Telecommunications

TS226

Rapport

TP de Codage de canal

Etudiants :
Ferron Kylian
Bousquet Nicolas

Professeurs :
Romain Tajan
Malek Ellouze

Décembre 2021

Table des matières

1 Introduction

Dans ce TP, on cherche à mettre en place différentes chaînes de communications numériques et à comprendre leurs différentes caractéristiques(performances en termes de probabilité d'erreur binaire, en termes de rendement et aussi en termes de débit).

Pour réaliser ce TP, deux fonctions ont été implémentées à savoir une fonction d'encodage et son inverse, une fonction de décodage. Ici, on s'intéresse au cas où les chaînes de communications numériques sont codées à partir d'un code convolutif.

4 codes différents seront utilisés et associés à un treillis grâce à la fonction poly2trellis. Ce trellis nous donnera notamment le tableau des états et des sorties à chaque étape de l'encodage ou décodage du mot de code considéré (u généré aléatoirement). Chacun de ces codes possède une entrée et deux sorties. Ces 4 codes convolutifs sont les suivants :

$$(2,3)_8$$

$$(5,7)_8$$

$$(13,15)_8$$

$$(131,171)_8$$

Après l'explication de ces algorithmes, nous étudierons l'impact de la mémoire sur le code et l'analyse des courbes TEB (Taux d'erreur binaire) obtenues sachant que les deux fonctions traitent les cas de fermeture ou non (retour à l'état 0).

2 Encodage

Pour la fonction d'encodage, appelée cc_encode, il faudra bien évidemment le mot de code u, le trellis associé au code convolutif, l'état initial de celui-ci(à savoir 0 dans notre cas) ainsi que la variable closed indiquant si l'étape de fermeture est prise en compte ou pas.

L'algorithme parcourt tout le mot de code et en fonction de chaque élément du mot et de l'état actuel de la chaîne va renvoyer le nouvel état de la chaîne qui sera utilisé à la prochaine itération. Grâce à l'attribut nextStates du treillis, il suffira donc de continuer de parcourir le mot tout en récupérant le nouvel état.

De la même manière, à chaque itération en fonction de l'état actuel et de l'élément du mot de code, on récupère grâce à l'attribut outputs du treillis, la sortie que l'on ajoutera au mot du code de sortie c. Les sorties de outputs sortent en décimal nécessitant d'être converties pour être ajoutées dans le code c (str2num étant gourmande en temps a donc été remplacée par str2double). Sous Matlab, ces deux attributs sont réprésentés de la manière suivante pour poly2trellis(3, [5,7]) par exemple :

État \ Entrée	0	1
0	0	2
1	0	2
2	1	3
3	1	3

FIGURE 1 – Tableau des états en fonction de l'état actuel et du symbole d'entrée

État \ Entrée	0	1
0	0	3
1	3	0
2	1	2
3	2	1

FIGURE 2 – Tableau des sorties en fonction de l'état actuel et du symbole d'entrée

Pour la fermeture (si closed == 1), il suffit de reprendre le même principe mais en forçant l'état suivant à se rapprocher le plus possible de 0 ou de retourner à 0 directement si possible (deux étapes pour le code convolutif mentionné plus haut).

Cette fonction d'encodage a été pensée pour acceuillir des codes convolutifs possédant un nombre de sorties plus conséquent (seulement deux sorties pour chacun des codes considérés) grâce à l'attribut du treillis numOutputSymbols retournant le nombre de sorties possibles.

3 Décodage

Pour la fonction de décodage, il a été choisi d'utiliser l'algorithme de Viterbi. Le but de cette fonction étant de retrouver parfaitement le code à partir d'un signal modulé par une BPSK, cet algorithme va permettre de ne pas avoir à parcourir tous les chemins du trellis. En effet, imaginons que la chaîne possède 4 états possibles, l'algorithme va permettre de travailler avec seulement 4 chemins notamment grâce aux poids.

