# Codes Correcteurs d'Erreurs
# Cours 1
# + Introduction
# + Codes linéaires en bloc

Marc Chaumont

November 12, 2008

# Sources

- "The Art of Correcting Coding", Robert H. Morelos-Zaragoza, 2002
- Cours de Pierre Abbrugiati, Université de Nice,
- Cours de Marc Uro, INT Evry.

# Plan

# Code détecteur/correcteur d'erreur

Par codes, on peut entendre plusieurs concepts distincts :

- les codes pour la cryptographie,
- les codes pour la compression,
- les codes pour la correction d'erreur.

Dans ce cours, nous nous interessons uniquement aux codes correcteurs d'erreurs.

# Constat

Dans la grande majorité des cas, une transmission de données se fait en utilisant une voie de communication qui n'est pas entièrement fiable : le **canal de communication**.

Autrement dit, les **données**, lorsqu'elles circulent sur cette voie, sont **susceptibles d'être altérées**.

Bref, il faut des mécanismes de **détection et de correction de ces erreurs**...

# Schéma classique de la théorie de l'information

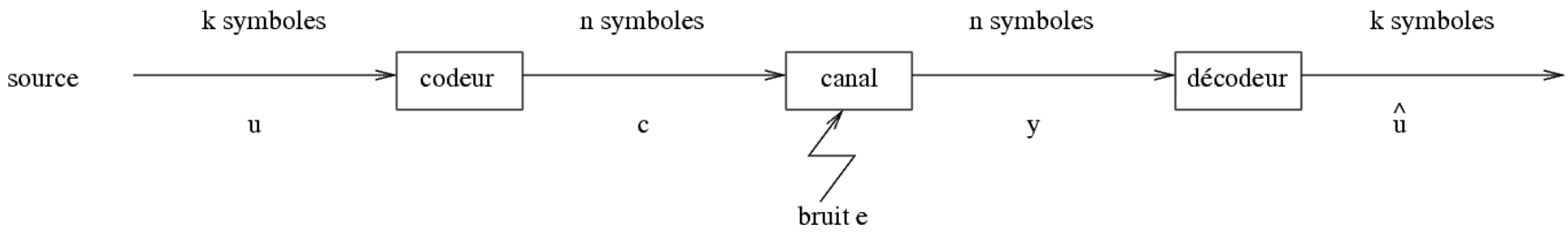



Figure: Transmission avec codage détecteur/correcteur d'erreurs

# Exemple de canal de communication

- Sur internet, (paquets IP) le code correcteur se limite à la détection des erreurs (somme de contrôle). La correction est alors réalisée par une nouvelle demande de transmission du message (protocole TCP).
- Dans le cas du disque compact, les erreurs peuvent être causées par des rayures ou des impuretés du support, elles sont moins fréquentes mais beaucoup plus volumineuses. La norme de la société Philips impose la capacité de correction d'erreurs dans le cas d'une rayure de 0,2 millimètre, dans la pratique, le code utilisé corrige jusqu'à 4096 bits consécutifs soit une rayure de plus d'un millimètre de large.
- Communications sans fils : GSM, satelite, sous-marine...

# Exemple de code **détecteur** d'erreur : **le code de parité et le CRC**

**le code de parité :**

Généralement, on ajoute à 7 bits de données 1 bit valant 1 s'il y a un nombre impair de 1, et 0 sinon. Si à la réception un des 8 bits est erroné, il y a détection d'erreur.

**contrôle de redondance cyclique : CRC**

Les séquences binaires sont traitées comme des polynômes dont les coefficients correspondent à la séquence binaire. On ajoute à la séquence binaire le reste d'une division polynomiale (division par le polynôme générateur). À la réception le reste de la disision reçu et le reste de la division calculé doivent coincider ou alors il y a erreur de transmission.

# Exemple de code détecteur et correcteur d'erreur : **le code de répétition**

**Technique de codage :**
Pour un bit d'information, 3 bits sont envoyés (cad codés) tels que :

$$0 \longrightarrow 000$$
$$1 \longrightarrow 111$$

**Technique de décodage :**
Le décodage se fait par vote majoritaire. Par exemple, si le mot reçu est 001, alors on déduit que le bit émis était 0.

