\
a+d &= (1,0,0,1,1,0,0)\\
b+c &= (0,1,1,0,0,1,1)\\
b+d &= (0,1,0,1,0,1,0)\\
c+d &= (0,0,1,1,0,0,1)\\
a+b+c &= (1,1,1,0,0,0,0)\\
a+b+d &= (1,1,0,1,0,0,1)\\
a+c+d &= (1,0,1,1,0,1,0)\\
b+c+d &= (0,1,1,1,1,0,0)\\
a+b+c+d &= (1,1,1,1,1,1,1)\\
&\quad (0,0,0,0,0,0,0)
\end{aligned}$$

On remarque que pour les 16 vecteurs ci-dessus, deux quelconques d'entre eux débutent toujours avec des 4-uples distincts. On peut donc les utiliser pour coder toute suite de `0` et de `1` de longueur 4 en la remplaçant par le 7-uple ayant le même début. Par exemple,

$$\texttt{1100} \longrightarrow \texttt{1100110} \quad \text{et} \quad \texttt{0110} \longrightarrow \texttt{0110011}.$$

Les justifications ne seront pas abordées dans l'exposé mais ce code permet de détecter et de corriger une erreur apparaissant en une position quelconque du 7-uple. De plus, le décodage est aisé. On l'obtient en calculant 3 produits scalaires (somme des produits composante à composante, le tout calculé dans $\mathbb{Z}_2$) entre le 7-uple reçu et les trois vecteurs

$$x = (0,0,0,1,1,1,1), \quad y = (0,1,1,0,0,1,1) \quad \text{et} \quad z = (1,0,1,0,1,0,1).$$

On procède comme suit. Imaginons que le message à transmettre soit `0110011`, mais que le message reçu soit $\texttt{011}\underline{\texttt{1}}\texttt{011} = (0,1,1,1,0,1,1) = s$. On calcule dans $\mathbb{Z}_2$

$$\begin{aligned}
\langle s,x\rangle &= 0.0+1.0+1.0+1.1+0.1+1.1+1.1 = 1\\
\langle s,y\rangle &= 0.0+1.1+1.1+1.0+0.0+1.1+1.1 = 0\\
\langle s,z\rangle &= 0.1+1.0+1.1+1.0+0.1+1.0+1.1 = 0.
\end{aligned}$$

Ensuite, $[100]_2$ étant égal à 4, cela signifie que le 4ième chiffre transmis est erroné. La probabilité que les 4 chiffres codés par une suite de longueur 7 soit correctement décodés après transmission sur le canal bruité est $p^7 + 7p^6q$ et vaut $0,850306$ si $p = 0,9$. En effet, soit aucune erreur ne s'est produite soit une erreur s'est produite.

Pour conclure, on peut comparer les deux codes rencontrés dans cet exposé. Avec le code $R3$ (triple répétition), la longueur du message envoyé est multipliée par 3. Par contre, pour $H[7,4]$, elle n'est multipliée que par $7/4$. Le second code a dès lors, sur ce point, un avantage certain. Les deux codes peuvent corriger une erreur (parfois plus pour $R3$). Pour envoyer un message de 4 chiffres significatifs sur un canal bruité avec $p = 0,9$, avec $R3$ la probabilité que le message soit correctement décodé est de l'ordre de 89% et pour $H[7,4]$, elle est de 85%.

Références

[1] S. Ling, C. Xing, Coding Theory, a first course, Cambridge University Press (2004).

[2] B. Martin, Codage, cryptologie et applications, Presses polytechniques et universitaires romandes (2004).

[3] E. Martínez-Moro, C. Munuera, D. Ruano (Ed.), *Advances in Algebraic Geometry Codes*, Series on Coding Theory and Cryptology - Vol. 5, World Scientific.

[4] V. Pless, Introduction to the Theory of Error-Correcting Codes, Wiley-Interscience series in discrete mathematics and optimization (1998).

[5] M. Rigo, Pirates informatiques et mathématique modulaire, (2007), `http ://www.discmath.ulg.ac.be/mam/`.

[6] M. Rigo, La matrice cachée de Google, (2008), `http ://www.discmath.ulg.ac.be/mam/`.

[7] C. E. Shannon, A Mathematical Theory of Communication, *Bell System Technical Journal*, Vol. 27, pp. 379–423, 623–656, (1948).

M. Rigo, Université de Liège, Institut de Mathématiques, Grande Traverse 12 (B37), B-4000 Liège. `M.Rigo@ulg.ac.be`