A partir d'une matrice de 1000 appelée mat_poids (1000 a été choisi pour que ce nombre ne soit jamais atteignable par le poids) et en parcourant le signal observé, les poids sont calculées pour chaque itération et pour chaque état. Cette matrice est initialisée en plus à partir de s_i qui vaut pour chaque cas 0 comme pour l'encodage. Sachant que tous les chemins sont analysés mais qu'il existe pour chaque étape seulement 4 poids dans la matrice, on retrouve tout l'intérêt de l'algorithme de Viterbi en ne gardant que le poids le plus faible des chemins qui se rejoignent sur un même état. De plus, une matrice appelée mat_states de même taille que la précédente retrace les états précédents dans lesquels chaque chemin est passé.

Bien sûr, le calcul des poids pour chaque étape et donc le suivi de chaque chemin pour ce calcul n'est possible que grâce aux mêmes attributs du trellis utilisés dans la partie encodage. Les sorties étant en décimal, l'utilisation de fonctions de conversion comme de2bi est nécessaire.

Ensuite, encore pour chaque étape, on remplit la matrice mat_u (taille identique à ses deux compères) par un 0 ou un 1 en fonction du chemin pris. Le but final de cet algorithme est de trouver le poids final le plus faible à la fin de la boucle. À partir de ce résultat, on parcourt mat_u à l'envers en prenant en compte mat_states et sa liste d'états précédents pour chaque chemin.

Pour la gestion de la fermeture au niveau de ce décodage, il suffit d'enlever à u les q derniers bits sachant que :

$$q = log_2(trellis.numStates)$$

Pour cette fonction de décodage contrairement à celle d'encodage, elle n'est fonctionnelle que pour des codes convolutifs comme ceux utilisés dans le TP (une entrée, deux sorties).

4 Analyse et étude de l'impact de la mémoire sur le code

Voici les courbes de TEB et de TEP obtenues pour chacun des codes convolutifs avec et sans fermeture :

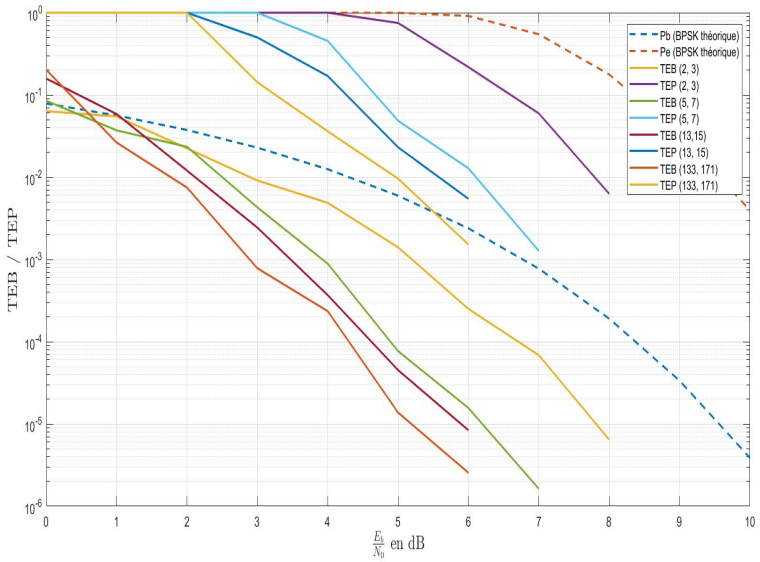

FIGURE 3 – Courbes des TEB/TEP pour les differents types de codes avec treillis ouvert