# Codes blocs versus codes convolutifs

## codes blocs versus codes convolutifs

Les codes correcteurs d'erreur (ECC) peuvent être divisés en 2 classes :

- **les codes en bloc**: Ils traitent chaque bloc d'information indépendamment les uns des autres. Chaque mot de code est indépendant des autres mots de code.
- **les codes convolutifs**: La sortie d'un codeur convolutif dépend de l'information courante à coder ainsi que de l'information précédente et l'état du codeur.

Note 1 : Le choix d'un code dépend de l'application.

Note 2 : Historiquement, les codes convolutifs ont été préférés pour leur décodage "souple" et la croyance selon laquelle les codes bloc ne pouvaient pas être décodés de manière "souple".

Note 3 : Les meilleurs codes connus à ce jour (début du 21ème siècle) sont les codes blocs (irréguliers à faible densité de parité)

# Qu'attend-on d'un bon code

Un bon code doit avoir :

- un bon rendement (taux) c'est-à-dire un grand nombre de bits d'information par rapport aux bits codés.
- une bonne capacité de détection et correction d'erreurs,
- une procédure de décodage (et de codage) suffisamment simple et rapide.

Tout le problème de la théorie des codes correcteurs d'erreurs est là : construire des codes qui détectent et corrigent le plus d'erreurs possible, tout en allongeant le moins possible les messages, et qui soient faciles à décoder.

# Plan

# Rappel : Alphabet et mot

- Un **alphabet** est un ensemble fini non vide, ses éléments sont appelés **lettres** ou **symboles**. Dans le **cas binaire** l'alphabet est l'ensemble $\{0, 1\}$ que l'on notera $\mathbb{F}_2$.
- Un **message** ou un **mot d'information** ou **vecteur d'information** ou **bloc d'information** ou **code source** est une suite à valeur dans un alphabet, il correspond à une **suite de symboles**.

... exemple de mot appartenant à ${\mathbb{F}_2}^4$ : 0011.
... dit autrement, dans le cas binaire, un mot est une suite de 0 et de 1 ou une suite de bits ! $\mathbb{F}^n$ *: espace vectoriel de dimension n sur le corps fini* $\mathbb{F}$*.*

# Principe général

**Cadre du cours :** nous nous limiterons au cas binaire (alphabet $\mathbb{F}_2$).

## Principe

Tous les codes correcteurs d'erreur (ECC) reposent sur le même principe : de la **redondance** est ajoutée à de l'**information**.

## Principe général (codage en blocs)

Un message est découpé en **blocs** de $k$ bits (codage en blocs), et un même algorithme est appliqué sur chaque bloc :

- ou bien on ajoute des **bits de contrôle** à la fin de chaque bloc,
- ou bien on modifie complètement les blocs mais on évite que deux blocs différents soient transformés en un même bloc.

