

---

Techniques de transmission et traitement du signal

**Simulation d'une chaîne de transmission  
numérique avec Matlab®**

---

Alexis NOOTENS  
16139@student.ecam.be

Armen HAGOPIAN  
14040@student.ecam.be

ECAM Brussels  
Promenade de l'Alma 50  
1200 Woluwe-Saint-Lambert  
Belgique

5 mai 2018

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Implémentation</b>	<b>2</b>
2.1	Fichier principale . . . . .	2
2.2	Paramètres . . . . .	2
2.3	Émetteur . . . . .	2
2.4	Canal . . . . .	5
2.5	Receveur . . . . .	6
<b>3</b>	<b>Performances</b>	<b>7</b>
<b>4</b>	<b>Conclusion</b>	<b>7</b>
<b>A</b>	<b>Fichiers sources</b>	<b>10</b>
A.1	main.m . . . . .	10
A.2	parameters.m . . . . .	10
A.3	sender.m . . . . .	11
A.4	channel.m . . . . .	12
A.5	receiver.m . . . . .	12

## 1 Introduction

L'objectif de ce projet est simuler la couche physique d'un protocole de communication, c'est-à-dire le niveau 1 du modèle OSI. La simulation est réalisée avec le logiciel Matlab® édité par Mathworks®. Les contraintes imposées dans la simulation sont de tenir compte de plusieurs émetteurs et receveurs pouvant communiquer en même temps. Pour répondre à cette contrainte, la couche physique implémentée utilise le multiplexage fréquentiel.

Ce document reprend la conception du projet et les choix qui ont dû y être décidés, accompagnés de leur explication.

## 2 Implémentation

La section 2 décrit le travail apporté dans les fichiers qui composent le projet. Ces fichiers peuvent être consultés à l'annexe A. Ils consistent en :

- main.m** lance les scripts dans l'ordre logique.
- parameters.m** configure paramètres de simulation.
- sender.m** génère les données aléatoirement, puis sépare les canaux fréquentiellement.
- channel.m** simule un canal de communication en atténuant et filtrant les signaux.
- receiver.m** démodule les signaux reçus et tente de recomposer le signal émis.

### 2.1 Fichier principale

Le fichier principale, aussi nommé « main » par son nom anglais, se charge de lancer la simulation dans un ordre logique. Le fichier est consultable à l'annexe A.1. Son contenu est minime, il commence par nettoyer le plan de travail de variables et figures résiduelles. Il lance ensuite les scripts dans l'ordre parameters → sender → channel → receiver.

Une fois la simulation terminée, il affiche une figure comparant le signal dans un canal émis par sender, au signal recomposé dans ce même canal par le receveur. La figure 1 présente un exemple de cette comparaison. On peut y apercevoir que le signal recomposé est décalé par rapport au signal émis, et que son amplitude aux pics isolés est quasiment divisé par deux. Cela est normal étant donné que l'on retrouve plus de fréquence dans un pic isolé que dans une succession à la même amplitude. Ce pic souffrira donc plus fortement au filtrage fréquentiel.

