M1 E3A - Voie André Ampère

UE455

Codage de source

Enseignant:
Michel Kieffer

Rédigé par:
Pierre-Antoine Comby

école normale supérieure paris-saclay

université PARIS-SACLAY

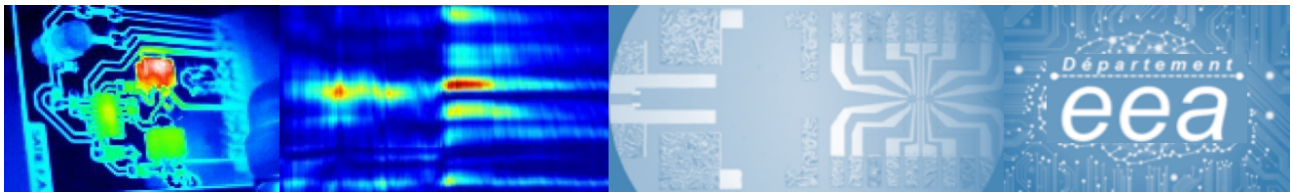


Table des matières

Chapitre 0

Rappel de probabilité

1 Rappels mathématiques pour le codage de source

Signaux et variables aléatoires Les signaux qu'on cherche à compresser (texte, échantillons de voix, musique, images, vidéo...) sont décrits comme des réalisations de suites de variables aléatoires.

Une variable aléatoire X est décrite par son domaine $\mathcal{X}$, c'est-à-dire l'ensemble des valeurs que X peut prendre (aussi appelé alphabet).

$\mathcal{X}$ peut être à valeurs discrètes (par exemple singletons $\in \{0,1\}$, $\{0,\dots 255\}$, ou triplets $\{(0,0,0)...(255,255,255)\}$ dans le cas de couleurs), ou à valeurs continues ($\in \mathbb{R}$, un intervalle $[a,b] \subset \mathbb{R}$)

2 Variables aléatoires discrètes

X est en plus caractérisée par sa *probability mass function* (loi de probabilité) $p_i = Pr(X = i), \quad i \in \mathcal{X}$

Les p_i sont tels que :

- $p_i \geq 0, \quad \forall i \in \mathcal{X}$
- $\sum_{i\in\mathcal{X}} p_i = 1$

Moyenne de X (ou espérance) : $E(X) = \sum_{i\in\mathcal{X}} ip_i$

Example : $\mathcal{X} = \{1,2,3\}$ avec $p = 1 = 0.5, p_2 = 0.25, p_3 = 0.25$.

On a $E(X) = 1 \times 0.5 + 2 \times 0.25 + 3 \times 0.25 = 1.75$

Variance de X : $V(x) = E[(X - E(X))^2] = \sum_{i\in\mathcal{X}} (i - E(x))^2 p_i$

(suite) : $\mathcal{X} = \{1,2,3\}$ avec $p = 1 = 0.5, p_2 = 0.25, p_3 = 0.25$.

On a $V(x) = (1 - 1.75)^2 \times 0.5 + (2 - 1.75)^2 \times 0.25 + (3 - 1.75)^2 \times 0.25 = 0.69$

Écart-type de X $\sigma(X) = \sqrt{V(X)}$

Générer des réalisations de VA : Génération d'une suite de réalisations d'une VA X tel que $\mathcal{X} = \{0, 1\}$ avec $p_0 = 0.9, \quad p_1 = 0.1$

```
x=rand(1,10)>0.9
% rand : generateur de loi uniforme
% randn : generateur de loi gaussienne
```

Pour générer une suite de réalisations correspondant aux exemples précédents

```
x=rand(1,10)
y= (x<0.5) + 2*(x<0.75 & x>0.5) + 3*(x>0.75)
```

On considère deux variables aléatoires X_1 et X_2 d'alphabet $\mathcal{X}$.

Indépendance X_1 et X_2 sont indépendantes si et seulement si

$$Pr(X_1 = i, X_2 = j) = Pr(X_1 = i).Pr(X_2 = j)$$

Ainsi $E(X_1X_2) = E(X_1).E(X_2)$

Dans le cas général,

$$Pr(X_1 = i, X_2 = j) = Pr(X_1 = i/X_2 = j).Pr(X_2 = j) = Pr(X_2 = j/X_1 = i).Pr(X_1 = i)$$

Si X_1 et X_2 sont indépendants $Pr(X_1 = i/X_2 = j) = Pr(X_1 = i)$ pour tous i, j
$\sum_{j\in\mathcal{X}} Pr(X_2 = j/X_1 = i) = 1$ mais $\sum_{i\in\mathcal{X}} Pr(X_2 = j/X_1 = i) =?$

Marginalisation On suppose connaître $Pr(X_2 = j/X_1 = i)$ et $Pr(X_1 = i)$.
D'après la règle du produit,

$$Pr(X_2 = j) = \sum_{i\in\mathcal{X}} Pr(X_2 = j, X_1 = i) = \sum_{i\in\mathcal{X}} Pr(X_2 = j/X_i = i).Pr(X_1 = i)$$

Chapitre 1

Introduction - Motivation

Pourquoi a-t-on besoin du codage de source ? Un téléviseur HD affiche des images de 1920 x 1080 pixels. Chaque pixel est formé d'une combinaison de 3 couleurs RGB, chacune des couleurs étant codée par un nombre sur 8, 10 ou 12 bits. À raison de 25, 50 ou 60 images par seconde, le débit nécessaire est R= 1920 x 1080 x 3 x 8 x 25 = 1,22 Gbits par seconde.

En 4G, le débit maximal est de 100 Mbits par seconde (quand on est seul dans la zone) et en ADSL, il est de 20 Mbits par seconde.

Il faut compresser les données avec un facteur 100 au minimum.

"Le taux minimum c'est 25 images par seconde, pour pas avoir l'impression de regarder un dessin animé japonais." "Des émissions mettent volontairement moins pour qu'on ait l'impression d'avoir trop bu en regardant la télé."

Comment faire de la compression ? Quelles propriétés du signal vidéo peut-on utiliser pour réaliser de la compression ? On utilise la redondance statistique. Par exemple, pour la vision (ou tech 3D) on utilise les petites différences pour obtenir la profondeur. De la même façon, en stéréo on a deux micros pour l'enregistrement. C'est la ressemblance entre les deux signaux qui nous intéresse ici pour effectuer la compression.

La compression est possible à cause de plusieurs propriétés :

- La corrélation temporelle (ressemblance entre deux image successives d'une vidéo ou échantillons audio successifs).
- La corrélation spatiale (le fait que certaines zones présentes sont relativement uniforme, ressemblance entre deux pixels voisins).
- La corrélation spectrale (ressemblance entre les composantes R, G et B d'une image).
- Les propriétés statistiques du signal (un texte contient plus de "e" que de "z" en général).

Exemple d'une chaîne de compression vidéo On considère une vidéo mono-vue (pas en 3D) codée en niveaux de gris.

Transformation : On applique une transformation à l'image, c'est à dire que l'on cherche à exprimer les blocs de l'image dans une base autre que la base canonique, pour permettre une

compression plus facile.
Par exemple :

$\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$ permet de décrire les zones constantes

$\begin{pmatrix} 1 & 1 \\ -1 & -1 \end{pmatrix}$ pour décrire les variations verticales, et

$\begin{pmatrix} 1 & -1 \\ 1 & -1 \end{pmatrix}$ pour une variation horizontale.

Quantification : Elle permet de représenter les pixels transformés sur un nombre limité de bits (et d'en mettre plein à 0). Cette opération est irréversible.

Codeur entropique : Il exploite la redondance statistique des coefficients quantifiés (il prend les successions et les compresse). On ne peut pas encore envoyer les données sur le réseaux, il faut appliquer un standard de compression.

Standard de compression : Détermine la manière dont les bits en sortie sont organisés. (constitue les paquets, les numérote etc.)

On peut alors transmettre l'image. Une fois récupérée, on applique un décodage entropique, puis un désindexation et enfin une transformation inverse.

La désindexation permet d'obtenir à partir d'un coefficient quantifié, un nombre qui a la même dynamique qu'un coefficient transformé.(exemple : permet de retrouver un nombre à la bonne échelle, même si ce nombre n'est pas exactement le même).

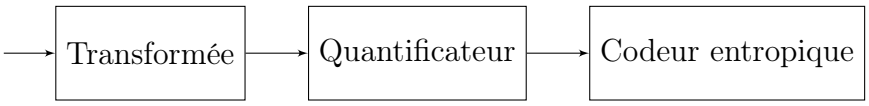


Figure 1.1 – utilisation d'un décodeur local

Figure 1.2 – Chaine de compression

Figure 1.3 – Chaine de décompression

Comment transmettre une vidéo ? appliquer ce shcéma (JPEG) à chaque image et transmettre ? non, trop lourd. On peut comparer l'image 2 à l'image précédente non pas 1 mais $\hat{1}$ l'estimée de l'image reçue par le récepteur, et l'on envoie la différence. Puis pour envoyer l'image 3, on estime la différence précédente, on y ajoute l'image estimée de la première image et on calcule la différence, que l'on envoie au travers des différentes transformations. Et ainsi de suite.

De ce fait, au niveau du récepteur on mémorise l'image précédente à laquelle on ajoute les différences.

Cette structure de codeur était celle des premiers codeurs vidéo (H261, MPEPZ). H265 encore utilisé a aussi cette structure.

Notations :
I_n : image numéro n
$\hat{I_n}$: image numéro n au décodeur
$f(\hat{I_n}) = \tilde{I_{n+1}}$: image n+1 prédite

On va étudier deux cas :

1. pas de décodeur au niveau du codeur mais on a un prédicteur.

$$\begin{aligned} \tilde{I_{n+1}} &= T^{-1}(Q^{-1}(Q(T(I_{n+1} - f(I_n))))) + \tilde{I_{n+1}} \\ &= I_{n+1} - f(I_n) + E_{n+1} + \tilde{I_{n+1}} \text{ avec, E le bruit} \end{aligned}$$

   $f(I_n)$ et $\tilde{I_{n+1}}$ ne se compensent pas totalement.

2. On fait la prédiction a partir des images codées précédemment.

$$\begin{aligned} \tilde{I_{n+1}} &= T^{-1}(Q^{-1}(Q(T(I_{n+1} - f(\hat{I_n}))))) + f(\hat{I_n} \\ &= I_{n+1} - f(\hat{I_n}) + E_n + f(\hat{I_n}) \\ &= I_{n+1} + E_n \end{aligned}$$

   L'utilisation d'un décodeur au niveau du codeur (décodeur local) permet d'éviter une accumulation des erreurs au niveau des images décodées.

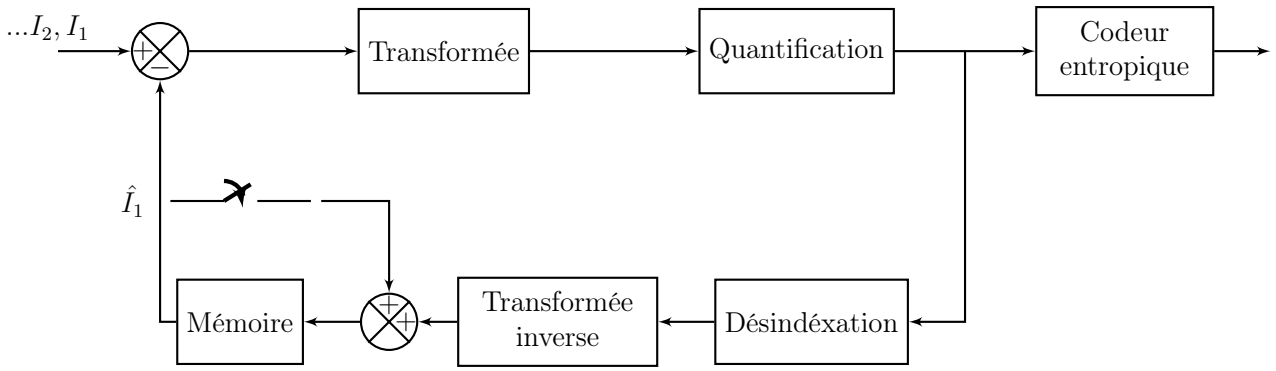


FIGURE 1.4 – Utilisation d'un décodeur local

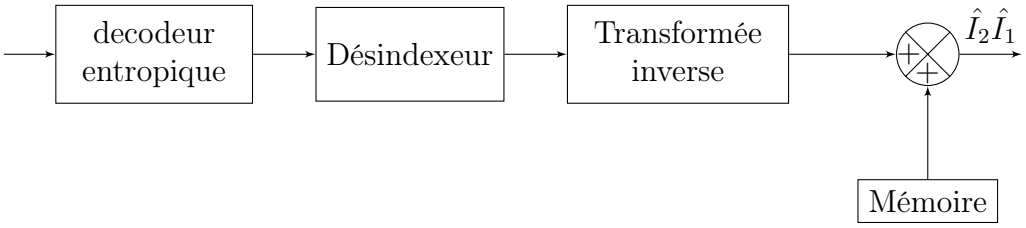


FIGURE 1.5 – Décodeur à mémoire

Chapitre 2

Codage entropique (sans pertes)

1 Modèles de sources

On s'intéresse à la compression sans perte d'un message constitué de symboles appartenant à un alphabet fini $\mathcal{X}$.

Ces symboles sont :

- les caractères d'un texte
- la sortie du quantificateur d'un codeur vidéo.
- ...

On fait l'hypothèse que le message est une réalisation d'une suite de variables aléatoires X_1,...,X_n.

1.1 Modèle stationnaire sans mémoire

C'est le modèle le plus simple, on suppose que :

- les VA x_i sont distribuées de la même manière, quelque soit n (stationnarité).
- les VA sont indépendantes entre elles.

C'est un mauvais modèle, mais il est simple.
Pour décrire ce modèle, il suffit de disposer de $p_i = Pr(X_n = i), \quad \forall i \in \mathcal{X}$.

