Projet de Telecommunications

Sakun Withanage et Amélie Rouge

Janvier 2025

Contents

1 Introduction

Ce projet a pour objectif d'étudier l'impact du codage canal, également connu sous le nom de codage correcteur d'erreurs, dans une chaîne de communication numérique. Pour ce faire, nous implémenterons la couche physique d'une transmission satellite selon le standard DVB-S (Digital Video Broadcasting - Satellite). Cette couche physique repose sur une concaténation de deux types de codes : un code convolutif et un code bloc. L'étude se concentrera sur une transmission fixe, modélisée par un canal à bruit additif blanc gaussien (AWGN).

2 Implantation du modulateur/démodulateur

Nous allons implémenter une modulation QPSK avec un filtre mise en forme en racine de cosinus surrélévés avec un roll-off de 0.35.

2.1 Chaîne de transmission

La chaîne de transmission sur porteuse associée à une modulation de type QPSK est :

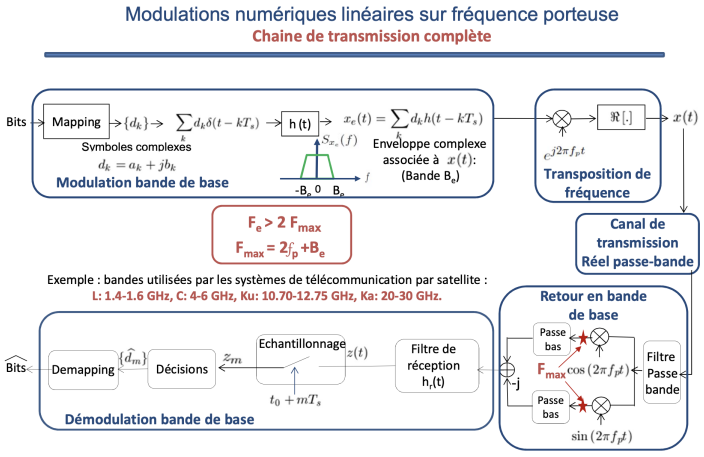

Figure 1: Chaîne de transmission sur porteuse QPSK (cours de 1A)

Cependant, le projet a été réalisé avec la chaîne passe bas équivalent :

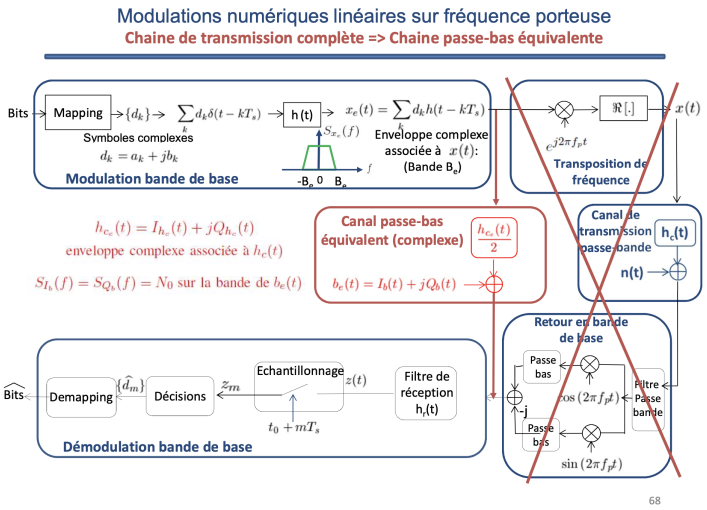

Figure 2: Chaîne passe bas équivalent (cours de 1A)

En utilisant directement la chaîne de transmission passe-bas équivalente, on élimine la nécessité de manipuler le signal à une porteuse de fréquence élevée, ce qui renforce la robustesse face aux distorsions.

2.2 facteur de surréchontillonage

Le critère de Nyquist impose que : $F_e \geq 2B$.
Avec B la largeur de bande et Fe la fréquence d'échontillonage.

Or, $B = (1+\alpha) \cdot \frac{R_s}{2}$
Avec Rs le débit symboles.

Donc $F_e \geq (1+\alpha) \cdot R_s$ (1)

On a aussi que le facteur de surréchontillonage se définit comme le rapport entre la fréquence d'échontillonage er le débit symbole :

$N_s = \frac{F_e}{R_s}$ (2)

Ainsi en combinant (1) et (2) nous obtenons :

$N_s \geq (1+\alpha)$

Ainsi la valeure minimale de Ns est de 2. Cependnt pour la suite du projet

nous fixerons Ns = 5 afin d'avoir une meilleure visualisation des signaux .