# Illustration d'un codage en blocs

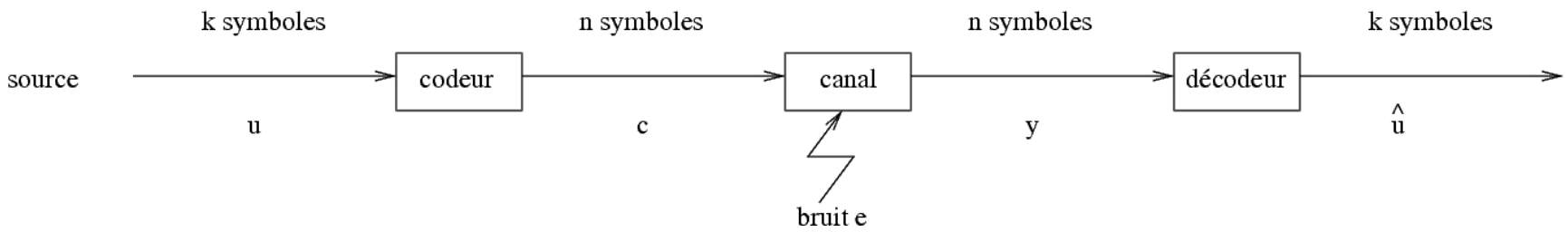



Figure: Transmission avec codage correcteur d'erreur

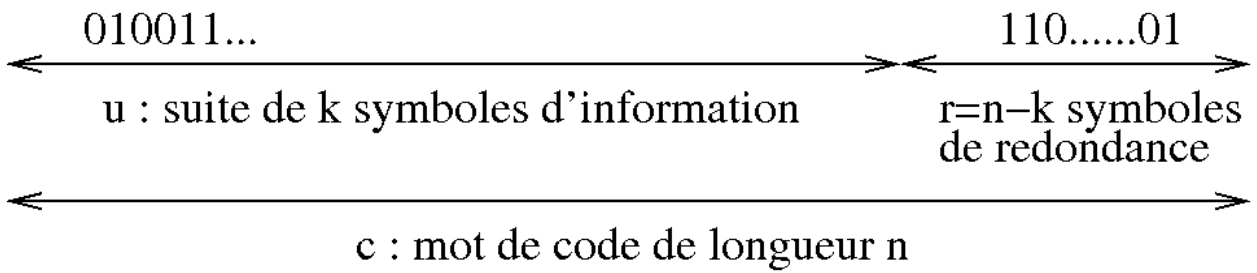



Figure: Formation d'un mot de code par ajout de redondance (code en blocs)

# Définition d'un code

## Code

Un **code** est une **application injective** (tout élément de l'ensemble d'arrivée a au plus un antécédent dans l'ensemble de départ) $\Phi : \{0,1\}^k \rightarrow \{0,1\}^n$.

Le paramètre $k$ est appelé la **dimension** du code $\phi$ et le parametre $n$ est appelé la **longueur** du code.

## code $\neq$ mot de code

L'ensemble des éléments de $C = \{\Phi(m), m \in \{0,1\}^k\}$ sont appelés les **mots de code** de $\Phi$ (par opposition aux éléments "originels" qui sont appelés **mot de sources**).

Par abus de langage on nommera $C$ le code.

# Distance de Hamming et distance minimale de Hamming

## Distance de Hamming

La **distance de Hamming**, dans le cas binaire ($\mathbb{F}_2$) entre deux vecteurs $x$ et $y$ de dimension $n$ correspond au nombre de composantes pour lequel ces deux vecteurs diffèrent.

$$d_H(x, y) = |\{i : x_i \neq y_i, 0 \leq i \leq n\}|$$

## Distance minimale d'un code C

Soit un code $C$, sa distance minimale de Hamming, $d_{min}$, est définie comme la distance minimum entre toutes les paires de mots de code de $C$ :

$$d_{min}(C) = \min_{\substack{(x,y)\in C \\ x\neq y}} d_H(x, y)$$

# Illustration sur le code binaire de répétition

Le plus simple des ECC est le **code binaire de répétition** de longueur 3. Il consiste à répéter chaque bits d'information trois fois; un "0" est codé (000) et un "1" est codé (111).

La **distance de Hamming** entre les mots de code (000) et (111) est 3.

Puisqu'il n'y a que deux mots de code pour ce code, la **distance minimale** vaut également 3.

# Les trois principaux paramètres d'un code

**Code $[n, k, d_{min}]$**

La notation $[n, k, d_{min}]$ sera utilisée pour dénoter les paramètres d'**un code en bloc** de taille $n$, qui code $k$ bits et possède une distance minimale $d_{min}$.

Remarque : Le taux du code (rendement) est $\frac{k}{n}$ c'est à dire le nombre de bits d'information par bits codés.