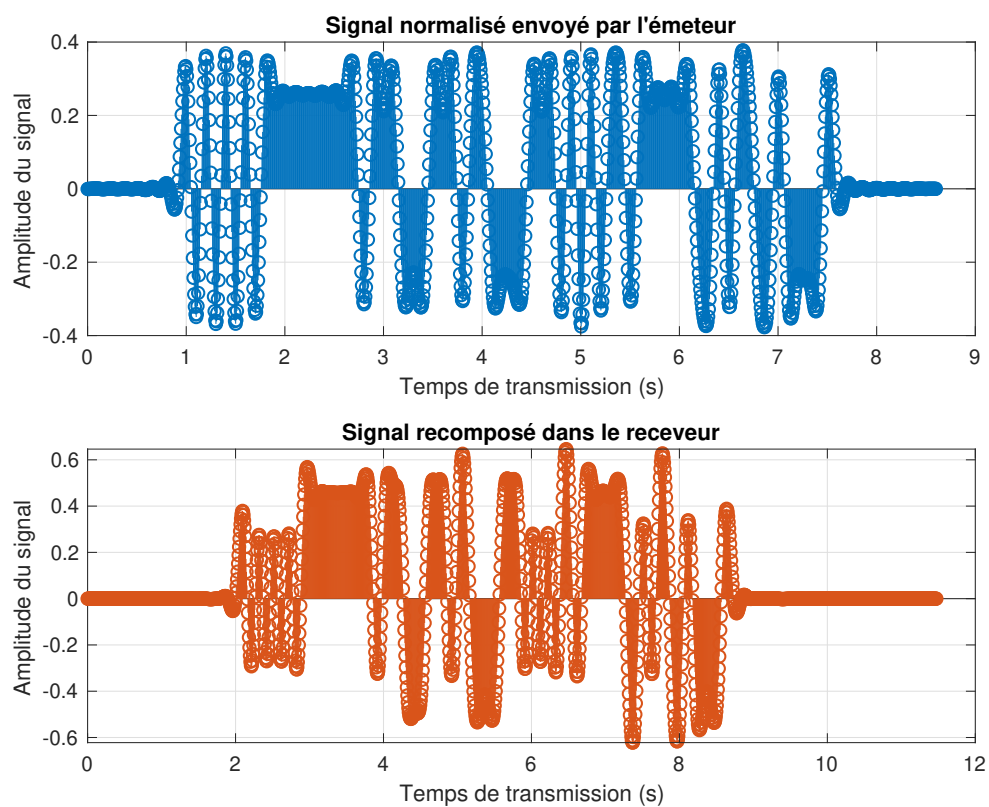
### 2.2 Paramètres

Le fichier **parameters.m** offre un accès rapide et concentré aux différents paramètres influençant la simulation tel que le nombre de canaux fréquentiel disponibles, la taille du message à envoyer, ou encore la vitesse d'envoi. Ce fichier est consultable à l'annexe A.2. Chaque paramètre est accompagné d'un commentaire expliquant dument son utilité.

### 2.3 Émetteur

Le fichier **emetteur.m** consultable à l'annexe A.3 commence la simulation à proprement dire. Il débute par générer des bits aléatoire suivant une distribution normale. Il rajoute une séquence de départ pour la forme uniquement, cette dernière n'est pas exploitée dans le receveur. Les bits sont ensuite codés dans un code PAM-2 où les 0 deviennent -1 et les 1 restent 1.

Afin de diviser le spectre fréquentiel pour y définir des canaux, il faut s'assurer que les messages envoyées se limite à leur bande allouée. Pour ce faire nous utilise le filtre en cosinus surélevé qui à la merveilleuse propriété que pour un signal avec un durée de symbole  $\xi$  seconde, il ne consommera que  $\frac{\xi}{2}$  largeur de spectre dans le domaine fréquentiel. C'est la meilleur efficacité spectrale atteignable mathématiquement. Un soucis survient avec l'utilisation de ce filtre, c'est que le signal codé maintient



**Figure 1 :** Comparaison entre le signal en bande de base émis dans l'émetteur et celui recomposé dans le receveur. Le signal reçu est décalé d'approximativement 1 seconde par rapport au signal émis. Le signal reçu est également plus sévèrement atténué aux pics isolés qui contiennent des plus hautes fréquences.

son amplitude maximale durant un bref instant, puis chute rapidement. Ce type de réponse n'est pas utilisable en pratique car la fréquence de capture d'échantillons dans le receveur n'est pas monotone et stable, ni même en phase. Pour adresser ce problème, le filtre maintient son plafond durant un instant plus allongé, mais aux prix de flancs montant et descendant plus raides car la période d'expression d'un symbole n'est pas augmenté. Ces flancs plus raides entraînent une plus grande consommation de bande spectrale, nous définissons une marge au signal de  $1 + \alpha$  largeur de bande avec  $0 \leq \alpha \leq 1$ , dénommé « facteur de roll-off ». Ce facteur contrôle le temps de « plafond » du symbole dans le domaine fréquentiel. Dans notre implémentation, nous avons choisi un facteur de 0,4 arbitrairement.

La description du filtre précédemment faite s'applique au domaine continue. Quand nous passons dans le domaine discret, le temps d'échantillonnage doit être pris en compte. Si nous n'utilisons qu'un seul échantillon pour convoluer le filtre en cosinus surélevé, sa réponse impulsionnelle sera semblable à une impulsion de Dirac, avec une consommation fréquentielle infinie, ce qui est l'opposé de ce qui est désiré. Si nous utilisons une infinité d'échantillon, nous obtiendrons une réponse parfaite semblable au domaine continue. Maintenant que les extrémités sont définies, il est nécessaire de savoir combien d'échantillons aux minimums sont nécessaires pour garder les spectres de différents canaux séparés. L'équation (5) répond à cette question. En partant de l'équation (1) qui fixe les contraintes.

Soit  $T_n$  le taux d'échantillonnage nécessaire pour restreindre les canaux à leur bande sans qu'ils ne s'empêtent,  $T_b$  le taux d'échantillonnage du signal avant filtrage,  $N$  le nombre de canaux actifs et  $\alpha$  le facteur de roll-off. Puisque le premier canal non-modulé sera « single-sideband »  $\frac{1}{2T_b}$ , et les suivants seront « double-sideband »  $\frac{1}{T_b}$  :

Par le théorème de Nyquist

$$\frac{1}{2T_n} \geq \left[ \frac{1}{2T_b} + (N-1) \frac{1}{T_b} \right] (1 + \alpha) \quad (1)$$

Puisque  $\beta$  est le rapport  $T_n \div T_b$

$$\frac{\beta}{2T_b} \geq \left[ \frac{1}{2T_b} + (N-1) \frac{1}{T_b} \right] (1 + \alpha) \quad (2)$$

