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

13 juin 2018

Table des matières

Introduction

L'objectif de ce projet est de simuler la couche physique d'un protocol de communication, c'est-à-dire le niveau 1 du modèle OSI. La simulation est réalisée à l'aide du logiciel MATLAB® édité par Mathworks®. Une contrainte imposée dans la simulation est de tenir compte de plusieurs émetteurs et receveurs pouvant communiquer simultanément. 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.

1 Implémentation

La section 1 décrit le modus operandi réalisé dans les fichiers qui composent le projet. Ces fichiers peuvent être consultés à l'annexe B. 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.

1.1 Fichier principale

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

Une fois la simulation terminée, il affiche également une figure comparant le signal dans un canal émis par l'émetteur, 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 délayé par rapport au signal émis, et que ses amplitudes aux pics isolés est quasiment divisées 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.

1.2 Paramètres

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

1.3 Émetteur

Le fichier `emetteur.m`, consultable à l'annexe B.3, commence la simulation proprement dite. Il débute par générer une séquence de bits aléatoire suivant une distribution normale grâce à `randn()`.

1.3.1 Séquence de départ

Il rajoute une séquence de départ pour la forme uniquement. Cette dernière n'est pas exploitée dans le receveur, mais trouve son utilité dans l'appréciation qualitative du canal par nos yeux. Cette séquence consiste premièrement en 4 oscillations de symboles opposés $(1, 0, 1, 0, 1, 0, 1, 0)$, puis de 8 symboles

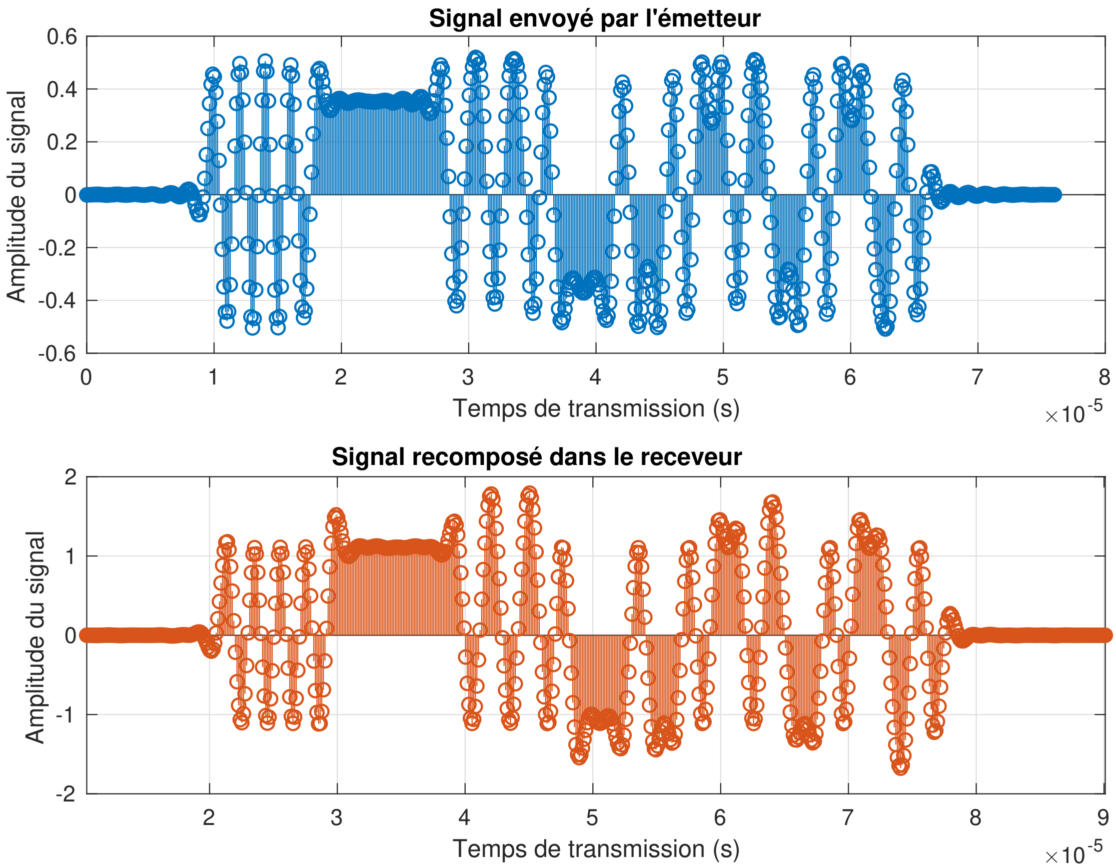

Figure 1 : Comparaison entre le signal en bande de base émis dans l'émetteur et celui recomposé dans le receveur. Le signal reçu est décalé d'approximativement 1 seconde par rapport au signal émis. Le signal reçu est également plus sévèrement atténué aux pics isolés qui contiennent des plus hautes fréquences. Le rapport d'amplitude n'est pas conservé car le receveur ne tente pas de compenser la perte de tension dans le canal. À la place le signal est juste normalisé à 200 mW avant d'être émis.

identiques $(1, 1, 1, 1, 1, 1, 1, 1)$. L'intérêt de cette séquence est de constater la réponse du canal aux plus hautes fréquences du signal, et l'amplitude RMS d'un message constant. Cette séquence de départ est aisément distinguable à la figure 1.

Après l'ajout de la séquence de départ, les bits sont codés dans un code PAM-2 où les 0 deviennent -1 et les 1 restent 1.

1.3.2 Cosinus surélevé

Afin de diviser le spectre fréquentiel pour y définir des canaux, il faut s'assurer que les messages envoyés se limite à leur bande allouée. Pour ce faire nous pouvons utiliser le filtre en sinus cardinal qui à la merveilleuse propriété que pour un signal avec un durée de symbole de T_b seconde, il ne consommera que $\frac{1}{2T_b}$ largeur de spectre dans le domaine fréquentiel. C'est mathématiquement la meilleur efficacité spectrale atteignable.