Exemple: On considère une source binaire X, à valeurs dans $\mathcal{X} = \{0, 1\}$, et

$$p_0 = Pr(X = 0), \quad p_1 = Pr(X = 1) = 1 - p_0$$

Définition

L'information associée à chaque symbole de la VA X est

$$I(i) = -log_2 p_i \text{ (en bits/symbole)}$$

Remarque: C'est une fonction décroissante sur]0,1] qui s'annule en 1. En effet, l'information associée à quelque chose de certain ($p_i = 1$) est nulle, alors que celle associée à quelque chose de peu probable ($p_i \to 0$) est très grande ($\to \infty$).

Définition

L'information moyenne associé aux symboles de X ou à X est appelée *entropie* de la source X.

$$H(X) = -\sum_{i \in \mathcal{X}} p_i log_2(p_i) = \sum_{i \in \mathcal{X}} p_i log_2(\frac{1}{p_i})$$

On rappelle que $log_2(2^n) = n$.

Example : Pour la source binaire :
si $p_0 = 0.5$ et $p_1 = 0.5$ alors $H(X) = 1$ bit/symbole.
si $p_0 = 0.01$ et $p_1 = 0.99$ alors $H(X) = 0.08$ bit/symbole.

Proposition *(Propriété de l'entropie)*

On considère deux VA X_1 et X_2

$$\begin{aligned} H(X_1, X_2) &= -\sum_{i \in \mathcal{X}, j \in \mathcal{X}} Pr(X_1 = i, X_2 = j) log_2(Pr(X_1 = i, X_2 = j)) \\ &= H(X_2|X_1) + H(X_1) \\ &= H(X_1) + H(X_2) \iff X_1 \perp X_2 \end{aligned}$$

Plus généralement, si on a N VA iid alors $H(X_1, ..., X_N) = NH(X_1)$.
Et on a

$$\boxed{H(X) \leq log_2|\mathcal{X}|}$$ (nombre d'élément de $\mathcal{X}$).

1.2 Modèle de Markov d'ordre 1

Dans ce modèle, la probabilité d'apparition du symbole n ne dépend que de la réalisation du symbole $n-1$.

Les probabilités de transition vérifient donc pour une source stationnaire :

$$Pr(X_n = a_n | X_{n-1} = a_{n-1}, X_{n-2} = a_{n-2}, ..., X_1 = a_1) = Pr(X_n = a_n | X_{n-1} = a_{n-1})$$

On considère des sources de Markov d'ordre 1 stationnaires, donc :

$$Pr(X_n = a_n | X_{n-1} = a_{n-1}) = Pr(X_{n-1} = a_n | X_{n-2} = a_{n-1}), \forall n$$

Définition

Pour décrire une source de Markov d'ordre 1 stationnaire, il suffit de décrire ses *probabilités de transition :*

$$Pr(X_n = i|X_{n-1} = j) = p_{i|j}, \quad \forall i \in \mathcal{X}, \forall j \in \mathcal{X}$$

Exemple: Comment estimer les probabilités de transition ? on a la séquence : a a b b a c a b b a

- Modèle sans mémoire :
  on estime $\hat{p_a} = \frac{5}{10}$, $\hat{p_b} = \frac{4}{10}$ et $\hat{p_c} = \frac{1}{10}$

- Modèle de Markov d'ordre 1 :

$$\begin{aligned} Pr(X_n = i, X_{n-1} = j) &= Pr(X_n = i|X_{n-2} = j)Pr(X_{n-1} = j) \\ &= \frac{Pr(X_n = i, X_{n-2} = j)}{Pr(X_{n-1} = j)} \end{aligned}$$

  Avec $j = a$, si $i = a$ alors

$$Pr(X_n = a|X_{n-1} = a) = \frac{\text{nombre de paires aa}}{\text{nombre de paires débutant par a}} = \frac{1}{4}$$

  si $i = b$ alors

$$Pr(X_n = b|X_{n-1} = a) = \frac{\text{nombre de paires ab}}{\text{nombre de paires débutant par a}} = \frac{2}{4}$$

Définition

On défini la matrice de transition :

$$\Pi = (p_{a_j|a_i})_{(i,j)} = \begin{pmatrix} p_{a_1|a_1} & p_{a_2|a_1} & \cdots & p_{a_j|a_1} \\ p_{a_1|a_2} & p_{a_2|a_2} & \cdots & p_{a_j|a_2} \\ \vdots & & & \\ p_{a_1|a_J} & p_{a_2|a_j} & \cdots & p_{a_J|a_J} \end{pmatrix}$$

avec $J = |\mathcal{X}|$ nombre d'éléments de $\mathcal{X}$.
La somme de chaque ligne de Π vaut 1.

Exemple: On considère une source de Markov binaire :

$$p_{0|0} = 0.9, \quad p_{1|0} = 0.1, \quad p_{0|1} = 0.3, \quad p_{1|1} = 0.7$$

On a donc :

$$\mathbf{\Pi} = \begin{pmatrix} 0.9 & 0.1 \\ 0.3 & 0.7 \end{pmatrix}$$

Définition

Pour cette source de Markov, on peut être intéressé par les *probabilités stationnaires* $Pr(x_n = i)$ et on note :

$$\begin{aligned}\mathbf{p}^T &= [Pr(X_n = a_1), \ldots, Pr(X_n = a_J)] \\ &= [p_{a_1}, ..., p_{a_J}]\end{aligned}$$

On peut montrer que $\mathbf{p}$ satisfait :

$$\boxed{\mathbf{p}^T = \mathbf{p}^T\mathbf{\Pi}}$$

Proposition *(Entropie d'une source de Markov d'ordre 1)*

$$\begin{aligned}H(X) &= -\sum_{i\in\mathcal{X}} p_i \sum_{j\in\mathcal{X}} p_{i|j} log_2(p_{i|j}) \\ &= -\sum_{i\in\mathcal{X}}\sum_{j\in\mathcal{X}} p_{i,j} log_2(p_{i|j})\end{aligned}$$

où $p_{i,j} = Pr(X_n = i, X_{n-1} = j)$

Démonstration : On considère une source de markov d'ordre 1 à valeur dans $\mathcal{X}$ et unvecteur de longueur n de VA de cette source $(X_1 \ldots X_n) \in \mathcal{X}^n$:

$$H(X_1 \ldots \mathcal{X}_n) = -\sum_{x_1,\ldots x_n\in\mathcal{X}^n} P((X_1...X_n) = (x_1..x_n)) \log_2(P((X_1...X_n) = (x_1..x_n)))$$

On a modèle d'ordre 1 donc :

$$\begin{aligned}&= \sum_{\mathbf{x}\in\mathcal{X}^n} P(\mathbf{X} = \mathbf{x}) \left(\sum_{i=2}^{n} \log_2(P(X_i = x_i|X_{i-1} = x_{i-1})) + \log_2(P(X_1 = x_1))\right) \\ &= -\sum_{i=2}^{n} \sum_{x_i,x_{i-1}\in\mathcal{X}^2} P(X_i = x_i, X_{i-1} = x_{i-1}) \log_2(P(X_i = x_i|X_{i-1} = x_{i-1}))\end{aligned}$$

Pour une chaine stationnaire

$$= (n-1) \sum_{x_1,x_2\in\mathcal{X}^2} P(X_1 = x_1, X_2 = x_2) \log_2(P(X_1 = x_1|X_2 = x_2)) + H(X_1)$$

L'entropie par symbole (débit d'entropie) de cette source est alors

$$\lim_{n\to+\infty} \frac{1}{n} H(X_1...X_n) = -\sum_{x_1,x_2\in\mathcal{X}^2} P(X_1 = x_1, X_2 = x_2) \log_2(P(X_1 = x_1|X_2 = x_2))$$

■

Remarque: Avec un modèle de Markov, si on essaie de "créer" des mots lettre par lettre, plus on monte dans les ordres, plus la structure de la langue commence à apparaître. À partir de l'ordre 3, il apparaît des mots qui seraient potentiellement de la langue considérée. Ce modèle peut être adapté, comme par exemple pour le correcteur orthographique des téléphones.

L'idée générale de la compression d'une source de Markov est d'effectuer des prédictions pour avoir accès aux informations, de sorte qu'il ne reste à transmettre que ce que l'on ne peut pas prédire.

2 Codes

2.1 Définitions et propriétés

On considère une source X à valeurs dans $\mathcal{X} = \{a_1, ..a_J\}$.

Définition

Un *code* est un ensemble binaire de $\{0,1\}^*$ (union de tous les ensembles $\{0,1\}^2 = \{00, 11, 01, 10\}$, $\{0,1\}^3 = ...$ ...). Un code est donc un sous-ensemble de $\{0,1\}^*$.

Définition

Une fonction de codage $c : \mathcal{X} \to C$ associe à chaque élément de $\mathcal{X}$, un élément de C.

Définition

La *longueur d'un mot de code* associé à $x \in \mathcal{X}$ est notée

$$l(x) = l(c(x)) = \text{ nombre de bits de } c(x)$$

Pour une source sans mémoire avec $p_j = Pr(X = a_j)$. La longueur moyenne du code C associé à $\mathcal{X}$ est:

$$\bar{l} = \sum_{j=1}^{J} p_j l(a_j)$$

L'objectif du codage sans perte est de minimiser $\bar{l}$ tout en étant capable de décoder le code sans ambiguïté.

Définition

Un code C (et sa fonction de codage associée c) est dit *non singulier* si:

$$x_1 \neq x_2 \Rightarrow c(x_1) \neq c(x_2)$$

Définition

L'extension C^* d'un code C est l'ensemble de toutes les suites, finies et infinies, de mots provenant du code C.

Exemple: Si C = $\{0, 10\}$ alors $\{C^* = \{0, 10, 00, 010, 100, 000, 1010, ...\}$

Proposition

Un code C est décodable de façon unique si son extension C^* est non singulière.
Si on prend deux successions de symboles dans $\mathcal{X}$:

$x_1x_2...x_N \neq x'_1x'_2...x_{N'} \Rightarrow c(x_1, ...x_N) = c(x_1)c(x_2)...c(x_N) \neq c(x'_1)...c(x'_N)$

Définition

Un code C est un code préfixe si aucun mot de code n'est le début d'un autre mot de code.

Exemple: $\mathcal{X} = \{1, 2, 3, 4\}$

X	Singulier	Non singulier	Décodable de manière unique	Préfixe
1	0	0	10	0
2	0	010	00	10
3	0	01	11	110
4	0	10	110	111

Remarque: "Décodable de manière unique" implique "Non singulier".

2.2 Inégalités

Proposition *(Inégalité de Kraft)*

Un code binaire préfixe peut exister à condition que ses longueurs de mot de code $l_1, ...l_J$ satisfassent :

$$\sum_{j=1}^{J} 2^{-l_j} \leq 1$$

Démonstration : Condition nécessaire :

Soit l_{max} la plus grande des longueurs. Le nombre de feuilles à la profondeur l_{max} que peut porter un arbre dont la racine est à la profondeur l_j est $2^{l_{max}-l_j}$.

Le nombre maximum de feuilles d'un arbre de profondeur l_{max} est $2^{l_{max}}$.

On a $\sum_{j=1}^{J} 2^{l_{max}-l_j} \leq 2^{l_{max}}$ d'où le résultat.

Condition suffisante : ? ■

Proposition *(Inégalité de Kraft-McMillan)*

Un code binaire décodable de manière unique peut exister à condition que ses longueurs de mot de code $l_1, ...l_J$ satisfassent :

$$\sum_{j=1}^{J} 2^{-l_j} \leq 1$$

Démonstration : Par l'absurde ■

Remarque: Attention, ce sont des théorèmes d'existence : ils ne peuvent pas servir à vérifier qu'un code est préfixe ou décodable de manière unique.

Exemple: Le code $\{1, 00, 10\}$ n'est pas préfixe, pourtant on a $\sum 2^{-l_i} = 2^{-1} + 2^{-2} + 2^{-2} = 1$. On sait seulement qu'il existe un code préfixe dont les mots de code ont ces longueurs.

Remarque: On sait que "préfixe" implique "décodable de manière unique". En revanche, si un code est décodable de manière unique, on peut seulement dire qu'il existe un code préfixe équivalent, sans qu'il soit le même.

Corollaire

Pour qu'un code binaire soit préfixe (ou décodable de manière uniquer), *il faut* que ses longueurs de mot de code $l_1, ...l_J$ satisfassent :

$$\sum_{j=1}^{J} 2^{-l_j} \leq 1$$

3 Code de longueur minimale

On considère une source X sans mémoire d'alphabet $\mathcal{X} = \{a_1, ..., a_J\}$ et de probabilités $p_j = Pr(X = a_j)$. On souhaite construire un code préfixe de longueur moyenne minimale associé à X. Si on note $l_1, ..., l_J$ les longueurs des mots de codes associés à $a_1, ...a_J$ la longueur moyenne sera :

$$\bar{l} = \sum_{j=1}^{J} p_j l_j$$

On cherche à minimiser $\bar{l}$ en s'assurant que le code reste préfixe, c'est à dire que :

$$\sum_{j=1}^{J} 2^{-l_j} \leq 1 \text{ ou } \sum_{j=1}^{J} 2^{-l_j} - 1 \leq 0$$

Il s'agit d'un problème d'optimisation sous contrainte que l'on résout en introduisant le Lagrangien, avec μ le multiplicateur de Lagrange :

$$L(l_1, ...l_J, \mu) = \sum_{j=1}^{J} p_j l_j + \mu \times (\sum_{j=1}^{J} 2^{-l_j} - 1)$$

Une condition nécessaire d'optimisation est que :

$$\begin{cases} \frac{\partial L}{\partial l_j} = 0 & \forall j = 1, ..., J \\ \frac{\partial L}{\partial \mu} = 0 \end{cases}$$

On a donc :

$$\begin{cases} \frac{\partial L}{\partial \mu} & = \sum_{j=1}^{J} 2^{-l_j} - 1 = 0 \\ \frac{\partial L}{\partial l_j} & = p_j - \mu . 2^{-l_j} . \ln 2 = 0 \ , \ \forall j = 1, ..., J \end{cases}$$

En sommant ces égalités et en injectant l'égalité précédente de Kraft, on obtient :

$$\mu = \frac{1}{ln2}$$

En remplaçant dans $p_j - \mu . \ln 2 . 2^{-l_j} = 0$, et en passant au log en base 2 :

$$\boxed{l_j = -log_2 p_j}$$

Pour ce choix de longueurs de mots de code, on obtient :

Proposition

Le meilleur code préfixe a une longueur moyenne au minimum égale à l'entropie de la source.

$$\boxed{\bar{l} = \sum_{j=1}^{J} p_j (-log_2 p_j) = H(x)}$$

Remarque: L'entropie (avec une source sans mémoire) donne une borne supérieure de la quantité d'information réelle contenue dans un message. Dans le cas d'un texte, l'entropie au sens d'une source de Makorv diminue quand on augmente l'ordre, jusqu'à se stabiliser au bout d'un certain ordre.

