وزارة التعليم العالي والبحث العلمي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
جامعة عبد الحميد ابن باديس مستغانم
Université Abdelhamid Ibn Badis de Mostaganem
كلية العلوم والتكنولوجيا
Faculté des Sciences et de la Technologie

N° d'ordre : M...../GE/2018

MEMOIRE DE FIN D'ETUDES
MASTER ACADEMIQUE

Filière : Électronique

Spécialité : Systèmes des Télécommunications

Thème

Analyse de l'Efficacité et la Fiabilité d'une Chaîne de Transmission Numérique du Son avec Compression de Données, Détection et Correction des Erreurs

Présenté par :

KAOUDER Abdelkader
KACEM Fatima

Soutenu le 01/07/2018 devant le jury composé de :

Président : Mr. BOUKORTT Abdelkader

Examinatrice : Mme. BENCHELLAL Amel

Examinateur : Mr. YAGOUBI bouabdellah

Encadreur : Mr. ABED Mansour

Année Universitaire : 2017/2018

Dédicaces

Je dédie ce travail :

A mes chers parents, pour leurs patiences, leur soutien et leur Encouragements, Que dieu leur procure bonne santé et longue vie ;

A mes frères et ma sœur Wafaa ;

A celui que j'aime beaucoup et que m'a soutenue tout au long de ce projet : Mon Fiancé « Abdelkader »;

A mon encadreur « Mr. ABED Mansour »;

A toute ma famille de prés ou de loin.

KACEM Fatima

Dédicaces

Je dédie ce modeste travail :

A mes très chers parents,
pour leur Encouragement et leurs sacrifices qu'ils ont endurés. A mon frères et mes sœurs ;

A ma future femme « Kacem Fatima »;

A Mon encadreur « Mr. ABED Mansour » qui m'a aidé à améliorer mes connaissances en me donnant informations et conseils ;

A toute ma famille de prés ou de loin.

KAOUDER Abdelkader

Sommaire

Liste des symboles

T	Période d'un symbole.
T_e	Période d'échantillonnage.
F_e	Fréquence d'échantillonnage.
F_{max}	Fréquence maximal.
q	Le pas de quantification.
V	La plage de quantification.
B	Nombre de bits de la représentation numérique.
$(S/B)_q$	Signal sur bruit quantifié.
Δ	Intervalle de quantification.
A	Amplitude.
$m[n]$	Un message binaire émis constitué de n bits.
$A[k]$	k vecteurs, chacun contenant n/k bits.
$e(t)$	Un signal modulé obtenu par la transformation des symboles en signaux.
$h_e(t)$	Filtre de mise-en-forme, généralement de forme carrée.
$x(t)$	Signal numérique émis.
C	Capacité du canal.
N_0	Puissance du bruit.
σ^2	Variance du bruit.
B	Largeur de bande.
P_s	Puissance de signal.
$p(\ )$	Probabilité.
$H(\varphi)$	Entropie de source discrète sans mémoire.
K	Le radix (nombre de symboles).
$\bar{L}$	La longueur moyenne.
L_{min}	La longueur minimale.
η	L'efficacité de codage de source.
$g(x)$	Polynôme générateur.
G	Matrice génératrice.
I_k	Matrice identique.
n	nombre de bits du message après codage de canal (longueur du mot de code).
k	nombre de bits du message sans codage (longueur du message après codage de source)

n-k	Nombre de bits de contrôle (ou de redondance).
p_{err}	Probabilité d'erreur.
r	Rendement d'un code.
m	Message.
v	Mot de code.
C_i	Les bits de contrôle de parité.
D_j	Les bits du message utile (de source discrète).
S	Le syndrome.
R	Mot de code reçu.
H^T	Matrice de contrôle transposée.
e	Vecteur d'erreur
$\hat{v}$	Mot de code corrigé.
d_{min}	Distance minimale de code.
$\lfloor . \rfloor$	La partie entière.
$f(n,k,m)$	Forme générale d'un code convolutif.
N	Nombre de bits à la sortie de codeur.
k	Nombre de bits à l'entrée de codeur (bits de message de source).
m	Nombre de registre à décalage (mémoire).
L	La longueur de contrainte du code.
S_j	L'entrée à l'instant j (bit actuel à coder).
X_j	La sortie à l'instant j (sortie actuel du codeur).
e_j^1, ...	Registre à décalage.
$H(D)$	La réponse impulsionnelle.
D	La taille de fichier en octets.
Q	Le nombre d'octets utilisés pour coder un signal.
P	Le nombre de piste. (en stéréo on utilise deux pistes).
A	Intervalle de variation des valeurs des échantillons.
N	Nombre de bits utilisés pour coder l'échantillon.

Liste des abréviations

AWGN	Additive White Gaussian Noise.
Bit	Binary digit.
PCM	Pulse Code Modulation.
RLE	Run-Lenght Encoding.
WAV	Wave Forme Audio Vector.
MPEG	Moving Pictures Experts.
WMA	Windows Media Audio.
ACC	Advanced Audio Coding.
MIDI	Musical Instrument Digital Interface.
CN	Communication Numérique.
CRC	Le code cyclique de redondance (en anglais CRC pour Cyclic Redundancy Check).
LBC	Les codes linéaires en blocs (en anglais LBC pour Linear Block Code).
ML	Maximum Likelihood (la vraisemblance maximale).
SR	Shift Register (registre à décalage).
TEB	Taux d'Erreur par Bit(en anglais *BER* pour Bit Error Rate).
VLC	Variable Code Length (Code à longueur variable).

Liste des Figures

Chapitre I

Chapitre II

Liste des Tableaux

Introduction générale

Introduction générale

De nos jours, nous vivons dans un monde où les communications jouent un rôle primordial tant par la place qu'elles occupent dans le quotidien de chacun, que par les enjeux économiques et technologiques dont elles font l'objet. Nous avons sans cesse besoin d'augmenter les débits de transmission tout en gardant ou en améliorant la qualité de ceux-ci. Mais sans un souci de fiabilité, tous les efforts d'amélioration seraient ridicules car cela impliquerait forcément à ce que certaines données soient retransmises. C'est dans la course au débit et à la fiabilité que les techniques de codage de canal entrent en jeux.

En particulier, la numérisation du signal a permis de concevoir des systèmes de communications très performants. Ceci n'était pas possible à réaliser sans introduire de nouveau blocs primordiaux comme le codage de source, le codage de canal, le filtrage adapté, etc. Ces développements révolutionnaires dans le domaine de la théorie de l'information permettent à l'heure actuelle de réaliser des systèmes de communications numériques de très hauts débits tout en respectant les exigences de probabilités d'erreur qui deviennent de plus en plus strictes. En fait, avec tous ces térabits par seconde 'volant' à travers le monde en câbles en cuivre, fibres optiques et liaisons radio, la question qui se pose est : Combien ces réseaux sont-ils fiables ? Si on perd 0.001% seulement de données en un réseau fonctionnant à un gigabit par seconde, ça s'élève à dix mille bits par seconde ! Une histoire typique de journal contient 1000 mots, ou bien environ 42000 bits d'information. Imaginons qu'on perd une histoire de journal en fils chaque quatre secondes, ceci fait 900 histoires par heure ! Ceci dit, même un taux d'erreur en bits (TEB) petit ne devient alors pratique lorsque les débits binaires croient considérablement comme enregistré dans la dernière décennie. Par conséquent, le TEB demeure un critère fondamental d'évaluation des systèmes de communications numériques, en plus de l'efficacité spectrale et de la complexité du récepteur. Ceci explique pourquoi on exige actuellement des TEBs allant jusqu'à 10^{-12}, soit alors un bit reçu erroné sur 10^{12} bits émis !

D'autre part, la transmission d'un signal à grande distance, en particulier pour les systèmes sans fils, est obligatoirement perturbée, notamment à cause de diverses sources de perturbation électromagnétique, mais aussi à cause d'autres utilisateurs partageant le même réseau lorsqu'il s'agit d'un accès multiple et du bruit thermique inévitable au niveau du récepteur.

Vers la fin des années 30, Claude Shannon démontra qu'à l'aide de "contacteurs" (interrupteurs) fermés pour vrai et ouverts pour faux il était possible d'effectuer des opérations logiques en associant le nombre 1pour vrai et 0 pour faux. Ce codage de l'information à deux états est nommé base binaire. C'est grâce à ce codage que fonctionnent les ordinateurs. Il consiste à utiliser deux états électriques différents pour différencier les deux états, le 0 et le 1, et ainsi coder les informations.

Un système de communications numériques permet la transmission de messages entre un émetteur et un destinataire via un canal de communication. Le canal permet de transmettre des signaux (souvent un signal électrique binaire). Il s'agit alors de transformer le message à émettre (par exemple un texte) en une suite binaire. Cette opération est dite codage de source. Le destinataire doit être capable de décoder la séquence de signaux reçus pour pouvoir lire le message émis. Dans un souci d'efficacité, la séquence de signaux doit être la plus courte possible. On s'intéresse donc à des codages qui minimisent la taille de la séquence émise : On parle de codage de source avec compression sans pertes de données ; le concept discuté dans le premier chapitre de ce mémoire.

Cependant, il existe un autre type de compression dite destructive ou compression avec pertes. Les formats de compression les plus connues et qui appartiennent à cette catégorie sont le JPG pour l'image, JPEG pour la vidéo et la télévision numérique et le mp3 pour les sigaux audio. Ce dernier schéma est discuté dans le deuxième chapitre.

Puisque le médium peut introduire des erreurs, certains signaux émis peuvent être perdus ou altérés lors de la transmission. Dans ce cas, pour que le destinataire puisse décoder correctement le message reçu, on utilise des codages spécifiques, permettant de détecter, voire corriger les erreurs. On parle de code détecteur/correcteur de l'erreur.

Origine de ce type de codes est la théorie de l'information initiée par C. Shannon dans les années 50. Ils sont utilisés pour la transmission (par satellite, téléphonie, disque laser, TV haute définition), qui luttent contre le bruit, traitement d'image et de la parole, cryptographie (signature, carte à puce), compression de données, etc.…

Un code correcteur d'erreur permet de corriger une ou plusieurs erreurs dans un mot de code en ajoutant aux informations des symboles redondants, autrement dits, des symboles de contrôle. Différents codes possibles existent mais dans ce document nous traiterons quelques types de codes linéaires en bloc les plus utilisés en pratique, les codes cycliques et les codes BCH ainsi que les codes convolutifs. Afin d'analyser la fiabilité de transmission pour chaque type de codes, nous avons considéré l'émission/réception des sons dans un canal bruité. La

mesure de fiabilité de canal après introduction de codeurs détecteurs et correcteurs d'erreurs s'effectue d'une part par la visualisation des sons reconstruites au niveau du récepteur sans et avec codage (appréciation audible, mesure subjective), et d'autre part, par calcul des TEB*s* réalisés pour chaque schéma de transmission (mesure objective, quantitative).

Le présent mémoire comporte cinq chapitres organisés selon la chronologie suivante :

-Le premier chapitre se consacre aux généralités sur la numérisation des signaux et codage de source avec compression sans pertes de données,

-Le deuxième chapitre décrit le codage et compression des signaux audio,

- Dans le chapitre trois on décrit le codage de canal par les codes linéaires en bloc,

-Le quatrième chapitre est une présentation détaillée du codage de canal par les codes convolutifs.

-Le chapitre cinq présente les tests les plus pertinents avec visualisation et discussion des résultats obtenus afin d'examiner l'efficacité et la fiabilité de transmission des sons travers un canal bruité dans le cas de codage de canal et sans codage de canal. L'effet de compression sans pertes suivie de détection et correction des erreurs sur la transmission d'un texte brut est aussi testé.

Enfin, nous clôturons le présent travail par des conclusions et suggestions pour les projets ultérieurs. Une annexe est fournie à la fin de ce document contenant les codes sources en Matlab concernant le codage de source avec compression non-destructive et le codage de canal par les codes linéaires en blocs et les codes convolutifs.

Chapitre I

Généralités sur la Numérisation des Signaux et Codage de Source

I.1 Introduction

Le rôle d'un système de télécommunications numérique est de transmettre à distance des informations d'un émetteur à un ou plusieurs récepteurs au travers d'un canal de manière efficace et fiable et à coût réduit. L'information peut être soit analogique soit numérique. Dans l'ordinateur, le signal est numérique et utilise deux tensions pour représenter le bit (binary digit). Le signal correspondant à la séquence binaire et circulant sur le support de transmission est soit un signal analogique soit un signal numérique. Le choix est fait selon les caractéristiques du support et ceux du signal à transmettre [1].

La technique de transmission du signal numérique tel qu'il était produit est appelée *transmission en bande de base* tandis que la transmission analogique induisant une modulation d'une porteuse sinusoïdale par un signal numérique basse fréquence est appelée *transmission par transposition de fréquence*. Cette opération résulte en une occupation fréquentielle plus importante.

Presque tous les transports d'information s'effectuent aujourd'hui en numérique : Téléphone, télévision numérique, internet, etc. Le processus de numérisation de signal consiste à effectuer un échantillonnage du signal analogique acquis par un capteur convenable suivi d'une quantification et un codage. Cette transformation engendre toujours des pertes d'information.

Notons que la numérisation du signal résulte des études effectuées par Nyquist et Shannon. Ce dernier est le fondateur de la théorie fondamentale de l'information à savoir le codage de source, le codage de canal et la relation entre débit maximal de transmission sans erreurs, le rapport signal sur bruit (*RSB*) et la largeur de bande des systèmes de communications numériques.

I.2 Objectifs de ce chapitre

L'article repère de Shannon en théorie de l'information publié en 1948, et ses raffinements par d'autres chercheurs, étaient en réponse directe aux besoins des ingénieurs électriciens pour pouvoir concevoir des systèmes qui sont à la fois efficaces et fiables.

Comme nous avons vu en cours, une communication efficace entre une source et un destinateur est atteinte à travers le codage de source. Une communication fiable à travers un canal bruité est atteinte à travers le codage de canal.

Ce premier chapitre s'occupe de codage de source et a pour objectifs de :

- Construire la suite binaire correspondante à l'information à transmettre à partir d'une source analogique.
- Connaitre les éléments de base constituant un système de communications numériques.

- Déterminer le débit maximal de transmission sur erreurs en relation avec la largeur de bande de canal, la puissance du signal et la puissance du bruit en se basant sur le théorème de Shannon de capacité de canal.
- Identifier les propriétés d'une source discrète sans mémoire.
- Comprendre la notion de l'information et sa relation avec l'incertitude et la surprise.
- Mesurer la quantité d'information gagnée lorsqu'un symbole issu d'une source discrète sans mémoire est observé.
- Déterminer l'entropie de la source comme une mesure de base de l'information.
- Comprendre le théorème de Shannon de codage de source et son application dans la compression sans pertes de données.
- Différencier entre un codage sans statistiques et un codage entropique.
- Effectuer un codage de source en utilisant les codes préfixes et mesurer la performance du codage élaboré.
- Effectuer un codage de source avec une efficacité optimale en utilisant les codes entropiques, tel que le code de Huffman, et les codes arithmétiques qui ne nécessitent pas une connaissance au préalable de la loi de probabilité de la source.

I.3 Numérisation des Signaux

I.3.1 Représentations du signal

Les signaux rencontrés dans la nature sont par nature analogiques, pour pouvoir les traiter numériquement, il faut d'abord les numériser pour obtenir un signal discret ou signal numérique ou encore signal digital. On désigne par signal l'information relative à une grandeur physique qui évolue dans le temps. Les signaux les plus couramment utilisés sont les signaux électriques. Mais ces signaux sont les plus souvent des traductions de signaux physiques comme des signaux acoustiques, sismiques, de température ou de pression … L'obtention des signaux électriques à partir des variations d'une grandeur naturelle se fait à l'aide d'un capteur (comme par exemple un capteur de température) ou d'un transducteur (comme l'antenne).

a) Signal analogique

Un signal analogique (Figure 1.1) est un signal continu qui peut prendre une infinité de valeurs. C'est un signal physique, réel, acquis en utilisant un capteur ou un transducteur convenable.

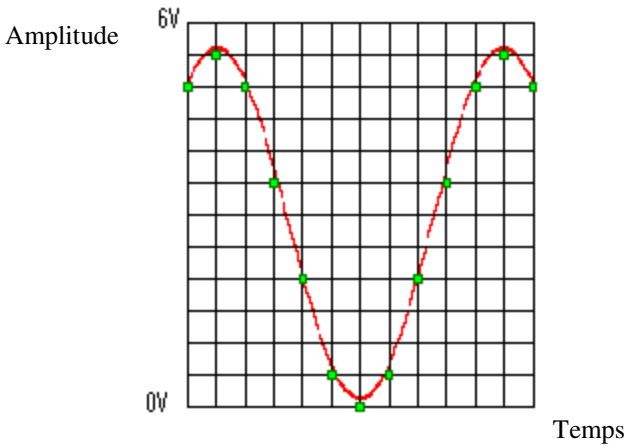


Figure 1.1 : Exemple d'un Signal analogique [2].

Le signal analogique souffre d'un inconvénient majeur : il est très fragile au bruit, aux interférences et parasites. En outre, son enregistrement, son traitement ou sa transmission lui fait subir différents types de dégradations, qui altèrent rapidement sa qualité.

b) Signal numérique

Le signal numérique comme représenté par exemple dans la Figure 1.2 est un signal discret (discontinu). Le terme numérique qualifie en particulier toute donnée ou variable dont les valeurs sont des nombres entiers. Il représente la valeur arrondie d'une grandeur physique analogique à un instant donné

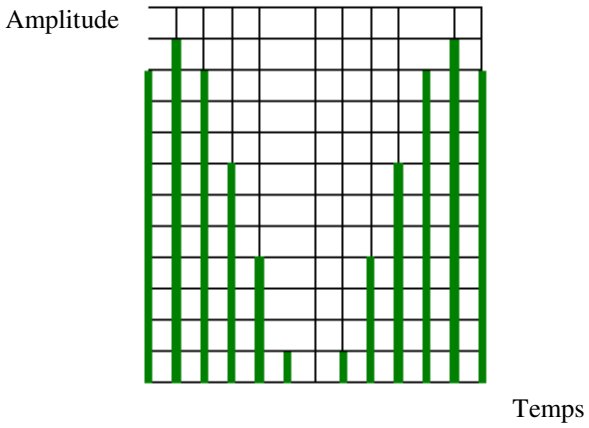


Figure 1.2 : Exemple d'un signal numérique [2].

Le signal numérique est particulièrement stable. Il se prête remarquablement aux traitements les plus complexes et aux copies cumulatives à travers lesquelles l'information qu'il porte est totalement préservée.

I.3.2 Numérisation

La numérisation d'un signal (Figure 1.3) consiste à convertir un signal analogique qui contient une quantité infinie d'amplitudes en un signal numérique contenant lui une quantité finie de valeurs. Cette conversion comporte trois opérations fondamentales :

- Echantillonnage,
- Quantification des échantillons obtenus,
- Codage des échantillons quantifiés.

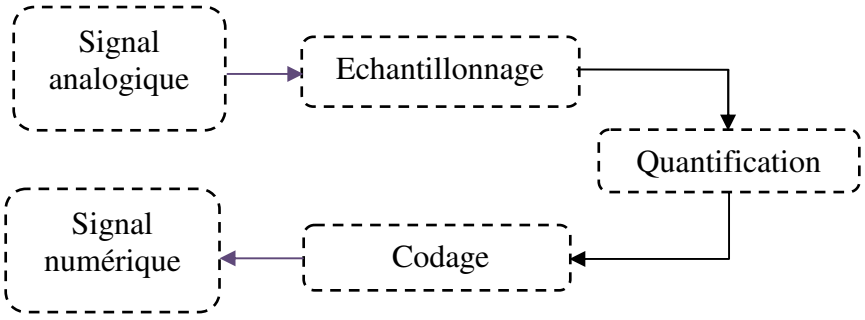


Figure 1.3 : Les étapes de numérisation d'un signal analogique.

a) Échantillonnage

L'échantillonnage (Figure 1.4) consiste à remplacer une fonction continue dans le temps ou dans l'espace par la suite des valeurs qu'elle prend en des instants ou des zones discrets périodiques. Ces valeurs prises à des instants d'échantillonnage T_e doivent suffire pour reconstituer le signal analogique de la source sans engendrer une perte d'information.

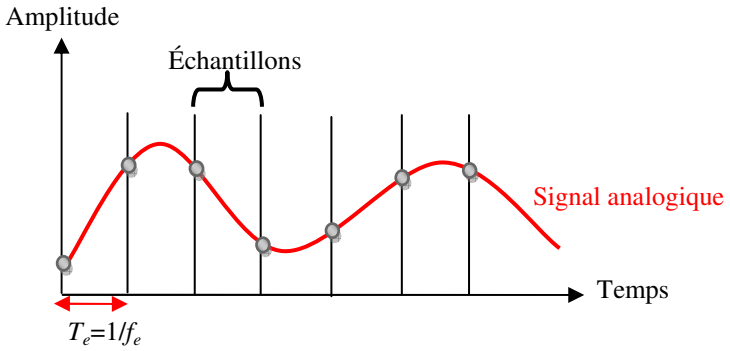


Figure 1.4 : Exemple d'un signal échantillonné.

Plus la fréquence d'échantillonnage f_e est grande, plus le nombre d'échantillons est grand, plus le signal numérique ressemble le plus au signal analogique et donc meilleure est la numérisation. En revanche, la suite binaire obtenue est plus volumineuse ce qui complique son stockage et traitement. De plus, ceci engendre une diminution de débit de transmission.

On choisit cette fréquence par l'application de théorème de Shannon (certains disent qu'il s'agit de celui de Nyquist) qui précise que la fréquence minimale d'échantillonnage d'un signal

doit être au moins le double de la fréquence maximale f_{max} contenue dans le signal à échantillonner, soit :

$$f_e \geq 2 f_{\max} \tag{1.1}$$

Si cette règle n'est pas respectée, des fréquences parasites qui n'appartiennent pas au signal de départ apparaissent. Ce phénomène est le repliement spectral ou aliasing.

Dans le cas d'un signal périodique, pour ne pas perdre d'information dans un signal, la distance entre deux échantillons doit être inférieure à la demi-période du signal :

$$T_{e2} - T_{e1} < T/2 \tag{1.2}$$

b) Quantification

La conversion analogique numérique (Figure 1.5) implique également après échantillonnage une opération qui consiste à remplacer la valeur exacte analogique de l'échantillon par la plus proche valeur approximative extraite d'un ensemble fini. Cette opération s'appelle la quantification.

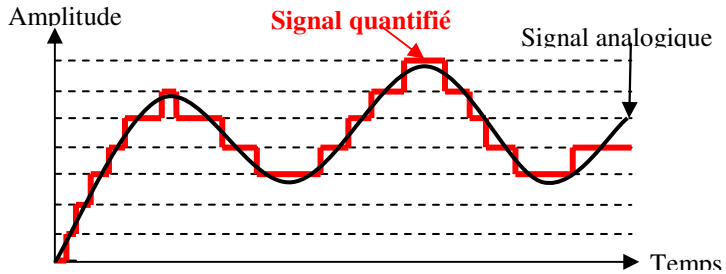


Figure 1.5 : Exemple d'un signal quantifié.

Une fois que l'amplitude du signal a été échantillonnée et qu'une valeur a été obtenue, l'étape suivante ayant pour but de réduire cette valeur à une forme numérique consiste à quantifier cette valeur en l'arrondissant à l'unité la plus proche sur une échelle de mesure étudiée spécialement.

Les paramètres d'une quantification uniforme sont :

- q : le pas de quantification (Volt),
- V : la plage de quantification (Volt),
- B : nombre de bit de la représentation numérique,

Le nombre de valeurs représentées est donnée par :

$$2^B = V/q \tag{1.3}$$

Un signal quantifié évolue ne peut prendre que de valeur de tension quantifiée. Chaque valeur isolée est ensuite convertie en grandeur binaire. Il se pose alors le problème de la précision ; chaque valeur échantillonnée sera approchée par la valeur la plus proche.

b.1) Bruit de quantification

La valeur exacte des différents échantillons n'est pas utilisée, on se contente de rapprocher chaque échantillon à une échelle de 2^B niveaux appelés échelle de quantification. Il n'y a donc qu'un ensemble de 2^B valeurs possibles pour les échantillons quantifiés. L'erreur systématique que l'on connaît en assimilant la valeur réelle de l'échantillon au niveau de la quantification la plus proche est appelée bruit de quantification. Ce bruit est déterminé par :

$$(S/B)_q = \frac{Puissance\ du\ signal}{Puissance\ du\ bruit\ de\ quantification} \qquad (1.4)$$

$(S/B)_q$: signal sur bruit quantifié.

b.2) Précision de quantification

Elle doit être adaptée au signal numérisé, c'est-à-dire que la valeur analogique maximale du signal à numériser doit être codée par la valeur numérique maximale, idem pour les valeurs minimales. Plus elle comprend de valeurs différentes, plus le codage sera précis, mais en revanche, l'information sera plus volumineuse à stocker.

c) Codage

La troisième phase est le codage, c'est-à-dire la traduction sous forme binaire d'un message de données consiste à traduire en suite binaire de 0 et 1, un cas spécial du signal numérique.

On peut transmettre le résultat de la quantification sur *b* bits. La résolution du signal obtenu en sortie va dépendre du convertisseur utilisé, autrement dit de l'électronique mise en œuvre. La limite théorique de la résolution est définie par le nombre de bits du convertisseur analogique numérique.

L'exemple de la Figure 1.6 montre un signal analogique codé sur deux bit. Il y a donc 2^2 valeurs qui seront stockées qui sont : 00, 01, 10 ou 11. Plus le nombre de bits utilisé pour le codage est important meilleure est la précision de numérisation. Cependant, la suite binaire à transmettre est plus longue ce qui diminue le débit de communication.

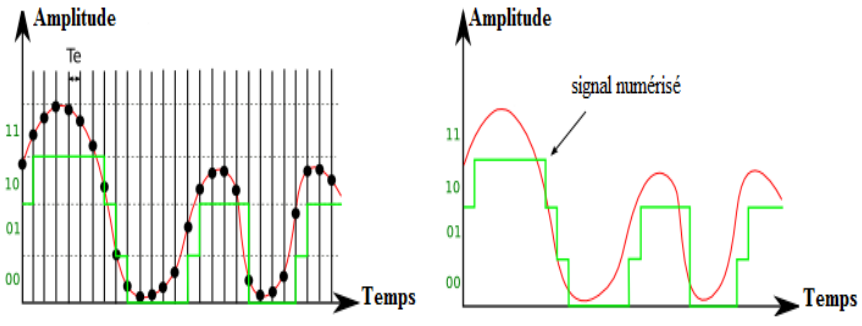


Figure 1.6 : Exemple d'un signal analogique codé sur 2 bits [3].

Après le codage de source, le signal obtenu est discret et binaire. Nous devons alors utiliser un filtre de mise en forme pour le lissage et obtenir le signal analogique à transmette à travers le canal de communication.

I.3.3 Avantages de numérisation

Les avantages de la numérisation sont nombreux. Les plus importants sont [4] :

-*Fiabilité de la transmission* : L'information transmise étant une séquence binaire, les valeurs représentées par un signal appartiennent à un ensemble discret et limité (un bit a deux valeurs, dibit à 4 valeurs, un tribit 8 valeurs...). Ainsi, contrairement à une source d'information analogique, il est possible d'utiliser des techniques de reconnaissance lors de la déformation du signal transmis (qu'il soit analogique ou numérique). Après reconnaissance par discrimination, le signal est régénéré (répété) offrant ainsi une transmission fiable.

-*Banalisation de l'information transmise* : Indépendamment de la source, l'information transmise correspond à des séquences binaires. Ainsi, le support véhicule un type d'information : le bit. Ce dernier peut aussi bien représenter du texte, de la voix, de l'image (fixe ou animée). Par conséquent, au lieu de constituer un réseau par type d'information, il est possible de constituer un réseau multi-usage.

-*Compression* : Les algorithmes de compression informatiques pourront être utilisés dès la phase de numérisation terminée. Dans ce cas, il est possible de choisir entre tel ou tel algorithme selon le contenu informationnel d'origine.

-*Cryptage* : De la même façon que la compression, il est possible d'appliquer des techniques de cryptage issues du monde informatique.

-*Protection contre les erreurs* : Lorsque l'application le permet (pas de contrainte temporelle élevée), des techniques de protection contre les erreurs peuvent être mises en œuvre offrant ainsi encore plus de fiabilité. Le codage de canal constitue la technique la plus performante dans ce contexte.

I.4 Schéma de Base d'un Système de Communications Numériques

Par rapport à l'approche purement analogique, la numérisation du signal présente de grands avantages techniques qui ont permis de développer des systèmes et dispositifs électroniques révolutionnaires en termes de nombre et de qualité de fonctionnalités diverses qui sont rendues désormais possibles et à moindre coût, et donc à la portée de tout le monde. La chaîne simplifiée d'un système de communications numériques est donnée par la Figure 1.7.

La Figure 1.7 représente la communication appelée *point-à-point* [5], c'est-`a-dire d'une seule source à un seul destinataire, et qui est la base pour les autres modèles de communications, à savoir :

-Communication *multi-utilisateurs* (plusieurs sources - un destinataire),

- Communication *broadcast* (une source - plusieurs destinataires),

- Réseaux *adhoc* (plusieurs sources - plusieurs destinataires),

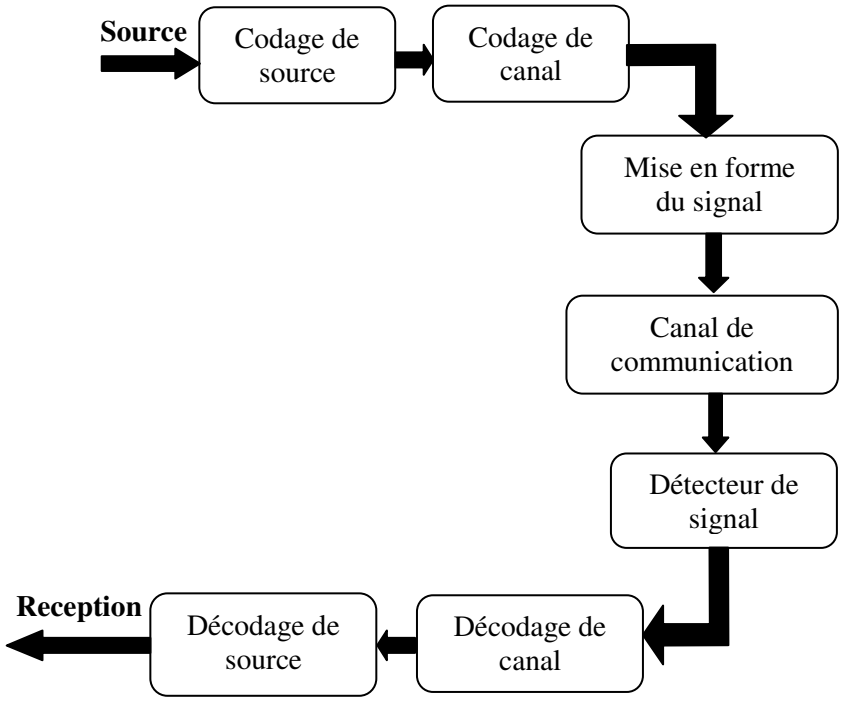


Figure 1.7 : Schéma de base d'un système de communications numériques [5].

Un système de communications numériques est constitué de :

– Source : Produit le message à transmettre,

– Codeur de source : Sert à comprimer le signal numérique en exploitant les redondances fréquentes dans le signal numérique,

– Codeur de canal : Ce bloc ajoute une information supplémentaire au signal (le codeur de canal ajoute une redondance pour protéger l'information contre les erreurs introduites par un canal de communication bruité). Le nombre de bits à transmettre est augmenté, ce qui permet de détecter et corriger les erreurs due aux imperfections du canal de transmission. On utilise des codes détecteurs et correcteurs de l'erreur tels que : Les codes cycliques (Cyclic Redundancy Check), les codes de Hamming, etc. Le lecteur est sollicité de voir Chapitre 3 pour plus de détails.

– Mise en forme du signal : Ce module (voir la Figure 1.8) convertit les bits en un signal analogique approprié capable de propager sur le canal de communication. Le filtre de mise en forme de réponse impulsionnelle $h_e(t)$ performe un lissage des valeurs binaires discrètes [5].

– Canal de communication : Le canal proprement dit représente le lien ou le support de transport de l'information entre les deux entités communicantes, mais il comprend aussi les

dispositifs en entrée et en sortie du support de transmission qui vont aider à l'émission, à la réception et à l'extraction correcte des données numériques. Pour envoyer le signal à travers le canal, la source a besoin d'un système d'adaptation (physique pour mettre en forme le signal). La Figure 1.8 présente un schéma général un canal de transmission [5].

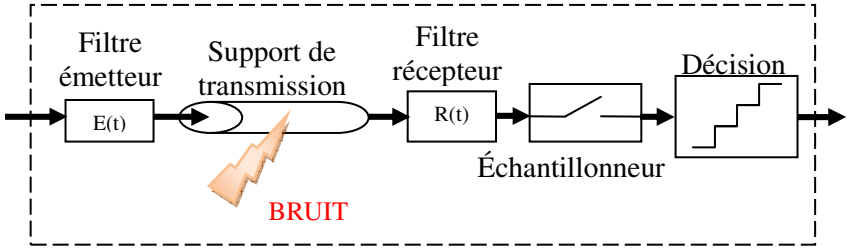


Figure 1.8 : Schéma d'un canal de transmission numérique [5].

– Détecteur de signal : se basant sur l'observation bruitée du signal, le détecteur doit décider quel message a été émis. La procédure de détection dépend des techniques de mise-en-forme utilisés, aussi que du canal de communication.

– Décodeur de canal : Ce module détecte et corrige certaines erreurs présentes dans le signal numérique fournit au récepteur. Le nombre d'erreurs corrigées et détectée dépend du nombre de bit ajouté au signal informatif lors du codage de canal.

– Décodeur de source : Ce bloc décompose le signal numérique issu de décodeur de canal et récupère le message de source.

I.5 Capacité de Canal

La capacité de canal (ou capacité de Shannon) est une quantité très importante en communications numériques, car elle établie la relation claire et détaillée entre le débit du canal, la largeur de bande de canal, la puissance du signal et la puissance du bruit :

$$D_{\max}=C=B\log_2(1+RSB) \quad (bits/sec) \tag{1.5}$$

B **:** Largeur de bande (en Hz),
RSB : Rapport signal sur bruit linéaire défini comme suit :

$$RSB=\frac{Puissance\ du\ signal}{Puissance\ du\ bruit} \tag{1.6}$$

Puisque le *RSB* est donné en général en dB**,** il faut tout d'abord le convertir en valeur linéaire :

$$RSB=10^{RSB_{dB}/10} \tag{1.7}$$

Le théorème de Shannon de capacité de canal donné par (1.5) indique que, pour un système de communication numérique de largeur de bande de canal *B* et rapport signal sur bruit moyen fixe (RSB_{dB}=constant), le débit théorique maximal de transmission pour garantir une communication sans erreurs ne peut jamais dépasser la valeur *C*, dite capacité de canal, et ceci quelque soit la complexité du système.

I.5.1 Capacité de canal perturbé par un bruit blanc additif Gaussien (BBAG)

La capacité d'un canal continu à largeur de bande B (Hertz) perturbé par un BBAG centré (AWGN en anglais pour *additive white Gaussian noise*), de densité spectrale de puissance $N_0/2$ (Watt/Hz) est donnée par :

$$C = B\log_2[1 + (P_S/P_B)] = B\log_2[1 + (P_S/P_B)] \quad (1.8)$$

I.5.2 Conséquences de l'expression de capacité de canal

La capacité de Shannon montre les trois paramètres clés du système numérique à savoir la largeur de bande B, la puissance transmise moyenne P_s, et la puissance du bruit N_0 qui sont reliés pour un bruit centré, par la forme qui suit :

$$P_B=N_0B=\sigma^2 \quad (1.9)$$

σ^2 : variance du bruit.

La capacité C dépend linéairement de la largeur de bande B et d'une façon logarithmique avec le rapport signal sur bruit P_S/P_B. Il en résulte qu'il est plus facile d'augmenter la capacité de l'information d'un canal de communication en élargissant sa largeur de bande que croitre la puissance d'émission pour une variance prédéterminée du bruit.

Remarque importante :

La supposition de canal gaussien est généralement valide en pratique. Cette conséquence est directement reliée au théorème central limite (aussi appelé théorème de la limite centrale ou centrée) qui établit la convergence en loi de la somme d'une suite de variables aléatoires vers la loi normale. Intuitivement, ce résultat affirme que toute somme de variables aléatoires indépendantes et identiquement distribuées tend vers une variable aléatoire gaussienne. Ceci est vérifié en pratique du fait que le bruit au niveau du récepteur provient de plusieurs sources aléatoires indépendantes et donc leur somme tend vers une distribution gaussienne. On parle alors de bruit gaussien additif. Notons que la sommation des sources est due au fameux principe de superposition.

Exemple : La capacité d'un système de communications numériques de RSB_{dB}=30 dB, fonctionnant sur un canal de bande passante B=10 MHz est donnée par :

$$C = B\log_2(1 + RSB) = 10^7\log_2[1 + 10^3] \quad (1.10)$$

soit :

$$C = 10^7\frac{\ln(1001)}{\ln 2} = 99.6\ \text{Mbits/s} \quad (1.11)$$

Pour augmenter le débit de transmission, on doit soit augmenter la largeur de bande (mais ceci va augmenter le cout du système), soit augmenter le rapport signal sur bruit. La capacité

augmente en logarithme de *RSB*, une croissance qui est très lente, donc pour augmenter le débit il est plus pratique et plus efficace d'augmenter la largeur de bande tout en utilisant un minimum de *RSB* (on ne peut pas se permettre de transmettre avec une puissance indéfiniment grande). Ceci explique pourquoi les systèmes de communications actuelles, comme le WiFi, fonctionnent à des fréquences de l'ordre de GHz ce qui permet d'atteindre des largeurs de bande de l'ordre de MHz. La définition du *C* permet aussi de comprendre pourquoi lorsqu'on s'éloigne de l'émetteur le débit diminue. En fait, dans une telle situation, la puissance du signal diminue tandis que la puissance du bruit augmente car le nombre d'interférences augmente ce qui réduit le débit comme exprimé par Eq. (1.5).

Pour améliorer le nombre de bits reçus sans erreurs, et donc minimiser le *taux d'erreur en bit* (TEB ou BER en anglais pour *bit error rate*) :

$$TEB = \frac{Nombre\,de\,bits\,reçus\,erronés}{Nombre\,de\,bits\,émis} \tag{1.12}$$

Le système de communication numérique doit inclure un codage de canal. Ceci est introduit dans la section suivante.

I.6 Théorème Fondamentale de la Théorie de l'Information

On définit un canal de transmission comme un système physique permettant la transmission d'une information entre deux points distants. Les critères d'évaluation d'un système de communications numériques sont :

-Efficacité spectrale définie comme étant le rapport du débit binaire D_b sur la largeur de bande *B*. Ce paramètre est à maximiser,

- Probabilité d'erreur par bit P_e (ou TEB). C'est un paramètre à minimiser ce qui conduit au théorème fondamental cité ci-après,

- Complexité du récepteur qui doit être le moindre possible afin de permettre des applications en temps-réel et éviter l'encombrement.

En 1948, Shannon énonce dans "A Mathematical Theory of Information", le théorème fondamental de la théorie de l'information ; aussi connu par le deuxième théorème de Shannon :

« Tout canal de transmission, perturbé par un bruit, admet un paramètre C, appelé capacité du canal, tel que pour tout $\varepsilon>0$ *et pour* $R< C$*, il existe un code de taux R permettant la transmission du message avec un taux d'erreurs binaire de* ε *».*

En d'autres termes, si le débit *R* d'une source discrète sans mémoire est inférieur à *C*, où *C* est la capacité du canal ; alors on peut transmettre le contenu de la source sur le canal bruité avec une probabilité d'erreur aussi petite que souhaitée. Transmettre avec un débit plus faible que *C* revient à ajouter des bits de redondance au message de source. Cette opération s'appelle

codage de canal. Il est cependant à noter que, contrairement au théorème du codage de source, le deuxième théorème de Shannon n'explicite pas le code qui permet d'atteindre les performances annoncées ni le moyen de le construire.

I.7 Notion de l'Information

I.7.1 Quantité d'Information [6]

Supposons qu'une expérience probabilistique inclut les observations d'une sortie émise par une source discrète durant une durée de temps (intervalle de signalisation).

Cette sortie de source est modélisée comme étant une variable aléatoire discrète « S » qui prend des symboles à partir d'un alphabet fini :

$$\varphi = \{s_0, s_1 \ldots s_{K-1}\} \tag{1.13}$$

avec des probabilités :

$$P(S=s_k)=p_k, \quad k=0,1\ldots,K-1 \tag{1.14}$$

Bien évidement la loi de probabilité de la source satisfait la condition :

$$\sum_{k=0}^{K-1} p_k = 1 \tag{1.15}$$

On suppose que les symboles émis par la source durant les intervalles de signalisation sont *statistiquement indépendants*. Rappelons que deux symboles s_i et s_j sont statistiquement indépendants si l'information sur s_i ne donne aucune information sur s_j. La source S remplissant ces conditions est dite *source discrète sans mémoire* ; sans mémoire dans le sens que le symbole émis à n'importe quel moment est indépendant des choix précédents.

La question est comment peut-on trouver une mesure en combien d'information est produite par une telle source ? Pout répondre à cette problématique, il est à noter que l'idée d'information est intimement liée à celle de l'incertitude ou surprise, comme décrit ci-dessous.

Considérons un évènement $S=s_k$, décrivant l'émission du symbole s_k par la source avec une probabilité p_k, comme définie en (1.14). Il est évident que, si la probabilité $p_k=1$ et $p_i=0$ pour tout $i \neq k$, alors il n'y a pas de surprise, et donc pas d'information, lorsque le symbole s_k est émis, car nous savons quel message de source devra être. Si, d'autre part, les symboles de la source se produisent avec différentes probabilités, quand le symbole s_k est émis par la source que lorsque le symbole s_i, $i \neq k$, avec une plus grande probabilité est émis. Donc, les mots incertitude, surprise, et information sont tous reliés.

Avant que l'évènement $S=s_k$ se produise, il y a une quantité d'incertitude. Lorsque l'évènement $S=s_k$ se produit il y a une quantité de surprise. Après la production de l'évènement $S=s_k$, il y a une quantité d'information et l'incertitude est donc résolue.

On définie la quantité d'information gagnée (ou acquise) après observation de l'évènement (symbole de source discrète sans mémoire) $S=s_k$ qui se produit avec une probabilité p_k ; la fonction logarithmique:

$$I(s_k) = \log\left(\frac{1}{p_k}\right) \tag{1.16}$$

D'après la définition, les propriétés suivantes sont satisfaites :

1) $I(s_k)=0$ pour $p_k=1$: Il est clair que si nous sommes absolument certains de la réalisation d'un évènement, même avant ce qu'il se produit, il n'y a aucune information gagnée car il n'y a ni surprise ni incertitude.

2) $I(s_k)\geq 0$; $0\leq p_k \leq 1$: Ceci veut dire que la réalisation d'un évènement $S=s_k$ produira soit une certaine information, soit aucune information mais elle ne résulte jamais en une perte d'information.

3) $I(s_k) > I(s_l)$ pour $p_k < p_l$: C'est-à-dire, le moins probable est l'évènement plus grande est l'information gagnée lorsque cet évènement se produit car, dans ce cas, il y aura plus de surprise et plus d'incertitude et donc plus d'information.

4) $I(S_k\ S_l) = I(S_k) + I(S_l)$,si S_k et S_l sont statiquement indépendant.

De nos jours, dans la pratique de communication, on utilise le logarithme à base de deux, l'unité résultante de l'information est appelée *bit*, la contraction de *bi*nary digi*t*. On écrit alors :

$$I(S_k) = \log_2\left(\frac{1}{p_k}\right) = -\log_2(p_k) \text{ (bits)} \tag{1.17}$$

pour $k=0, 1, 2, \ldots, K-1$

K désigne le nombre de symboles distincts émis par la source, appelé aussi le radix.

I.7.2 Notion de « bit » en relation avec l'information [6]

Pour une source discrète sans mémoire d'alphabet φ contenant deux symboles équiprobables, c'est à dire :

$$p_1= p_2=1/2 \tag{1.18}$$

On a :

$$I\,(S_k) =1 \text{ bit}, k=1,2 \tag{1.19}$$

Il en résulte la notion fondamentale suivante : Un bit est la quantité d'information qu'on gagne d'une source discrète sans mémoire lorsque l'un des deux évènements possibles équiprobables se produit.

Exemple :

Calculons l'information gagnée d'une source discrète d'alphabet φ telle que :

$$\varphi = \{s_0, s_1, s_2, s_3\},\ p_0=\frac{1}{8},\ p_1=\frac{1}{4},\ p_2=\frac{1}{8},\ p_3=\frac{1}{2} \tag{1.20}$$

En substituant dans l'équation (1.13), nous obtenons :

$$I(s_0) = \log_2\left(\frac{1}{p_0}\right) = \log_2(8) \tag{1.21}$$

soit:

$$I(s_0) = I(s_2) = 3 \text{ bits}, \tag{1.22}$$

$$I(s_1) = \log_2\left(\frac{1}{p_1}\right) = \log_2 4 = 2 \text{ bits}, \tag{1.23}$$

$$I(s_3) = \log_2\left(\frac{1}{p_3}\right) = \log_2 2 = 1 \text{ bit}. \tag{1.24}$$

D'après ces résultats numériques, nous déduisons que :

- Lorsque s_0 (ou s_2) est émis, la quantité d'information est maximale.
- Lorsque le symbole s_3 est émis, la source produit le minimum d'information car la probabilité de produire s_3 est la plus grande.

I.7.3 Définition de l'entropie [6]

$I(s_k)$ est un variable aléatoire discrète qui prend les valeurs sur l'alphabet I= {$I(s_0)$, $I(s_1)$, $I(s_2)$, …. $I(s_{K-1})$,} avec des probabilités p_0, p_1, …… p_{K-1}, respectivement. La valeur moyenne de $I(s_k)$ sur l'alphabet de source φ est donnée par :

$$H(\varphi) = E[I] = \sum_{k=0}^{K-1} p_k I(s_k)$$

$$= \sum_{k=0}^{K-1} p_k \log\frac{1}{p_k} \text{ (bits)} \tag{1.25}$$

La quantité importante $H(\varphi)$ est appelée *entropie* de la source discrète sans mémoire d'alphabet φ. C'est une mesure de contenu moyenne de l'information par symbole de source. Elle établit le lien entre l'information fournie par une source et la distribution de probabilités des symboles observés ou issus de cette source.

L'entropie d'une source discrète sans mémoire comme définie en (1.25) est délimitée par :

$$0 \le H(\varphi) \le \log_2 K \tag{1.26}$$

où pour rappel K est le radix (nombre de symboles distincts).

D'après la définition, les propriétés suivantes sont satisfaites :

1) $H(\varphi) = 0$, si et seulement si $p_k=1$ pour certaine k et les autres probabilités sont nulles. Cette limite inférieure de l'entropie correspond à la *non-incertitude*.

2) $H(\varphi) = \log_2 K$, si et seulement si $p_k=1/k$ pour tout k, c'est-à-dire, les symboles dans l'alphabet φ sont équiprobables. Cette limite supérieure de l'entropie correspond au *maximum d'incertitude*.

