Simulation d'une chaîne de transmission numérique avec Matlab

David Nicaise

3 mai 2016

Table des matières

Première partie

Introduction

Chapitre 1

Paramètres

Avant de commencer à décrire notre chaîne, attardons-nous quelques instants sur les paramètres que nous utiliserons dans la suite de l'exposé :

N : Nombre de ressources
M : Taille des messages
R : Débit binaire [bits/sec]
type : Type de récepteur
beta : Facteur de suréchantillonnage pour filtre d'émission
gamma : Facteur de suréchantillonnage pour DAC
alpha : Facteur de rolloff pour filtre d'émission
L : Vecteur de troncage pour filtre d'émission
P_t : Puissance transmise sur le cable [Watts]
Z_c : Impédance du cable [Ω]
EbN0_DB : Rapport entre énergie du bit et DSP de l'AWGN [dB]
EbN0 : Même que précédent, mais sans le facteur logarithmique
alpha_n : Facteur d'atténuation pour canal
ordre_filtre : Ordre du polynôme à générer pour filtre analogique

Deuxième partie

Émetteur

Chapitre 1

Message

1.1 Génération

Une séquence de synchronisation doit être générée pour le récepteur à cadence R ($\frac{1}{T\_b}$). Or la longueur de cette séquence M_s est dépendante de la longueur du message à envoyer (pour éviter que le message ne contienne une séquence similaire qui fausse la synchronisation). Selon les essais, une séquence d'un dixième de la longueur du message (M) est suffisante pour une synchronisation satisfaisante.

Le message est simplement généré en utilisant la fonction rand qui renvoie des chiffres compris dans l'intevalle (0,1). Ensuite ces chiffres sont arrondis avec la fonction round. N messages sont générés pour les N ressources.

$$\text{M\_s} = \text{round}(\text{rand}(\text{round}(\text{M}/10),\text{N}))\,;$$

Séquence de synchronisation

Le message est créé de la même manière que le message de synchronisation, en fonction de la longueur M définie dans param.m.

$$\text{M\_d} = \text{round}(\text{rand}(\text{M},\text{N}))\,;$$

Génération du message

Les deux séquences sont concaténées.

$$\text{M\_t} = [\text{M\_s}\,;\text{M\_d}]\,;$$

Message à envoyer

1.2 Modulation PAM

Une modulation PAM binaire a été choisie pour ce projet, nous transformons les bits 1 et 0 en symbôles +1 et -1 (d=1).

M_t_PAM = (M_t*2)-1 ;

Un simple calcul permet de passer des bits aux symbôles

1.3 Suréchantillonage

Comme expliqué au chapitre suivant, le filtre d'émission utilisé est suréchantillonné d'un facteur β par rapport à notre débit binaire R. Ainsi notre message devra être cadencé au même rythme que le filtre, nous devons donc suréchantillonner le message. Pour cela nous utilisons une fonction Matlab prédéfinie, qui ajoute $\beta - 1$ zéros entre les échantillons de notre message, si on suréchantillonne d'un facteur β

M_t_PAM_oversampled = upsample(M_t_PAM,beta) ;

Version suréchantillonnée du message

Chapitre 2

Génération du filtre d'émission

Un fichier séparé emissionfiltre.m a été créé dans le but de choisir le filtre d'émission en fonction du type de chaîne que nous utilisons.

Un filtre de Nyquist est utilisé, de manière à limiter l'interférence entre symbôles. Plus précisement, un filtre de Nyquist en cosinus surélévé (avec un facteur de rolloff α). En fonction du type de chaîne, le filtre est soit en racine carrée (récepteur idéal, filtre adapté) ou pas (récepteur simple, pas de filtre adapté).

Dans le cas du récepteur idéal, la solution de compromis entre filtre adapté et critère de Nyquist est l'égalisation linéaire MMSE, dans ce cas nous implémentons un filtre FIR (filtre à réponse impulsionnelle finie) que nous suréchantillonnons d'un facteur β. La réponse impulsionnelle est tronquée via un facteur L, qui contient simplement le nombre de périodes T_b à garder de part et d'autre de l'axe des ordonnées.

rcosFIRsqrt = rcosfir(alpha,L,beta,T_b, 'sqrt')' ;
Filtre de Nyquist en cosinus surélevé, en racine carrée

Dans le cas du récepteur simple, le même filtre est utilisé, sauf qu'il est convolué avec lui même pour obtenir un vrai sinus cardinal (où chaque période T_b passe bien par zéro).

rcosFIRsqrt = rcosfir(alpha,L,beta,T_b, 'sqrt')' ;
filtre = conv(rcosFIRsqrt,rcosFIRsqrt) ;
Filtre de Nyquist en cosinus surélevé