En rajoutant  $2T_b$  de chaque côté

$$\beta \geq \left[ 1 + (N-1) 2 \right] (1 + \alpha) \quad (3)$$

En considérant le pire cas de  $\alpha = 1$

$$\beta \geq 2 + (N-1) 4 \quad (4)$$

En simplifiant

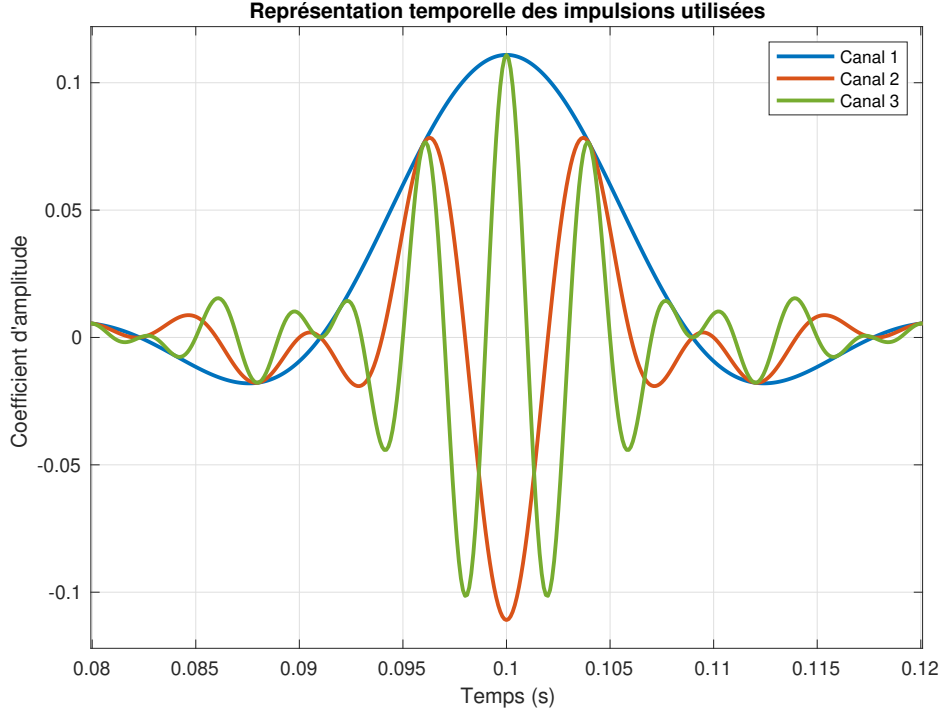
$$\beta \geq 4N - 2 \quad (5)$$

Et puisque nous n'aimons pas ce qui n'est pas linéaire, nous prenons dans la simulation  $\beta = 4N$ .

Un dernier paramètre à décider dans la construction du filtre numérique et la longueur de la réponse impulsionnelle qui va servir à convoluer le signal. Pour que le filtrage respecte parfaitement les caractéristiques que nous lui avons défini, il lui faut une longueur infini. Ceci n'étant pas possible en pratique, la réponse doit être tronquée tout en gardant le maximum de puissance. Par essais empiriques, nous avons déterminé que 20 fois le taux de sur-échantillonnage est une bonne valeur.

Les filtres générés et leur réponse impulsionnelle obtenues, nous les modulons par les porteuses avant d'être convoluées avec le signal. Nous prenons la liberté d'éloigner les porteuses un peu plus que nécessaire avec le paramètre  $L$  défini dans la simulation présentée comme 1,25. Ce paramètre multiplie la bande de fréquence séparant 2 porteuses côté-à-côté. Nous avons fait ce choix car autrement, les fréquences de coupures de filtres servant à séparer les canaux se touchent à  $-3$  dB, laissant possiblement beaucoup

d'interférence entre canaux. Si nous pouvons nous permettre cette séparation, c'est parce que nous avons calculé le taux de sur-échantillonnage avec un facteur de roll-off à 1, or dans la simulation nous l'avons configuré à 0,4. Cela nous offre de la marge pour séparer les canaux, tant que  $L \leq \frac{1+1}{1+0,4} = 1,4287$ . Nous obtenons les impulsions présentées à la figure 2. Cette figure présente les 3 impulsions nécessaires pour multiplexer en 3 canaux. Il est intéressant de constater que l'impulsion du canal 2 est inversée en amplitude.



**Figure 2 :** Représentation temporelle des impulsions utilisées pour multiplexer en 3 canaux. Chaque des impulsions est sa réponse en bande de base modulée par sa porteuse. Il est intéressant de constater que l'impulsion du canal 2 est inversée en amplitude.

Les signaux modulés, leur puissance est normalisée à une quantité de milliwatt par seconde en divisant/multipliant les amplitudes par le facteur adéquat. La puissance de chaque canal est évaluée en intégrant la norme au carré sur l'impédance caractéristique du milieu de propagation.