I.8 Codage de Source

1.8.1 Définitions élémentaires

Le codage de source sert à fournir une représentation efficace des données (un taux de compression important) tout en préservant l'information essentielle qu'elles portent. Il est employé pour le stockage ou la transmission de ces données. Le codage de source est d'autre part connecté à d'autres applications techniques telles que la classification d'images, la reconnaissance vocale, etc. On appelle données le résultat de numérisation des signaux analogiques et le dispositif qui performe ce codage est appelé codeur de source (Figure 1.10).

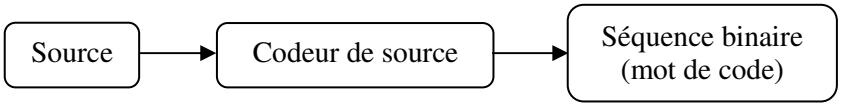


Figure 1.10 : Schéma de base d'un codeur de source [6].

1.8.2 Codes à longueur variable [6]

La loi de probabilité d'une source peut être exploitée d'une façon intuitive comme suit : Si certains symboles de la source sont connus d'être plus probable que d'autres, alors nous pouvons exploiter cette propriété dans la génération de code de source en affectant des mots de code courts aux symboles de source les plus fréquents et de mots de codes longues aux symboles rares. Ce type de code de source est connu comme code à *longueur variable* (VLC en anglais pour *variable-length code*).

Exemple : Code Morse

Dans le code Morse, les lettres de l'alphabète et les numéros sont codés par tiret «**-**» et point par «**.**». Dans l'anglais (ou français), la lettre E se produit beaucoup plus fréquent que la lettre Q. Le code Morse code « E » par un seul point, le mot le plus court en code Morse, et « Q » par « - - - », le mot de code le plus long en Morse.

Un codeur de source efficace doit satisfaire deux exigences importantes :

a) Les mots de codes produits par le codeur sont en forme binaire,

b) Le code de source est uniquement décodable, tel que la séquence originale de source pourra être reconstruite parfaitement à partir de la séquence binaire codée.

1.8.3 Longueur moyenne de code et efficacité de codage

La longueur moyenne $\bar{L}$ d'un code à longueur variable est définie en termes des probabilités des différents symboles p_k ainsi que la longueur en bits l_k codant chaque symbole l_k :

$$\bar{L} = \sum_{k=0}^{K-1} pk\ l_k \tag{1.27}$$

Soit $L_{\min}$ la valeur minimale possible de $\bar{L}$. L'efficacité de codage de source est donnée par :

$$\eta = L_{min} / \bar{L} \tag{1.28}$$

avec $\bar{L} \geq L_{min}$ et η toujours ≤ 1

Un code de source est dit parfait si : $\bar{L} = L_{min}$ ce qui correspond à $\eta = 1$.

I.9 Théorème de Shannon de Codage de Source [6]

Ce théorème s'annoce comme suit : « Etant donnée une source discrète sans mémoire d'entropie $H(\varphi)$, la longueur moyenne de mot de code $\bar{L}$ pour n'importe quel schéma de codeur de source sans distorsion est limitée comme suit :

$$\bar{L} \geq H(\varphi)\ . \tag{1.29}$$

On déduit que $L_{min} = H(\varphi)$. Ce théorème identifie alors l'entropie de la source comme étant la limite fondamentale de codeur de source pour une compression sans pertes de données. Nous ne pouvons en aucun cas construire un codeur de source ayant une longueur moyenne inférieure à l'entropie. Il s'agit d'un autre résultat fondamental en théorie de l'information. Par conséquent, l'efficacité du codeur peut s'écrire comme suit :

$$\eta = H(\varphi) / \bar{L} \tag{1.30}$$

Par ailleurs, la redondance du codage est définie comme suit :

$$r = 1 - \eta \tag{1.31}$$

Un codage est donc optimal si la redondance r est nulle, ce qui correspond à une efficacité maximale (η=1). Cependant, on verra par la suite que la détection et la correction des erreurs effectuées par un codage de canal judicieux ne peut se faire que s'il y a de la *redondance*, mais *maitrisée*.

Il est très important de noter qu'il est toujours possible de rendre la longueur moyenne de code aussi proche de la limite fondamentale $H(\varphi)$ en effectuant un codage en bloc de n symboles, où n est suffisamment grand de sorte d'avoir

$$\overline{L} = \frac{1}{n}\overline{L_n} \approx H(\varphi) \tag{1.32}$$

On parle dans ce d'une source étendue d'ordre n dont l'entropie $H(\varphi^n)$ s'exprime comme suit :

$$H(\varphi^n) = n\,H(\varphi) \tag{1.33}$$

Ce qui ramène à une efficacité de codage donnée par :

$$\eta = \frac{H(\varphi^n)}{\overline{L_n}} = \frac{H(\varphi)}{\overline{L_n}/n} \tag{1.34}$$

I.10 Compression Sans Pertes de Données

Pour une transmission efficace de signal, l'information redondante doit être enlevée (supprimé) du signal avant transmission. Cette opération s'appelle compression sans pertes de données. Parmi les codages les plus utilisés dans ce domaine on cite le codage préfixe.

I.10.1 Codage préfixe

Un code préfixe est définit comme étant un code pour lequel aucun mot de code n'est le préfixe de n'importe quel autre mot de code. Tableau 1.1 donne une illustration de code préfixe.

Tableau 1.1 : Exemple d'illustration de code préfixe [6].

Symbole de source	Probabilité de production	Code I	Code II	Code III
s_0	0.5	0 (l_0=1)	0 (l_0=1)	0 (l_0=1)
s_1	0.25	1 (l_1=1)	10 (l_1=2)	01 (l_1=2)
s_2	0.125	00 (l_2=2)	110 (l_2=3)	011 (l_2=3)
s_3	0.125	11 (l_3=2)	111 (l_3=3)	0111 (l_3=4)

Le code II est un code préfixe. Ce type de codes présente une propriété très importante : Il est toujours uniquement décodable mais l'inverse n'est pas toujours vrai. Par exemple, code III est uniquement décodable grâce au début de chaque mot de code qui doit être zéro. Cependant, ce n'est pas un code préfixe.

I.10.2 L'inégalité de Kraft-McMillan [6]

Un code préfixe doit satisfaire l'inégalité suivante, dite inégalité de Kraft-McMillan :

$$\sum_{k=0}^{K-1} 2^{-l_k} \leq 1 \tag{1.35}$$

où pour rappel l_k désigne la longueur de chaque mot de code en bits. Les constatations suivantes sont à noter :

- Un code de source qui satisfait l'inégalité de Kraft-McMillan est probablement un code préfixe (voir par exemple le code II),
- Un code de source qui ne satisfait pas cette inégalité est certainement un code non préfix.

I.10.3. Codage de Huffman

David Huffman a proposé en 1952 une méthode statistique qui permet d'attribuer un mot de code binaire aux différents symboles à compresser (pixels ou caractères par exemple). Ce code est largement utilisé souvent combiné avec d'autres méthodes de compression [7].

I.10.3.1 Propriétés du codage de Huffman

La longueur de mot de code de Huffman n'est pas identique pour tous les symboles. Il s'agit d'un code à longueur variable préfixe car aucun mot de code n'est le préfixe d'un autre. Par rapport à un codage de taille constante (c.-à-d. codage sans statistique), on observe des réductions de taille de l'ordre de 20% à 50% [7].

I.10.3.2 Principe de fonctionnement du codage de Huffman [7]

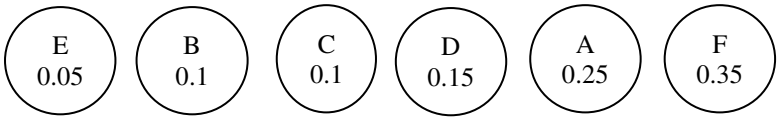
L'algorithme de Huffman opère sur une forêt, c'est-à-dire ensemble d'arbres. Il crée un arbre ordonné à partir de tous les symboles et leur fréquence d'apparition (loi de probabilité).

La construction de l'arbre se fait en ordonnant dans un premier temps les symboles par fréquence d'apparition de la plus petite vers la plus grande. Les deux symboles de plus faible fréquence d'apparition (probabilité d'occurrence) sont retirés de la liste et rattachés à un nœud dont le poids vaut la somme des fréquences d'apparition (somme de probabilités des deux symboles).

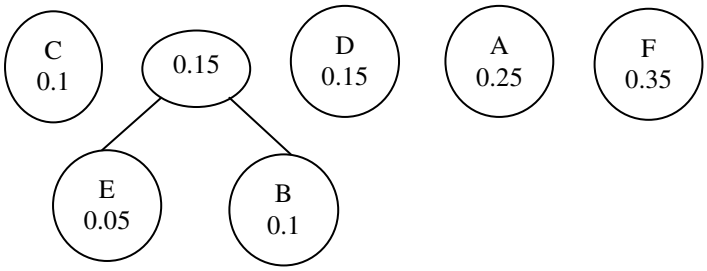
Le symbole de plus faible poids est affecté à la branche zéro, l'autre à la branche un, et ainsi de suite jusqu'à obtenir un seul nœud parent appelé racine. Le mot de code d'un symbole est obtenu en suivant le chemin depuis la racine de l'arbre jusqu'à la feuille associée à ce symbole.

Exemple 1 : Soit une source d'alphabet $\varphi = \{A, B, C, D, E, F\}$ avec la loi de probabilité :
$p(A)=0.25$, $p(B)=0.1$, $p(C)=0.1$, $p(D)=0.15$, $p(E)=0.05$, $p(F)=0.35$.
Traçons l'arbre binaire de Huffman :

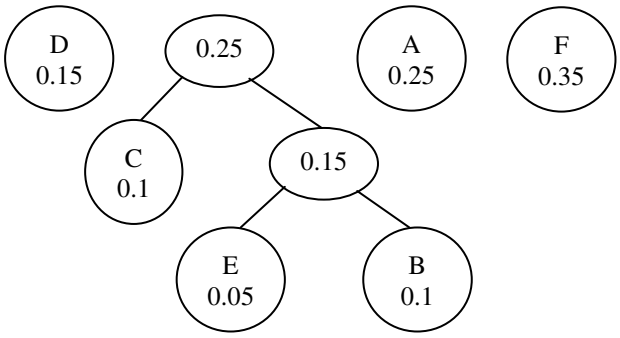
-Etape 1



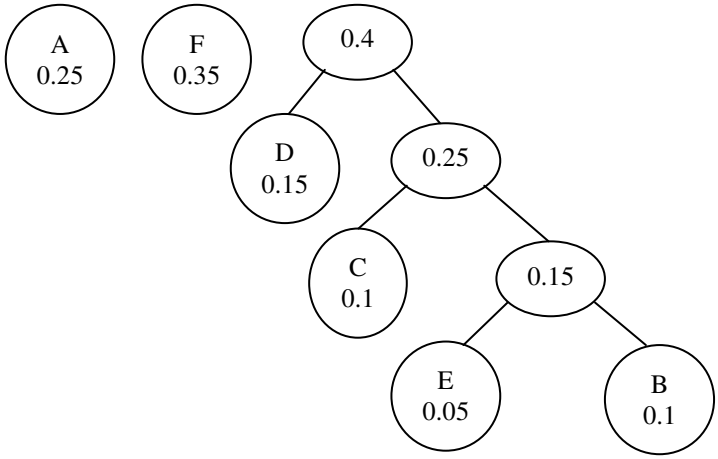
-Etape 2



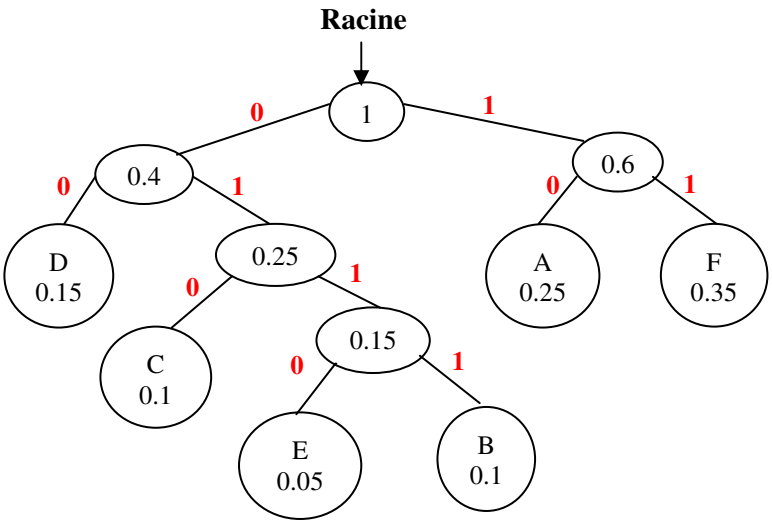
-Etape 3



-Etape 4



-Etape 5



Le Tableau 1.2 suivant contient le mot de code de chaque symbole.

Tableau 1.2 : Mots de code des caractères A, B, C, D, E, F.

Symbole	Mot de code	Longueur l_k	Probabilité p_k
A	10	2	0.25
B	0111	4	0.1
C	010	3	0.1
D	00	2	0.15
E	0110	4	0.05
F	11	2	0.35

Exemple 2 : Soit l'alphabet de source suivant : $\varphi = \{A,B\}$ tel que p_A=0.8 et p_B=0.2 codé par $\Phi(A)$=0 et $\Phi(B)$=1. L'entropie de la source est :

$$H(\varphi) = -\sum_{k=0}^{K-1} p_k \log_2 p_k = -\sum_{k=0}^{1} p_k \log_2 p_k = -0.8\log_2 0.8 - 0.2\log_2 0.2 = 0.72 \text{ bit} \qquad (1.36)$$

La longueur moyenne est :

$$\overline{L} = \sum_{k=0}^{K-1} p_k l_k = 0.8\times 1 + 0.2\times 1 = 1 \text{ bit} \qquad (1.37)$$

L'efficacité du codage vaut alors :

$$\eta = 0.722 \qquad (1.38)$$

Soit une redondance :

$$r = 1 - \eta = 28\% \qquad (1.39)$$

Autrement dit, même en choisissant le code le plus court possible (qui est, dans ce cas, de 1 bit par symbole), on perd 28% du temps de transmission puisque les bits supplémentaires n'apportent pas d'information.

Afin d'augmenter l'efficacité et réduire la redondance, nous avons vu que ceci peut toujours se faire en considérant le codage de la source étendue. Par exemple, si on code des couples de symboles au lieu des symboles eux-mêmes, on obtient ainsi (Tableau 1).

Tableau 1.3 : Mots de code de la source étendue $\varphi^2 = \{AA,AB,BA,BB\}$.

Symboles de φ^2	Probabilité	Mot de code	l_k
AA	0.64	0	1
AB	0.16	11	2
BA	0.16	100	3
BB	0.04	101	3

En reprenant l'expression (1.33), l'entropie de la source étendue d'ordre 2 est :

$$H(\varphi^2) = 2\,H(\varphi) = 1.444 \text{ bits} \qquad (1.40)$$

et la longueur moyenne devient :

$$\overline{L_2} = \sum_{k=0}^{K-1} p_k \, l_k = 0.64 + 2 \times 0.16 + 3 \times (0.16 + 0.04) = 1.56 \text{ bits} \tag{1.41}$$

Soit une efficacité :

$$\eta = \frac{H(\varphi^2)}{\overline{L_2}} = \frac{H(\varphi)}{\overline{L_2}/2} \simeq 0.93 \tag{1.42}$$

et une redondance :

$$r = 7\% \tag{1.43}$$

au lieu de 28%. En revanche, le coût à payer est une complication des opérations de codage et de décodage ainsi que l'augmentation de la complexité numérique et de l'espace de stockage.

Afin de comparer la performance de code de Huffman par rapport à d'autres types de codes, on propose un code non-préfixe, uniquement décodable pour coder la source *S* comme présenté en Tableau 1.4.

Tableau 1.4 : Exemple d'un code non-préfixe uniquement décodable.

Symbole de source	Mot de code de source	l_k
s_0	0	1
s_1	011	3
s_2	01	2
s_3	0111	4
s_4	01111	5

Ce code est uniquement décodable car le début de chaque symbole est directement détecté par le bit 0. L'inconvénient est que ce code a besoin de 5 bits pour coder s_4 pour garantir cette propriété. L'inégalité de Kraft McMillan est bien évidemment vérifiée mais le code est moins performant que le code de Huffman car sa longueur moyenne est plus grande, et donc son efficacité est moins meilleure.

I.10.3.3 Optimalité du codage de Huffman dans le cas dyadique

Le codage de Huffman possède la propriété d'être optimal dans le cas dyadique, c'est-à-dire lorsque les probabilités des symboles p_k vérifient :

$$p_k = 2^{-\lceil -\log_2 p_k \rceil} \tag{1.44}$$

où $\lceil . \rceil$ est le symbole de l'arrondissement vers le plus grand. Dans ce cas, les longueurs l_k des mots de code de Huffman vérifient :

$$l_k = \lceil -\log_2 p_k \rceil = -\log_2 p_k \tag{1.45}$$

En passant à la moyenne il vient que :

$$\overline{L} = \sum_{k=0}^{K-1} p_k \, l_k = -\sum_{k=0}^{K-1} p_k \log_2 p_k = H(\varphi) \tag{1.46}$$

Le code de Huffman est donc optimal ; il atteint l'entropie dans le cas dyadique.

Notons que le code de Huffman apparait partout en pratique : Dans les algorithmes de compression de type gzip, pkzip, winzip, bzip2 ; les images compressées de formats jpeg et png ; dans l'audio compressé mp3 ainsi que dans plusieurs algorithmes récents comme celui de Google Brotli proposé en 2014.

I.10.3.4 Avantages et limites du codage statistique préfixe : Code de Huffman comme référence

L'étude des codes statistiques préfixes permet de déduire ce qui suit :

- Le code de Huffman, comme celui de Shannon, est idéal dans le cas dyadique car il atteint l'entropie.
- Le codage de Huffman peut être toujours rendu optimal en effectuant un codage par blocs (source étendue d'ordre n). En fait, pour des petits alphabets, on a un codage de Huffman qui est de plus en plus efficace en augmentant la taille des blocs de symboles de source. Ceci permet de réaliser des longueurs moyennes proches de la limite fondamentale de Shannon qui est l'entropie (Eq. (1.32)).
- Cependant, il est désirable d'avoir des procédures de codage efficace qui fonctionnent pour des blocs de symboles plus longs. Pour cette situation, le code de Huffman n'est plus idéal puisqu'il nécessite le calcul de toutes les probabilités de K^n symboles de la source étendue afin de construire l'arbre complet du codage, où pour rappel, K est le nombre de symboles distincts. En outre, un tel codage n'est pas pratique pour des raisons multiples :
    1. La mémoire peut être indisponible pour plusieurs applications,
    2. Le décodage avec des grandes tailles de blocs sera largement inefficace et consomme un temps de traitement énorme,
    3. Une mauvaise estimation de la loi de probabilité de la source aura un impact majeur sur l'efficacité du codage effectué.
- Pour remédier à ces limites, un meilleur schéma est celui qui peut facilement être étendue à des blocs plus longs sans être obligé de refaire tous les calculs. C'est le cas du codage arithmétique faisant l'objet de la section suivante, qui est en fait une version avancée du codage de Shannon-Fano-Elias.

I.10.4 Codage arithmétique

Le codage arithmétique est un codage entropique utilisé en compression de données sans perte. Il permet une meilleure compression que le codage de Huffman, sauf en cas dyadique où les deux méthodes sont équivalentes en plus du codage de Shannon. En fait, comme nous l'avons vu en paragraphe I.10.3.4, l'un des inconvénients majeurs du code de Huffman est sa complexité en calcul et mémoire lorsque nous voulons regrouper par blocs les lettres de l'alphabet. Le codage arithmétique permet d'obtenir des performances optimales par codage de paquets regroupés avec la moindre complexité numérique.

I.10.4.1 Intérêt et principe de base

L'intérêt principal du codage arithmétique est que chaque symbole peut être codé sur un nombre non-entier de bits. L'algorithme ne code pas les données de la source symbole par symbole mais plutôt par paquets de symboles dont la longueur est définie en fonction de la capacité du calculateur à coder des nombres réels plus ou moins grands.

Ce qui différencie le codage arithmétique des autres codages sources est qu'il code le message par paquets, de taille quelconque -théoriquement- mais dans la pratique on ne peut coder que des morceaux d'une quinzaine de symboles en moyenne. Grâce à un algorithme proche du codage par intervalle, chacun des paquets de la source est représenté par un nombre flottant là où Huffman code chaque symbole par un nombre entier de bits. Le problème qui en résulte pour le codage Huffman est qu'un caractère ayant une probabilité très forte d'apparition sera codé sur au moins un bit. Par exemple, si on cherche à coder un caractère représenté à 90%, la taille optimale du code du caractère sera de 0.15 bit alors que Huffman codera ce symbole sur au moins 1 bit, soit 6 fois trop.

Un autre avantage du codage arithmétique est qu'il est un codage adaptatif pour le cas général d'une source avec mémoire. Plutôt que de supposer connue une fois pour toutes la distribution de probabilité conjointe de la source $p(x_1,x_2,\ldots,x_K)$ (par modélisation et/ou estimation sur toute la source), il est possible d'estimer au fur et à mesure les probabilités conditionnelles $p(x_i \text{ x}_1,x_2,\ldots,x_K)$. Il est à noter que le codage arithmétique est optimal dans le cas général des sources avec mémoire [8].

Proposition : Soit une source avec/sans mémoire, produisant K symboles $x_1,x_2,\ldots,x_K$ selon la distribution de probabilité $\mathbf{p}=(p_1,p_2,\ldots,p_K)$. Pour le codage arithmétique on a [9] :

$$H(\mathbf{p}) \le \overline{L} < H(\mathbf{p}) + \frac{2}{K} \tag{1.47}$$

I.10.4.2 Algorithme de compression

Afin d'illustrer l'algorithme de compression par codage arithmétique, nous proposons de compresser le mot "FARTF" à l'aide du codage arithmétique sans avoir aucune connaissance apriori de la loi de probabilité de la source à l'émission.

La première étape consiste alors à déterminer la fréquence d'apparition de chaque lettre dans le mot à coder puis déduire la probabilité d'occurrence de chacun. Nous comptons deux 'F', un 'A', un 'R' et un 'T'. Nous en générons alors une probabilité de présence dans le mot soit 40% de chance de trouver un F et 20% de chance pour trouver les autres lettres. Par la suite, nous affectons à chaque lettre un intervalle entre 0 et 1 de la manière suivante :

- La lettre 'F' a une probabilité d'occurrence de 0.4. Son intervalle est donc [0,0.4[,
- La lettre 'A' a une probabilité d'occurrence de 0.2. Son intervalle est donc [0.4,0.6[, etc.

Nous obtenons dès lors le Tableau 1.5.

Tableau 1.5 : Sous-intervalle de l'intervalle unité de chaque lettre du mot «"FARTF"» à coder.

Lettre	Probabilité d'occurrence	Intervalle
F	0.4	[0,0.4[
A	0.2	[0.4,0.6[
R	0.2	[0.6,0.8[
T	0.2	[0.8,1.0[

La deuxième étape de compression consiste à représenter le mot "FARTF" par un nombre flottant appartenant à un sous-intervalle de l'intervalle unité. L'algorithme itératif ci-dessous est appliqué :

1) À l'état initial, le mot commence avec l'intervalle unité [0,1[. On écrit alors :

$$\text{Borne inférieure (BI) du mot}=0 \tag{1.48}$$

$$\text{Borne supérieure (BS) du mot}=1 \tag{1.49}$$

2) Pour chaque lettre rencontrée, nous appliquons la mise à jour suivante :

- La borne inférieure du mot est modifiée comme suit :

$$BI=BI + (BS - BI) \times \text{Borne inférieure de la lettre} \tag{1.50}$$

- La borne supérieure du mot est modifiée telle que :

$$BS=BI + (BS - BI) \times \text{Borne supérieure de la lettre} \tag{1.51}$$

3) Le mot de source est codé par n'importe quel nombre réel appartenant à l'intervalle [BI,BS[.

Le Tableau 1.6 montre le déroulement de l'algorithme de compression sur le mot "FARTF".

Tableau 1.6 : Mises à jour des bornes inférieure et supérieure du mot "FARTF".

Lettre	Borne Inférieure	Borne Supérieure
	0.0	1.0
F	0.0	0.4
A	0.16	0.24
R	0.208	0.224
T	0.2208	0.224
F	0.2208	0.22208

Dès lors, tout nombre flottant entre 0.2208 et 0.22208 constitue le format compressé du mot "FARTF". Si par exemple on choisit le nombre 0.2208, le mot de source en binaire est donné par la suite :

$$(0.2208)_{10}=(001110001000\ldots.)_2 \quad (1.52)$$

I.10.4.3 Algorithme de décompression

De la même manière, la procédure de la décompression est aussi simple que celle de la compression. Prenons par exemple le nombre 0.2208 qui code le mot "FARTF". L'algorithme du décodage arithmétique s'effectue en deux étapes :

1) Le prochain caractère du mot est celui dont l'intervalle contient le nombre du mot actuel. Dans cet exemple, le nombre 0.2208 est dans l'intervalle de F donc la première lettre est F.

2) On modifie le nombre représentant le mot à l'aide de l'expression itérative suivante :

Nombre du mot=(Nombre du mot-Borne inférieure de la lettre)/Probabilité de la lettre (1.53)

Par exemple, après décodage du caractère F, le nouveau nombre du mot devient :

$$\text{Nombre du mot}=(0.2208 - 0.0) / 0.4 = 0.552 \quad (1.54)$$

Ce qui correspond au caractère L et ainsi de suite. Le Tableau 1.11 suivant montre les différentes étapes de la décompression appliquées sur le mot "FARTF".

Tableau 1.7 : Décompression du mot "FARTF" à partir du nombre flottant 0.2208.

Mot	Lettre	Nouveau nombre du mot
	F	0.552
F	A	0,76
FA	R	0,8
FAR	T	0,0
FART	F	
FARTF		

Ainsi, nous avons reconstitué sans pertes, sans connaissance au préalable de la loi de probabilité et avec la moindre complexité le mot "FARTF" issu de la source.

Notons que le codage et le décodage arithmétiques sont facilement performés par les fonctions prédéfinies de Matlab *arithenco* et *arithdeco*, respectivement.

I.4 Conclusion

Dans ce chapitre, nous avons examiné la transformation du signal analogique (réel) en signal numérique (artificiel) en passant par trois étapes primordiales qui sont : L'échantillonnage, la quantification et le codage. En suite, nous avons détaillé la mesure de l'information en termes de quantité d'information et d'entropie ainsi que le codage de source avec et sans pertes.

Les avantages des systèmes numériques dont la chaine est présentée en Section I.3 sont certains. Cependant, le passage au signal numérique s'accompagne d'une perte d'information puisque on ne conserve que certain nombre d'échantillons du signal analogique.

Lorsqu'il si s'agit de transmette de l'information sur un canal, l'objectif prioritaire est de minimiser la taille de la représentation de l'information. Il s'agit d'un codage de source avec compression de données. Ce codage a pour but de générer une séquence binaire avec la moindre complexité et donc avec le moindre nombre possible de bits sous contrainte de ne pas perdre de l'information. D'autre part, le codage de source a pour effet de réduire la redondance et donc améliore l'efficacité du système de communication numérique. Cependant, on cherche d'une part à maximiser le taux de compression et, d'autre part, à minimiser la complexité de mise en œuvre des algorithmes de codage/décodage ainsi que leur temps de traitement.

Dans le chapitre qui suit, nous décrivons le principe de fonctionnement d'un type de compression très important dans tous les systèmes de communications numériques : Il s'agait de Codage et Compression des Signaux Audio. Nous décrivons leur caractéristiques du son, leur propriétés, principe de compression le plus utilisé (MP3).

Chapitre II

Codage et Compression des Signaux Audio

II.1 Introduction

Le son fait partie intégrante de notre quotidien. Avant tout, il nous permet de communiquer et de recevoir de l'information, et nous pouvons nous réjouir des sons de la nature et écouter la musique. Le son peut également nous avertir d'un danger. [9]

Le type le plus courant d'enregistrement audio numérique est appelé modulation par impulsions codées (Pulse Code Modulation, PCM) et le MP3 (MPEG Layer-3). La série des MPEG-1, qui a été déclinée en trois versions (couches) : le MPEG-1 couche 1, couche 2 puis couche 3 (communément appelé "MP3"), ont permis d'atteindre des débits de plus en plus faibles, selon la nécessité de l'application (broadcast, stockage de données...).[10] Dans ce chapitre, nous présentons les différents types de codage et de compression des signaux audio en se focalisant sur les schémas de compressions avec pertes ou aussi dite destructive.

II.2 Les bases de l'audio

II.2.1 L'oreille humaine [9]

L'oreille est un organe sophistiqué, délicat et complexe qui consiste en trois parties comme présentées dans la Figure 2.1.

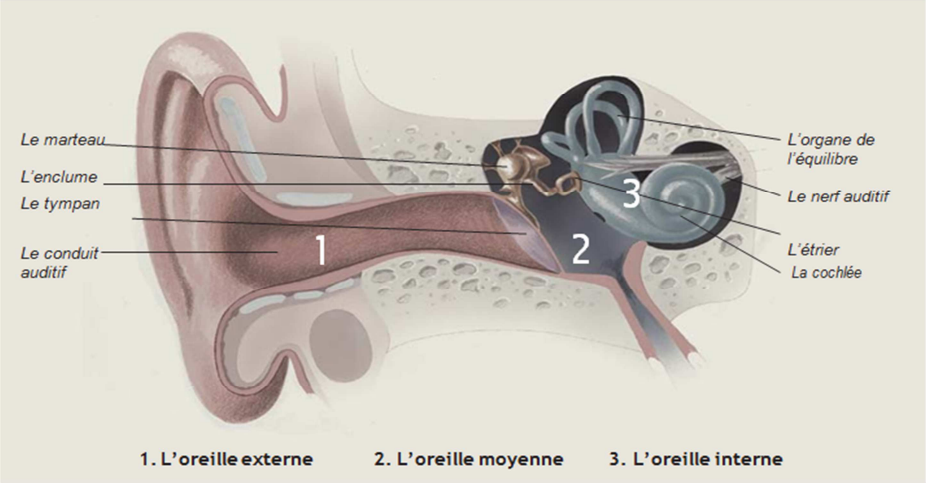


Figure 2.1: L'oreille humaine [9].

a. L'oreille externe :

L'oreille externe fonctionne comme une sorte de parabole qui capte les ondes sonores et canalise les mouvements au tympan qui est mis en vibration.

b. L'oreille moyenne :

Dans l'oreille moyenne se trouve trois osselets appelés le marteau, l'enclume et l'étrier. Ces osselets constituent un mécanisme de leviers transmettant les vibrations du tympan à l'oreille interne, également appelée la cochlée (Figure 2.2).

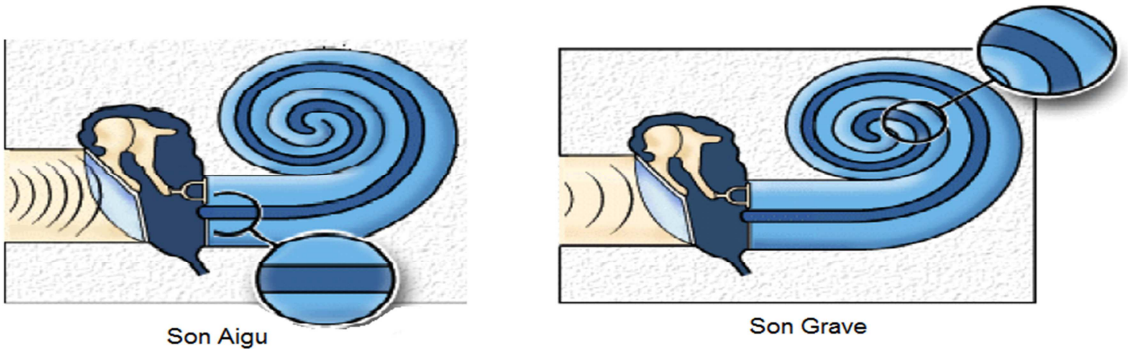


Figure 2.2: Fonctionnement de l'oreille moyenne [11].

c. L'oreille interne :

L'oreille interne est une structure osseuse remplie de liquide ayant la forme d'un colimaçon. Dans la cochlée (Figure 2.3) se trouve environ 20.000 cellules ciliées (*cellules sensorielles*) qui sont activées par le mouvement des ondes du liquide. Ces cellules stimulent les fibres nerveuses qui parviendront jusqu'au cerveau.

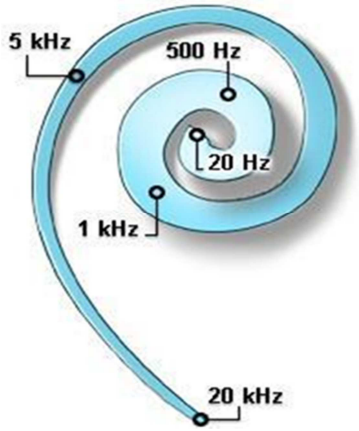


Figure 2.3: Distribution des fréquences dans la cochlée [11].

De cette façon, l'oreille arrive à capter les ondes sonores, à les transformer en mouvements dans les os, à les changer en mouvements d'ondes dans le liquide pour enfin finir comme des influx nerveux qui peuvent être interprètes par le cerveau [9].

Lorsque la pression acoustique est transmise aux liquides de l'oreille interne, l'onde de pression va déformer la membrane basilaire en un lieu qui dépend de la fréquence. Les fréquences aigües agissent à la base de la cochlée et les fréquences graves à l'apex [9].

II.2.2 Acquisition des signaux audio

La captation d'une source sonore se fait en général à l'aide d'un microphone. Les surpressions et dépressions locales de l'air, produites par la propagation du son, produisent des mouvements sur une membrane. Ces mouvements génèrent un courant électrique par induction magnétique. C'est ce courant, ou plus précisément ses variations dans le temps qui constituent le signal [12].

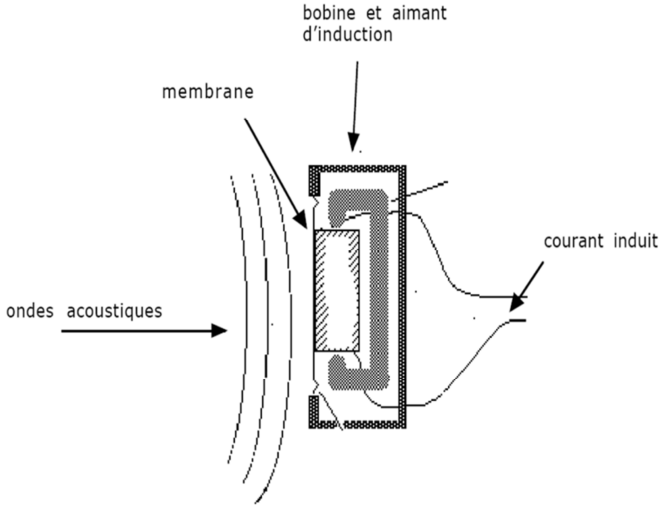


Figure 2.4: Captation du son (acquisition) [12].

II.3 Caractéristiques du son [13]

Un son est caractérisé essentiellement par trois paramètres : la hauteur, le timbre et le volume :

a. Hauteur :

La hauteur est le paramètre qui permet de différencier un son aigu d'un son grave. On dit qu'un son aigu a une fréquence élevée alors qu'un son grave a une fréquence basse.

b. Timbre :

Le timbre est le paramètre qui permet de différencier un son d'un autre son. Comme toute onde, le son peut être décomposée en une somme d'ondes sinusoïdales élémentaire qu'on appelle harmoniques. Toutes ces harmoniques additionnées donnent le timbre d'une source sonore quelconque (Figure 2.5).

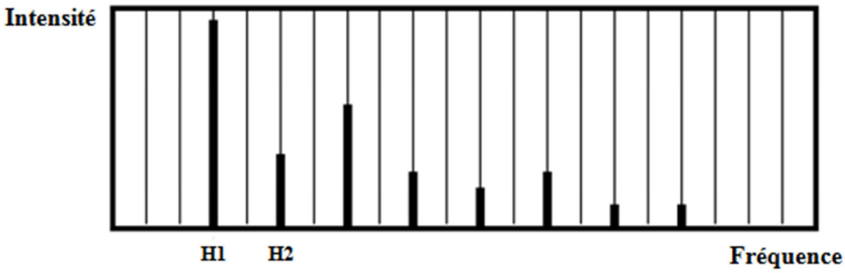


Figure 2.5: les harmoniques d'un timbre [13].

c. Volume :

Le volume est la force avec laquelle l'air frappe le tympan. Sur une forme d'onde, il se traduit par une amplitude plus grande.

II.4 Numérisation d'un signal audio sans compression :

La numérisation du signal telle qu'elle a été décrite précédemment n'introduit aucune perte, si ce n'est les défauts produits par des caractéristiques de numérisation insuffisants [14]. On parle dans le cas des signaux audio de PCM pour Pulse Code Modulation.

La modulation par impulsion codée(MIC) ou PCM est une technique de modulation consiste à échantillonner un signal continu en temps et en amplitude a des instants discrets, quantifier les échantillons en un nombre fini de niveaux et assigner un mot-code binaire a chacun de ces échantillons quantifiés [15].

La voix audio possède une bande de fréquence allant de 300 Hz à 3.4 kHz. La conversion du signal en PCM pour une transmission sur une ligne téléphonique digitale nécessite une fréquence minimale d'échantillonnage [16].

$$f_s=2\ f_e \tag{2.1}$$

$$f_s=2\times 3.4=6.8\ \text{kHz} \tag{2.2}$$

le standard utilisé est f_s=8 kHz.

Le résultat de cette mesure est codé sur huit bits. On peut donc coder 2^8 = 256 niveaux. Comme la sensibilité de l'oreille varie non selon l'amplitude d'un signal, mais selon le logarithme de cette amplitude, les niveaux d'amplitude sont découpés en plages logarithmiques. Le signal qui en résulte a un débit de 64 kbits /s [17].

Compte tenu des contraintes théoriques de restitution d'un signal analogique de bonne qualité à partir d'un signal échantillonné, La formule suivante permet de calculer la taille de stockage d'un signal audio numérique non compressé [18]:

$$D=F_e\times Q\times P\times T \tag{2.3}$$

- ✓ D : la taille de fichier en octets.
- ✓ F_e : fréquence d'échantillonnage.
- ✓ Q : le nombre d'octets utilisés pour coder un signal.
- ✓ P : le nombre de piste. (en stéréo on utilise deux pistes).
- ✓ T : le temps d'enregistrement en seconde.

Exemple : on a imposé aux CD audio un format de stockage du son ayant les données suivantes :

- ✓ La fréquence d'échantillonnage (F_e) : 44.1 KHz.
- ✓ La quantification (Q) : 16 bits (2 octets).
- ✓ Son stéréo.

Avec ce format imposé on peut calculer la place qu'occupe une minute de musique.
Numériquement en trouve :

$$D=44100\times2\times2\times60 \quad (2.4)$$

$$D = 10\ Mo \quad (2.5)$$

Il faut compter environ 40 Mo pour stocker 4 minutes seulement sur un CD au format CDA (CD Audio).

II.5 Pourquoi la compression ?

Comme on a vu plus haut, les fichiers audionumériques PCM sont volumineux. D'âpres l'équation (2.3), le débit binaire de 1.4 Mbits /sec pour un signal stéréo en qualité standard (16 bits, 44.1 kHz), produit un fichier de 630 Moctets pour une heure de musique, ceci sans compter ni les bits de correction d'erreur ni les bits de canal éventuels.

Or, les possibilités de stockage des supports sont limitées par leur densité d'enregistrement. De même, la transmission des séquences binaires audionumériques, via le câble ou via les ondes hertziennes, doit s'accommoder de canaux a bande passante forcément limitée (voir l'équation (2.3)). Cette limitation justifie aussi que l'on réduise au maximum l'information à transmettre. Le meilleur exemple en est la transmission des fichiers audio par l'internet, qui a popularisé les techniques de compression audio, et en particulier le format MP3 [19].

II.6 Types de compression

Comme son nom l'indique, la compression consiste à réduire la taille physique de blocs d'informations. Un compresseur utilise un algorithme qui sert à optimiser les données en utilisant des considérations propres au type de données à compresser; un décompresseur est donc nécessaire pour reconstruire les données originelles grâce à l'algorithme inverse de celui utilisé pour la compression. A cet effet, on distingue deux grandes catégories de compressions :

a. La compression sans pertes [19]:

La compression sans pertes (Figure 2.6(a)) est aussi appelée compression non destructive ou encore compactage. Les algorithmes de compactage sont utilisés pour compresser tous types de données : des données textuelles, des images, du son, des programmes, etc.

En plus des méthodes décrites en détails dans le chapitre 1, on note aussi l'algorithme RLE pour Run-lenght encoding. Celui-ci est utilisé lorsque les données à compresser contiennent très fréquemment des valeurs identiques. Cet algorithme est plutôt utilise dans la compression des images comme par exemple BMP, PCX et TIFF.

b. Compression avec pertes [19]:

La compression avec pertes (Figure 2.6(b)) permet d'éliminer quelques informations pour avoir le meilleur taux de compression possible, tout en gardant un résultat qui soit le plus proche possible des données originales. C'est le cas par exemple de certaines compressions d'images comme jpeg ou de sons, telles que le MP3.

Le principe de base est fondé sur l'imperfection de l'être-humain : Les données multimédias (audio, vidéo) peuvent tolérer un certain niveau de dégradation sans que les capteurs sensoriels (œil, tympan, etc.) ne discernent une dégradation significative.

La compression avec pertes est aussi appelée communément compression destructrice. Il est important de noter que les algorithmes de compression destructrice sont utilisés pour compresser des données graphiques, audio et vidéo Ils ne pourraient être appliqués sur des données textuelles ou sur celles d'un programme au risque de les rendre illisibles ou inexécutables.

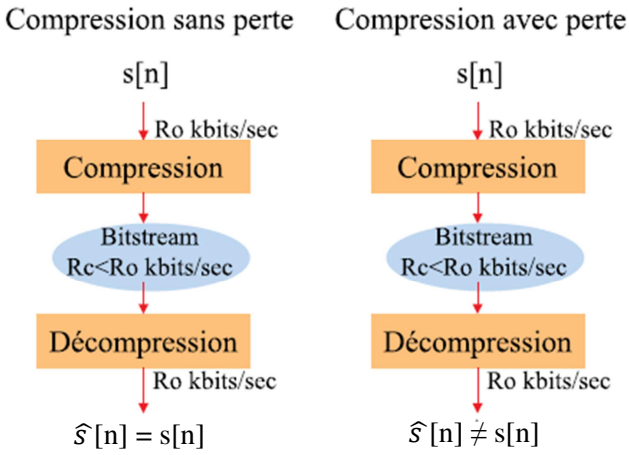


Figure 2.6 : Différence entre compression de données (a) sans pertes et (b) avec pertes. R_c est le débit binaire après compression [19].

II.7 Les formats audio [13]

a. WAV (Waveform Audio Vector):

Mis au point par Microsoft et IBM, ce format mono ou stéréo est l'un des plus répandus. Le format "Disque compact" 44.1 kHz, 16 bits et stéréo nous servira de référence pour le calcul du poids et ratio des autres formats. Le suffixe des fichiers créés est .wav.

b. MP3 (MPEG-1 Layer III):

MP3 est l'abréviation de Mpeg- 1 Audio Layer 3. Cet algorithme de compression prend naissance en 1987. L'ISO en fera un standard dans les années 92-93. Les taux de compression (ratio) sont d'ordinaire de 1 pour 10 (1:10) (1:4 à 1:12). Ceci est obtenu par suppression de zones dans la bande passante dont les fréquences sont inaudibles par l'oreille humaine.

c. AAC (Advanced Audio Coding) ou MPEG-2 AAC:

Il fait partie des successeurs du MP3. L'AAC, est une extension du MPEG-2 et a été amélioré en MPEG-4, MPEG-4 Version 2 et MPEG-4 Version 3. Il a été reconnu fin avril 1997. Le suffixe des fichiers créés est .aac, .mp4, .m4a.

d. WMA (Windows Media Audio):

Créé par Microsoft à partir des recommandations MPEG 4 en 1999, ce format est utilisé par le logiciel Windows Media Player. Ce format est lié à une gestion pointue des droits d'auteurs. Le suffixe des fichiers créés est .wma.

e. Real Audio :

Format propriétaire adapté aux débits limités d'Internet pour la diffusion de sons ou de vidéos en streaming. Le suffixe des fichiers créés est .ra .rm ou .ram.

f. Le format MIDI :

MIDI (Musical Instrument Digital Interface) est une norme développée par les fabricants de synthétiseurs au début des années 1980 pour permettre l'interconnexion de 16 instruments de musique électroniques et on indique sur quelle note jouer, sur quel instrument, à quel volume, etc.

II.8 Techniques de compression d'un signal audio avec pertes

Comme on l'a déjà cité, les systèmes de compression sans perte peuvent rarement atteindre des taux de compression supérieurs à 2 :1, alors que les systèmes avec pertes peuvent atteindre des taux supérieurs à 10 :1. Pour permettre de telles performances, ces derniers font appel à des modèles de type psychoacoustique, on les appelle dés lors des codes perceptuels [19]. Les termes codeur/décodeur ou codec sont souvent utilisés (Figure 2.7).

Le principe de compression de sons est de retirer des données audio les fréquences inaudibles par l'oreille humaine capable de discerner des sons entre 0.20KHz et 20 KHz, ainsi que les sons masqués. Ce principe sera détaillé dans la section suivante. En fait, le modèle perceptuel est fondé sur un modèle mathématique qui tente de décrire comment l'oreille humaine, avec ses imperfections, réagit avec le son, pour but d'éliminer régulièrement l'information qui est inaudible ou devenue inaudible.

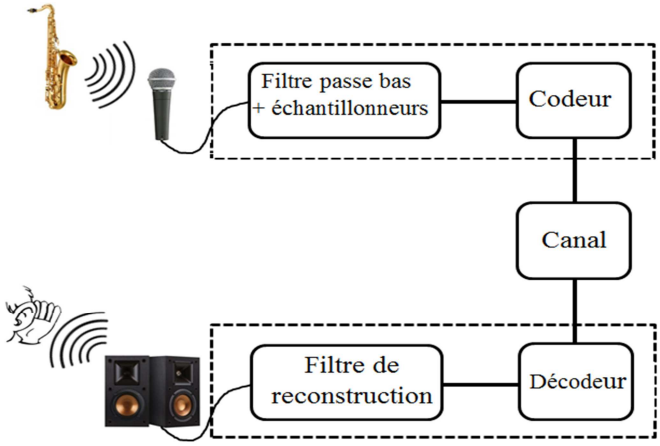


Figure 2.7 : Système de compression [11].

L'information exacte à éliminer dépend du codec utilisé, Certains codecs sont faits pour enlever certaines fréquences pour que la compression soit meilleure pour les voix [20]

II.8.1 Le modèle psycho-acoustique

Afin de retirer l'information audio non-pertinente, il faut pouvoir la déterminer, ce qui concrètement signifie qu'il faut pouvoir détecter dans le signal les parties sonores inaudibles. Un petit détour par la Psychoacoustique n'est donc pas inutile. Cette science étudie la manière dont nous percevons les sons et la manière dont nous réagissons en leur présence. En particulier, la psychoacoustique s'intéresse aux conditions d'audibilité d'un son [19].