Proposition ***(Code préfixe à longueur minimale)***

- Si $p_i \geq p_j$, $l_i \leq l_j$
- Les noms de codes associé aux deux symboles les moins probables sont de même longueur
- Les noms de codes associé aux deux symboles les moins probables ne diffère que d'un seul bit à la fin.

3.1 Codage de Huffmann

On considère une source X à valeurs dans $\mathcal{X}$, de probabilités associées $p_i = Pr(X = a_i)$, pour $i \in \mathcal{X}$.

Principe d'un code de Huffmann À partir de ces propriétés, on peut fabriquer un code de Huffmann.

Initialement, on considère une source $X^{(0)}$ à J symboles $\mathcal{X} = \{a_1, \dots a_J\}$

1. **Entrée** $\chi = \{1...N\}, \mathbf{p} = (p_1...p_n)^T T$
2. Si $N = 2$
   - $\mathcal{C} = \{0, 1\}$
   - renvoyer $\mathcal{C}$
3. Sinon :
   - Trouve les indices i et j des deux symboles les moins probables.
   - Construire $\begin{array}{l} \chi' = \{1, ..., i-1, i+1, ..., j-1, j+1, ..., N\} \\ \mathbf{p}' = (p_1, ..., p_{i-1}, p_i + p_j, p_{i+1}, ..., p_{j-1}, p_{j+1}, ..., p_N) \end{array}$
   - $\mathcal{C}' = (\chi', \mathbf{p}')$.
   - On fabrique $\mathcal{C}$ à partir de $\mathcal{C}'$ en rajoutant 0 ou 1 à c'_i Soit :

$$\mathcal{C} = \{c_1, ..., c_{i-1}, (c'_i; 0), c_{i+1}, ..., c_{j-1}, (c'_i, 1)\}$$

Proposition

On montre que la longueur moyenne $\bar{l}$ d'un code de Huffmann associé à une source satisfait

$$H(x) \leq \bar{l} \leq H(X) + 1$$

On peut obtenir des codes plus performants en codant des paires de symboles, des triplets... des N-uplets de symboles. Avec ce type de technique, la longueur moyenne $\bar{l}$ par symbole de la source initiale satisfait :

$$H(X) \leq \bar{l} \leq H(X) + 1$$

L'inconvénient est la taille de la table de Huffmann à gérer.

Remarque:

- Ce type de code est efficace lorsque $H(x) \gg 1$.
- Dans le cas ou $H(x) < 1$ il est possible de construire un code de Huffman pour des groupes de M symboles. Dans ce cas on a :

$$MH(x) \leq \overline{l_M} < MH(x) + 1$$

Il reste cependant plusieurs problèmes :

- Grande taille du code de Huffmann
- Il faut donner les infos nécessaires aux décodeur pour le décodage :
  - On transmet le code(couteux)
  - On transmet le vecteur de probabilité $\mathbf{p}$ (plus complexe)
  - On utilise un code standardisé (ex : JPEG)

Exercice : Coder l'algorithme de Huffman dans le langage de votre choix.

3.2 Codage arithmétique

On considère une source binaire X avec $p_0 = Pr(X = 0)$ et $p_i = Pr(X = i)$. Cette source génère un message $x_{1:N}$ de longueur N. On va associer un code $\underline{c}(x_{1:N})$ à $x_{1:N}$ qui sera un nombre dans l'intervalle $[0, 1[$.

Exercice : Construire une fonction `a = binaire(x,m)` qui donne les m premier bits de la représentation binaire de $x \in [0, 1]$.

On essaie de représenter $(x_{1:N})$ avec peu de bits si $x_{1:N}$ est très probable et avec plus de bits si $x_{1:N}$ est moins probable.

3.2.1 Algorithme de codage arithmétique en précision inifinie

On considère une source binaire sans mémoire décritre par $p = (p_0, p_1)^T$ et ayant généré $x = (x_1...x_n)$ à coder.

1. **Initialisation** $l_0 = 0, \quad h_0 = 1, \quad i = 1$

2. **Étapes : pour n allant de 1 à N**
   (a) Si $x_n = 0$ alors $\begin{cases} l_n = l_{n-1} \\ h_n = l_{n-1} + (h_{n-1} - l_{n-1})p_0 \end{cases}$
   (b) Si $x_n = 1$ alors $\begin{cases} l_n = l_{n-1} + (h_{n-1} - l_{n-1})p_0 \\ h_n = h_{n-1} \end{cases}$
   (c) On a $h_N - l_N = p(x_{1:N})$
3. On pose
$$\lambda(\mathbf{x}) = \frac{l_N + h_N}{2} \text{ et } \mu(\mathbf{x}) = \lceil -\log_2(h_n - l_n)) \rceil$$
4. On a alors le code arithmétique associé :
$$\overline{c}(\mathbf{x}) = \lfloor \lambda(\mathbf{x}) \rfloor_{\mu(\mathbf{x})+\mathbf{1}}$$

où $\lfloor a \rfloor_\lambda$ est la représentation binaire de a tronquée à λ bits. (Exemple : $\lfloor 0,1011 \rfloor_2 = 0,10$)

3.2.2 Codes arithmétique implantable

Le code arithmétique vu en 3.2.1 possède plusieurs défauts :

- Il faut coder tout le vecteur généré pour la source pour obtenir le code arithmétique. Ceci introduit du délai de codage.
- Les bornes inf et sup de l'intervalle de codage deviennent de plus en plus proches il faut les représenter en précision infinie.

L'idée du coadge arithmétique "pratique" est d'émettre ou de stocker sur le disque les bits de code dès qu'ils sont déterminé sans ambiguité. On peux alors dilater l'intervalle de codage.

1. **Entrée : p**, **x**
2. **Initialisation**$l_0 = 0, \quad h_0 = 1, \quad , f = 0, n = 1$
3. **Étapes : pour k allant de 1 à N**
   (a) Si $x_k = 0$ alors $\begin{cases} l_k = l_{k-1} \\ h_k = l_{k-1} + (h_{k-1} - l_{k-1})p_0 \end{cases}$
   (b) Si $x_k = 1$ alors $\begin{cases} l_k = l_{k-1} + (h_{k-1} - l_{k-1})p_0 \\ h_k = h_{k-1} \end{cases}$
4. (a) Si $[l_k, h_k[\subset [0, 0.5[$:
   $c = [c, 0\underbrace{1...1}_{f}[$ et $[l_k, h_k[= [2l_k, 2h_k[$
   (b) Sinon si $[l_k, h_k[\subset [0.5, 1[$:
   $c = [c, 1\underbrace{0...0}_{f}[$ et $[l_k, h_k[= [2l_k - 1, 2h_k - 1[$
   (c) Sinon si $[l_k, h_k[\subset [0.25, 0.75[$:
   $f = f + 1$, $[l_k, h_k[= [2l_k - 0.5, 2h_k - 0.5[$
5. Tant que $[l_k, h_k[\subset [0, 0.5[$ ou $[l_k, h_k[\subset [0.5, 1[$ ou $[l_k, h_k[\subset [0.25, 0.75[$:
   aller en 4

Remarque: en pratique le codage arithmétique se fait sur des intervalles entiers en partant de $[l_0, h_0[= [0, 2^M]$ après découpages, les bornes des intervalles sont arrondies. On test si :

$$\begin{aligned}[l_k, h_k] &\subset [0, 2^{M-1}[\\ &\subset [2^{M-1}, 2^M[\\ &\subset [2^{M-2}, 2^{M-1} + 2^{M-2}]\end{aligned}$$

Les probabilités d'apparition de chaque symbole sont estimée en cours de codage.

3.2.3 Algorithme de décodage arithmétique en précision infinie

Le décodeur arithmétique va chercher à déterminer les selections des sous intervalles faites par le codeur.

Entrée : $\mathbf{c}, \mathbf{p}, N$

1. Initialisation : $[l_0, h_0[= [0, 1[$ On note $\tilde{\lambda}$ le nombre dont la représentation est $\mathbf{c}$
2. **Pour i allant de 1 à n** :
   - Si $\tilde{\lambda} \in [l_{i-1}, l_{i-1} + p_0(h_{i-1} - l_{i-1})]$ alors : $x_i = 0$ et $[l_i, h_i[= [l_{i-1}, l_{i-1} + p_0(h_{i-1} - l_{i-1})]$
   - sinon $x_i = 1$ et $[l_i, h_i[= [l_{i-1} + p_0(h_{i-1} - l_{i-1}), h_{i-1}]$

3.2.4 Performance

Il faut montrer que $c(x_{1:N}) \in [l_N, h_N[$ (et qu'alors on pourra décoder), et que cette procédure de codage est efficace, ie $E(\mu(X_1...X_N)) \simeq NH(x)$

- On sait que $c(\mathbf{x}) \leq \mu(\mathbf{x})$ et on veut montrer que :

$$\lambda(x) - \lambda(\mathbf{x}) \leq \frac{h_N - l_N}{2}$$

On considère la représentation binaire de $\lambda(\mathbf{x})$:

$$\tilde{\lambda} = \sum_{i=1}^{n} a_i 2^{-i}$$

Pour $\tilde{\lambda}(\mathbf{x})$ on a :

$$\tilde{\lambda} = \sum_{i=1}^{\mu(\mathbf{x})+1} a_i 2^{-i}$$

D'où :

$$\begin{aligned}\lambda(\mathbf{x}) - \tilde{\lambda}(\mathbf{x}) &= \sum_{i=\mu(\mathbf{x})+2}^{\infty} a_i 2^{-i} \\ &\leq \sum 2^{-i} \\ &\leq 2^{-\mu(\mathbf{x}+1)}\end{aligned}$$

Or

$$\begin{array}{ccccc} 2^{-\mu(\mathbf{x})+1} & = & 2^{-(\lceil -\log_2(h_n-l_n)\rceil+1)} & & \\ 1-\log(h_n-l_n) & \leq & \lceil -\log_2(h_n-l_n)\rceil+1 & \leq & -\log_2(h_n-l_n)+1+1 \\ -2+\log(h_n-l_n) & < & \lceil -\log_2(.)\rceil+1 & \leq & -1+\log_2(.) \\ \frac{h_n-l_n}{4} & \leq & 2- & \leq & \frac{h_n-l_n}{2} \end{array}$$

- L'efficacité du codage est montrée en calculant la longueur moyenne du code obtenu :

$$\bar{l} = \sum_{\mathbf{x}\in\mathcal{X}^N} p(\mathbf{x})\lambda(\mathbf{x})$$

En utilisant le premier encadrement de $\lambda(\mathbf{x})$, alors

$$\begin{array}{ccccc} -\sum_{\mathbf{x}} p(\mathbf{x})(\log_2(p(\mathbf{x}))-1) & \leq & \bar{l} & < & -\sum_{\mathbf{x}} p(\mathbf{x})(\log_2(p(\mathbf{x}))-2) \\ NH(X)+1 & \leq & \bar{l} & < & NH(X)+2 \end{array}$$

Cette procédure est efficace pour $N \to +\infty$, avec l'∞ petit. Par exemple, pour $N = 100$, alors la longueur moyenne sera proche à 1% de l'entropie.

3.3 Code de Lempel-Ziv-Welch

Idée Avant l'algorithme arithmétique adaptatif, il fallait avoir les probabilités d'apparition des symboles (Huffmann, codage arithmétique,...). Le code LZW permet de faire une compression sans passer par la phase d'estimation des probabilités des symboles. Par exemple pour un texte, l'alphabet est gros, et le codeur arithmétique ne fonctionne pas très bien pour les gros alphabets, car il utilise d'abord une phase de binarisation.

3.3.1 Codage

Le codage de LZW est une variante des codes LZ77 et LZ78 qui utilise un dictionnaire et n'a pas besoin des probabilités de la source. C'est un cde universel.

On considère une source X à valeurs dans $\mathcal{X} = \{a, b, c\}$.

1. On initialise le dictionnaire de codage à l'aide des symboles de l'alphabet $\mathcal{X}$.
2. On cherche la plus longue suite de symboles restant à coder et appartenant au dictionnaire et on la code.
3. On rajoute cette suite suivi du premier symbole non codé ω au dictionnaire.
4. Aller en 2 tant qu'il existe des symboles à coder.

Exemple: On a à coder la séquence *aababababca*.

a	*b*	*c*	*aa*	*ab*	*ba*	*aba*	*abac*
0	1	2	3	4	5	6	7

On code *a*, on émet 0 et on ajoute *aa* au dictionnaire.
On code *a*, on émet 0 et on ajoute *ab* au dictionnaire.
On code *b*, on émet 1 et on ajoute *ba* au dictionnaire.
On code *ab*, on émet 4 et on ajoute *aba* au dictionnaire.
On code *aba*, on émet 6 et on ajoute *abac* au dictionnaire.
On code *c*, on émet 2.

3.3.2 Décodage

Le décodage se fait

1. à partir du dictionnaire initial
2. à chaque décodage d'un mot, on rajoute ce mot suivi de ω au dictionnaire
3. ω n'est déterminé que si on a décodé le mot suivant

Exemple: On a à décoder 001462.