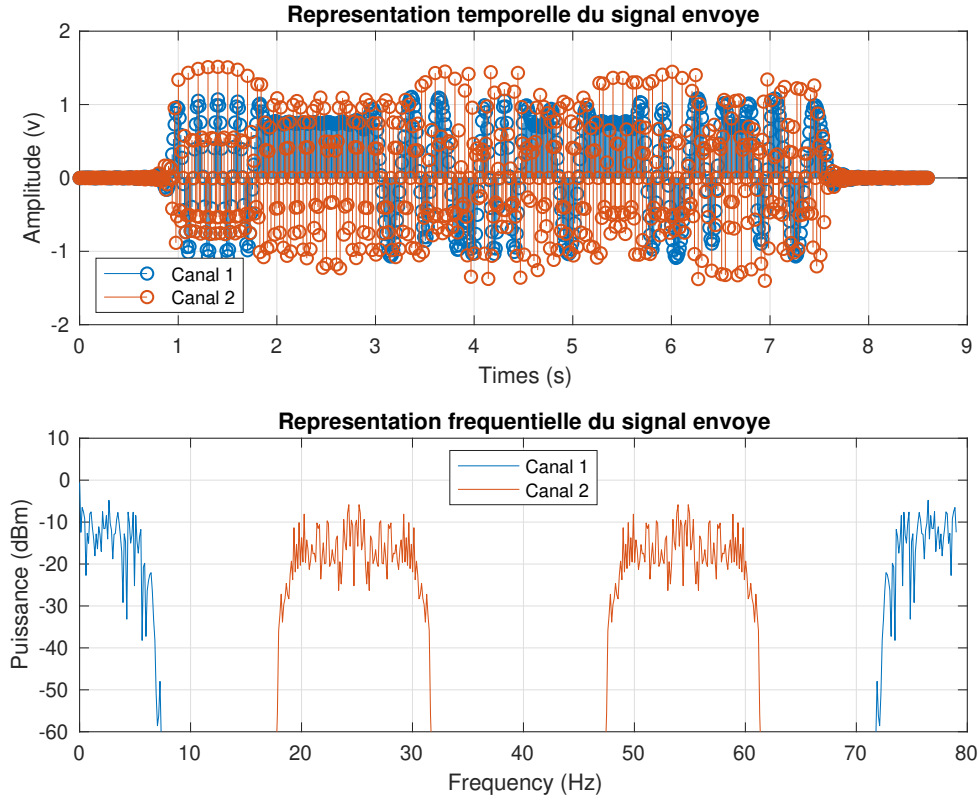
$$P[s] = \frac{\frac{1}{N} \sum_{n=0}^N s[n]^2}{Z_0} \quad (6)$$

Le script `sender.m` affiche un récapitulatif de la transmission en temporel et fréquentiel avant de sommer tous les canaux pour les envoyer sur le seul lien physique disponible, le câble ou le rayonnement électro-magnétique. Un exemple de ce récapitulatif est présenté à la figure 3.

## 2.4 Canal

Le fichier `canal.m` consultable à l'annexe A.4 à pour mission de simuler les effets du canal de communication. Nous nous attendons à ce qu'il se comporte comme un filtre passe-bas, c.-à-d. une atténuation des amplitudes et un décalage temporel. Tout cela accompagné de bruit parasite.

Nous nous attendons à ce que le bruit soit de type AWGN, Additive white Gaussian noise, pouvant être symbolisé par une variable aléatoire de distribution normale et de moyenne nulle  $g[k] \sim \mathcal{N}(0, \sigma^2)$ . Tout commence par la génération d'un vecteur aléatoire de distribution normale de même taille que le vecteur de donnée. Ce vecteur aléatoire est ensuite filtré pour ne pas contenir de fréquence supérieure à celle de Nyquist. L'intensité du bruit se règle en multipliant le vecteur obtenu, qui est de variance 1, par l'écart-type de la variance désiré. Un facteur d'atténuation  $A$  borné tel que  $0,6 \leq A \leq 0,9$  est également



**Figure 3 :** Signaux de 50 bits chacun envoyés sur 2 canaux fréquentiels au rythme de 10 bit/s avec une fréquence d'échantillonnage de 80 Hz.

généralisé. Le script effectue ensuite  $S_2[k] = A \cdot S_1[k] + g[k]$  pour appliquer l'atténuation et le bruit. Du « zero-padding » est ajouté au début du signal pour simuler un décalage temporel.

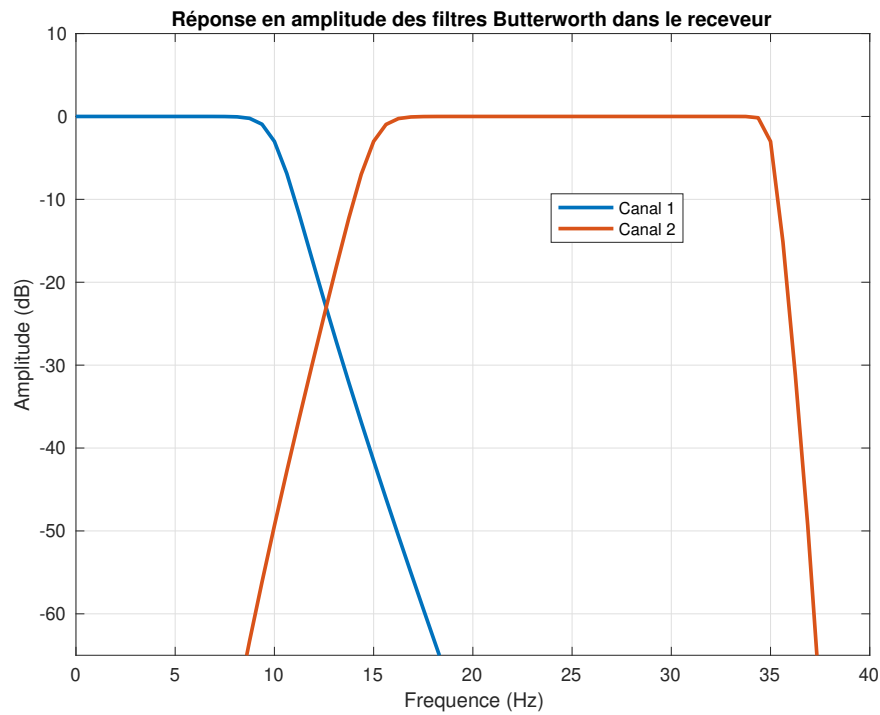
## 2.5 Receveur

Le fichier `receiver.m` consultable à l'annexe A.5 se charge de ramener les signaux transmis en bande de base et prend des décisions sur les valeurs mesurées à tous les  $T_b$  instants en espérant retrouver le signal original.