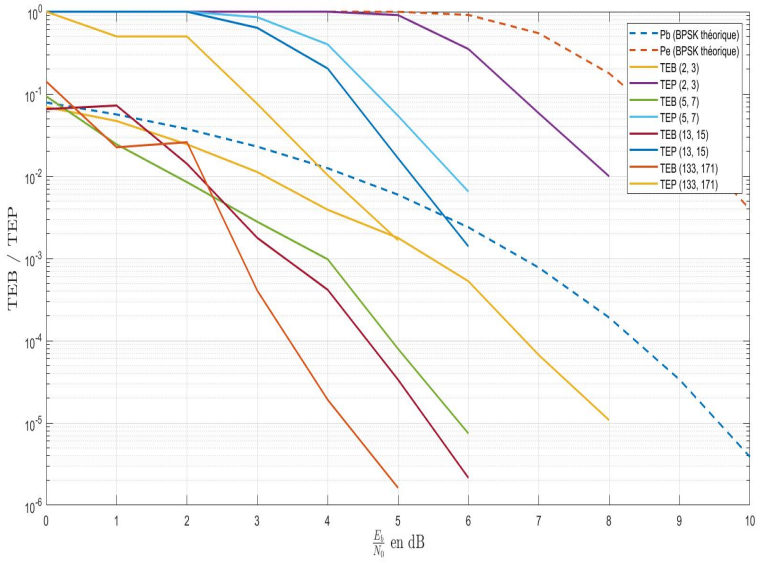

FIGURE 4 – Courbes des TEB/TEP pour les differents types de codes avec treillis fermé

Le rendement choisi pour ces courbes est de 0.5. Tout d'abord, on peut confirmer le fait que le nombre de bits faux est plus fort pour un SNR (rapport signal/bruit) faible. On peut ajouter que plus la mémoire est grande, plus le nombre de bits faux diminue. De plus, on remarque que plus le code convolutif possède de mémoires, plus le gain de codage va être fort. En effet, ce

codage de canal permet de rajouter de la redondance dans le message permettant de minimiser l'impact d'une erreur sur la compréhension du message. Ainsi une mémoire plus longue incluent plus d'opérationscdonc un débit réduit mais plus de fiabilité avec un gain de codage plus fort.

La différence entre une chaîne sans fermeture et une chaîne avec fermeture n'est pas notable à part pour le cas du dernier code convolutif $(133, 171)_8$. En effet, plus la mémoire est grande, plus il y aura d'étapes de fermeture impliquant sûrement un gain plus important.

Pour le débit, on remarque que plus la mémoire du code convolutif est importante, plus le débit est faible. Même si des valeurs trop faibles ne nous ont pas permis d'afficher concrètement les valeurs du débit, on remarque que celui-ci se divise par deux pour chaque incrémentation de la mémoire.

Pour la distance minimale, elle peut être trouvée pour chaque code convolutif avec la fonction MATLAB distspec. On remarque qu'il y a une proportionnalité entre le nombre de mémoires et la distance minimale.

Le tableau suivant regroupe les différentes valeurs caractérisant ce qui a été évoqué plus haut (ce tableau est similaire pour une chaîne sans fermeture à la différence d'un gain de codage plus fort pour le dernier code convolutif) :

Type	$(2, 3)_8$	$(5, 7)_8$	$(13, 15)_8$	$(133, 171)_8$
Mémoire	1	2	3	6
Nombre d'états	2	4	8	64
Distance minimale	3	5	6	8
Gain de codage (en dB)	1,8	3,8	3,9	4,5

5 Évaluation des performances

Pour évaluer les performances des codes convolutifs sans passer par les longs temps d'attente pour chaque réalisation, la méthode des impulsions a été codée dans le fichier impulsion.m mais n'est pas assez concluante pour se servir de lui afin de comparer nos TEP.

6 Conclusion

Ce projet nous a permis de mettre en pratique le codage par codes convolutifs ainsi que son décodage avec l'algorithme de Viterbi. Nous avons pu aborder les notions de diagramme d'états ou de trellis de manière concrète. Nous avons constaté les avantages en précision de chacun de ces codes mais aussi leur contrainte temporelle. En effet, les chaînes codées par des codes avec des mémoires importantes sont plus fiables et génèrent moins d'erreurs mais ont un débit plus faible nécessitant plus de temps pour transmettre l'information. C'est à partir de ces informations que l'on peut conclure que le choix d'encodage relève d'un compromis entre vitesse et précision (à vouloir rouler trop vite on perd les pédales).