Ce filtre de Nyquist est dupliqué de manière à traiter N ressources :

$$\text{filtre\_n} = \text{repmat(filtre,1,N)}\,;$$
Matrice à N filtres

Pour que l'on puisse traiter nos N ressources sur le même canal, nous de définir des bandes de fréquences pour la n$^{\text{ième}}$ ressource selon :

$$\begin{aligned} p_0(t) &= g(t) \\ p_{\text{n}}(t) &= g(t)cos(\omega_{\text{n}}t) \qquad n \in [1...N-1] \end{aligned}$$

Déplacement des filtres d'émission dans des bandes de fréquences

Pour ceci, nous définissons un vecteur fréquence radiale qui déplacera la n$^{\text{ième}}$ ressource à la fréquence voulue. Cette fréquence de déplacement devra être supérieure à 2 fois la bande passante de notre ressource en bande de base ($\frac{2}{T\_b}$) pour éviter tout chevauchement.

$$\text{omega\_n} = 2\text{*pi*2(0 :N-1)/T\_b}\,;$$
Vecteur de fréquences radiales

Un vecteur temps est défini, qui fait la même longueur que notre filtre d'émission, et va par pas de $T\_n = \frac{T\_b}{\beta}$:

$$\text{time=(L(1)*beta*T\_n :T\_n :L(2)*beta*T\_n)'}\,;$$
Vecteur temps[1]

Le n$^{\text{ième}}$ filtre d'émission sera multiplié point à point avec un cosinus, ce qui permettra de le déplacer à la fréquence voulue.

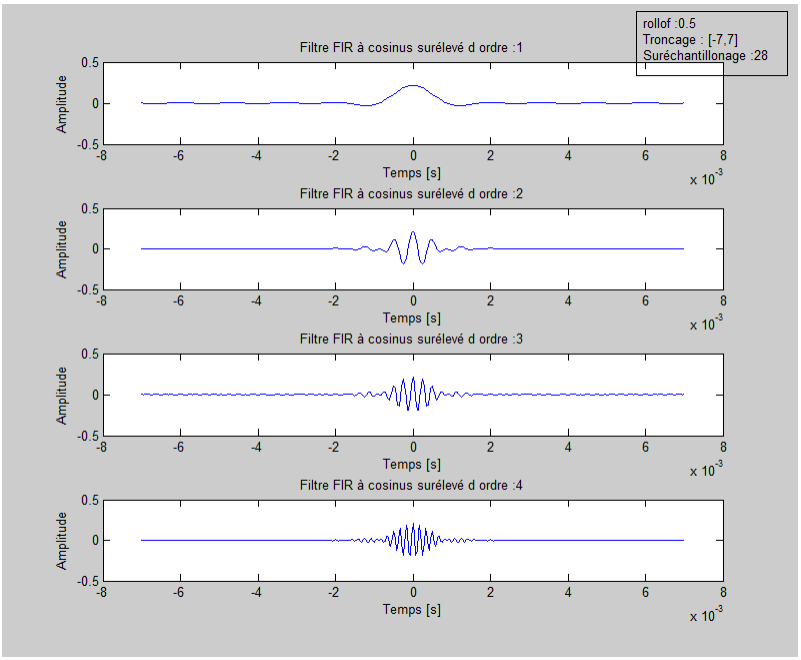
$$\text{cosine\_translate=}$$
$$\text{cos(repmat(omega\_n,size(time,1),1).*repmat(time,1,N))}\,;$$
Matrice cosinus

$$\text{p\_n} = \text{cosine\_translate.*filtre\_n}\,;$$
Filtre d'émission $p_n(t)$