2.3 Efficacité spectrale

L'efficacité spectrale mesure la quantité d'information que le système peut transmettre par unité de bande passante. Elle quantifie la capacité du système à utiliser efficacement le spectre de fréquences disponible.

Elle est définie par :

$\eta = \frac{R_b(bits/s)}{B(Hz)}$
Avec Rb le débit binaire et B la largeur de bande.

Or, pour un ordre de modulation M :
$R_b = R_s \cdot \log_2(M)$ et $B = R_s \cdot (1 + \alpha)$

En combinant les équations précédentes :

$\eta = \frac{R_s \cdot \log_2(M)}{R_s \cdot (1+\alpha)} = \frac{\log_2(M)}{1+\alpha}$

A.N :

$\alpha = 0.35$ et $M = 4$ (car QPSK)

$\eta = \frac{2}{1.35} = 1,48$ bits/s/Hz

2.4 Implémentation

Dans cette partie nous allons voir les différentes choix techniques mis en place afin d'obtenir un taux d'erreur binaire nul (sans canal).

2.4.1 Mapping

Pour respecter la norme DVB-S, nous devons obtenir la constellation suivante :

La constellation de la figure 1 correspond à ce qu'on obtien par pskmod avec un déphasage de $\frac{\pi}{4}$. Ainsi, le mapping suivant permet d'avoir la constellation de la fugure 1:

```
symboles=(1-2*bits(1:2:end))+1j*(1-2*bits(2:2:end));
```

2.4.2 Choix du SPAN

Le paramètre SPAN correspond à la durée totale du filtre, en racine de cosinus surélevée, en nombre de symboles. Nous avons fait le choix de SPAN=6. Nous voyons sur la figure suivante qui 6 apartien bien à l'intervalle de confiance (en rouge)

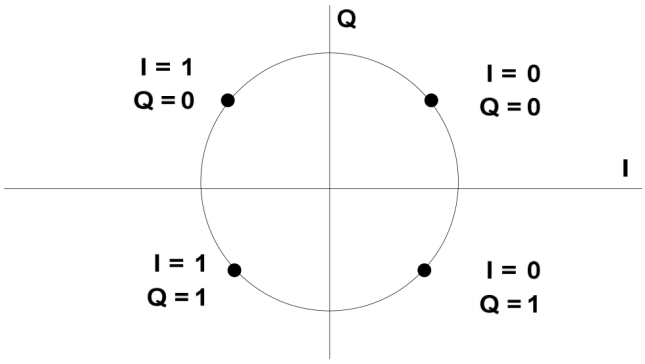

Figure 3: Constellation QPSK respectant la norme DVB-S

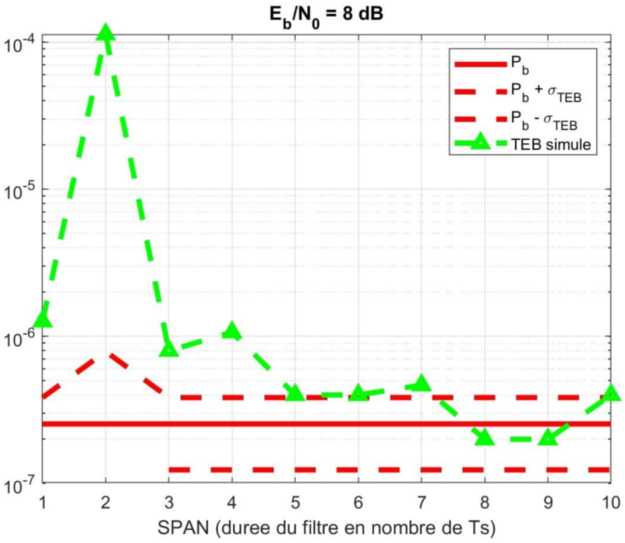

Figure 4: Ecart type SPAN

2.4.3 Retards introduits par le filtre

Un filtre numérique introduit un retard de longueur la moitié de sa longueur. Ainsi avec le filtre de réception et d'émission le retard totale introduit est égale à la longueur du filtre soit SPAN * Ns qui fait que les premiers bits ne peuvent pas être utilisés ainsi on ajoute des 0 à lors de la mise en forme pour pouvoir obtenir les informations sur les derniers bits codés de la façon suivante :

```
modulation= [kron(symboles, [1, zeros(1, Ns-1)]) zeros(1,retard)]
```

De même lors de la récéption nous compensant le retard introduit de cette manière :

```
signal_recu=filter(h_m1, 1, signal_bruite);
demodulation = signal_recu(retard+1:Ns:end);
```

2.5 Implémentation avec le canal AWGN

Dans cette partie nous allons ajouter à la chaîne précédement mis en place, le canal AWGN (bruit additif blanc gaussien).

