Projet :
Réalisation d'un modem de fréquence selon la recommandation V21 de l'Union Internationale des Télécommunications

Table des matières

Table des figures

Liste des Codes

1. Introduction

1.1. But

Le but de ce projet est de construire numériquement un modem (modulateur-démodulateur). A fin de tester nos compétences acquises en traitement du signal numérique, nous devons coder un modem de fréquence en matlab. Un signal numérique binaire est transmis via un canal de transmission virtuel après avoir été modulé en fréquence. Lors de cette transmission un bruit est ajouté. Le but est donc de démoduler le signal en sortie du canal, et de le traiter afin de retrouver le message initial, malgré le bruit.

Dans une première partie le traitement du bruit se fera à l'aide d'un filtre, puis nous implémenterons la recommandation V21 de l'Union Internationale des Télécommunications.

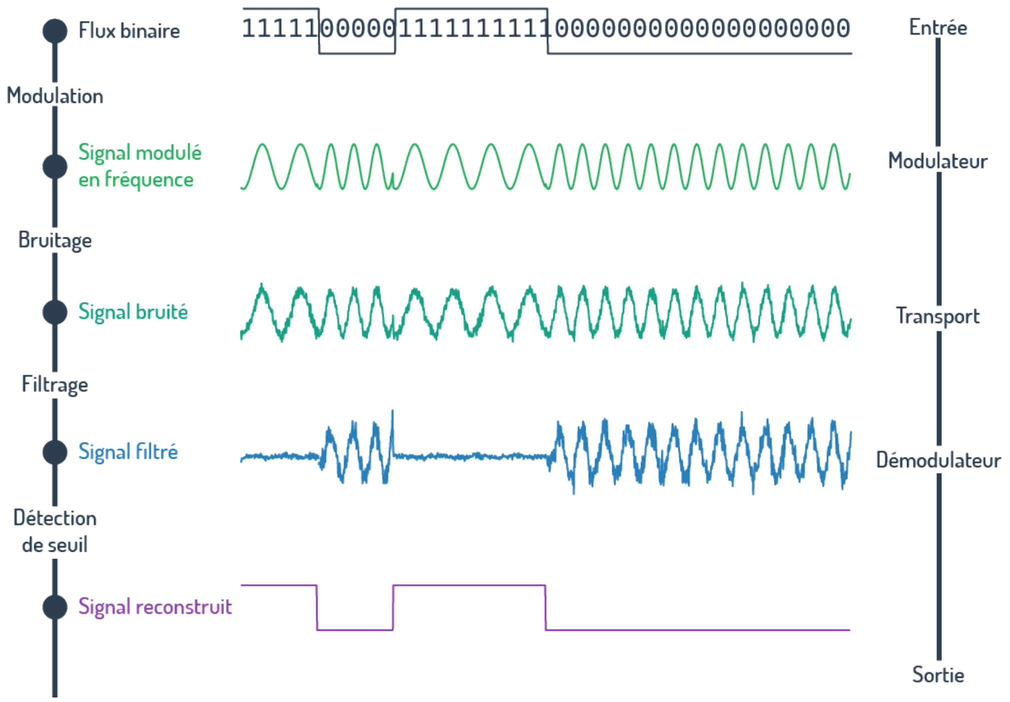


Figure 1 – Principe du projet

1.2. Auteurs

Nous sommes un binome d'élèves de l'école d'ingénieur ENSEEIHT à Toulouse (École nationale supérieure d'électrotechnique, d'électronique, d'informatique, d'hydraulique et des télécommunications) en première année, en filière Sciences du Numérique (SN) du **groupe L** :

- Félix PARAIN
- Sébastien PONT

2. Modem de fréquence - Démodulation par filtrage

2.1. Construction du signal modulé en fréquence

2.1.1. Génération du signal NRZ

Dans un premier temps, nous générons aléatoirement un signal binaire composé de `Nb_bit = 100` bits.

```
Nb_bit = 100;
Donnee = randi([0, 1], 1, Nb_bit);
```

Code 1 – Génération aléatoire d'un signal binaire

Selon la norme V21, nous voulons transmettre le signal à `Debit = 300` $bits \cdot s^{-1}$. De plus, nous avons une fréquence d'échantillonnage définie à `Fe = 48000` Hz. Nous en dédsuisons le nombre `Ns` d'échantillons à générer par bit (0 ou 1).

```
Debit = 300;
Fe = 48000; Te = 1 / Fe;
Ts = 1 / Debit;
Ns = round(Ts / Te);
```

Code 2 – Détermination du nombre N_s d'échantillons

On peut ainsi générer le signal NRZ, et le tracer (Figure 2).

```
NRZ = kron(Donnee, ones(1, Ns));
t = (0:Te:(Ns * Nb_bit - 1) * Te);
```