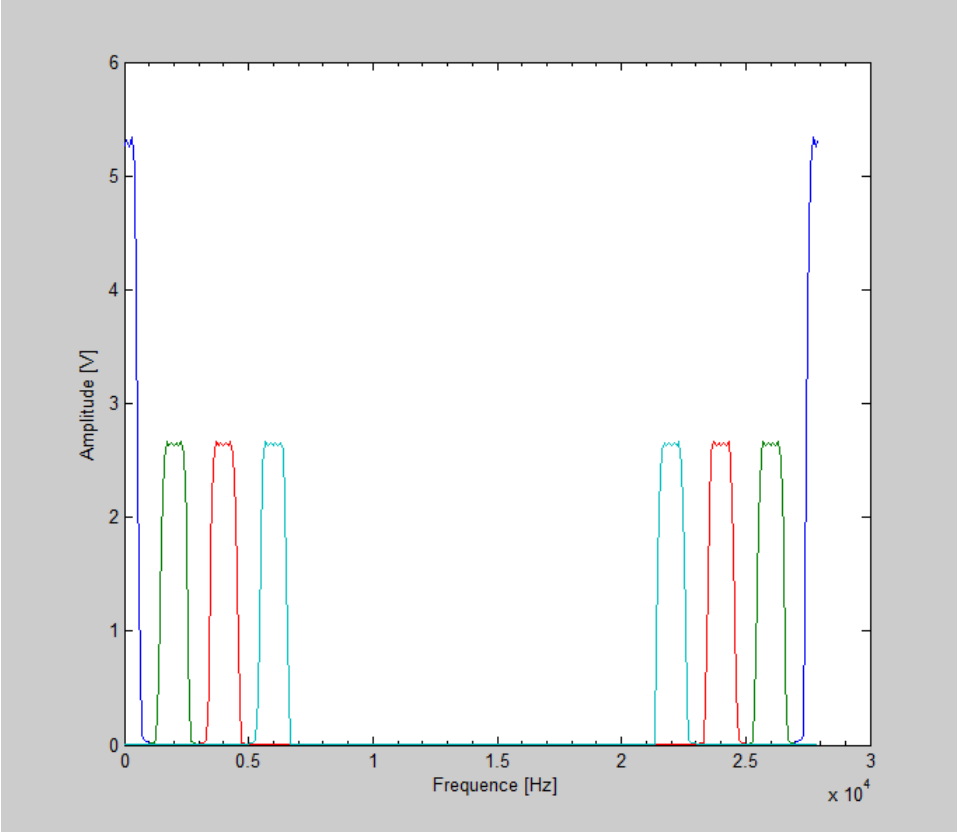
Enfin, notons que notre facteur de suréchantillonnage β devra valoir au moins le double de la fréquence la plus haute de notre spectre, pour éviter tout repli spectral.

$$\text{beta} = 2\text{*(4*N-2)}\,;$$
On prend $\beta = 4 \times f_{max}$

1. Celui-ci est défini pour le récepteur idéal, pour le filtre de Nyquist complet, il aura une taille double, vu la convolution



Filtres d'émission en racine de Nyquist, en temporel



Filtres d'émission en racine de Nyquist, en fréquentiel

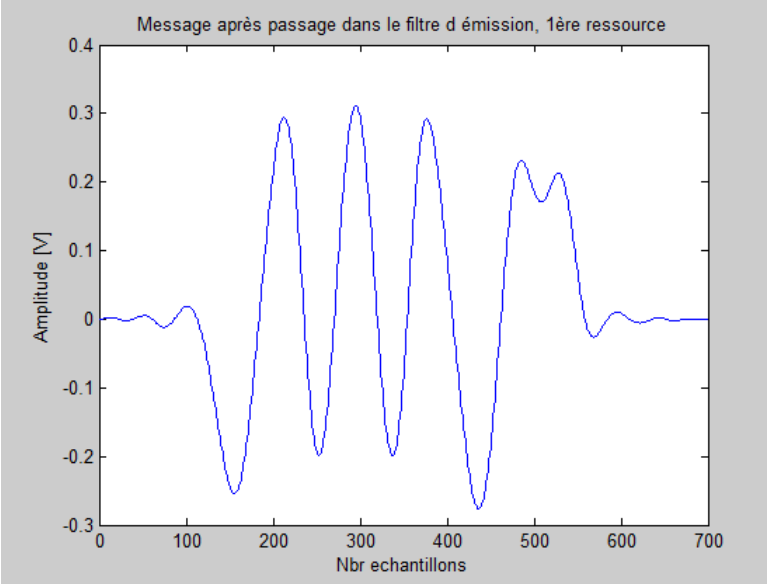
Chapitre 3

Passage dans le filtre d'émission

Notre message se trouve convolué avec la réponse impulsionnelle du filtre d'émission.

```
for d = 1 :N
      s_n( :,d) = conv(p_n( :,d),M_t_PAM_oversampled( :,d)) ;
end ;
```

Convolution



Message après passage dans filtre d'émission, length(M_t)= 15

Chapitre 4

Mise en forme

4.1 Interpolation

Nous simulons le passage à l'analogique en faisant passer notre message convolué dans un DAC.

s_n_interpol = interpft(s_n, length(s_n)*gamma) ;

Interpolation

4.2 Ajustement puissance

En stationnarisant le signal s_n_interpol (s(t) dans l'équation suivante), nous pouvons prouver que sa variance vaut :

$$s(t) = \sum_{k=-\infty}^{+\infty} a_{\mathrm{k}} p(t - kT_{\mathrm{b}} - T_0)$$

$${\sigma_{\mathrm{s}}}^2 = C_{\mathrm{UU}}(0)\frac{{\sigma_{\mathrm{a}}}^2}{T_{\mathrm{s}}}$$