On décode a, on ajoute $a\omega$ (3) au dictionnaire.
On décode a, donc le (3) est aa et on ajoute $a\omega$ (4) au dictionnaire.
On décode b, donc le (4) était ab et on ajoute $b\omega$ (5) au dictionnaire.
On décode ab, donc le (5) était ba et on ajoute $ab\omega$ (6) au dictionnaire.
On décode $ab\omega$, qui était en fait un aba et on ajoute $aba\omega$ (7) au dictionnaire.
On décode c donc le (7) était $abac$.

En pratique :

- les codes générés sont à nouveau codés à l'aide d'un code de Huffmann
- la taille du dictionnaire est limitée, les mots les moins utilisés sont effacés.

Chapitre 3

Quantification

1 Introduction

Définition

- La *quantification* est une génération non réversible qui permet à une source X à valeur dans $\mathcal{X}$ d'associer un index $I \in \{0, ...M-1\}$ ou M est le nombre de cellule de quantification. La
- *quantification inverse* consiste à associer une estimée $\hat{X}$ de X à valeur dans $\mathcal{X}$ à un index I.
- Pour cela on introduit une fonction de quantification :

  $$q : \mathcal{X} \to \{0, ..., M-1\}$$

  et des fonction de quantification inverse:

  $$r : \{0, ..., M-1\} \to \mathcal{X}$$

- Pour une réalisation x de $\mathcal{X}$,$i = q(x)$ est l'index de quantification.

  $$\hat{x} = r(q(x)) = Q(x)$$

  est la valeur reconstruite/quantifiée.

Remarque: Si $\mathcal{X} = \mathbb{N}$ ou $\mathbb{Z}$ ou $\mathbb{R}$ on réalise une quantification vectorielle. Si $\mathcal{X} = \mathbb{R}^n$ on réalise une quantification vectorielle.

Les quantifications sont optimisées de manière à obtenir un débit minimale sous contrainte de distortion ou bien une distortion minimale sous contrainte de débit. La distorsion étant une mesure de l'erreur entre X et $\hat{X}$.

Un quantificateur est défini par des intervalles de quantification $b_0 < b_1 < ... < b_n$ et par des valeurs de reconstitution $y_1, ..., y_n$: si on a $x \in [b_{i-1}; b_i[$ alors $Q(x) = y_i$. Dans la suite, on va voir comment régler de façon efficace les b_i et les y_i.

2 Distorsion et mesure de distorsion

Définition

On introduit une mesure de distorsion pour les performances d'un quantificateur qui toute les bonnes propriétés d'une distance symétrique, définie positive). Les principales mesures considérées sont:

- la mesure de distorsion en valeur absolue, $d(x, y) = |x - y|$
- la mesure de distorsion quadratique, $d(x, y) = (x - y)^2$
- le mesure de distorsion de Hamming, $d(x, y) = \begin{cases} 0 & \text{si } x = y \\ 1 & \text{sinon} \end{cases}$

Pour une source X décrite par une distribution de probabilité $f_X(x)$, la distorsion introduite par un quantificateur $Q(x)$, est la moyenne de la mesure de la distorsion :

$$D = \int_{-\infty}^{\infty} d(x, Q(x)) f_X(x) dx = E(d(X, Y))$$

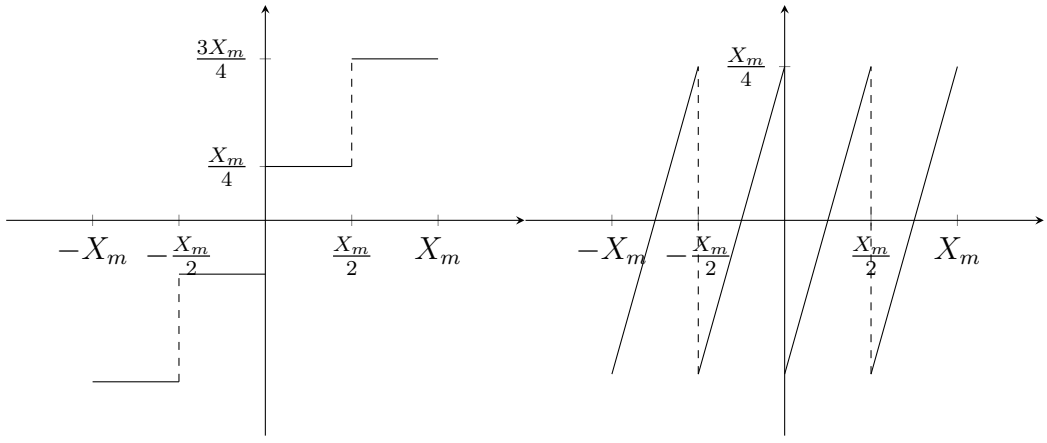
Pour la mesure de distorsion quadratique, on a :

$$D = \int_{-\infty}^{\infty} (x - Q(x))^2 f_X(x) dx$$

3 Quantification scalaire

3.1 Quantification uniforme d'une source uniforme

On considère une source X uniformément distribuée sur $[-X_{max}; X_{max}]$. On considère aussi un quantificateur uniforme, c'est à dire que les intervalles de quantification sont tous de même taille, à M niveaux de sortie, situés au milieux des intervalles de quantification. Prenons par exemple, un quantificateur à 4 niveaux de quantification :



On note Δ l'intervalle de quantification, dans ce cas, $\Delta = \frac{2X_{max}}{M}$.

Pour déterminer la distorsion qui va être introduite par le quantificateur, on calcule simplement :

$$D = \int_{-X_{max}}^{X_{max}} \frac{1}{2X_{max}}(x - Q(x))^2 dx$$

Comme $Q(x)$ est connu et constant sur un intervalle de quantification, on découpe simplement l'intervalle en sous-intervalles de quantification :

$$D = \sum_{i=0}^{(M-1)} \int_{i\Delta}^{(i+1)\Delta} \frac{1}{2X_{max}}(x - (i + \frac{1}{2})\Delta)^2 dx$$

On calcule

$$I = \int_{i\Delta}^{(i+1)\Delta} (x - (i + \frac{1}{2})\Delta)^2 \frac{1}{2X_{max}} dx$$

on pose $u = x + X_{max} - (x + \frac{1}{2})\Delta)$

$$= \int_{-\Delta/2}^{\Delta/2} \frac{u^2}{2X_{max}} du$$

$$I = \frac{\Delta^3}{24X_{max}} = \frac{X_{max}^2}{3M^3}$$

Dans D, on a M intégrales égales à I donc

$$D = \frac{X_{max}^2}{3M^2}$$

L'énergie de la source est mesurée par sa variance

$$\sigma^2 = \int_{-\infty}^{+\infty} x^2 f(x) dx$$

$$= \int_{-X_{max}}^{X_{max}} \frac{x^2}{2X_{max}} dx$$

$$\sigma^2 = \frac{X_{max}^2}{3}$$

On obtient donc $D = \frac{\sigma^2}{M^2}$.

Proposition

Sans codage entropique, le nombre de bits nécessaires pour représenter un niveau de reconstruction est $R = \lceil \log_2 M \rceil$, d'où

$$\boxed{D = \sigma^2 2^{-2R}}$$

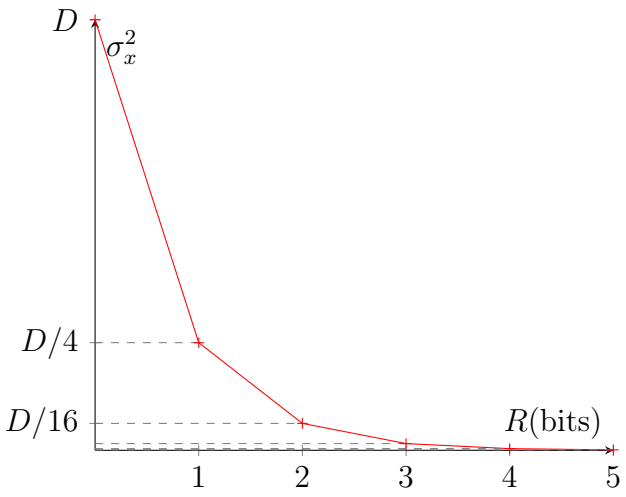
La distorsion maximale est égale à l'énergie de la source, et diminue très rapidement quand on augmente le nombre de bits du quantificateur.

Remarque: Dans certains cas (source non uniforme), un codage entropique peut permettre de réduire le débit.

Rapport signal à bruit :

$$RSB = \frac{\sigma^2}{D} = 2^{2R}$$
$$RSB_{dB} = 10\log_{10}(2^{2R}) = 6.02R\text{decibel}$$

Le RSB est utilisé comme mesure de qualité en audio, image...



La pente a l'origine est de $-2\ln(2)\sigma_x^2$. La distorsion décroit très vite avec R

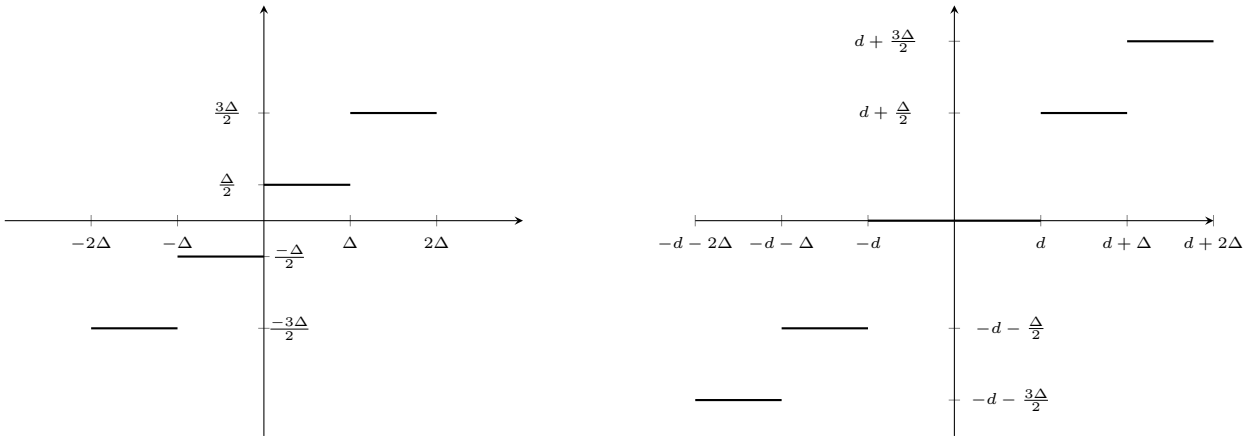
On peux aussi tracer la courbe distortion- débit (R= f(D))

3.2 Quantification uniforme d'une source quelconque

On considère une source X décrite par sa ddp $f_X(x)$, quantifiée par un quantificateur uniforme à M niveaux de pas Δ. Si M est fini on peux considérer deux type de quantificateur :

Exercice

1. Générer N réalisation d'une source Gaussienne $\mathcal{N}(0,1)$.
2. Implanter un quantificateur uniforme sans zone morte et la fonction de reconstruction associée.
3. Faire de Meme pour un quantificateur avec zone morte.
4. Tracer dans les deux cas la courbe débit distorsion en supposant que les index de quantification sont codées à l'aide d'un codeur entropique.



(a) Quantificateur sans zone morte (b) Quantificateur avec zone morte

On considère une source X discrète décrite par une ddp $f_X(x)$. On cherche le quantificateur non-uniforme à M niveaux de sortie qui minimise la distorsion de quantification pour une norme de distorsion quadratique.

On cherche à minimiser la distorsion

$$D = \int_{-\infty}^{+\infty} (x - Q(x))^2 f_X(x) dx$$

$$D = \underbrace{\int_{-\infty}^{(-M/2+1)\Delta} (x - Q(x))^2 f_X(x) dx}_{\text{distorsion de surcharge}} + \underbrace{\sum_{i=1}^{M-2} \int_{-\frac{M}{2}\Delta+i\Delta}^{-\frac{M}{2}\Delta+(i+1)\Delta} (x - Q(x))^2 f_X(x) dx}_{\text{distorsion de granularité}} + \underbrace{\int_{(\frac{M}{2}-1)\Delta}^{+\infty} (x - Q(x))^2 f_X(x) dx}_{\text{distorsion de surcharge}}$$

On peux bornée l'erreur de quantification au centre entre $\frac{-\Delta}{2}$ et $\frac{\Delta}{2}$à l'extérieur l'erreur n'est pas bornée : Pour M fixé on a :

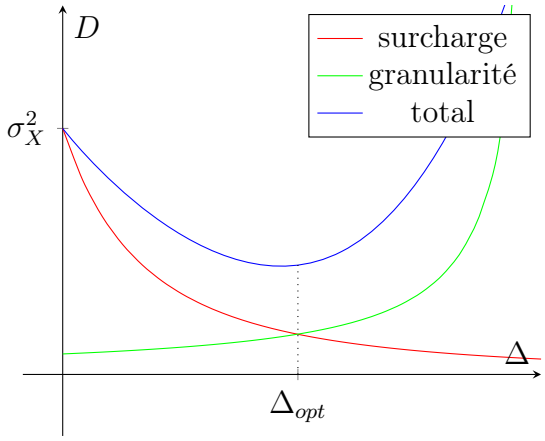


FIGURE 3.2 – Evolution de la distorsion en fonction du pas de quantification

3.3 Quantification non uniforme

On considère une source X à valeur réelle $\mathcal{X} = \mathbb{R}$ décrite par une ddp $f_X(x)$. On chercje à quantifier cette source à l'aide d'un quantificateur non uniforme sur M niveau décrit par :

- $M+1$ bornes des intervalles de quantification $b_0 = \infty < b_1 < ... < b_M = +\infty$
- M valeurs de reconstruction $y_1 ... y_M$.

On choisit les paramètres de quantificateur de manière à minimiser une distorsion construite à partir d'une mesure quadratique.