Code 3 – Génération du signal NRZ

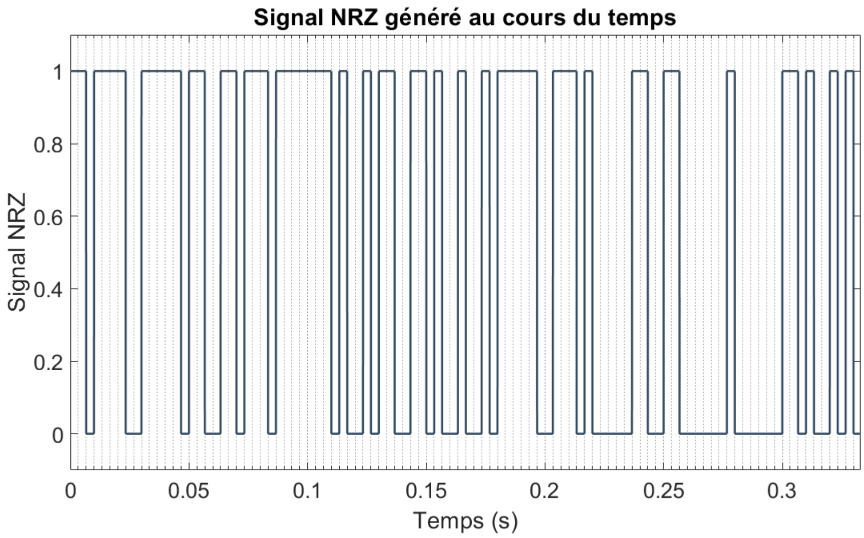


Figure 2 – Signal généré NRZ

Grâce à la grille, on peut compter par exemple 15 bits en 0.05 secondes, soit un débit de $15 \times \frac{1}{0.05} = 300 bit \cdot s^{-1}$. Nous pouvons de plus estimer la densité spectrale de ce signal NRZ en utilisant la fonction `periodogram` de matlab.

```
periodogram(NRZ, [], length(NRZ),1/Ts)
```

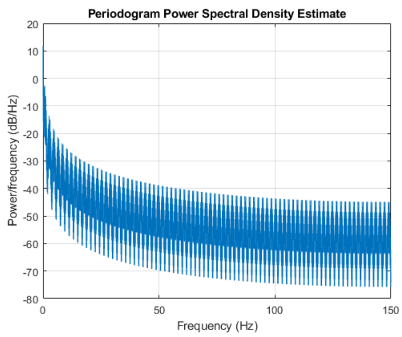


Figure 3 – Periodogramme de NRZ

2.1.2. Génération du signal modulé en fréquence

Nous allons maintenant générer le signal modulé en fréquence selon l'équation suivante :

$$x(t) = (1 - NRZ(t)) \times cos(2\pi F_0 t + \phi_0) + NRZ(t) \times cos(2\pi F_1 t + \phi_1)$$

Comme nous devrons d'abord faire un filtrage (après bruitage), pour l'instant, les fréquences F_0 et F_1 sont éloignées l'une de l'autre :

$$\begin{cases} F_0 & = & 6000Hz \\ F_1 & = & 2000Hz \end{cases}$$

```
F0 = 6000; F1 = 2000;
phi0 = rand * 2 * pi;
phi1 = rand * 2 * pi;
x = (1 - NRZ) .* cos(2 * pi * F0 * t + phi0) + NRZ .* cos(2 * pi * F1 * t + phi1);
plot(t, x);
```

Code 4 – Génération du signal modulé

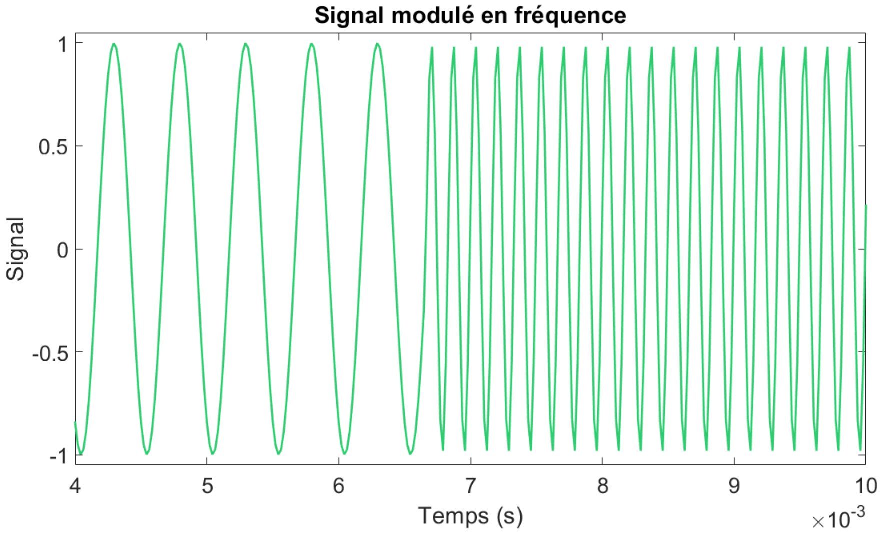


FIGURE 4 – Signal modulé en fréquence

Sur la Figure 4 nous avons zoomé pour bien voir la modulation en fréquence : sur l'intervalle $[4 \cdot 10^{-3}; 6,6 \cdot 10^{-3}]$, nous retrouvons la fréquence F_1, et sur l'intervalle $[6,6 \cdot 10^{-3}; 10 \cdot 10^{-3}]$, la fréquence F_0. Calculons la densité spectrale de puissance théorique de ce signal.

- **Type de signal.** ϕ_0 et ϕ_1 sont des variables aléatoires indépendantes. On se penche donc vers un signal aléatoire. Montrons qu'il est stationnaire :

$$\left\{ \begin{array}{rcl} m_x & = & E[x(t)] \\ R_x(\tau) & = & E[x(t)x^*(t-\tau)] \end{array} \right.$$

Notons que $NRZ(t)$, ϕ_0 et ϕ_1 sont indépendantes, donc :

$$\left\{ \begin{array}{rcl} m_x & = & E[cos(2\pi F_0 t + \phi_0)] - E[NRZ(t)] \times E[cos(2\pi F_0 t + \phi_0)] + E[NRZ(t)] \times E[cos(2\pi F_1 t + \phi_1)] \\ R_x(\tau) & = & \ldots \end{array} \right.$$

$$\iff \left\{ \begin{array}{rcl} m_x & = & 0 \\ R_x(\tau) & = & R_{NRZ}(\tau) \times \frac{1}{2}(cos(2\pi F_1 \tau) - cos(2\pi F_0 \tau)) \end{array} \right.$$

Ainsi, m_x et R_x ne dépendent pas du temps. Alors, $x(t)$ est stationnaire.