II.8.1.1 Seuil d'audibilité :

Les volumes audibles sont entre 0 à 120 dB (voir le Diagramme de Fletcher de la Figure 2.8) mais, le seuil de sensibilité dépend de la fréquence. Les parties dont les volumes sont inférieurs au seuil ne sont pas audibles.

Le seuil fréquentiel d'audibilité (Figure 2.9) était déterminé par Wegel sur la base d'une étude menée sur plusieurs sujets jeunes et sains. Ce seuil est illustré aux basses fréquences qu'aux fréquences moyennes entre 1 et 4 KHz.

Tous les sons dont le spectre est en dessous de cette courbe sont inaudibles, ils peuvent donc être retirés du flux audio [19].

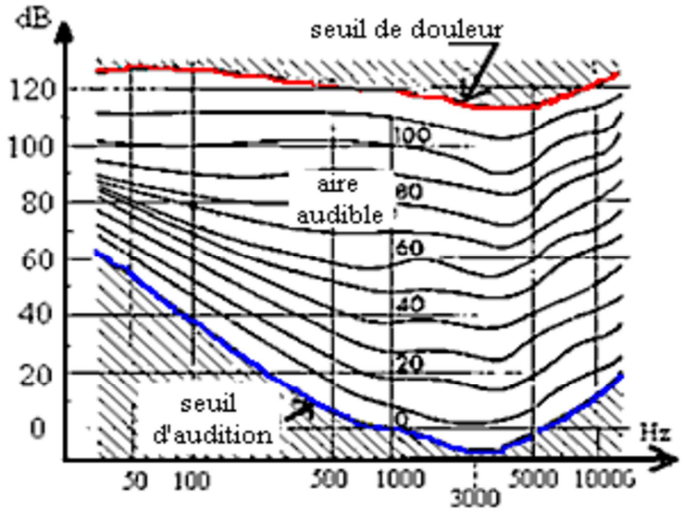


Figure 2.8: Diagramme de Fletcher [11].

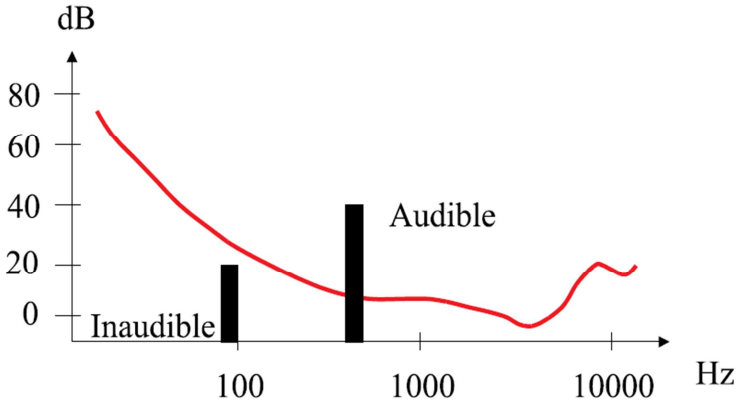


Figure 2.9 : Le seuil d'audibilité [19].

II.8.1.2 Masquage fréquentiel [11] :

Le seuil d'audibilité est modifié en présence d'un son masquant. On parle de masquage fréquentielle comme illustré dans la Figure 2.10.

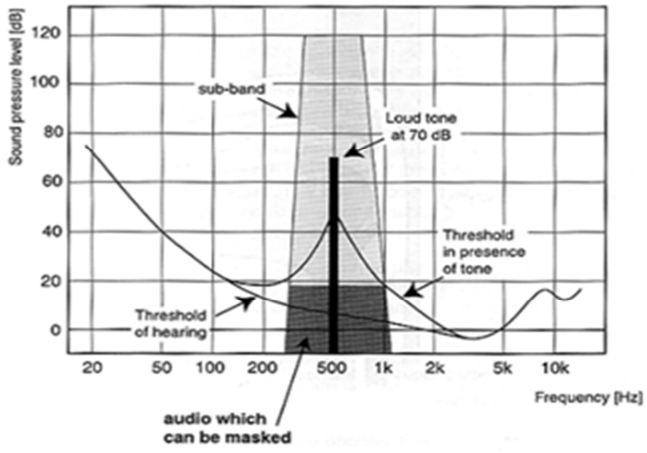


Figure 2.10: Masquage fréquentiel [11].

On voit clairement que le ton à 70 dB modifie le seuil d'audibilité de telle sorte que des sons audibles deviennent inaudibles. On dit que le ton à grande amplitude spectrale peut cacher ou maquer d'autres sons. On parle de masquage fréquentiel [11].

II.8.1.3 Masquage temporel [11] :

Le phénomène de masquage s'exprime aussi temprellement : un son de forte intensité masque les sons d'intensité plus faible immédiatement postérieurs (post-masquage) et antérieurs (pré-masquage) comme illusté en Figure 2.11.

Le pré-masquage concerne un intervallle de temps de quelques millisecondes seulement, alors que le post-masquage s'étend sur plusieurs dixaines de millisecondes.

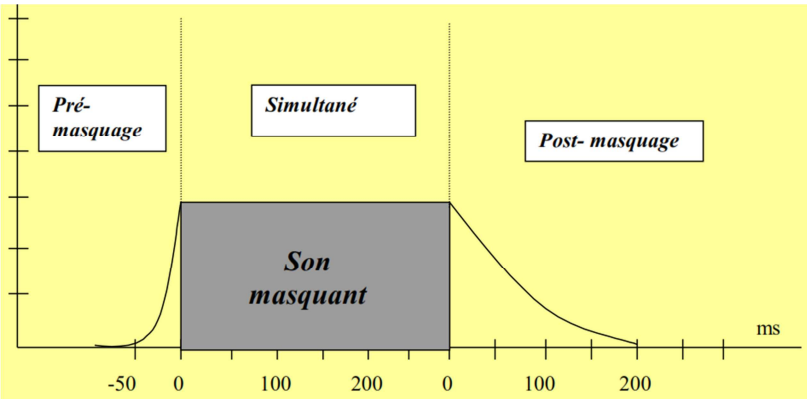


Figure 2.11 : Masquage temporel [11].

II.8.1.4 Bandes critiques [11] :

Fletcher, puis Zwicker, ont mis en évidence l'existance de bandes fréquentielles critiques appelées barks. Ils en ont dénombré 24 sur l'ensemble du domaine audible.

II.8.1.5 Fonctionnement d'un algorithme de compression de type perceptuel [11] :

En fonction de ce qui procède, un algorithme de type perceptuel permet de réaliser les tâches décrites dans la Figure 2.12.

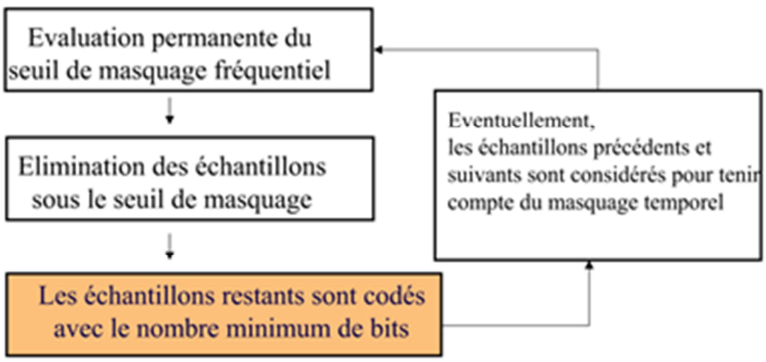


Figure 2.12 : Fonctionnement d'un algorithme de compression de type perceptuel [19].

Remarquons que l'algorithme de la Figure 2.12 demande l'évaluation permanente du seuil de masquage, à intervalles de temps réguliers. De plus, les échantillons du signal dont l'intensité est supérieure au seuil de masquage ne sont pas encodés complètement, mais avec un nombre de bits réduit au minimum.

II.8.1.6 Masquage du bruit de quantification- réduction de nombre de bits de quantification [11] :

Les échantillons dont l'intensité est au-dessous du seuil de masquage sont éliminés du flux binaire. On démontre que la puissance du bruit de quantification est égale à $Q^2/12$, avec :

$Q=A/2^N$ est le pas de quantification.

$A=(V_{max}-V_{min})$: intervalle de variation des valeurs des échantillons.

N : Nombre de bits utilisés pour coder l'échantillon.

Coder avec moins de bits revient à augmenter le bruit de quantification. On peut démontrer que le rapport signal sur bruit de quantification vaut (en dB) :

$$RSB(dB)= 6.02\ N+1.76 \qquad (2.6)$$

II.8.1.7 Codage par sous-bandes fréquentielles (sub-bande coding) [11] :

Les essais effectués ont montré que la considération d'un bruit de quantification identique de 20 Hz à 20 KHz n'est pas une solution acceptable : Il est nécessaire alors de travailler par bandes fréquentielles.

En fait, le spectre complet du signal est découpé en 32 intervalles de largeurs égales. Il correspond à ce découpage 32 filtres de type passe-bande. Ceci est illustré dans la Figure 2.13.

Pour conserver le même débit binaire (bitrate), chaque signal filtré est ensuite ramené à la fréquence d'échantillonnage Fs/32 (sous-échantillonnage) . Fs : une Fréquence d'échantillonnage donnée (32 KHz, 44.1 KHz, 48 KHz ou autre).

Figure 2.13 : Technique du banc de filtres de largeurs égales [11].

II.8.1.8 Déroulement d'un algorithme perceptuel à base du modèle psychoacoustique [13]:

L'algorithme permet de réaliser deux rôles distincts :

a. Un rôle filtreur

Eliminer du signal toutes les fréquences inférieures à 15 Hz et supérieures à 20 KHz.

b. Un rôle échantillonneur

- Diviser le spectre audible en 32 bandes de 720 Hz chacune,
- Comparer la valeur moyenne de chaque bande à la valeur correspondant sur la courbe de réponse de l'oreille,
- Toute bande en dessous de la courbe de réponse est éliminée,
- Chaque bande retenue est divisée en 12 sous-bandes échantillonnées sur 16 bits,
- On compare encore une fois la valeur des sous-bandes à la courbe de référence,
- On quantifie les différences à la courbe, et on code ces différences sur un nombre minimum de bits,
- Ces opérations sont renouvelées tous les 8 millièmes de seconde.

Exemple d'illustration : Prenons l'exemple de la Figure 2.14.

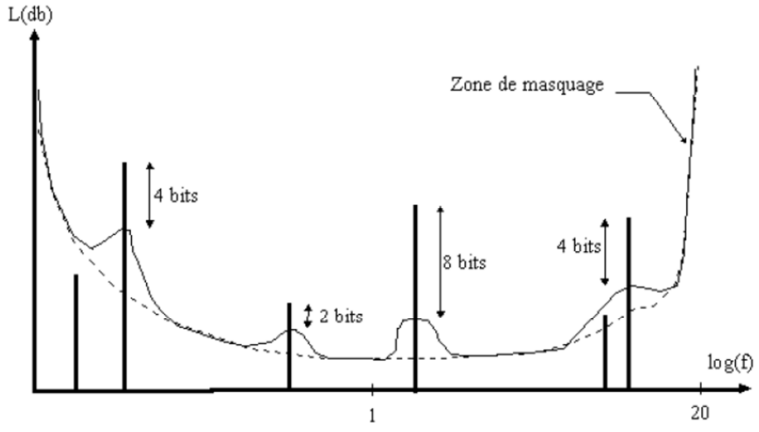


Figure 2.14 : Quantification des niveaux de fréquences audibles [13].

On distingue les propriétés suivantes :

- Nous avons 6 bandes (traits verticaux) qui traversent ou non la zone de masquage,
- Deux d'entre eux restent sous la courbe du seuil d'audibilité modifié et donc elles sont brejetées,
- Les quatre autres sont échantillonnées en 12 autres sous-bandes,
- Pour chaque sous-bande on afectera un nombre de bits spécifique alloué au codage des échantillons comme expliqué dans la section II.8.1.6.

II.8.2 La compression MPEG Audio

a. L'histoire

Le MP3 est une technique de compression des formats audio-numériques. Le brevet en a été déposé par l'institut allemand de recherche Franuhofer. Au départ les chercheurs de cet institut travaillaient sur la compression de séquences vidéo. Les chercheurs de l'nstitut ont traillé, dés le depart, en accord avec les normes établies par le MPEG (Moving Pictures Experts Group), un groupe d'experts au sein de l'ISO définissant le standard international pour la compression vidéo. Le MPEG-1 Layer 1 fut la 1^{ére} norme a voir le jour. La 3^{éme} versionde cette technique est la MPEG-1 Layer 3 dont la variante spécifiquement audio est la norme MPEG-1 Audio Layer 3.[18]

b. MPEG-1

Il existe 3 modes de compression, ce sont les Layers I, II, et III exigeant plus ou moins de temps de calcul. Le choix d'un très bas débit entraîne inévitablement une perte de qualité du signal original, en particulier dans la diminution de la bande passante du signal restitué. [21]

Le schéma de base représenté dans la figure 2.15.

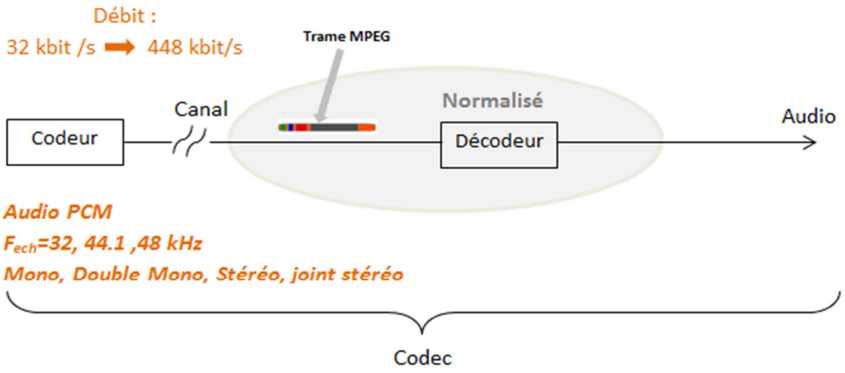


Figure 2.15 : Schéma de base du MPEG 1. [11]

c. Les couches (layers) [22]

On distingue trois couches (Figure 2.16)

Layer I : Est le plus simple qui vise une utilisation domestique. Il utilise un filtrage à 32 sous-bandes de même largeur, une allocation de bits adaptative, et une compensation de bloc. En fonction de la complexité de l'encodeur, une haute qualité audio (proche du CD), implique un débit entre 256 et 384 Kbps pour un programme stéréo. Layer I est notamment utilisé dans le système de cassette numérique DCC.

Layer II : permet une compression plus élevée que Layer I et se destine aussi bien aux domaines audio domestique que professionnel, comme l'émission radio et télévision et les

télécommunications. En fonction de la complexité de l'encodeur, une haute qualité audio (CD) implique un débit entre 192 et 256 Kbps pour un programme stéréo.

Layer III : ajoute des filtres hybrides, une quantification non-uniforme, et un codage de type Huffman. C'est le mode qui offre le plus de compression. Layer III étend les applications du MPEG dans les télécommunications à bande étroite ou bien à certains domaines spécialisés de l'audio professionnelle. Il a été banalisé par l'usage des fichiers MP3.

Couche 1	Couche 2	Couche 3
Mode : mono, double mono, stéréo, joint stéréo (intensité)	**Mode :** mono, double mono, stéréo, joint stéréo (intensité)	**Mode :** mono, double mono, stéréo, joint stéréo (intensité)
Débit « transparent »	**Débit « transparent »**	**Débit « transparent »**
192 kbit/s (mono)	128 kbit/s (mono)	64 kbit/s (mono)
384 kbit/s (stéréo)	256 kbit/s (stéréo)	128 kbit/s (stéréo)
Complexité :	**Complexité :**	**Complexité :**
Codeur : +	Codeur : +	Codeur : + + +
Décodeur : -	Décodeur : -	Décodeur : +
Délai : -	Délai : +	Délai : +
Application :	**Application :**	**Application :**
MultiMedia, DCC	Broadcast (satellite)	MultiMedia (mp3)

Figure 2.16 : Couches 1, 2 et 3 du standard MPEG-1 de compression audio [11].

d. Schéma du codeur couche 1/2 (Figure 2.17):

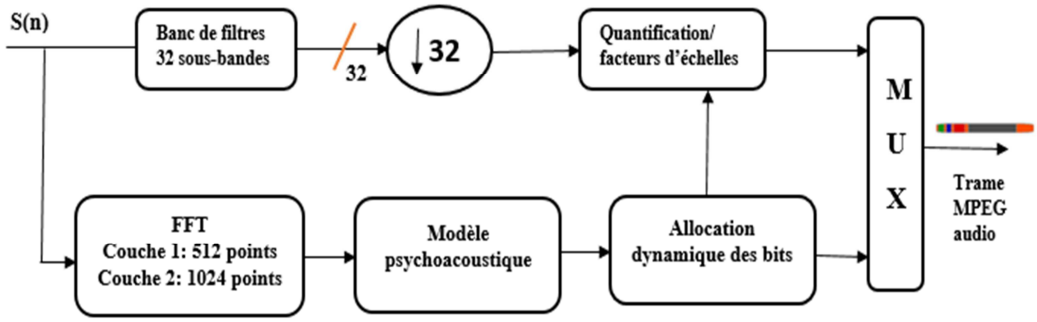


Figure 2.17 : Schéma du codeur couche 1/2 [11].

e. Schéma du décodeur couche 1/2 (Figure 2.18):

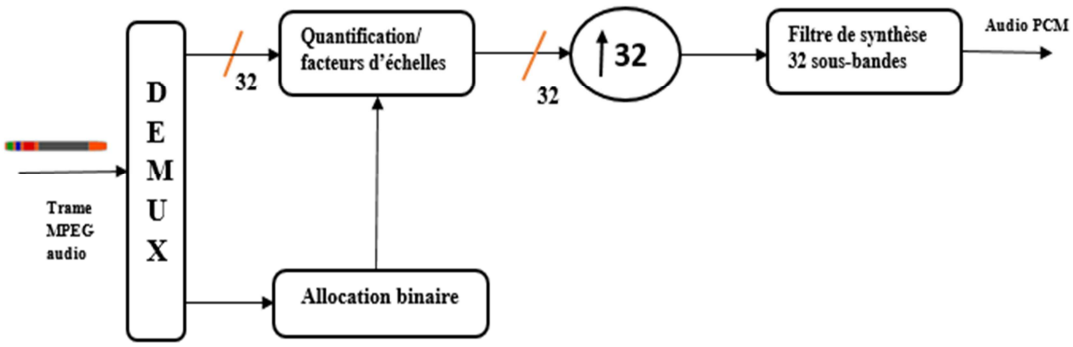


Figure 2.18 : Schéma du décodeur couche1/2 [11].

f. Trame MPEG audio couche 1/2 :

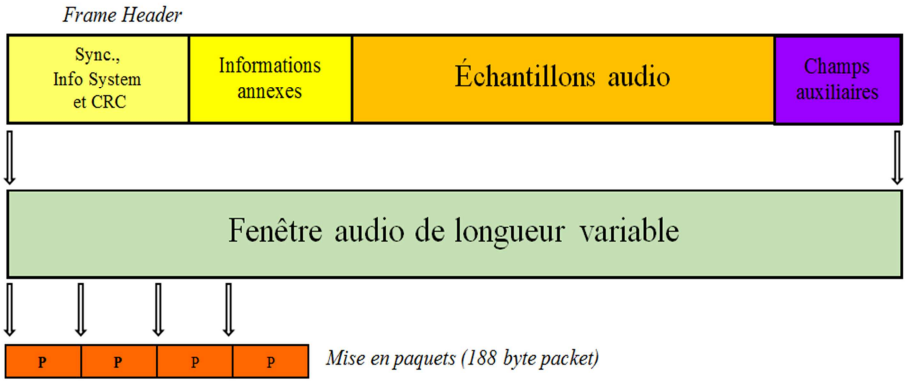


Figure 2.19 : Trame MPEG audio couche 1/2 [11].

Layer 1 :
Information principale = 32 × 12 échantillons = 384 PCM échantillons = 8 ms à 48 kHz
Layer 2 :
Information principale = 32 × 36 échantillons = 1152 PCM échantillons = 24 ms à 48 kHz

g. Schéma du codeur MPEG1-couche 3 « mp3 » :

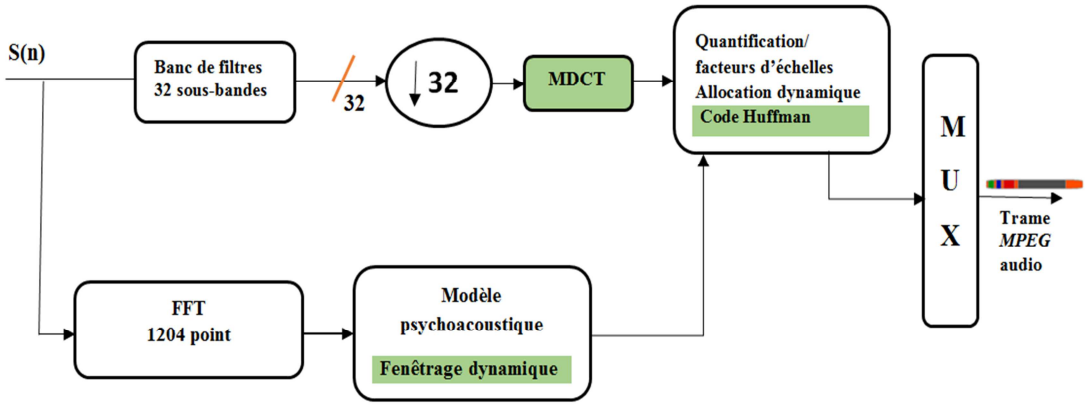


Figure2.20 : Schéma du codeur mp3 [11].

h. Schéma du décodeur MPEG1-couche 3 « mp3 »

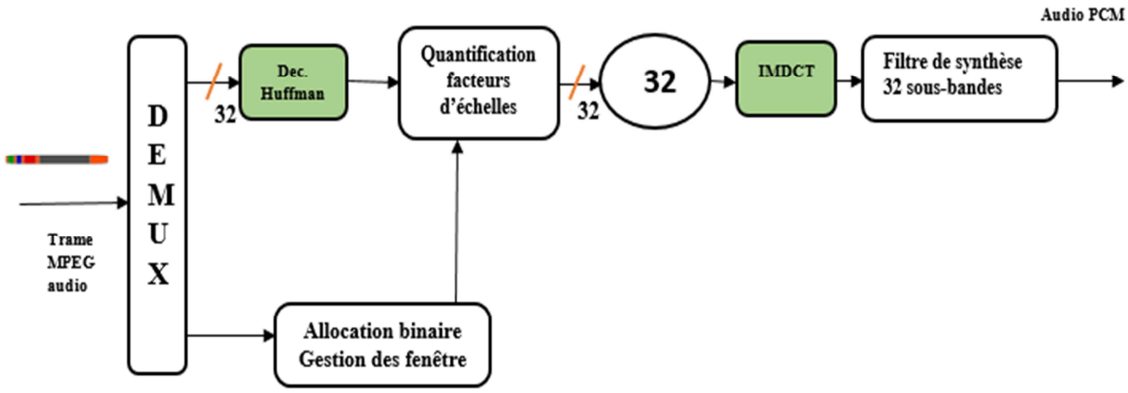


Figure 2.21 : Schéma du décodeur mp3 [11].

D'après les Figures 2.20-2.21, on constate que le codec mp3 inclue deux blocs supplémentaires comparé au MPEG audio couches 1 et 2 : La transformée de cosinus discrète largement utilisée dans la compression destructive d'images (exemple JPEG) et le codage de Huffman effectuant une compression sans pertes. Ceci explique le taux de compression très important de ce format.

II.9 Conclusion :

La compression du son est massivement utilisée dans tous les domaines. Ainsi nous avons à notre disposition de très nombreuses techniques, chacune optimisée pour une application bien précise. Si dans certains cas on peut se poser la question de son intérêt dans le futur, il parait évident que ces techniques vont encore progresser et toujours nous permettre d'améliorer les réseaux. En gagnant des octets par la compression, on se donne les moyens de rajouter toujours plus de fonctionnalités et ainsi d'offrir de nouvelles perspectives aux utilisateurs [10]. Parmi Ces fonctionnalités on cite : la détection et la correction des erreurs, cryptographie….

Dans le chapitre qui suit, nous décrivons le rôle et le principe de fonctionnement d'un bloc très important dans tous les systèmes de communications numériques : Il s'agit de codage de canal avec l'attention focalisée sur les notions théoriques les plus fondamentales ainsi que les propriétés des codeurs linéaires en blocs.

Chapitre III

Codage de canal par les codes linéaires en blocs

III.1 Introduction

La présence inévitable de bruit à travers le canal de transmission cause des erreurs entre les séquences émises et les séquences reçues d'un système de communication numérique.

Le codage de canal des systèmes de communications numériques (CNs) est une étape primordiale dans le traitement de l'information numérique. Le récepteur reconstitue les informations émises en performant l'opération inverse : le décodage de canal.

Le but du codeur de canal est de réduire la probabilité d'erreur due au bruit inévitable présent sur le canal (interférences, parasites, dégradation, distorsion, bruit thermique, etc.) en ajoutant des bits au message de source appelés bits de contrôle ou bits redondance. Ceci permet de réaliser un taux d'erreur en bits (TEB) convenable pour le fonctionnement du système conçu. On dit alors que le rôle de codage de canal est de rendre la transmission plus *fiable* tandis que le rôle de codage de source est d'améliorer l'*efficacité* de la numérisation de la source analogique en incluant très souvent une compression sans pertes de données.

Puisque les systèmes actuels exigent des valeurs très petites de TEB (téléphone fixe :10^{-4}, communications optiques : 10^{-9}, transmission vidéo numérique jusqu'à 10^{-12}) ; la recherche de codes « parfaits » comme annoncés par le théorème de Shannon de codage de canal constitue un champ de recherche ouvert et prometteur. La fiabilité du codage effectué est évaluée en mesurant le TEB sur un canal bruité sans et avec codage de canal. Autant que le TEB est petit autant que le codeur de canal est plus performant. Nous verrons par la suite que le code détecteur de l'erreur est de plus en plus fiable que sa complexité augmente, ce qui revient à augmenter le nombre de bits de contrôle ainsi que compliquer l'algorithme utilisé pour les générer. Cependant, deux inconvénients majeurs persistent par conséquent : Diminution de débit et complexité élevé de décodeur.

III.2 Principe générale de codage de canal

L'opération de codage de canal consiste à ajouter les bits de contrôles à l'information pour améliorer la *fiabilité*, et pour cela il y'a plusieurs méthodes qui sont proposées, comme par exemple la méthode de division euclidienne qui sert à obtenir un mot de code d'un code polynomial de polynôme générateur g(x) associé au mot initial.

Toujours dans les opérations de codage, il y'a une autre méthode appelée « méthode matricielle » son principe est : pour encoder un message on le multiplie par une matrice génératrice *G*, on obtient un mot de code, ensuite après passage par un canal bruité, une erreur *e* s'ajoute au mot de code. On s'intéresse tout d'abord à la matrice génératrice des codes *systématiques*.

Dans le codage de canal il y a deux grandes catégories qui sont *les codes détecteurs d'erreur* comme le code cyclique de redondance, et *les codes détecteurs et correcteurs d'erreur* comme le code de Hamming. Ces erreurs sont dues principalement à la dégradation de l'information lors de propagation sur le canal, l'affaiblissement (rapport signal sur bruit diminue) du signal émis, interférences, bruit… Cependant, pour un canal bruité (exemple : canal de communication sans fils), le TEB requis est très faible pour assurer une fiabilité acceptable de transmission.

Exemple :

La probabilité d'erreur sur une ligne téléphonique est $p=10^{-4}$ et peut même atteindre 10^{-7}. En fibre optique, on peut actuellement atteindre des *TEB* de l'ordre de 10^{-12}. Il est évident que à force que le *TEB* diminue, le codage de canal devient plus fiable (Figure 3.1).

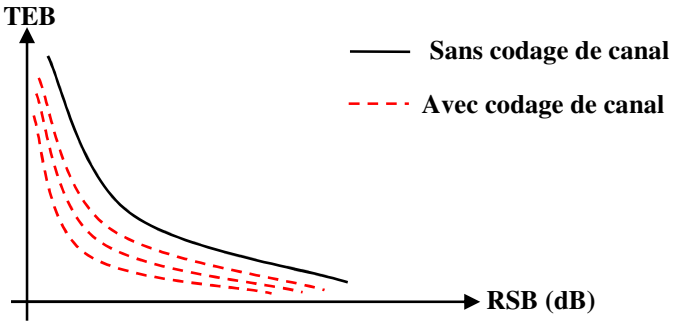


Figure 3.1 : TEB en fonction de RSB avec et sans codage de canal.

III.2.1 Principe générale

Chaque suite des bits à transmettre (trame, signal utile portant l'information de la source) est *augmentée* par une suite de bits de redondance ou de contrôle.

Pour chaque suite de k bits transmis, on ajoute r bits. On dit alors que l'on utilise un code $C(n,k)$.

Où :

n : nombre de bits du message après codage de canal (longueur du mot de code)

k : nombre de bits du message (longueur du message de source)

$r=n-k$: nombre de bits de contrôle (ou de redondance).

III.2.2 Probabilité d'erreur en une suite binaire

Soit une séquence binaire de n bits, soit p la probabilité qu'un bit soit mal transmis. La probabilité d'avoir r bits erronés parmi n bits transmis est :

$$p_r=C_n^r\, p^r(1-p)^{n-r} \tag{3.1}$$

avec

$$C_n^r = \frac{n!}{r!(n-r)!} \tag{3.2}$$

III.2.3 Probabilité d'erreurs détectées

Les messages erronés n'étant pas détectés dans tous les cas, il est nécessairement de calculer une probabilité de symbole les reconnaître [8].

Rappelons qu'un message est erroné s'il n'est pas identique au mot de code émis, (même si la partie qui comporte l'information n'est pas erronée).

Supposons que :

- La probabilité d'erreur soit la même, p, pour chaque bit (la valeur de p est évidemment très petite) ; la probabilité qu'un bit soit correctement transmis est donc :

$$q = 1- p \tag{3.3}$$

- Les erreurs sur les bits soient indépendantes les unes des autres (ce n'est pas toujours le cas) ; le modèle de la situation est alors « un schéma de Bernoulli ».

Exemple : Soit X le nombre de bits erronés dans un message, X vari de 0 à $n = 8$ en suivant la loi binomiale de paramètres n et p.

Une configuration où k bits sur n sont inexacts à une probabilité $p^k (1-p)^{n-p}$.

Si C_n^r désigne le nombre de ces configurations, la probabilité $p(k)$ que l'erreur d'un message porte sur positions est donc :

$$p(k) = p_r(X = k) = C_n^k \, p^k \, (1-p)^{n-k} = C_8^k \, p^k \, (1-p)^{8-k} \tag{3.4}$$

D'où la probabilité $p\,(0)$ que la transmission soit parfaite :

$$p(0) = p_r(X = 0) = q^8 \tag{3.5}$$

Et la probabilité d'erreur d'un message :

$$p_{err} = 1- q^8 \tag{3.6}$$

III.3 Codage en bloc

Un code par blocs code tous les mots de longueur k par certains mots de longueur n, supérieur à k, noté $C(n,k)$. Le nombre n est appelé longueur de mot de code [8] [22].

Toute suite binaire de longueur n peut apparaître à la réception, nous lui réserveront dorénavant le terme de message, qu'elle représente ou non un mot de code.

Bien qu'absolument nécessaire, la redondance alourdit la transmission en temps et charge la mémoire du système, il est donc important d'évaluer le rendement nommé encore taux de transmission du code donné par le rapport :

$$r = k / n \tag{3.7}$$

Puisqu' il faut utiliser n bits pour k bits d'information à communiquer, le coût du codage est donc proportionnel à $1/r$.

Il est évident que tout codage doit être *injectif*, c'est-à-dire que chaque mot de code ne peut coder qu'un seul mot d'information.

On note que pour un code en blocs $C(n,k)$ il y'a :

2^k: Mots d'information de k bits à coder

2^n: Messages reçus possibles, de longueur n, dont :

2^k: Sont des mots de code.

Après le codage, le décodage est nécessaire au niveau de récepteur. Cette opération consiste à calculer les distances entre le mot de code reçu et tous les mots du code valides.

III.3.1 Définition de code linéaire en bloc

Les codes linéaires en blocs (en anglais LBC pour Linear Block Code) sont largement utilisés en pratique, soit seuls soit en combinaison avec d'autres familles de code comme par exemple les codes convolutifs.

Un codage est dit linéaire quand le code C vérifie :

$$C(u_1+u_2+\ldots.+u_p)=C(u_1)+C(u_2)+\ldots+C(u_p),\ \forall\ p\geq 1. \tag{3.8}$$

u_1, u_2,…, u_p sont les messages de source. Il est dit systématique lorsque les bits de contrôle $c_1,c_2,\ldots,c_r$ ($r=n-k$) s'ajoutent directement aux k bits de message. Le mot de code v résultant est de longueur $n=k+r$ tel que $[v_1,v_2,\ldots, v_n]=[m_1,m_2,\ldots,m_k,c_1,c_2,\ldots,c_r]$;

III.3.2 Codes systématiques

Il est pratique de construire un mot de code de longueur n en ajoutant à la suite des k bits $i_1\ldots\ i_k$ d'information, $r = (n-k)$ bits, les r bits appelées les bits de *contrôle* ou de *redondance*, forment la clé de contrôle.

Ce codage est dit *systématique* ainsi que tout code que l'on peut obtenir par un tel codage et qui est donc un code par bloc particulier.

Le contrôle est alors simple, toute suite de longueur k étant un mot d'information, le récepteur calcule la clé correspondant au mot formé par les k premiers bits du message. Si cette clé est différente de celle qui se trouve en fin du mot reçu, il y a anomalie, signe d'erreur au cours de la transmission [23].

III.3.3 Matrice génératrice

La matrice génératrice d'un code linéaire en bloc dans sa version systématique a la forme suivante :

$$G=(\ I_k \mid P),\ [I_k]=k\times k,\ [P]=k\times r;\ r=n-k \tag{3.9}$$

Où P est matrice de contrôle.

Pour coder un message m de taille k (m=[m_1 m_2 m_3….m_k]), il suffit de le multiplier par la matrice génératrice G. Le résultat est le mot de code v de taille n :

$$v=m\,G \tag{3.10}$$

[m]=1×k, [G]=k×n, [v]=1×n. Rappelons que l'addition se fait en modulo 2.

Exemple : Soit le code C (7,4) qui au vecteur d'information m = (m_1, m_2, m_3, m_4) associe le mot de code v= (m_1, m_2, m_3, m_4, c_1, c_2, c_3), avec $c_1 = m_1+m_2$, $c_2 = m_3+m_4$, et $c_3 = m_1+m_2+m_4$.

Figure 3.2 montre le circuit du codeur.

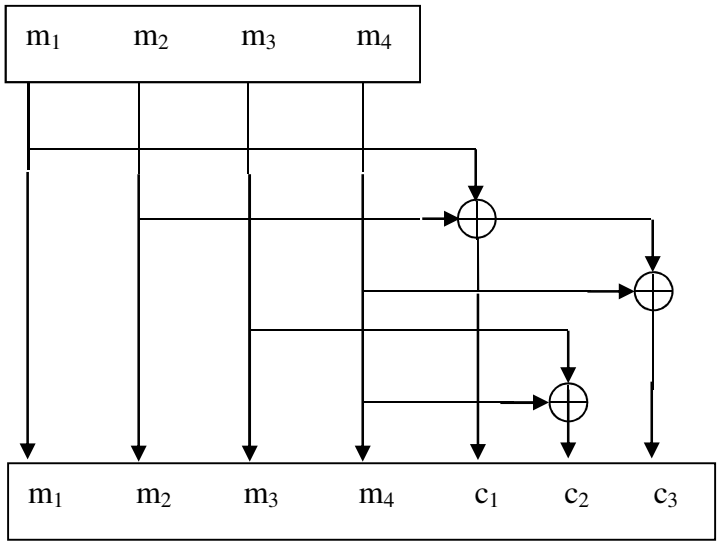


Figure 3.2: Circuit logique d'un codeur LBC (7,4).

Confirmant que ce code satisfait la condition de linéarité (3.6). Prenons deux messages de source m= (m_1, m_2, m_3, m_4) et u= (u_1, u_2, u_3, u_4). On a :

$C(m+u) = (m_1+ u_1, m_2+ u_2, m_3+ u_3, m_4+ u_4, m_1+ u_1+ m_2+ u_2, m_3+ u_3+ m_4+ u_4, m_1+ u_1+ m_2+ u_2+ m_4+ u_4)$

$= (m_1, m_2, m_3, m_4, m_1+ m_2, m_3+ m_4, m_1+ m_2+m_4) + (u_1, u_2, u_3, u_4, u_1+ u_2, u_3+ u_4, u_1+ u_2+ u_4)$

$= C(m)+ C(u)$

La matrice génératrice de ce code est donnée par :

$$G=\begin{pmatrix}1&0&0&0&1&0&1\\0&1&0&0&1&0&1\\0&0&1&0&0&1&0\\0&0&0&1&0&1&1\end{pmatrix} \tag{3.11}$$

Par exemple, le mot de code correspondant au message i = (1 0 1 0) est obtenu comme suit :

$$v = i\,G = (1\ 0\ 1\ 0\ 1\ 1\ 1) \tag{3.12}$$

Si maintenant nous considérons un codage non-systématique dont le mot de code associé au message m est donné par $v = (m_3, c_1, m_1, c_3, m_2, m_4, c_2)$, la matrice génératrice est obtenue par simple permutation des colonnes de la matrice G calculée en haut selon les positions de bits de message et de bits de contrôle. On obtient ainsi la matrice G_1 suivante :

$$G_1 = \begin{pmatrix} 0\,1\,1\,1\,0\,0\,0 \\ 0\,1\,0\,1\,1\,0\,0 \\ 1\,0\,0\,0\,0\,0\,1 \\ 0\,0\,0\,1\,0\,1\,1 \end{pmatrix} \tag{3.13}$$

III.3.4 Poids d'un code Hamming

Le poids de Hamming d'un mot est le nombre de bits à 1 qu'il contient appelé aussi moment d'un code. La distance de Hamming entre deux mots de même longueur est définie par le nombre de positions binaires qui diffèrent entre ces deux mots. On l'obtient par le poids de Hamming de la somme binaire des 2 mots. La distance de Hamming d'un code est la distance minimum entre tous les mots du code.

Exemple : Le poids de Hamming de mot (01001100) est 3.

III.3.5 Distance de Hamming

Les messages transmis sont supposés découpés en blocs (ou mots) de longueur n écrits avec l'alphabet {0,1}. Un code (binaire) est un sous-ensemble C de l'ensemble de tous les mots possibles. On dit que n est la longueur de C.

La distance de Hamming entre deux mots $x = (x_1 \ldots x_n)$ et $y = (y_1 \ldots y_n)$, que l'on notera d(x,y) est le nombre d'indices i tels que $x_i \neq y_i$.

La distance minimale du code C est le minimum des d(x,y) pour x et y des mots différents de C (on suppose que C a au moins 2 mots). On la notera toujours d. Le code linéaire est ainsi noté $C(n,k,d)$.

Exemple : Considérons $C = \{c_0, c_1, c_2, c_3\}$ avec :

$c_0 = (00000)$; $c_1 = (10110)$; $c_2 = (01011)$; $c_3 = (11101)$

C'est un code de longueur 5 et de distance $d = 3$.

III.3.6 Limite de détection et de correction d'un code LBC

Un code linéaire en bloc $C(n,k,d)$ permet de détecter

$$d-1 \text{ erreurs} \tag{3.14}$$

et de corriger

$$t = \left\lfloor \frac{d-1}{2} \right\rfloor \quad \text{erreurs} \tag{3.15}$$

où $\lfloor . \rfloor$ est le symbole de la partie entière.

III.4 Codes détecteurs d'erreur

Cette première catégorie de codes permet de détecter l'erreur sans pouvoir la corriger, c'est-à-dire, il n'est pas possible de localiser la position du bit erroné dans le mot de code reçu. Ces codes permettent de réinterroger l'émetteur afin de retransmettre de nouveau le message de source jusqu'à être reçu sans erreurs.

III.4.1 Code de parité

Le code de parité est un code linéaire en bloc, systématique, détecteur d'un nombre impair d'erreurs mais ne permet pas les corriger car nous ne savons pas d'où vient l'anomalie.

Si le message d'information est de taille k, le bit de contrôle k+1 est un bit de parité, soit :

$$a_{k+1}=a_1+a_2+\ldots+a_k \tag{3.16}$$

+ : Désigne le symbole du « ou exclusif (xor) ».

Exemples : Code de parité C(k,k+1) de Tableau 3.1

Tableau 3.1: Exemple d'un code de parité.

Message	Mot de code
1101011001	11010110010
100	1001
11111000111001111	111110001110011110

D'après (3.11), on déduit que la matrice génératrice d'un code de parité (4,3,2) s'écrit :

$$G=\begin{pmatrix}1&0&0&1\\0&1&0&1\\0&0&1&1\end{pmatrix} \quad \text{avec} \quad P=\begin{pmatrix}1\\1\\1\end{pmatrix} \tag{3.17}$$

Rappelons que la détection performante des erreurs sans pouvoir les détecter permet de réinterroger l'émetteur afin de retransmettre de nouveau le message jusqu'à être reçu sans erreurs. C'est le cas du code CRC (Cyclic Redundancy Check) par exemple qui est très utilisé dans les réseaux Ethernet.

III.4.2 Code cyclique de redondance

Le code cyclique de redondance (en anglais CRC pour Cyclic Redundancy Check) est un code linéaire systématique détecteur d'erreurs. La méthode de calcul de bits de redondance constituant le CRC est décrite ci-après.

Soit la suite binaire $M=m_1m_2\ldots m_n$ constituant le message de source à transmettre sur le canal.

La représentation générale sous forme polynomiale de M s'écrit par le polynôme M(x) :

$$M(\mathrm{x})=m_n+m_{n-1}\mathrm{x}+\ldots+m_1\mathrm{x}^{n-1} \tag{3.18}$$

III.4.2.1 Émission d'un CRC

On ajoute à droite du message binaire à émettre « M » un code de contrôle tel que le polynôme correspondant au message plus le code de contrôle soit divisible par le polynôme générateur choisi. Le calcul du CRC à la transmission inclut les étapes suivantes :

1) Choix de polynôme générateur, noté $G(x)$, d'ordre m.
2) Transformation de $G(x)$ en un mot binaire comme nous l'avons fait avec le message « M ».
3) Ajout de m zéros au message binaire à transmettre. Le mot résultant « M_e » contient alors $(n+m)$ bits.
4) À cette étape, si nous choisissons la méthode de la *division euclidienne*, on transforme M_e en forme polynomiale puis on effectue une division euclidienne de $M_e(x)$ sur $G(x)$ dans laquelle on ne tient pas compte du quotient. Le reste de division représente le CRC à ajouter au mot avant transmission. Sinon, la *division binaire* est plus simple et donc plus recommandée : Elle consiste à ajouter itérativement au mot binaire M_e en base 2 ('ou exclusif xor'), le mot correspondant au polynôme générateur de telle sorte que si le bit de poids fort du reste intermédiaire est '0' on décale de reste intermédiaire. Cette opération de décalage-sommation est poursuivie jusqu'à ce que le mot obtenu soit inférieur au polynôme générateur. Ce mot correspond au CRC tandis que le quotient est sans intérêt.

II.4.2.2 Réception d'un CRC et détection d'erreurs

Le mot reçu contient le message de la source plus le CRC de m bits. Il doit être divisible par le polynôme générateur. La vérification de transmission sans erreurs se fait par division euclidienne en base 2 ou par division binaire de mot reçu et du polynôme générateur qui doit résulter en un reste nul.

Le code CRC est très utilisé car il est connu par sa performance très élevée de détection d'erreurs dû essentiellement à sa méthode particulière de calcul de bits de redondance en dépendance du polynôme générateur. Nous le savons d'ailleurs de la théorie de l'information que le codage est plus performant lorsqu'il est plus complexe. On retrouve ce code très souvent dans les réseaux locaux ainsi que les réseaux de télécommunications.

Par exemple, le CRC-16 ($G(x)=x^{16}+x^{15}+x^2+1$) et CRC-CCITT ($G(x)=x^{16}+x^{12}+x^5+1$) utilisés en téléphonie sans fils détectent des erreurs simples et doubles, des erreurs impaires, plusieurs bits successifs erronés, dits salves, de 16 bits ou moins, 99.997% des salves de 17 bits et jusqu'à 99.998% des salves de 18 bits ou plus.

Exemple 1 : Soit le polynôme générateur $G(x)=x^4+x^2+x$.

a. On souhaite transmettre le message suivant :1111011101.

La suite 1111011101 est représentée par le polynôme (n=10) :

$$M(x) = 1.x^9+1.x^8+1.x^7+1.x^6+0.x^5+1.x^4+1.x^3+1.x^2+0.x^1+1.x^0$$
$$= x^9+x^8+x^7+x^6+x^3+x^2+1 \quad (3.19)$$

Dans ce cas : $G(x)= x^4+x^2+x$ ce qui correspond au mot binaire de polynôme générateur : 10110. M=1111011101 et donc M_e=11110111010000 (ajout de 4 zéros correspondant à l'ordre de polynôme générateur m=4). La division binaire est effectuée comme suit

```
1 1 1 1 0 1 1 1 0 1 0 0 0 0
1 0 1 1 0
---------------------------
0 1 0 0 0 1 1 1 0 1 0 0 0 0
  1 0 1 1 0
---------------------------
  0 0 1 1 1 1 1 0 1 0 0 0 0
      1 0 1 1 0
---------------------------
      0 1 0 0 1 0 1 0 0 0 0
        1 0 1 1 0
---------------------------
        0 0 1 0 0 1 0 0 0 0
            1 0 1 1 0
---------------------------
            0 0 1 0 0 0 0 0
                1 0 1 1 0
---------------------------
              0 0 1 1 0 0
```

Le CRC est par conséquent **1100**. Le mot de code à transmettre sur le canal est 11110111010**1100**. La division binaire du mot reçu et le polynôme générateur donne le résultat suivant :

```
1 1 1 1 0 0 0 1 0 1 0 1 0 1
1 0 1 1 0
---------------------------
0 1 0 0 0 0 0 1 0 1 0 1 0 1
  1 0 1 1 0
---------------------------
  0 0 1 1 0 0 1 0 1 0 1 0 1
      1 0 1 1 0
---------------------------
    0 1 1 1 1 0 1 0 1 0 1
      1 0 1 1 0
---------------------------
      0 1 0 0 0 1 0 1 0 1
        1 0 1 1 0
---------------------------
        0 0 1 1 1 0 1 0 1
            1 0 1 1 0
---------------------------
            0 1 0 1 1 0 1
              1 0 1 1 0
---------------------------
              0 0 0 0 0 1
```

Le reste de la division n'est pas nul ; il existe alors des erreurs dans le mot reçu.

III.5 Codes correcteurs d'erreur

III.5.1 Codage par répétition

Le codage par répétition est très utilisé dans les systèmes de communications pour détection et correction des erreurs introduites par le canal bruité. Ce codage est très pratiqué dans nos communications par langage oral. Par exemple, il est très courant lorsque nous ne comprenons pas une personne parlant de lui demander de répéter. Avec les fragments d'information recueillis à chaque répétition, on finit par reconstruire le message tel qu'élaboré à la source, dans l'esprit de cette personne. Le décodage s'effectue en faisant confiance au vote majoritaire.