# Plan

# Capacité de détection d'erreurs

## Capacité de détection d'erreurs d'un code $[n, k, d_{min}]$

Le nombre d'erreurs détectables au maximum est $d_{min} - 1$

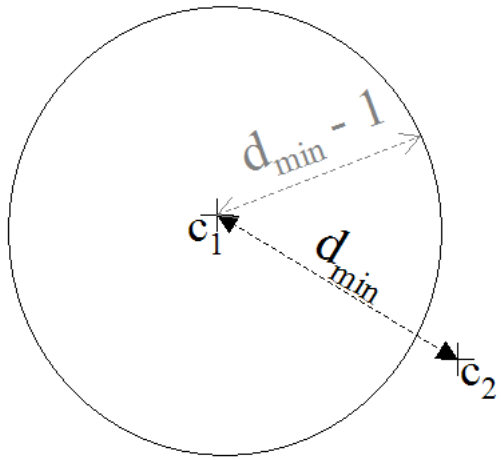



Figure: Schéma naïf de la plus grande sphere (en 2D) de détection

# Sphere de Hamming

## Sphere de Hamming

Une sphère de Hamming $S_t(x)$, de rayon $t$ et centré en $x \in \mathbb{F}_2^n$ est l'ensemble des vecteurs à une distance de x plus petite ou égale à $t$

$$S_t(x) = \{y \in \mathbb{F}_2^n | d_H(x, y) \leq t\}$$

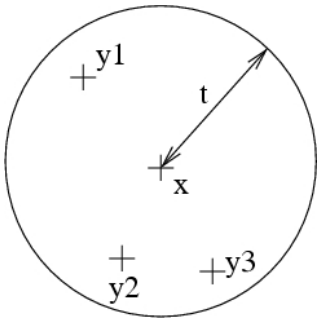



Figure: Schéma naïf d'une sphère (en 2D)

# Illustration sur le code binaire à répétition [3,1,3]

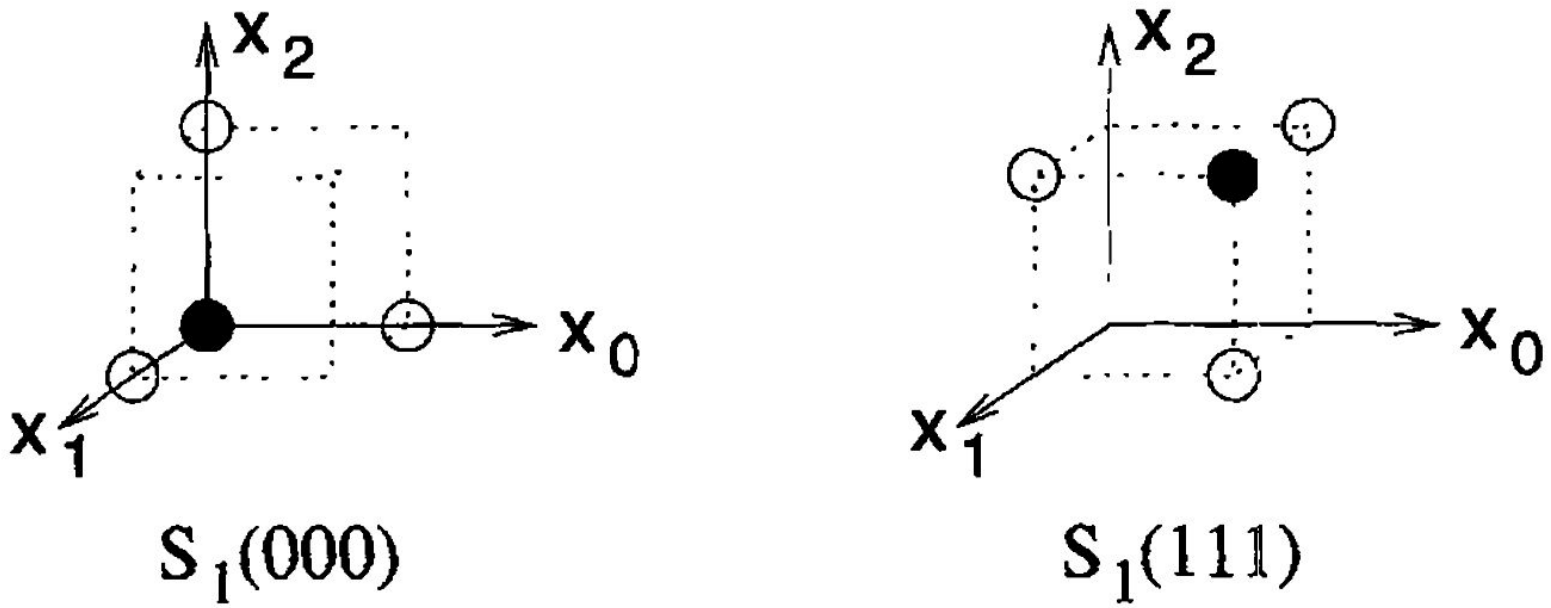



Figure: Les 2 sphères de rayon 1 autour des deux mots de code du code binaire à répétition [3,1,3]

Note : Les sphères sont disjointes.