Les conditions nécessaires pour avoir D minimale sont :

- $\frac{\partial D}{\partial y_i} = 0, \forall i = 1 \dots M$
- $\frac{\partial D}{\partial b_i} = 0, \forall i = 0 \dots M$

1ère condition d'optimalité

$$\begin{aligned}\frac{\partial D}{\partial y_i} &= \frac{\partial}{\partial} y_i \int_{b_{i-1}}^{b_i} (x - y_i)^2 f_X(x) dx \\ &= -2 \int_{b_{i-1}}^{b_i} (x - y_i) f_X(x) dx\end{aligned}$$

Ainsi,

$$\begin{aligned}\frac{\partial D}{\partial y_i} = 0 \Leftrightarrow & \int_{b_{i-1}}^{b_i} x f_X(x) dx = y_i \int_{b_{i-1}}^{b_i} f_X(x) dx \\ \Rightarrow & \boxed{y_i = \frac{\int_{b_{i-1}}^{b_i} x f_X(x) dx}{\int_{b_{i-1}}^{b_i} f_X(x) dx}, \quad i = 1 \dots M}\end{aligned}$$

2ème condition d'optimalité

$$\begin{aligned}\frac{\partial D}{\partial b_i} &= \frac{\partial}{\partial b_i} \int_{b_i}^{b_{i+1}} (x - y_{i+1})^2 f_X(x) dx + \frac{\partial}{\partial b_i} \int_{b_{i-1}}^{b_i} (x - y_i)^2 f_X(x) dx \\ &= -(b_i - y_{i+1})^2 f_X(b_i) + (b_i - y_i)^2 f_X(b_i)\end{aligned}$$

Ainsi,

$$\begin{aligned}\frac{\partial D}{\partial b_i} = 0 &\Leftrightarrow -(b_i - y_{i+1})^2 + (b_i - y_i)^2 = 0 \\ &\Leftrightarrow (y_{i+1} - y_i)(b_i - y_{i+1} + b_i - y_i) = 0 \\ &\Leftrightarrow \boxed{b_i = \frac{y_i + y_{i+1}}{2}, \quad i = 1 \dots M-1}\end{aligned}$$

Proposition

Les bornes des intervalles de quantification sont au milieu de deux valeurs de reconstructions consécutives.

$$b_i = \frac{y_i + y_{i+1}}{2}, \quad i = 1 \dots M-1$$

et les valeurs de reconstructions optimales vérifient :

$$\hat{y_i} = E[\{X|X \in [b_i, b_{i+1}]\}] = \frac{\int_{b_{i-1}}^{b_i} x f_X(x) dx}{\int_{b_{i-1}}^{b_i} f_X(x) dx}$$

3.3.1 Algorithme de Lloyd -Max

1. Initialisation : $b_0^{(0)} < b_1^{(0)} < \dots < b_n^{(0)}$ choisis arbitrairement, $k = 1$.
2. $y_i^{(k)}$ obtenus à partir des $b_i^{(k-1)}, i = 0 \dots M$ en utilisant la 1ère condition d'optimalité
3. $b_i^{(k)}$ obtenus à partir des $y_i^{(k)}, i = 1 \dots M$ en utilisant la 2ème condition d'optimalité
4. $k = k + 1$
5. tant qu'il y a des variations des y_i ou des b_i, aller en 2.

Remarque: Essayer d'implémenter l'algorithme de Lloyd-Max à l'aide de Matlab, C ou Python pour les fétichistes. Pour Matlab, utiliser la fonction `quad` afin de calculer les intégrales. Prendre une ddp gaussienne.

Essayer de retrouver ceci pour $M = 4$ et $\sigma = 1$

i	b_i	y_i
1	-0.98	-1.51
2	0	-0.45
3	0.98	0.45
4	$+\infty$	1.51

Prendre l'infini égal à 10.

Remarque: On ne quantifie jamais sur le domaine des pixels, car les ddp y sont immondes. On effectue des transformées, et ensuite les ddp sont sympa (gaussiennes, laplaciennes), ce qui permet d'avoir des algorithmes efficaces.

La plupart du temps, les quantifications sont uniformes (JPEG, JPEG200, H264...). On n'utilise des quantifications non uniformes que dans le cas d'applications très précises, où le gain de 2 ou 3 dB sur le RSB est vraiment nécessaire.

3.3.2 Comportement asymptotique

Pour étudier le comportement asymptotique d'un quantificateur non uniforme, on suppose M grand (les intervalles de quantification seront petits), $f_X(x) \approx f_X(y_i)$ sur $[b_{i-1}, b_i]$. On note $\Delta_i = b_i - b_{i-1}$.

On a

$$P_i = Pr(X \in [b_{i-1}, b_i]) \approx f_X(y_i)\Delta_i$$

$$D = \sum_{i=1}^{M} \int_{b_{i-1}}^{b_i} (x - y_i)^2 f_X(x) dx$$

$$D = \sum_{i=1}^{M} f_X(y_i) \int_{b_{i-1}}^{b_i} (x - y_i)^2 dx$$

On suppose que y_i est au milieu de l'intervalle $[b_{i-1}, b_i]$

$$\int_{b_{i-1}}^{b_i} (x - y_i)^2 dx = \int_{-\Delta_i/2}^{\Delta_i/2} u^2 du = [\frac{u^3}{3}]_{-\Delta_i/2}^{\Delta_i/2} = \frac{2}{3}\frac{\Delta_i^2}{8} = \frac{\Delta_i^3}{12}$$

En réinjectant dans D, on a

$$D = \sum_{i=1}^{M} f_X(y_i)\frac{\Delta_i^3}{12}$$

On pose $\alpha_i^3 = f_X(y_i)\Delta_i^3$

$$D = \frac{1}{12}\sum_{i=1}^{M} \alpha_i^3$$

Si on calcule

$$\sum_{i=1}^{M} \alpha_i = \sum_{i=1}^{M} (f_X(y_i))^{1/3}\Delta_i \approx \int_{-\infty}^{+\infty} (f_X(x))^{1/3} dx = cste$$

On doit trouver les Δ_i et les α_i qui minimisent D sous la contrainte $\sum_{i=1}^{M} \alpha_i = C$. On introduit donc le Lagrangien du problème :

$$L(\alpha_1, \dots \alpha_M, \lambda) = \frac{1}{12}\sum_{i=1}^{M} \alpha_i^3 + \lambda \left(\sum_{i=1}^{M} \alpha_i - C \right)$$

Condition d'optimalité :

$$\frac{\partial L}{\partial \alpha_i} = 0 \Rightarrow \frac{1}{4}\alpha_i^2 + \lambda = 0, \quad i = 1 \dots M$$
$$\Rightarrow \alpha_i^2 = -4\lambda$$

Les α_i sont donc tous égaux, d'où

$$\alpha_i = \frac{1}{M}\int_{-\infty}^{+\infty} (f_X(x))^{1/3} dx$$

On avait $\alpha_i^3 = f_X(y_i)\Delta_i^3$. Ainsi, si $f_X(y_i)$ est grand, Δ_i est petit, et inversement.

On peut donc calculer la distorsion :

$$D = \frac{1}{12}\sum_{i=1}^{M} \frac{1}{M^3}\left(\int_{-\infty}^{+\infty} (f_X(x))^{1/3} dx\right)^3$$
$$= \frac{1}{12}\left(\int_{-\infty}^{+\infty} (f_X(x))^{1/3} dx\right)^3 \cdot \frac{1}{M^2}$$

Or, $M = 2^R$ d'où

$$D = \frac{1}{12}\left(\int_{-\infty}^{+\infty} (f_X(x))^{1/3} dx\right)^3 .2^{-2R}$$

Proposition

Comme pour la quantification uniforme, on a une décroissance en 2^{-2R} de la distorsion en fonction du débit. Ici, il y a un facteur multiplicatif qui dépend de la ddp de la source.
Pour une source quelconque de variance σ_X^2 le comportement débit / distorsion sera :

$$\boxed{D(R) = K\sigma_X^2 2^{-2R} \text{ avec } K = \frac{1}{12\sigma_X^2}\left(\int_{-\infty}^{+\infty} (f_X(x))^{1/3} dx\right)^3}$$

Soit un rapport signal sur bruit :

$$RSB = \frac{\sigma_X^2}{D} = K.2^{2R}$$

Remarque: Pour une source Gaussienne centrée on peux calculer K et on a :

$$D = \sigma^2 2^{-2R}$$

4 Quantification vectorielle

Définition

Un quantificateur vectoriel pour une source $\mathbf{X} \in \mathbb{R}^n$ est défini:

- par une fonction de quantification :

$$q : \mathbb{R}^n \to \{0, ..., n-1\}$$

- par une fonction de reconstruction:

$$q^{-1} : \{0, ..., n-1\} \to \mathbb{R}^n$$

On note $\mathbf{y_m} = q^{-1}(m)$ les valeurs de reconstruction du quantificateur.

On étend la notion de distorsion et de mesure de distorsion au cas vectoriel :

Proposition ***(Mesure de distortion vectorielle)***

Mesure de distortion quadratique :

$$d_2(\mathbf{x}, \mathbf{y}) = \frac{1}{n} \sum_{i=1}^{n} |x_i - y_i|^2$$

Mesure de distortion en valeur absolue :

$$d_1(\mathbf{x}, \mathbf{y}) = \frac{1}{n} \sum_{i=1}^{n} |x_i - y_i|$$

La distorsion introduite par le quantificateur Q est alors l'espérance de la mesure de distorsion :

$$D = E[d(\mathbf{x}, Q(\mathbf{x}))]$$

4.1 Condition d'optimalité

On considère une quantification vectorielle d'une source $\mathbf{X} \in \mathbb{R}^n$ sur M niveau de quantification. On alors la distorsion :

$$D = \int_{\mathbb{R}}^{n} \|\mathbf{x} - Q(\mathbf{x})\|_2^2 f_{\mathbf{x}}(\mathbf{x}) \mathrm{d}\mathbf{x}$$

On partitionne $\mathbb{R}^n$ en cellule de quantification $\mathcal{C}^k, k \in [0...M-1]$ au seins desquelles la valeur de reconstructio vaut $\mathbf{y_k}$.

Comment choisir $\mathcal{C}^k$ et y_k pour minimiser $\mathbf{y_k}$?

$$D = \sum_{k=0}^{M-1} \int_{\mathcal{C}^k} \|\mathbf{x} - \mathbf{y_k}\|^2 f_{\mathbf{X}}(\mathbf{x}) \mathrm{d}\mathbf{x}$$

Proposition

Si on fixe les $\mathcal{C}^k$:

$$\mathbf{y_k} = \frac{\int_{\mathcal{C}^k} \mathbf{x} f_{\mathbf{X}}(\mathbf{x}) \mathrm{d}\mathbf{x}}{\int_{\mathcal{C}^k} f_{\mathbf{X}}(\mathbf{x}) \mathrm{d}\mathbf{x}}$$

$\mathbf{y_k}$ est le barycentre de $\mathcal{C}^k$ en utilisant la densité$f_{\mathbf{X}}(\mathbf{x})$.
Si on fixe les $\mathbf{y_k}$ alors pour minimiser D il faut prendre :

$$\mathcal{C}^k = \{\mathbf{x} \in \mathbb{R}^n, \quad \|\mathbf{x} - \mathbf{y_k}\| \leq \|\mathbf{x} - \mathbf{y_l}\|, \forall l \neq k\}$$

Démonstration : Si on suppose $\mathcal{C}^k$fixé alors pour fixer y_k on calcule :

$$\frac{\partial D}{\partial \mathbf{y_k}} = -2 \int_{\mathcal{C}^k} (\mathbf{x} - \mathbf{y_k}) f_{\mathbf{X}}(x) \mathrm{d}\mathbf{x}$$

On en déduit :

$$\mathbf{y_k} = \frac{\int_{\mathcal{C}^k} \mathbf{x} f_{\mathbf{X}}(\mathbf{x}) \mathrm{d}\mathbf{x}}{\int_{\mathcal{C}^k} f_{\mathbf{X}}(\mathbf{x}) \mathrm{d}\mathbf{x}}$$

■

Pour un couple (poids, taille), on peut :

- quantifier indépendamment le poids et la taille avec deux quantificateurs scalaires
- les quantifier simultanément avec une quantification vectorielle

4.2 Algorithme de Linde-Buzo-Gray

Similaire à l'algorithme K-means

Cet algorithme permet de fabriquer une quantification vectorielle optimale (optimum local) pour un ensemble de K points de $\mathbb{R}^N$: $\underline{x}_1 \dots \underline{x}_K$ et quantifiés sur M niveaux.

Pour cela, on introduit la mesure de distorsion $d(\underline{x}, \underline{y}) = \frac{1}{N}||\underline{x} - \underline{y}||^2$.

Le quantificateur va minimiser la distorsion.

Initialisation :

- On sélectionne M points $\underline{y}_1^{(0)}, \dots, y_M^{(0)}$ parmi les $\underline{x}_1, \dots, \underline{x}_K$ au hasard.
- $l = 0, D_l = +\infty$

1. On réalise une quantification de $\underline{x}_1, \dots, \underline{x}_K$ en se servant de $\underline{y}_1^{(l)}, \dots, \underline{y}_M^{(l)}$.

$$q(\underline{x}_i) = \underline{y}_j \text{ ssi } ||\underline{x}_i - \underline{y}_j^{(l)}||^2 \leq ||\underline{x}_i - \underline{y}_k^{(l)}||^2, \quad \forall k \neq j$$

   On obtient une partition de l'espace en cellules de quantification $\mathcal{C}_j^{(l)}, j = 1, \dots, M$ appelées cellules de Voronoï.