Exemple : On considère un code correcteur d'erreur $C(n,k)$ pour lequel $k = 2$ et n est un entier pair tel que $n \geq 6$, et dont les mots de codes v sont obtenus à partir des mots d'informations $u = (u_1,u_2)$ en les répétant $(n/2- 1)$ fois. En d'autres termes, le mot de code obtenu à partir de u = (u_1, u_2) où (u_1, u_2) appartient à $\{0,1\}^2$ s'écrit :

$$v = (u_1, u_2, u_1, u_2, \ldots., u_1, u_2) \tag{3.20}$$

Le message est répété $(n/2- 1)$ fois. Ceci correspond à la matrice G ci-dessous :

$$G = \left(\begin{array}{c|c|c|c|c} 1\,0 & \underbrace{1\,0 \mid 1\,0 \mid \ldots}_{(n/2\,-1)\text{ fois}} & 1\,0 \\ 0\,1 & 0\,1 \mid 0\,1 \mid \ldots & 0\,1 \end{array} \right) \tag{3.21}$$

La matrice G n'est rien que une répétition de $(n/2)$ fois la matrice d'identité I_2. Ceci produira un mot de code de taille n pour un message de k=2 bits. Par exemple, pour n=6, la matrice génératrice devient :

$$G = \begin{pmatrix} 1\,0\,1\,0 \\ 0\,1\,0\,1 \end{pmatrix} \tag{3.22}$$

Il est clair que la distance de Hamming est égale à $n/2$ (qui correspond au message 01 et 10). Puisqu'il s'agit d'un code linéaire, le nombre maximal de bits erronés que ce code garantit de pouvoir toujours détecter est égale à d-1=$n/2$ -1. Le nombre maximal de bits erronés que ce code peut corriger est égale à $t = \left\lfloor \frac{d-1}{2} \right\rfloor = \frac{n}{4} - \frac{1}{2}$.

III.5.2 Code de Hamming

III.5.2.1 Définition

Un code de Hamming [22] est un code correcteur linéaire. Il permet la détection et la correction automatique d'une erreur si elle ne porte que sur une lettre du message.

Un code de Hamming est parfait, ce qui signifie que pour une longueur de code donnée, il n'existe pas d'autre code plus compact ayant la même capacité de correction. En ce sens, son rendement est maximal.

Le code de Hamming est de paramètres $(2^m-1,\ 2^m-m-1,3)$, il permet de détecter 3 erreurs et de corriger une seule. Plusieurs méthodes permettent de construire un code de Hamming. Une approche consiste à rechercher les codes cycliques de distance minimale égale à trois.

III.5.2.2 Code de Hamming non-systématique

Les bits de contrôle de parité C_i sont en position 2^i ; i=0, 1, 2, … Les bits du message utile (de source discrète) D_j occupent le reste du mot.

Pour mieux comprendre, prenons l'exemple d'un code de Hamming (7,4) :

Le message utile est : $D_{y1}\, D_{y2\ldots}\, D_{j\ldots}\, D_0$

Le nombre de bits de contrôle est : r=7-3=3 bits

7	6	5	4	3	2	1
D_3	D_2	D_1	C_2	D_0	C_1	C_0

$$\begin{cases} C_0 \text{ est placé dans la position } 2^0=1 \\ C_1 \text{ est placé dans la position } 2^1=2 \\ C_2 \text{ est placé dans la position } 2^2=4 \end{cases} \tag{3.23}$$

Pour retrouver les erreurs dans un code de Hamming, nous procédons comme suit :

- Si les bits de contrôle de réception C_i', i=1,2,…,k valent *zéro*, donc il n'y a pas d'erreurs.
- Sinon la valeur des bits de contrôle en décimale indique la position de l'erreur.

Le calcul des paramètres C_i' s'effectue comme suit :

- Si C_0'=1, les valeurs possible de $C_2'C_1'C_0'$ sont : 0 0 **1,**0 1 **1,**1 0 **1,** 1 1 **1,** c'est-à-dire **:** 1,3,5,7.
- Si C_1'=1, les valeurs possible de $C_2'C_1'C_0'$ sont : 0 **1 0,**0 **1 1,**1 **1 0,** 1 **1 1,** c'est-à-dire **:** 2,3,6,7.
- Si C_2'=1, les valeurs possible de $C_2'C_1'C_0'$ sont : **1** 0 0**,1** 0 1**,1** 1 0**,** **1** 1 1**,** c'est-à-dire **:** 4,5,6,7.

Exemple 1 : Soit le mot de code suivant :

7	6	5	4	3	2	1
1	0	1	0	1	1	0

$$\begin{cases} C_2' \text{ vaut } 4+5+6+7 : 0+1+1+1=0 \\ C_1' \text{ vaut } 2+3+6+7 : 1+1+0+1=1 \\ C_0' \text{ vaut } 1+3+5+7 : 0+1+1+1=1 \end{cases} \quad (3.24)$$

$C_2'C_1'C_0'$ vaut 0 1 1, c'est-à-dire 3 en décimal ; il y a une erreur à l'indice 3 de mot de code donné.

Exemple 2 : On souhaite envoyer le message 1010. La forme générale du mot de code de Hamming (7,4) associé est donnée comme suit :

7	6	5	4	3	2	1
D_3	D_2	D_1	C_2	D_0	C_1	C_0

On a alors : (D_3 D_2 D_1 D_0)=(1010) ; ce qui donne :

1	0	1	C_2	0	C_1	C_0

avec : C_0=3+5+7 et sa position est 2^0=1

C_1=3+6+7 et sa position est 2^1=2

C_2=5+6+7 et sa position est 2^2=4

Par conséquent, le mot de code est :

1	0	1	**0**	0	**1**	**0**

Si le mot de code reçu est 1010110, la structure générale pour détection des erreurs est:

7	6	5	4	3	2	1
1	0	1	0	1	1	0

La combinaison $C'_2\ C'_1\ C'_0$ est calculée comme suit :

C'_0=1+3+5+7=0+1+1+1=1

C'_1=2+3+6+7=1+1+0+1=1

C'_2=4+5+6+7=0+1+0+1=0

$C'_2\ C'_1\ C'_0$=011≠ 000 donc il existe des erreurs.

En ce qui concerne la correction, puisqu'il s'agit d'un code de Hamming, la correction est parfaite s'il s'agit d'une seule erreur (position $(011)_{10}$=3), c'est-à-dire, le code peut corriger une erreur unique quelque soit sa position.

Il est aussi intéressant de déterminer la matrice génératrice de code de Hamming non-systématique. Pour cela, rappelons qu'en tenant compte des positions des différents bits dans le mot de code, les bits de contrôle dans le cas d'un code (7,4) sont exprimés comme suit :

$$C_0=D_0+D_1+D_3 \qquad C_1=D_0+D_2+D_3 \qquad C_2=D_1+D_2+D_3 \tag{3.25}$$

Le mot de code non-systématique (v_6 v_5 v_4 v_3 v_2 v_1 v_0) s'écrit alors :

$$v_6=D_3 ; v_5=D_2 ;$$

$$v_4=D_1; v_3=D_1+D_2+D_3; v_2=D_0; v_1=D_0+D_2+D_3; v_3=D_0+D_1+D_3$$

D'où :

$$G=\begin{pmatrix}1&0&0&1&0&1&1\\0&1&0&1&0&1&0\\0&0&1&1&0&0&1\\0&0&0&0&1&1&1\end{pmatrix} \tag{3.26}$$

III.5.3 Correction par la méthode de syndrome

III.5.3.1 Définition de syndrome

La matrice de contrôle d'un code linéaire en bloc (version systématique) a la forme suivante :

$$H=G^T=(P^T \mid I_r),\ [P^T]=r\times k,[I_r]=r\times r;\ r=n-k \tag{3.27}$$

Le syndrome est un vecteur de (n-k) élément défini par :

$$S=R\,H^T \tag{3.28}$$

R : mot de code reçu

où

$$H^T=\begin{pmatrix}P\\ \hdashline I_r\end{pmatrix} \tag{3.29}$$

$$[R]=[r_1,r_2,\ldots..,r_n]=1\times n \tag{3.30}$$

$$[H]=(n-k)\times n \tag{3.31}$$

$$[H^T]=n\times (n-k) \tag{3.32}$$

$$[S]=(1,n)\times (n,n-k)=1\times(n-k) \tag{3.33}$$

Etant donné un mot de code reçu, si S_i=0 (i=1, …, n-k) alors il n'y a pas d'erreur, s'il existe un seul S_j (j=1, …, n-k) qui est non nul, il y a des erreurs. Il est important de noter que la distance minimale d'un code linéaire en bloc peut être directement déterminée comme étant le nombre des colonnes linéairement indépendantes de la matrice de contrôle H.

Exemple 1 : soit le mot de code v= [1 0 0 1 0 1 1] transmis sur un canal bruité. Le mot de code reçu est : R= [1 0 0 1 0 0 1].

On vérifie tout d'abord que v est un mot de code valide. Pour faire, on calcule le syndrome associé à v :

$$S=v\,H^T \tag{3.34}$$

$$v=[1\ 0\ 0\ 1\ 0\ 1\ 1]=[v_1\ v_2\ v_3\ v_4\ v_5\ v_6\ v_7] \tag{3.35}$$

$$H^T=\begin{pmatrix} 1\ 1\ 0 \\ 0\ 1\ 0 \\ 1\ 1\ 1 \\ 1\ 0\ 1 \\ 1\ 0\ 0 \\ 0\ 1\ 0 \\ 0\ 0\ 1 \end{pmatrix} \tag{3.36}$$

$S=(s_1\ s_2\ s_3)=v\,H^T$

$S_1=v_1+v_3+v_4+v_5=0$

$S_2=v_1+v_2+v_3+v_6=0$

$S_3=v_1+v_2+v_4+v_6=0$

S=(0 0 0), donc v est un mot de code valide

- Généralisation :

$$S=v\,H^T=0 \tag{3.37}$$

Pour un mot de code reçu R=[1 0 0 1 0 0 1]

$$S=R\,H^T \tag{3.38}$$

On trouve:

$$s_1=0,\ s_2=1,\ s_3=0$$

$S=(s_1\ s_2\ s_3)$=(0 1 0); $S_2\neq 0$, il existe des erreurs dans le mot reçu.

III.5.3.2 Correction par le syndrome

Sur un canal bruité, le mot de code reçu s'écrit :

$$R=v+e \tag{3.39}$$

$$e=(e_1\ e_2 \ldots e_n)$$

e : vecteur d'erreur.

Le syndrome peut s'écrire alors de la façon suivante :

$$S=R\,H^T=(v+e)\,H^T=v\,H^T+e\,H^T \tag{3.40}$$

$S=e\,H^T$: expression pour corriger l'erreur.

$S=R\,H^T$: expression pour tracé le circuit de syndrome.

Le mot de code est corrigé comme suite :

$$\hat{v}=R+e \tag{3.41}$$

Vérification : $S=\hat{v}\,H^T$ doit être un vecteur zéros.

Exemple 1 : Soit le code linéaire systématique défini par la matrice de contrôle :

$$P=\begin{pmatrix}1&0&1\\0&1&1\\1&1&0\\1&1&1\end{pmatrix} \tag{3.42}$$

Dans ce cas: $[P]=k\times r=4\times 3$, ce qui donne : $k=4$, $r=3$ et donc $n=r+k=7$.

Il s'agit d'un code LBC systématique de longueur $n=7$ dont quatre bits de message à coder et trois bits de contrôle.

La forme de P est connue en théorie de codage et elle correspond au code de Hamming dans sa version systématique. La distance minimale de ce code est toujours $d=3$. La désignation complète de ce code s'écrit alors $C(7,4,3)$.

Si nous voulons coder le message 10011101, par exemple, nous procédons en deux étapes :

1) Découpage du message en paquets de k bits,

2) Multiplication de chaque paquet par la matrice génératrice donnant ainsi le mot de code : $v=mG$, m étant le message de source de longueur $k=4$. Dans ce cas :

$$G=(I_k \mid P)=\ G=(I_4 \mid P)\ =\begin{pmatrix}1&0&0&0&1&0&1\\0&1&0&0&0&1&1\\0&0&1&0&1&1&0\\0&0&0&1&1&1&1\end{pmatrix} \tag{3.43}$$

$$m_1=[1\ 0\ 0\ 1]\rightarrow v_1=m_1G=[1\ 0\ 0\ 1]\begin{pmatrix}1&0&0&0&1&0&1\\0&1&0&0&0&1&1\\0&0&1&0&1&1&0\\0&0&0&1&1&1&1\end{pmatrix}=1\ 0\ 0\ 1\ 0\ 1\ 0 \tag{3.44}$$

$$m_2=[1\ 1\ 0\ 1]\rightarrow v_2=m_2G=[1\ 1\ 0\ 1]\begin{pmatrix}1&0&0&0&1&0&1\\0&1&0&0&0&1&1\\0&0&1&0&1&1&0\\0&0&0&1&1&1&1\end{pmatrix}=1\ 1\ 0\ 1\ 0\ 1\ 1 \tag{3.45}$$

Le mot à transmettre sur le canal est donc 10010101101011.

Selon (3.14), la matrice de contrôle a la forme suivante :

$$H=\begin{pmatrix}1&0&1&1&1&0&0\\0&1&1&1&0&1&0\\1&1&0&1&0&0&1\end{pmatrix} \tag{3.46}$$

Vérifiant maintenant si les messages suivants sont corrects : 1111100, 0111000. Rappelons qu'un mot de code R est reçu correctement si $R\,H^T=0$, avec

$$H^T=\begin{pmatrix}P\\I_r\end{pmatrix}=\begin{pmatrix}1&0&1\\0&1&1\\1&1&0\\1&1&1\\1&0&0\\0&1&0\\0&0&1\end{pmatrix} \tag{3.47}$$

Pour R_1=1111100 on trouve

$$R_1\,H^T=011 \tag{3.48}$$

Il existe des erreurs dans R_1.

Pour R_2=0111000 on trouve

$$R_2\,H^T=010 \tag{3.49}$$

Il existe des erreurs dans R_2.

Un autre point important. Nous savons d'après la théorie de codage que chaque ligne de H^T correspond à une erreur simple. Donc le nombre des erreurs simples est n. Le syndrome S est de taille r. Le nombre de combinaisons possibles du syndrome est donc 2^r dont un état de zéros partout dans le cas d'une transmission sans erreurs. Le nombre de combinaisons doubles est donc 2^r-n-1.

Dans notre exemple, le nombre des erreurs simples est :

$$n=7$$

et le nombre des erreurs doubles est:

$$2^3\text{-}7\text{-}1=0$$

Ce qui est attendu car le code de Hamming est de distance d=3 et donc ne peut corriger qu'une seule erreur :

$$t=\left\lfloor\frac{d-1}{2}\right\rfloor=\lfloor 1.5\rfloor=1 \tag{3.50}$$

La finalisation de correction consiste à construire la table d'erreur/syndrome en se basant sur les règles suivantes :

1) Chaque ligne de H^T correspond à une erreur simple,
2) La $i^{ème}$ ligne correspond à l'erreur en bit d'indice i-1. Ceci correspond à une erreur en bit r_{i-1} donc à une valeur de 1 dans le vecteur d'erreur e à la position i-1 où $e=e_0e_1….e_{n-1}$ relié au syndrome par la relation (3.20).

Tableau 3.2 présente la table d'erreur/syndrome de l'exemple 1 pour les erreurs simples et le cas sans erreur.

Tableau 3.2: Présentation de la table d'erreur/syndrome de l'exemple 1.

Syndrome	Erreur	Observation
000	0000000	Pas d'erreurs
001	0000001	Erreur simple en septième bit r_6 (ceci correspond à la $7^{ème}$ ligne de H^T)
010	0000010	Erreur simple en sixième bit r_5 (ceci correspond à la $6^{ème}$ ligne de H^T)
011	0100000	Erreur simple en deuxième bit r_1 (ceci correspond à la $2^{ème}$ ligne de H^T)
100	0000100	Erreur simple en quatrième bit r_4 (ceci correspond à la $5^{ème}$ ligne de H^T)
101	1000000	Erreur simple en premier bit r_0 (ceci correspond à la $1^{ère}$ ligne de H^T)
110	0010000	Erreur simple en troisième bit r_2 (ceci correspond à la $3^{ème}$ ligne de H^T)
111	0001000	Erreur simple en quatrième bit r_3 (ceci correspond à la $4^{ème}$ ligne de H^T)

Examinons à titre d'exemple la correction du mot reçu 0100101 par la méthode de syndrome. Le vecteur S correspondant à ce mot est :

$$S = [0\ 1\ 0\ 0\ 1\ 0\ 1]\ H^T = [1\ 1\ 0] \tag{3.51}$$

La correction de l'erreur s'effectue en ajoutant le vecteur d'erreur correspondant au syndrome 110 au mot reçu. Á partir de la table d'erreur/syndrome ci-dessus on trouve le mot corrigé $\hat{v}$ comme suit :

$$\hat{v} = R+e^* = [0\ 1\ 0\ 0\ 1\ 0\ 1] + [0\ 0\ 1\ 0\ 0\ 0\ 0] = [0\ 1\ 1\ 0\ 1\ 0\ 1] \tag{3.52}$$

On vérifie que :

$$\hat{v}\ H^T=0 \tag{3.53}$$

Exemple 2: Soit un code LBC dont la matrice de contrôle est donnée par:

$$P=\begin{pmatrix}1&1&1\\1&0&1\\0&1&1\\1&1&0\end{pmatrix} \tag{3.54}$$

La table erreur/syndrome pour les erreurs simples et le cas sans erreurs est donnée par Tableau 3.3.

Tableau 3.3: Présentation de la table d'erreur/syndrome de l'exemple 2.

Syndrome	Erreur	Observation
000	0000000	Pas d'erreurs
001	0000001	Erreur simple en septième bit r_6 (ceci correspond à la $7^{ème}$ ligne de H^T)
010	0000010	Erreur simple en sixième bit r_5 (ceci correspond à la $6^{ème}$ ligne de H^T)
011	0010000	Erreur simple en troisième bit r_2 (ceci correspond à la $3^{ème}$ ligne de H^T)
100	0000100	Erreur simple en cinquième bit r_4 (ceci correspond à la $5^{ème}$ ligne de H^T)
101	0100000	Erreur simple en deuxième bit r_1 (ceci correspond à la $2^{ème}$ ligne de H^T)
110	0001000	Erreur simple en premier bit r_3 (ceci correspond à la $4^{ème}$ ligne de H^T)
111	1000000	Erreur simple en premier bit r_0 (ceci correspond à la $1^{ère}$ ligne de H^T)

Pour définir les limitations de correction par la méthode de syndrome, essayons de corriger le message 0100101.Le syndrome correspondant est :

$$s = [0\ 1\ 0\ 0\ 1\ 0\ 1]\ H^T = [0\ 0\ 0] \tag{3.55}$$

Le syndrome est nul et donc le code ne peut ni détecter ni corriger les erreurs. Il s'agit d'une erreur multiple. En fait, ce résultat s'explique par le fait que la matrice de contrôle de cet exemple n'est rien qu'une permutation de la matrice P donnée dans l'exemple 1. Il s'agit encore une fois d'un code de Hamming systématique de distance d=3. Par conséquent, le nombre d'erreurs qui peuvent être détectées est d-1 = 2 erreurs et une erreur unique peut être corrigée (t=1). La méthode de syndrome simplifie la détection et la correction des erreurs mais ne permet pas d'améliorer les performances de codage effectué

III.6 Conclusion

Les codes linéaires en bloc présentent d'énormes avantages pratiques en particulier fiabilité en détection et correction des erreurs introduites par le canal de transmission, facilité d'implémentation, très faible complexité matérielle et numérique ainsi que le coût réduit. Les performances de détection et de correction dépendent directement de la distance de Hamming du code utilisé. Autant que cette distance augmente, mieux sont les performances obtenues. Cependant, ceci revient à augmenter la longueur de mot de code et le nombre de bits de contrôle

ce qui réduit le débit et augmente la complicité de décodage surtout en utilisant des algorithmes de codage plus complexes.

Dans le chapitre qui suit, nous décrivons le principe de fonctionnement d'un autre type de codes linéaires très souvent utilisé dans les systèmes de communications numériques. Il s'agit de fameux codes convolutifs. Nous décrivons leur principe de génération, leurs propriétés, ainsi que leur algorithme de décodage le plus utilisé (algorithme de Vitrebi).

Chapitre IV

Codage de canal par les codes convolutifs

IV.1 Introduction

Les codes convolutifs, introduits en 1955 par Elias, peuvent être considérés comme un cas particulier des codes en bloc linéaires, mais un point de vue plus large nous fera découvrir que la structure convolutive additionnelle munit le code linéaire de propriétés favorables qui facilitent à la fois son codage et améliorent ses performances.

Dans un premier temps, nous présenterons les codes convolutifs comme un cas particulier des codes en bloc linéaires, avant d'exploiter dans un deuxième temps leur structure spécifique pour aboutir à un procédé de décodage à maximum de vraisemblance de faible complexité [24].

IV.2 Définition

Le codeur qui engendre un code convolutif comporte un *effet de mémoire.* Un codeur convolutif binaire (n, k, m) est défini par la donnée de m+1, matrices binaires $k \times n$ et le codage d'une séquence binaire infinie u est égal à $v = u.G$, G est une matrice binaire infinie. On note :

n : nombre de bits à la sortie de codeur,

k : nombre de bits à l'entrée de codeur (bits de message de source),

m : nombre de registre à décalage (mémoire),

IV.3 Principe du codage convolutif

- Les codes convolutifs forment une classe extrêmement souple et fiable de codes correcteurs d'erreur.
- Ce sont les codes les plus utilisés dans les communications fixes et mobiles.
- Les codes convolutifs ont les mêmes caractéristiques que les codes en bloc sauf qu'ils s'appliquent à des *séquences infinies de symboles d'information et génèrent* des séquences *infinies de symboles de code.*
- Le codeur qui engendre un code convolutif comporte *un effet de mémoire.*
- Le mot code ne dépend pas que du bloc de k symboles entrant, mais aussi des m mots de code qui l'ont précédé, stockés dans un registre.

Il est important de noter, que selon le théorème fondamental de codage de canal, la complexité du codeur est nécessaire à l'obtention de bonnes performances :

- Pour les codes en bloc : n et k doivent être grands,
- Pour les codes convolutifs : il suffit que m soit grand.

IV.4 Propriétés des Codes Convolutifs

- Le rendement du code est :

$$r = \frac{k}{n} \tag{4.1}$$

- La longueur de contrainte du code est donnée par :

$$L = (m+1)\,k \tag{4.2}$$

- La *distance libre* est la borne inférieure des distances de Hamming entre toutes les séquences de sortie du codeur.
- La *distance minimale* est la plus petite distance entre des chemins partant du même point et y revenant.
- *Linéarité* : les mots de code associés à une combinaison linéaire de séquences d'entrée correspondent à la combinaison linéaire des mots de code de chacune des ces séquences.
- *Stationnarité* : Lorsqu'un message source, décalé dans le temps, est envoyé sur l'encodeur, on doit retrouver à la sortie, le mot de code correspondant décalé de la même manière dans le temps.
- Code convolutif *systématique*:

  Mot code : $C = (X_1\ Y_1\ X_2\ Y_2\ \ldots\ X_j\ Y_j\ \ldots)$

  Avec $X_j = (X^1_j\ X^2_j \ldots X^k_j)$ Information

  $Y_j = (Y^1_j\ Y^2_j \ldots Y^k_j)$ Contrôle

IV.5 Différence par rapport aux codes linéaire

Les codes en bloc traitent chaque bloc d'information *indépendant* les uns des autres : Deux messages différents sont codés d'une façon indépendante l'un de l'autre : la notion de *mémoire* n'est pas utilisée.

Pour les codes convolutifs, la sortie d'un codeur convolutif dépend d'un symbole courant à coder ainsi que du symbole *précédent* et du résultat de codage du symbole précédent. C'est un codage avec mémoire (registre à décalage, noté en anglais SR pour Shift Registre).

Exemple : Soit un codeur convolutif non- systématique (Figure 4.1): $r = 1/2$; $m = 2$; $k = 1$; $n = 2$.

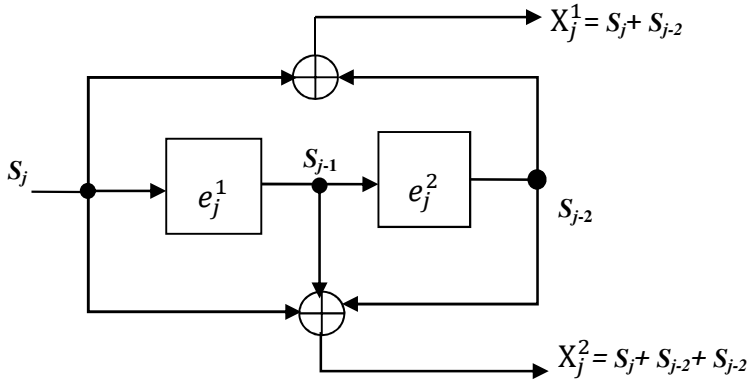


Figure 4.1: Exemple de représentation du codeur convoutif par circuit logique.

À chaque pas de temps *j* :

- On combine les valeurs de l'entrée et de la mémoire pour calculer les sorties,
- Chaque registre à décalage est mis à jour par la valeur qui figure à son entré.

La notation utilisée :

S_j : l'entrée à l'instant *j* (bit actuel à coder).

X_j : la sortie à l'instant *j* (sortie actuel du codeur).

e_j^1, e_j^2…… registre à décalage.

IV.6 Représentation du code convolutif :

Il y a deux méthodes pour la représentation du code convolutif.

IV.6.1 Représentation Numérique

La représentation numérique contient deux méthodes qui sont :

- Transformée en D,
- Matrice de transfert.

IV.6.1.1 Transformée en D

Une séquence de symboles est représentée par une série formelle en la variable *D*. Cette variable représente l'opérateur de retard unitaire (opérateur de décalage).

$$\begin{cases} s(D)=s_0+s_1.D+s_2.D^2+\ldots\ldots+s_j.D_j \\ x^i(D)=x_0^i+x_1^i.D+x_2^i.D^2+\ldots\ldots+x_j^i.D^j \end{cases} \quad (4.3)$$

La puissance en *D* indique le nombre de décalage

Exemple 1 : Soit le codeur convolutif non- systématique de la Figure 4.1. On a:

$$\begin{cases} e^1(D) = D.s(D) \\ e^2(D) = D.\ e^1(\mathrm{D}) = D^2.\ s(D) \\ x^1(D) = s(D) + \mathrm{e}^2(D) = (1 + D^2).\ s(D) \\ x^2(\mathrm{D}) = s(D) + e^1(D) + e^2(D) = (1 + D + D^2).\ s(D) \end{cases} \quad (4.4)$$

La transformée en D donne la réponse impulsionnelle sous la forme :

$$H(D)=(1+D^2 \qquad 1+D+D^2) \qquad ([H(D)]=k\times n) \quad (4.5)$$

On peut tracer le circuit logique du codeur covolutif sachant $H(D)$ et vice-versa.

Exemple 2 : Soit le codeur convolutif de la Figure 4.2.

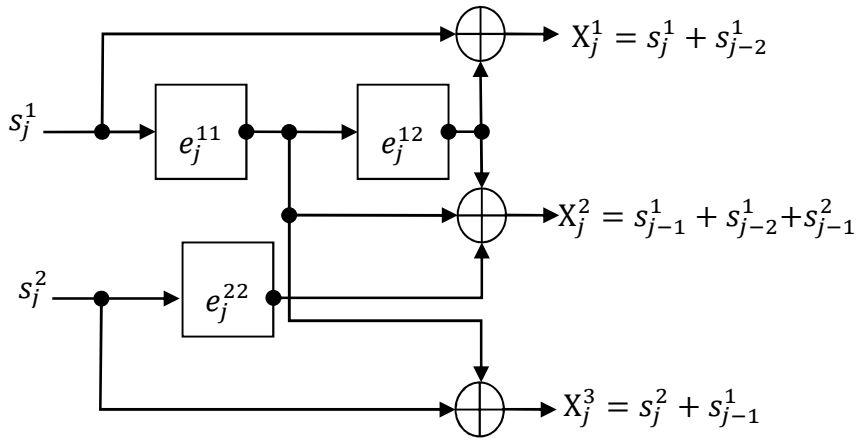


Figure 4.2: Circuit logique de codeur convoutif à deux entrées, trois sorties et 2 mémoires (r=2/3, n=3, m=2, L=(m+1) k=6).

Ce code est de la forme : (3,2,2)

La transformée en D :

$$[H(D)]=k \times n=2\times 3 \quad (4.6)$$

$$\begin{cases} \mathrm{X}_j^1 = s_j^1 + s_{j-2}^1 \\ \mathrm{X}_j^2 = s_{j-1}^1 + s_{j-2}^1 + s_{j-1}^2 \\ \mathrm{X}_j^3 = s_j^2 + s_{j-1}^1 \end{cases}$$

$$\begin{cases} \mathrm{X}^1(D) = s^1(D) + e^{12}(D) \\ \mathrm{X}^1(D) = s^1(D) + D^2\, s^1(D) = (1+D^2)\, s^1(D) + 0.\ \mathrm{s}^2(\mathrm{D}) \end{cases}$$

$$\begin{cases} \mathrm{X}^2(\mathrm{D}) = \mathrm{e}^{11}(\mathrm{D}) + \mathrm{e}^{12}(\mathrm{D}) + \mathrm{e}^{21}(\mathrm{D}) \\ \mathrm{X}^2(D) = D\, s^1(D) + D^2\, s^1(D) + D\, s^2(D) = (D+D^2)\, s^1(D) + D^2 s^2(D) \end{cases}$$

$$\begin{cases} \mathrm{X}^3(D) = e^{11}(D) + s^2(D) \\ \mathrm{X}^3(D) = D\, s^1(D) + 1.s^2(D) \end{cases} \qquad \Rightarrow H(D)=\begin{pmatrix} 1+\mathrm{D}^2 & \mathrm{D}+\mathrm{D}^2 & \mathrm{D} \\ 0 & \mathrm{D}^2 & 1 \end{pmatrix} \quad (4.7)$$

IV.6.1.2 Matrice de transfert

La matrice de transfert donne la relation entrée-sortie sous forme matricielle (binaire ou octave). On l'écrit pour chaque étage de sortie. Elle est constituée de matrices de transfert intermédiaires (polynômes générateurs G_1, G_2…).

La matrice de transfert globale est la concaténation des matrices précédentes. Elle a k lignes et $(m+1)\times n$ colonnes.

Exemple : Prenons le codeur convolutif de Figure 4.2.

- Relations entrée/sortie

$$\begin{cases} X_j^1 = s_j^1 + s_{j-2}^1 \\ X_j^2 = s_{j-1}^1 + s_{j-2}^1 + s_{j-1}^2 \\ X_j^3 = s_j^2 + s_{j-1}^1 \end{cases} \quad (4.8)$$

- Matrices de transfert intermédiaires:

$$\begin{cases} G_1 = \begin{pmatrix} \text{Sortie1=f (entrée1)} \\ \text{Sortie1=f (entrée2)} \end{pmatrix} = \begin{pmatrix} 1\ 0\ 1 \\ 0\ 0\ 0 \end{pmatrix}_{(2)} = \begin{pmatrix} 5 \\ 0 \end{pmatrix}_{(8)} \\ G_2 = \begin{pmatrix} \text{Sortie2=f (entrée1)} \\ \text{Sortie2=f (entrée2)} \end{pmatrix} = \begin{pmatrix} 0\ 1\ 1 \\ 0\ 1\ 0 \end{pmatrix}_{(2)} = \begin{pmatrix} 3 \\ 2 \end{pmatrix}_{(8)} \\ G_3 = \begin{pmatrix} \text{Sortie3=f (entrée1)} \\ \text{Sortie3=f (entrée2)} \end{pmatrix} = \begin{pmatrix} 0\ 11 \\ 1\ 0\ 0 \end{pmatrix}_{(2)} = \begin{pmatrix} 2 \\ 4 \end{pmatrix}_{(8)} \end{cases} \quad (4.9)$$

La matrice de transfert globale est donnée par :

$$G=(G_1\ G_2\ G_3)=\begin{pmatrix} 1\ 0\ 1 & 0\ 1\ 1 & 0\ 1\ 0 \\ 0\ 0\ 0 & 0\ 1\ 0 & 1\ 0\ 0 \end{pmatrix}_{(2)}=\begin{pmatrix} 5 & 3 & 2 \\ 0 & 2 & 4 \end{pmatrix}_{(8)} \quad (4.10)$$

IV.6.2 Représentations graphiques

IV.6.2.1 Diagramme d'état

Chaque bloc de n éléments binaires en sortie du codeur dépend :

- Du bloc de k éléments binaires présents à son entrée ;
- Des m blocs de k éléments binaires contenus dans sa mémoire.

Ces $m\times k$ éléments binaires définissent l'état du codeur.

Les conventions adoptées sont :

- Lorsque l'élément binaire d'entrée du codeur est égal à '0' (resp. '1'), le couple binaire en sortie du codeur est porté par la branche en trait solide (resp. en tirets).
- Seules deux transitions sont possibles à partir de chacun des états.
- Les étiquettes de chaque branche correspondent aux sorties du codeur.

Il y a $2^{m\times k}$ états possibles. Pour l'exemple de la Figure 4.1, le nombre d'états possibles est $2^{2\times 1}=4$.

Les quatre états possibles pour le diagramme d'état (Figure 4.3) sont : '00' '01' '10' '11'.

Généralement, on considère un état initial de « zéro partout » pour les registres à décalages.

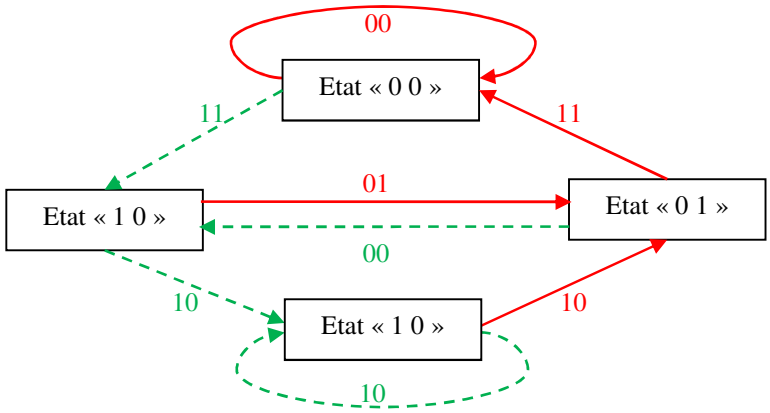


Figure 4.3 : Diagramme d'état d'un codeur convoluif de l'exemple de la Figure 4.1.

La distance minimale est le poids du chemin partant de '00' et revenant le plus vite possible (en comptant le minimum de nombre de 1 un tel que le poids est supérieur à zéro).
Pour l'exemple précédent, le chemin le plus court est : 11 01 11. Le poids est 5, donc la distance minimale est : d_{min}=5.

IV.6.2.2. L'arbre du code

L'arbre du code (Figure 4.4) est un développement du diagramme d'état en fonction du temps discrétisé**.** Les conventions adoptées sont :

- Le temps s'écoule de la gauche vers la droite
- Lorsque l'élément binaire d'entrée du codeur est égal à '0' (resp. '1'), le couple binaire en sortie du codeur est porté par la branche supérieure (resp. Inférieure).
- Les branches se séparent en un point appelé nœud. Chaque nœud donne naissance à 2^k (q^k) branches.
- Quelque soit l'état initial du codeur, après (m + *1*) décalages à l'entrée du codeur, tous les états du codeur peuvent être atteints.

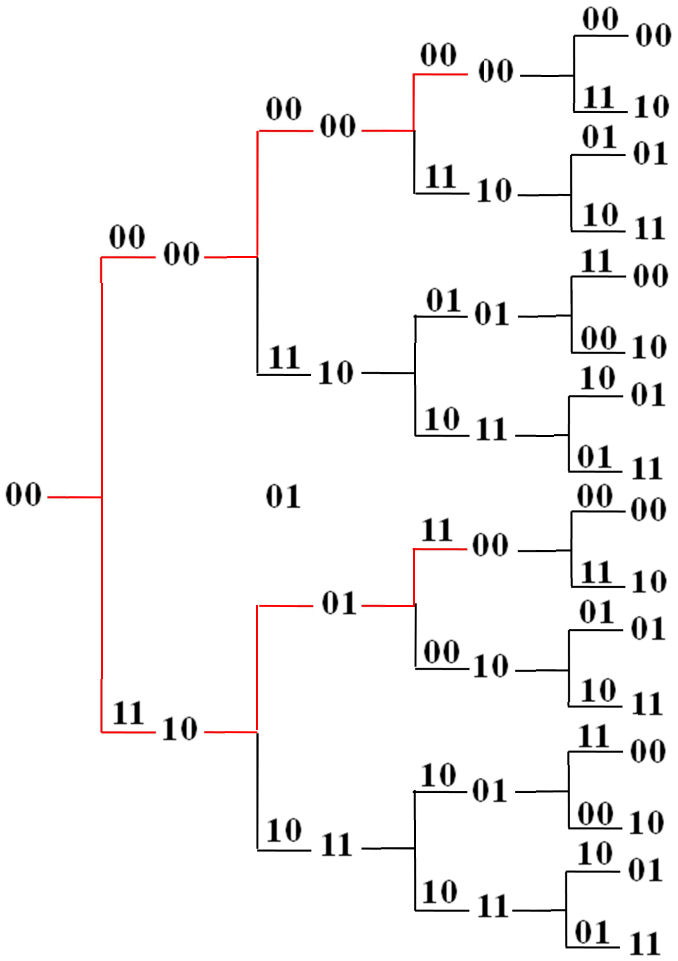


Figure 4.4 : L'arbre de code convolutif de l'exemple de la Figure 4.1.

Partant de l'état « 00 » à l'instant $t=j$, il existe deux chemin pour atteindre l'état « 00 » à l'instant $t=j+3$:

Chemin 1 : 11 01 11

Chemin 2 : 00 00 00

La distance minimale de ce code est donc : $d_{min}=5$

IV.6.2.3. Treillis du code

Les conventions adoptées pour cette représentation sont :

- Lorsque l'élément binaire d'entrée du codeur est égal à '0' (resp. '1'), le couple binaire en sortie du codeur est porté par la branche en trait solide (resp. en tirets).
- De chaque nœud partent 2^k (q^k) branches.
- En chaque nœud convergent 2^k (q^k) branches.
- Les étiquettes de chaque branche correspondent aux sorties du codeur.

Exemple : Représentons le treillis de code relatif au codeur de la Figure 4.1. Il y a $2^m=2^2=4$ états possibles : « 00 » « 01 » « 10 » « 11 ».

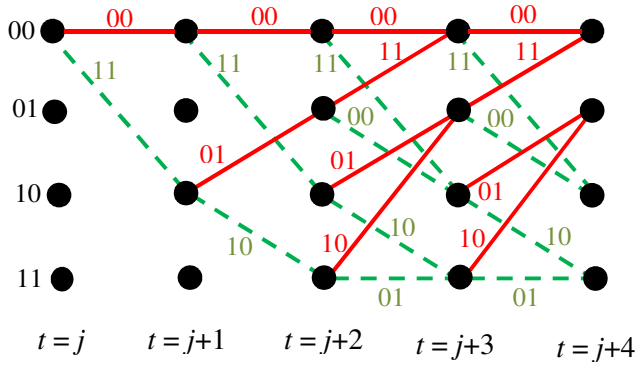


Figure 4.5 : Treillis de codeur convolutif de l'exemple de la Figure 4.1.

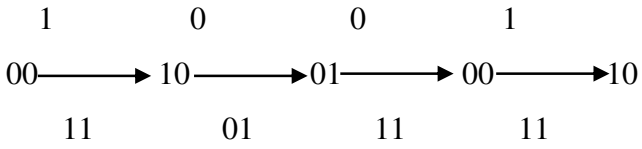
Remarque : Après (m + 1) décalages, quelque soit l'état initial du codeur, le motif du treillis se répète.

Comme pour le diagramme en arbre, partant de l'état « 00 » à l'instant t = j, il existe deux chemins pour atteindre l'état '**00**' à l'instant $t = j + 3$:

Chemin 1 : 00 00 00

Chemin 2 : 11 01 11, poids=5 qui correspond à d_{min}=5.

Cherchons maintenant le mot de code correspondant à la séquence d'information « 1 0 0 1… »



Le mot de code associé à « 1 0 0 1… » est : 11 01 11 11

S'il s'agit d'un codage systématique, le mot à transmettre est donc : v=**1**11**0**01**0**11**1**11

IV.7 Codes particuliers

IV.7.1 Code convolutif systématique

Un code convolutif est dit systématique lorsqu'une partie de ses sorties est exactement égale à ses entrées. Cela revient à dire que la réponse impulsionnelle est de la forme :

$$H(D)= (1 \quad \ldots\ldots \quad \ldots\ldots) \qquad (4.11)$$

Exemple : Soit le codeur convolutif systématique de la Figure 4.6.

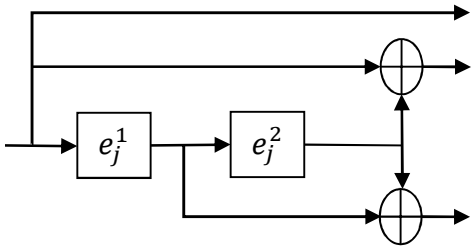


Figure 4.6 : Codeur convolutif systématique (r=1/3, n=3, m=2, L=3).

La réponse impulsionnelle de ce code est :

$$H(D)= (1 \quad 1+D^2 \quad D+D^2) \tag{4.12}$$

et la matrice de transfert a la forme :

$$G= (4 \quad 5 \quad 3)_{\text{octal}} \tag{4.13}$$

Figure 4.7 présente le treillis de code du codeur systématique de la Figure 4.6.

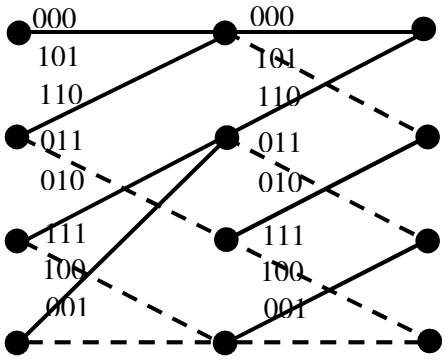


Figure 4.7 : Treillis de codeur convolutif systématique de l'exemple de la Figure 4.6.

IV.7.2 Code convolutif récursif Systématique

Un code convolutif est dit récursif lorsqu'une partie de la sortie est alors réintroduite dans le registre à décalage. Figure 4.8 illustre un codeur convolutif récursif systématique.

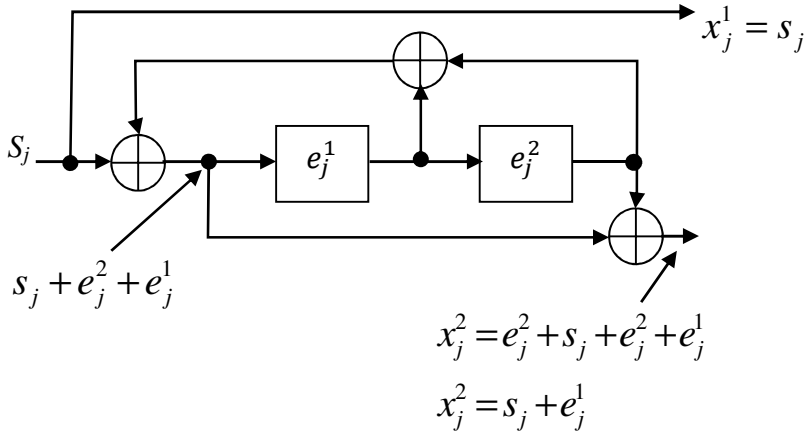


Figure 4.8 : Circuit logique d'un codeur convolutif récursif systématique (r=1/2, n=2, m=2, L=3).

Ce code se caractérise par la réponse impulsionnelle calculée comme suit :

$$\begin{cases} X^1(D)=1.\ s(D) \\ X^2(D)=s(D)+e^1(D) \\ e^1(D)=D\ [s(D)+\ e^2(D)+\ e^1(D)] \\ e^2(D)=D\ e^1(D) \\ e^1(D)=D\ [s(D)+\ D\ e^1(D)+\ e^1(D)] =D\ s(D)+D^2\ e^1(D)+\ D\ e^1(D) \\ s(D)= (1+D+D^2)\ e^1(D) \\ e^1(D)=\frac{D}{1+D+D2}\ s(D) \\ X^2(D)=s(D)+\frac{D}{1+D+D2}\ s(D)= \frac{1+D^2}{1+D+D^2} \end{cases} \quad (4.14)$$

La réponse impulsionnelle est donc :

$$H(D)= (1 \quad \frac{1+D^2}{1+D+D^2}) \quad (4.15)$$

IV.7.2 Les Codes Catastrophique

Un code catastrophique est un code qui génère un nombre infini d'erreurs tel que :

- Une séquence d'information de poids infinie est codée par une séquence de poids fini,
- Le décodeur, recevant une séquence de poids fini, estimera que la séquence d'entrée était constituée d'un mot de poids fini suivi de zéros.

IV.8 Décodage Convolutif -Algorithme de Viterbi

Il s'agit d'un algorithme de décodage (des décisions sont prises sur chaque transition du chemin choisi dans le treillis) qui calcule le chemin dont la vraisemblance est maximale (notée ML pour Maximum Likelihood). Il s'agit donc d'un algorithme de décodage ML par séquence (chemin). Ce chemin correspond à la séquence décodée [25].

Le déroulement de l'algorithme de Viterbi se fait comme suit [25] :

- À chaque instant, deux branches appartenant à deux chemins différents, convergent vers chaque nœud,
- De ces deux chemins, l'un est plus vraisemblable, c'est-à-dire se trouve à une distance plus petite de la séquence reçue, que l'autre chemin,
- Les distances étant additives, il est possible de ne conserver en chaque nœud que le chemin le plus vraisemblable, appelé *survivant*,
- Si deux chemins sont aussi vraisemblables, un seul chemin est arbitrairement conservé.

Exemple : Reprenons le code convolutif de Figure 4.1 et considérons que la séquence à l'entrée du codeur est '1 0 0 1'.

Si le codeur est dans l'état '00' à l'instant initial, la séquence correspondante en sortie du codeur est '11 01 11 11'.

Considérons un canal binaire symétrique introduisant une erreur en position 4. La séquence reçue à l'entrée du décodeur est '11 00 11 11'.

- Voici le déroulement de l'algorithme de Viterbi [25]
  - À l'instant t = 0 (Figure 4.9)

Deux branches partent de l'état '00'. Elles sont respectivement aux distances **2** et **0** du premier couple binaire reçu. Reportons ces deux distances, appelées *métriques de branche* sur le treillis.

- À l'instant *t* = 1 (Figure 4.9)

Évaluons la distance entre le deuxième couple binaire reçu et les quatre branches qui partent des états '00' et '10', puis reportons ces quatre métriques sur le treillis.

En sommant les métriques de branches appartenant à un même chemin, nous obtenons les *métriques cumulées.*

Nous avons désormais quatre chemins qui permettent d'accéder, en t = 2, aux quatre états possibles du codeur.