Un soucis survient avec l'utilisation de ce filtre en pratique, c'est que le signal codé maintient son amplitude maximale, dénommée plafond, durant un bref instant puis chute brusquement. Ce type de réponse n'est pas utilisable en pratique car la fréquence de capture d'échantillons dans le receveur n'est ni monotone, ni même en phase. Pour adresser ce problème, nous utilisons une version alternative du sinus cardinal nommée le filtre en cosinus surélevé. Ce filtre a la particularité de maintenir son plafond plus longtemps, mais aux prix de flancs montant et descendant plus raides car la période d'expression d'un symbole n'est pas augmentée. Ces flancs plus raides entrainent une plus grand consommation de bande spectrale. Le cosinus surélevé défini une largeur de bande consommée $1 + \alpha$ avec α dénommé « facteur de roll-off » compris dans l'interval $0 \leq \alpha \leq 1$. Un $\alpha = 0$ rend le filtre égale à celui d'un sinus cardinal. Ce facteur contrôle le temps de plafond d'un symbole dans le domaine temporel. Dans notre implémentation, nous avons choisi un facteur $\alpha = 0{,}4$ arbitrairement.

1.3.3 Sur-échantillonnage

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 un consommation fréquentielle infinie, ce qui est l'opposé de ce qui est désiré. Et si nous utilisons une infinité d'échantillon, nous obtiendrons une réponse parfaite semblable au domaine continue. Connaissant les extrémités, nous nous intéressons à savoir combien d'échantillons au minimum 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'empiètent, T_b le taux d'échantillonnage du signal avant filtrage, N le nombre de canaux actifs et α 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

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

Puisque β est le rapport $T_n \div T_b$

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

En rajoutant $2T_b$ de chaque côté

$$\beta \geq \Big[1 + (N-1)\,2\Big](1+\alpha) \tag{3}$$

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

$$\beta \geq 2 + (N-1)\,4 \tag{4}$$

En simplifiant

$$\beta \geq 4N - 2 \tag{5}$$

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 essaies empiriques, nous avons déterminé que 20 fois le taux de sur-échantillonnage est une bonne valeur.

1.3.4 Modulation

Les filtres générés et leur réponse impulsionnelle obtenues, nous les modulons par les porteuses avant d'être convolués avec le signal. Les fréquences des porteuses sont obtenues en séparant chaque canal par $\frac{1}{2Tb}$ 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.

Les signaux modulés, leur puissance est normalisée à une quantité de milliwatt en multipliant les amplitudes par le facteur adéquat. Nous normalisons la puissance du signal après la modulation car la puissance de la porteuse s'additionne au signal utile. Si nous le faisions avant, nous retrouverions une puissance deux fois plus grande dans le premier canal, sans porteuse, que dans les autres. La puissance de chaque canal est évaluée en intégrant la norme au carré sur l'impédance caractéristique du milieu de propagation. Dans notre simulation, nous avons choisi d'évaluer la puissance par $Z_0 = 1\,\Omega$ et nous normalisons à $200\,\mathrm{mW}$.

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

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 cable ou le rayonnement électromagnétique. Un exemple de ce récapitulatif est présenté à la figure 3.

1.4 Canal

Le fichier `canal.m`, consultable à l'annexe B.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 temporelle. Tout cela accompagné de bruit parasite.

Nous nous attendons à ce que le bruit sois 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 désirée. Un facteur d'atténuation A borné tel que $0{,}6 \leq A \leq 0{,}9$ est également généré. 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.

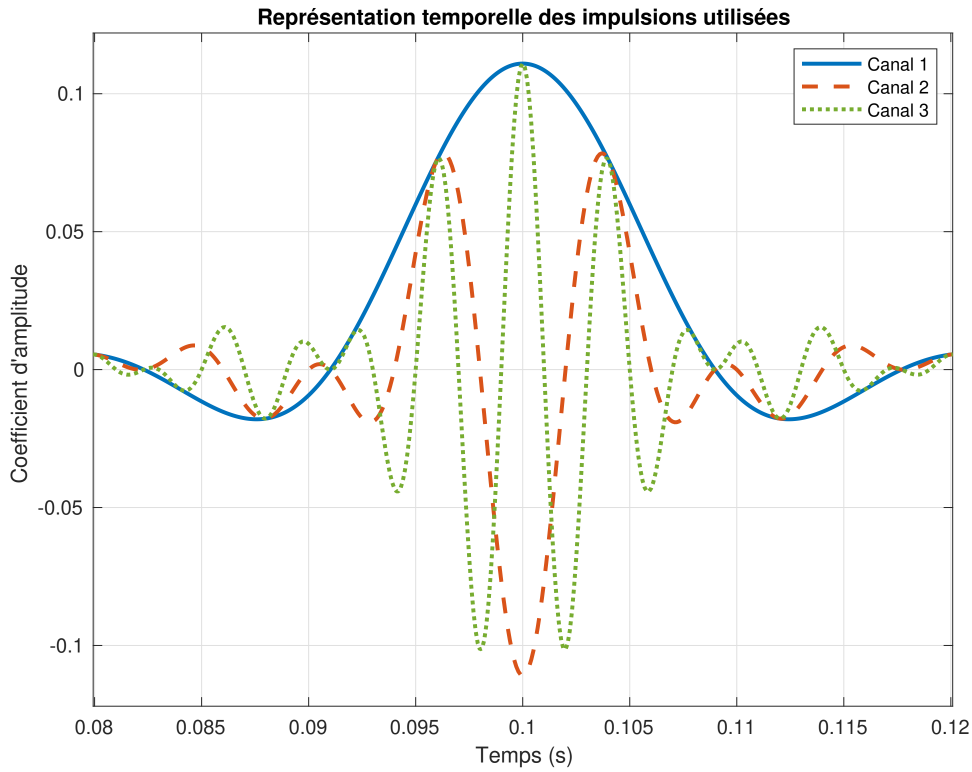

Figure 2 : Représentation temporelle des impulsions utilisées pour multiplexer en 3 canaux. Chaque des impulsions est sa réponse en bande de base modulée par sa porteuse. Il est intéressant de constater que l'impulsion du canal 2 est inversée en amplitude.