- **Densité spectrale de puissance.**

$$S_x(f) = TF[R_x(\tau)]$$

$$S_x(f) = \frac{1}{4}(S_{NRZ}(f - f_1) + S_{NRZ}(f + f_1) - S_{NRZ}(f - f_0) - S_{NRZ}(f + f_0))$$

De la même manière que nous avons estimé la densité spectrale du signal NRZ, nous calculons celle du signal $x(t)$.

2.2. Canal de transmission à bruit additif, blanc et Gaussien

Nous ajoutons maintenant du bruit blanc gaussien au signal modulé. Rappelons qu'un bruit blanc est un bruit de moyenne nulle et un bruit gaussien est un bruit distribué selon une loi normale. Pour cela, nous ajoutons une pertubation de puissance déduite P_b en fonction du rapport signal sur bruit $SRN_d b$ que nous voulons :

$$P_b = \frac{P_x}{10^{\frac{SNR_{dB}}{10}}}$$

Nous calculons d'abord la puissance P_x du signal $x(t)$ (ligne 1), puis la puissance P_b (ligne 3). Nous ajoutons enfin le bruit au signal $x(t)$ (ligne 4).

```
Px = mean(abs(x).^2);           % puissance du signal
SNRdb = 50;                     % rapport signal sur bruit
Pb = Px / (10^(SNRdb / 10));    % puissance du bruit
x = x + Pb * randn(1, Ns * Nb_bit);
```

Code 5 – Ajout de bruit blanc gaussien

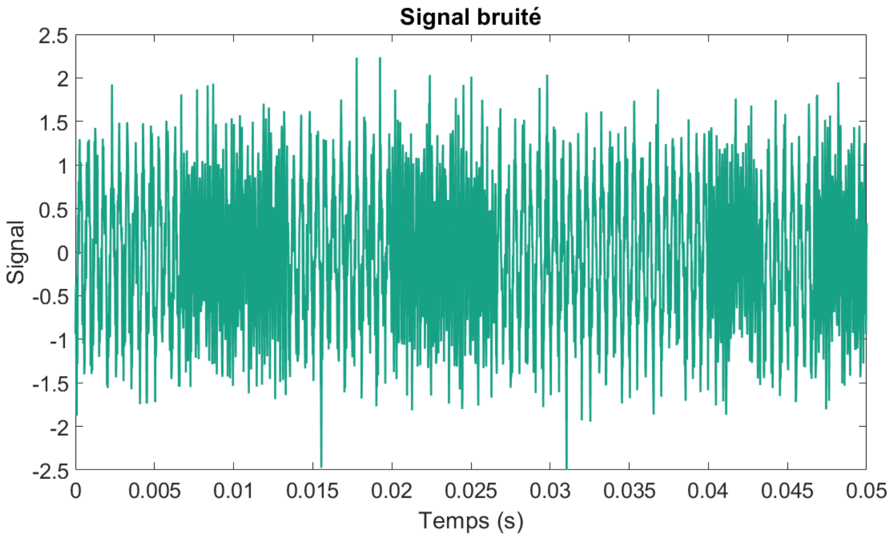


FIGURE 5 – Signal bruité

La Figure 5 représente le signal bruité. La modulation en fréquence est bien visible.

2.3. Démodulation par filtrage

Dans un premier temps, pour démoduler le signal, nous allons le filtrer. En effet, nous savons que les bits 0 sont modulés à une fréquence $F_0 = 6000Hz$, et les bits 1 sont modulés à une fréquence $F_1 = 2000Hz$. Ainsi, en implémentant un filtre passe-bas pour commencer, nous filtrerons tous les bits 0, il ne restera plus que les bits 1, et nous saurons alors reconstruire le signal. Nous allons faire pareil avec un filtre passe-haut, qui ne gardera que les bits 0.

Pour cela nous définissons d'abord une fréquence de coupure `Fc = 4500` Hz, l'`ordre = 101` du filtre, et le `retard`

```
Fc = 4500;
Ordre = 101;
Retard = (Ordre - 1) / 2;
```

2.3.1. Filtre passe-bas

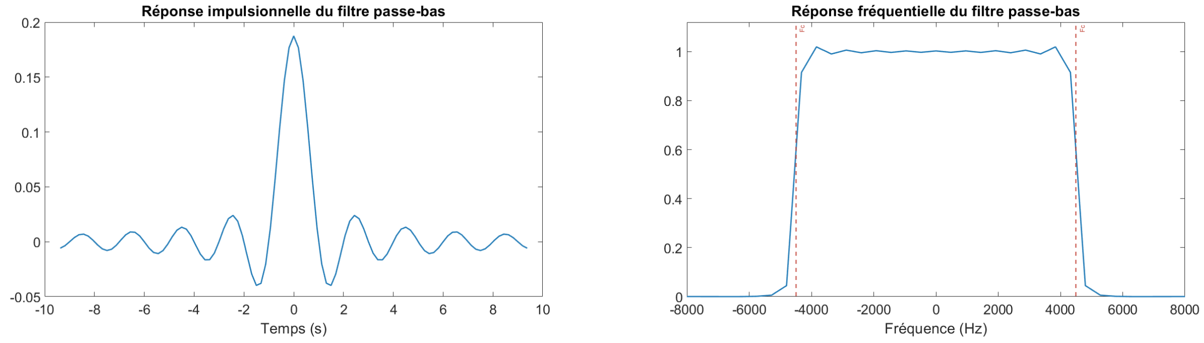
Nous choisissons un simple sinus cardinal comme filtre.

```
1 h_bas = 2 * Fc * Te * sinc(2 * Fc * Te * (-Retard:Retard));
2 H_bas = fft(h_bas);
3 y_bas_retarde = filter(h_bas, 1, [x zeros(1, Retard)]);
4 y_bas = y_bas_retarde(Retard + 1:end);
```

Code 6 – Filtre passe-bas

La ligne 4 permet de ne pas perdre de données. La ligne 5 permet de supprimer le retard. Voici les résultats que nous avons obtenus :

- **Réponses du filtre.** Voici les réponses fréquentielle et impultionnelle du filtre passe-bas. Notez que les lignes en pointillés sur la Figure 6.b marquent la fréquence de coupure $F_c = 4500Hz$.