# Capacité de correction

## Capacité de correction d'un code $C$

La capacité $t$ de correction d'erreur d'un code $C$ est le plus grand rayon des sphères de Hamming pour tous les mot de code tels que pour toute les paires $x, y \in C$ différentes, les sphères soient disjointes :

$$t = \max_{(x,y)\in C} \{l | S_l(x) \cap S_l(y) = \varnothing, x \neq y\}$$

## Capacité de correction et distance minimale d'un code C

Un code de distance minimale $d_{min}$ est suceptible de corriger $t = \lfloor (d_{min} - 1)/2 \rfloor$ erreurs. Plus précisément, si le mot $y$ reçu après transmission comporte au plus $t$ composantes erronnées, il est possible de déterminer sans ambiguïté le mot de code émis $c$.

# Illustration sur le code binaire à répétition [3,1,3]

**Codage :**

$$0 \longrightarrow 000$$
$$1 \longrightarrow 111$$

- La distance minimum est $d_{min} = 3$,
- Le nombre d'erreur maximum **détectable** est de $d_{min} - 1 = 2$ erreurs.
- Le nombre d'erreur maximum **corrigeable** est de $t = \lfloor (d_{min} - 1)/2 \rfloor = 1$ erreur.

Dérouler les $\neq$ cas...

# Calcul de la distance minimale

**Dans le cadre général**, il est pratiquement impossible de **déterminer la distance minimale** : il y a $(2^k-1)+(2^k-2)+(2^k-3)+...+1 = 2^k * (2^k+1) - 1/2 * (2^k+1)^2 - 1/2 * 2^k + 1/2$ distances à calculer. Pour $k = 50$, cela fait 633825300114114137798398181376 distances à calculer.

L'avantage des **code linéaires**, c'est que pour calculer $d_{min}$ cela nécessite seulement de calculer les **poids de Hamming** des $2^k - 1$ mot-de-code $\neq 0$! Avec k=50 cela fait seulement 1125899906842623 calculs !

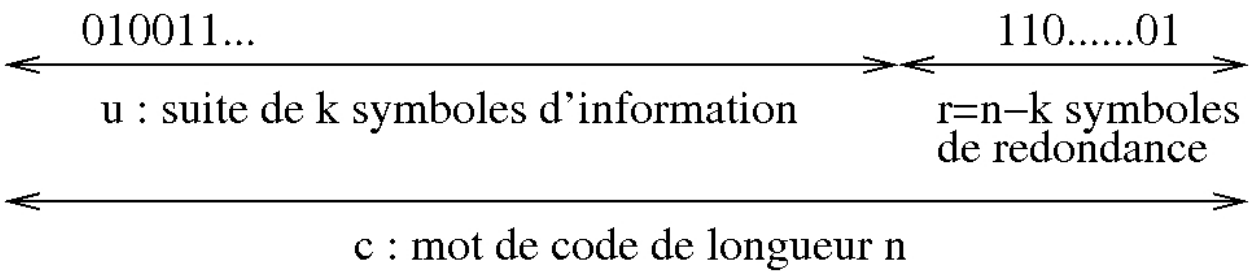
# Plan

# Exercice sur un codage en blocs

## On désire protéger un message (un mot) contre les erreurs

Soit un bloc $u$ composé de $k$ symboles. On adjoint $r$ symbole au bloc $u$; les $r$ symboles sont calculés par une fonction $\Psi$ appliquée sur $u$ et connue du codeur et du décodeur. Ces $r$ symboles forment un bloc $v = \Psi(u)$.
On appelle ces $r$ symboles:

- la redondance,
- ou les symboles de contrôles.