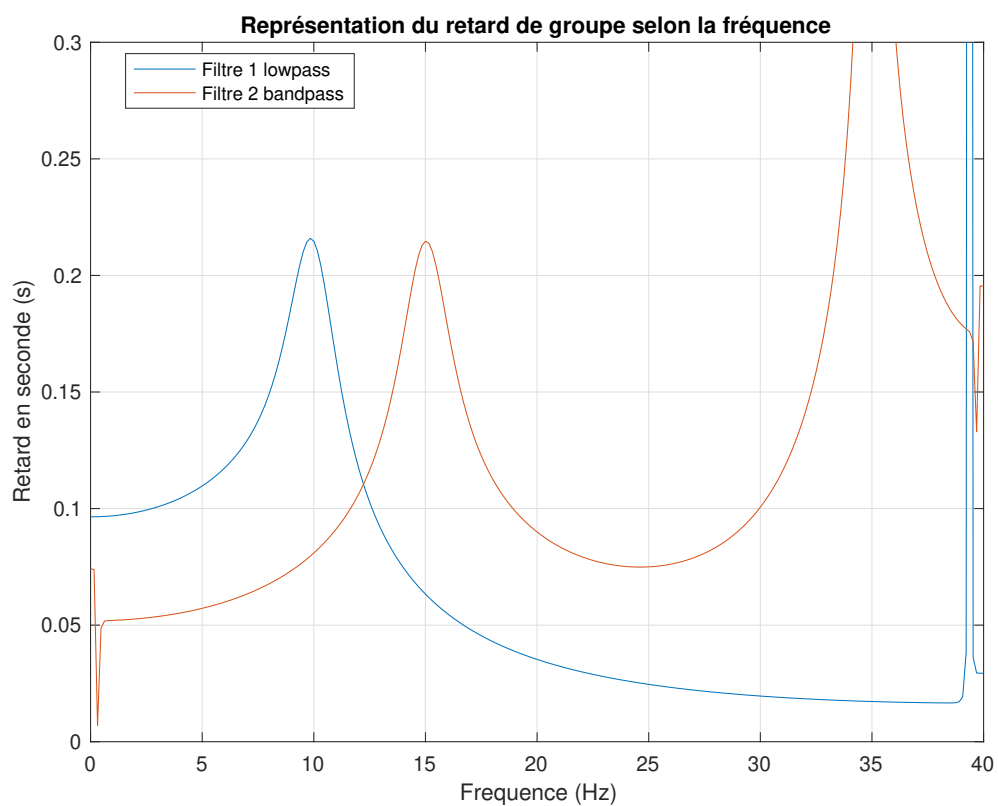
Le script commence par générer des filtres de type Butterworth qui vont lui permettre de séparer les canaux. Pour filtrer, on commence par obtenir la fraction polynomiale représentant le filtre analogique. De cette fonction, on calcule la réponse fréquentielle entre la fréquence 0 et la fréquence d'échantillonnage. On applique la transformée de Fourier inverse à cette réponse fréquentielle, et nous obtenons la réponse impulsionnelle du filtre. Il suffit dès lors de convoluer les échantillons temporels de notre signal avec les réponses impulsionnelles calculées et nous séparons ainsi les bandes spectrales. Nous calculons des filtres d'ordre 10, la figure 4 et 5 présentent les filtres permettant de séparer les deux canaux utilisés dans cette simulation.

Une fois les canaux séparés, il faut les ramener en bande de base. On ne peut pas simplement diviser les signaux par un cosinus de même fréquence que la porteuse car le déphasage entre les deux laisserait une composante sinusoidale parasite dans nos échantillons démodulés. Pour obtenir notre signal en bande de base, nous multiplions encore le signal modulé avec un cosinus à même fréquence, multiplier deux cosinus amène à additionner leur fréquence. Nous filtrons ensuite ce signal doublement multiplié par un filtre passe-bas pour supprimer les porteuses envoyées dans de plus haute fréquence, et les fréquences restantes sont notre signal en bande de base. En procédant de cette manière, nous contournerons le problème d'estimation de la phase de la porteuse.

