R&T Semestre 3 2023/2024

SAE31 : Etude et mise en œuvre d'un système de transmission

Reprise en main de Matlab

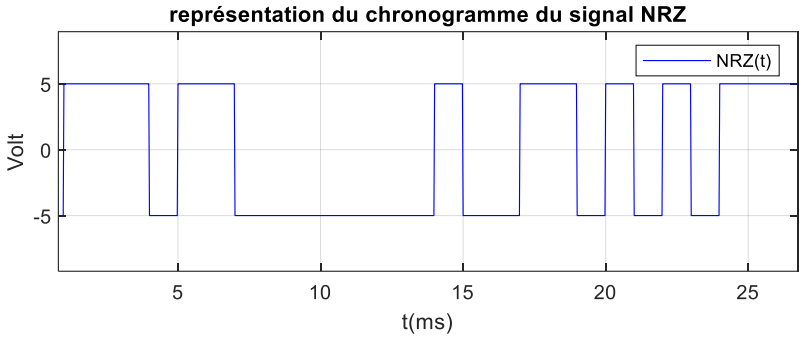
I / Révisions sur Matlab

II / Codage bande de base

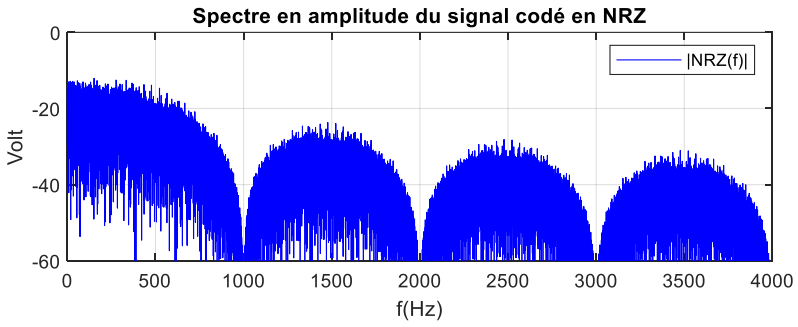
1) Codage NRZ.

On lance le programme NRZ et on a les deux représentations suivantes :

Chronogramme temporel :



Spectre en amplitude :



On observe bien les bits qui passent entre 5 et –5 à la fin des bits ce qui correspond bien au code NRZ de plus son spectre correspond bien au NRZ (D'après le cours)

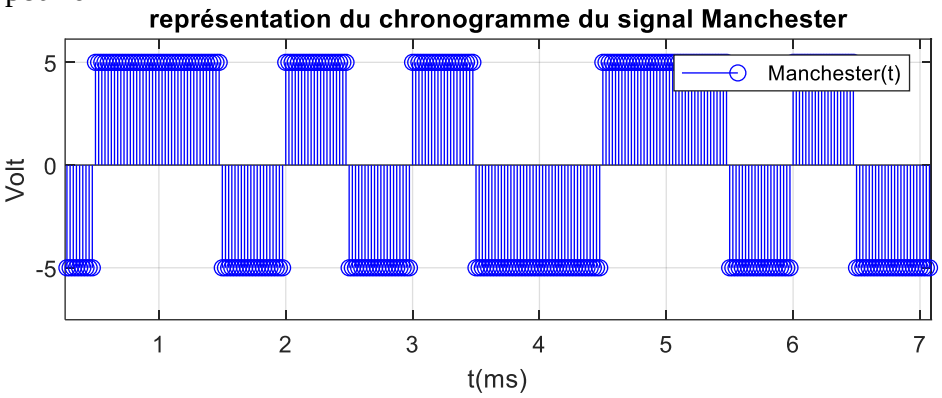
```
%% Création du signal NRZ
signal_NRZ=[]; %initialisation du signal codé en NRZ
symbole_1=5*ones(1,Nech_bit); % retourne un tableau de 1 ligne avec un nombre de
colonnes Nech avec que des 5
symbole_0=-5*ones(1,Nech_bit);% retourne un tableau de 1 ligne avec un nombre de
colonnes Nech avec que des -5
for n=1:Nb %codage des différents bits
if (data(n)==1) %la boucle itère toutes les valeurs de data et vérifie la condition
= 1
signal_NRZ=[signal_NRZ symbole_1]; % le signal NRZ de sortie prend la valeur au n
argument correspondant au n argument de symbole_1 soit 5
else
```

```
signal_NRZ=[signal_NRZ symbole_0];% le signal NRZ de sortie prend la valeur au n
argument correspondant au n argument de symbole_0 soit -5
end
```