Le bruit blanc gaussien a été calculé avec :

$$\sigma_{n_I}^2 = \sigma_{n_Q}^2 = N_0 F_e = \frac{E_s}{\frac{E_s}{N_0}} F_e = \frac{P_x T_s}{\frac{E_s}{N_0}} F_e = \frac{P_x N_s}{2\log_2(M)\frac{E_b}{N_0}},$$

Nous allons ici observer le TEB obtenu par passage dans le canal bruité

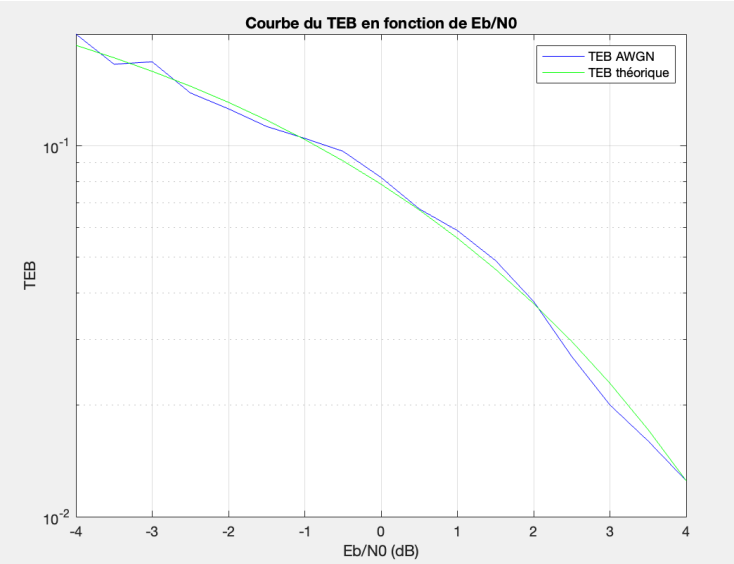

Figure 5: Simulation TEB

Les courbes de la figure 5 mettent en évidence une concordance satisfaisante entre les TEB simulés et théoriques, confirmant ainsi la validité de l'implémentation de la chaîne de transmission avec le canal AWGN.

3 Ajout du codage canal

Dans cette partie nous allons ajouter à la chaîne précédement implémenté un code Reed-Solomon RS(204,188) suivi d'un code convolutif (7,1/2), pouvant être poinçonné pour obtenir différents taux de codage (1/2, 2/3, 3/4, 5/6 et 7/8). Pour optimiser les performances, un entrelaceur convolutif est ajouté entre les deux codes.

3.1 Introduction du code convolutif

Voici les résultats obtenus en ajoutant le code convolutif (décodage hard et soft) :

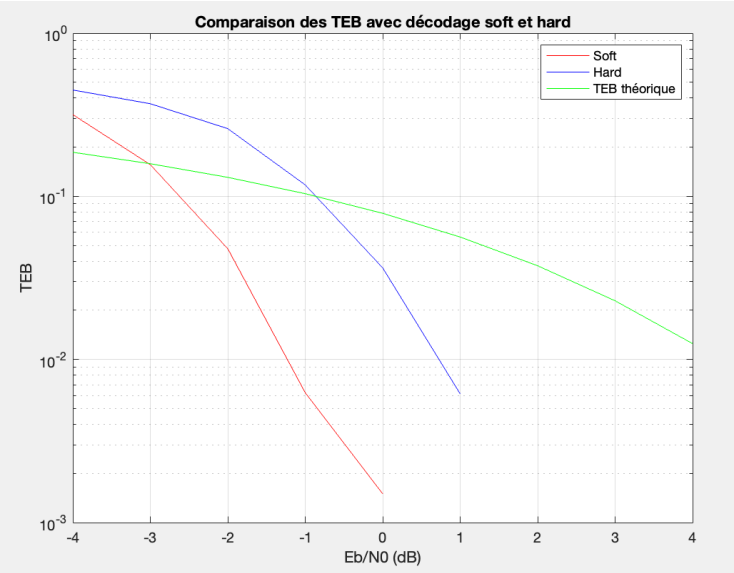

Figure 6: TEB code convolutif

Nous observons, tout d'abord que les résultats se sont amélioré en ajoutant du codage. Nous observons aussi que le decodage soft est plus performant que le décodage hard. Ceci est bien le résultat attendu, ceci s'explique par le fonctionnement du décodage hard et soft.