Variance du signal interpolé

Or, σ_{a} vaut soit -1 ou+1. La variance ${\sigma_{\mathrm{s}}}^2$ vaut la puissance moyenne ($\mathrm{P_{av}}$) du signal. Et $\mathrm{T_s}$ vaut $\mathrm{T_b}$ puisqu'on travaille en PAM binaire. Nous pouvons donc déduire :

$$P_{\mathrm{av}} = \frac{E_{\mathrm{b}}}{T_{\mathrm{b}}} = \frac{\int_{-\infty}^{+\infty} p^2(t)\,\mathrm{d}t}{Z_{\mathrm{c}} \times T_{\mathrm{b}}} = \frac{\sum_{i=1}^{n} p^2(iT_{\mathrm{a}}) \times T_{\mathrm{a}}}{Z_{\mathrm{c}} \times T_{\mathrm{b}}} = \frac{\sum_{i=1}^{n} p^2(iT_{\mathrm{a}})}{Z_{\mathrm{c}} \times \gamma \times \beta}$$

$$A = \sqrt{\frac{P\_t}{{\sigma_{\mathrm{s}}}^2}}$$

On peut donc écrire :

```
for f=1 :N
      p_n_interpol = interpft(p_n, length(p_n)*gamma) ;
      sigma_s(1,f) = sum(p_n_interpol( :,f).^2)/beta/gamma/Z_c ;
      A(1,f) = sqrt (P_t/sigma_s(1,f)) ;
      s_n_interpol_adj( :,f) = s_n_interpol( :,f).*A(1,f) ;
end ;
```

Adaptation de la puissance du signal émis

Troisième partie

Canal

Chapitre 1

Délai

Un délai aléatoire entre 0 et T_b est simulé, pour cela, un nombre aléatoire de zéros entre 0 et length(s_n_interpol_adj/length(M_t)) sont rajoutés :

```
tau_n = round(rand(1,N)*length(s_n_interpol_adj)/length(M_t)) ;
```

Nombre de zéros aléatoire

```
r_n_delay = zeros(length(s_n_interpol_adj)+max(tau_n),N) ;
```

Nouvelle matrice pour les délais

Le délai est ensuite rajouté dans la séquence existante

```
for b = 1 :N
    r_n_delay(tau_n(b)+1 :length(s_n_interpol_adj)+tau_n(b),b) =
                                                    s_n_interpol_adj( :,b) ;
end ;
```

Rajout du délai

Chapitre 2

Atténuation

L'atténuation est réalisée en multipliant le signal par un facteur :

r_n_delay_att = r_n_delay.*alpha_n ;
L'atténuation du signal sur le canal

Chapitre 3

Bruit

Le bruit blanc Gaussien AWGN est généré à partir d'une fonction Matlab préfaite qui génère des nombres aléatoires distribués selon une loi normale, de moyenne nulle et de variance unité.

```
noise = randn(length(r_n_delay_att),N);
```

A ce bruit généré, il faut encore ajuster la variance pour qu'elle soit en adéquation avec le paramètre EbN0 demandé. Ceci consiste d'abord à convertir EbN0 exprimé en décibel, en échelle linéaire :

```
EbN0 = 10^(EbN0_DB/10);
```

Il nous faut calculer l'énergie du bit :

```
for g=1 :N
    E_b(1,g) = sum(r_n_delay_att( :,g).^2)/length(r_n_delay_att)*T_b/Z_c;
end;
```

Ensuite on calcule la DSP de l'AWGN :

```
N0 = E_b/EbN0;
E_AWGN = N0/2/T_b*2*N;
```

Nous avons calculé la puissance moyenne que doit avoir notre AWGN en fonction de la bande passante totale de nos multiples ressources. Adaptons maintenant notre bruit :

```
for c = 1 :N
    A( :,c) = sqrt(E_AWGN( :,c)*Z_c);
    AWGN( :,c) = noise( :,c).*A(1,c);
end;
```

Chapitre 4

Multiplexage

Jusqu'ici les ressources ont été traitées séparement sur le canal, dans une situation réelle, un seul et unique signal est envoyé, c'est pour cela que nous devons sommer ces ressources :

$$\begin{gathered}\text{r\_n\_delay\_att\_sum} = \text{sum}(\text{r\_n\_delay\_att}, 2)\,; \\ \text{AWGN\_sum} = \text{sum}(\text{AWGN}, 2)\,; \\ \text{r\_t} = \text{r\_n\_delay\_att\_sum} + \text{AWGN\_sum}\,;\end{gathered}$$

Les ressources sommées

Quatrième partie

Récepteur

Chapitre 1

Génération filtres analogiques

1.1 Chebyshev

Des filtres analogiques sont implémentés pour filtrer notre signal venant du canal. Pour ceci, nous décidons d'utiliser des filtres analogiques de type Chebyshev. Une fonction Matlab intégrée nous permet de récupérer les coefficients des filtres en spécifiant : l'ordre des coefficients, la (ou les) fréquence(s) radiale(s) de coupure, le ripple (ondulation en bande passante), le type (passe-bas, bande-passante).