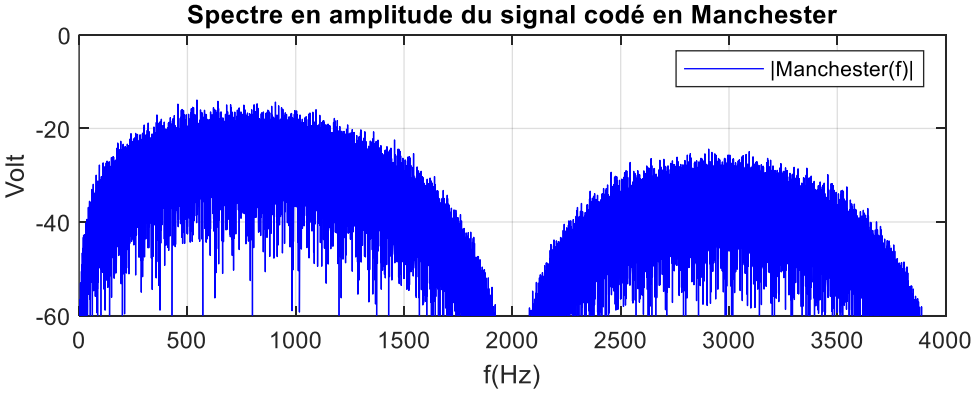
2) Codage Manchester

```
%% Création du signal Manchester
signal_Manchester=[];              %initialisation du signal codé en Manchester
symbole_1 = 5*ones(1,Nech_bit / 2);% retourne un tableau de 1 ligne avec un nombre
de colonnes Nech avec que des 5
symbole_0=-5*ones(1,Nech_bit / 2);% retourne un tableau de 1 ligne avec un nombre
de colonnes Nech avec que des -5

for n=1:Nb      %codage des différents bits
     if (data(n)==1) %la boucle itère toutes les valeurs de data et vérifie la
condition = 1
        signal_Manchester=[signal_Manchester symbole_1 symbole_0]; % le signal
Manchester de sortie prend la valeur au n argument correspondant au n argument de
symbole_1 soit 5
     else
        signal_Manchester=[signal_Manchester symbole_0 symbole_1];% le signal
Manchester de sortie prend la valeur au n argument correspondant au n argument de
symbole_0 soit -5
     end

end
```

On obtient donc un code avec pour les 1 : -5 puis 5 avec transition au milieu du bit et l'inverse pour 0



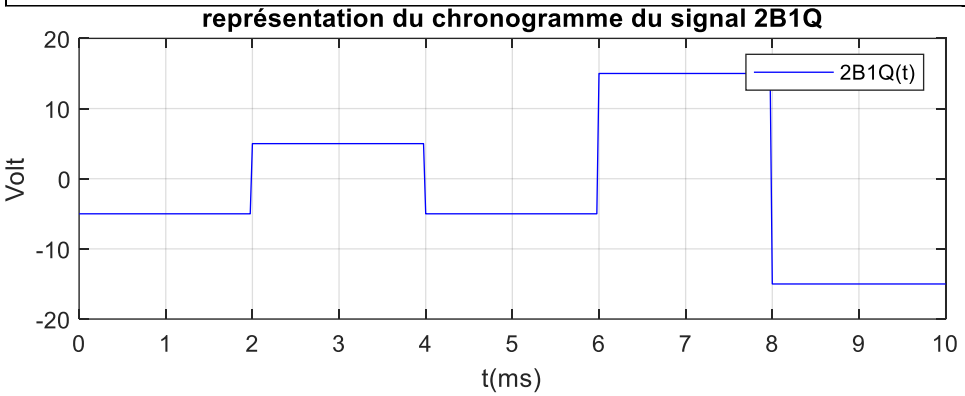
On obtient aussi ce spectre en qui correspond à celui que l'on trouve dans le cours :



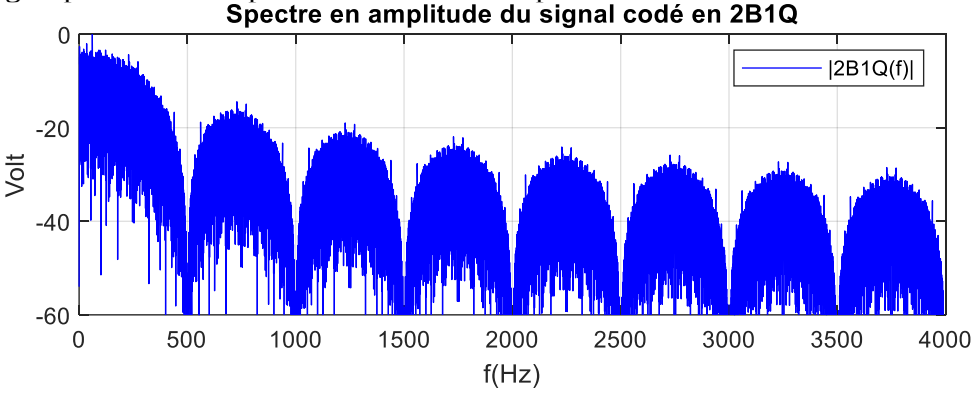
3) Autres codages :