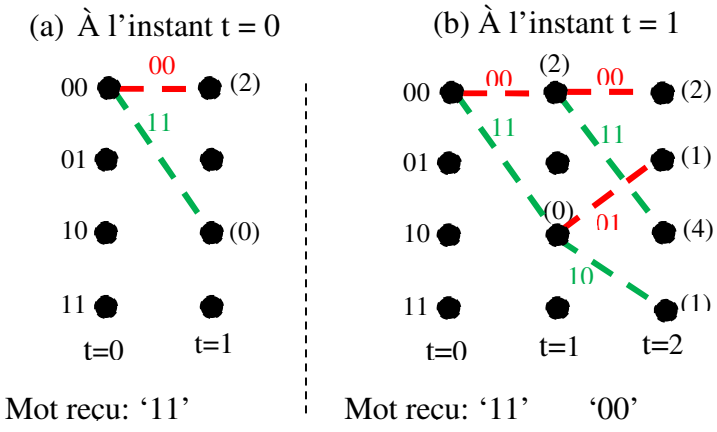


Figure 4.9 : Exemple de déroulement de l'algorithme de Viterbi : (a) à l'instant t=0 et (b) à t=1.

- À l'instant t = 2 : (Figure 4.10)

Il existe désormais deux chemins qui convergent vers chaque nœud du treillis.

Nous avons donc à :

1. Calculer les métriques de branche,
2. Calculer les métriques cumulées pour chaque chemin atteignant en t = 3, un nœud donné du treillis,
3. En chaque nœud, ne retenir que le survivant.

➢ À l'instant t = 3 : (Figure 4.10)

On procède de la même façon.

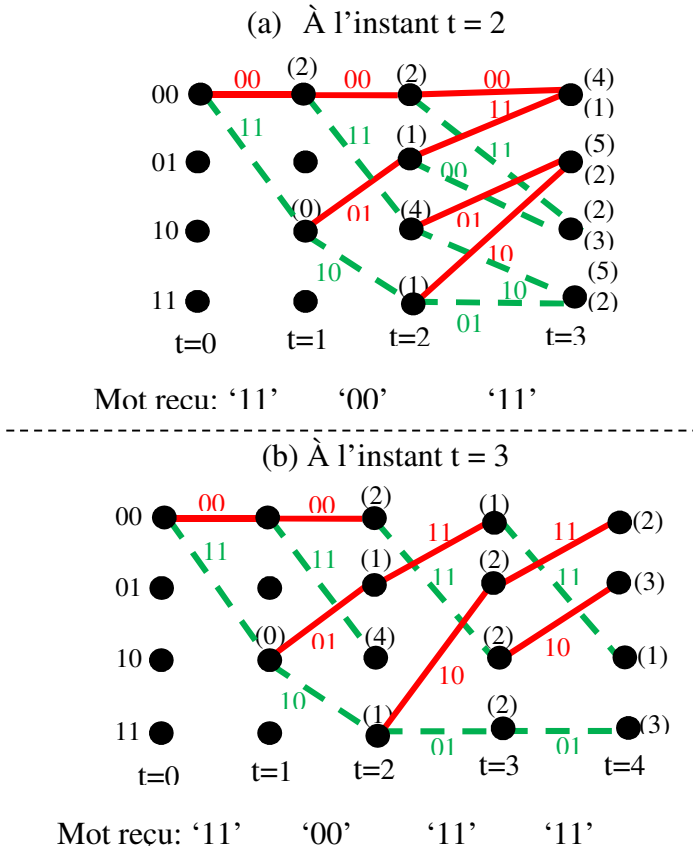


Figure 4.10 : Exemple de déroulement de l'algorithme de Viterbi : (a) à l'instant t=2 et (b) à t=3.

Finalement, le chemin le plus vraisemblable est celui qui arrive en '10' (Figure 4.11). En remontant le treillis de la droite vers la gauche, on voit que la séquence la plus vraisemblable est celle qui part de '00' à t = 0 et qui arrive à '10' à t = 4. Elle correspond au code vraisemblablement émis : '11 01 11 11'. Ce code correspond à une séquence sur l'entrée du codeur égale à '1001'. L'erreur en position 4 est donc corrigée.

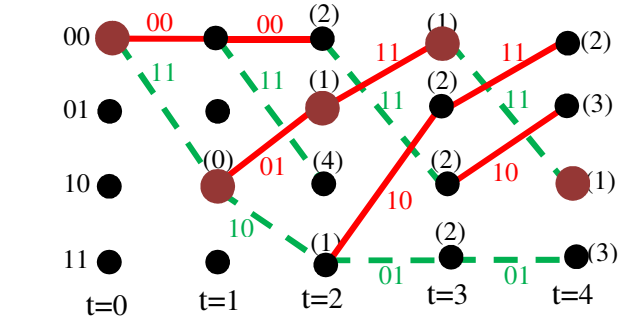


Figure 4.11 : Chemin de décodage de Viterbi relatif à la séquence reçu '11 00 11 11'.

IV.10 Applications Des Codes Convolutifs

IV.10.1 Les codes convolutifs dans GSM (Figure 4.12)

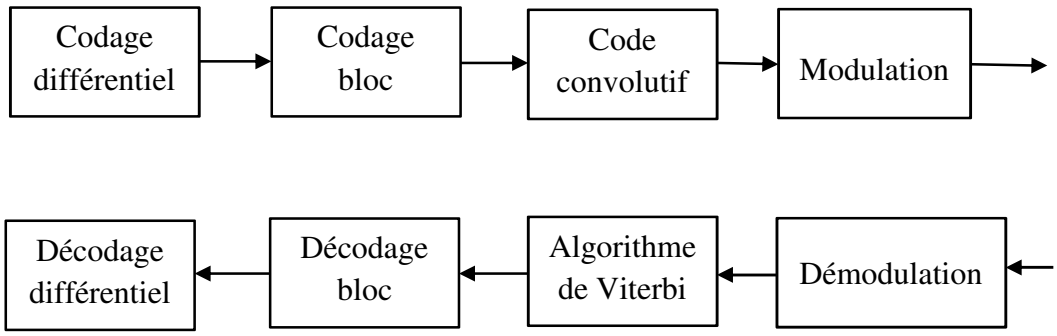


Figure 4.12: Schéma de bloc de transmission de voix dans la norme GSM [26].

IV.10.2 Les codes convolutifs dans transmission satellitaire (Figure4.13)

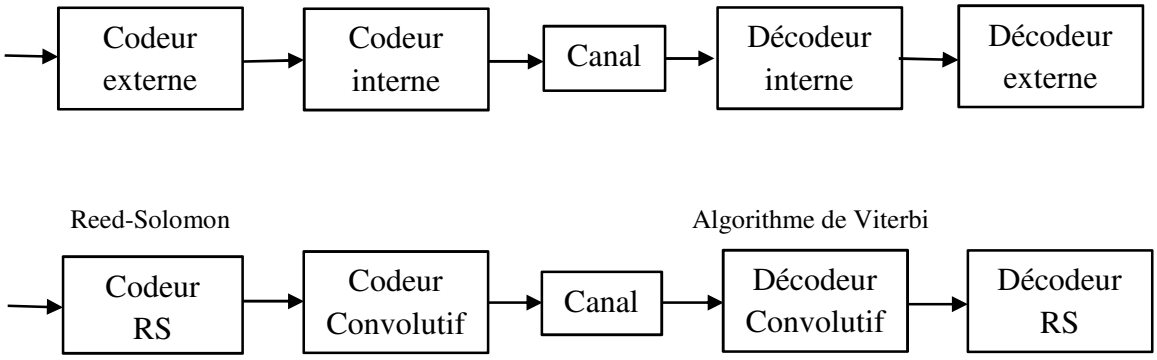


Figure 4.13 : Exemple de chaine de transmission satellite-terrestre correspondante à la télévision numérique. Le code convolutif est de rendement r=1/2 et de longueur de contrainte L=*7* [27].

IV.11 Conclusion

Les codes convolutifs sont aussi des codes linéaires, mais contrairement aux codes LBC qui traitent chaque bloc d'information indépendamment les uns des autres, les codes convolutifs délivrent un mot de code binaire dépendant du symbole courant à coder ainsi que du symbole précédent et du résultat de codage du symbole précédent. On parle d'effet de mémoire.

La performance des codes convolutifs est d'autant meilleure que le nombre de mémoires augmente mais ceci complique davantage le processus de décodage qui utilise souvent l'algorithme de Viterbi. Le même problème de compromis entre débit, fiabilité et complexité se pose. Cependant, il est connu dans la pratique de codage que les codes convolutifs forment comme même une famille extrêmement souple et fiable de codes détecteurs et correcteurs d'erreur. Ceci explique leur utilisation très courante dans les réseaux de communications fixes et mobiles.

Dans chapitre cinq, nous décrivons les schémas synoptiques relatifs à un système numérique de transmission des signaux audio sans et avec codage en considérant une compression sans pertes. Nous visualisons ensuite une série de résultats obtenus en termes de sons reçus et de TEBs réalisés dans les différents tests. Des interprétations détaillées sont aussi explicitées à la fin de ce chapitre.

Chapitre V

Résultats de l'application de codage de canal sur la transmission des signaux audio

V.1Introduction

Ce chapitre présente un ensemble de tests illustrant l'amélioration de la fiabilité de transmission numérique en introduisant le codage de source avec compression de données et le codage de canal. Notre but principal est de limiter les effets perturbateurs du bruit sur l'information de source en minimisant la probabilité d'erreur comme indiqué par le deuxième théorème de Shannon. De plus, nous avons testé un schéma de compression avec pertes lors de transmission des signaux audio en introduisant un sous-échantillonnage de facteur N=8 à l'émission. En réception, nous avons effectué un sur-échantillonnage de même facteur N. En fait, cette idée était inspirée de la compression de matrices de chrominance utilisée dans la vidéo et plus particulièrement en télévison numérique (les fameux formats (4 :2 :2), (4 :1 :1)…).

Notre principal intérêt est de considrer un système numérique d'émission-réception des signaux audio sur un canal perturbé par un bruit Gaussienne blanc additif (AWGN) et nous démontrons que l'utilisation d'un code détecteur d'erreur convenable permet de réduire considérablement le TEB réalisé par rapport à une transmission sans codage de canal.

V.2 Compression par l'algorithme de Huffman avec codage de canal :

Considérons un exemple de transmission numérique d'un texte brut de longueur L=35 selon deux schémas :

Schéma 1 : Compression par l'algorithme de Huffman sans codage de canal,

Schéma 2 : Compression par l'algorithme de Huffman avec codage de canal.

Le texte brut est :

« SYSTEMES DES TELECOMMUNICATIONS »

On considère deux valeurs de RSB. Les messages reçus par les deux schémas sont reportés dans le Tableau 5.1 tandis que la comparaison en termes de nombre de caractères reçus erronés est affichée au Tableau 5.2.

Tableau 5.1: Messages reçus par les deux schémas pour différentes valeurs de RSB.

	Message reçu par Schéma 1	Message reçu par Schéma 2
RSB= 10 dB	MEDUYDSUYMIONUOEMNOEDCOM YDTYU EOOIA	SYSTEMES DES TELECOMMUNICATIOIOO
RSB= 17 dB	CSDUMOOYOI U NOCADELODDOOSMLNCLDIY	SYSTEMES DES TELECOMMUNICATIONS

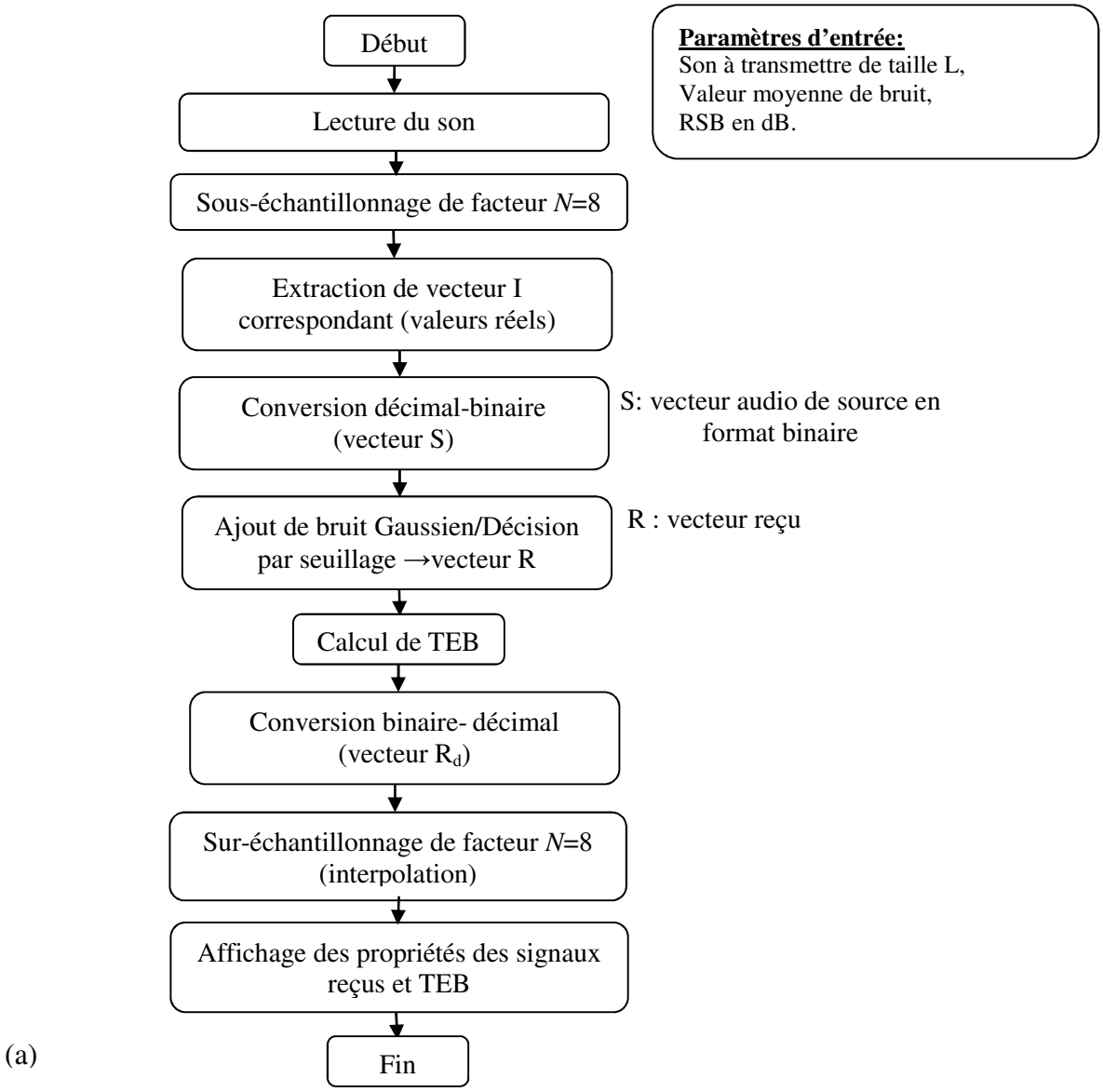
Tableau 5.2: Comparaison des résultats de transmission d'un texte brut sur un canal bruité avec compression sans pertes et correction d'erreurs.

	Nombre de caractères reçus erronés (Schéma 1) N_{e1}	Nombre de caractères reçus erronés (Schéma 2) N_{e2}	N_{e1}/L	N_{e2}/L
RSB= 10 dB	33	3	94%	0.08%
RSB= 17 dB	29	0	82%	0%

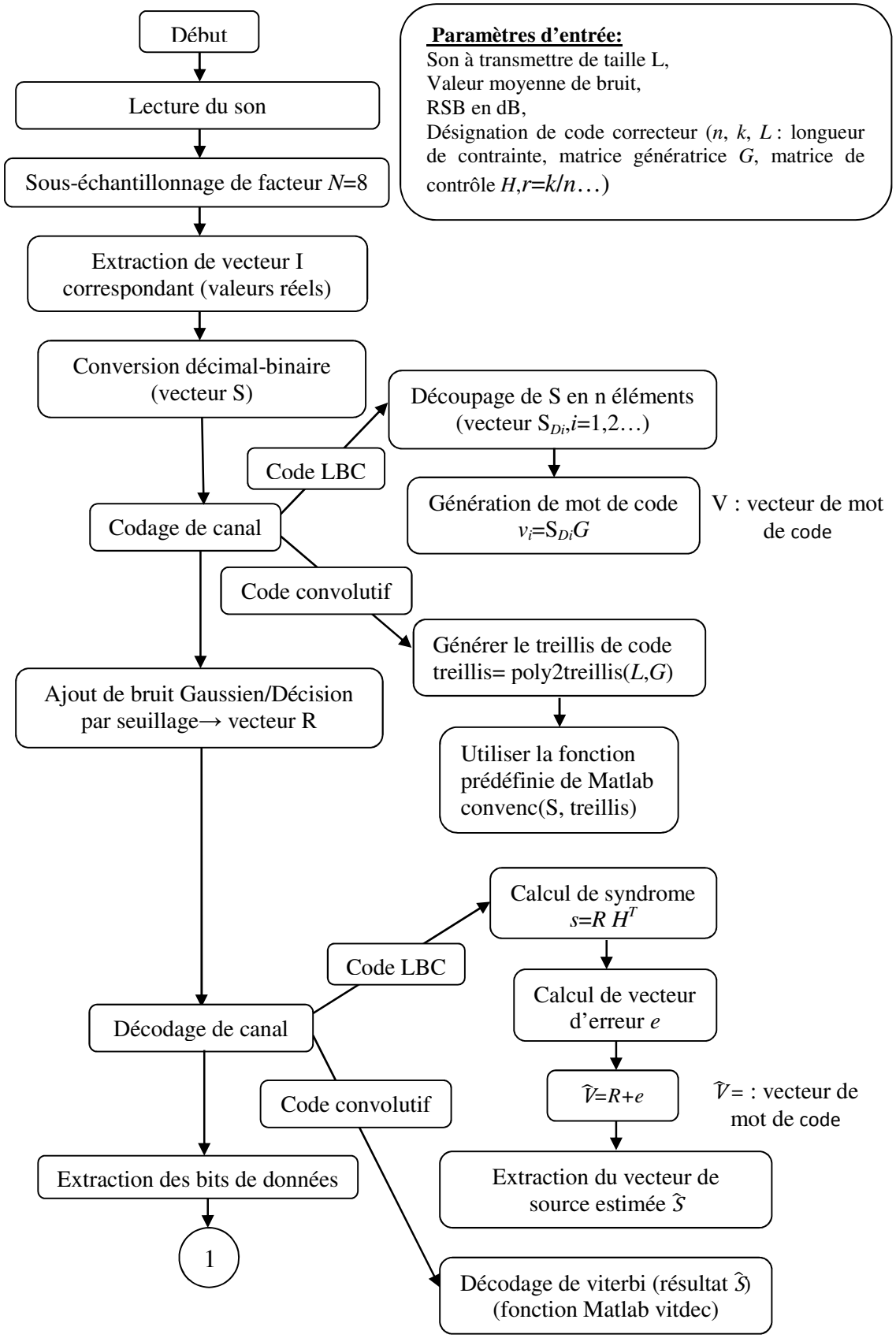
V.3 Schéma synoptique d'un système numérique de transmission de sons sans et avec codage de canal

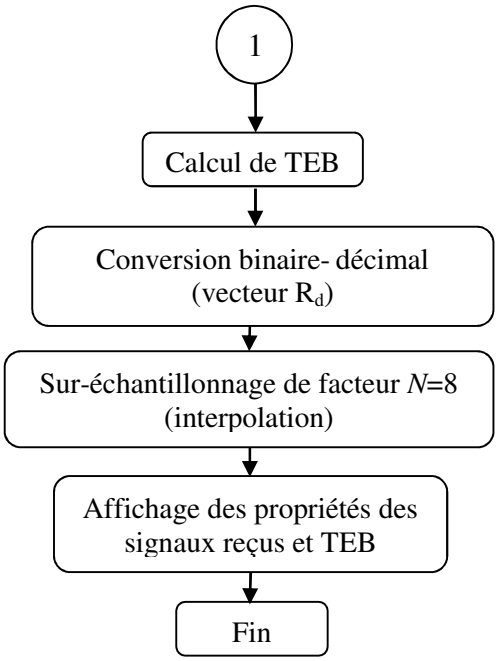
Figure 5.1 présente les organigrammes permettant de lire, traiter, coder, décoder et visualiser les résultats obtenus à savoir le son de source, le son bruité et le son reçu sans et avec correction d'erreurs ainsi que le TEB réalisé. Les signaux audio sources sont compressés, de format wav et mp3. Les codes Matlab détaillés permettant le calcul des paramètres de sortie et l'affichage des résultats sont présentés en Annexe A.

a) Organigramme relatif à une transmission du son sans codage de canal



(a)


Début
Lecture du son
Sous-échantillonnage de facteur N=8
Extraction de vecteur I correspondant (valeurs réels)
Conversion décimal-binaire (vecteur S)
Codage de canal
Code LBC
Découpage de S en n éléments (vecteur S_{Di},i=1,2…)
Génération de mot de code $v_i=S_{Di}G$
V : vecteur de mot de code
Code convolutif
Générer le treillis de code treillis= poly2treillis(L,G)
Utiliser la fonction prédéfinie de Matlab convenc(S, treillis)
Ajout de bruit Gaussien/Décision par seuillage→ vecteur R
Paramètres d'entrée:
Son à transmettre de taille L,
Valeur moyenne de bruit,
RSB en dB,
Désignation de code correcteur (n, k, L : longueur de contrainte, matrice génératrice G, matrice de contrôle H,r=k/n…)
Décodage de canal
Code LBC
Calcul de syndrome $s=R\,H^T$
Calcul de vecteur d'erreur e
$\hat{V}=R+e$
$\hat{V}$= : vecteur de mot de code
Code convolutif
Extraction des bits de données
Extraction du vecteur de source estimée $\hat{S}$
1
Décodage de viterbi (résultat $\hat{S}$) (fonction Matlab vitdec)



(b)

Figure 5.1 : Organigrammes d'un système numérique de transmission des sons :(a) sans codage de canal et (b) avec codage de canal.

IV.4 Résultats Expérimentaux et Discussion

En ce qui suit, nous allons examiner l'effet de codage de canal sur la transmission des sons à travers un canal bruité caractérisé par un RSB exprimé en dB. Le bruit est supposé Gaussien de valeur moyenne *moy* et de variance :

$$\sigma^2 = 10^{RSB_{dB}/10} \qquad (5.1)$$

La comparaison entre la qualité du son reçu sans et avec codage de canal se base sur : (1) Entendre des sons à la réception sans et avec codage et (2) tracés des TEB en fonction de RSB_{dB}.

Quatre différents schémas de codeurs sont considérés à savoir : Code de Hamming systématique (7,4,3), code de Hamming non-systématique (7,4,3), code de Hamming systématique (15,11,3) ainsi qu'un code convolutif dont le circuit logique et les caractéristiques sont présentés dans Figure 5.2.

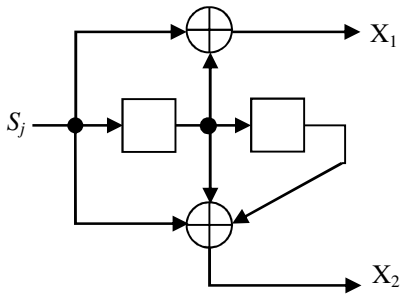
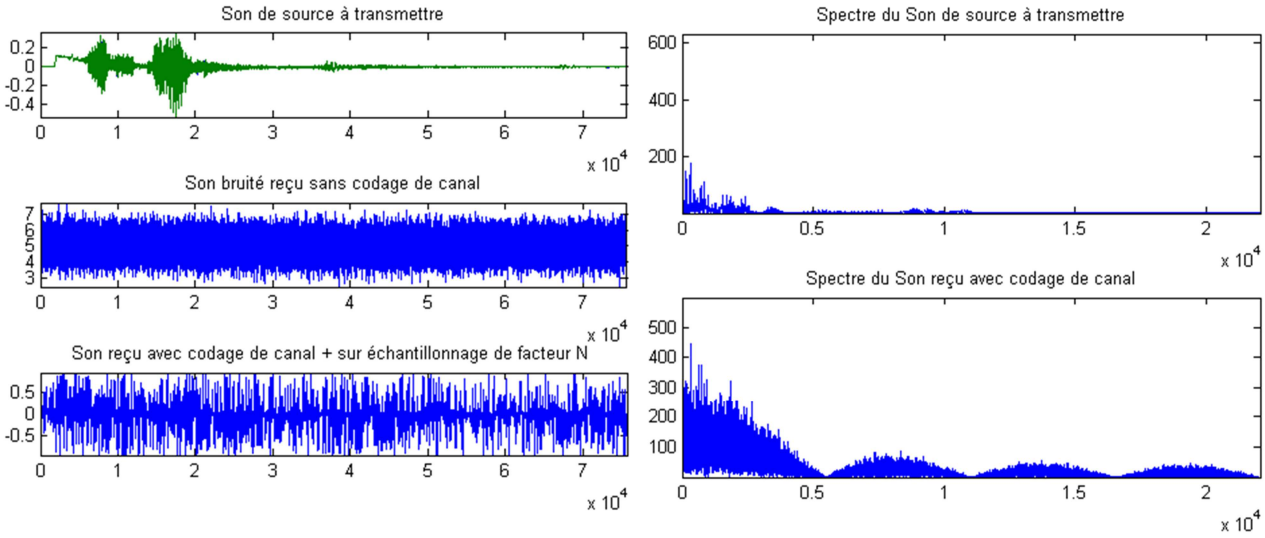


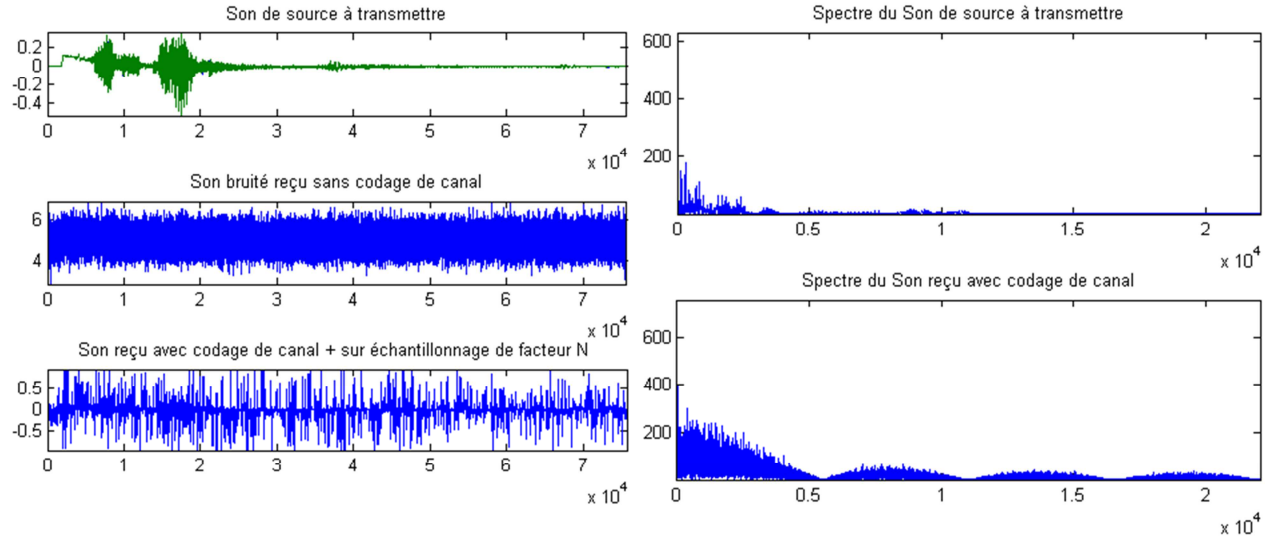
Figure 5.2 : Circuit logique de codeur convolutif $r=k/n=1/2$, $m=2$, $G=(6\ 7)_8=(110\ 111)_2$et$L=(m+1)k=3$.

Test 1 : Effets de codage de canal sur la transmission d'un son wav ('Test 1')

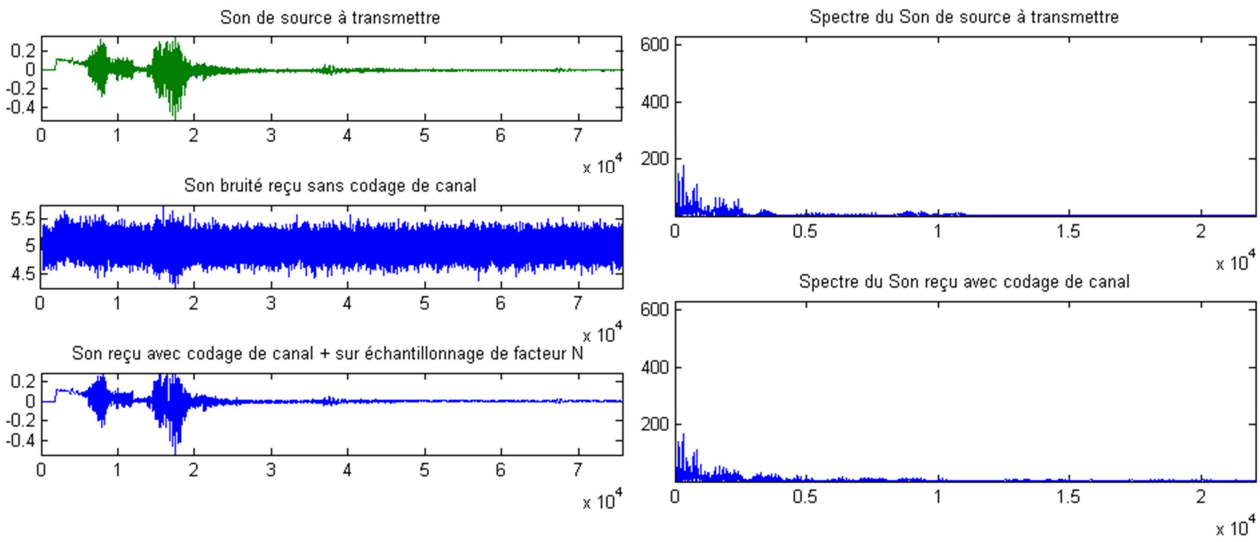
a.1. Code de Hamming systématique (7,4,3)



(a)



(b)



(c)

Figure 5.3 : Résultats de codage de canal relatifs au test 1 en utilisant le code de Hamming systématique (7,4,3) : (a) RSB=4 dB, (b) RSB=7 dB et (c) RSB=17 dB.

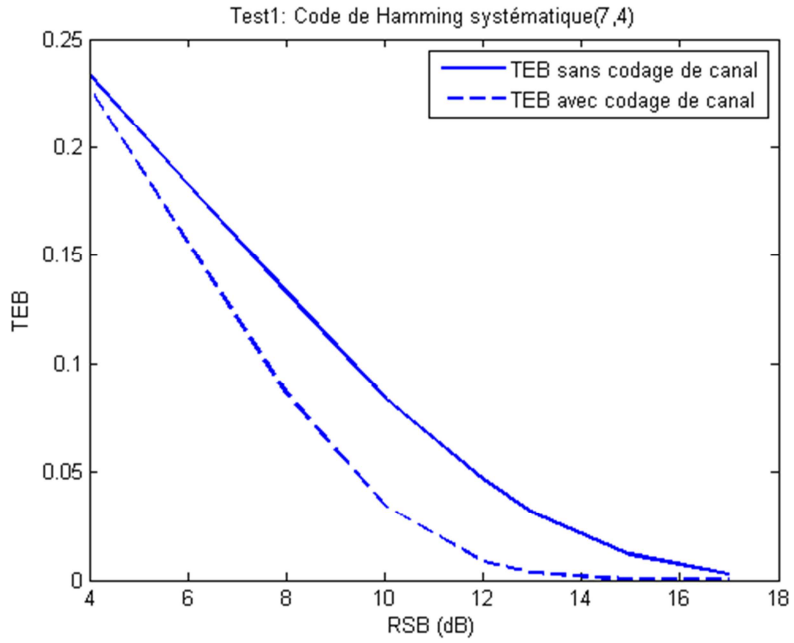
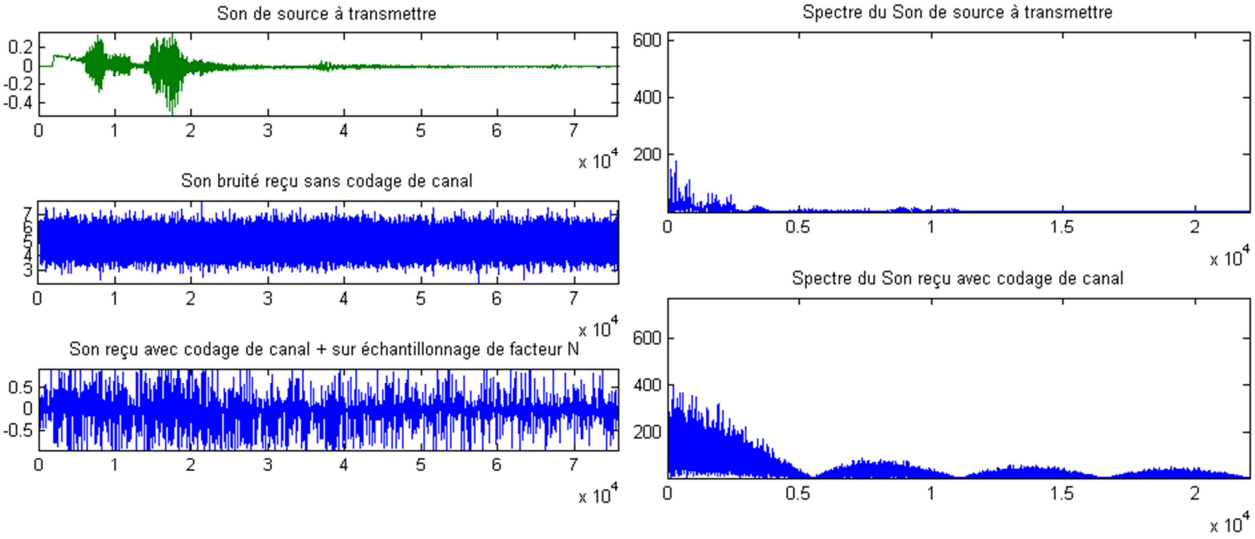
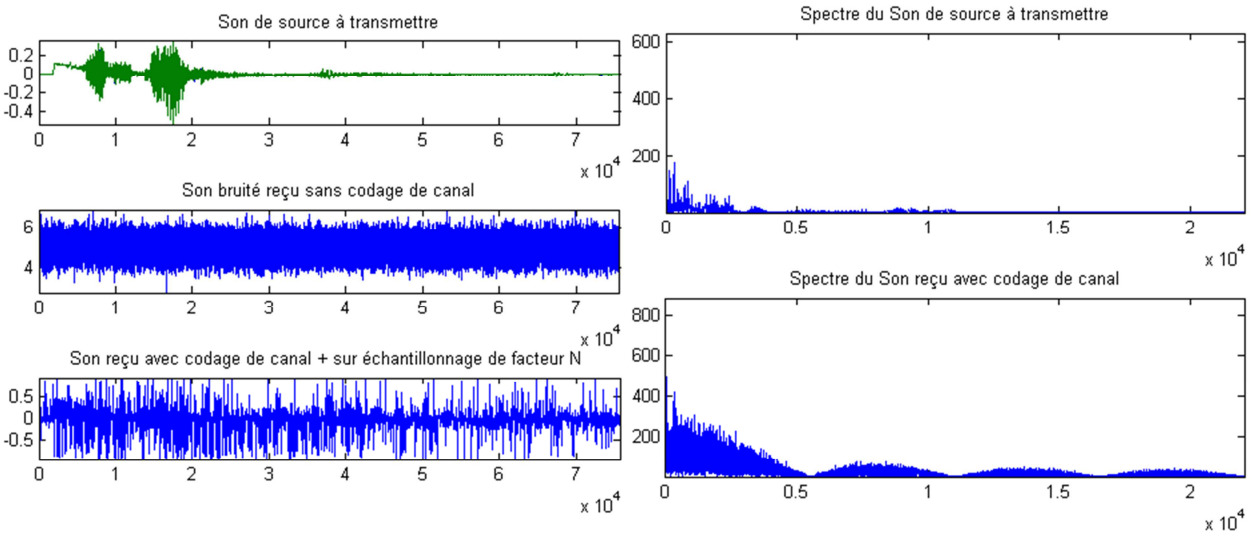


Figure 5.4 : Tracés du TEB relatifs au test 1en utilisant le code de Hamming systématique (7,4).

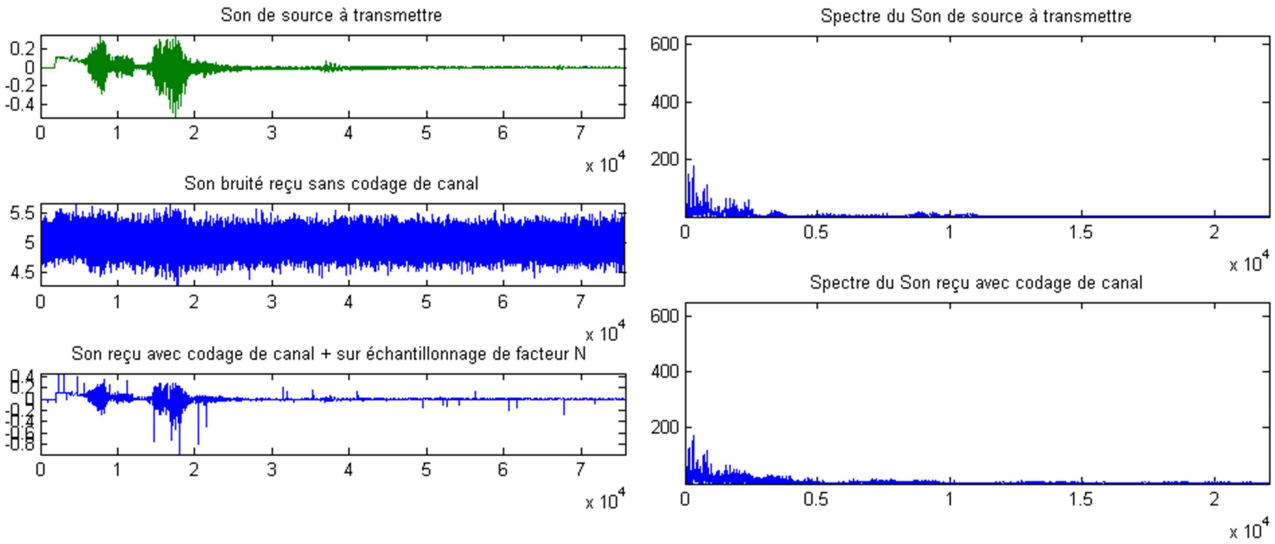
a.2. Code de Hamming non-systématique (7,4,3)



(a)



(b)



(c)

Figure 5.5 : Résultats de codage de canal relatifs au test 1 en utilisant le code de Hamming non-systématique (7,4,3) : (a) RSB=4 dB, (b) RSB=7 dB et (c) RSB=17 dB.

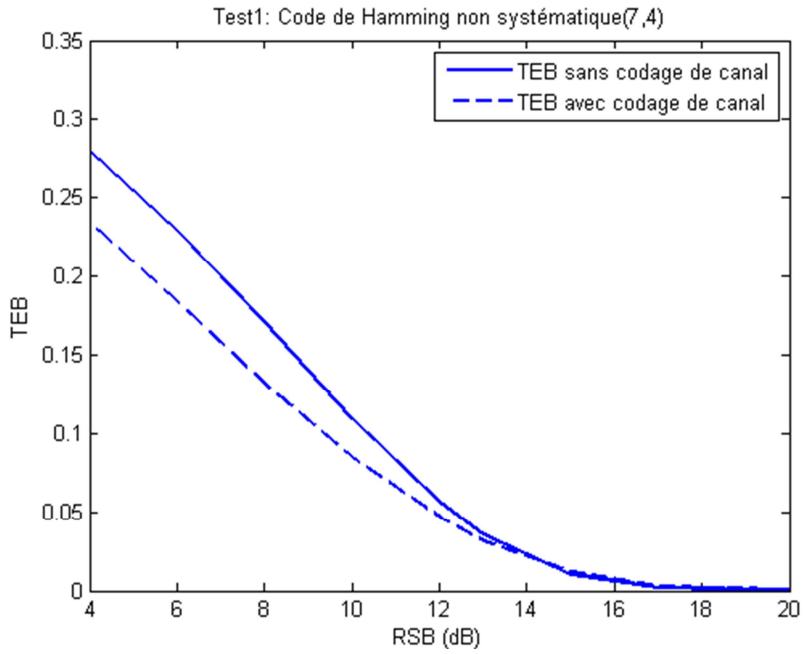
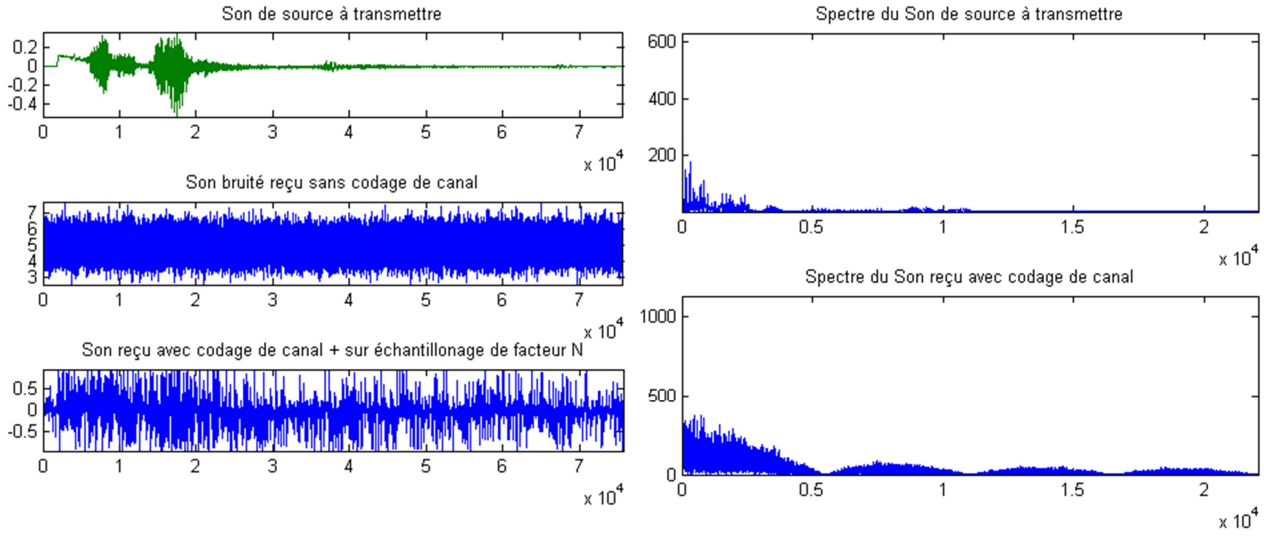
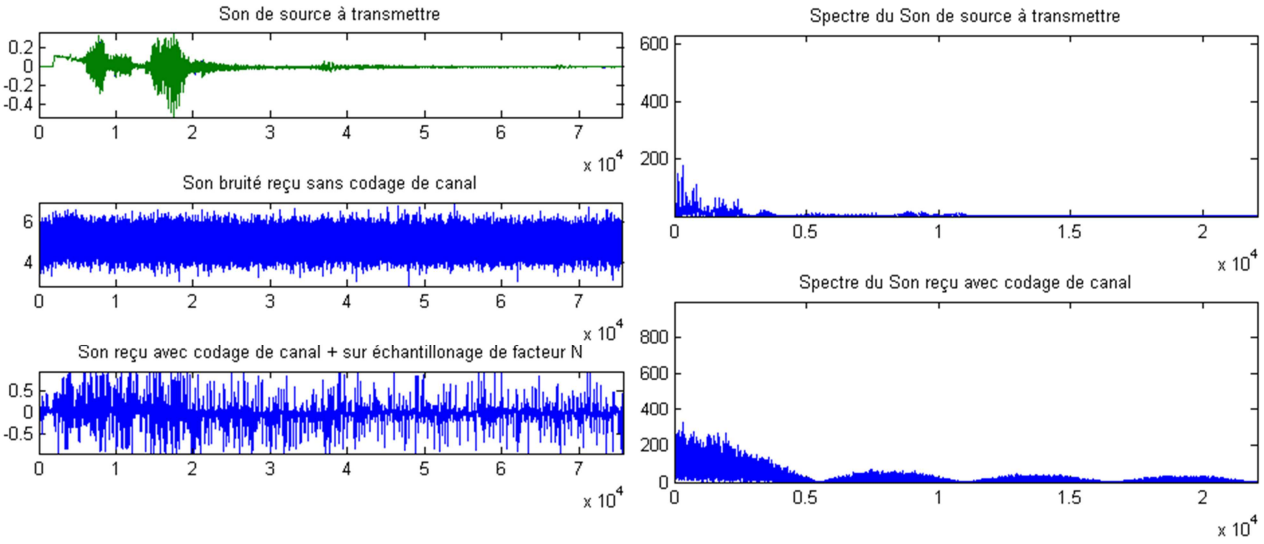


Figure 5.6 : Tracés du TEB relatifs au test 1 en utilisant le code de Hamming non-systématique (7,4).

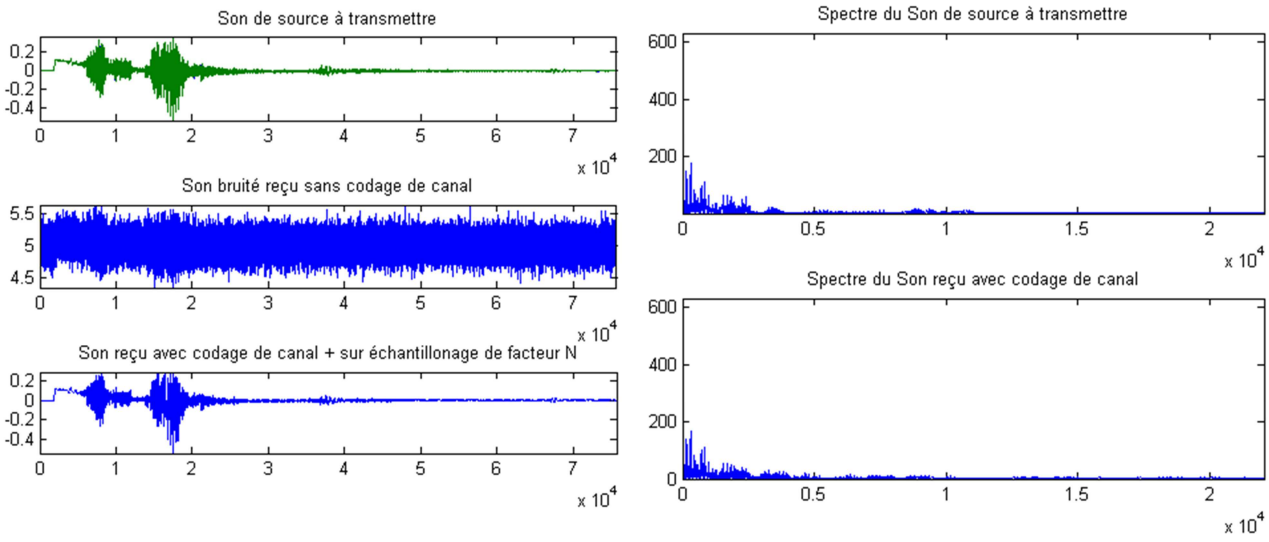
a.3. Code de Hamming systématique (15,11,3)



(a)



(b)



(c)

Figure 5.7 : Résultats de codage de canal relatifs au test 1 en utilisant le code de Hamming systématique (15,11,3) : (a) RSB=4 dB, (b) RSB=7 dB et (c) RSB=17 dB.