Le script receveur affiche un récapitulatif de la transmission reçue, comme l'émetteur affiche une



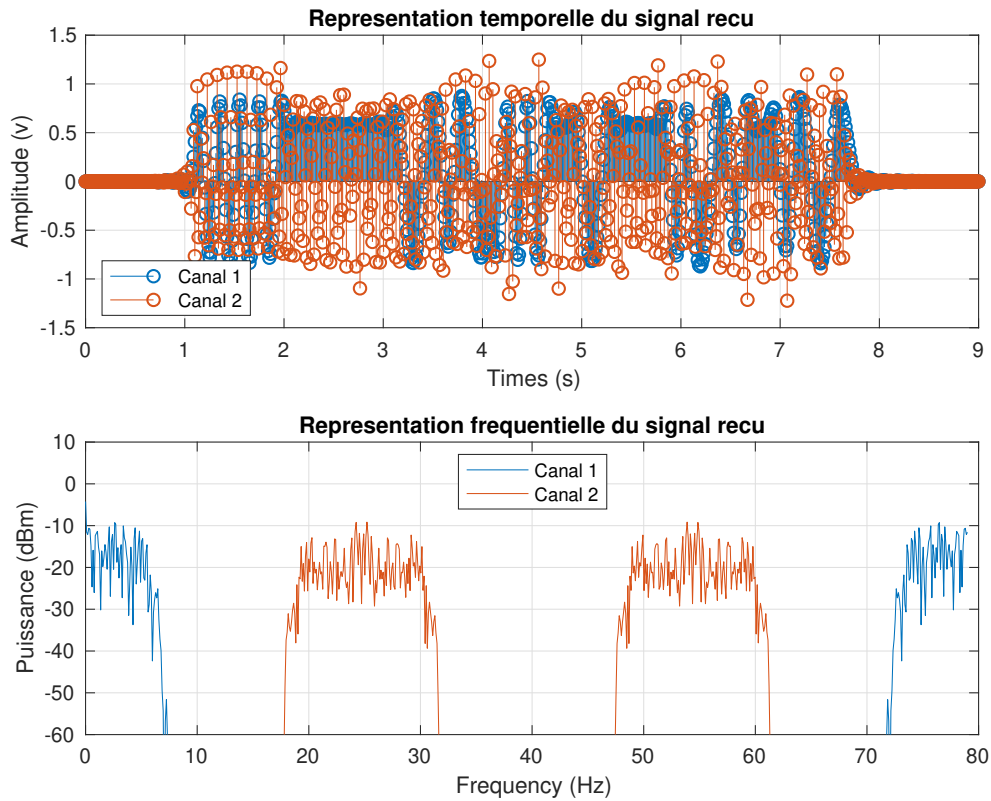
**Figure 4 :** Réponse en amplitude des filtres de type Butterworth utilisés dans le receveur pour séparer les canaux.



**Figure 5 :** Retard de groupe des filtres de type Butterworth utilisés dans le receveur pour séparer les canaux. Le filtre 1 est un lowpass avec une fréquence de coupure à 10 Hz. Le filtre 2 est un bandpass avec une fréquence de coupure basse à 25 Hz et fréquence de coupure haute à 35 Hz.



récapitulatif de la transmission envoyée. Ces récapitulatifs permettent de comparer les signaux émis avant d'être sommé dans l'environnement de transmission, *e.g.* un câble, et les signaux reçus après les avoir séparés par filtrage. La figure 6 du receveur doit être comparé avec la figure 3 de l'émetteur. On peut constater que : temporellement le receveur a un message décalé de quelques échantillons ; et fréquemment un signal atténué de quelques dBm.