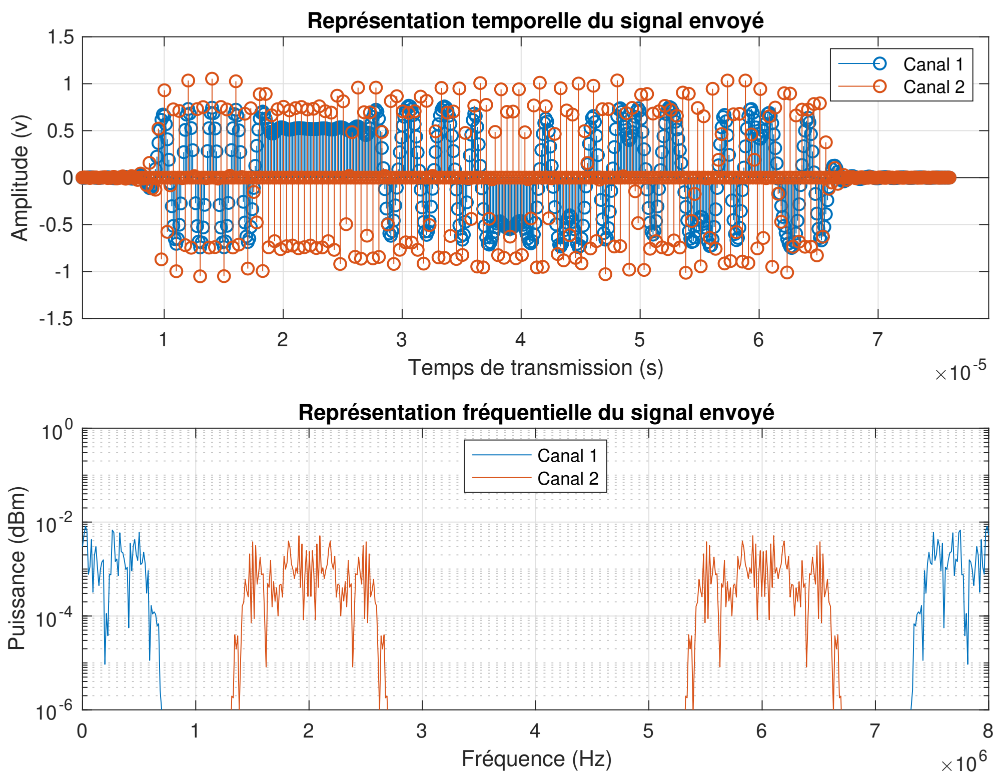

Figure 3 : Signaux de 50 bits chacun envoyés sur 2 canaux fréquentiels au rythme de 10 bit/s avec une fréquence d'échantillonage de 80 Hz.

1.5 Receveur

Le fichier `receiver.m`, consultable à l'annexe B.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.

1.5.1 Filtrage

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 fréquentiel entre la fréquence 0 et la fréquence d'échantillonnage. On applique la transformé de Fourier inverse à cette réponse fréquentiel, 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 des séparer les deux canaux utilisé dans cette simulation.

1.5.2 Démodulation

Une fois les canaux séparés, il faut les ramener en bande de base. On ne peut pas simplement diviser les signaux pas un cosinus de même fréquence que la porteuse car le déphasage entre les deux laisserait une composante sinusoïdale 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 envoyée dans de plus haute fréquence, et les fréquences restantes sont notre signal en bande de base. En procédant de cette manière, nous contournons le problème d'estimation de la phase de la porteuse.

1.5.3 Comparaison

Le script receveur affiche un récapitulatif de la transmission reçue, comme l'émetteur affiche une 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 fréquentiellement un signal atténué de quelques dBm.

Une fois le signal en bande de base recomposé, le recevoir 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 simulation se permet de calculer le délai généré pour savoir où le signal commence. Ceci est possible car nous disposons des réponses en phase de chaque filtres appliqués tout au long des calculs. La figure 7 présente un exemple de capture d'échantillons effectuée par le receveur.

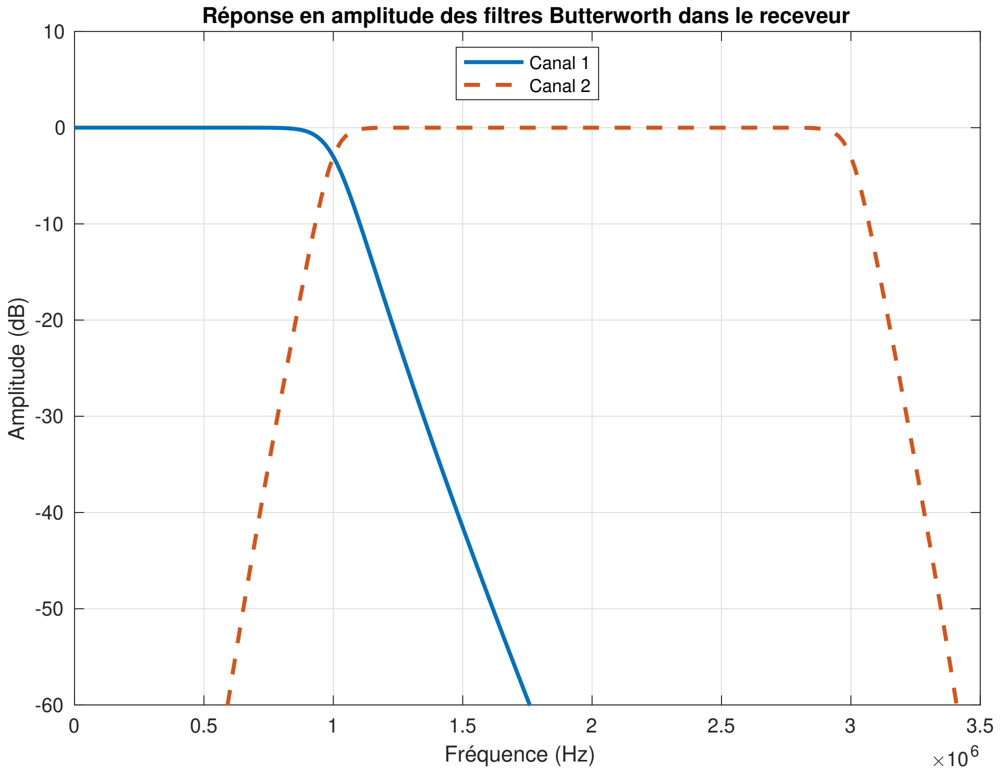

Figure 4 : Réponse en amplitude des filtres de type Butterworth utilisés dans le receveur pour séparer les canaux. Leurs pentes se rencontrent à $-3\,\mathrm{dB}$.