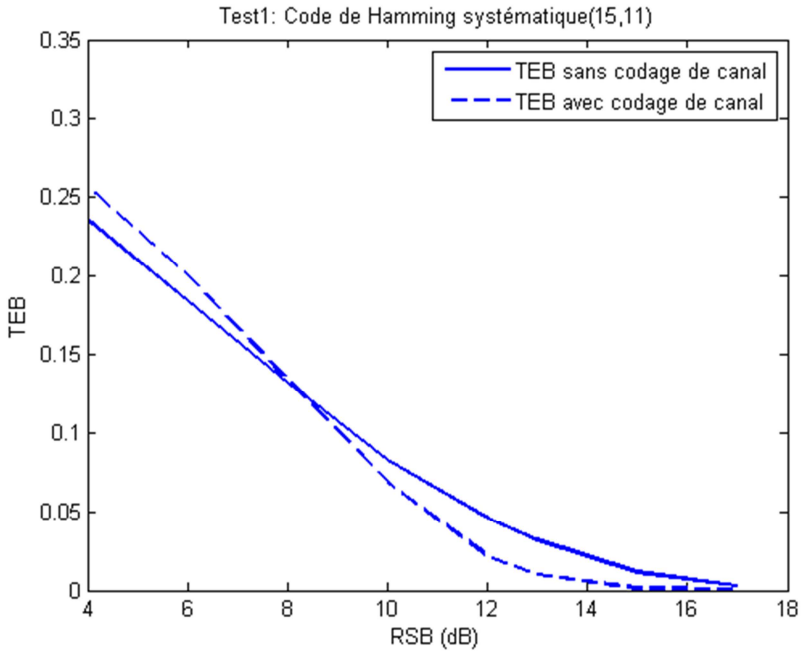
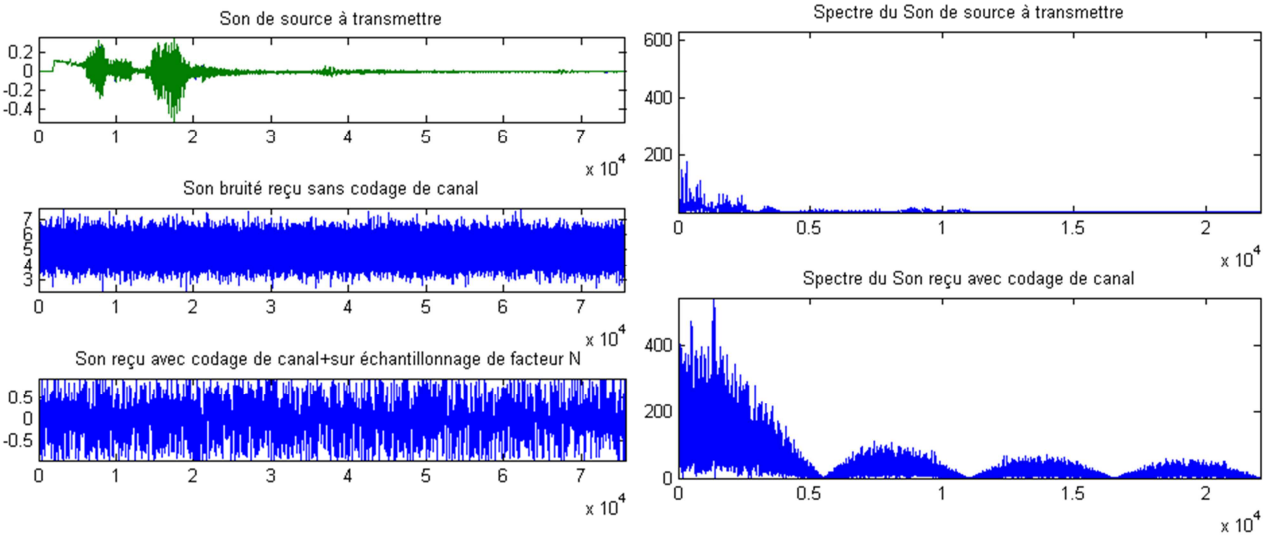
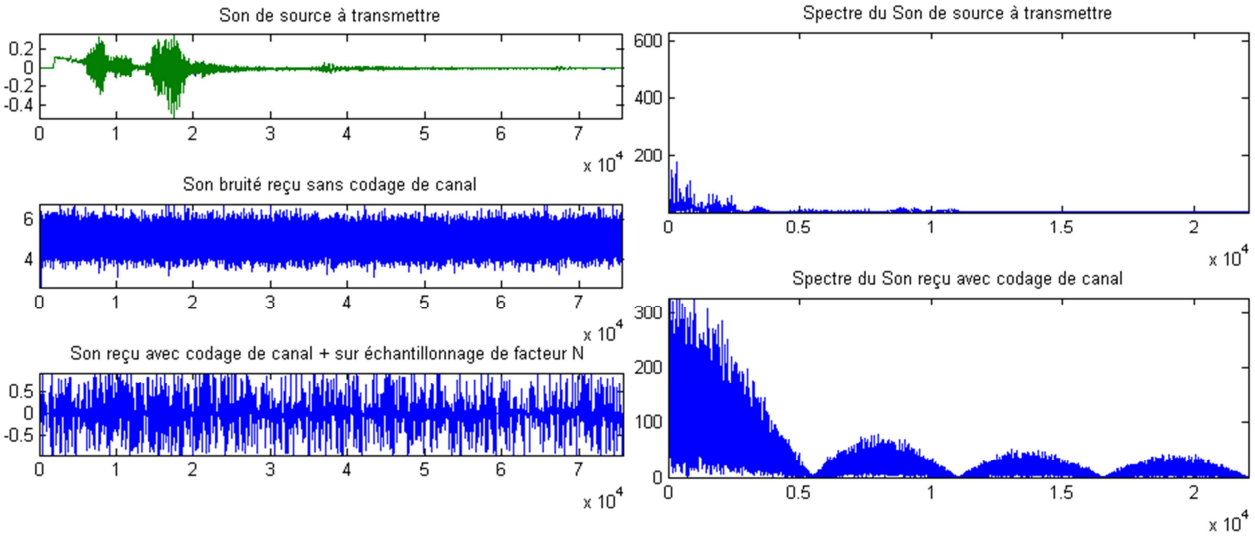


Figure 5.8 : Tracés du TEB relatifs au test 1 en utilisant le code de Hamming systématique (15,11).

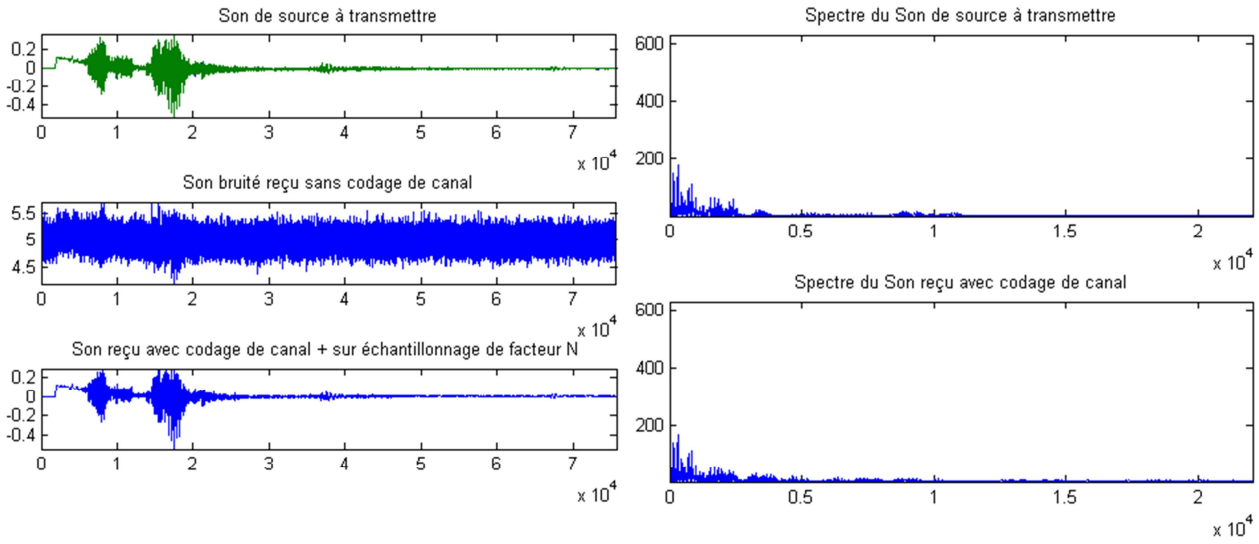
a.4. Code convolutif r=1/2, G=(171 133), L=7, profondeur=8



(a)



(b)



(c)

Figure 5.9 : Résultats de codage de canal relatifs au test 1 en utilisant le code convolutif *r*=1/2, *G*=(171 133), *L*=7, profondeur=8 : (a) RSB=4 dB, (b) RSB=7 dB et (c) RSB=17 dB.

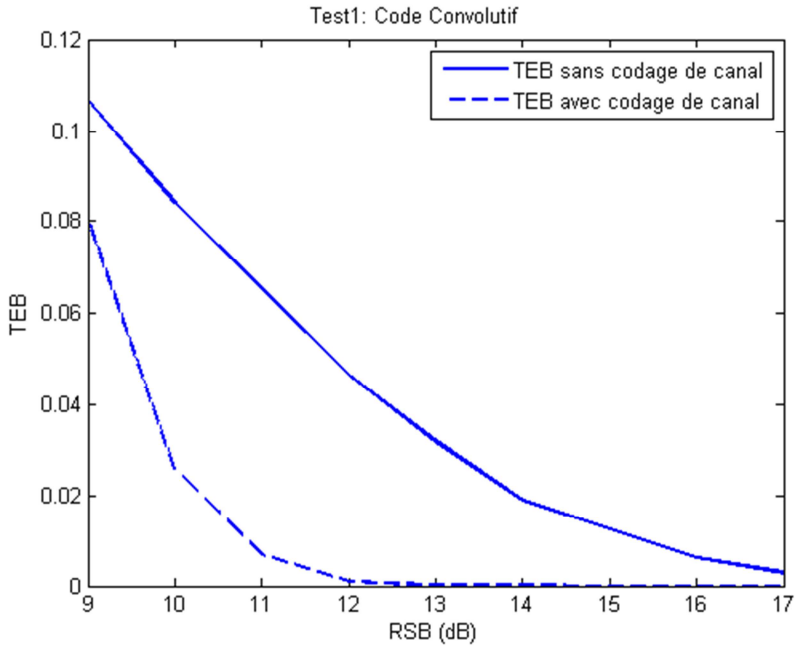
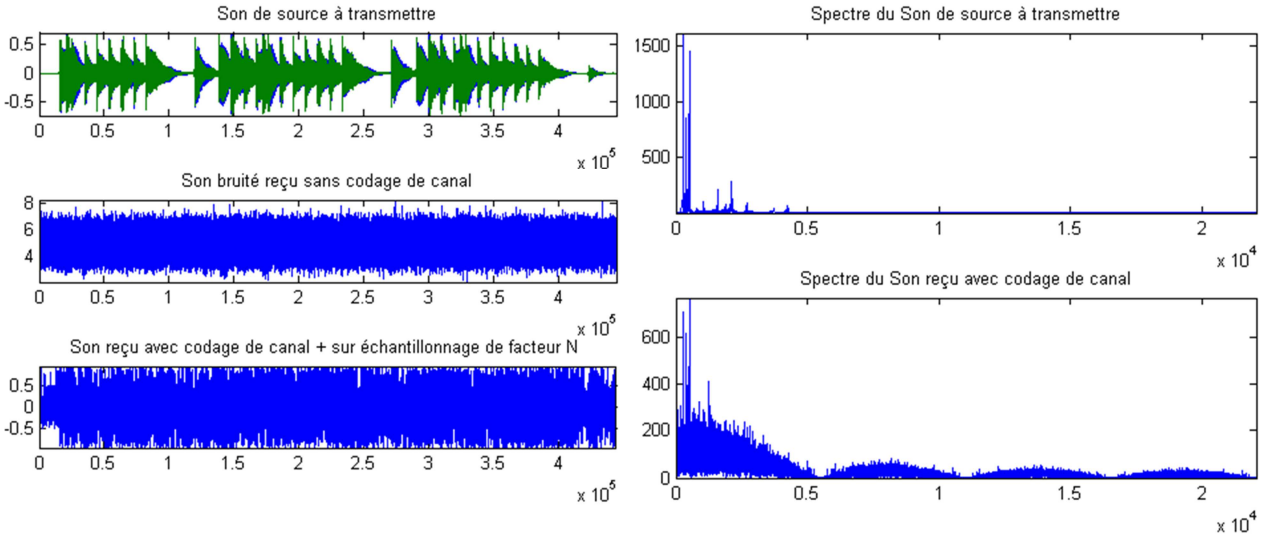


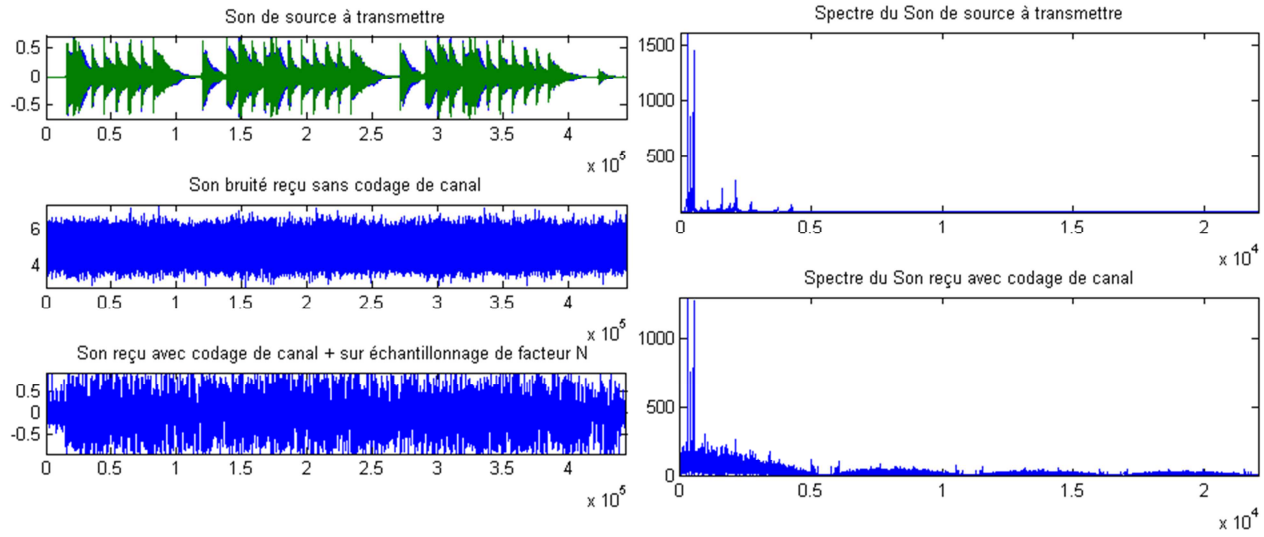
Figure 5.10: Tracés du TEB relatifs au test 1 en utilisant le code convolutif *r=1/2, G=(171 133), L=7, profondeur=8.*

Test 2 : Effets de codage de canal sur la transmission d'un son mp3 ('Test 2')

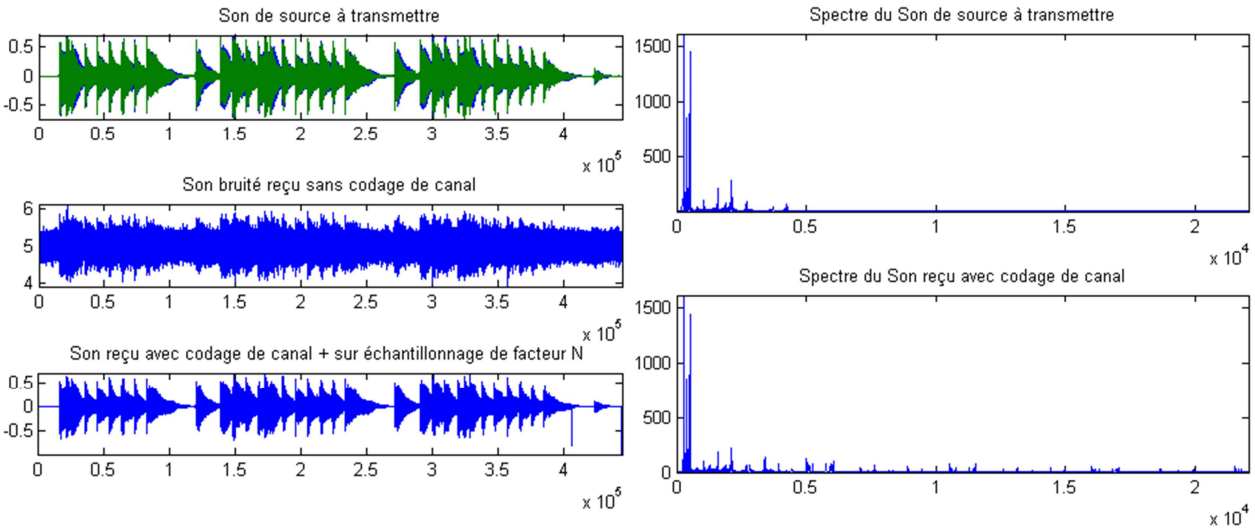
b.1. Code de Hamming systématique (7,4,3)



(a)



(b)



(c)

Figure 5.11 : Résultats de codage de canal relatifsau test 2 en utilisant le code de Hamming systématique (7,4,3) : (a) RSB=4 dB, (b) RSB=7 dB et (c) RSB=17 dB.

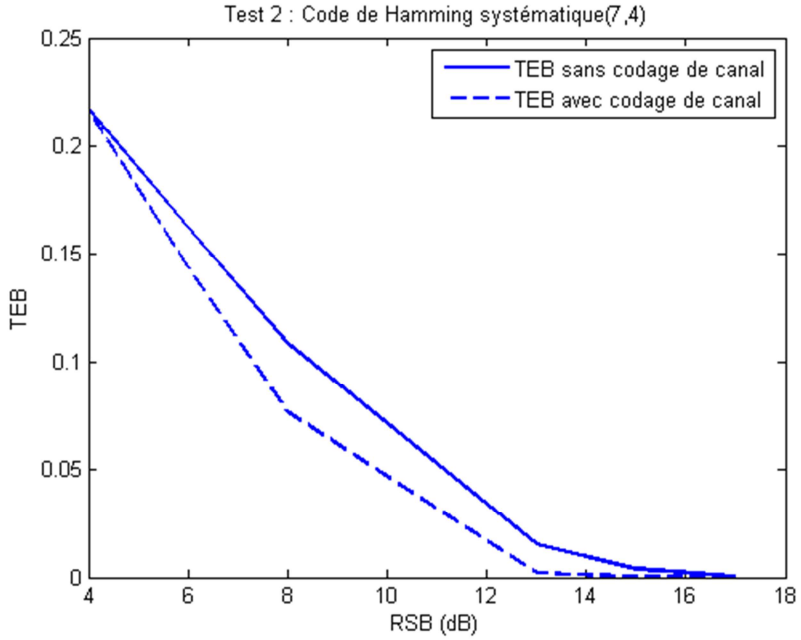
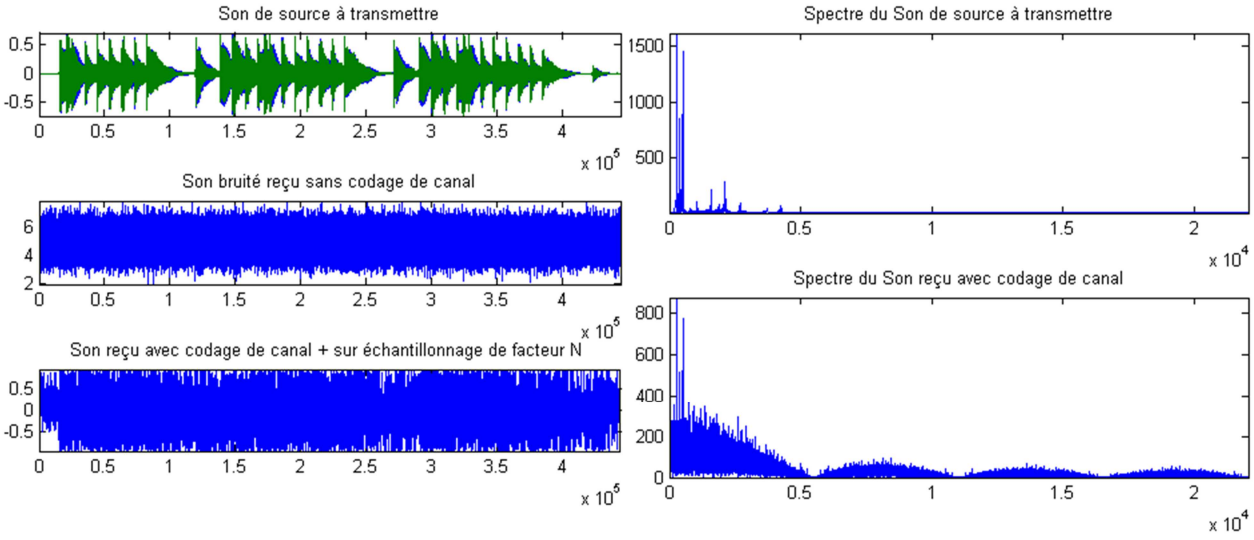
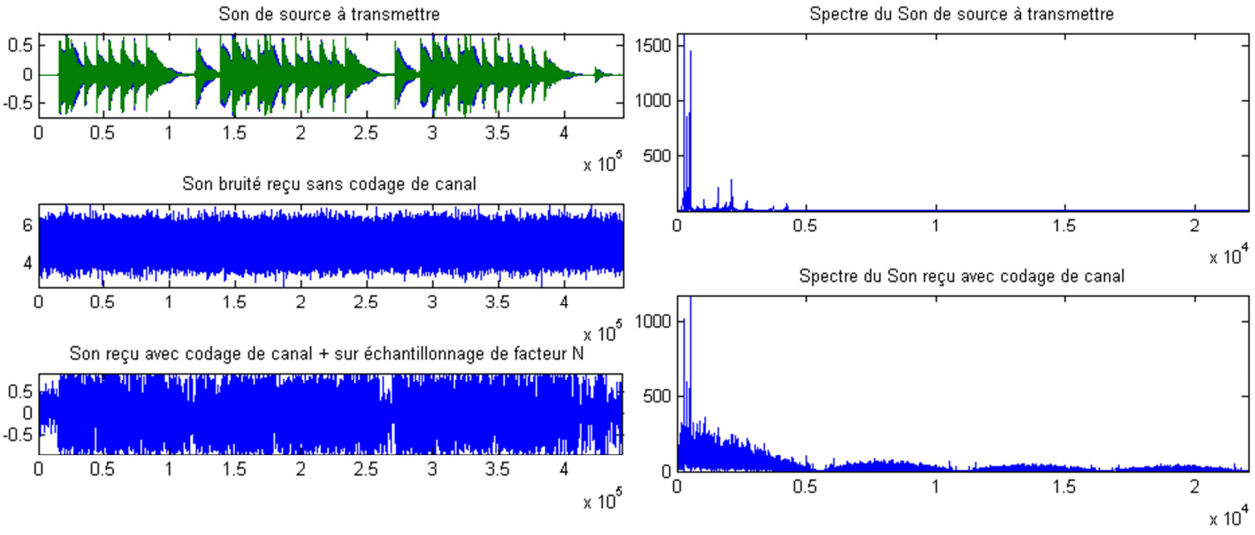


Figure 5.12: Tracés du TEB relatifs au test 2 en utilisant le code de Hamming systématique (7,4).

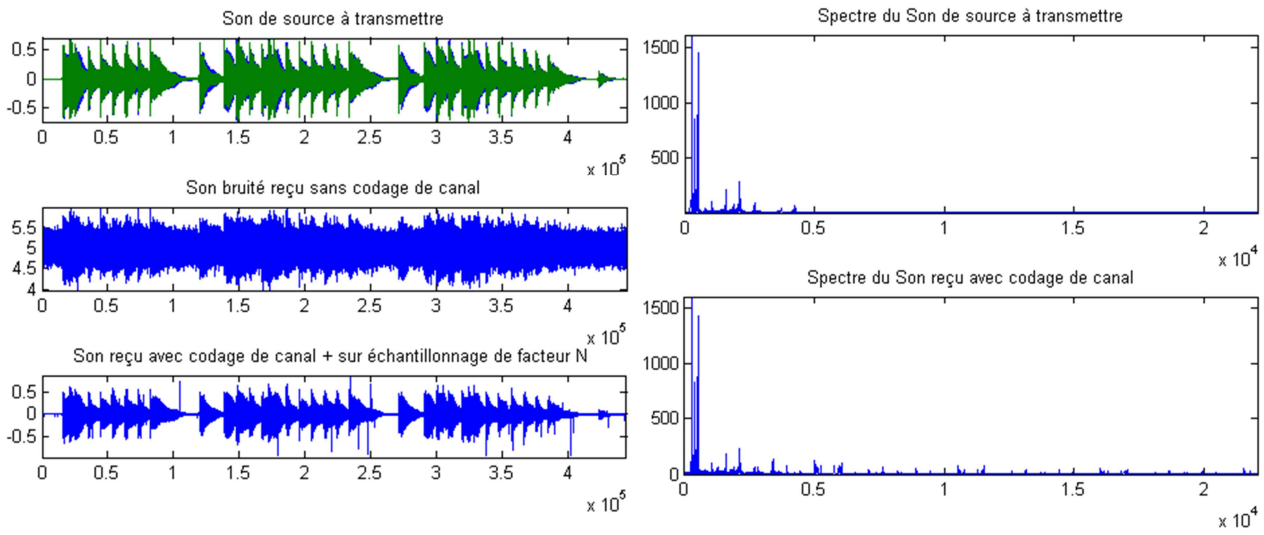
b.2. Code de Hamming non-systématique (7,4,3)



(a)



(b)



(c)

Figure 5.13: Résultats de codage de canal relatifsau test 2 en utilisant le code de Hamming non-systématique (7,4,3) : (a) RSB=4 dB, (b) RSB=7 dB et (c) RSB=17 dB.

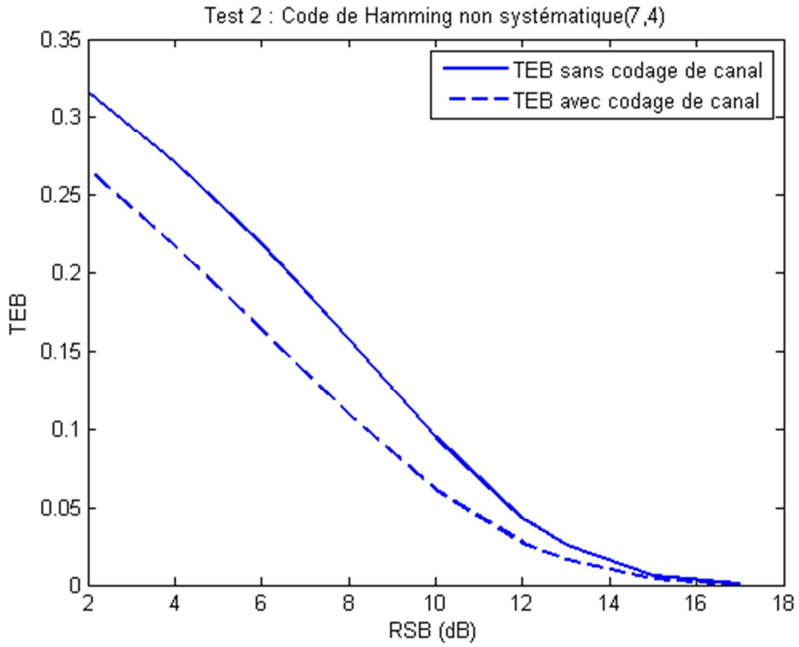
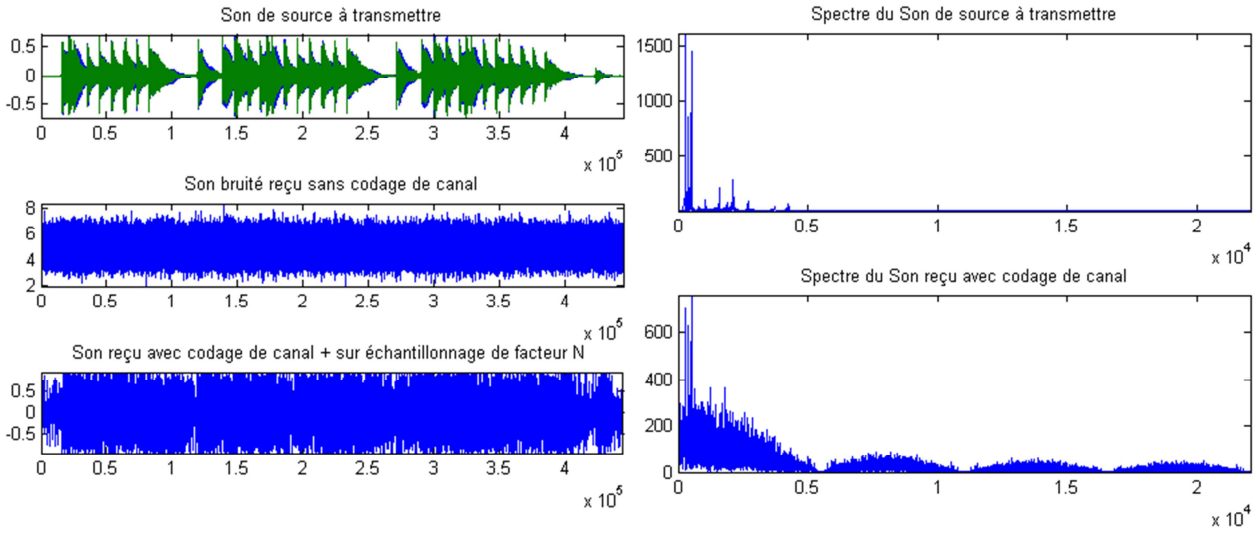
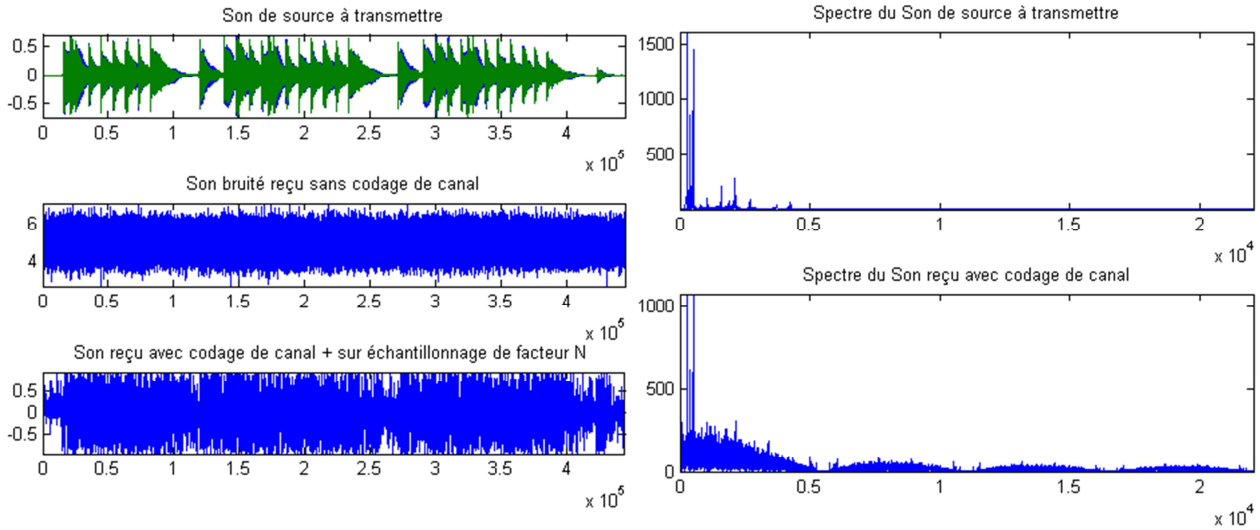


Figure 5.14: Tracés du TEB relatifs au test 2 en utilisant le code de Hamming non-systématique (7,4).

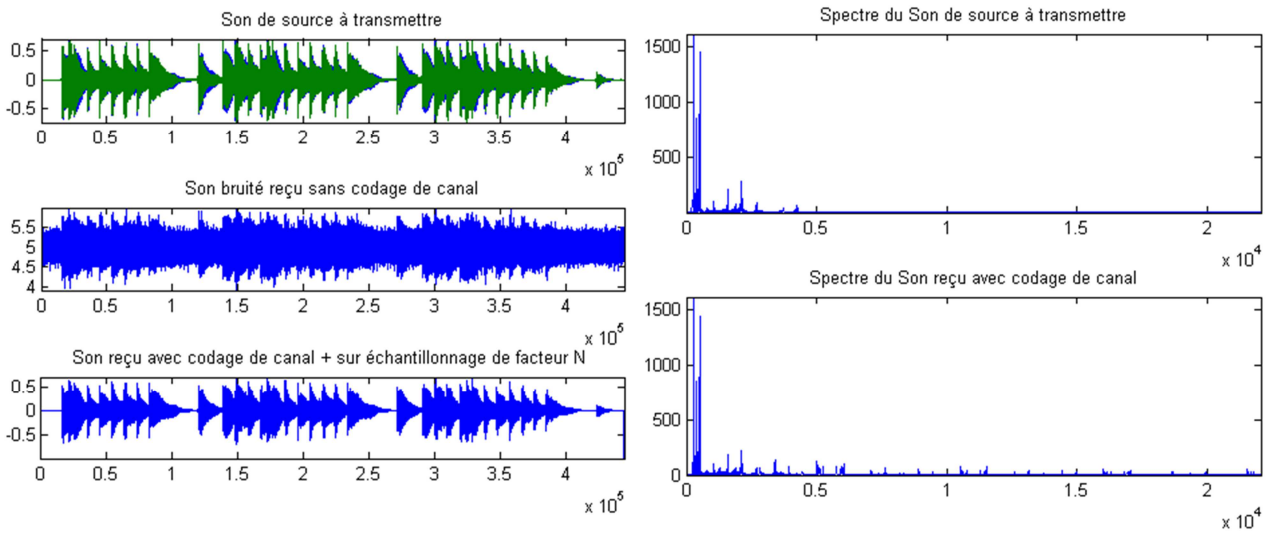
b.3. Code de Hamming systématique (15,11,3)



(a)



(b)



(c)

Figure 5.15 : Résultats de codage de canal relatifs au test 2 en utilisant le code de Hamming systématique (15,11,3) : (a) RSB=4 dB, (b) RSB=7 dB et (c) RSB=17 dB.

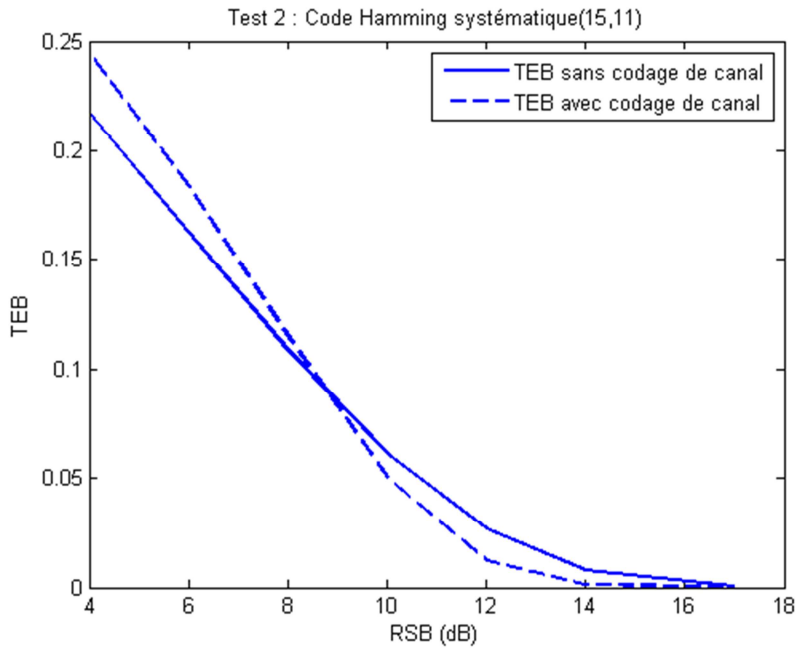
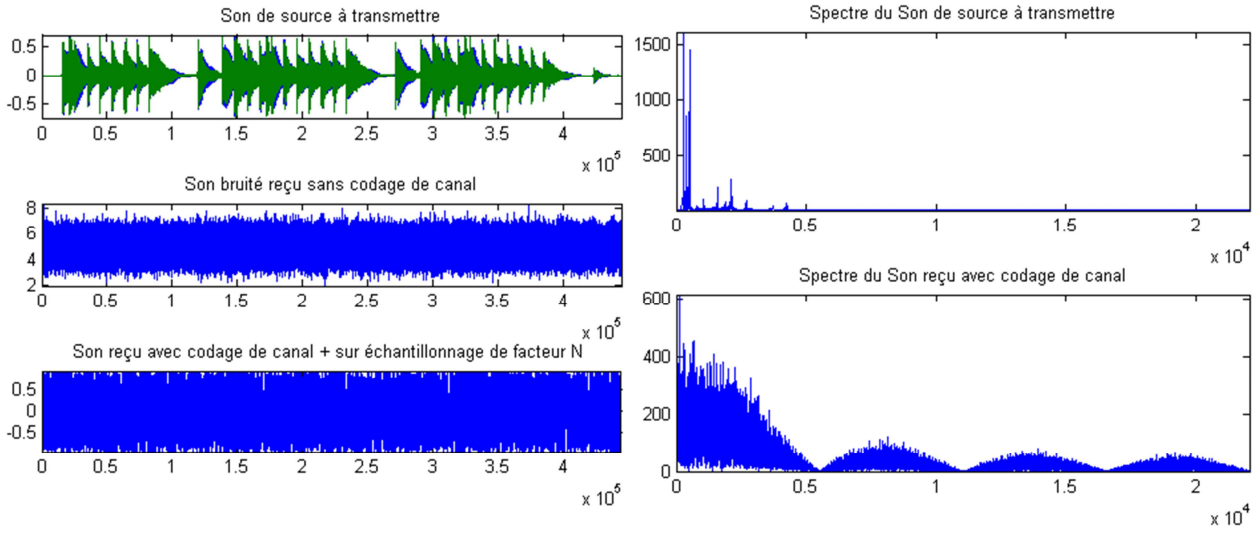
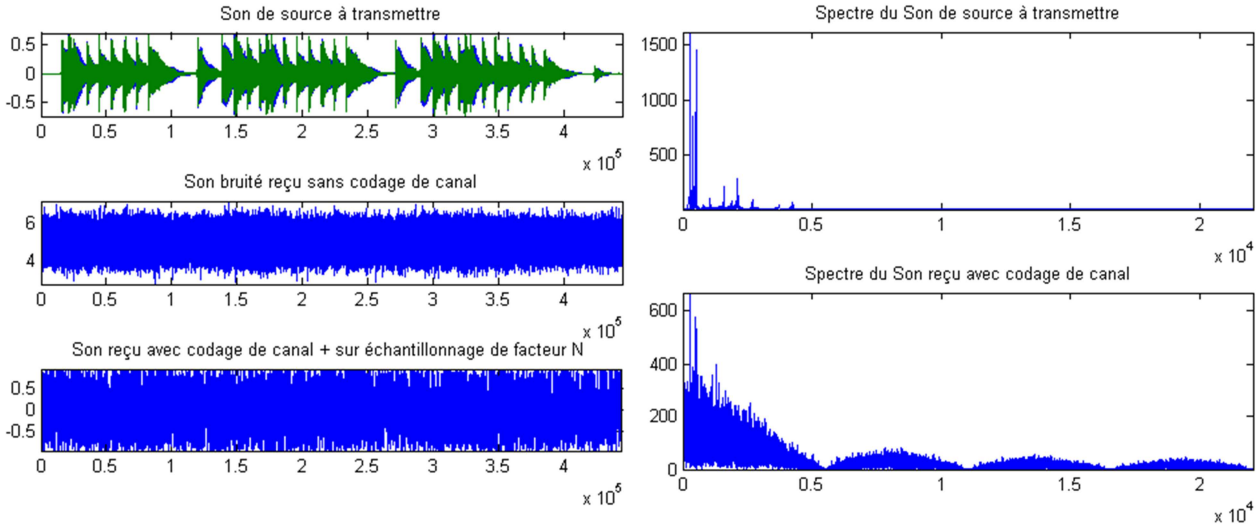


Figure 5.16 : Tracés du TEB relatifs au test 2 en utilisant le code de Hamming systématique (15,11).

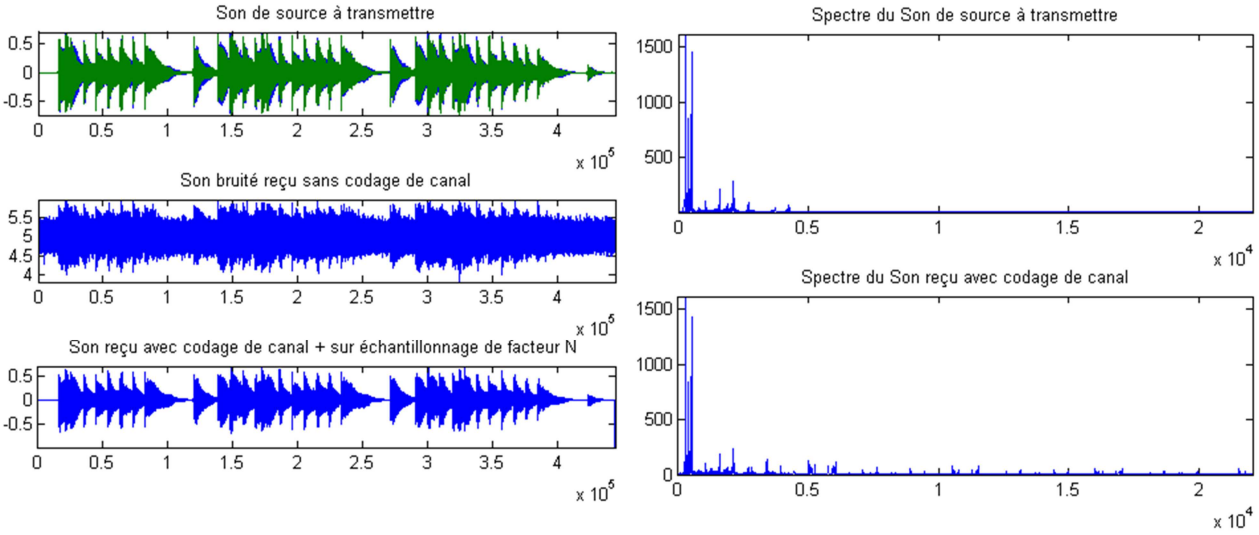
b.4. Code convolutif r=1/2, G=(171 133)$_8$, L=7, profondeur=8



(a)



(b)



(c)

Figure 5.17 : Résultats de codage de canal relatifs au test 2 en utilisant le code convolutif *r=1/2, G*=(171 133), *L*=7, *profondeur*=8 : (a) RSB=4 dB, (b) RSB=7 dB et (c) RSB=17 dB.