# Exercice sur un codage en blocs

La concaténation de $u$ (l'information) et $v$ (la redondance) donne un **mot de code** de longueur $n = k + r$.

**mot de code $\neq$ code**

L'ensemble de tous les mots obtenus (de longueur $n$ par concaténation de $u$ et $v$) de cette façon forme un **code en blocs** de **longueur** $n$ et de **dimension** $k$. La fonction $\Psi$ détermine le codage.

# Exercice : Schéma général du codage

Le codage consiste à faire correspondre à chaque groupe de $k$ **symboles d'information**, un **mot de code** particulier.

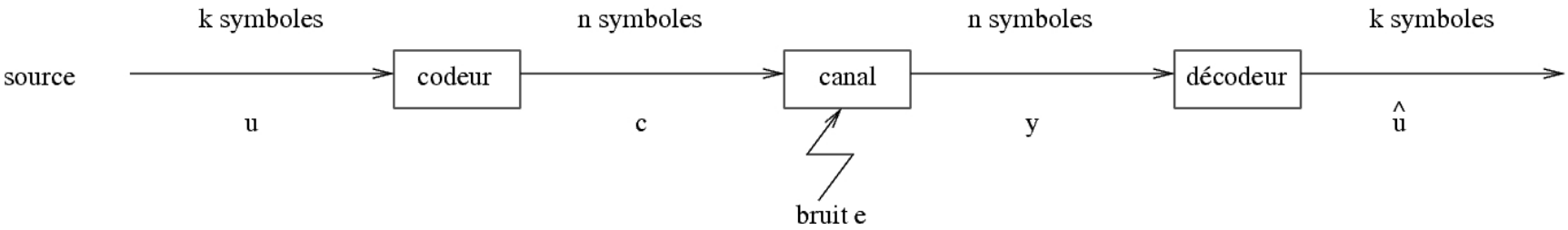



Figure: Transmission avec codage correcteur d'erreur

# Exercice : Exemple du codage en blocs

## Détection d'une erreur de transmission

Soit $y = (y_1, ..., y_n)$ un mot reçu par un récepteur. La détection d'erreur est alors très simple :

- Soit $\omega = \Psi(y_1, ..., y_k)$ la redondance calculée par le récepteur sur les k premiers symboles reçus.
- Si la redondance calculée $\omega$ est égale à la redondance reçue $(y_{k+1}, ..., y_n)$ le mot $y$ appartient au code, sinon il y a **détection d'une erreur**.

## Syndrome

Le syndrome correspond à l'erreur entre la redondance calculée et la redondance reçue. Dans le cas binaire le syndrome vaut $s = \omega - (y_{k+1}, ..., y_n)$ *mod* 2. Un syndrome nul indique qu'il n'y a pas eu d'erreur de transmission.

# Exercice : Un code de Hamming

Soit $u = (u_1, u_2, u_3, u_4)$ un bloc de 4 bits à protéger. Trois symboles de contrôle sont adjoints à $u$ pour former un mot de code. Les symboles de contrôles sont calculés comme ceci (fonction $\Psi$):

$$\begin{array}{lllllllll} v_1 & = & & & u_2 & + & u_3 & + & u_4 \\ v_2 & = & u_1 & + & & + & u_3 & + & u_4 \\ v_3 & = & u_1 & + & u_2 & & & + & u_4 \end{array}$$

en effectuant les additions modulo 2.

# Exercice : Un code de Hamming

## Rappel sur l'addition modulo 2

$$0 + 0 \ mod \ 2 = 0$$
$$0 + 1 \ mod \ 2 = 1$$
$$1 + 0 \ mod \ 2 = 1$$
$$1 + 1 \ mod \ 2 = 0$$

ce qui revient à faire un OU EXCLUSIF en logique booléenne.

## Rappel sur la soustraction modulo 2

$$a + b \ mod \ 2 = a - b \ mod \ 2$$

# Exercice : Exemple d'un code de Hamming

- L'information $u = (u_1, u_2, u_3, u_4)$
- et la redondance $v = (v_1, v_2, v_3) = \Psi(u)$
- permettent d'obtenir le **mot de code** $c = (c_1, c_2, c_3, c_4, c_5, c_6, c_7) = (u_1, u_2, u_3, u_4, v_1, v_2, v_3)$.

