

# Rapport projet Transmissions numériques

MEGUIRA Levi  
CLAYBROUGH Jonathan

17 janvier 2018

## Contexte

Ce projet de communication numérique vise à simuler sous Matlab une chaîne de transmissions au standard DVB-S. Il se divise en trois grandes parties que sont :

1. La création d'un couple modulateur/ démodulateur.
2. Le codage canal avec code RS, code convolutif poinçonnage et entrelacement.
3. La synchronisation avec une structure en boucle fermée pour corriger les erreurs de phases.

## Table des matières

<b>1</b>	<b>Organisation du travail</b>	<b>1</b>
<b>2</b>	<b>Structure du code</b>	<b>1</b>
<b>3</b>	<b>Chaine de base</b>	<b>1</b>
<b>4</b>	<b>Modulation-démodulation</b>	<b>1</b>
<b>5</b>	<b>Bruit canal</b>	<b>2</b>
<b>6</b>	<b>Codage canal</b>	<b>2</b>
6.1	Code de Reed Solomon . . . . .	2
6.2	Entrelacement . . . . .	3
6.3	Code convolutif et poinçonnage . . . . .	3
<b>7</b>	<b>Retard de phase</b>	<b>3</b>
<b>8</b>	<b>Correction du retard de phase</b>	<b>4</b>
<b>9</b>	<b>Conclusion</b>	<b>4</b>
<b>10</b>	<b>conclusion</b>	<b>5</b>

## 1 Organisation du travail

Nous étions un groupe de deux, alors plutôt que de travailler deux fois les mêmes choses, nous nous sommes répartis les tâches à faire puisqu'elles étaient relativement indépendantes, puis avons mis en commun pour former la chaîne entière.

Cette répartition fut possible grâce à la structure modulaire du projet. Toutes les opérations ont été effectuées via la plateforme Github, avec un dépôt visible à l'adresse suivante <https://github.com/LeviMe/ProjetTransmissionsNumeriques>.

## 2 Structure du code

## 3 Chaîne de base

## 4 Modulation-démodulation

L'implémentation de cette première partie se fait avec une chaîne passe-bas équivalente à la chaîne de transmission sur porteuse du cas simulé. Elle envoie un signal complexe plutôt que deux signaux réels en quadrature, et ce de la façon suivante :

1. Des bits d'informations sont modulés en QPSK, c'est à dire sous la forme de symboles complexes  $c_k = a_k + jb_k$  ou  $a_k, b_k \in \{-1, +1\}^2$
2. Ces symboles sont passés par un filtre de mise en forme en racine de cosinus surélevé et envoyés sur un canal.
3. Le canal ajoute un bruit gaussien d'une densité spectrale de puissance spécifiée.
4. Le signal atteint le récepteur ou il est passé par un filtre adapté et échantonné aux instants optimaux.
5. Enfin il est divisé en ses parties réelles et imaginaires puis passé par un détecteur à seuil pour prendre les décisions et retrouver les symboles QPSK d'émissions.

## 5 Bruit canal

Le canal est supposé créant du bruit blanc gaussien. La fonction canal ajoute donc un bruit blanc gaussien complexe au signal complexe reçu avec le dB souhaité.

Pour calculer la puissance du bruit :

$\sigma = 1$  (variance des symboles) On a pris une porte normée donc puissance de signal à 1 aussi.  
 $E_s = 2 \times E_b$  donc  $\sigma^2 = \frac{1}{4 \cdot E_b/N_0}$  et

$$E_b/N_0_{dB} = 10 \log(E_b/N_0)_{4 \cdot E_b/N_0} \text{ d'où } E_b/n_0 = 10^{(dB/10)}$$

```
function [ signalBruite]= canal( dB, signal)
Eb_N0 = 10^(dB/10);
sigma = 1 / (2 * 10 ^ (Eb_N0 / 10));
bruit_I = sqrt(sigma) * randn(1, length(signal));
bruit_Q = sqrt(sigma) * randn(1, length(signal));
bruit = bruit_I + 1j * bruit_Q;
signalBruite = signal + bruit;
end
```

## 6 Codage canal

Le codage canal se décompose en plusieurs parties que sont dans l'ordre :

1. Un code de Reed-Solomon réduit (204,188).
2. Un entrelaceur conforme au standard DVB-S c'est à dire de *paramètres* 17 et 12.
3. Un code convolutif de polynômes générateur 171, 133, qui est poinçonné via la matrice  $[1, 1, 0, 1]$ .

Toutes ces fonctions sont implémentés dans une seule est même fonction **codage** qui prend un vecteur de **bits** et tous les interrupteurs relatifs aux étapes de son fonctionnement et renvoie le vecteur codé.

Le fonctionnement par étapes consiste à initialiser un **bits\_codes** à **bits** et à lui faire subir toutes

les opérations de codages suivant la configuration d'interrupteurs choisie. C'est lui qui sera renvoyé en sortie de la fonction.

```
function [bits_codes] = codage(bits, RS_encoding, interleaving,
    puncturing, convencoding)
```

## 6.1 Code de Reed Solomon

Le code de Reed-Solomon est par essence non-binaire; il prend en entrée des symboles qui dans notre cas seront des octets. Pour effectuer la conversion à partir de valeurs binaires, on va utiliser les fonctions `reshape` puis `bi2de` et obtenir ainsi le vecteur `symboles` huit fois plus courts.