Prenons l'exemple du filtre passe-bas :

Le filtre d'émission utilisé, en racine de Nyquist avec un cosinus surélevé, stipule une bande-passante de $\frac{1+alpha}{T\_b}$. Cette bande passante est répartie de manière égale dans les fréquences positives et négatives, nous devons donc diviser cette valeur par 2 si nous voulons obtenir la fréquence de coupure.

[filter_coeff_b(1, :),filter_coeff_a(1, :)]
=
cheby1(ordre_filtre, 0.00501,((1+alpha)/2/T_b)*2*pi, 'low', 's') ;
Génération du filtre de Chebyshev pour la 1ère ressource

Cette fonction Matlab nous donne 2 jeux de coefficients, que nous utiliserons dans le chapitre suivant pour générer une réponse en fréquence. Notons aussi la présence de 's' qui demande à Matlab de travailler dans le domaine analogique (si non spécifié, le domaine discret est utilisé par défaut).

Les autres filtres en bande-passante sont générés de manière équivalente, en prenant en compte le déplacement fréquentiel pour les ressources supérieures à 1 :

```
for k = 1 :N-1
    [filter_coeff_b(k+1, :),filter_coeff_a(k+1, :)] =
        cheby1(ordre_filtre/2, 0.0501, [(((2*k) - (1+alpha)/2)/T_b)*2*pi,
            (((2*k) + (1+alpha)/2)/T_b)*2*pi], 'bandpass', 's') ;
end ;
```

Génération des coefficients pour les ressources supérieures à 1

1.2 Réponse fréquentielle

Les coefficients que nous fournissent la fonction cheby1 doivent être transformés en réponse fréquentielle, pour pouvoir obtenir après la réponse impulsionnelle. Pour cela, nous utilisons une fonction Matlab intégrée, freqs, qui s'occupe de transformer les coefficients en réponse en fréquence. Nous devons néanmoins lui spécifier un vecteur fréquence, pour lesquelles elle calculera la réponse en fréquence :

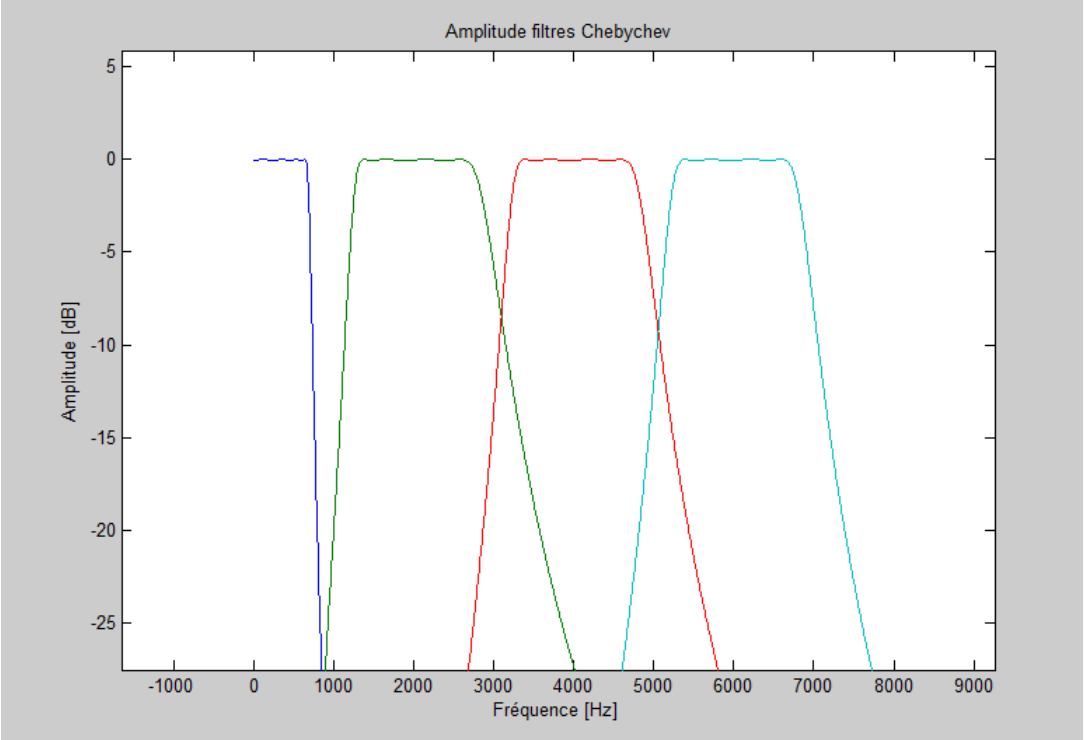
```
w = 2*pi*linspace(0, 1/2/T_a, beta*gamma*100) ;
```

Vecteur fréquence radiale de 0 à $\frac{1}{2*T\_a}$ *avec beta*gamma*100 fréquences également espacées*