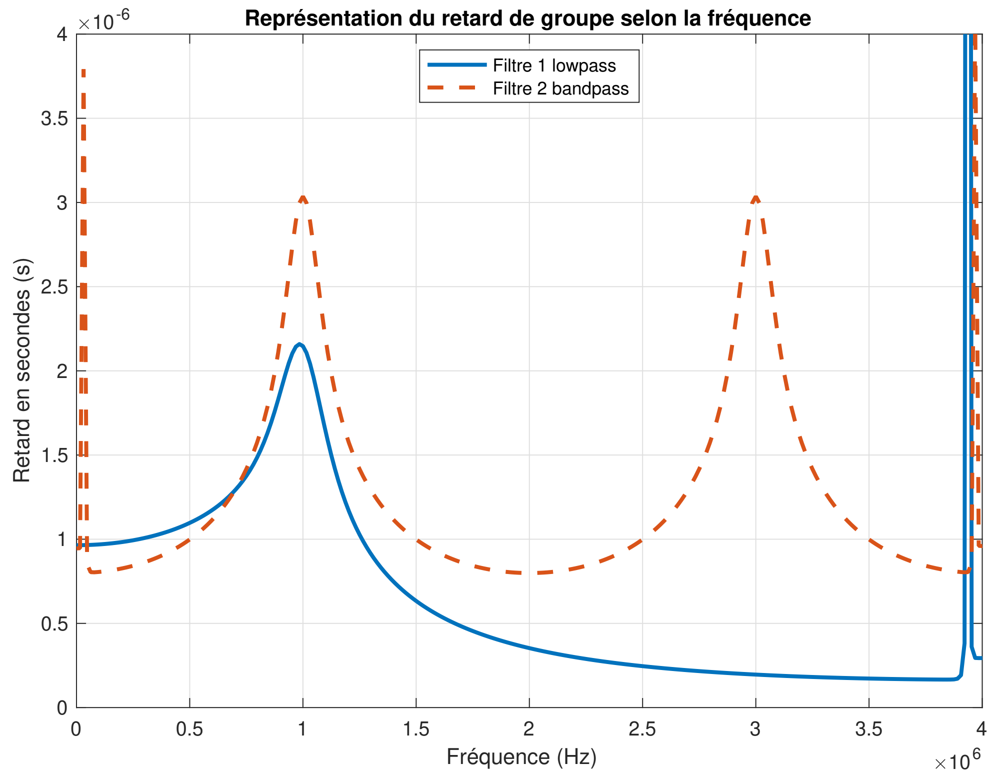

Figure 5 : Retard de groupe des filtres de type Butterworth utilisés dans le receveur pour séparer les canaux. Le filtre 1 est un passe-bas avec une fréquence de coupure à 1 MHz. Le filtre 2 est un passe-bande avec une fréquence de coupure basse à 1 MHz et fréquence de coupure haute à 3 MHz. Le filtre 2 est d'un degré double par rapport au 1 pour obtenir une pente semblable en amplitude. Cela se ressent dans le retard de groupe par une augmentation.

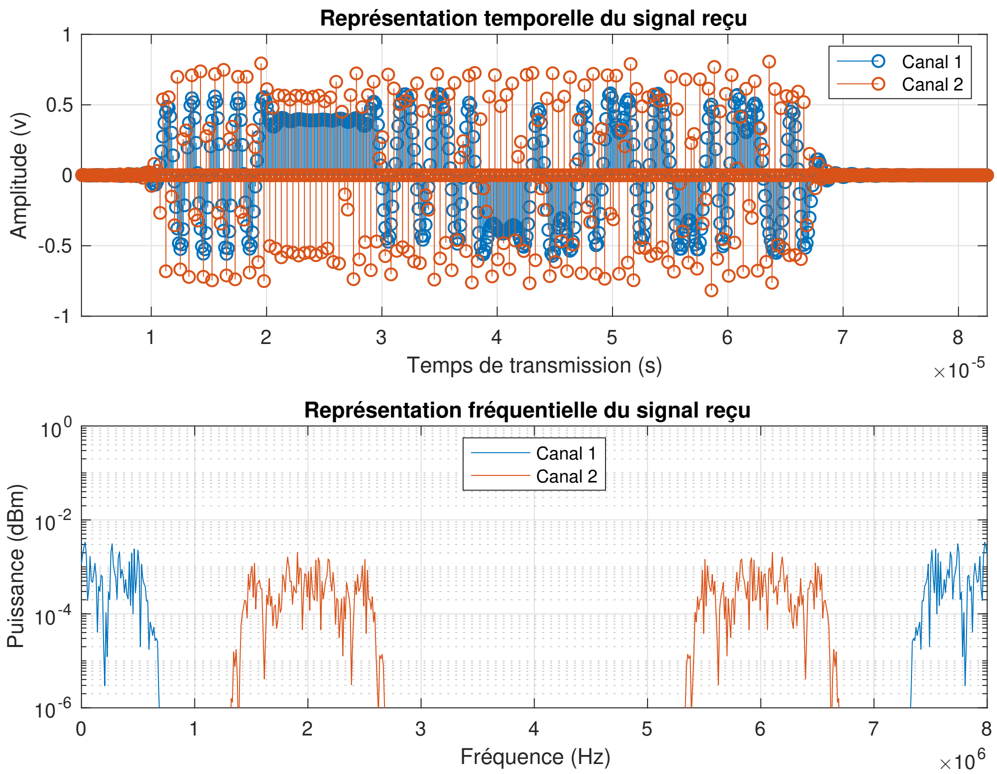

Figure 6 : Signaux de 50 bits chacun envoyés sur 2 canaux fréquentiels au rythme de 10 bit/s avec une fréquence d'échantillonage de 80 Hz.