2B1Q

```
data = transpose(data);
data = reshape(data, 2, []);
data = transpose(data);


%% Création du signal 2B1Q
signal_2B1Q=[];             %initialisation du signal codé en 2B1Q
symbole_00 = -15 *ones(1,Nech_bit *2 );
symbole_01 = -5 *ones(1,Nech_bit *2 );
symbole_11= 5 *ones(1,Nech_bit *2);
symbole_10 = 15*ones(1,Nech_bit*2);

for n=1:size(data,1)      %codage des différents bits
     if (data(n, :) == [0 0])
        signal_2B1Q=[signal_2B1Q symbole_00];
     elseif (data(n, :) == [0 1])
        signal_2B1Q=[signal_2B1Q symbole_01];
     elseif (data(n, :) == [1 1])
        signal_2B1Q=[signal_2B1Q symbole_11];
     elseif (data(n, :) == [1 0])
        signal_2B1Q=[signal_2B1Q symbole_10];

     end

end
```



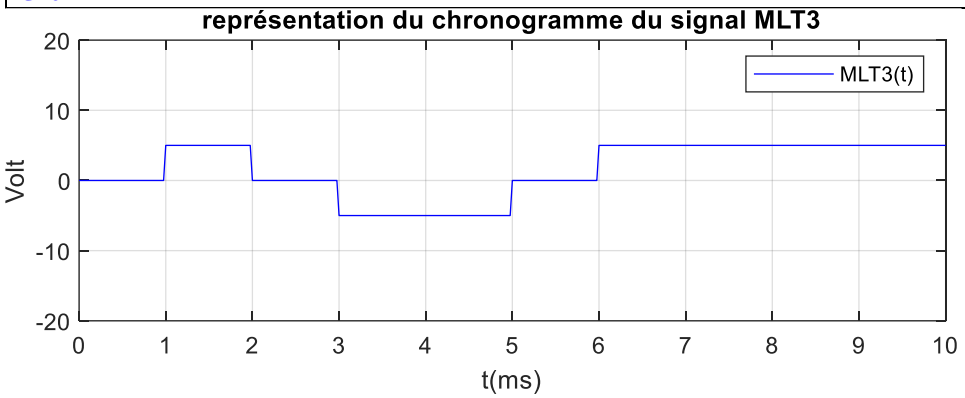
On obtient bien le code 2B1Q avec 4 états différents -5, 5, 15, -15 (soit A, -A, 3A, -3A) en groupant les bits 2 par 2 suivant la table spécifiée dans notre boucle for.



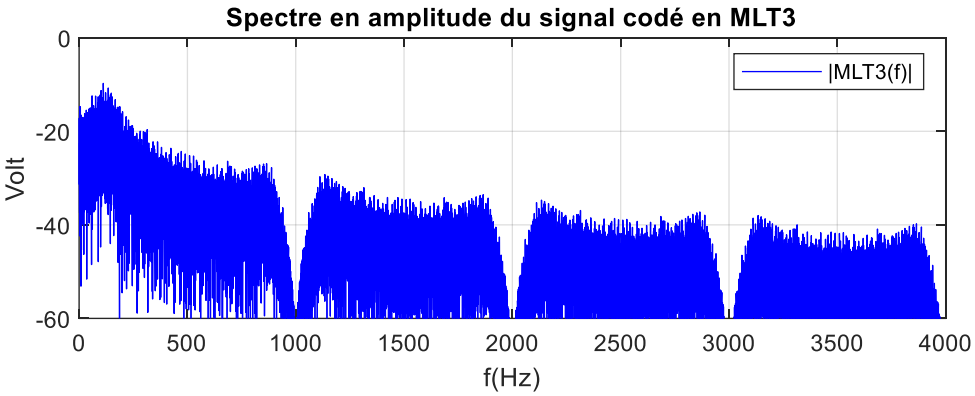
On voit sur le spectre que le premier lobe s'arrête à 500Hz soit $\frac{D}{2}$. Cela s'explique par sa valence qui est de 4 et non de 2 comme pour les codes utilisé précédemment.