(a) Réponse impultionnelle du filtre passe-bas (b) Réponse fréquentielle du filtre passe-bas

FIGURE 6 – Réponses du filtre passe-bas

❑ **Densité spectrale de puissance du signal non filtré.**

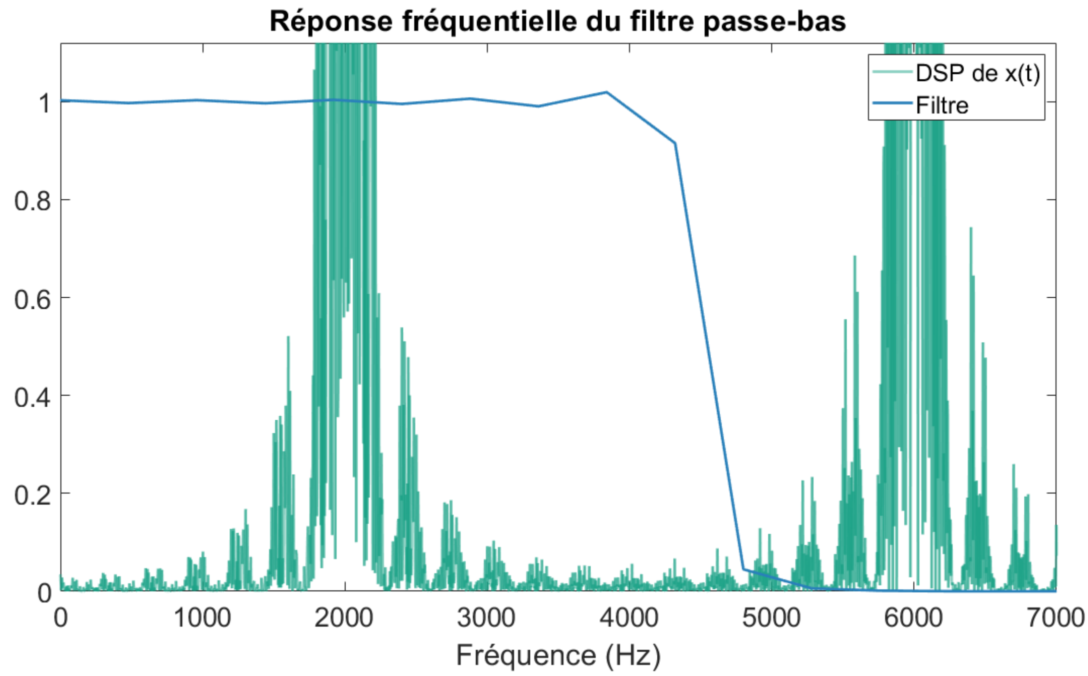
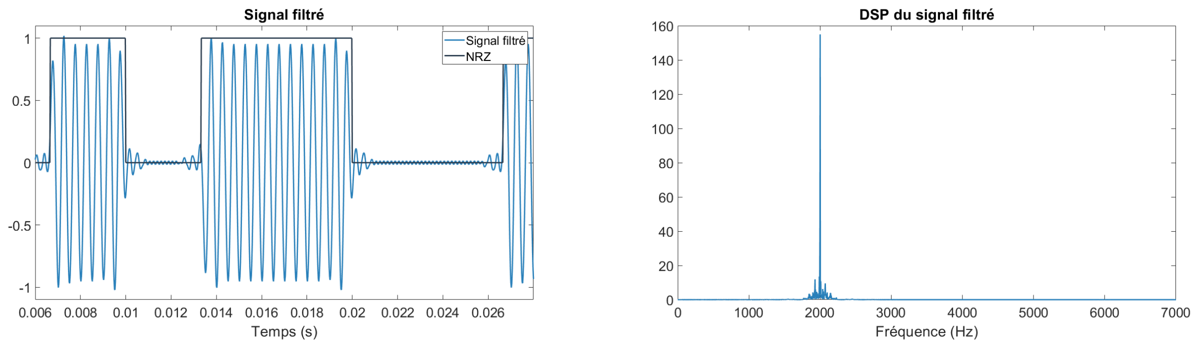


FIGURE 7 – Densité spectrale de puissance du signal non filtré (passe-bas)

Nous pouvons voir que le filtre coupe bien les hautes fréquences, notamment les fréquences autour de 6000 Hz, représentant les bits 0. La démodulation va donc bien petre possible.