Nous ne détaillerons que le calcul de la réponse en fréquence pour la 1ière ressource, la manipulation est similaire pour les N-1 restantes :

```
H( :,1) = freqs(filter_coeff_b(1, :), filter_coeff_a(1, :), w).' ;
```

Réponse fréquentielle pour la 1ière ressource



Réponse en fréquence des filtres analogiques pour N ressources

1.3 Réponse impulsionnelle

Pour calculer la réponse impulsionnelle, nous devons reconstruire le spectre complet de la réponse fréquentielle. Pour cela nous devons reproduire la partie du spectre image entre la fréquence de mi-échantillonnage et la fréquence d'échantillonnage, que freqs ne donne pas par défaut. Cette partie, de part la transformée de Fourier, est l'image retournée et conjuguée de notre réponse H.

$$\mathrm{HH} = [\mathrm{H}\,;\, \mathrm{conj}(\mathrm{H}(\mathrm{end\text{-}1}\ \mathrm{:\text{-}1}\ \mathrm{:2}\,;\,\mathrm{:}))]\,;$$

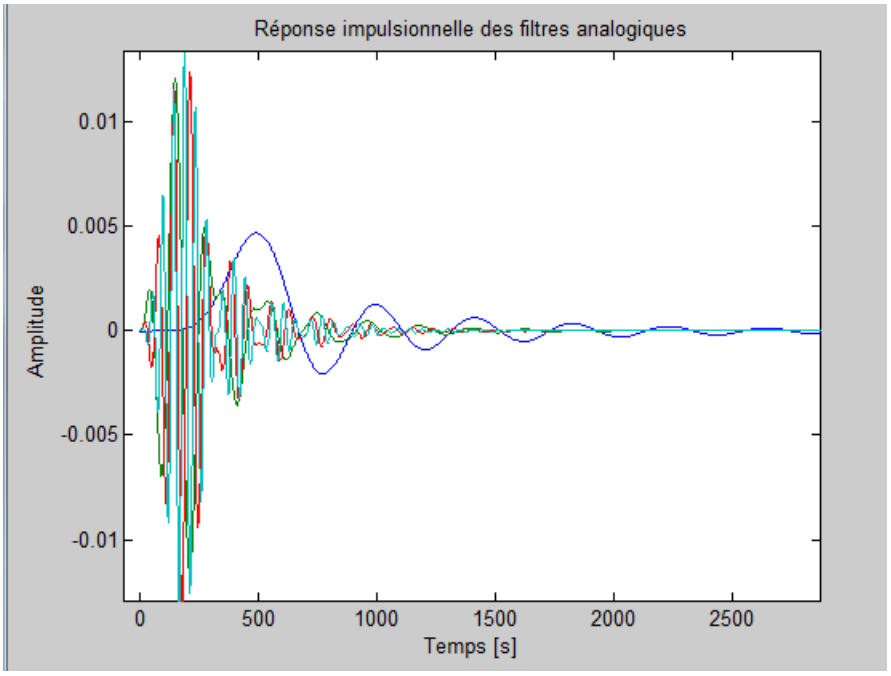
Reconstruction du spectre de la réponse fréquentielle

Nous forçons la fréquence DC à zéro (qui se retrouve à la fréquence mi-échantillonnage) :

$$\mathrm{HH}(1,\,\mathrm{:}) = 0\,;$$
$$\mathrm{HH}((\mathrm{length}(\mathrm{HH})\mathrm{+2})/2,\,\mathrm{:}) = 0\,;$$

Nous calculons enfin la réponse impulsionnelle :

$$\mathrm{h} = \mathrm{ifft}(\mathrm{HH})\,;$$



Réponse impulsionnelle des filtres analogiques

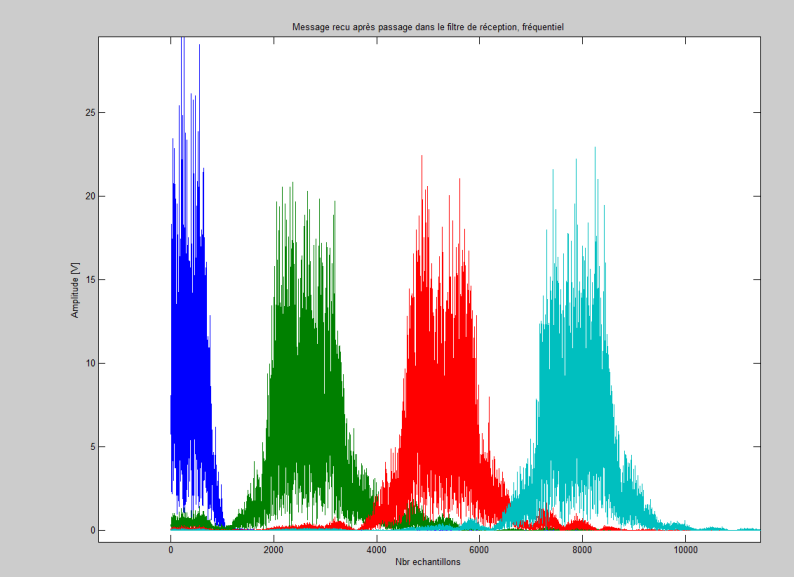
Chapitre 2

Démultiplexage

Pour pouvoir récupérer chaque ressource, que nous avons déplacées à des fréquences particulières, nous devons filtrer notre unique signal reçu sur le canal. Pour ceci nous convoluons simplement nos filtres analogiques avec le message recu r_t :

```
for j = 1 :N
        r_n( :,j) = conv (h( :,j),r_t) ;
end ;
```