2. On calcule la distorsion :

$$D^{(l+1)} = \frac{1}{K} \sum_{i=1}^{K} ||\underline{x}_i - q(\underline{x}_i)||^2$$

3. Actualisation des valeurs de reconstruction. On considère l'ensemble des points appartenant à
$$\mathcal{C}_j^{(l)} = \{\underline{x}_1^{(l)}, \ldots, \underline{x}_{K_l}^{(l)}\}, \quad j = 1, \ldots, M$$
On calcule le barycentre des points de $\mathcal{C}_j^{(l)}$ qui constitue la nouvelle valeur de reconstruction.
$$\underline{y}_j^{(l+1)} = \frac{1}{K_l} \sum_{i=1}^{K_l} \underline{x}_i^{(l)}$$
4. $l = l + 1$
5. Aller en 1 tant que $D^{l-1} - D^l > \epsilon$

L'algorithme LBG converge vers une minimum local de la distorsion.
En pratique :

- on transmet pour chaque $\underline{x}_i$ l'index j de la cellule de Voronoi $\mathcal{C}_j$ auquel il appartient.
- il faut transmettre au récepteur l'ensemble des points de reconstruction $\underline{y}_1^{(\bar{l})}, \ldots, \underline{y}_M^{(\bar{l})}$ où $\bar{l}$ est l'index final des itérations.
- la phase de réglage du quantificateur peut se faire sur seulement $L << K$ points si on a beaucoup de points.

5 Quantification scalable

non traité en 2018-2019 L'objectif est de permettre une flexibilité en terme de compression débit-distorsion. Une possibilité est d'avoir recours à un quantificateur scalable.

On considère une source uniforme sur $[-X_{max}; X_{max}]$ quantifiée sur 8 niveaux.
Si on a K échantillons à quantifier, on obtient $3K$ bits.

Les K bits de poids fort sont placés dans un premier paquet.
Les K bits de poids intermédiaire sont placés dans un second paquet.
Les K bits de poids faible sont placés dans un troisième paquet.

Des utilisateurs disposant d'un mauvais canal recevront le premier paquet : la distorsion sera élevée.

Des utilisateurs disposant d'un excellent canal recevront les 3 paquets et auront une distorsion faible.

Remarque: Cette méthode permet de réaliser une quantification sans avoir à connaître l'état du canal.

Chapitre 4

Codage prédictif

L'idée du codage prédictif est d'utiliser les corrélations (ressemblances) temporelles ou spatiales du signal à compresser.

Rappels sur la corrélation On considère une source X qui émet un signal constitué de $x_1, \ldots, x_N$ considérés comme une réalisation d'une suite de variables aléatoires $X_1, \ldots, X_N$ de moyenne nulle.

La fonction de corrélation permet de mesurer la ressemblance entre échantillons voisins :

$$\gamma_x(n,k) = E(X_n X_{n+k})$$

Pour un signal stationnaire (dont les caractéristiques statistiques n'évoluent pas au cours du temps :

$$\gamma_x(n,k) = \gamma_x(k) = E(X_n X_{n+k}), \forall n$$

En pratique, on estime la corrélation à partir des échantillons du signal à compresser.

- Estimateur biaisé :

$$\hat{\gamma_x^B}(k) = \begin{cases} \dfrac{1}{N}\displaystyle\sum_{i=1}^{N-k} x_i x_{i+k}, \forall k \geq 0 \\ \dfrac{1}{N}\displaystyle\sum_{i=-k}^{N} x_i x_{i+k}, \forall k \leq 0 \end{cases}$$

- Estimateur non biaisé

$$\hat{\gamma_x^{NB}}(k) = \frac{1}{N-|k|}\sum_{i=1}^{N-k} x_i x_{i+k} \forall k \geq 0$$

Remarque: On préfère l'estimateur biaisé car il est plus stable numériquement.

Avec Matlab, on l'obtient avec :

```
[c,k] = xcorr(x,'biased');
plot(k,c); grid;
```

$\gamma_x(k)$ est maximale en 0 et est égale à l'énergie σ^2 du signal.

1 Codage prédictif en boucle ouverte

Schéma en boucle ouverte :

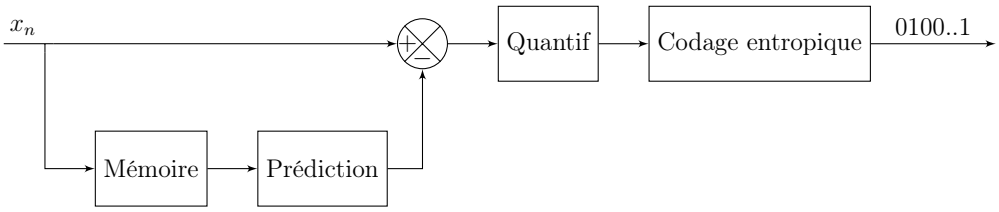


FIGURE 4.1 – Emetteur en boucle ouverte

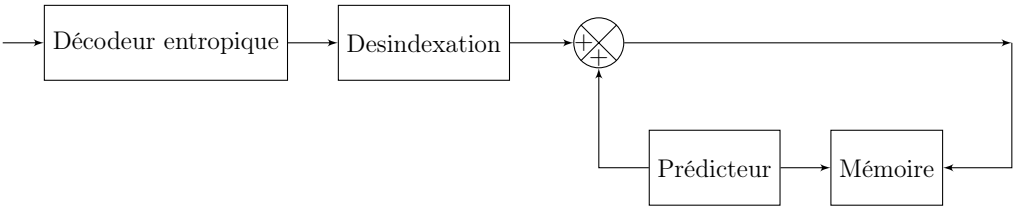


FIGURE 4.2 – Recepteur en boucle ouverte

Ca marche mais la quantification introduit des erreurs qui peuvent s'accumuler. On s'en sort en réinitialisant le codeur régulièrement.

1.1 Prédicteur linéaire optimal à 1 pas

On souhaite prédire la valeur de X_n à partir de la valeur de X_{n-1}.
Le prédicteur sera linéaire :

$$\hat{X_n} = a_1 X_{n-1}$$

On cherche la valeur de a_1 qui minimise $e = E((X_n - \hat{X_n})^2)$

$$\begin{aligned} e &= E((X_n - a_1 X_{n-1})^2) \\ &= E(X_n^2 - a_1^2 X_{n-1}^2 - 2a_1 X_{n-1} X_n) \\ &= E(X_n^2) + a_1^2 E(X_{n-1}^2) - 2a_1 E(X_{n-1} X_n) \\ e &= \gamma_x(0) + a_1^2 \gamma_x(0) - 2a_1 \gamma_x(1) \text{ par stationnarité} \end{aligned}$$

$$\frac{\partial e}{\partial a_1}|_{\hat{a_1}} = 0 \Leftrightarrow 2\hat{a_1}\gamma_x(0) - 2\gamma_x(1) = 0$$

$$\Rightarrow \hat{a_1} = \frac{\gamma_x(1)}{\gamma_x(0)}$$

Remarque: Lorsque le signal sera très corrélé, $\gamma_x(1) \approx \gamma_x(0)$ et $\hat{a_1} \approx 1$. Pour un signal peu corrélé, $\gamma_x(1) \approx 0$ et $\hat{a_1} \approx 0$.

Pour la valeur de $\hat{a_1}$ obtenue, on a

$$\begin{aligned}\hat{e} &= \gamma_x(0) + (\frac{\gamma_x(1)}{\gamma_x(0)})^2\gamma_x(0) - 2\frac{\gamma_x(1)^2}{\gamma_x(0)} \\ &= \frac{\gamma_x(0)^2 - \gamma_x(1)^2}{\gamma_x(0)} \leq \gamma_x(0)\end{aligned}$$

$\hat{e}$ est l'énergie de la partie qui n'a pas pu être prédite de $x_1, \ldots, x_N$.

Lorsque le signal est fortement corrélé, $\gamma_x(1) \simeq \gamma_x(0)$ et $\hat{a_1} \simeq 1$.
Le résidu de prédiction a une variance plus faible. Si on le quantifie, il permettra de reconstituer le signal initial avec une distorsion plus faible.

1.2 Prédiction à p pas

On cherche à prédire $\mathbf{X_n}$ à partir des échantillons précédents $X_{n-1}, \ldots, X_{n-p}$.

$\hat{X_n} = a_1X_{n-1} + \cdots + a_pX_{n-p} = \mathbf{a}^T\mathbf{X_n}$ avec $\mathbf{a}^T = (a_1 \ldots a_p)$ et $\mathbf{X_n}^T = (X_{n-1} \ldots X_{n-p})$

On cherche $\mathbf{a}$ minimisant

$$\begin{aligned}e &= E((X_n - \hat{X_n})^2) \\ &= E((X_n - \mathbf{a}^T\mathbf{X_n})^2) \\ &= E(X_n^2) + \mathbf{a}^T E(\mathbf{X_n}\mathbf{X_n}^T)\mathbf{a} - 2\mathbf{a}^t E(X_n\mathbf{X_n}) \\ \text{Or, } E(X_n\mathbf{X_n}) &= (E(X_nX_{n-1}), \ldots, E(X_nX_{n-p}))^T \\ &= (\gamma_x(1), \gamma_x(2), \ldots, \gamma_x(p))^T = \mathbf{c} \\ \text{De plus, } E(\mathbf{X_n}\mathbf{X_n}^T) &= \begin{bmatrix} E(X_{n-1}X_{n-1}) & E(X_{n-1}X_{n-2}) & \ldots & E(X_{n-1}X_{n-p}) \\ E(X_{n-2}X_{n-1}) & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ E(X_{n-p}X_{n-1}) & \ldots & \ldots & E(X_{n-p}X_{n-p}) \end{bmatrix} \\ &= \begin{bmatrix} \gamma_x(0) & \gamma_x(1) & \ldots & \gamma_x(p-1) \\ \gamma_x(1) & \gamma_x(0) & & \vdots \\ \vdots & & \ddots & \gamma_x(1) \\ \gamma_x(p-1) & \ldots & \gamma_x(1) & \gamma_x(0) \end{bmatrix} = \mathbf{R}\end{aligned}$$

donc $e = \gamma_x(0) + \mathbf{a}^T\mathbf{R}\mathbf{a} - 2\mathbf{a}^T\mathbf{c}$

$$\left.\frac{\partial e}{\partial \mathbf{a}}\right|_{\hat{\mathbf{a}}} = 0 \quad \Leftrightarrow \quad \mathbf{0} + 2\mathbf{R}\hat{\mathbf{a}} - 2\mathbf{c} = 0 \quad \Rightarrow \quad \hat{\mathbf{a}} = \mathbf{R}^{-1}\mathbf{c}$$

Pour cette valeur de $\hat{\mathbf{a}}$, on a

$$\begin{aligned}\hat{e} &= \gamma_x(0) + \mathbf{c}^T\mathbf{R}^{-1}\mathbf{R}\mathbf{R}^{-1}\mathbf{c} - 2\mathbf{c}^T\mathbf{R}^{-1}\mathbf{c} \\ &= \gamma_x(0) - \mathbf{c}^T\mathbf{R}^{-1}\mathbf{c} \leq \gamma_x(0)\end{aligned}$$

Ce prédicteur à p pas est en général plus efficace que le prédicteur à 1 pas mais il est plus complexe.

1.3 Mise en oeuvre du prédicteur

Il faut que le récepteur disposent également des coefficients de prédiction optimal. Il peuvent :

- être transmis mais cela coute du débit
- Etre réestimés au récepteur mais cela rend le récepteur plus complexe.

Pour le réglage du prédicteur, on distingue plusieurs méthodes :

1.3.1 Fenêtres fixes

On découpe le signal en blocs de M échantillons et on recalcule le prédicteur sur chaque bloc.

Les échantillons de la m-ième fenêtres (pour$(m-1)M+1$ à mM) :

- servent à extimer $\hat{\gamma}_x(k)$ et les coefficients de prédiction utilisé pour la $m+1$-ème fenêtre.
- sont comprimés en utilisant le prédicteur optimal calculé das la fenêtre $m-1$.
- Le recepteur fait les mêmes opérations.

Remarque: Pour la première fenêtre on utilise le prédicteur à 1 pas.

Cette méthode ne fonctionne que pour des compressions sans pertes ou a débit élevé.

Avantages :

- sa simplicité de mise ne œuvre
- permet de s'adapter aux non-stationnarités

Inconvénient :

- débit nécessaire à la transmission des $\hat{\mathbf{a}}_{opt}$.

1.3.2 Fenêtres glissantes

On travaille sur une fenêtre glissante contenant les N échantillons décalés : $\hat{\mathbf{X}}_n = (\hat{x}_{n-1}, ..., \hat{x}_{n-N})^T$.

- On estime $\gamma_x(k)$ à partir des échantillons.
- On en déduit le prédicteur à p pas.
- On prédi x_n et on compresse le résidu.
- Au récepteur dans le cas c'un codage sans pertes on dispose également de $x_{n-N}...x_{n-1}$. et on va pouvoir faire les mêmes opérations pour obtenir $\hat{x_n}$ auquel il rajoutera le résidu du codeur.

Remarque: Pour les N premiers échantillons on peux utiliser un prédicteur à 1 pas avec $a=0$ ou $a=0$

Avantages :

- Il est très adaptatif.
- On ne transmet plus $\hat{\mathbf{a}}_{opt}$

Inconvénient :

- et bien... c'est pas simple.

2 Schéma de prédiction en boucle fermée

Dans les schémas de codage avec pertes le récepteur dispose de $\tilde{x}_{n-M}...\tilde{x}_{n-1}$ qui sont différents de $x_{n-M}...x_{n-1}$.

Les fonctions de corrélations estimées ainsi que les prédicteurs seront différents. L'erreur entre le signal initial et le signal reconstitué aura tendance à augmenter.

Pour résoudre ce problème le codeur va comprendre un décodeur "local" qui va permettre d'estimer $\tilde{x}_{n-M}...\tilde{x}_{n-1}$. Ensuite γ_x et le prédicteurs seront estimée à partir de $\tilde{x}$ et non de x.

On montre qu'avec ce schéma les erreurs de quantification ne s'accumule pas.

$$x_n - \tilde{x}_n = x_n - \hat{x}_n + \hat{x}_n + \tilde{x}_n = e_n - \tilde{e}_n$$

Sur le n-ième échantillion on a seulement l'erreur de quantification associé a cet échantillon.