❑ **Signal filtré.**



(a) Signal filtré (b) DSP du signal filtré

FIGURE 8 – Signal filtré (passe-bas)

Nous remarquons que le signal a été filtré comme voulu :
- les bits 0 sont filtrés (Figure 8.a)
- les hautes fréquences ont été coupées, il ne reste plus que les fréquences inférieures à 4500Hz (Figure 8.b)

2.3.2. Filtre passe-haut

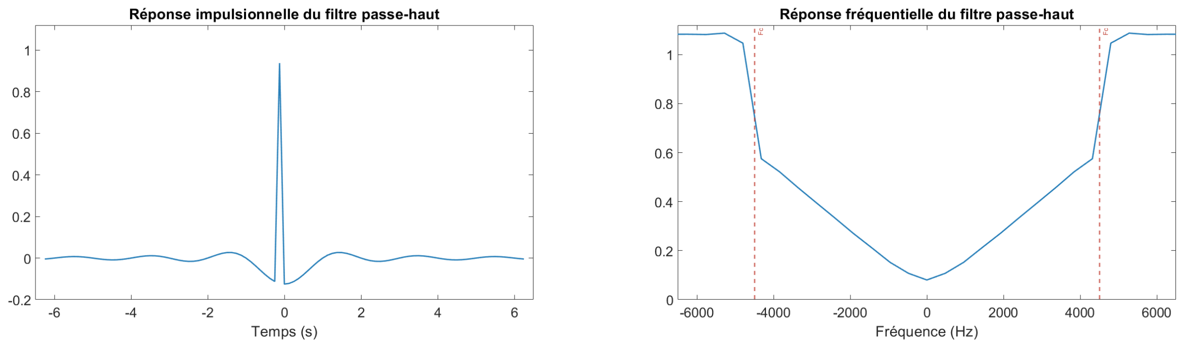
De manière similaire, nous implémentons le filtre passe-haut.

```
h_haut = -h_bas;
h_haut(Retard) = 1 - Fc / Fe;
H_haut = fft(h_haut);
y_haut_retarde = filter(h_haut, 1, [x zeros(1, Retard)]);
y_haut = y_haut_retarde(Retard + 1:end);
```

Code 7 – Filtre passe-haut

Et voici les résultats :

- **Réponses du filtre.** Voici les réponses fréquentielle et impultionnelle du filtre passe-haut. De même,les lignes en pointillés sur la Figure 9.b marquent la fréquence de coupure $F_c = 4500Hz$.



(a) Réponse impultionnelle du filtre passe-haut (b) Réponse fréquentielle du filtre passe-haut

FIGURE 9 – Réponses du filtre passe-haut

- **Densité spectrale de puissance du signal non filtré.**

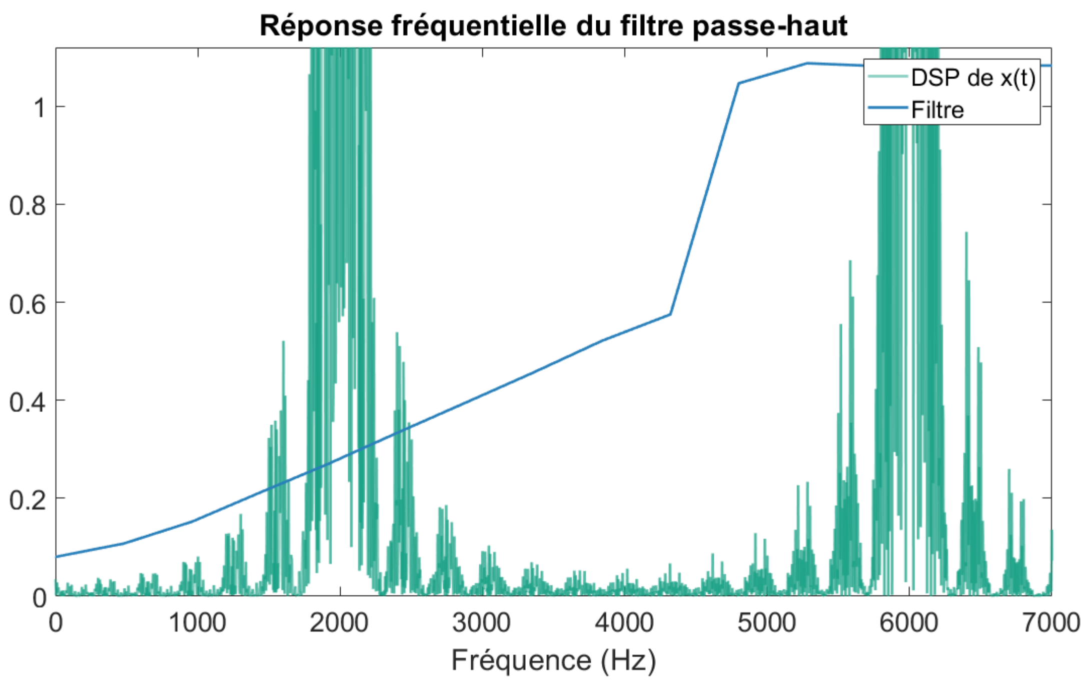
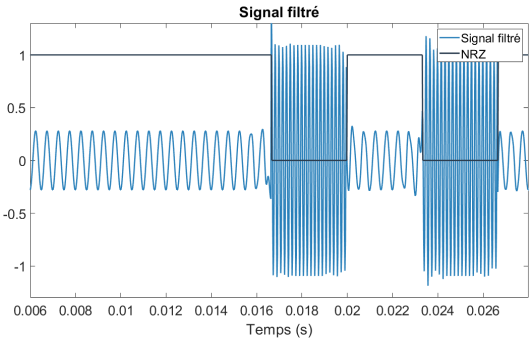


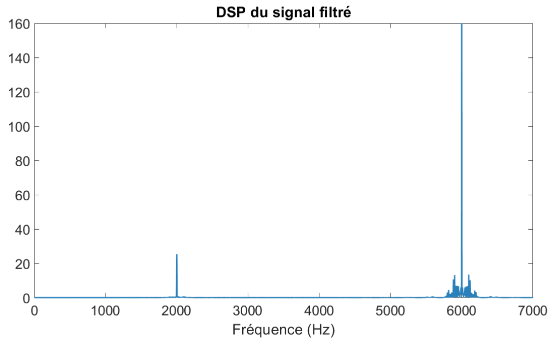
FIGURE 10 – Densité spectrale de puissance du signal non filtré (passe-haut)

On remarque cette fois-ci que le filtre ne permet pas de couper toutes les fréquences basses, la démodulation sera toujours possible, mais moins bien.