# Exercice : Questions

1. Enumérer l'ensemble des mots de code,
2. Donner la dimension, la longueur et la distance minimum du code,
3. Donner le taux de ce code,
4. Donner la capacité de détection,
5. Donner la capacité de correction.
6. Un mot $y$ reçu après une transmission est $y = (1000110)$. Y-a-t-il eu erreur de transmission ?
7. Donner la valeur du syndrome.

# Correction :

# correction

# Plan

# Définition

## Code linéaire

Un **code linéaire** $C$ de longueur $n$ est un sous-espace vectoriel de $\mathbb{F}^n$. Cela signifie que le **codage** peut être réalisé par des **multiplications matricielles**.

## Code linéaire : Définition bis

Un code C $[n, k, d_{min}]$ est dit linéaire s'il existe une matrice $G$ de dimension $k \times n$ dont les coefficients sont dans $\mathbb{F}$ tels que l'ensemble des mots de code soient obtenus par le produit matriciel entre les mots de source u et G. Pour $\mathbb{F}_2^n$ :

$$C = \{y | y = u.G, u \in \{0,1\}^k\}$$

# Remarque

Le code de Hamming [7, 4, 3] que nous avons vu dans l'exercice précédent est un code linéaire.

Une matrice G ($k \times n$) de ce code est (à redémontrer chez soi):

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

# Circuit imprimés

## Circuits imprimés

Un codeur et un décodeur (linéaire en blocs) peut être réalisée par utilisation de quelques portes ET et OU exclusif.

## Rappel sur l'addition modulo 2 et le produit modulo 2

| | |
|---|---|
| $0+0 \ mod \ 2 = 0$ | $0.0 \ mod \ 2 = 0$ |
| $0+1 \ mod \ 2 = 1$ | $0.1 \ mod \ 2 = 0$ |
| $1+0 \ mod \ 2 = 1$ | $1.0 \ mod \ 2 = 0$ |
| $1+1 \ mod \ 2 = 0$ | $1.1 \ mod \ 2 = 1$ |

L'addition revient à faire un OU EXCLUSIF en logique booléenne.
Le produit revient à faire un ET en logique booléenne.

# Plan

# Introduction de la forme matricielle

**Un code linéaire $[n, k, d_{min}]$ est un sous-espace vectoriel de dimension k**

Soit $C$ un code linéaire (en blocs) $[n, k, d_{min}]$. Puisque $C$ est un sous-espace vectoriel de dimension $k$, il possède une base (de vecteurs) $\{v_0, v_1, ..., v_{k-1}\}$, telle que chaque mot de code $c \in C$ peut être représenté comme une combinaison linéaire des éléments dans la base :

$$c = u_0 v_0 + u_1 v_1 + ... + u_{k-1} v_{k-1}$$

où $u_i \in \{0, 1\}, 0 \leq i < k - 1$.

# Matrice génératrice

Sous forme matricielle, cela peut se ré-écrire:

$$c = uG$$

où

$$G = \begin{pmatrix} v_0 \\ v_1 \\ \cdot \\ \cdot \\ \cdot \\ v_{k-1} \end{pmatrix} = \begin{pmatrix} v_{0,0} & v_{0,1} & \cdots & v_{0,n-1} \\ v_{1,0} & v_{1,1} & \cdots & v_{1,n-1} \\ \cdots & \cdots & \cdots & \cdots \\ v_{k-1,0} & v_{k-1,1} & \cdots & v_{k-1,n-1} \end{pmatrix}.$$

Cette matrice est appelée **matrice génératrice**.

# Matrice de vérification de parité - matrice de contrôle

Puisque $C$ est un espace vectoriel de dimension $k$, il existe un **espace dual** $C^{\perp}$, généré à partir des lignes de la matrice $H$, appelée matrice de **vérification de parité** telle que :

$$GH^t = 0$$

où $H^t$ dénote la transposée de $H$.

$$\forall c \in C, cH^t = 0$$

# Représentation systématique

## Codage systématique

Un codage est dit **systématique** lorsque l'on retrouve dans le mot codé les $k$ symboles d'information dans $k$ positions déterminées.