En effet, pour le décodage soft, l'algorithme de viterbi utilise la distance euclédien pour faire le décodage. Alors que pour le décodage hard c'est la distance de Hamming qui est utilisée. Le décodage par distance euclédienne est plus performant car il est plus précis (ce n'est pas une décision binaire qui est donnée à l'algorithme de viterbi mais une valeur entre -1 et 1). on peut le voir ici :

Décodage soft :

```
demodulation_soft_r=real(demodulation);
demodulation_soft_i=imag(demodulation);
demodulation_soft=zeros(1,2*length(demodulation));
demodulation_soft(1:2:end)=demodulation_soft_r;
demodulation_soft(2:2:end)=demodulation_soft_i;
decoded_soft = vitdec(demodulation_soft, trellis, 5*k, 'trunc', 'unquant');
```

De plus, nous observons que dans la fonction vitdec qui permet d'appliquer l'algorithme de viterbi, il y a le paramètre "unquant" qui signifie que les données en entrée de l'algorithme sont non quantifiés.

Décodage hard :

```
bits_recu_hard=demodulation_soft<0;
bits_recu_hard = reshape(bits_recu_hard, 1, []);
decoded_hard = vitdec(bits_recu_hard, trellis, 5*k, 'trunc', 'hard');
```

Ici on observe qu'on fait une décision binaire par rapport au seuil 0. Ceci explique le paremètre "quant" dans le vitdec qui signifie que les données sont quantifiées.

3.1.1 Poinçonnage

Ici nous alons ajouter la matrice de poinçonnage P = [1101]. Voici le résultat obtenu :

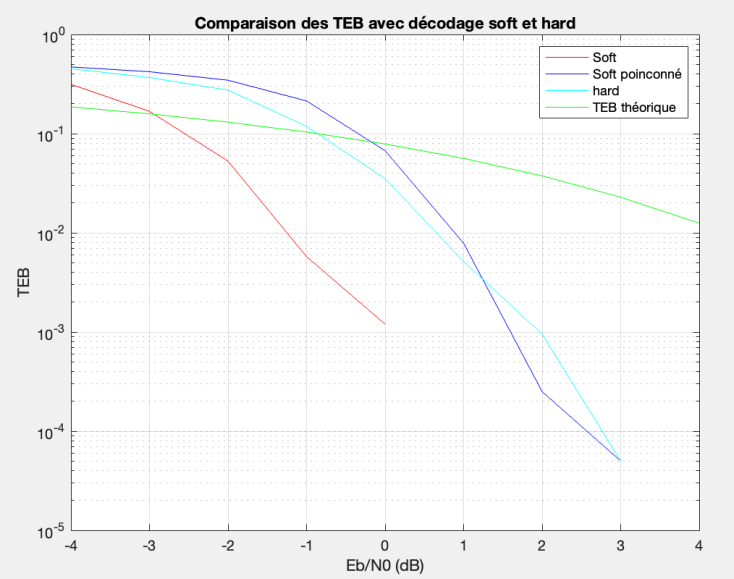

Figure 7: TEB code convolutif et poinçonnage

Nous observons qu'en ajoutant la matrice de poinçonnage, nous perdons en terme de TEB. En effet, le poinçonnage consiste à enlever des bits avant l'émission (les bits correspondant aux 0 dans la matrice). Cela permet de réduire la consommation de la bande passante car en ajoutant la matrice de poinçonnage on fait passer le taux de codage de $\frac{1}{n}$ (code convolutive) à $\frac{n-1}{n}$. A la reception les bits supprimés sont remplacés par des 0, cela explique le TEB élévée.

On remarque aussi que l'on a utilisé un décodage soft pour le poinçonnage. Ceci s'explique par le fait qu'on n'utilise pas le décodage hard en cas de poinçonnage,

car il perd les informations de confiance associées aux bits, ce qui rend le système moins robuste et augmentant le taux d'erreur binaire dans un code déjà affaibli par la réduction de redondance.