❏ **Signal filtré.**



(a) Signal filtré



(b) DSP du signal filtré

FIGURE 11 – Signal filtré (passe-haut)

On voit bien que tout n'est pas filtré, par exemple sur la Figure 11.a, on voit qu'il reste des fréquences autour de 2000 Hz. Mais les bits 1 ont une amplitude beaucoup plus faible que les bits 0 (Figure 11.a). La démodulation sera donc correcte.

2.3.3. Détection d'énergie

Il ne reste maintenant plus qu'à traduire le signal modulé en une suite de bits (0 et 1). Nous décidons de garder notre signal filtré par le passe-bas. Pour cela on utilise un détecteur d'énergie :

— sur la durée d'un bit T_s, on mesure l'énergie du signal.

— si cette énergie est supérieure à un seuil K, alors le bit est 1

Note : l'énergie d'un signal $X = \{x_i; i \in [\![1, N_s]\!]\}$ de N_s échantillons se calcule selon :

$$\sum_{n=1}^{N_s} x_n^2$$

```
K = 11; % seuil d'energie determine experimentalement
y_redecoupe = reshape(y_bas , Ns , Nb_bit);
X = sum(y_redecoupe.^2, 1);
Donnee_retrouve = X > K;
```

Code 8 – Détection d'énergie

De plus, puisque nous connaissons le signal avant perturbation, nous pouvons calculer le taux d'erreur, en comparant le signal initial, et le signal reconstruit :

```
Nb_erreur = sum(Donnee ~= Donnee_retrouve);
taux_erreur = Nb_erreur / Nb_bit;
```

Code 9 – Taux d'erreur

Nous obtenons ainsi un **taux d'erreur nul**.

2.3.4. Reconstitution de l'image codée

Nous pouvons maintenant décoder l'image codée pour deviner le lieu et le personnage :

```
load DonneesBinome1.mat;
Nb_bit = 84000;
Donnee = bits;
...
reconstitution_image(Donnee_retrouve);
```

Code 10 – Reconstitution de l'image

FIGURE 12 – Image reconstituée

Nous reconnaissons ainsi le graphe représentant **Charles Camichel** dans **la cours de l'école**.

2.3.5. Application à la recommandation V21

Si nous voulons respecter la recommandation V21, nous devons maintenant changer les fréquences associées aux bits 0 et 1 :

- $F_0 = 1\ 180$ Hz
- $F_1 = 980$ Hz

Nous nous doutons que les filtres implantés ne pourront plus distinguer les bits, car leur fréquences seront trop rapprochées. En effet, en changeant la fréquence de coupures, nous obtenons une taux d'erreur faible, mais non nul.

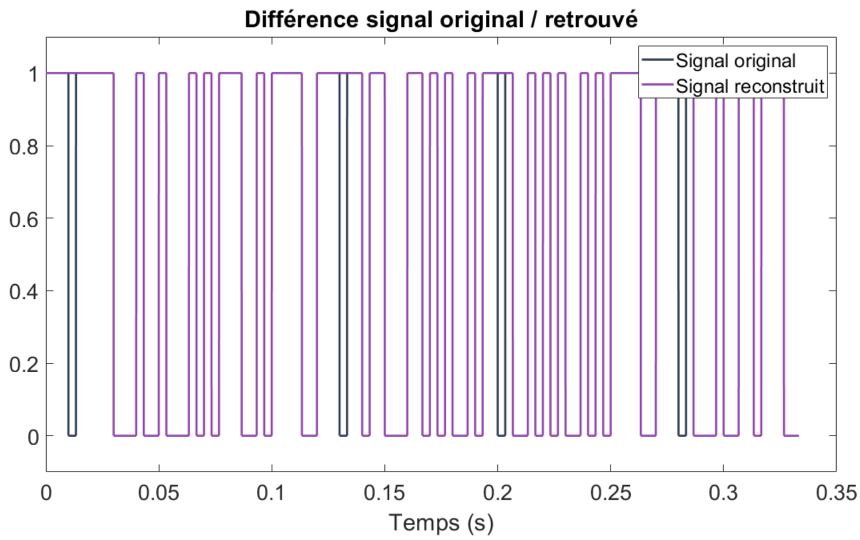


FIGURE 13 – Signal reconstitué pour un taux d'erreur de 4%

3. Modem de fréquence V21 - Démodulateur FSK

3.1. Synchronisation idéale

3.1.1. Présentation du récepteur

La Figure 14 illsutre le nouveau récepteur que nous allons implémenter dans un premier temps.

Pour l'instant nous supposons que les phases des cosinus du signal initial, avant transport, et celles du signal après transport, au niveau du récepteur sont les mêmes. Détaillons les calculs de ce récepteur :
Soit :

$$\begin{cases} I_{0,0} & = & \int_0^{T_s} cos(2\pi F_0 t + \phi_0)(cos(2\pi F_0 t + \phi_0)dt \\ I_{0,1} & = & \int_0^{T_s} cos(2\pi F_0 t + \phi_0)(cos(2\pi F_0 t + \phi_1)dt \\ I_{1,0} & = & \int_0^{T_s} cos(2\pi F_0 t + \phi_1)(cos(2\pi F_0 t + \phi_0)dt \\ I_{1,1} & = & \int_0^{T_s} cos(2\pi F_0 t + \phi_1)(cos(2\pi F_0 t + \phi_1)dt \end{cases}$$

De plus, on rappelle :

$$x(t) \quad = \quad (1 - NRZ(t)) \times cos(2\pi F_0 t + \phi_0) + NRZ(t) \times cos(2\pi F_1 t + \phi_1)$$

A l'étape 2.1 (Figure 14) on multiplie $x(t)$ par $cos(2\pi F_0 t)$. De même, à l'étape 32.2, on multiplie $x(t)$ par $cos(2\pi F_1 t)$. Puis, aux étapes 3.1 et 3.2, on calcule les intégrales des résultats des étapes précédentes. Enfin, à l'étape 4, on soustrait l'un à l'autre. Au final on trouve :

- Si $NRZ = 0$:

$$R = I_{0,1} - I_{0,0} \approx -\frac{T_s}{2} < 0$$

- Si $NRZ = 1$:

$$R = I_{1,1} - I_{1,0} \approx \frac{T_s}{2} > 0$$

Ainsi, si la sortie du récepteur est positive, le bit retrouvé est 1, sinon, le bit retrouvé est 0.