Démultiplexage des N ressources



Message recu après démultiplexage

Chapitre 3

Récepteur idéal

3.1 Mise à l'échelle

Nous normalisons simplement en divisant par la plus grand valeur de r_n :

```
factor_resize = max(r_n) ;
for j = 1 :N
    r_n_resized( :,j) = r_n( :,j)./factor_resize(j) ;
end ;
```

Mise à l'échelle

3.2 ADC

Il suffit de sous-échantillonner notre signal recu pour passer l'analogique (T_a) au numérique (T_n) :

```
r_n_resized_digital = downsample(r_n_resized,gamma) ;
```

ADC

3.3 Filtre adapté

Dans le récepteur idéal, nous utilisons un filtre adapté (celui que nous utilisions pour le filtre d'émission) pour maximiser le SNR, de par l'inégalité de Schwarz :

```
for i=1 :N
    y_n( :,i) = conv(r_n_resized_digital( :,i),p_n( :,i)) ;
end ;
```

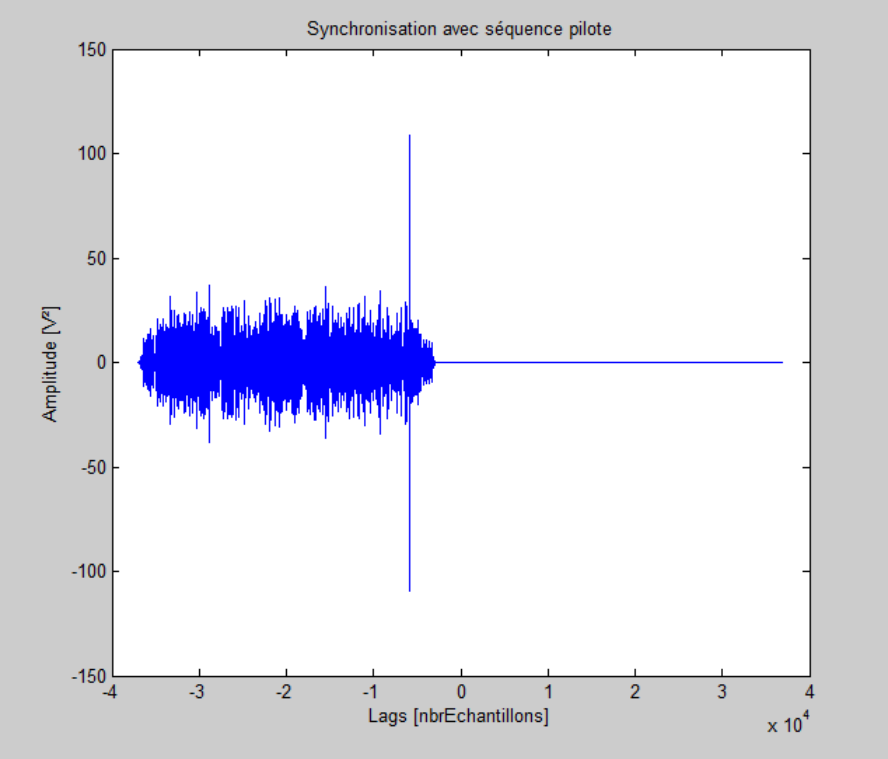
Convolution avec filtre adapté

3.4 Synchronisation et prise de décision

Pour pouvoir déterminer où notre signal commence, il faut effecturer une corrélation entre la séquence pilote (à débit égal) et le signal recu. Pour ceci nous effectuons une corrélation croisée qui décale un signal et multiplie les composantes des deux signaux entre elles, elle effectue l'opération pour tous les décalages possibles. Ainsi le maximum de cette corrélation nous renseignera l'endroit du début de la synchronisation.
Ensuite il suffira de passer les symbôles de synchronisation et collecter les symbôles (tous les beta échantillons). Ces symbôles, s'ils sont positifs, seront considérés comme un 1, et inversément s'ils sont négatifs, c'est qu'ils valent 0 :

```
M_s_PAM = M_s.*2-1 ;
M_s_oversampled = upsample(M_s_PAM,beta) ;
for i=1 :N
      [xc,lags] = xcorr(M_s_oversampled( :,i),y_n( :,i)) ;
      [max_xc, index_max_xc] = max(xc) ;
      index_debut = lags(index_max_xc) ;
      M_reconstr( :,i) =
                  y_n((abs(index_debut)+1+(length(M_s)*beta) :
                        beta :abs(index_debut)+1+(length(M_s)*beta)+
                              ((M-1)*beta)),i) ;
      M_decision( :,i) = M_reconstr( :,i) ;
      for j=1 :length(M_decision)
            if M_decision(j,i) > 1
                  M_decision(j,i) = 1 ;
            elseif M_decision(j,i) <1
                  M_decision(j,i) = 0 ;
            end ;
      end ;
end ;
```