3.2 Introduction du code bloc de Reed Solomon

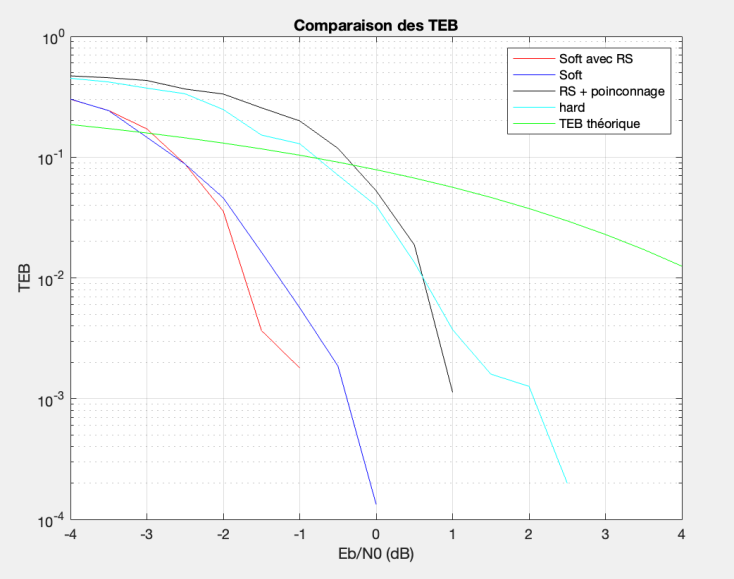

Figure 8: TEB code Reed Solomon (en rouge : Conv + RS)

Nous observons que le RS (sans poinçonnage) est sans surprise le plus performant (en terme de TEB). On observe néanmoins que RS a du mal à corriger à Eb/N0 faible (donc bruit élevée). En effet, le code a une capacité de correction de $\frac{204-188}{2} = 8$ symboles . Ainsi, le code RS(204,188) peut corriger jusqu'à 8 symboles erronés dans un mot de code de longueur 204 symboles et en cas de bruit élevée le nombre d'erreurs peut dépasser 8. Puis on voit que lors que Eb/N0 augmente, le TEB diminue. De plus, l'intêret de concaténer les deux codes convolutif et par bloc est de pourvoir rendre la chaîne plus robuste et réduire le TEB car le code Reed Salomon est capable de corriger les erreurs par blocs et le code convolutif est capable de corriger les erreurs aléatoires.

3.3 Entrelaceur convolutif

Le rôle de l'entrelaceur est de dispercer les erreurs en cas de rafales d'érreurs pour faire en sorte que les codes convolutifs et par blocs puissent les corriger plus facilement. En effet, à la partie précédente nous avons vu que les codes RS peuvent avoir leurs limites en terme de capcité de correction et que les codes convolutifs sont plus adaptés pour corriger les erreurs aléatoires. Le fait de les

dispercer permet d'augmenter la probabilité de correction d'une erreur ce qui permet de diminuer le TEB.

3.3.1 Implémentation de l'entrelaceur

Nous obtenons un TEB constant pour cette partie. Le résultat est bien évidemnt faussé mais nous n'avons pas réussi à découvrir pourquoi. Nous avons bien pris en compte le retard caussé par l'entrelacement : //

```
oct = bitToOctet(bits_rs);
retard_ent = zeros(1,12*17*(12-1));
mat_ent = [oct'  retard_ent];
entrelace = convintrlv(mat_ent,12,17);
bits_ent= OctetTobit(entrelace);
```

Nous avons besoin de convertir les bits en octets avant de donner à la fonction convintrlv puis nous devons considérer le retard caussé par l'entrelaceur et le desentrelaceur qui est de nrows * slope * (nrows -1). A la suite de cela les opérations habituelles continuent jsuqu'au moment de desentrelacement où on enlève ce retard.

```
decode = bitToOctet(decode_conv);
desen = convdeintrlv(decode, 12, 17);
retard_enleve = desen(1,12*17*(12-1)+1:end);
bits_recu = OctetTobit(retard_enleve);
```

Malgré cela notre résultat n'est pas cohérant.

4 Bibliographie

MathWorks : https://fr.mathworks.com/help/comm/ref/convintrlv.html
Cours de codage canal (Marie-Laure Boucheret)
Cours de Telecommunications 1A SN (Nathalie Thomas)

Remerciements :

Mme. Nathalie Thomas
M. Benoît Escrig
Mme. Marie-Laure Boucheret