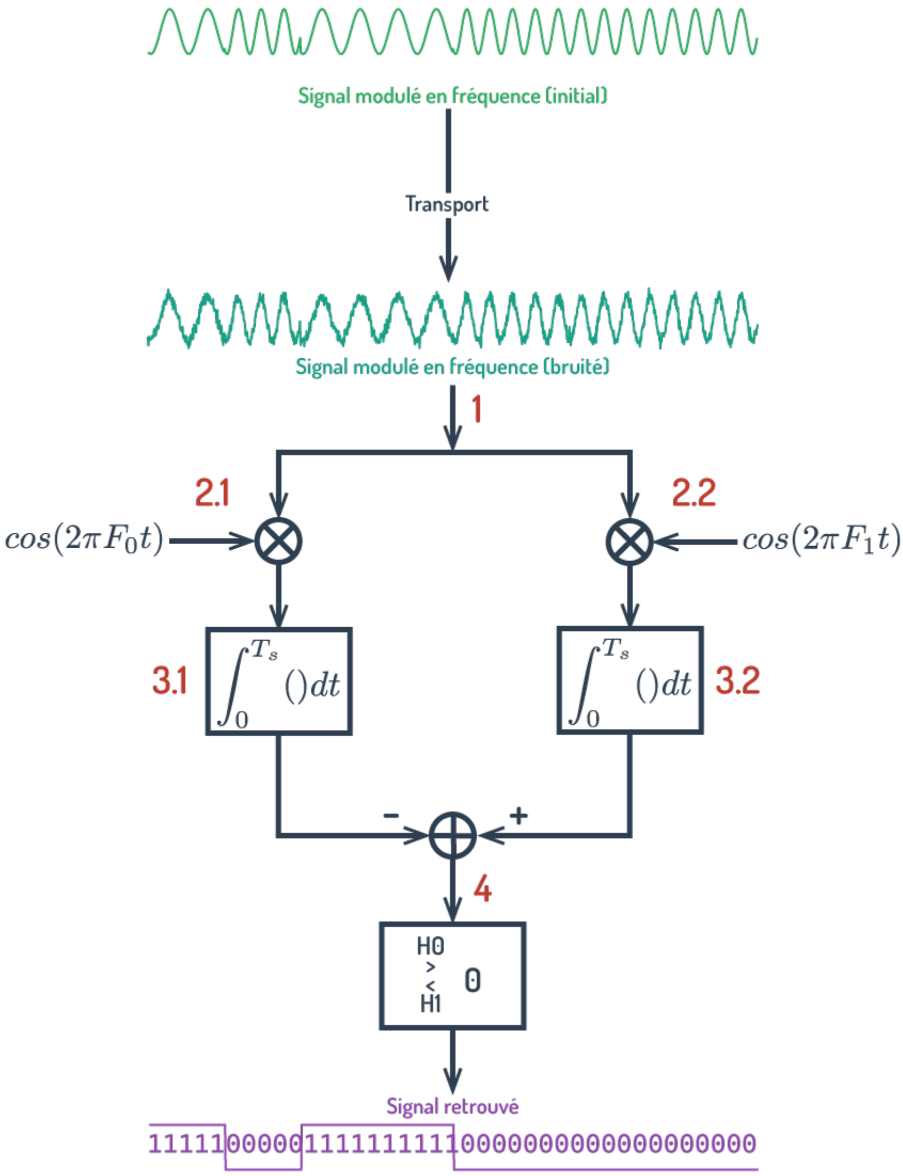


FIGURE 14 – Schéma de démodulation FSK avec synchronisation idéale

3.1.2. Implémentation du récepteur

```
x0 = x .* cos(2 * pi * F0 * t + phi0);
x1 = x .* cos(2 * pi * F1 * t + phi1);

x0_reshape = reshape(x0, Ns, Nb_bit);
X0 = sum(x0_reshape * Ts, 1);

x1_reshape = reshape(x1, Ns, Nb_bit);
X1 = sum(x1_reshape * Ts, 1);

X2 = X1 - X0;
Donnee_retrouve_2 = X2 > 0;

Nb_erreur_2 = sum(Donnee ~= Donnee_retrouve_2);
taux_erreur_2 = Nb_erreur_2 / Nb_bit
```

Code 11 – Récepteur V21 à synchronisation idéale

Le `phi0` et `phi1` des lignes 1 et 2 sont les mêmes que ceux déifnis aux lignes 2 et 3 du Code 4. Toujours pour un rapport signal sur bruit de 50dB, on trouve on taux d'erreur de 0%.