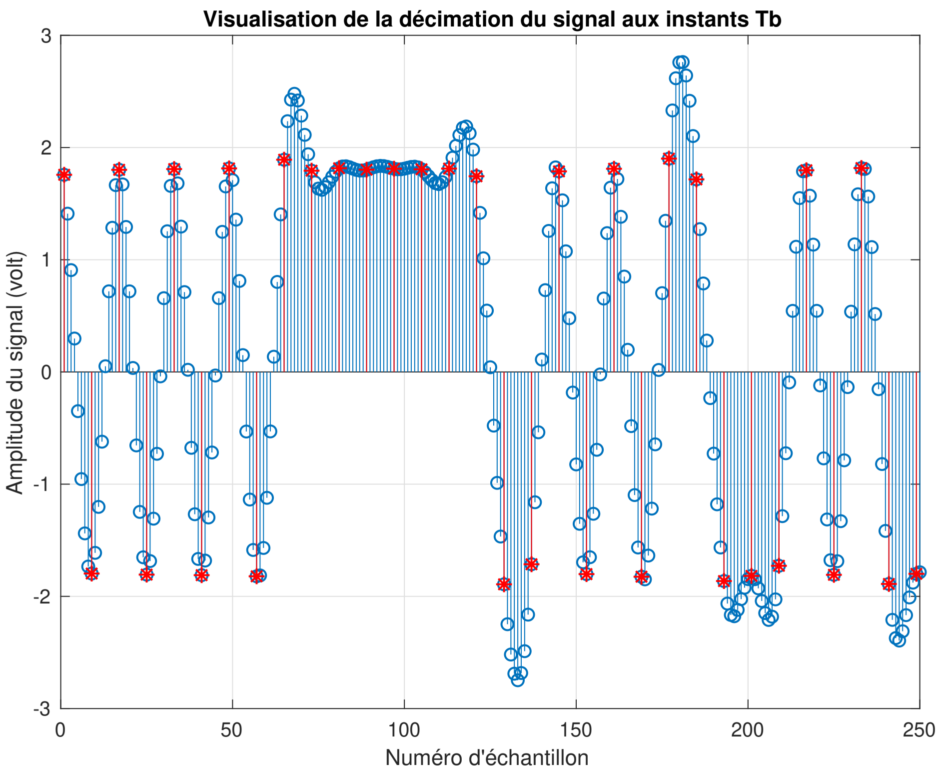

Figure 7 : Visualisation de la capture d'échantillons pour le décideur.

2 Performances

Dans le cas d'un code PAM-2, la relation E_b/N_0 au SNR s'écrit [1]

$$\frac{E_b}{N_0} + 10\log_{10}\left(\frac{R_b}{B}\right) = \mathrm{SNR_{dB}} \tag{7}$$

Où sont définies les variables

E_b	l'énergie d'un bit
N_0	la puissance du bruit pour un spectre partant de 0 à l'infini
R_b	le taux de transmission des bits
B	la largueur de bande utilisée
SNR	le rapport de puissance $P_{\text{signal}}/P_{\text{noise}}$ en decibels

Puisque dans notre cas $R_b = B$, cela implique que $\log_{10}(R_b/B) = 0$, et nous pouvons l'ôter de l'équation pour ne laisser que la relation

$$\frac{E_b}{N_0} = \mathrm{SNR_{dB}} \tag{8}$$

Cette relation nous facilite grandement le calcul de E_b/N_0, nécessaire pour réaliser un diagramme BER.

Le diagramme BER, ou diagramme Bit-Error-Rate, est un digramme présentant le rapport du nombre de bits erronés reçus dans un interval de temps étudié, en fonction du rapport d'énergie d'un bit sur le bruit E_b/N_0. Pour dessiner ce diagramme, nous désirons fixer un E_b/N_0 en abscisse, et reporter son taux d'erreur en l'ordonnée. Mais encore faut-il savoir fixer un E_b/N_0. C'est ici que l'équation (8) vient grandement nous aider car elle nous permet de fixer un E_b/N_0 en modifiant l'amplitude du bruit simulé.

Depuis la formule du SNR qui <u>n'est pas</u> exprimée en decibels

$$\mathrm{SNR} = \frac{P_{\text{signal}}}{P_{\text{noise}}} = \frac{\sum_{n=0}^{N} \text{signal}[n]^2}{\sum_{n=0}^{N} \text{noise}[n]^2} \tag{9}$$

Et connaissant le rapport de puissance que nous souhaitons obtenir

$$10^{(\text{dB}/10)} = \mathrm{SNR} \tag{10}$$

Nous pouvons calculer une ratio qui nous amène au niveau de bruit désiré

$$\text{ratio} = \frac{\mathrm{SNR}}{10^{(\text{dB}/10)}} \tag{11}$$

Et nous normalisons le bruit en le multipliant par la racine de ce ratio pour arriver au SNR voulu

$$\text{noise leveled} = \text{noise} \times \sqrt{\text{ratio}} \tag{12}$$

Cette operation s'effectue dans le fichier `channel.m`.

Nous pouvons dès lors tracer le diagramme BER de notre méthode de transmission en lui adjoignant le BER mathématiquement optimal d'une modulation PAM-2 défini par l'équation 13. Cette relation est donnée et acceptée.

$$\mathrm{BER_{PAM\text{-}2}} = \frac{1}{2}\,\mathrm{erfc}\left(\sqrt{\frac{E_b}{N_0}}\right) \tag{13}$$

La figure 8 présente le diagramme BER mathématiquement optimal calculé depuis la fonction d'erreur 13, et le diagramme BER de notre projet évalué dans MATLAB.