## Représentation systématique

Si un codage peut être mis sous forme systématique, la matrice génératrice $G$ d'un code linéaire en blocs $[n, k, d_{min}]$ peut être mis sous forme systématique $G_{sys}$ par des opérations élémentaires sur les lignes et/ou des permutations sur les colonnes. La matrice $G_{sys}$ est composée de 2 sous-matrices : la matrice identité $k \times k$ (noté $I_k$) et la matrice $k \times (n-k)$ de parité (notée $P$) telles que :

$$G_{sys} = (I_k|P)$$

# Matrice $H_{sys}$

## Calcul de la matrice de contrôle $H_{sys}$ à partir de $G_{sys}$

Puisque le produit de la matrice génératrice $G$ et de la matrice de vérification de parité transposée $H^t$ égale zéro ($G.H^t = 0$), il est simple d'obtenir la matrice de contrôle $H_{sys}$ à partir de $G_{sys}$.

$$SiG_{sys} = (I_k|P) \Rightarrow H_{sys} = (-P^t|I_{n-k}) = (P^t|I_{n-k})$$

**Démo :** $G_{sys}.H_{sys}^t = (I_k|P).(\frac{-P}{I_{n-k}}) = -P + P = 0.$

# Exemple - code binaire linéaire $[4, 2, 2]$

Matrice génératrice:

$$G = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

Matrice sous forme systématique (permutation de la deuxième et quatrième colonne) :

$$G_{sys} = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

La sous-matrice de vérification de parité vaut alors:

$$P = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

La matrice de vérification de parité (sous forme systématique ($H_{sys} = (P^t | I_{n-k})$):

$$H_{sys} = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}$$

# Exemple - code binaire linéaire $[4, 2, 2]$

Soit $u = (u_0, u_1)$ les bits d'information et $c = (c_o, c_1, c_2, c_3)$ les mots de code. La sous-matrice de vérification de parité vaut :

$$P = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

Donc les bits de contrôles $u.P$ sont tels que :

$$c_2 = u_0 + u_1$$
$$c_3 = u_0$$

| *bits d'information* | *mot de code* |
|---|---|
| (00) | (0000) |
| (01) | (0110) |
| (10) | (1011) |
| (11) | (1101) |

# Plan

# Exercice - code de Hamming $[7, 4, 3]$

- Donner la matrice génératrice de ce code (sous forme systématique),
- Donner la matrice de contrôle,
- Donner le résultat du codage de u = (1010),
- Calculer le produit de y = (1000110) par $H^t$. Qu'en déduisez-vous de y ?,
- Combien de sommes et de produits nécessite le produit matriciel (codage) ? Cela vous parait-il exagéré (complexité) ?

# Correction - code de Hamming $[7, 4, 3]$

# Correction - code de Hamming $[7, 4, 3]$

# Plan

# Le poids de Hamming = la distance !

## Poids de Hamming

Le **poids de Hamming** $w(x)$ d'un vecteur x de $\mathbb{F}_2{}^n$ est le nombre de composantes non nulles de x.

Il est donc immédiat que $\forall x \in \mathbb{F}_2{}^n, w_H(x) = d_H(x, \underline{0})$

# Exemple de calcul de poids

- $x = (1110100)$, $w(x) = 4$
- $y = (0101100)$, $w(y) = 3$

# Le poids = la distance !

Pour les codes binaires linéaires :

$$\begin{aligned} d_H(x,y) &= d_H(x+y,\underline{0}) \\ &= w_h(x+y) \end{aligned}$$

Par linéarité, $x+y \in C$ d'où :

**Une jolie propriété des codes linéaires**

Calculer la **distance minimum** de Hamming est équivalent à calculer le **poids minimum** de Hamming de tous les mots de code non nuls (il y a $2^k - 1$ mots de code).

# Exercice - code de Hamming $[7, 4, 3]$

Calculer la distance minimale du code de Hamming $[7, 4, 3]$.