Algorithme de synchronisation et décision



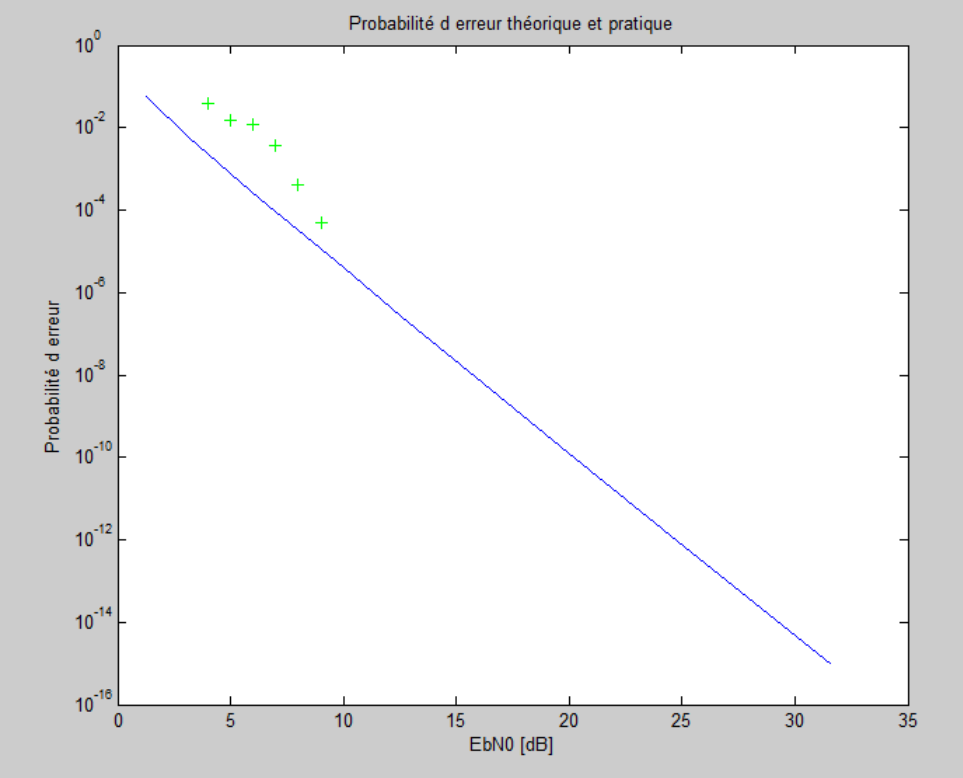
Résultat de la corrélation croisée

Pour vérifier si notre message a bien été décodé, il nous suffit d'appliquer un xor entre ce que nous recevons et ce que nous avons envoyé :

error = sum(xor(M_d,M_decision))
Afficher le nombre d'erreur par ressource

3.5 Probabilité d'erreur due au bruit

En ayant récupéré les erreurs pour un rapport EbN0 donné, nous pouvons tracer le diagramme d'erreur.



Probabilité d'erreur, en fonction de EbN0

La probabilité d'erreur théorique est dessinée en fonction de :

$$P_{\mathrm{e}} = \frac{1}{2} erfc\left(\sqrt{\frac{E_{\mathrm{b}}}{N_0}}\right)$$

Comme nous pouvons voir sur le schéma ci-dessus, nous ne suivons pas exactement la courbe.

Chapitre 4

Récepteur simplifié

Nous n'avons pas eu le temps de mettre en place ce récepteur.

Chapitre 5

Conclusion

Nous pouvons conclure que notre chaîne de transmission fonctionne selon le comportement attendu. Néanmoins certains phénomènes inexpliqués persistent. Notamment lors de la génération d'une chaîne à 4 ressources (le graphique sur la probabilité d'erreur ci-dessus a été calculé sur base de 6 ressources), nous retrouvons beaucoup d'interférence entre symbôles sur la dernières ressource qui n'a pas être diminuée en allongeant la longueur du filtre FIR, ou en augmentant le facteur de rolloff. Les distorsions linéaires en bande passante des filtres analogique auraient pu être diminuées en utilisant des filtres de Butterworth qui n'ont pas de ripple.