3 Problèmes connus

Conclusion

1. https://nl.mathworks.com/help/comm/ug/awgn-channel.html

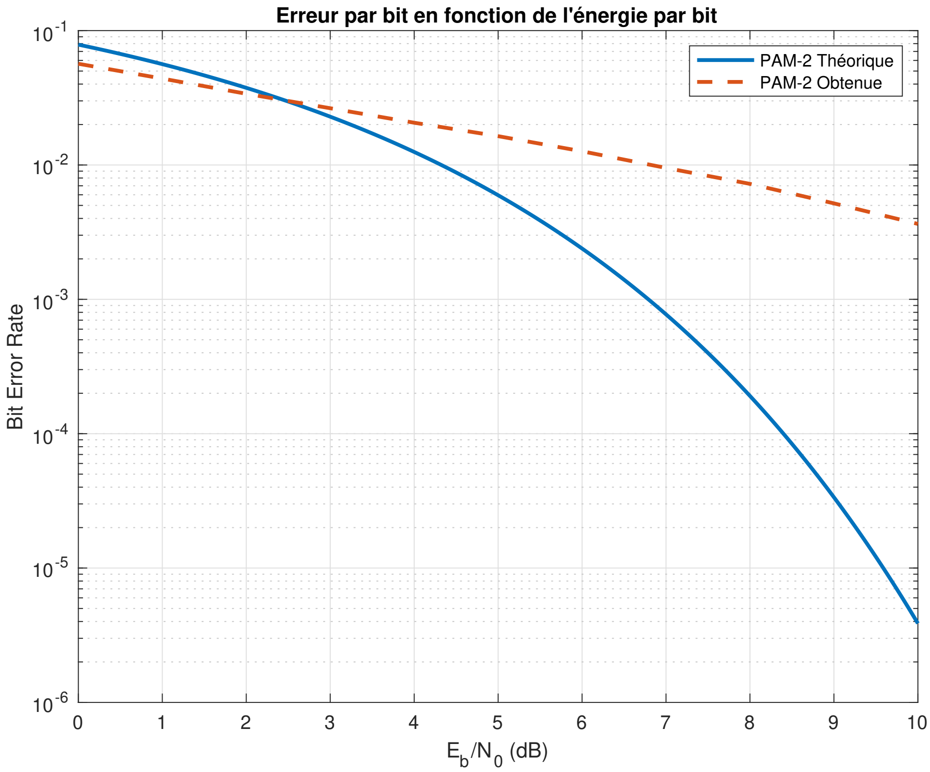

Figure 8 : Courbe du taux d'erreur par bit BER en fonction de l'énergie par bit E_b/N_0, aussi appelé SNR par bit. La figure reprend la courbe optimale mathématiquement et la courbe évaluée grâce à MATLAB. La courbe évaluée se comporte étonnement en faisant mieux que l'optimum théorique avant 2,5 dB et pire après.

	Canal 1	**Canal 2**	**Canal 3**	**Canal 4**
1 actif	0.4310	n/a	n/a	n/a
2 actifs	0.4101	0.4456	n/a	n/a
3 actifs	0.3918	0.4706	0.4703	n/a
4 actifs	0.3762	0.5279	0.4161	0.5271

Table 1 : Taux d'erreur moyen variant selon le nombre de canaux actifs.

A Figures arbitraires

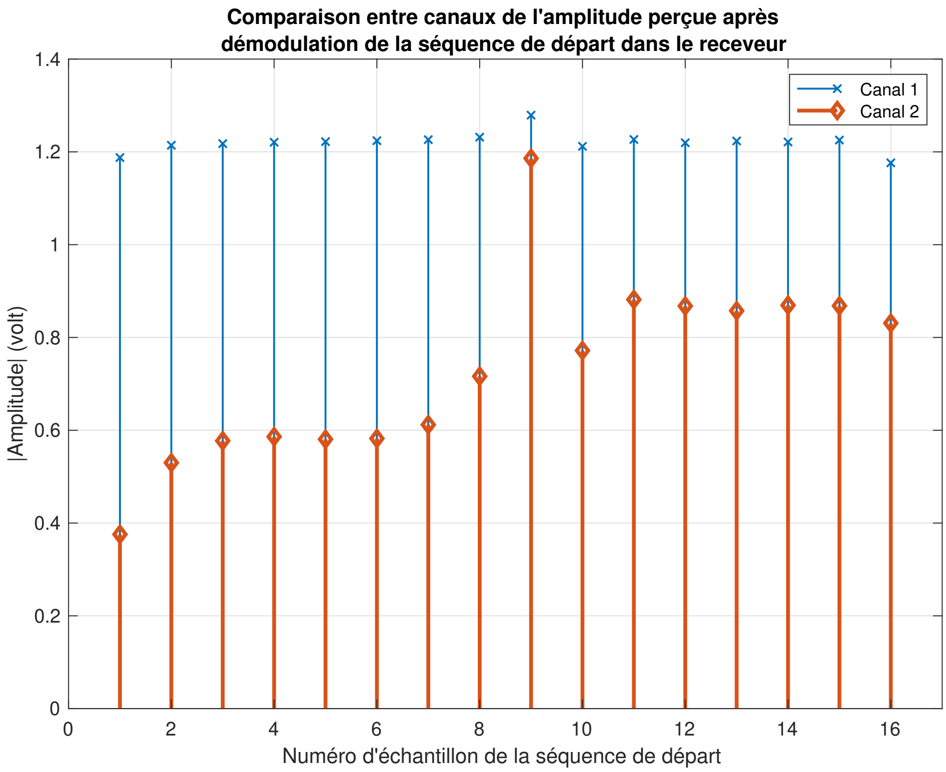

Figure 9 : Atténuation de l'amplitude de la séquence de départ perçue dans le receveur.