Ce vecteur sera encodé pour donner un vecteur `symboles_codes` puis à son tour converti en valeurs binaires via une opération analogue à la première.

```
%comment
if RS_encoding
    symboles=bi2de(reshape(bits,[],8));
    symboles_codes = step(enc,symboles);
    bits_codes = reshape(de2bi(symboles_codes),[],1);
end
```

## 6.2 Entrelacement

L'entrelacement au standard DVB-S fonctionne avec deux paramètres appelés `nrows` et `slope` qui signifient ——— et dont la valeur est fixée respectivement à 17 et 12.

De ces paramètres dépend la valeur du padding à introduire pour obtenir une transmission sans erreur notée D.

```
nrows = 17; slope = 12;
D = nrows*(nrows-1)*slope;
...
if interleaving
    bits_codes = convintrlv([bits_codes;zeros(D,1)], nrows,slope);
end
```

## 6.3 Code convolutif et poinçonnage

Le code convolutif requis par l'énoncé de paramètres  $(7, 1/2)$  et de polynômes générateurs  $g_1 = 171_{oct}$  et  $g_2 = 133_{oct}$ .

La syntaxe de son implémentation Matlab se fait via la création d'un objet d'encodage `trellis` qui peut accueillir ou non une matrice de poinçonnage. C'est ce qui est fait dans le code suivant où les deux configurations sont envisagées.

On aurait également pu procéder en l'absence de poinçonnage avec une matrice  $[1, 1, 1, 1]$ , c'est à dire celle qui conserve le signal de départ.

```

puncturing_matrix = [1 1 0 1];
trellis = poly2trellis([7],[171 133]);
-----
if puncturing
    bits_codes = convenc(bits_codes,trellis,puncturing_matrix);
else
if convencoding
    bits_codes = convenc(bits_codes,trellis);
end
end

```

## 7 Retard de phase

Le retard de phase est relativement trivial à implémenter, puisqu'il suffit de prendre le signal complexe et de le multiplier par  $e^{j\phi}$ .

```

function [ symb_d ] = Dephasage( phi, ecartFreq, symb )
d_phi = phi.*ones(1,length(symb)) + [1:length(symb)].*2.*pi.*
    ecartFreq;
symb_d = symb.*exp(j*d_phi);
end

```

## 8 Correction du retard de phase

```
function [ symb_corrige ] = PLL(A, B, symb)
%PLL Detection et correction du dephasage
%  Detection et correction du dephasage du signal du a la
%  difference de
%  frequence des horloges a l'emission et la reception

symb_corrige = zeros(1, length(symb));
phi_est = zeros(1, length(symb)+1);
out_det = zeros(1, length(symb));
w = zeros(1, length(symb));
NCO_mem=0;      % initialisation NCO
filtre_mem=0;   % initialisation de la memoire du filtre

%PLL
for i=1:length(symb)
symb_corrige(i) = symb(i)*exp(-j*phi_est(i));
out_det(i)= -imag(symb_corrige(i).^4);

% filtre de boucle
w(i)=filtre_mem+out_det(i); % memoire filtre + sortie detecteur
filtre_mem=w(i);
out_filtre=A*out_det(i)+B*w(i); % sortie du filtre a l'
    instant i :  $F(z)=A+B/(1-z^{-1})$ 

%NCO
phi_est(i+1)=(out_filtre+NCO_mem); %  $N(z)=1/(z-1)$ 
NCO_mem=phi_est(i+1);
end
end
```

## 9 Conclusion

Nous avons pu implémenter l'entièreté de la chaîne de transmission et vérifier un certain nombre de résultats attendus.

%raw Data

taux d erreurs sans codage=

0.0546	0.0504	0.0429	0.0368	0.0322
--------	--------	--------	--------	--------

taux d erreurs avec codage RS et interleaving =

0.0548	0.0542	0.0436	0.0401	0.0288
--------	--------	--------	--------	--------

taux d erreurs avec codage RS sans interleaving=

0.0571	0.0556	0.0442	0.0395	0.0319
--------	--------	--------	--------	--------

taux d erreurs scode =

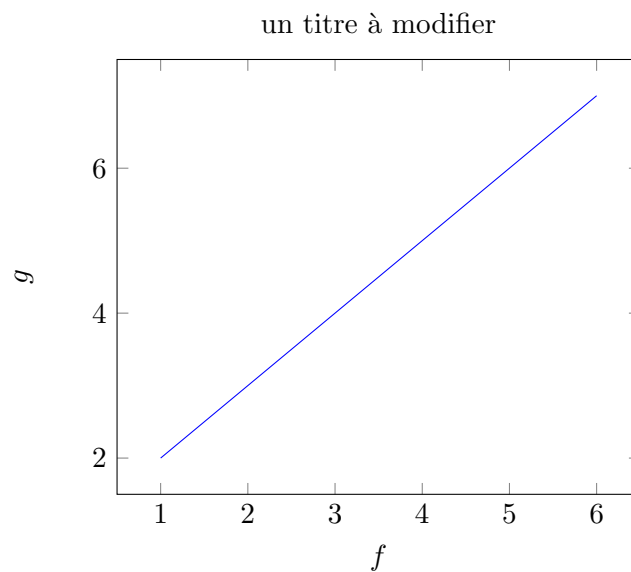
0.0564	0.0538	0.0427	0.0375	0.0274
--------	--------	--------	--------	--------

taux d erreurs avec RS, interleaving, convencoding =

0.0529	0.0426	0.0370	0.0196	0.0147
--------	--------	--------	--------	--------

taux d erreurs =

0.2094	0.1934	0.1561	0.1106	0.0649
--------	--------	--------	--------	--------



## 10 conclusion