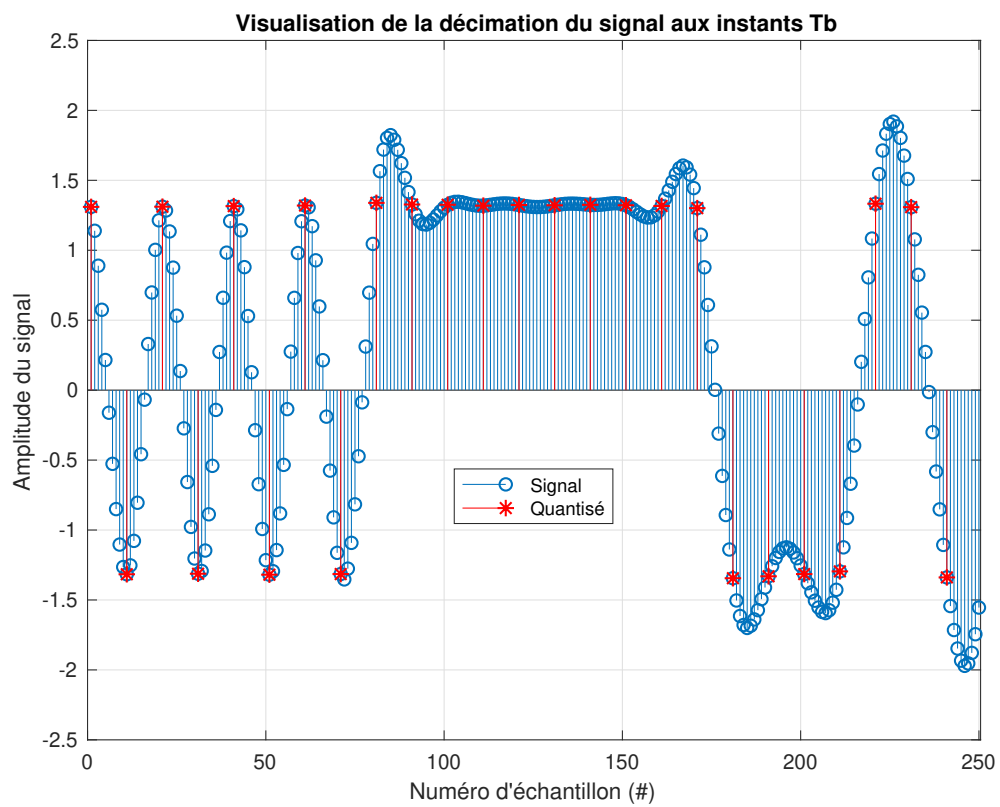


**Figure 6 :** Signaux de 50 bits chacun envoyés sur 2 canaux fréquentiels au rythme de 10 bit/s avec une fréquence d'échantillonnage de 80 Hz.

Une fois le signal en bande de base recomposé, le receveur doit repérer début du signal et capturer un échantillon tous les  $T_b$  instant. Depuis cet échantillon capturé, il doit prendre la décision de choisir si le code transmis était +1 ou -1. La figure 7 présente un exemple de capture d'échantillons effectuée par le receveur.

### 3 Performances

### 4 Conclusion



**Figure 7 :** Visualisation de la capture d'échantillons pour le décideur.

## A Fichiers sources

### A.1 main.m

```
1 % This work is licensed under the Creative Commons Attribution 4.0
2 % International License. To view a copy of this license, visit
3 % http://creativecommons.org/licenses/by/4.0/ or send a letter to
4 % Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.
5 clear, close all
6
7 parameters
8 % generate data and send it
9 sender
10 % add noise and delay
11 channel
12 % filter data and read it
13 receiver
14
15 % compare the sent signal with the received one
16 figure
17 subplot(2,1,1)
18 stem(linspace(0, len1*Tn, len1), s1(:,1));
19 title('Signal normalise envoye par l''emetteur')
20 xlabel('Temps de transmission (s)')
21 ylabel('Amplitude du signal')
22 grid
23
24 subplot(2,1,2)
25 len3 = size(s2,1);
26 stem(linspace(0, len3*Tn, len3), s2(:,1), 'Color', [0.85 0.33 0.1]);
27 title('Signal recompose dans le receveur')
28 xlabel('Temps de transmission (s)')
29 ylabel('Amplitude du signal')
30 grid
31
32 % report QS
33 disp("Taux d'erreurs :")
34 disp(sum(xor(x, decoded))/size(x,1))
```

### A.2 parameters.m

```
1 % This work is licensed under the Creative Commons Attribution 4.0
2 % International License. To view a copy of this license, visit
3 % http://creativecommons.org/licenses/by/4.0/ or send a letter to
4 % Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.
5
6 codesymbol = @(x)x.*2-1;
7
8 % System
9 N = 2; % available channels
10 M = 50; % message size (bits)
11
12 % Sender
13 R = 10; % bit rate
14 Tb = 1/R; % bit duration
15 roll = 0.40; % rolloff factor
16 L = 1.25; % bandwidth xTb
17 beta = 4*N; % upsampling factor
18 Tn = Tb/beta; % upsampling rate
19 span = 20; % rcos span for thinner bandwidth consumption
20 pwr = 1; % channel power in mW
21
22 % Channel
23 ZO = 50; % characteristic impedance
24 shift = 4; % samples delay
```

```

25
26 % Receiver
27 impulseL = 128;
28 startSeq = [1 0 1 0 1 0 1 0 ... % test the channel response
29             1 1 1 1 1 1 1 1]; % set an unique sequence

```

### A.3 sender.m

```

1 % This work is licensed under the Creative Commons Attribution 4.0
2 % International License. To view a copy of this license, visit
3 % http://creativecommons.org/licenses/by/4.0/ or send a letter to
4 % Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.
5
6 x = randi([0 1], M, N);
7 % append the start sequence
8 x = [startSeq*ones(1, N); x];
9 a = codesymbol(x);
10 % shape to impulse
11 rcos = rcosdesign(roll, span, beta);
12 a = upsample(a, beta);
13 s1 = conv2(rcos, 1, a);
14 len1 = size(s1, 1);
15 % carrier frequencies
16 carfreq = (0:N-1)'*L*2/Tb;
17
18 %% plot impulsions
19 iX = linspace(0, span/1e2, 1e2*span+1);
20 iY = rcosdesign(roll, span, 1e2);
21 plot(iX, iY' * ones(1, N) .* ...
22      cos(carfreq*linspace(0, 2*pi, span*1e2+1)))
23 ylim([-max(iY)*1.1 +max(iY)*1.1])
24 title("Représentation temporelle des impulsions utilisées")
25 ylabel("Coefficient d'amplitude"), xlabel("Temps (s)")
26 legend(strcat("Canal ", num2str((1:N))))
27 grid
28 clear iX iY
29
30 %% modulate by carriers
31 t = (0:Tn:(len1-1)*Tn)'*ones(1,N);
32 s1High = s1.*cos(2*pi*carfreq'.*t);
33
34 % normalise power to 'pwr' mW
35 pwrTimesSec = pwr*len1*Tn; % mW per second * transmission time
36 avgPower = bandpower(s1High)/Z0*1000/(pwrTimesSec);
37 s1High = s1High./sqrt(avgPower);
38
39 % sum all channels before transmission
40 data = sum(s1High, 2);
41
42 %% plot visual representation of the transmission
43 figure
44 subplot(2,1,1)
45 stem(linspace(0, len1*Tn, len1), s1High)
46 title('Représentation temporelle du signal envoyé')
47 ylabel('Amplitude (v)'), xlabel('Times (s)')
48 legend(strcat("Canal ", num2str((1:N))), 'Location', 'SouthWest')
49 grid
50
51 subplot(2,1,2)
52 plot(linspace(0, 1/Tn-1, len1), pow2db(abs(fft(s1High/len1)).^2/Z0)+30)
53 ylim([-60 10])
54 title('Représentation fréquentielle du signal envoyé')
55 ylabel('Puissance (dBm)'), xlabel('Frequency (Hz)')
56 legend(strcat("Canal ", num2str((1:N))), 'Location', 'North')
57 grid