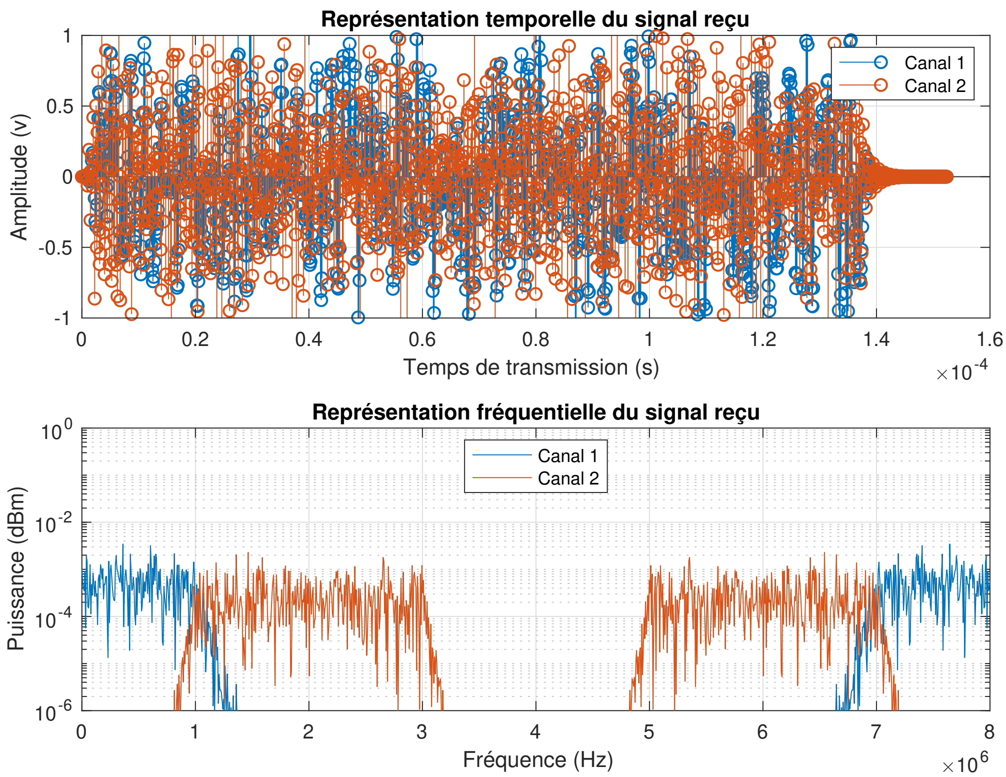

Figure 10 : Sortie des filtres dans le receveur servant à séparer les canaux fréquentiels, dans le cas d'un signal fortement bruité. Cette figure permet de montrer les limites de bande passante des filtres.

B Fichiers sources

B.1 main.m

```
% This work is licensed under the Creative Commons Attribution 4.0
% International License. To view a copy of this license, visit
% http://creativecommons.org/licenses/by/4.0/ or send a letter to
% Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.
%clear, close all

parameters
% generate data and send it
sender
% add noise and delay
channel
% filter data and read it
receiver

% compare the sent signal with the received one
figure
subplot(2,1,1)
stem(linspace(0, len1*Tn, len1), s1(:,1));
title('Signal normalise envoye par l''emetteur')
xlabel('Temps de transmission (s)')
ylabel('Amplitude du signal')
grid

subplot(2,1,2)
len3 = size(s2,1);
stem(linspace(0, len3*Tn, len3), s2(:,1), 'Color', [0.85 0.33 0.1]);
title('Signal recompose dans le receveur')
xlabel('Temps de transmission (s)')
ylabel('Amplitude du signal')
grid

% report QS
disp("Taux d'erreurs :")
errorRate = sum(xor(x, decoded))/size(x,1);
disp(errorRate)
```

B.2 parameters.m

```
% This work is licensed under the Creative Commons Attribution 4.0
% International License. To view a copy of this license, visit
% http://creativecommons.org/licenses/by/4.0/ or send a letter to
% Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

codesymbol = @(x)x.*2-1;

% System
N = 2;        % available channels
M = 1e6;      % message size (bits)

% Sender
R = 1e6;      % bit rate
Tb = 1/R;     % bit duration
roll = 0.40;  % rolloff factor
beta = 4*N;   % upsampling factor
Tn = Tb/beta; % upsample sampling rate
span = 20;    % rcos span for thinner bandwidth consumption
pwr = 200;     % channel power in mW

% Channel
shift = 4;    % samples delay
variance = 1; % noise variance
```

```
% Receiver
impulseL = 128;
startSeq = [1 0 1 0 1 0 1 0 ... % test the channel response
            1 1 1 1 1 1 1 1];   % set an unique sequence
```

B.3 sender.m

```
% This work is licensed under the Creative Commons Attribution 4.0
% International License. To view a copy of this license, visit
% http://creativecommons.org/licenses/by/4.0/ or send a letter to
% Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

x = randi([0 1], M, N);
% append the start sequence
x = [startSeq'*ones(1, N); x];
a = codesymbol(x);
% shape to impulse
rcos = rcosdesign(roll, span, beta);
a = upsample(a, beta);
s1 = conv2(rcos, 1, a);
len1 = size(s1, 1);

% carrier frequencies
carfreq = (0:N-1)'*2/Tb;

% modulate by carriers
t = (0:Tn:(len1-1)*Tn)'*ones(1,N);
s1High = s1.*cos(2*pi*carfreq'.*t);

% normalise power to 'pwr' mW
power = sum(s1High.^2, 1)/len1;
ratio = (pwr*1e-3)./power;
s1High = s1High.*sqrt(ratio);

% sum all channels before transmission
data = sum(s1High, 2);

% plot impulsions
iX = linspace(0, span/1e2, 1e2*span+1);
iY = rcosdesign(roll, span, 1e2);
plot(iX, iY' * ones(1, N) .* ...
     cos(carfreq*linspace(0, 2*pi, span*1e2+1))')
ylim([-max(iY)*1.1 +max(iY)*1.1])
title("Representation temporelle des impulsions utilisees")
ylabel("Coefficient d'amplitude"), xlabel("Temps (s)")
legend(strcat("Canal ", num2str((1:N)')))
grid
clear iX iY

% plot visual representation of the transmission
figure
subplot(2,1,1)
stem(linspace(0, len1*Tn, len1), s1High)
title('Representation temporelle du signal envoye')
ylabel('Amplitude (v)'), xlabel('Times (s)')
legend(strcat("Canal ", num2str((1:N)')), 'Location', 'NorthEast')
grid

subplot(2,1,2)
semilogy(linspace(0, 1/Tn-1, len1), abs(fft(s1High/len1)).^2)
ylim([10^-6 10^0])
title('Representation frequentielle du signal envoye')
ylabel('Puissance (dBm)'), xlabel('Frequency (Hz)')
legend(strcat("Canal ", num2str((1:N)')), 'Location', 'North')
grid
```