Remarque: La qualité du prédicteur va influencer le débit et dans une moindre mesure la distorsion

Chapitre 5

Codage par transformée

1 Principe du codage par transformée

On considère une source $\mathcal{X} \in \mathbb{R}^n$ ou $\in \mathbb{R}^{n \times m}$ (images). Et une réalisation de cette source $\mathbf{x} \in \mathbb{R}^n$ ou $\mathbf{X} \in \mathbb{R}^{n \times m}$.

La représentation naturelle de $\mathbf{x}$ est la base canonique de $\mathbb{R}^n$. Si $\mathbf{x}$ représente par exemple une suite de N échantillons d'un signal audio, les composantes de $\mathbf{x}$ auront plus ou moins la meme variance, mais ils ne seronts (en général) pas indépendant entre eux. POur exploiter cette propriété, on peux :

- réaliser un codage prédictif (voir chapitre 4)
- réaliser un codage par transformée

Dans le second cas on essaie d'exprimer $\mathbf{x}$sur une "meilleure" base que la base canonique.

2 Transformations

Proposition

Une base de $\mathbb{R}^n$ est ensemble de n vecteurs $\{\mathbf{u_1}...\mathbf{u_n}\}$ linéairement indépendants. Si on pose

$$\mathbf{U} = [\mathbf{u_1}...\mathbf{u_n}]$$

Alors $\mathbf{U}$ est inversible.

Transformation inverse

Définition

On dispose de $\mathbf{t}$ vecteur dont les composantes sont exprimées dans la base $\{\mathbf{u_1}...\mathbf{u_n}\}$ dans la base canonique On a :

$$\mathbf{x} = \sum_{i=1}^{n} t_i \mathbf{u_i} = \mathbf{Ut}$$

Transformation directe

Définition

On dispose de $\mathbf{x}$ dont les composantes sont données dans la base canonique. On veux obtenir son vecteur de composante $\mathbf{t}$ dans la base $\{\mathbf{u_1}...\mathbf{u_n}\}$:

$$\mathbf{t} = \mathbf{U}^{-1}\mathbf{x}$$

Base unitaires

Définition

Une base est dite *unitaire* ssi

$$\begin{cases} < \mathbf{u}_i, \mathbf{u}_j >= 0 & \forall i \neq j \\ < \mathbf{u}_i, \mathbf{u}_i >= 1 & \forall i \in [1...n] \end{cases}$$

Proposition

Pour une base unitaire on a :

- transformée inverse : $\mathbf{x} = \mathbf{U}\mathbf{t}$
- transformée directe : $\mathbf{t} = \mathbf{U}^T\mathbf{x}$
- $\mathbf{U}^T\mathbf{U} = \mathbf{U}\mathbf{U}^T = \mathbf{I_n}$

une transformée dans une base unitaire préserve la norme quadratique.

Démonstration :

$$\|\mathbf{t}\|^2 = \mathbf{t}^T\mathbf{t} = \mathbf{x}^T\mathbf{U}^T\mathbf{U}\mathbf{x} = \mathbf{x}^T\mathbf{x} = \|\mathbf{x}\|^2$$

■

Remarque: Une transformée est unitaire lorsqu'elle implique une base unitaire.

Si on veux transformée une image $\mathbf{X} \in \mathbb{R}^{n\times n}$ il faut considéré un ensemble de matrice $\mathbf{U_1}...\mathbf{U_{n\times n}}\}$ base de $\mathbb{R}^{n\times n}$. Pour réaliser la transformation :

- On construit un vecteur $\mathbf{x} \in R^{n^2}$ à partir des lignes ou des colonnes de $\mathbf{X}$.
- Construire la matrice de transformation $\mathbf{U} \in \mathbb{R}^{n^2\times n^2}$
- Calculer $\mathbf{t} = \mathbf{U}^{-1}\mathbf{x}$
- Reformer $\mathbf{T} \in \mathbb{R}^{n\times m}$ à partir de t.

Proposition

Une transformée pour une image $\mathbf{x} \in \mathbb{R}^{n \times m}$ est dites *séparable* s'il existe une matrice $\mathbf{U_s}$ telle que la matrice $\mathbf{T}$ transformée définie précédement puisse s'écrire

$$\mathbf{T} = \mathbf{U_s}^{-1}\mathbf{x}(\mathbf{U_s}^{-1})^T$$

et si U_s est unitaire :

$$\mathbf{T} = \mathbf{U_s}^T x \mathbf{U_s}$$

Remarque: Pour une transformée non séparable on a une complexité en $O(n^4)$. Pour une transformée séparable on a une complexité en $O(2n^3)$

3 Transformée de Karhuman-Loeve

On considère une source vectorielle, centrée et de matrice de covariance $\mathbf{\Gamma}$:

$$\mathbf{\Gamma} = E(\mathbf{X}.\mathbf{X}^T)$$

Remarque: $\mathbf{\Gamma}$ est symétrique.

On cherche une transformée discrète optimale décrite par $\mathbf{A}$ et une transformée inverse décrite par $\mathbf{B}$. qui soit *optimale* au sens de l'EQM. de reconstruction lorsqu'on réalise une réduction à m composantes du vecteur transformé.

Le vecteur transmformé est :

$$\mathbf{t} = \mathbf{A}\mathbf{x}$$

On ne garde que les m premières composantes de $\mathbf{T}$ pour obtenir $\mathbf{t}_m$.

$$\mathbf{t_m} = \mathbf{I_m}\mathbf{t} \quad \text{avec } \mathbf{I_m} = diag(\underbrace{1, ..., 1}_{m}, \underbrace{0, ..., 0}_{N-m})$$

Le vecteur reconstruit est alors :

$$\hat{\mathbf{x_m}} = \mathbf{B}\mathbf{t_m} = \mathbf{B}\mathbf{I}_m\mathbf{t} = \mathbf{B}\mathbf{I_m}\mathbf{A}\mathbf{x}$$

Théorème

La matrice de transformation directe qui minimise l'EQM de reconstruction lors d'une reduction à m composante d'une source X en considérant la transformée inverse est telle que

$$\mathbf{B} = \mathbf{A}^T$$

et où $\mathbf{A}$ est formée des vecteurs propres de $\mathbf{\Gamma}$ rangés par valeurs propres décroissantes.

Démonstration : À partir de l'expression de l'EQM on a :

$$\begin{aligned}
J_m &= \frac{1}{N}E((\mathbf{x}-\hat{\mathbf{x}})^T(\mathbf{x}-\hat{\mathbf{x}})) \\
&= \frac{1}{N}tr(E((\mathbf{x}-\hat{\mathbf{x}})(\mathbf{x}-\hat{\mathbf{x}})^T)) \\
&= \frac{1}{N}tr(\mathbf{t}-\mathbf{BIA}E(\mathbf{x}\mathbf{x}^T)(\mathbf{t}-\mathbf{BI_mA})^T) \\
&= \frac{1}{N}tr(\mathbf{t}-\mathbf{BIA\Gamma}(\mathbf{t}-\mathbf{BI_mA})^T) \\
&= \frac{1}{N}\left[tr(\mathbf{\Gamma})-tr(\mathbf{BI_mA\Gamma})-tr(\mathbf{\Gamma}(\mathbf{BI_mA})^T)+tr(\mathbf{BI_mA\Gamma A^TI_mB})\right]
\end{aligned}$$

À partir des propriétés élémentaire de la trace (symétrie, bilinéaire)

$$= \frac{1}{N}\left[tr(\mathbf{\Gamma})-tr(\mathbf{A\Gamma BI_m})-tr(\mathbf{AI_mB\Gamma})+tr(\mathbf{BI_mA\Gamma A^TI_m}\mathbf{B}^T)\right]$$

Lemme

Pour une fonction matricielle $\mathbf{Y}=f(\mathbf{A})$ on a :

$$D_{\mathbf{A}}(\mathbf{B}) = \frac{\partial tr(\mathbf{Y})}{\partial A}$$

qui est la matrice contenant les dérivées de $tr(\mathbf{A})$ par rapport aux composante de $\mathbf{A}$. Alors :

$$D_{\mathbf{A}}(\mathbf{AB}) = \frac{\partial tr(\mathbf{AB})}{\partial A} = \mathbf{B}^T$$

$$D_{\mathbf{A}}(\mathbf{ABA}^T) = \mathbf{AB}^T + \mathbf{AB}$$

$$D_{\mathbf{A}}(\mathbf{CABA^TD}) = \mathbf{C}^T\mathbf{D}^T\mathbf{AB}^T + \mathbf{DCAB}$$

Avec le lemme on a :

$$\frac{\partial J_m}{\partial \mathbf{A}} = \frac{1}{N}\left[\mathbf{0}-2\mathbf{\Gamma BI_m}^T+(\mathbf{BI_m})^T(\mathbf{I_mB}^T)^T\mathbf{A\Gamma}+(\mathbf{I_mB^T})(\mathbf{BI_m})\mathbf{A\Gamma}\right]=0$$

On en déduit :

$$\mathbf{I_mB}^T(\mathbf{I}-\mathbf{I_mA})\mathbf{\Gamma}=\mathbf{0} \qquad ((1))$$

On réecrit alors le critère sous la forme :

$$\begin{aligned}
J_m &= \frac{1}{N}tr((\mathbf{I}-\mathbf{BI_mA})\mathbf{\Gamma})-\frac{1}{N}tr((I-\mathbf{BI_mA})(\mathbf{BI_mA})^T) \\
&= \frac{1}{N}tr((\mathbf{I}-\mathbf{BI_mA})\mathbf{\Gamma})-\frac{1}{N}tr(\mathbf{A}^T\mathbf{I_mB}^T(\mathbf{I}-\mathbf{BI_mA}))
\end{aligned}$$

En général Γ etant inversible, en utilisant (1) on déduit :

Si $m=n$ on doit avoir $J_n=0$ soit :

$$tr((\mathbf{I}-\mathbf{BA})\mathbf{\Gamma})=0$$

Une condition suffisante est que :

$$\mathbf{A} = \mathbf{B}^{-1}$$

Avec (1), mais pour tout $\mathbf{\Gamma}$ on a alors : [1]

$$\mathbf{I_m B B}^T = \mathbf{I_m B}^T \mathbf{B I_m}$$

On a alors :

$$I_m = \frac{1}{N} tr(\mathbf{I} - \mathbf{B I_m A \Gamma})$$

Si on prend une matrice $\mathbf{D}$ diagonale inversible et qu'on remplace $\mathbf{B}$ par $\mathbf{BD}$ $\mathbf{A} = \mathbf{B}^{-1}$ va être remplacé par $\mathbf{D}^{-1}\mathbf{A}$

$$\begin{aligned} J_m &= \frac{1}{N}((\mathbf{I} - \mathbf{BD I_m D}^{-1}\mathbf{A})\mathbf{\Gamma}) \\ &= \frac{1}{N}((\mathbf{I} - \mathbf{B I_m A})\mathbf{\Gamma}) \end{aligned}$$

Cette propriété permet de choisir $\mathbf{B}$ unitaire c'est à dire $\mathbf{B}^T\mathbf{B} = \mathbf{I}$ soit $\mathbf{A} = \mathbf{B}$. Alors on remplace dans l'expression de l'EQM :

$$J_m = \frac{1}{N} tr(\mathbf{\Gamma}) - \frac{1}{N} tr(\mathbf{A}^T \mathbf{I_m A \Gamma})$$

Ainsi pour minimiser l'EQM il faut minimiser :

$$tr(\mathbf{A}^T \mathbf{I_m A \Gamma}) = tr(\mathbf{I_m A \Gamma A}^T)$$

On note $\mathbf{a_1}^T ... \mathbf{a_N}^T$ les lignes de $\mathbf{A}$. On a alors :

$$K_m = tr(\mathbf{I_m A \Gamma A^T}) = \sum_{i=1}^{m} \mathbf{a_i}^T \mathbf{\Gamma a_i}$$

qu'il faut minimiser sous le contraintes :

$$\mathbf{a_i}^T \mathbf{a_i}, \forall i \in [1...N]$$

On introduit donc le lagrangien :

$$\mathcal{L}(\mathbf{a_1 ... a_N}, \lambda_1 ... \lambda_N) = \sum_{i=1}^{m} \mathbf{a_i}^T \mathbf{\Gamma a_i} - \sum_{i=1}^{N} \lambda_i (\mathbf{a_i}^T \mathbf{a_i} - 1)$$

On a donc :

$$\frac{\partial \mathcal{L}}{\partial \mathbf{a_i}} = \begin{cases} 2\mathbf{\Gamma a_i} - 2\lambda_i a_i & = 0 \quad \forall i \in [1...m] \\ -2\lambda_i \mathbf{a_i} & = 0 \quad \forall i \in [m+1...N] \end{cases}$$

On a donc $\mathbf{\Gamma a_i} = \lambda_i \mathbf{a_i}$. Les $\mathbf{a_i}$ sont les vecteurs propres de Γ_i associées aux valeurs propres λ_i. Alors

$$K_m = \sum_{i=}^{m} \lambda_i$$

qui est maximisé lorsque les $veca_i$ sont associés aux m plus grandes valeurs propres de Γ.

Ouf ! ■

1. À développer : écriture par bloc des matrice, $B^T B$ est diagonale

3.1 Mise en oeuvre pratique

On dispose d'un signal sonore $x_1...x_M$ centré.