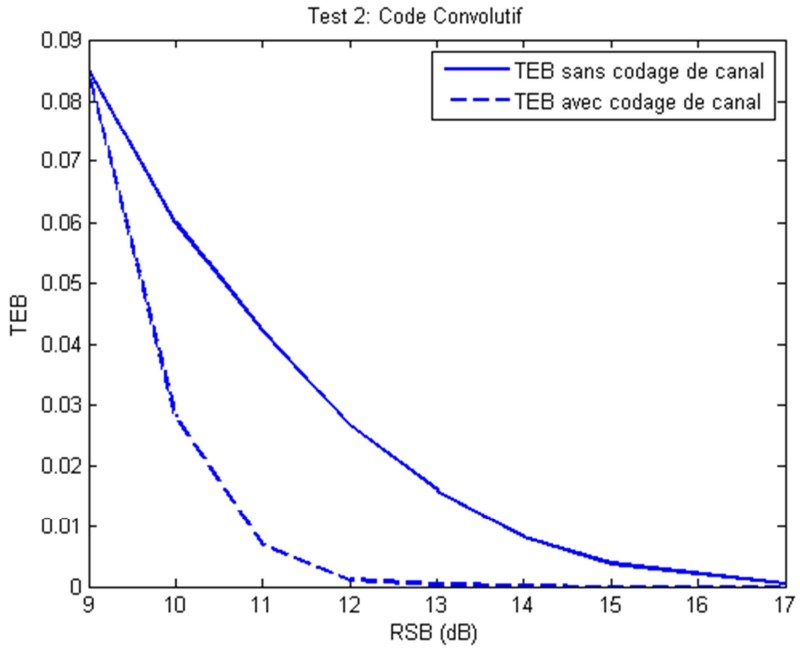
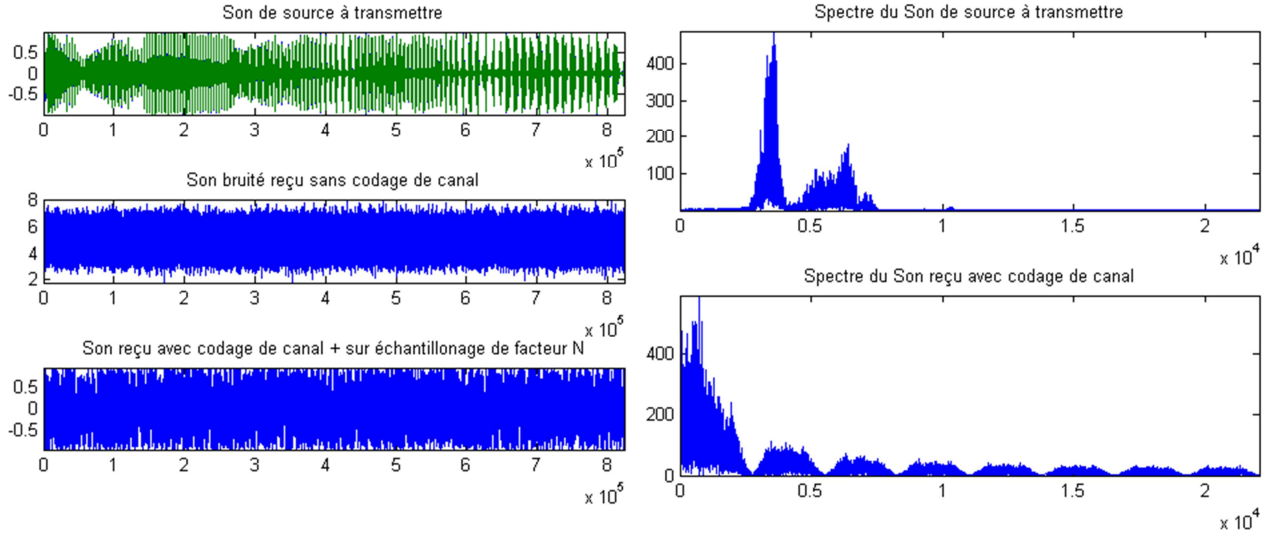


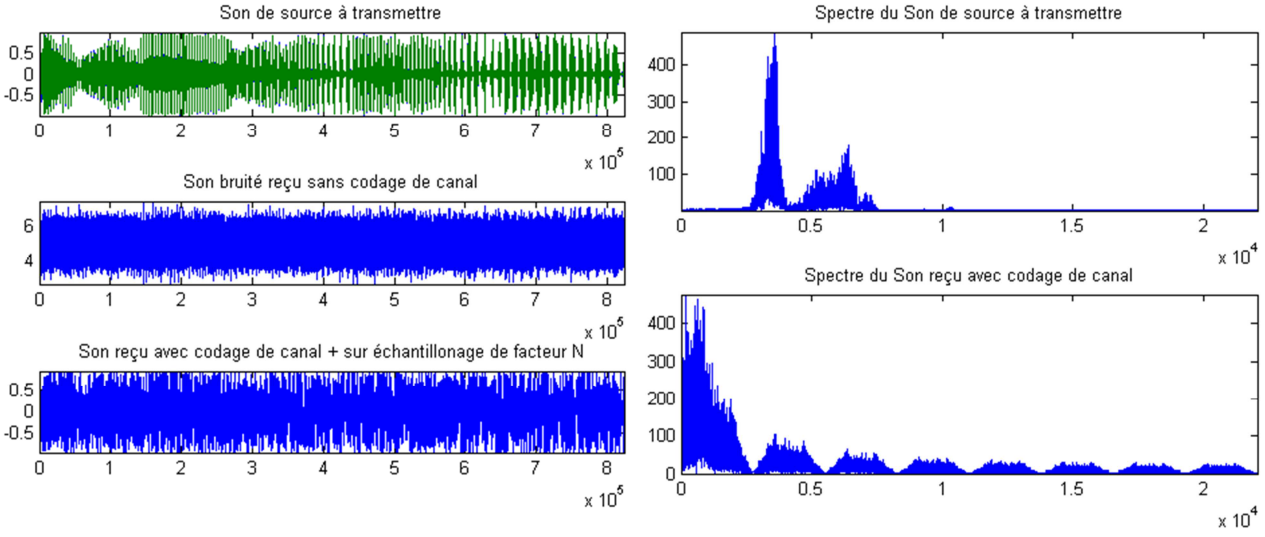
Figure 5.18: Tracés du TEB relatifs au test 2 en utilisant le code le code convolutif *r=1/2, G=(171 133), L=7, profondeur=8.*

Test 3 : Effets de codage de canal sur la transmission d'un son mp3 ('TEST 3')

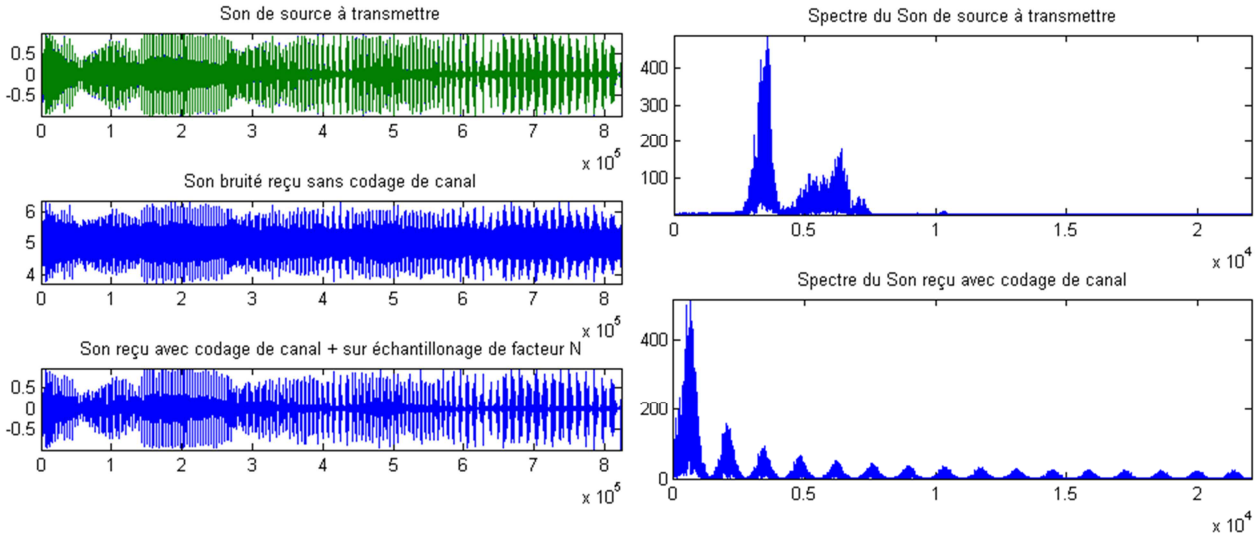
c.1. Code de Hamming systématique (7,4,3)



(a)



(b)



(c)

Figure 5.19 : Résultats de codage de canal relatifs au test 3 en utilisant le code de Hamming systématique (7,4,3) : (a) RSB=4 dB, (b) RSB=7 dB et (c) RSB=17 dB.

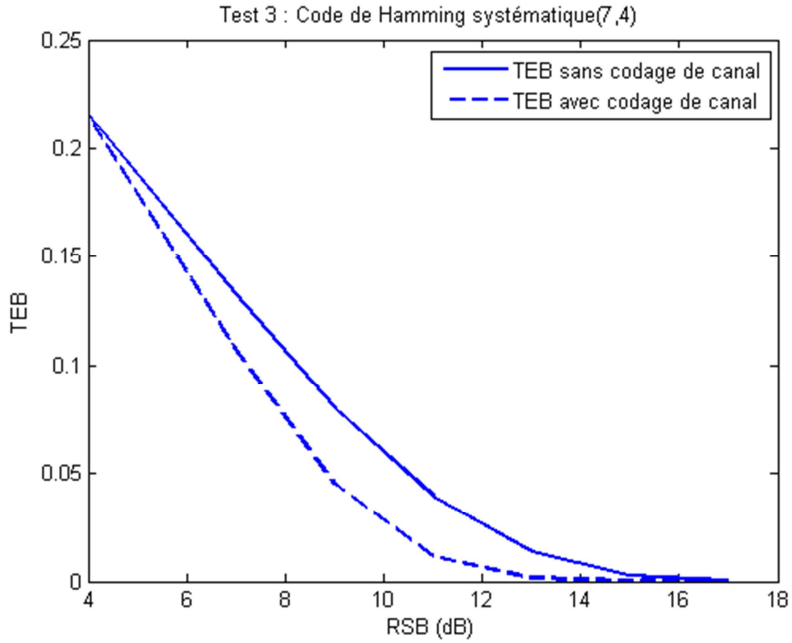
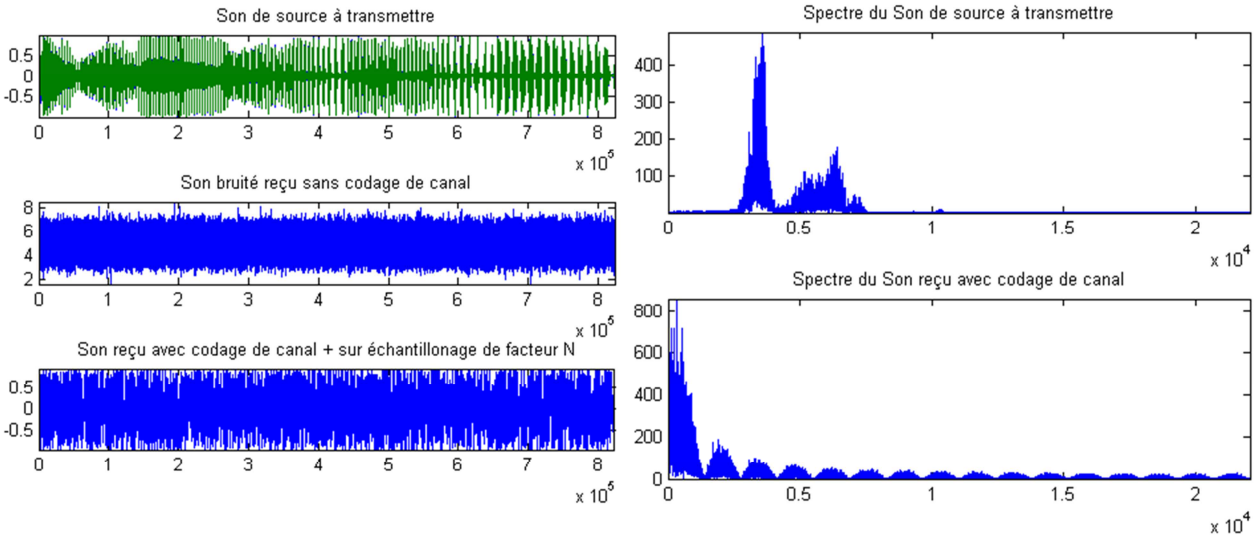
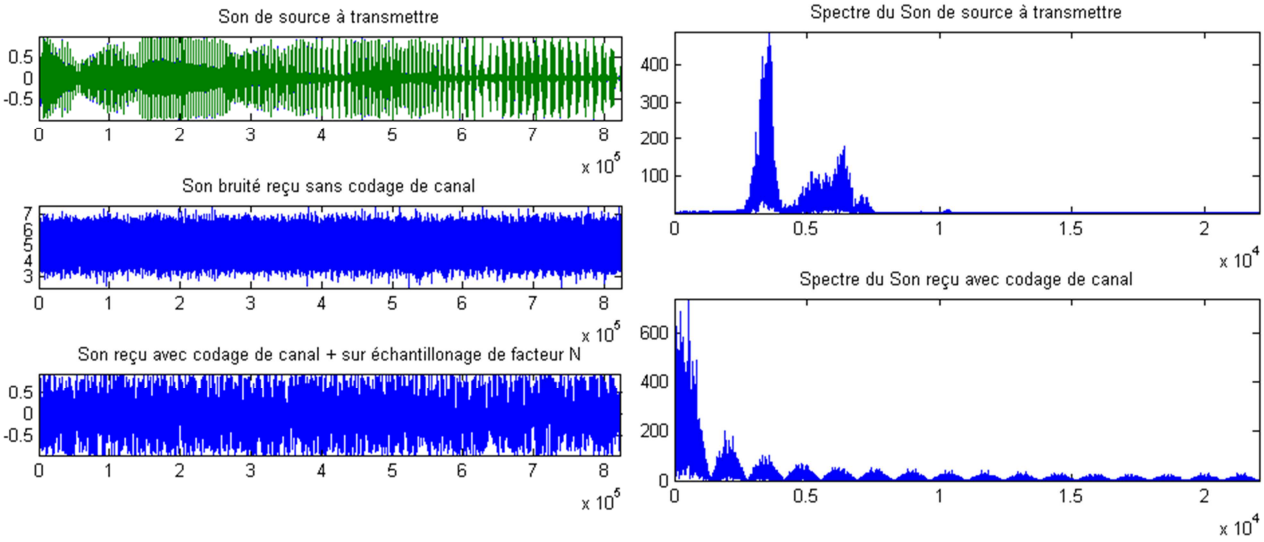


Figure 5.20: Tracés du TEB relatifs au test 3 en utilisant le code de Hamming systématique (7,4).

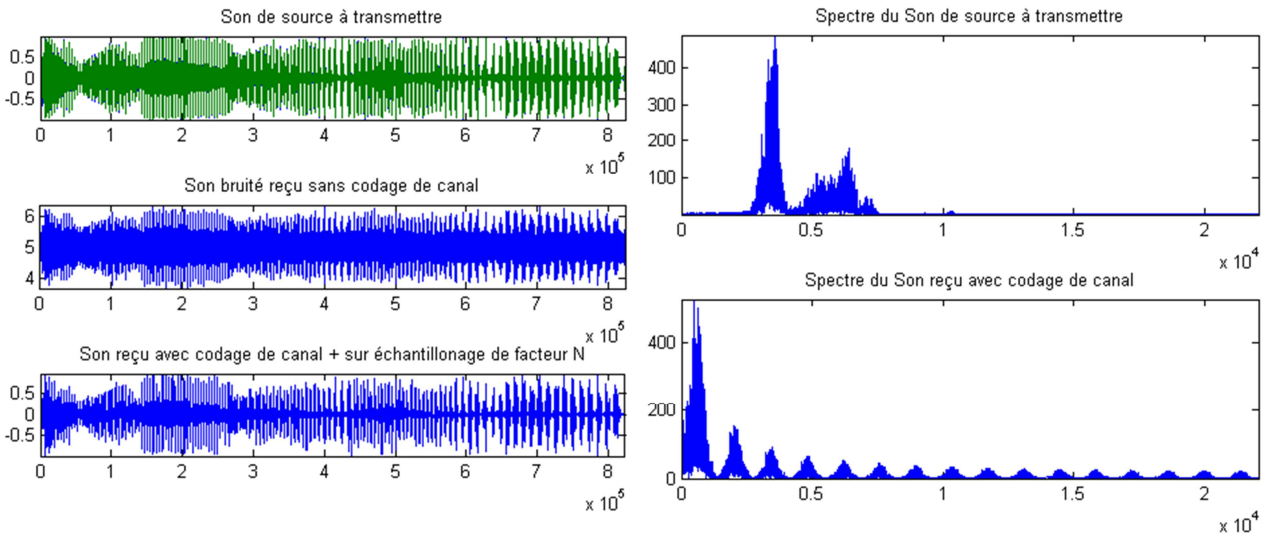
c.2. Code de Hamming non-systématique (7,4,3)



(a)



(b)



(c)

Figure 5.21: Résultats de codage de canal relatifs au test 3 en utilisant le code de Hamming non-systématique (7,4,3) : (a) RSB=4 dB, (b) RSB=7 dB et (c) RSB=17 dB.

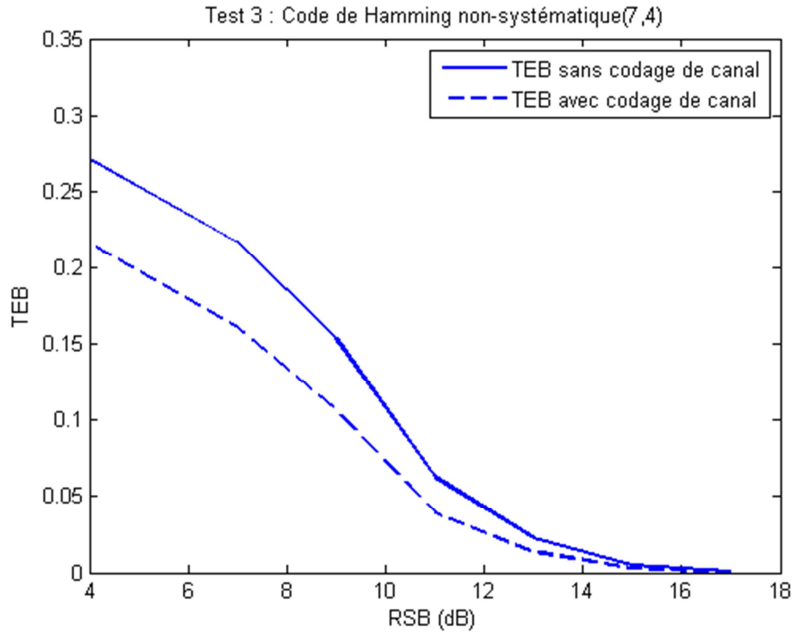
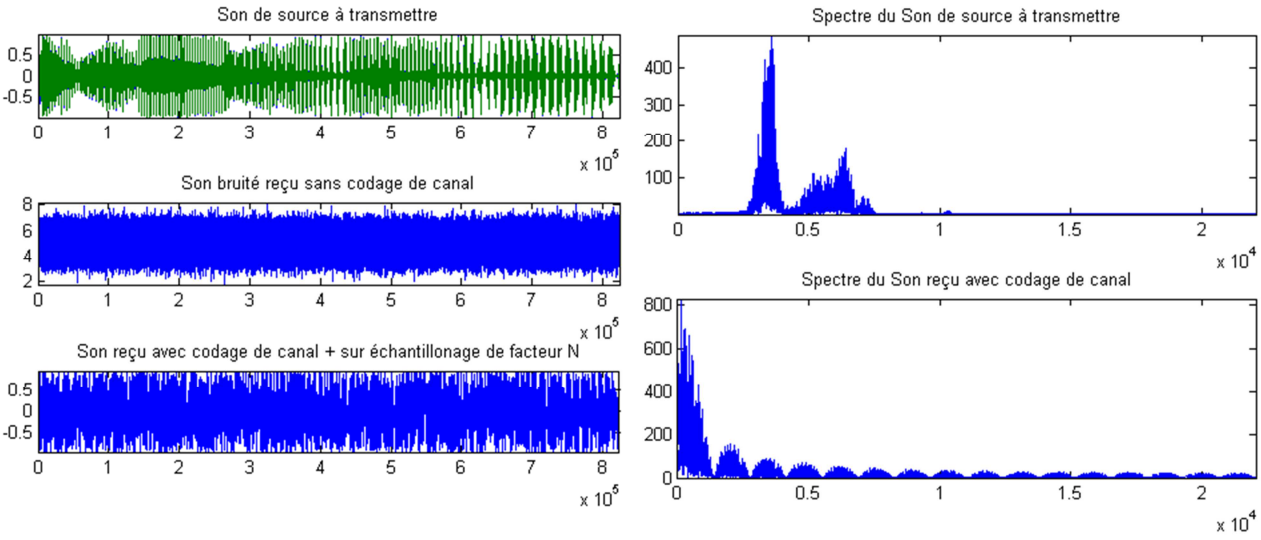
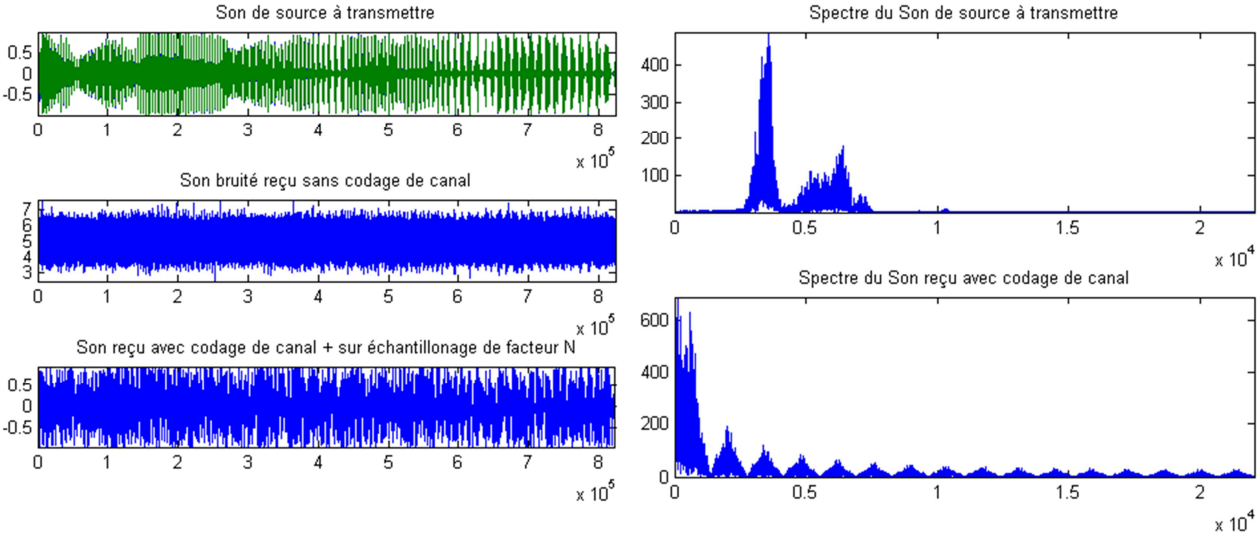


Figure 5.22: Tracés du TEB relatifs au test 3 en utilisant le code de Hamming non-systématique (7,4).

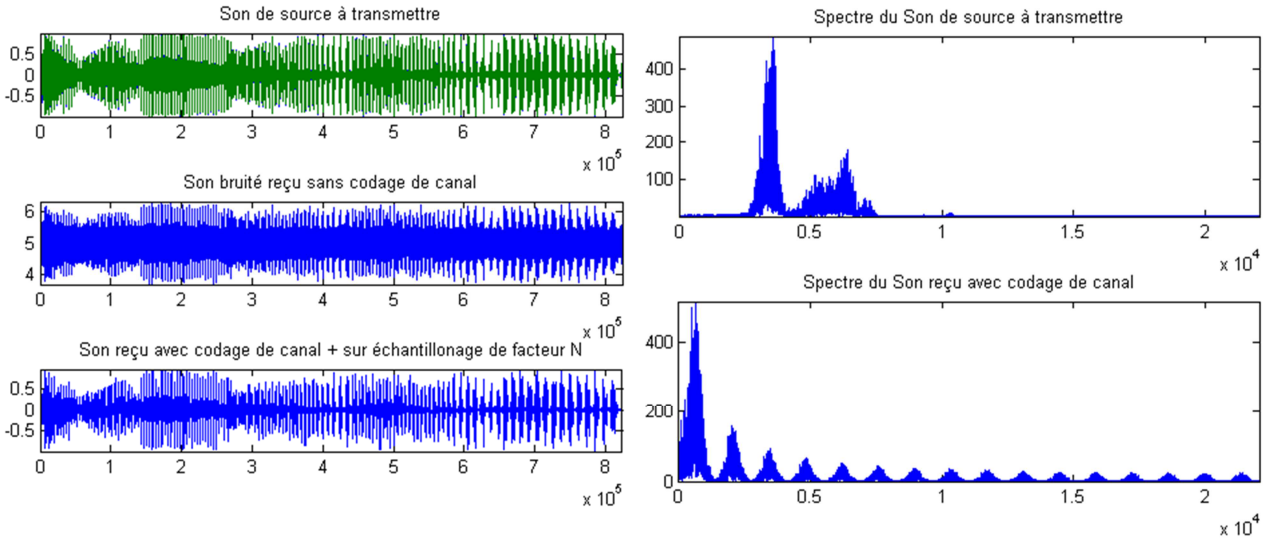
c.3. Code de Hamming systématique (15,11,3)



(a)



(b)



(c)

Figure 5.23 : Résultats de codage de canal relatifs au test 3 en utilisant le code de Hamming systématique (15,11,3) : (a) RSB=4 dB, (b) RSB=7 dB et (c) RSB=17 dB.

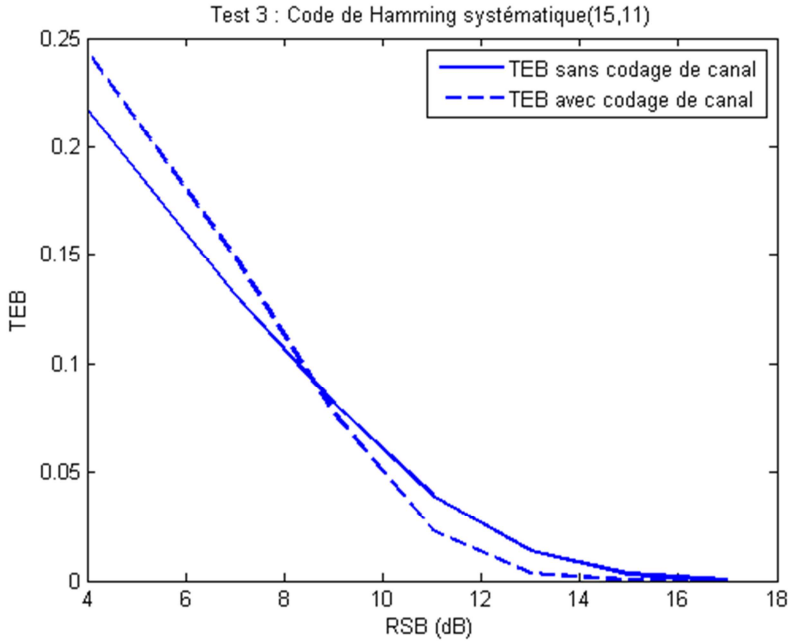
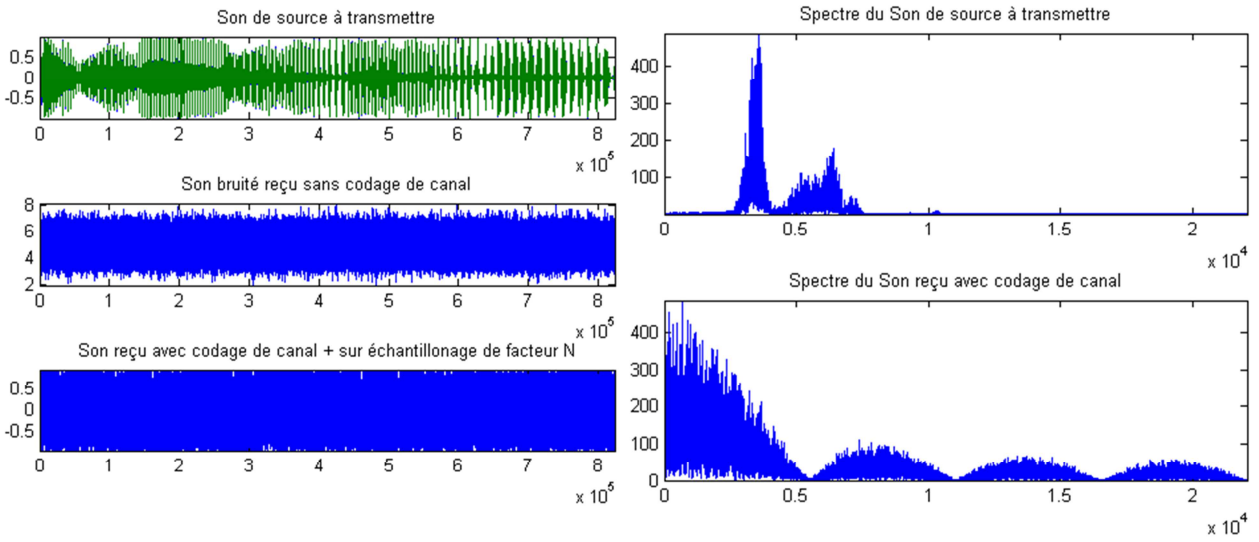
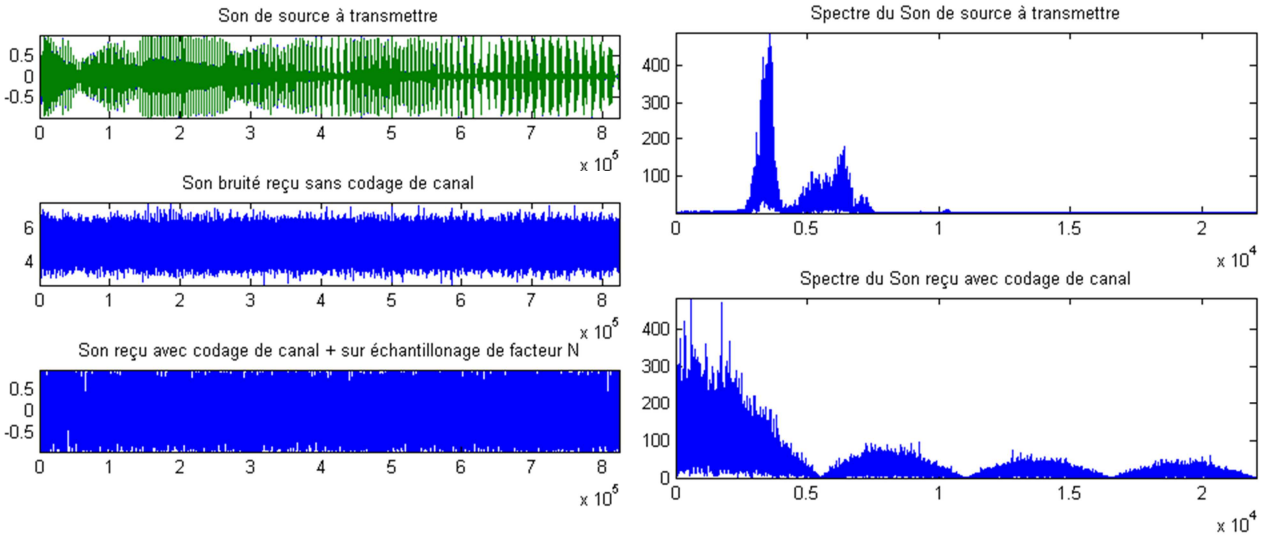


Figure 5.24 : Tracés du TEB relatifs au test 3 en utilisant le code de Hamming systématique (15,11).

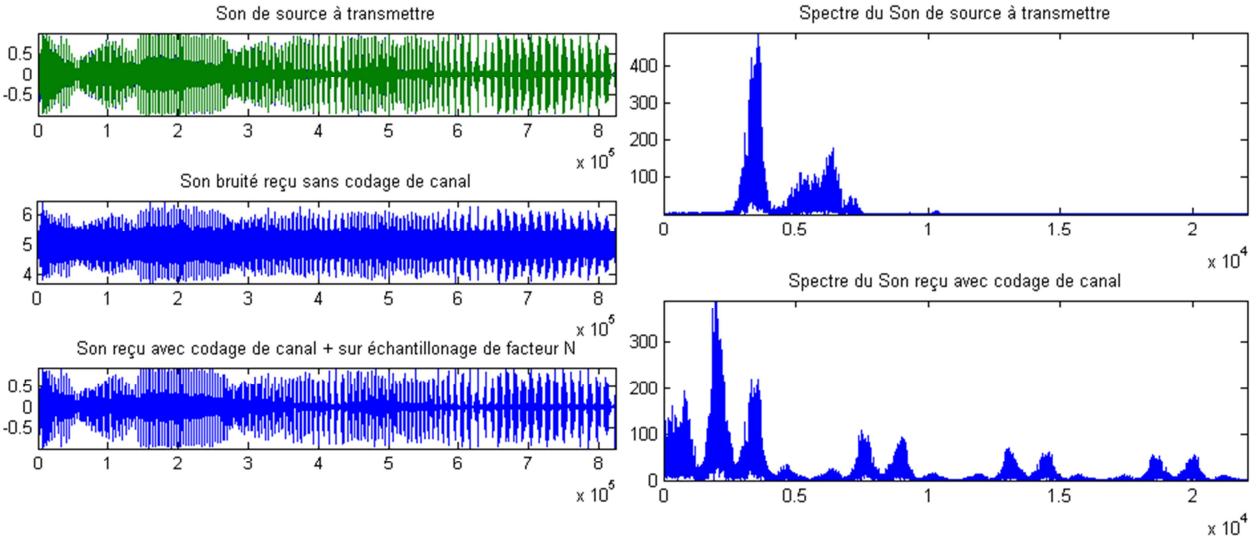
c.4. Codeconvolutif *r*=1/2, *G*=(171 133), *L*=7, profondeur=8



(a)



(b)



(c)

Figure 5.25 : Résultats de codage de canal relatifs au test 3 en utilisant le code convolutif *r*=1/2, *G*=(171 133), *L*=7, profondeur=8: (a) RSB=4 dB, (b) RSB=7 dB et (c) RSB=17 dB.

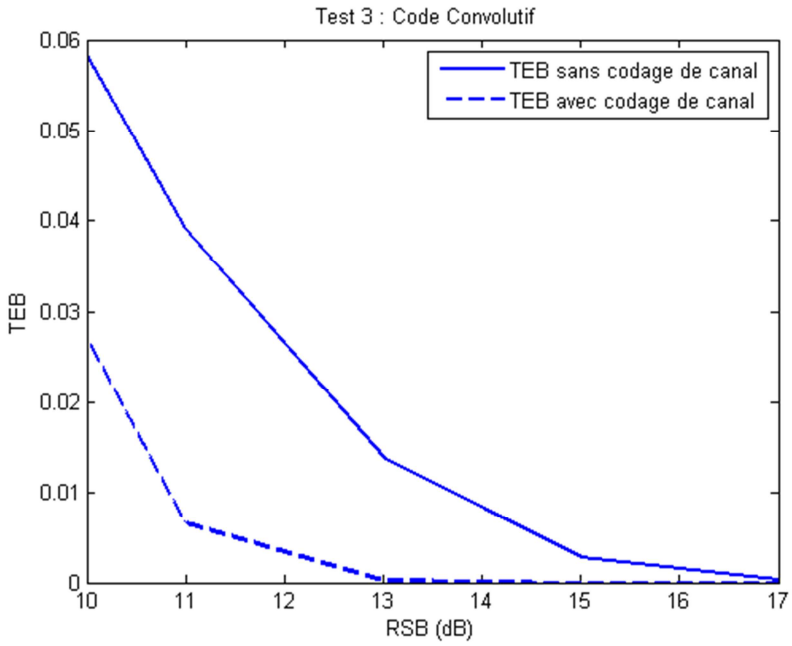
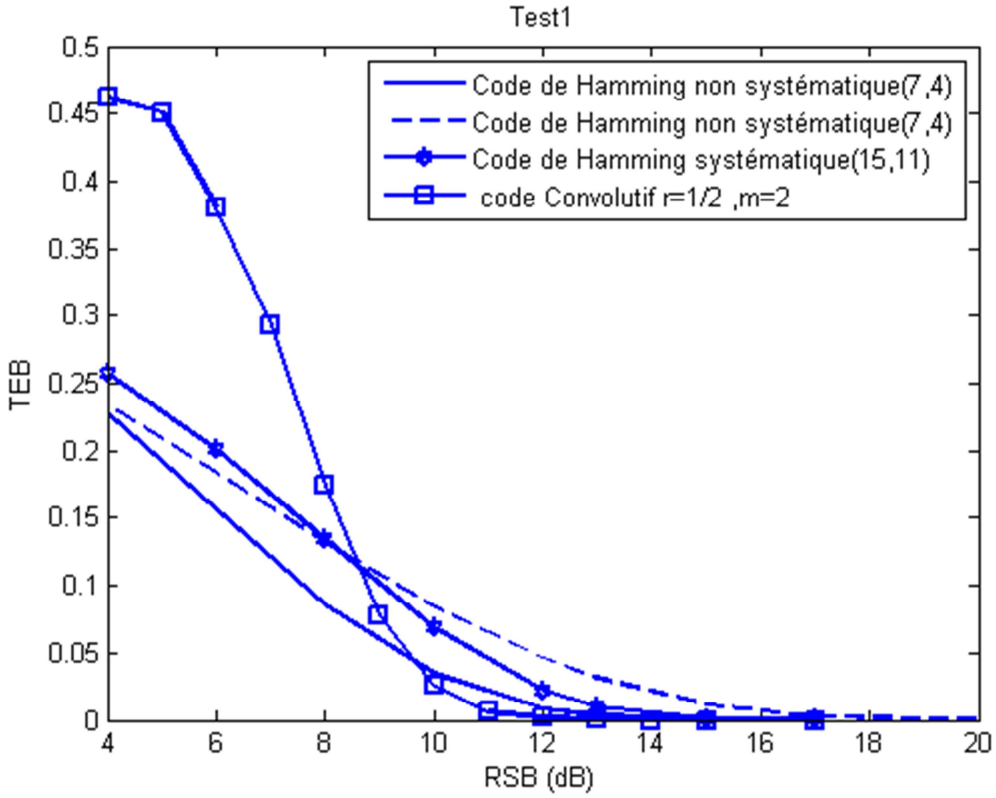
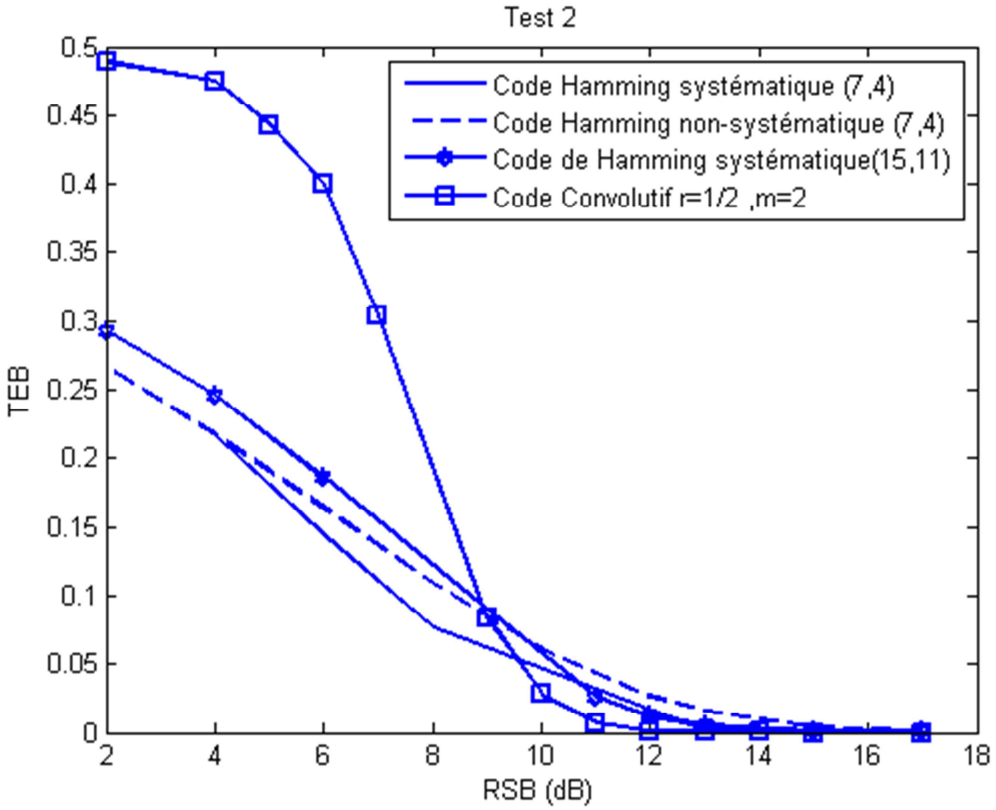


Figure 5.26: Tracés du TEB relatifs au test 3 en utilisant le code le code convolutif *r*=1/2, *G*=(171 133*), L*=7, *profondeur=8.*

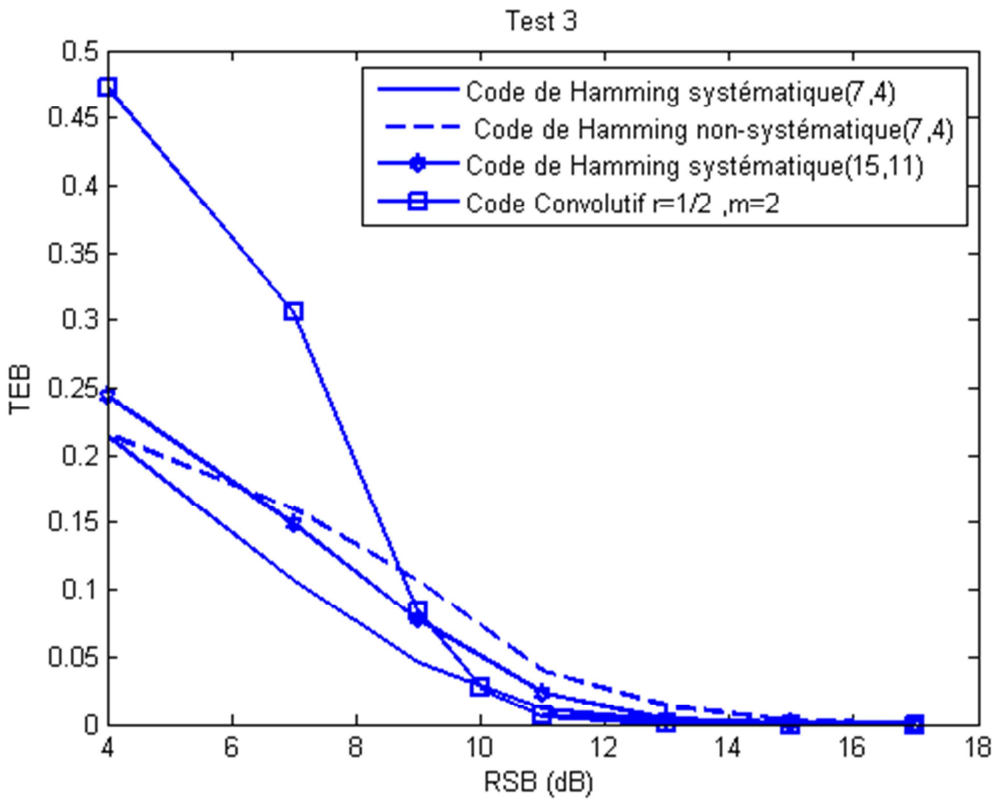
Figure 5.27 présente une comparaison entre les différents TEBs réalisés par les quatre codes considérés dans cette étude relatifs aux trois tests de transmission des signaux audio.



(a)



(b)



(c)

Figure 5.27 : Comparaison entre les différents TEBs réalisés par les quatre codes étudiés : (a) test 1, (b) test 2 et (c) test 3.

V.5 Interprétations, Synthèse et Conclusions

D'après l'examen visuel des résultats, l'amélioration de la qualité des signaux audio reçus en introduisant le codage de canal est nettement évident même si le canal de propagation est fortement bruité. Cependant, une analyse objective et beaucoup plus précise des résultats s'obtient à partir des différents tracés des TEBs comme détaillé ci-après.

En premier lieu, Il est important de signaler que les tracés des TEBs correspondent à la valeur moyenne des TEBs réalisés en menant une série d'essais pour chaque valeur de RSB (simulations Monte-Carlo). C'est très naturel dans des situations pareilles car il s'agit d'un bruit aléatoire inconnu perturbant le canal de propagation à chaque instant. Si un calculateur puissant était disponible, avec une RAM assez grande, nous devrions mener des centaines d'essai pour chaque valeur de variance de bruit.

De point de vue général, les tracés des TEBs montrent clairement la réduction considérable de la probabilité d'erreur en introduisant le codage de canal par rapport à un schéma de transmission de données de la source sans codage de canal.

Pour le test 1, un son de format wav est transmis. Le code de Hamming systématique (7,4) performe le mieux sur toute la plage de RSB suivi par le code de Hamming systématique (15,11) jusqu'à RSB≈10 dB environ ou le code convolutif commence à mieux réduire le TEB. Le code de Hamming non-systématique (7,4) est le moins performant pour RSB>10 dB environ. Au-dessous de 10 dB, le code convolutif produit le TEB le plus élevé ce qui correspond à la moindre fiabilité.

Pour le test 2, un son de format mp3 est transmis. Le code de Hamming non-systématique (7,4) est le moins performant pour RSB>9 dB. Dans cette même plage, le code convolutif surpasse les autres codes suivi par le code de Hamming systématique (15,11) et puis par le code de Hamming systématique (7,4) qui est le plus performant pour des faibles valeurs de RSB <9 dB.

Pour le test 3, un autre signal audio de format mp3 est transmis. Le code de Hamming systématique (7,4) est nettement mieux performant que les autres codes jusqu'à RSB<10 dB environ. Au-dessus de cette valeur, le code convolutif est le plus fiable. Les codes les moins performants sont le code de Hamming non-systématique (7,4) suivi de code de Hamming systématique (15,11).

Ces résultats sont attendus car nous savons que plus le code utilisé est complexe meilleur est le codage effectué en terme de TEB réalisé. Le code linéaire en bloc est de plus en plus fiable que le nombre de bits de redondance croit. Ceci explique pourquoi le code de Hamming non-systématique (7,4) n'améliore pas forcément la qualité du son reçue par rapport au code de Hamming (7,4) (bien qu'il soit plus rapide) puisque le nombre de bits de contrôle est le même. D'autre part, le code convolutif est d'autant plus performant que le nombre de mémoires augmente ce qui justifie la performance modérée du code convolutif choisi pour des petites valeurs de RSB par rapport aux codes LBC puisqu'il ne possède que 2 mémoires. Nous voulions le comparer à d'autres codes convolutifs de mémoire plus grande mais le problème principal se pose dans le temps de traitement très considérable requis par les codes convolutifs, et ceci pour effectuer le codage et le décodage dû essentiellement à la complexité de l'algorithme de Viterbi.

Une autre contrainte concerne la taille de l'information de la source elle-même qui est très importante du fait que l'enregistrement audio occupe une taille très grande. Prenons par exemple le signal du test 1. Il s'agit d'un enregistrement audio type PCM de durée 2 secondes. Le vecteur audio correspondant est de taille L=75712 en mono. Si on considère un codage sur 8 bits, la taille du vecteur binaire devient 8×L= 605696. Prenons par exemple le code de Hamming (7,4). Les vecteurs de source binaires sont découpés en paquets de 4 bits. Le nombre de paquets à coder est donc égal à P=8×L /4=2×L. Chaque paquet est élargi de 3 bits de redondance. La dimension totale de mots de code à transmettre après codage de canal est égal à P×4+P×3=P×7 soit un nombre total de bits égal à ! Cette dimension est doublé pour un son enregistré en stério soit environ ! Ceci nous a poussés à inclure un sous-échantillonneur de facteur N afin de réduire la taille des données à coder et par conséquent diminuer le temps de transmission. Les sons reçus sont de qualité acceptacle et se rapprochent des signaux audios originaux après effectuer une interpolation par sur-échantillonnage de facteur N au niveau du récepteur. Cette idée était inspirée des formats de sous-échantillonnage de la chrominance utilisé dans la compression vidéo.

Conclusion générale

Conclusions et Recommandations pour les Projets Futurs

Dans ce travail, nous avons confirmé un résultat fondamental de la théorie de l'information qui annonce qu'un codage de canal judicieux permet de réduire la probabilité d'erreur et de la rendre aussi petite que l'on veut. Ceci est effectué par un ajout de bits de contrôle, générés à partir des données de source selon un algorithme spécifique, qui permettent au récepteur de vérifier si des erreurs étaient introduites dans le mot reçu et de les corriger par la suite au limite d'un pouvoir déterminé de correction.

Nous avons choisi l'application de codes correcteurs sur la transmission des signaux audio afin de pourvoir effectuer des comparaisons visuelles pertinentes entre les schémas sans et avec codage de canal. D'autre part, cette application d'identifier les problèmes relatifs aux communications multimédia à savoir la lourdeur des paquets binaires à transmettre, le temps de traitement considérable ainsi que la sensibilité au bruit de canal surtout lorsqu'il est fortement perturbé.

L'augmentation de la complexité du codage est soit effectuée par l'ajout de nombre plus important de bits de contrôle pour les codes LBCs, soit par l'augmentation du nombre de mémoires pour les codes convolutifs. Cependant, deux inconvénients apparaissent : une diminution de débit de transmission et la complexité de processus de décodage. Dans ce sens, le code de Hamming est un code parfait lorsqu'il s'agit d'une erreur unique puisqu'il peut la corriger quelle que soit sa position. Cependant, pour permettre la correction des erreurs multiples, d'autres familles de codes sont disponibles à savoir les codes BCH qui performent beaucoup mieux que les codes de Hamming et donc ils sont préférables à utiliser dans les canaux fortement bruités. Les codes convolutifs sont de leurs parts les plus fiables à condition d'augmenter le nombre de mémoires (registres à décalage). Leur décodage est aussi très simple à implémenter. Ceci explique leur utilisation dans tous les systèmes numériques commercialisés à l'heure actuelle (communications satellitaires, réseaux GSM, norme UMTS...), en général combinés avec d'autres codes linéaires en bloc.