B.4 channel.m

```
% This work is licensed under the Creative Commons Attribution 4.0
% International License. To view a copy of this license, visit
% http://creativecommons.org/licenses/by/4.0/ or send a letter to
% Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

% gaussian noise
noise_1 = randn([numel(data) 1]);
[bf,af] = butter(1, 0.99);
noise_f = ifft(freqz(bf, af, impulseL, 'whole', 1/Tn));
noise_2 = conv(noise_f, noise_1);
noise_2 = noise_2(1:end-impulseL+1);

if exist('forcedPwr', 'var')
    % for fine-grained control of the noise in diagram.m
    power_data = sum(data.^2);
    power_noise = sum(noise_2.^2);
    ratio = (power_data/power_noise)/10^(forcedPwr/10);
    noise_2 = noise_2.*sqrt(ratio);
    alpha = 1; std_dev = 1;
    ebn0(uniqIDX) = snr(data, noise_2);
else
    % damping factor; between 0.60<=x<=0.90
    alpha = (0.90-0.60)*rand([1 1])+0.60;
    % increase noise with variance
    std_dev = sqrt(variance);
end

data = alpha*data+std_dev*noise_2;
data = [zeros(shift,1); data];
```

B.5 receiver.m

```
% This work is licensed under the Creative Commons Attribution 4.0
% International License. To view a copy of this license, visit
% http://creativecommons.org/licenses/by/4.0/ or send a letter to
% Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

% calculate the bandwidth limits for each channel
cutoff = [carfreq-1/Tb carfreq+1/Tb]*2*Tn;
% pre-allocate filters matrix
H = zeros(impulseL, N);

% first channel lowpass
[bf,af] = butter(10, cutoff(1,2));
H(:,1) = ifft(freqz(bf, af, impulseL, 'whole', 1/Tn));

% others channels bandpass
for n = 2:N
    [bf,af] = butter(10, [cutoff(n,1) cutoff(n,2)]);
    H(:,n) = ifft(freqz(bf, af, impulseL, 'whole', 1/Tn));
end

% separate channels
s2High = conv2(data, 1, H);
len2 = size(s2High,1);

% demodulate
t = (0:Tn:(len2-1)*Tn)'*ones(1,N);
s2 = s2High.*cos(2*pi*carfreq'.*t);
s2(:,1) = s2High(:,1);
for n = 2:N
    [bf,af] = butter(5, carfreq(n)*2*Tn);
    impulse = ifft(freqz(bf, af, impulseL, 'whole', 1/Tn));
    s2(:,n) = conv(s2(:,n), impulse(1:+1:end), 'same'); % forward
    s2(:,n) = conv(s2(:,n), impulse(end:-1:1), 'same'); % backward
```

```
end

% filter the canal noise with the adequate filter
s2 = conv2(rcos, 1, s2);
% find filters delay
[~,i] = max(H);
% compensate the start trame
s2t = s2(span*beta+i+shift-1:end, :);
% generate the index vector
s2i = 1:beta:beta*size(x,1);
% extract the values at index
decoded = s2t(s2i,:);
% quantize the extracted values
decoded = decoded>0;

% hit markers *PEW* *PEW*
figure, hold on
stem(s2t(:,1))
stem(s2i, s2t(s2i,1), 'r*', 'MarkerSize', 8.0)
grid, hold off

% plot visual representation of the transmission
figure
subplot(2,1,1)
stem(linspace(0, len2*Tn, len2), s2High)
title('Representation temporelle du signal recu')
ylabel('Amplitude (v)'), xlabel('Times (s)')
legend(strcat("Canal ", num2str((1:N)')), 'Location', 'NorthEast')
grid

subplot(2,1,2)
semilogy(linspace(0, 1/Tn-1, len2), abs(fft(s2High/len2)).^2)
ylim([10^-6 10^0])
title('Representation frequentielle du signal recu')
ylabel('Puissance (dBm)'), xlabel('Frequency (Hz)')
legend(strcat("Canal ", num2str((1:N)')), 'Location', 'North')
grid
```

B.6 filters.m

```
close all

impulseL = 512;

gd = zeros(impulseL, N);
hold on

% first channel lowpass
[tmp1,tmp2] = butter(10, cutoff(1,2));
gd(:,1) = grpdelay(tmp1, tmp2, impulseL, 'whole', 1/Tn);
[hf,ff] = freqz(tmp1, tmp2, impulseL, 'whole', 1/Tn);
plot(ff(1:ceil(end/2)), ...
    20*log10(abs(hf(1:ceil(end/2)))));

% others channels bandpass
for n = 2:N
    [tmp1,tmp2] = butter(10, [cutoff(n,1) cutoff(n,2)]);
    gd(:,n) = grpdelay(tmp1, tmp2, impulseL, 'whole', 1/Tn);
    [hf,ff] = freqz(tmp1, tmp2, impulseL, 'whole', 1/Tn);
    plot(ff(1:ceil(end/2)), ...
        20*log10(abs(hf(1:ceil(end/2)))));
end

xlim([0 75]);
xlabel('Frequence (Hz)')
ylabel('Amplitude (dB)')
grid, hold off

```

```
figure, plot(ff,gd*Tn), grid
xlabel('Frequence (Hz)')
ylabel('Samples (sample x rad)')
```

B.7 diagram.m

```
% This work is licensed under the Creative Commons Attribution 4.0
% International License. To view a copy of this license, visit
% http://creativecommons.org/licenses/by/4.0/ or send a letter to
% Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

% 2-PAM best case
t = linspace(0, 10, 1e3);
y = 1/2*erfc(sqrt(10.^(t/10)));
semilogy(t, y)
grid

% our case
dB2test = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
tries = numel(dB2test);
BER = zeros([1 tries]);
ebn0 = zeros([1 tries]);

for uniqIDX = 1:tries
    forcedPwr = dB2test(uniqIDX);
    main;
    BER(uniqIDX) = sum(errorRate)/N;
end

hold on, semilogy(ebn0, BER), hold off
```