3.2. Erreur de synchronisation de phase porteuse

3.2.1. Introduction d'un déphasage

Si maintenant, on modifie les phase `phi0` et `phi1` du Code 11 de la manière suivante :

```
phi0 = rand * 2 * pi;
phi1 = rand * 2 * pi;
```

Code 12 – Introduction d'un déphasage

on trouve alors un taux d'erreur aléatoire, non nul. En effet, si nous reprenons les calculs précédents, on a maintenant :

$$\left\{\begin{array}{lll} I_{0,0} & = & \int_0^{T_s} cos(2\pi F_0 t + \phi_0)(cos(2\pi F_0 t + \phi_0')dt \\ I_{0,1} & = & \int_0^{T_s} cos(2\pi F_0 t + \phi_0)(cos(2\pi F_0 t + \phi_1')dt \\ I_{1,0} & = & \int_0^{T_s} cos(2\pi F_0 t + \phi_1)(cos(2\pi F_0 t + \phi_0')dt \\ I_{1,1} & = & \int_0^{T_s} cos(2\pi F_0 t + \phi_1)(cos(2\pi F_0 t + \phi_1')dt \end{array}\right.$$

On se retrouve alors avec du $\frac{T_s}{2}cos(\phi_0 - \phi_0')$. Selon le déphasage, cette quantité peut être positive ou négative. Ainsi le signe du résultat ne dépend plus que du bit initial, mais aussi de cette erreur aléatoire. Le récepteur ne fonctionne plus.

3.2.2. Présentation d'un nouveau récepteur

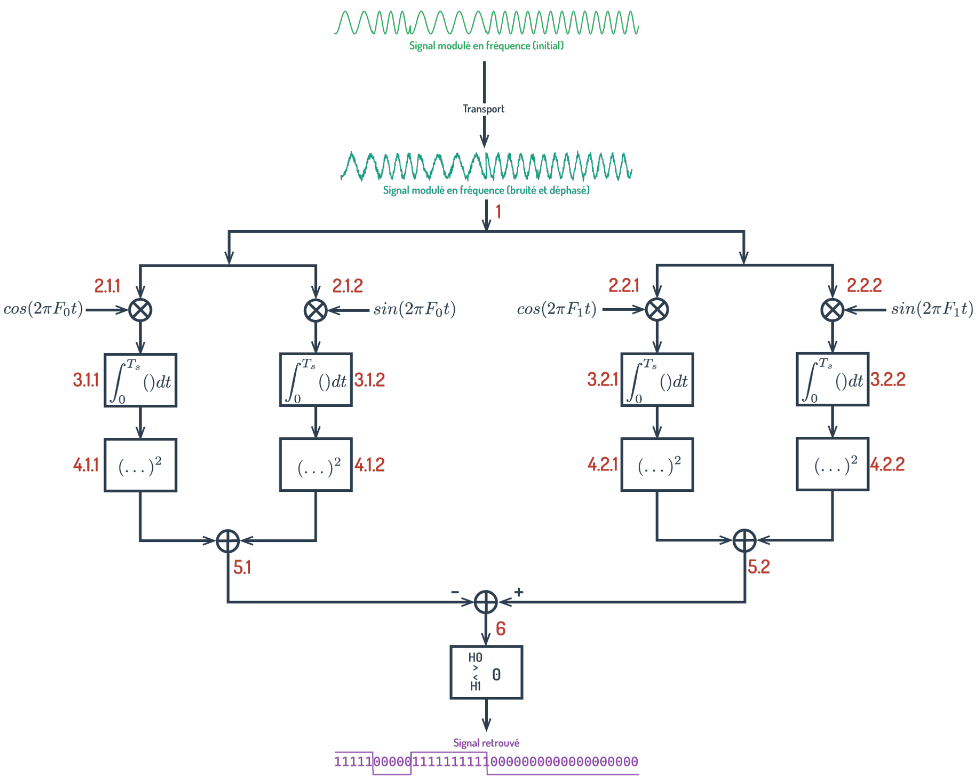


FIGURE 15 – Schéma de démodulation FSK avec déphasage.

Maintenant, si on reprend les calculs, notons R_i le résultat à l'étape i.

$$R_6 = R_{5.2} - R_{5.1}$$

Or,

- Si $NRZ = 0$, $R_{5.1} \approx \frac{T_s^2}{4}$ et $R_{5.2} = O(R_{5.1})$, d'où $R_6 = -R_{5.1} < 0$
- Si $NRZ = 1$, $R_{5.2} \approx \frac{T_s^2}{4}$ et $R_{5.1} = O(R_{5.2})$, d'où $R_6 = R_{5.2} > 0$

En fait, lors du développement des calculs, le terme $cos(\phi_0 - \phi_0')$ disparait à l'aide de la formule $cos^2 + sin^2 = 1$.

3.2.3. Implémentation du nouveau récepteur

```
x0c = x .* cos(2 * pi * F0 * t);
x0s = x .* sin(2 * pi * F0 * t);
x1c = x .* cos(2 * pi * F1 * t);
x1s = x .* sin(2 * pi * F1 * t);

x0c_reshape = reshape(x0c, Ns, Nb_bit);
X0c = sum(x0c_reshape * Te, 1).^2;

x0s_reshape = reshape(x0s, Ns, Nb_bit);
X0s = sum(x0s_reshape * Te, 1).^2;

x1c_reshape = reshape(x1c, Ns, Nb_bit);
X1c = sum(x1c_reshape * Te, 1).^2;

x1s_reshape = reshape(x1s, Ns, Nb_bit);
X1s = sum(x1s_reshape * Te, 1).^2;

X2 = (X1c + X1s) - (X0c + X0s);
Donnee_retrouve_3 = X2 > 0;

Nb_erreur_3 = sum(Donnee ~= Donnee_retrouve_3);
taux_erreur_3 = Nb_erreur_3 / Nb_bit;
```

Code 13 – Récepteur V21 avec déphasage

On retrouve alors un taux d'erreur de 0%.

4. Conclusion

Nous avons implémenté deux récepteurs différents
- un filtre simple, performant, mais demandant une grande largeur de bande fréquence (4 000Hz pour notre exemple).
- un démodultateur FSK, plus complexe, mais utilisant une bande de fréquence plus faible (200 Hz)