De règle générale, une augmentation dans la complexité du processus de codage de canal effectué permettra d'améliorer considérablement la qualité des signaux numériques reconstitués après décodage de canal. Cependant, l'implémentation matérielle des codeurs/décodeurs

correspondants nécessitent des processeurs de très grande vitesse de traitement et à grande mémoire surtout avec les débits actuels des systèmes numériques commercialisées qui ne cessent de croitre de jour en jour. L'enjeu primordial est donc d'optimiser le compromis entre un débit maximal, une fiabilité de transmission la plus élevée et la moindre complexité du circuit du récepteur.

Les recommandations pour les projets futurs sont assez nombreuses, parmi lesquelles nous citons :

- ✓ Considération d'une chaine de transmission plus proche de pratique incluant les filtres à l'émission et la réception tels que les filtres adaptés permettant de maximiser le rapport signal sur bruit aux instants d'échantillonnage et les filtres de Nyquist pour l'élimination des interférences entre les symboles (ISI pour intersymbol interference),
- ✓ Considération de différents types de modulation numérique au lieu du système de bande de base étudié dans ce mémoire,
- ✓ Ajout d'une étape d'entrelacement (interleaving), très utilisée en pratique. Ceci sert à disperser les mots de code selon un ordre connu par l'émetteur et le récepteur afin de distancer les erreurs, éviter les salves et donc améliorer le pouvoir de correction des erreurs. A titre d'exemple, dans la norme UMTS, le TEB décroit de 10^{-6} en utilisant deux codeurs convolutifs à 10^{-9} seulement avec l'utilisation d'un codeur Reed-Solomon entrelacé suivi par un codeur convolutif. Cependant, le prix à payer est la croissance en complexité du circuit récepteur,
- ✓ Considération d'autres types de codes détecteurs et correcteurs d'erreurs qui peuvent être plus performants à savoir les codes Reed-Solomon, les codes Reed-Muller, les codes turbo, etc.,
- ✓ Réalisation matérielle d'un système de communications numériques sans fils comme montré à la Figure 1.7 à base de circuits programmables pour transmission de données multiples (texte, image, son ...),
- ✓ Recherche d'autres algorithmes plus fiables de codage/décodage de canal,
- ✓ Test de nouvelles architectures à savoir la combinaison de plusieurs types de codes ou l'utilisation de codeurs en cascade. En UMTS toujours, l'insertion de deux codeurs convolutifs en cascade permet de réduire le TEB moyen de 10^{-3}, réalisé par un codeur convolutif unique, à 10^{-6},
- ✓ Implémentation matérielle sur carte FPGA ou DSP de codeurs/décodeurs de canal et comparaison de leurs performances sur des signaux audio réels.

Références bibliographiques

Références :

[1] Jean-Michel Mermet, Techniques de numérisation, cours de licence BDAN–IUT2 Grenoble, 2009-2010. Disponible sur : cyan1.grenet.fr/podcastmedia/cours-de-numerisation/2888.pdf.

[2] www.ac-grenoble.fr/college/jean-monnet.st-jorioz/techno/clg/Documents/ definition-analog ique-numerique.htm

[3] culturesciencesphysique.ens-lyon.fr/ressource/principe-numerisation.xml

[4] André Aoun, Techniques de transmission, Université Paul Sabatier (Toulouse III), 2002. Disponible sur : www.httr.ups-tlse.fr/pedagogie/cours/trans/techniques.htm.

[5] Iryna Andriyanova, Introduction aux communications numériques, cours de Master M1 ISIM, Université de Cergy-Pontoise, 31 janvier 2013. Disponible sur : http://perso etis.ensea.fr /andriyanova/documents/cours/polie-M1-UCP-v1.2.pdf.

[6] Simon Haykin, *Communication Systems*, 4ème édition, John Wiley & sons, 2001.

[7] http://www.commentcamarche.net/contents/1209-codage-de-huffman

[8] J. Badrikian, Mathématique pour téléinformatique, codes correcteurs, principes et exemples,

[9] Le son et l'audition-Hearing aids from Widex disponible sur: webfiles.widex.com /WebFiles/P%2000M%200710%20103.pdf

[10] Techniques multimédia pour le son disponible sur : httpdeptinfo.unice.frtwikip ubLinfoP lanningDesSoutenances20032004Raverdino-Guatteri.pdf

[11] Y.Grenier, G. Richard, Master IAD, Multimedia et Sons, mars 2005 disponible sur : https: //perso.telecom-paristech.fr/grichard/Enseignements/IAD/CodageMPEGAudioMasterIAD.pdf

[12] Le son en multimédia disponible sur : sites.google.comsiteabdelkrimabdelliteaching.

[13] FONDEMENTS DE MULTIMEDIA disponible sur : https:// hadhrim. files.wordpress .com /2015/12/le-multimc3a9diaf.pdf

[14] Chapitre 3 : Le son en multimédia disponible sur : http://slideplayer.fr/slide/3120452/

[15] chapitre5 modulation par impultions codées disponible sur : httpwcours.gel.ulaval. ca2015 aGEL3006default5noteschap5 _modulation_PCM_GEL3006_2015.pdf.

[16] Y.Chouinard/ M.Hefnawi / Y.Bouslimani/ ELG-4571 Systèmes de télécommunications/ GEF 411A Théorie de communication / GELE 4521 Télécommunications disponible sur : httpwww.site.uottawa.ca~chouinarCH01PVI.pdf.

[17] Codage du son_perso.modulonet. frplacurieRessourcesBTS1...ChapCodage % 20 du % 20 son.pdf

[18] Copyright- Enseignement des Métiers de la communication Malakoff disponible sur : httpw ww. ixenakis.comLa_Compression_Audio_Numerique.PDF.

[19] chapitre 5 La compression des signaux audio et vidéo disponible sur : www. Montefiore .ulg.ac.be/services/acous/STSI/file/ITAV_2010_Chap5.pdf

[20] httpwww.montefiore.ulg.ac.beservicesacousSTSIfileITAV_2010_Chap5.pdf

[21]docs.google.comviewera=v&pidsites&srcid=ZGVmYXVsdGRvbWFpbnxhYmRlbGtyaW1hYmRlbGxpfGd4OjM1YWY4OWE2ZjNlMmM0MzA

[22] Victor K. Wei, Generalized Hamming weights for linear codes, IEEE. Transaction on Information Theory, 37 :1412–1418, September 1991.

[23] [Pierre Csillag Introduction aux codes correcteurs, Ed.Ellipses,.

[24] O. Pothier, Codage de canal et turbo-codes, support de cours, Cours 5 : Les codes convolutifs, 2000.

[25] Introduction aux Communications, support de cours de l'université de Cergy Pontoise, 2012.

[26]http://www.memoireonline.com/06/10/3544/Conception-et-simulation-dun-brouilleur-GSM.html.

[27] Introduction aux Codes Correcteurs Pierre Csillag Ed Ellipses.

Annexe : Code source Matlab

Programme principal de transmission d'un texte brut

```
function codage_canal_texte_brut;
refaire=1;
while(refaire==1)
clc
message=input('Introduire le texte brut à transmettre:  \n');
mean=input('Introduire la valeur moyenne du bruit Gaussien additif:  \n');
RSB_dB=input('Introduire le rapport signal sur bruit (RSB ou SNR en dB):  \n');
RSB=10^(RSB_dB/10);
var=(1/RSB);
[entropie,longueur_moyenne,efficacite,sequence_decodee_brutie,sequence_decodee]=codage_de
codage_de_canal_huffman(message,mean,var);
% AFFICHAGE DES RESULTATS
sequence_decodee_brutie
sequence_decodee
refaire=input('Voulez vous continuer?  (taper 1 pour continuer)\n');
end;
```

Codeur/décodeur avec l'algorithme de Huffman

```
function
[entropie,longueur_moyenne,efficacite,sequence_decodee_brutie,sequence_decodee]=codage_de
codage_de_canal_huffman(message,mean,var)
 L=length(message)
  %probabilite des symbole----------------------------------------------------
compa=0;compb=0;compc=0;compd=0;compe=0;compf=0;compg=0;comph=0;compi=0;compj=
0;compk=0;compl=0;compm=0;compn=0;compo=0;compp=0;compq=0;compr=0;comps=0;com
pt=0;compu=0;compv=0;compw=0;compx=0;compy=0;
compz=0;compespace=0;comp0=0;comp1=0;comp2=0;comp3=0;comp4=0;comp5=0;comp6=0;c
omp7=0;comp8=0;comp9=0;compslach=0;
 symbole_de_source=[];
   for i=1:L
    if (message(i)=='a' || message(i)=='A') ;
   compa=compa+1;
    if compa==1
       rad(1)=1;
       symbole_de_source=[symbole_de_source,message(i)];
    end
    end
    if (message(i)=='b'|| message(i)=='B') ;
   compb=compb+1;
   if compb==1
        rad(2)=1;
```

```
    symbole_de_source=[symbole_de_source,message(i)];
 end
 end
 if (message(i)=='c' || message(i)=='C');
compc=compc+1;
 if compc==1
    rad(3)=1;
   symbole_de_source=[symbole_de_source,message(i)];
 end
 end
 if (message(i)=='d'|| message(i)=='D') ;
compd=compd+1;
if compd==1
    rad(4)=1;
    symbole_de_source=[symbole_de_source,message(i)];
 end
 end
 if (message(i)=='e' || message(i)=='E');
compe=compe+1;
 if compe==1
 rad(5)=1;
   symbole_de_source=[symbole_de_source,message(i)];
end
 end
 if (message(i)=='f' || message(i)=='F');
compf=compf+1;
 if compf==1
    rad(6)=1;
    symbole_de_source=[symbole_de_source,message(i)];
 end
 end
 if (message(i)=='g' || message(i)=='G');
compg=compg+1;
if compg==1
    rad(7)=1;
    symbole_de_source=[symbole_de_source,message(i)];
 end
 end
 if (message(i)=='h' || message(i)=='H');
comph=comph+1;
if comph==1
    rad(8)=1;
    symbole_de_source=[symbole_de_source,message(i)];
 end
 end
 if (message(i)=='i' || message(i)=='I');
compi=compi+1;
```

```
if compi==1
     rad(9)=1;
     symbole_de_source=[symbole_de_source,message(i)];
 end
 end
 if (message(i)=='j'|| message(i)=='J') ;
compj=compj+1;
 if compj==1
     rad(10)=1;
     symbole_de_source=[symbole_de_source,message(i)];
 end
 end
 if (message(i)=='k' || message(i)=='K');
compk=compk+1;
 if compk==1
     rad(11)=1;
     symbole_de_source=[symbole_de_source,message(i)];
 end
 end
 if (message(i)=='l'|| message(i)=='L') ;
compl=compl+1;
   if compl==1
     rad(12)=1;
    symbole_de_source=[symbole_de_source,message(i)];
 end
 end
 if( message(i)=='m' || message(i)=='M');
compm=compm+1;
 if compm==1
     rad(13)=1;
    symbole_de_source=[symbole_de_source,message(i)];
 end
 end
 if (message(i)=='n'|| message(i)=='N') ;
compn=compn+1;
if compn==1
     rad(14)=1;
     symbole_de_source=[symbole_de_source,message(i)];
 end
 end
 if (message(i)=='o' || message(i)=='O');
compo=compo+1;
if compo==1
     rad(15)=1;
    symbole_de_source=[symbole_de_source,message(i)];
end
 end
```

```
 if (message(i)=='p' || message(i)=='P');
compp=compp+1;
if compp==1
    rad(16)=1;
    symbole_de_source=[symbole_de_source,message(i)];
 end
 end
 if (message(i)=='q'|| message(i)=='Q') ;
compq=compq+1;
if compq==1
    rad(17)=1;
    symbole_de_source=[symbole_de_source,message(i)];
 end
 end
 if (message(i)=='r' || message(i)=='R');
compr=compr+1;
if compr==1
    rad(18)=1;
    symbole_de_source=[symbole_de_source,message(i)];
 end
 end
 if (message(i)=='s' || message(i)=='S');
comps=comps+1;
if comps==1
    rad(19)=1;
   symbole_de_source=[symbole_de_source,message(i)];
 end
 end
 if (message(i)=='t' || message(i)=='T');
compt=compt+1;
if compt==1
    rad(20)=1;
      symbole_de_source=[symbole_de_source,message(i)];
 end
 end
 if (message(i)=='u'|| message(i)=='U') ;
compu=compu+1;
 if compu==1
    rad(21)=1;
    symbole_de_source=[symbole_de_source,message(i)];
 end
 end
 if (message(i)=='v'|| message(i)=='V') ;
compv=compv+1;
if compv==1
    rad(22)=1;
    symbole_de_source=[symbole_de_source,message(i)];
```

```
 end
 end
 if (message(i)=='w'|| message(i)=='W') ;
compw=compw+1;
if compw==1
    rad(23)=1;
    symbole_de_source=[symbole_de_source,message(i)];
 end
 end
 if (message(i)=='x' || message(i)=='X');
compx=compx+1;
if compx==1
    rad(24)=1;
    symbole_de_source=[symbole_de_source,message(i)];
 end
 end
 if (message(i)=='y' || message(i)=='Y');
compy=compy+1;
if compy==1
    rad(25)=1;
    symbole_de_source=[symbole_de_source,message(i)];
 end
 end
 if (message(i)=='z'|| message(i)=='Z') ;
compz=compz+1;
if compz==1
    rad(26)=1;
    symbole_de_source=[symbole_de_source,message(i)];
 end
 end
 if message(i)==' ';
compespace=compespace+1;
 if compespace==1
    rad(27)=1;
    symbole_de_source=[symbole_de_source,message(i)];
end
 end
if message(i)=='0';
comp0=comp0+1;
 if comp0==1
    rad(28)=1;
    symbole_de_source=[symbole_de_source,message(i)];
end
end
if message(i)=='1';
comp1=comp1+1;
 if comp1==1
```

```
    rad(29)=1;
    symbole_de_source=[symbole_de_source,message(i)];
 end
end
 if message(i)=='2';
comp2=comp2+1;
if comp2==1
    rad(30)=1;
    symbole_de_source=[symbole_de_source,message(i)];
 end
 end
  if message(i)=='3';
comp3=comp3+1;
 if comp3==1
    rad(31)=1;
    symbole_de_source=[symbole_de_source,message(i)];
 end
  end
  if message(i)=='4';
comp4=comp4+1;
if comp4==1
    rad(32)=1;
    symbole_de_source=[symbole_de_source,message(i)];
end
  end
 if message(i)=='5';
comp5=comp5+1;
if comp5==1
    rad(33)=1;
    symbole_de_source=[symbole_de_source,message(i)];
 end
 end
  if message(i)=='6';
comp6=comp6+1;
if comp6==1
    rad(34)=1;
    symbole_de_source=[symbole_de_source,message(i)];
 end
  end
 if message(i)=='7';
comp7=comp7+1;
if comp7==1
    rad(35)=1;
    symbole_de_source=[symbole_de_source,message(i)];
end
 end
 if message(i)=='8';
```

```
    comp8=comp8+1;
    if comp8==1
        rad(36)=1;
        symbole_de_source=[symbole_de_source,message(i)];
     end
     end
      if message(i)=='9';
    comp9=comp9+1;
    if comp9==1
        rad(37)=1;
        symbole_de_source=[symbole_de_source,message(i)];
     end
      end
     if message(i)=='/';
    compslach=compslach+1;
     if compslach==1
        rad(38)=1;
        symbole_de_source=[symbole_de_source,message(i)];
    end
     end
   end
  radix=sum(rad);
 pa=compa/L;pb=compb/L;pc=compc/L;pd=compd/L;pe=compe/L;pf=compf/L;pg=compg/L;
ph=comph/L;pi=compi/L;pj=compj/L;pk=compk/L;pl=compl/L;pm=compm/L;pn=compn/L;
po=compo/L;pp=compp/L; pq=compq/L;pr=compr/L;ps=comps/L;pt=compt/L;pu=compu/L;
pv=compv/L;pw=compw/L;px=compx/L;py=compy/L;pz=compz/L;pcpa=compespace/L;
p0=comp0/L;p1=comp1/L;p2=comp2/L;p3=comp3/L;p4=comp4/L;p5=comp5/L;p6=comp6/L;
p7=comp7/L;p8=comp8/L;p9=comp9/L;pslach=compslach/L;
 %symbole
 %probabilite de chaque symbole-----------------------------------------
 radix=sum(rad);
proba=[pa pb pc pd pe pf pg ph pi pj pk pl pm pn po pp pq pr ps pt pu pv pw px py pz pcpa p0 p1
p2 p3 p4 p5 p6 p7 p8 p9 pslach];
j=1;
for i=1:length(proba)
    if proba(i)~=0
        proba_n(j)=proba(i);
        j=j+1;
    end;
end;
proba_n;
num_symbole=size(proba_n);
 somme_de_probabilite=sum(proba);
P=proba_n;
entropie=-sum(P.*log2(P))
length(P);
symbole_de_source
```

```
symbole=num2cell(symbole_de_source)
length(symbole);
[dict,longueur_moyenne] = huffmandict(symbole,P)
efficacite=(entropie/longueur_moyenne)*100
%codage-----------------------------------------------------------------
hcode = huffmanenco(message,dict)
[m,n]=size(hcode)
%decodage----------------------------------------------------------------
 deco_message_estime = huffmandeco(vecteur_source_estime,dict);
 sequence_decodee=char(deco_message_estime)';
deco_message_bruite=huffmandeco(texte_bruite,dict);
sequence_decodee_brutie=char(deco_message_bruite)'
```

Programme principal pour transmission d'un Son

```
function codage_canal_son_TEB;
clc
clear all
close all
refaire=1;
while(refaire==1)
clc
I=input('Introduire le nom de fichier audio à transmettre:  \n');
% exemple:
mean=input('Introduire la valeur moyenne du bruit Gaussien additif:  \n');
% exemple: 5
RSB_dB=input('Introduire le rapport signal sur bruit (RSB ou SNR en dB):  \n');
% exemple: 10
% CODES CONVOLUTIF IF FAUT DETERMINER LE PARAMETRE tblen: It is a positive
% integer scalar that specifies the traceback depth
tblen=input('Introduire la profondeur "tblen" pour l''algorithme de Viterbi:  \n');
% ceci est pour le codage convolutif. Exemple: 8
RSB=10^(RSB_dB/10);
var=(1/RSB);
[son_de_source,fs]=audioread(I);
%pause
%[son_source_estime,son_bruite,TEB]=codage_canal_son_Hamming7_4(son_de_source,fs,mea
n,var);  %  code de Hamming systématique

%[son_source_estime,son_bruite,TEB]=codage_canal_son_Hamming_non_syst7_4(son_de_sour
ce,fs,mean,var);  % code de Hamming non-systématique

%[son_source_estime,son_bruite,TEB]=codage_canal_son_Hamming15_11(son_de_source,fs,m
ean,var); %  code de Hamming systématique 15_11

[son_source_estime,son_bruite,TEB]=codage_canal_son_Hamming_conv(son_de_source,mean,v
ar,tblen);  % code convolutif
```

```
son_de_source=son_de_source(:,1);
son_de_source=son_de_source(1:8:length(son_de_source));
  % AFFICHAGE DES RESULTATS
figure(1)
   subplot(311)
   plot(son_de_source)
    A=min(son_de_source);
    AA=min(A);
   B=max(son_de_source);
   BB=max(B);
   axis([0 length(son_de_source) AA BB])
   sound(son_de_source,fs)
   pause
   title({'Son de source à transmettre'})
   figure(1)
   subplot(312)
   plot(son_bruite)
   min(son_bruite) ;
   max(son_bruite);
   axis([0 length(son_bruite) min(son_bruite) max(son_bruite)])
  sound(son_bruite,fs)
  pause
   title({'Son bruité reçu sans codage de canal'})
   figure(1)
   subplot(313)
   plot(son_source_estime)
  axis([0 length(son_source_estime) min(son_source_estime) max(son_source_estime)])
  sound(son_source_estime,fs)
   title({'Son reçu avec codage de canal'})
   figure(2)
   subplot(211)
   plot(son_de_source)
   %sound(son_de_source,fs)
   title({'Son de source à transmettre'})
   figure(2)
   subplot(212)
   plot(son_source_estime)
   title({'Son reçu avec codage de canal'})
       %AFICHAGE DE SPECTRE
N_fft=fs;
y_f=fft(son_de_source,N_fft);
Spect_y=abs(y_f);
Spect_y_plot=Spect_y(1:N_fft/2);
f=fs/2*(0:1/(N_fft/2-1):1);
figure(3)                          %%%%% le spectre de fréquence
subplot(211)
plot(f,Spect_y_plot)
```

```
axis([min(f) max(f) min(Spect_y_plot) max(Spect_y_plot)])
title({'Spectre du Son de source à transmettre'})
f_f=fft(son_source_estime,N_fft);
Spect_f=abs(f_f);
Spect_f_plot=Spect_f(1:N_fft/2);
f=fs/2*(0:1/(N_fft/2-1):1);
figure(3)                          %%%%% le spectre de fréquence
subplot(212)
plot(f,Spect_f_plot)
axis([min(f) max(f) min(Spect_f_plot) max(Spect_f_plot)])
title({'Spectre du Son reçu avec codage de canal'})
% FIN D AFFICHAGE DES RESULTATS
refaire=input('Voulez vous continuer?  (taper 1 pour continuer)\n');
end;
```

Codeur/décodeur de Hamming Systématique (7,4)- calcul de TEB

```
function
[son_source_estime,son_bruite,TEB]=codage_canal_son_Hamming7_4(son_de_source,fs,mean,
var)
y=son_de_source;
y1=y(:,1);
y2=y1(1:8:length(y1));
size(y1);
mu = lin2mu(y2);
 son_source=mu';
  son_source_vecteur=dec2bin(son_source);
  [m,n]=size(son_source_vecteur);
 size(son_source)
 son_source_vecteur_binaire=[];
 for i=1:m;
    for j=1:n;
      son_source_vecteur_binaire=[son_source_vecteur_binaire son_source_vecteur(i,j)];
    end
end
[bm,bn]=size(son_source_vecteur_binaire)
% Binarisation du signial
son_source_vecteur_binaire=son_source_vecteur_binaire/1;
for i=1:length(son_source_vecteur_binaire);
    if (son_source_vecteur_binaire(i)==49) son_source_vecteur_binaire(i)=1;
    else son_source_vecteur_binaire(i)=0;
end;
end;
 % CODE DE HAMMING SYSTEMATIQUE
G=[1 0 0 0 1 0 1
   0 1 0 0 0 1 1
   0 0 1 0 1 1 0
```

```
  0 0 0 1 1 1 1];
%Multiplication par la matrice G
count=1;mm=1; mot_de_code=[];
son_source_vecteur_binaire(1:count+3);
size(son_source_vecteur_binaire);
while count<=(length(son_source_vecteur_binaire)-4);
    sequence_a_coder(mm,:)=son_source_vecteur_binaire(count:count+3);
  mot_de_code=[mot_de_code,sequence_a_coder(mm,:)*G];
  count=count+4;
    mm=mm+1;
 end;
  for i=1:length(mot_de_code);
if mod(mot_de_code(i),2)==0 mot_de_code(i)=0;
else
   mot_de_code(i)=1;
end;
end;
mot_de_code;
%On génère le bruit
[m,n]=size(son_source);
bruit=mean+sqrt(var)*randn(m,n);
son_source_bruitee=son_source+bruit;   %son bruité sans codage de canal
size(y1);
size(bruit);
[m1,n1]=size(y1);
son_bruite=y1+mean+sqrt(var)*randn(m1,n1);
% AJOUT DE BRUIT%%%
mot_de_code_bruite=mot_de_code+mean+sqrt(var)*randn(1,length(mot_de_code));
son_source_vecteur_bruite=son_source_vecteur_binaire+sqrt(var)*randn(1,length(son_source_v
ecteur_binaire));
seuil=sum(son_source_vecteur_bruite)/length(son_source_vecteur_bruite);
%seuil=0.5;
for i=1:length(son_source_vecteur_bruite);
if son_source_vecteur_bruite(i)<seuil
   son_source_vecteur_binaire_bruite(i)=0;
else son_source_vecteur_binaire_bruite(i)=1;
end;
end;
% DECISION PAR SEUILLAGE SEUIL=0.5
seuil=sum(mot_de_code_bruite)/length(mot_de_code_bruite);
%seuil=0.5;
for i=1:length(mot_de_code_bruite);
if mot_de_code_bruite(i)<seuil mot_recu(i)=0;
else mot_recu(i)=1;
end;
end;
%devision de mot recu en paquet de 7 bit
```

```
count=1;mm=1;
mot_recu_paquet7=zeros(length(mot_recu)/7,7);
while count<=length(mot_recu);
   mot_recu_paquet7(mm,:)=mot_recu(count:count+6);
   count=count+7;
   mm=mm+1;
 end;
 %Detection d'erreur
H=[1 0 1 1 1 0 0 %matrice de controle
    0 1 1 1 0 1 0
    1 1 0 1 0 0 1];
Ht=H';
count=1;
while count<=length(mot_recu_paquet7);
    syndrome_d(count,:)=mot_recu_paquet7(count,:)*Ht;
    syndrome(count,:)=mod(syndrome_d(count,:),2);
    count=count+1;
end
vect_po=zeros(length(syndrome),1);      %vecteur postion d'error
for co1=1:length(syndrome);
    for co2=1:length(Ht);
        if(syndrome(co1,:)==Ht(co2,:));
            vect_po(co1)=co2;
        end
    end
end
vect_error=zeros(length(vect_po),7);    %vecteur error
mot_corriger=mot_recu_paquet7;          %mot corriger
for co=1:length(vect_po);
    if(vect_po(co)~=0);
        vect_error(co,vect_po(co))=1;
        mot_corriger(co,vect_po(co))= ~(mot_corriger(co,vect_po(co)));
    end
end

count=1;mm=1;

while mm<=length(mot_corriger);
   message_detecte(mm,:)=mot_corriger(mm,1:4);
   mm=mm+1;
 end;
count=1;
for i=1:size(message_detecte,1)
for j=1:size(message_detecte,2)
vecteur_source_binaire_estime(count)=message_detecte(i,j);
count=count+1;
end;
```

```
end;

count=1;vecteur_source_hex_estime=[];
while count<=(length(vecteur_source_binaire_estime)-8)
vecteur_source_binaire_estime_string=num2str(vecteur_source_binaire_estime(count:count+8-
1));
vecteur_source_hex_estime=[vecteur_source_hex_estime,bin2dec(vecteur_source_binaire_estim
e_string)];
count=count+8;
end;
vecteur_source_hex_estime(1,m*n)=0;
vecteur_source_hex_estime;
son_source_estime=[];count=1;
for i=1:m
     son_source_estime(i,1:n)=vecteur_source_hex_estime(count:count+n-1);
     count=count+n;
end
%inverse-----------------------------------------------------------------
  % son_source_estime=son_source_estime(1:(1/8):length(son_source_estime));
  son_source_estime=mu2lin(son_source_estime)
  %son_source_estime=son_source_estime(1:75700)
% CALCUL DE TEB
longueur_message=length(son_source_vecteur_binaire);
TEB=sum(xor(son_source_vecteur_binaire(1:longueur_message-
4),vecteur_source_binaire_estime))/longueur_message
 a=son_source_vecteur_binaire(1:longueur_message-4);
b=vecteur_source_binaire_estime;
 [nombre1,TEB_SC]=biterr(son_source_vecteur_binaire,son_source_vecteur_binaire_bruite)
[nombre2,TEB_AC]=biterr(a,b)
TEB=TEB_AC
TEB_AC
TEB_SC
```

Codeur/décodeur de Hamming non-systématique (7,4)- calcul de TEB

```
function
[son_source_estime,son_bruite,TEB]=codage_canal_son_Hamming_non_syst7_4(son_de_source
,fs,mean,var)
y=son_de_source;
y1=y(:,1);
y2=y1(1:8:length(y1));
size(y1);
mu = lin2mu(y2);
 son_source=mu';
son_source_vecteur=dec2bin(son_source);
[m,n]=size(son_source_vecteur);
son_source_vecteur_binaire=[];
```

```
for i=1:m
   for j=1:n
     son_source_vecteur_binaire=[son_source_vecteur_binaire son_source_vecteur(i,j)];
   end
end
[bm,bn]=size(son_source_vecteur_binaire);
%Binarisation du signal %%%%%%%%%%%%%%
son_source_vecteur_binaire=son_source_vecteur_binaire/1;
for i=1:length(son_source_vecteur_binaire)
   if (son_source_vecteur_binaire(i)==49) son_source_vecteur_binaire(i)=1;
   else son_source_vecteur_binaire(i)=0;
end;
end;
% CODE DE HAMMING NON SYSTEMATIQUE%%%%%%%%%%%%
G=[0 0 0 0 1 1 1
   0 0 1 1 0 0 1 %Matrice de controle
   0 1 0 1 0 1 0
   1 0 0 1 0 1 1];
%codage%%%
count=1;mm=1; mot_de_code=[];
son_source_vecteur_binaire(1:count+3);
while count<=(length(son_source_vecteur_binaire)-4)
    sequence_a_coder(mm,:)=son_source_vecteur_binaire(count:count+3);
    mot_de_code=[mot_de_code,sequence_a_coder(mm,:)*G];
  count=count+4;
    mm=mm+1;
 end;
 for i=1:length(mot_de_code)
if mod(mot_de_code(i),2)==0 mot_de_code(i)=0;
else
   mot_de_code(i)=1;
end;
end;
%creation de bruit gaussien%%%
[m,n]=size(son_source);
bruit=mean+sqrt(var)*randn(m,n);
son_bruite=son_source+bruit;   %son bruité sans codage de canal
size(y1);
size(bruit);
[m1,n1]=size(y1);
son_bruite=y1+mean+sqrt(var)*randn(m1,n1);
% AJOUT DE BRUIT
mot_de_code_bruite=mot_de_code+mean+sqrt(var)*randn(1,length(mot_de_code));
son_source_vecteur_bruite=son_source_vecteur_binaire+sqrt(var)*randn(1,length(son_source_v
ecteur_binaire));
seuil=sum(son_source_vecteur_bruite)/length(son_source_vecteur_bruite);
%seuil=0.5;
```

```
for i=1:length(son_source_vecteur_bruite)
if son_source_vecteur_bruite(i)<seuil son_source_vecteur_binaire_bruite(i)=0;
else son_source_vecteur_binaire_bruite(i)=1;
end;
end;
% DECISION PAR SEUILLAGE SEUIL=0.5
seuil=sum(mot_de_code_bruite)/length(mot_de_code_bruite);
for i=1:length(mot_de_code_bruite)
if mot_de_code_bruite(i)<seuil mot_recu(i)=0;
else mot_recu(i)=1;
end;
end;
%devision de mot recu en paquet de 7 bit
count=1;mm=1;
mot_recu_paquet7=zeros(length(mot_recu)/7,7);
while count<=length(mot_recu)
  mot_recu_paquet7(mm,:)=mot_recu(count:count+6);
  count=count+7;
  mm=mm+1;
 end;
 %Detection d'erreur
comp=[0 0 0
          0 0 1
          0 1 0
          0 1 1
          1 0 0
          1 0 1
          1 1 0
          1 1 1];
 %bit de redandance
c_vect=zeros(length(mot_recu_paquet7),3);
for count=1:length(mot_recu_paquet7)
   c2=xor(xor(mot_recu_paquet7(count,7-3),mot_recu_paquet7(count,7-
2)),xor(mot_recu_paquet7(count,7-1),mot_recu_paquet7(count,7)));
   c1=xor(xor(mot_recu_paquet7(count,7-5),mot_recu_paquet7(count,7-
4)),xor(mot_recu_paquet7(count,7-1),mot_recu_paquet7(count,7)));
   c0=xor(xor(mot_recu_paquet7(count,7-6),mot_recu_paquet7(count,7-
4)),xor(mot_recu_paquet7(count,7-2),mot_recu_paquet7(count,7)));
   c_vect(count,:)=[c2 c1 c0];
end
err_pos=zeros(length(c_vect),1);
mot_corriger=mot_recu_paquet7;
for count=1:length(c_vect)
   for count2=0:7
      if(c_vect(count,:)==comp(count2+1,:))
         err_pos(count,1)=8-count2;
      end
```

```
    end
end
for count=1:length(c_vect)
    if(err_pos(count)~=8)
        mot_corriger(count,err_pos(count))= ~mot_corriger(count,err_pos(count));
    end
end
 message_detecte=zeros(length(mot_corriger),4);
 for count=1:length(mot_corriger)
D0= mot_corriger(count,7-2);
D1=mot_corriger(count,7-4);
D2=mot_corriger(count,7-5);
D3=mot_corriger(count,7-6);
message_detecte(count,:)=[D0 D1 D2 D3];
 end
message_detecte_vecteur=[];
for i=1:size(message_detecte,1)
    for j=1:size(message_detecte,2)
       message_detecte_vecteur=[message_detecte_vecteur message_detecte(i,j)];
    end
end
count=1;
for i=1:size(message_detecte,1)
for j=1:size(message_detecte,2)
vecteur_source_binaire_estime(count)=message_detecte(i,j);
count=count+1;
end;
end;
count=1;vecteur_source_hex_estime=[];
while count<=(length(vecteur_source_binaire_estime)-8)
vecteur_source_binaire_estime_string=num2str(vecteur_source_binaire_estime(count:count+8-
1));
vecteur_source_hex_estime=[vecteur_source_hex_estime,bin2dec(vecteur_source_binaire_estim
e_string)];
count=count+8;
end;
vecteur_source_hex_estime(1,m*n)=0;
vecteur_source_hex_estime;
size(vecteur_source_hex_estime)
son_source_estime=[];count=1;
for i=1:m
      son_source_estime(i,1:n)=vecteur_source_hex_estime(count:count+n-1);
      count=count+n;
end
%inverse----------------------------------------------------------------------
   %son_source_estime=son_source_estime(1:(1/8):length(son_source_estime));
  son_source_estime=mu2lin(son_source_estime)
```

```
% CALCUL DE TEB
longueur_message=length(son_source_vecteur_binaire);
TEB=sum(xor(son_source_vecteur_binaire(1:longueur_message-
4),vecteur_source_binaire_estime))/longueur_message
a=son_source_vecteur_binaire(1:longueur_message-4);
b=vecteur_source_binaire_estime;
 [nombre1,TEB_AC]=biterr(son_source_vecteur_binaire,son_source_vecteur_binaire_bruite)
[nombre2,TEB_SC]=biterr(a,b)
TEB=TEB_AC
TEB_AC
TEB_SC
```

Codeur/décodeur de Hamming Systématique (15,11)

```
function
[son_source_estime,son_bruite,TEB]=codage_canal_son_Hamming15_11(son_de_source,fs,mea
n,var)
y=son_de_source;
y1=y(:,1);
y2=y1(1:8:length(y1));
size(y1);
mu = lin2mu(y2);
 son_source=mu';
son_source_vecteur=dec2bin(son_source);
[m,n]=size(son_source_vecteur);
son_source_vecteur_binaire=[];
for i=1:m;
    for j=1:n;
      son_source_vecteur_binaire=[son_source_vecteur_binaire son_source_vecteur(i,j)];
    end
end
[bm,bn]=size(son_source_vecteur_binaire);
 %binarisation du son
son_source_vecteur_binaire=son_source_vecteur_binaire/1;
for i=1:length(son_source_vecteur_binaire);
    if (son_source_vecteur_binaire(i)==49) son_source_vecteur_binaire(i)=1;
    else son_source_vecteur_binaire(i)=0;
end;
end;
% CODE DE HAMMING SYSTEMATIQUE 15-11
%matrice génératrice
G=[1 0 0 0 0 0 0 0 0 0 0 1 1 1 1
   0 1 0 0 0 0 0 0 0 0 0 1 1 1 0
   0 0 1 0 0 0 0 0 0 0 0 1 1 0 1
   0 0 0 1 0 0 0 0 0 0 0 1 1 0 0
   0 0 0 0 1 0 0 0 0 0 0 1 0 1 1
   0 0 0 0 0 1 0 0 0 0 0 1 0 1 0
```

```
    0 0 0 0 0 0 1 0 0 0 0 1 0 0 1
    0 0 0 0 0 0 0 1 0 0 0 0 0 1 1
    0 0 0 0 0 0 0 0 1 0 0 0 1 1 1
    0 0 0 0 0 0 0 0 0 1 0 0 1 1 0
    0 0 0 0 0 0 0 0 0 0 1 0 1 0 1];
%multiplication par la matrice génératrice (construirele mot de code)
count=1;mm=1; mot_de_code=[];
son_source_vecteur_binaire(1:count+10);
while count<=(length(son_source_vecteur_binaire)-11);
      sequence_a_coder(mm,:)=son_source_vecteur_binaire(count:count+10);
    mot_de_code=[mot_de_code,sequence_a_coder(mm,:)*G];
    count=count+11;
      mm=mm+1;
 end;
%binarisation par seuillage
 for i=1:length(mot_de_code);
if mod(mot_de_code(i),2)==0 mot_de_code(i)=0;
else
     mot_de_code(i)=1;
end;
end;
 %création du bruit
[m,n]=size(son_source);
bruit=mean+sqrt(var)*randn(m,n);
son_source_bruitee=son_source+bruit;   %son bruité sans codage de canal
size(y1);
size(bruit);
[m1,n1]=size(y1);
son_bruite=y1+mean+sqrt(var)*randn(m1,n1);  %son bruitée sans codage de canal
 % AJOUT DE BRUIT
mot_de_code_bruite=mot_de_code+mean+sqrt(var)*randn(1,length(mot_de_code));
 %%%%%%%%%%%%%%%%%%%%%%%%%
son_source_vecteur_bruite=son_source_vecteur_binaire+sqrt(var)*randn(1,length(son_source_v
ecteur_binaire));
seuil=sum(son_source_vecteur_bruite)/length(son_source_vecteur_bruite);
for i=1:length(son_source_vecteur_bruite);
if son_source_vecteur_bruite(i)<seuil son_source_vecteur_binaire_bruite(i)=0;
else son_source_vecteur_binaire_bruite(i)=1;
end;
end;
 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% DECISION PAR SEUILLAGE SEUIL=0.5
seuil=sum(mot_de_code_bruite)/length(mot_de_code_bruite);
%seuil=0.5;
for i=1:length(mot_de_code_bruite);
if mot_de_code_bruite(i)<seuil mot_recu(i)=0;
else mot_recu(i)=1;
```

```
end;
end;
mot_recu;
%Décodage%%%%%% %devision de mot recu en paquet de 7 bit
count=1;mm=1;
mot_recu_paquet15=zeros(length(mot_recu)/15,15);
while count<=length(mot_recu);
   mot_recu_paquet15(mm,:)=mot_recu(count:count+14);
   count=count+15;
   mm=mm+1;
 end;
%Detection d'erreur
H=[1 1 1 1 1 1 1 0 0 0 0 1 0 0 0     %matrice de controle
    1 1 1 1 0 0 0 0 1 1 1 0 1 0 0
    1 1 0 0 1 1 0 1 1 1 0 0 0 1 0
    1 0 1 0 1 0 1 1 1 0 1 0 0 0 1];
Ht=H';
count=1;
while count<=length(mot_recu_paquet15)
    syndrome_d(count,:)=mot_recu_paquet15(count,:)*Ht;
    syndrome(count,:)=mod(syndrome_d(count,:),2);
    count=count+1;
end
vect_po=zeros(length(syndrome),1);      %vecteur postion d'error
for co1=1:length(syndrome);
    for co2=1:length(Ht);
        if(syndrome(co1,:)==Ht(co2,:));
            vect_po(co1)=co2;
        end
    end
end
vect_error=zeros(length(vect_po),15);    %vecteur error
mot_corriger=mot_recu_paquet15;          %mot corriger

for co=1:length(vect_po);
    if(vect_po(co)~=0);
        vect_error(co,vect_po(co))=1;
        mot_corriger(co,vect_po(co))= ~(mot_corriger(co,vect_po(co)));
    end
end
size(mot_corriger);
count=1;mm=1;
while mm<=length(mot_corriger)
   message_detecte(mm,:)=mot_corriger(mm,1:11);
   mm=mm+1;
 end;
count=1;
```

```
for i=1:size(message_detecte,1);
for j=1:size(message_detecte,2);
vecteur_source_binaire_estime(count)=message_detecte(i,j);
count=count+1;
end;
end;
count=1;vecteur_source_hex_estime=[];
while count<=(length(vecteur_source_binaire_estime)-8);
vecteur_source_binaire_estime_string=num2str(vecteur_source_binaire_estime(count:count+8-
1));
vecteur_source_hex_estime=[vecteur_source_hex_estime,bin2dec(vecteur_source_binaire_estim
e_string)];
count=count+8;
end;
vecteur_source_hex_estime(1,m*n)=0;
son_source_estime=[];count=1;
for i=1:m
    son_source_estime(i,1:n)=vecteur_source_hex_estime(count:count+n-1);
    count=count+n;
end
%inverse-----------------------------------------------------------------------
 % son_source_estime=son_source_estime(1:(1/8):length(son_source_estime));
 son_source_estime=mu2lin(son_source_estime)
% CALCUL DE TEB
size(son_source_vecteur_binaire)
size(vecteur_source_binaire_estime)
longueur_message=length(son_source_vecteur_binaire);
TEB=sum(xor(son_source_vecteur_binaire(1:longueur_message-
10),vecteur_source_binaire_estime))/longueur_message;
 a=son_source_vecteur_binaire(1:longueur_message-10);
b=vecteur_source_binaire_estime;
size(son_source_vecteur_binaire_bruite)
size(a);
[nombre1,TEB_SC]=biterr(son_source_vecteur_binaire,son_source_vecteur_binaire_bruite)
[nombre2,TEB_AC]=biterr(a,b)
TEB=TEB_AC;
TEB_SC
TEB_AC
```

Codeur/décodeur convolutif r=1/2, m=2, G=(6 7)$_{(8)}$, L=3- calcul de TEB

```
function
[son_source_estime,son_bruite,TEB]=codage_canal_son_Hamming_conv(son_de_source,mean,v
ar,tblen)
y=son_de_source;
y1=y(:,1);
y2=y1(1:8:length(y1));
size(y1);
```

```
mu = lin2mu(y2);
 son_source=mu';
son_source_vecteur=dec2bin(son_source);
[m,n]=size(son_source_vecteur);
son_source_vecteur_binaire=[];
for i=1:m
    for j=1:n
      son_source_vecteur_binaire=[son_source_vecteur_binaire son_source_vecteur(i,j)];
    end
end
[bm,bn]=size(son_source_vecteur_binaire);
son_source_vecteur_binaire=son_source_vecteur_binaire/1;
for i=1:length(son_source_vecteur_binaire)
    if (son_source_vecteur_binaire(i)==49) son_source_vecteur_binaire(i)=1;
    else son_source_vecteur_binaire(i)=0;
end;
end;
%image_source_vecteur_binaire
% mean=val_moy;
% var=variance;
% [m,n]=size(image_source);
bruit=mean+sqrt(var)*randn(m,n);
% image_source_bruitee=im2double(image_source)+bruit;   %image bruitée sans codage de
canal
[m,n]=size(son_source);
bruit=mean+sqrt(var)*randn(m,n);
son_source_bruitee=son_source+bruit;   %son bruité sans codage de canal
size(y1);
size(bruit);
[m1,n1]=size(y1);
son_bruite=y1+mean+sqrt(var)*randn(m1,n1);
son_source_vecteur_bruite=son_source_vecteur_binaire+sqrt(var)*randn(1,length(son_source_v
ecteur_binaire));
seuil=sum(son_source_vecteur_bruite)/length(son_source_vecteur_bruite);
for i=1:length(son_source_vecteur_bruite);
if son_source_vecteur_bruite(i)<seuil son_source_vecteur_binaire_bruite(i)=0;
else son_source_vecteur_binaire_bruite(i)=1;
end;
end;
% CODAGE CONVOLUTIF
constlen = 7;G = [171 133];
%trellis = poly2trellis(constlen,G);    % tblen=8 parfait pour SNR=12 dB; constlen = 7;G = [171
133];
if isempty(tblen) tblen = 8;end;
trellis = poly2trellis(7,[177 133]);   % bon résultat avec tblen = 8 pour 10 dB
%trellis = poly2trellis([5 4],[23 35 0; 0 5 13],[1 1 1 0 0 1]);
%trellis = poly2trellis([5 4],[23 35 0; 0 5 13]);
```

```
mot_de_code=convenc(son_source_vecteur_binaire,trellis);
mot_de_code_bruite=mot_de_code+mean+sqrt(var)*randn(1,length(mot_de_code));
size(mot_de_code_bruite);
mot_de_code_bruite;
% DECISION PAR SEUILLAGE SEUIL=0.5
seuil=sum(mot_de_code_bruite)/length(mot_de_code_bruite);
%seuil=0.5;
for i=1:length(mot_de_code_bruite)
if mot_de_code_bruite(i)<seuil mot_recu(i)=0;
else mot_recu(i)=1;
end;
end;
mot_recu;
%tblen=5*constlen;
% Decodage de Viterbi
%vecteur_source_binaire_estime=vitdec(mot_recu, trellis, tblen, 'cont','unquant');
vecteur_source_binaire_estime=vitdec(mot_recu, trellis, tblen, 'cont', 'hard');   % parfait pour
tblen=8, SNR=12 dB
%vecteur_source_binaire_estime = vitdec(mot_recu, trellis, tblen, 'trunc', 'unquant',[1 1 1 0 0 1],
erasures); % Decode.
% [x,qcode] = quantiz(mot_recu,[-.75 -.5 -.25 0 .25 .5 .75],...
% 7:-1:0); % Values in qcode are between 0 and 2^3-1.
%vecteur_source_binaire_estime = vitdec(mot_recu, trellis, tblen,'cont','soft',3);
size(vecteur_source_binaire_estime);
vecteur_source_binaire_estime;
count=1;vecteur_source_hex_estime=[];
%vecteur_source_hex_estime=zeros(1,m*n);
while count<=(length(vecteur_source_binaire_estime)-8)
vecteur_source_binaire_estime_string=num2str(vecteur_source_binaire_estime(count:count+8-
1));
vecteur_source_hex_estime=[vecteur_source_hex_estime,bin2dec(vecteur_source_binaire_estim
e_string)];
count=count+8;
end;
vecteur_source_hex_estime(1,m*n)=0;
vecteur_source_hex_estime;
size(vecteur_source_hex_estime);
%[m,n]=size(image_source_vecteur);
son_source_estime=[];count=1;
for i=1:m
    son_source_estime(i,1:n)=vecteur_source_hex_estime(count:count+n-1);
    count=count+n;
end
%inverse-----------------------------------------------------------------
  son_source_estime=son_source_estime(1:1/8:length(son_source_estime));
 son_source_estime=mu2lin(son_source_estime);
size(son_source_estime)
```

```
 size(son_de_source)
% CALCUL DE TEB
longueur_message=length(son_source_vecteur_binaire);
TEB=sum(xor(son_source_vecteur_binaire(1:longueur_message),vecteur_source_binaire_estime
))/longueur_message;
[nChnlErrs TEB_AC] = biterr(son_source_vecteur_binaire(1:longueur_message-
8),vecteur_source_binaire_estime(9:length(vecteur_source_binaire_estime)))
%TEB=TEB_AC%
[nombre1,TEB_SC]=biterr(son_source_vecteur_binaire,son_source_vecteur_binaire_bruite)
% [nombre2,TEB_AC]=biterr(a,b)
```