1. On découpe le signal en vecteur de N composante $\mathbf{x_1} = (x_1...x_N), \mathbf{x_2} = (x_{N+1}...x_{2N})...\mathbf{x_K}$.
2. On calcule :
$$\hat{\mathbf{\Gamma}} \frac{1}{K} \sum_{k=1}^{K} \mathbf{x_k} \mathbf{x_k}^T$$
3. On calcule les vecteurs propres $\gamma_\mathbf{1}...\gamma_\mathbf{N}$ de $\mathbf{\Gamma}$ associé aux valeurs propres $\lambda_1 \geq ...\lambda_N$
4. On construit la matrice de transformation :
$$\mathbf{A} = \begin{bmatrix} \gamma_\mathbf{1}^T \\ \gamma_\mathbf{2}^T \\ \vdots \\ \gamma_\mathbf{N}^T \end{bmatrix} = \mathbf{B}^T$$
5. calculer :
$$\mathbf{t_k} = \mathbf{A x_k} \quad \forall k \in [1...K]$$
6. Mettre à 0 les $N - m$ derniers coefficient de $\mathbf{t_k}$.
7. Calculer
$$\hat{\mathbf{x}_\mathbf{k}^{(m)}} = \mathbf{B t_k^{(m)}} \quad \forall k \in [1...K]$$

Remarque: La KLT a cependant des défauts :

- Elles dépend du signal à compresser.
- Elle n'est pas séparable.

De ce fait elle est très peu utilisée en compression d'image

4 Transformée sous optimales

4.1 Transformée en cosinus discrete (DCT)

La DCT est une très bonne alternative à la KLT.

Définition

La transformée en cosinus discrète a pour matrice de transformation:

$$\mathbf{u_{i,k}} = \alpha_i \cos \frac{\pi(2k-1)i}{2N}$$

Avec $\alpha_0 = \frac{1}{\sqrt{N}}$ et $\alpha_i = \sqrt{\frac{2}{N}}$.

Remarque: La DCT est séparable. En terme de compaction d'énergie elle est presque aussi efficace que la KLT. Elle est utilisée en compression audio (mp4) vidéo (H261 à H265)

Annexe A

Implémentation des différents algorithmes

1 Codage d'Huffman

1.1 version simple

```
#!/usr/bin/python3

def get_2min(l):
    min1 = 0
    min2 = 1
    for k in range(1,len(l)):
        if l[k]<=l[min1]:
            min2 = min1
            min1 = k
        elif l[k]<=l[min2]:
            min2 = k
    return min1,min2

def huffman_rec(p):
    if len(p) == 2:
        C = ['1','0']
        print(p,C)
        return C
    else:
       min1,min2 = get_2min(p)
       min1,min2 = min(min1,min2),max(min1,min2)
       print(p,min1,min2)
       p_save=p.pop(min2)
       p[min1] = p[min1]+p_save
       C = huffman_rec(p)
       C.insert(min2,C[min1]+'1')
       C[min1] +='0'
       p.insert(min2,p_save)
       p[min1] -= p_save
       return C

p = [25,20,15,12,10,8,5,5]
print(huffman_rec(p))
```

1.2 version objet

```python
#!/usr/bin/env python3

import subprocess

class Noeud(object):
    def __init__(self,p=0,left=None,right=None,code='',name=''):
        self.left = left
        self.right= right
        if left != None and right != None :
            self.p = left.p + right.p
            self.name =left.name+right.name
        else:
            self.p = p
            self.name = name
        self.code = code
    def __lt__(self,other):
        return self.p<other.p
    def __repr__(self):
        return self.name

def create_tree(table_noeud):
    queue = table_noeud.copy()
    while len(queue) > 2:
        queue.sort()
        l=queue.pop(0)
        r=queue.pop(0)
        queue.append(Noeud(left=l,right=r))
    root= Noeud(left=queue[0],right=queue[1])
    return root

def gen_code(node,prefix=''):
    def gen_code_rec(node,prefix=''):
        if node.left != None:
            node.code = prefix
            t_1 = gen_code(node.left,prefix+'0')
            t_2 = gen_code(node.right,prefix+'1')
            return [*t_1,*t_2]
        else:
            node.code = prefix
            return (node.name,node.code)

    x = gen_code_rec(node,prefix)
    return x
#affichage à l'aide de graphviz
def draw_tree(node):
    if len(node.name) == 1: # feuille
        desc ='N{} [label="{}:{}",\
               fontcolor=blue, fontsize=16,\
               width=2, shape=box];\n'.format(node.code, node.name, node.code)
    else:
        desc = 'N{} [label="{}"];\n'.format(node.code,node.code)
        desc += draw_tree(node.left)
        desc += draw_tree(node.right)
```

```
54          desc += 'N{}:n -> N{}:e;\n'.format(node.code,node.left.code)
55          desc += 'N{}:s -> N{}:e;\n'.format(node.code,node.right.code)
56      return desc
57
58  def make_tree():
59      with open('graph.dot','w') as f:
60          f.write('digraph G {\n ')
61          f.write(' splines=ortho \n')
62          f.write('rankdir=RL;\n')
63          f.write(draw_tree(root_node))
64          f.write('{rank =same; N' + '; N'.join([n.code for n in table_noeud]) +';}\n')
65          f.write('}')
66      subprocess.call('dot -Tpng graph.dot -o graph.png', shell=True)
67
68  def decode_huffman(reverse, code):
69      res =""
70      while code:
71          for k in reverse:
72              if text.startswith(k):
73                  res +=reverse[k]
74                  text = text[len(k):]
75      return res
76
77  table = [('A', 25),('B', 20),('C', 15),('D', 12),
78           ('E', 10),('F', 8),('G', 5),('H', 5)]
79  table_noeud = [Noeud(name=x[0],p=x[1]) for x in table]
80  root_node= create_tree(table_noeud)
81  x= gen_code(root_node)
82  reverse_huffman = {x[i+1]:x[i] for i in range(0,len(x)-1,2)}
83  print(table_huffman)
```

2 Codage arithmétique

```python
#!/usr/bin/env python3

import numpy as np

X=[0,1,0,0,1,0,0,0,0,0]

def binary(n,m,b=2):
#    Convertie un nombre décimal en sa version binaire tronqué à m bits.
    binaire= np.floor(n*b**m) # on se décale dans les entiers et on floor
    return binaire,np.binary_repr(int(binaire))

def arithm(X,p):
    l=[0]; h= [1]
    for x in X:
        if x == 0:
            h.append(l[-1]+p*(h[-1]-l[-1]))
            l.append(l[-1])
        else:
            l.append(l[-1]+p*(h[-1]-l[-1]))
            h.append(h[-1])
            lmb = (l[-1]+h[-1])/2
            mu = int(-np.log2(h[-1]-l[-1]))+1
    code = binary(lmb,mu)
    return code,lmb,mu

def arithm_pratique(X,p):
    l = [0]; h =[1] ;f = 0;c =[] #inf, sup,follow, code
    for k in range(len(X)):
        print("for loop")
        if X[k] == 0:
            l.append(l[-1])
            h.append(l[-1]+p*(h[-1]-l[-1]))
        else:
           l.append(l[-1]+p*(h[-1]-l[-1]))
           h.append(h[-1])
        while ((l[-1]>=0 and h[-1]<0.5) or (l[-1]>=0.5 and h[-1]<1) or (l[-1]>= 0.25 and h[-1]<0.75)):
            if (l[-1]>=0 and h[-1]<0.5):
                c += [0]+[1]*f
                l[-1] *=2
                h[-1] *=2
            elif (l[-1]>=0.5 and h[-1]<1):
                c += [1]+[0]*f
                l[-1] = 2*l[-1]-1
                h[-1] = 2*h[-1]-1
            elif (l[-1]>= 0.25 and h[-1]<0.75):
                f +=1
                l[-1] = 2*l[-1]-0.5
                h[-1] = 2*h[-1]-0.5
    return c
print(arithm_pratique(X,p))
```

3 Codage LZW

```python
#!/usr/bin/env python3

import numpy as np

univers = ['a','b','c']
message = "aababababac"

def code_LZW(message, univers):
    msg = message
    dictionnaire = dict(zip(univers,[i for i in range (len(univers))]))
    w=""
    code =[]
    for c in msg:
        wc = w+c
        if wc in dictionnaire:
            w =wc
        else:
            code.append(dictionnaire[w])
            dictionnaire[wc] = len(dictionnaire)
            w = c
    if w:
        code.append(dictionnaire[w])
    return code,dictionnaire

def decode_LZW(code,univers):
    dictionnaire = dict(zip([i for i in range(len(univers))],univers))
    w = dictionnaire[code.pop(0)]
    msg = [w]
    for k in code:
        if k in dictionnaire:
            entry = dictionnaire[k]
        elif k == len(dictionnaire):
            entry = w +w[0]
        msg.append(entry)
        dictionnaire[len(dictionnaire)] = w+entry[0]
        w = entry
    print(dictionnaire)
    return ''.join(msg)

code,dictionnaire = code_LZW(message,univers)
msg = decode_LZW(code, univers)
print(code, dictionnaire, msg)
```

4 Quantification

4.1 Quantification uniforme

```python
#!/usr/bin/env python3

import numpy as np
```

```
4  import matplotlib.pyplot as plt
5
6  N = 1000
7  X = np.random.rand(N)
8  X_c = (X - 0.5)*10
9
10 def quantif_uniforme(M,X,xmin=-1,xmax=1,d=0):
11     """
12     réalise la quantification uniforme d'un vecteur sur M niveau
13     """
14     delta = 2 * xmax/M # pas de quantification
15     Q = np.zeros(len(X))
16     for k in range(len(X)):
17        q =  (X[k]/ delta)
18        if abs(q)<d: #seuil
19            Q[k] = 0
20            continue
21        elif abs(q)<2*delta:
22            if q <0:
23                Q[k] =-1
24            else:
25                Q[k] = 1
26            continue
27        else:
28            Q[k] = int(q)
29
30     return Q,delta
31
32 def reverse_quantif(Q,delta):
33     return Q*delta
34
35 Q,delta = quantif_uniforme(4,X_c)
36 Q_2,delta = quantif_uniforme(4,X_c,d=0.5):
37
38
39 print(len(Q),len(X_c))
40 plt.figure()
41 plt.grid()
42 plt.plot(X_c,Q,'.')
43 plt.plot(X_c,Q_2,'.')
44 plt.show()
45
46
```

4.2 Algorithme de Llyod-max

```python
#!/usr/bin/env python3
import numpy as np
from scipy import integrate
from scipy.stats import norm
import matplotlib.pyplot as plt

def ddp(x):
    mean = 0; sigma = 1
    return norm.pdf(x,mean,sigma)

def quant(centroids, X):
    bornes = (centroids[:-1]+centroids[1:])/2
    bornes = np.insert(bornes,0,-np.inf)
    bornes = np.append(bornes,np.inf)
    xquant =np.zeros(len(X))
    for k in range(len(X)):
        for i in range(len(bornes)):
            if X[k]>=bornes[i] and X[k] <bornes[i+1]:
                xquant[k] = centroids[i]
    return xquant
def llyodMax(X,M,maxiter=1000,eps=1e-6):
    #répartition uniforme des bornes
    step = (np.max(X)-np.min(X))/(M-2)
    Xmin = np.min(X)
    bornes = np.array([i*step+Xmin for i in range(M-1)])
    bornes = np.insert(bornes,0,-np.inf)
    bornes = np.append(bornes,np.inf)
    centroids = np.zeros(M)
    for  it in range(maxiter):
        old_centroids = centroids.copy()
        for i in range(M):
            centroids[i] = integrate.quad(lambda x: x*ddp(x),bornes[i],bornes[i+1])[0]\
                        /integrate.quad(lambda x: ddp(x),bornes[i],bornes[i+1])[0]
        bornes[1:-1] = (centroids[:-1]+centroids[1:])/2
        err = np.linalg.norm(centroids-old_centroids)
        if err < eps :
            break
    return centroids

M = 4
X = np.random.normal(0,1,1000)
centroids = llyodMax(X,M)
bornes = (centroids[:-1]+centroids[1:])/2
bornes = np.insert(bornes,0,-np.inf); bornes = np.append(bornes,np.inf)
plt.figure()
plt.plot(X)
plt.plot(quant(bornes,X))
plt.show()
```

4.3 Algorithme LBG

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial import Voronoi, voronoi_plot_2d
#initialisations clusters
M = 20;
N =100; #point par cluster
K = N*M
means = np.random.rand(M,2)*10
X = np.zeros((K,2))
plt.figure()
cov = np.array([[1,0],[0,1]])
for m in range(M):
    xi = np.random.multivariate_normal(means[m,:],cov,N)
    X[m*N:(m+1)*N] = xi
    plt.plot(xi[:,0],xi[:,1],'+')
plt.plot(means[:,0],means[:,1],'ob')
mean= np.mean(X,axis=0)
Y0 = np.random.multivariate_normal(mean, 10*cov, M)
plt.show()
Y0= means #triche
plt.plot(Y0[:,0],Y0[:,1],'ok')
plt.show()
def LBG(X,Y0,eps=1e-5,maxiter=1000):
    Y = Y0.copy()
    old_dist = np.inf
    cluster_index = np.zeros(K,dtype=int)
    for l in range(maxiter):
        dist= 0;
        for k in range(len(X)):
            quant_min =np.inf
            for j in range(len(Y)):
                if np.linalg.norm(X[k]-Y[j]) <np.linalg.norm(X[k]-quant_min):
                    quant_min = Y[j]
                    cluster_index[k] = j
            dist += sum((X[k]-quant_min)**2)
        for j in range(len(Y)):
            Y[j,:] = np.mean(X[cluster_index==j],axis=0)
        if dist-old_dist < eps:
            break
        else:
            old_dist = dist
    return Y
Y = LBG(X,Y0)
vor = Voronoi(Y)# black magic
voronoi_plot_2d(vor,show_vertices=False)
plt.plot(X[:,0],X[:,1],'+')
plt.plot(Y[:,0],Y[:,1],'ob')
plt.plot(Y0[:,0],Y0[:,1],'ok')
plt.show()
```

5 Codeur prédictif

Construire un schéma de prédiction en boucle fermée à fenêtre glissante dasn lequel Met p sont paramétrisable. On utilisera un quantificateur à zone morte de pas Δ paramétrisable.

6 KLT

Evaluer le KLT une restriction à m composante pour un signal sonore. Tracer l'EQM en fonction de m. Estimer le signal reconstruit.