```

## A.4 channel.m

```
1 % This work is licensed under the Creative Commons Attribution 4.0
2 % International License. To view a copy of this license, visit
3 % http://creativecommons.org/licenses/by/4.0/ or send a letter to
4 % Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.
5
6 % gaussian noise
7 noise_1 = randn([numel(data) 1]);
8 [bf,af] = butter(1, 0.5);
9 noise_f = ifft(freqz(bf, af, impulseL, 'whole', 1/Tn));
10 noise_2 = conv(noise_f, noise_1);
11 noise_2 = noise_2(impulseL/2:end-impulseL/2);
12
13 % damping factor; between 0.60<=x<=0.90
14 alpha = (0.90-0.60)*rand([1 1])+0.60;
15
16 % increase noise with variance
17 variance = 0;
18 std_dev = sqrt(variance);
19 noise_3 = noise_2*std_dev;
20
21 data = alpha*data+noise_3;
22 data = [zeros(shift,1); data];
```

## A.5 receiver.m

```
1 % This work is licensed under the Creative Commons Attribution 4.0
2 % International License. To view a copy of this license, visit
3 % http://creativecommons.org/licenses/by/4.0/ or send a letter to
4 % Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.
5
6 % calculate the bandwidth limits for each channel
7 cutoff = [carfreq-1/Tb carfreq+1/Tb]*2*Tn;
8 if N == 1, cutoff = [0 0.9999]; end
9 % pre-allocate filters matrix
10 H = zeros(impulseL, N);
11
12 % first channel lowpass
13 [bf,af] = butter(10, cutoff(1,2));
14 H(:,1) = ifft(freqz(bf, af, impulseL, 'whole', 1/Tn));
15
16 % others channels bandpass
17 for n = 2:N
18     [bf,af] = butter(10, [cutoff(n,1) cutoff(n,2)]);
19     H(:,n) = ifft(freqz(bf, af, impulseL, 'whole', 1/Tn));
20 end
21
22 % separate channels
23 s2High = conv2(data, 1, H);
24
25 % demodulate
26 len2 = size(s2High,1);
27 t = (0:Tn:(len2-1)*Tn)*ones(1,N);
28 s2 = s2High.*cos(2*pi*carfreq'.*t);
29 s2(:,1) = s2High(:,1);
30 for n = 2:N
31     [bf,af] = butter(5, carfreq(n)*2*Tn);
32     impulse = ifft(freqz(bf, af, impulseL, 'whole', 1/Tn));
33     s2(:,n) = conv(s2(:,n), impulse(1:1:end), 'same'); % forward
34     s2(:,n) = conv(s2(:,n), impulse(end:-1:1), 'same'); % backward
35 end
36
37 % filter the canal noise with the adequate filter
38 s2 = conv2(rcos, 1, s2);
39 % find filters delay
40 [~,i] = max(H);
```

```

41 % compensate the start frame
42 s2t = s2(span*beta+i+shift-3:end, :);
43 % generate the index vector
44 s2i = 1:beta:beta*size(x,1);
45 % extract the values at index
46 decoded = s2t(s2i,:);
47 % quantize the extracted values
48 decoded = decoded>0;
49
50 % hit markers *PEW* *PEW*
51 figure, hold on
52 stem(s2t(:,2))
53 stem(s2i, s2t(s2i,2), 'r*', 'MarkerSize', 8.0)
54 grid, hold off
55
56 %% plot visual representation of the transmission
57 figure
58 subplot(2,1,1)
59 stem(linspace(0, len2*Tn, len2), s2High)
60 title('Representation temporelle du signal recu')
61 ylabel('Amplitude (v)'), xlabel('Times (s)')
62 legend(strcat("Canal ", num2str((1:N))), 'Location', 'SouthWest')
63 grid
64
65 subplot(2,1,2)
66 plot(linspace(0, 1/Tn-1, len2), pow2db(abs(fft(s2High/len2)).^2/Z0)+30)
67 ylim([-60 10])
68 title('Representation frequentielle du signal recu')
69 ylabel('Puissance (dBm)'), xlabel('Frequency (Hz)')
70 legend(strcat("Canal ", num2str((1:N))), 'Location', 'North')
71 grid

```