MLT3

```
%% Création du signal MLT3
signal_MLT3=[];             %initialisation du signal codé en MLT3
nbr1 = 0;
symbole1 = 5 * ones(1,  Nech_bit);
etat = 0;
for n=1:size(data,2)      %codage des différents bits
     if (data(n) == 1)
          etat = cos(nbr1 * pi/2);
          nbr1 = nbr1 + 1;

     end
    signal_MLT3 = [signal_MLT3 symbole1 * etat];
end
```



On obtient ce chronogramme avec le code MLT3 qui correspond à un changement selon un cycle entre A, 0 et –A pour les bits à 1 et aucun changement d'état pour les bits a 0.



III / Modulation AM

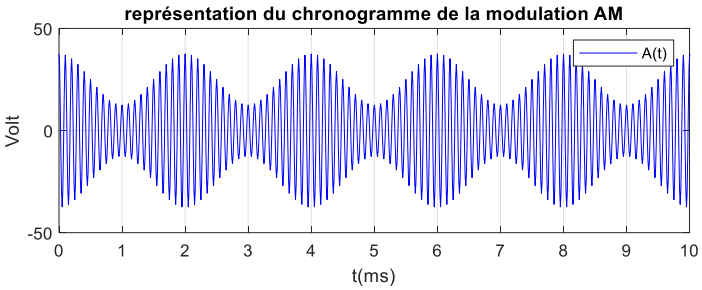
1) Modulant sinusoïdal

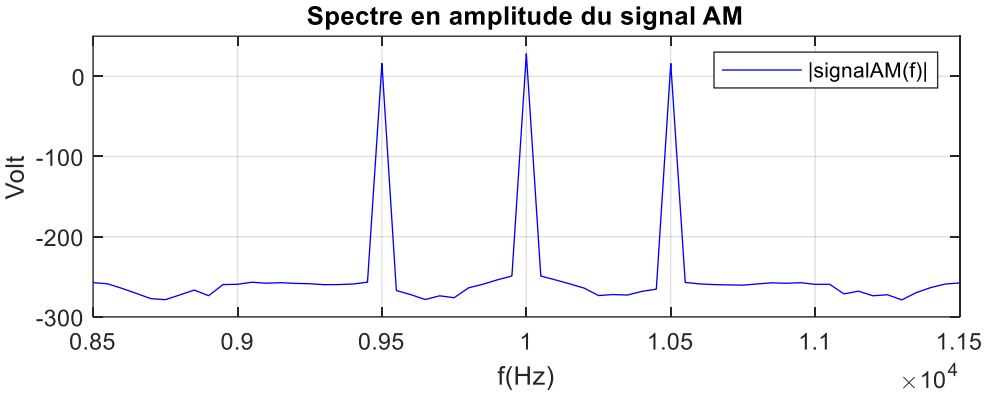
On complète le programme avec les lignes suivantes :

```
%% Création des différents signaux: modulant, porteuse et signal AM
porteuse = Ep * cos(2 * pi * fp * t);
modulant = cos(2 * pi * fm * t);
signalAM = Ep * (1 + m .* modulant) .* porteuse;


%% Calcul puis affichage des spectres
[X f]=spectre(signalAM,fe,Ne); %prend la sortie dans des variables X et f de la
fonction spectre
subplot(2,1,2);
plot(f,X,"b");
```

```
title('Spectre en amplitude du signal AM')
xlabel('f(Hz)')
ylabel('Volt')
legend('|signalAM(f)|')
axis([8500 11500 -300 50])  %affichage entre 0 et 2*D (Hz)
grid on
```

On obtient alors le chronogramme et le spectre suivant :





On voit bien sur la représentation spectrale le spectre du modulant à 500Hz des 2 côtés de la porteuse. Sur le chronogramme, on voit que la période du signal modulé est de 2ms qui est égale à la période d'un signal de 500Hz comme donné dans le programme.

2) Modulant audio

On va maintenant modifier le programme précédent afin de moduler en amplitude un signal audio.

Pour récupérer le signal audio et obtenir 2000 échantillons, on ajoute la ligne :

```
[modulant, fs] = audioread("musique.mp4", [100001, 102000]);
```

Pour normaliser le modulant, on met les lignes suivantes :

```
modulant_centre = modulant - mean(modulant);
modulant_cc = max(modulant_centered) - min(modulant_centered);
modulant_norm = modulant_centered * (2 / modulant_peak_to_peak);
```

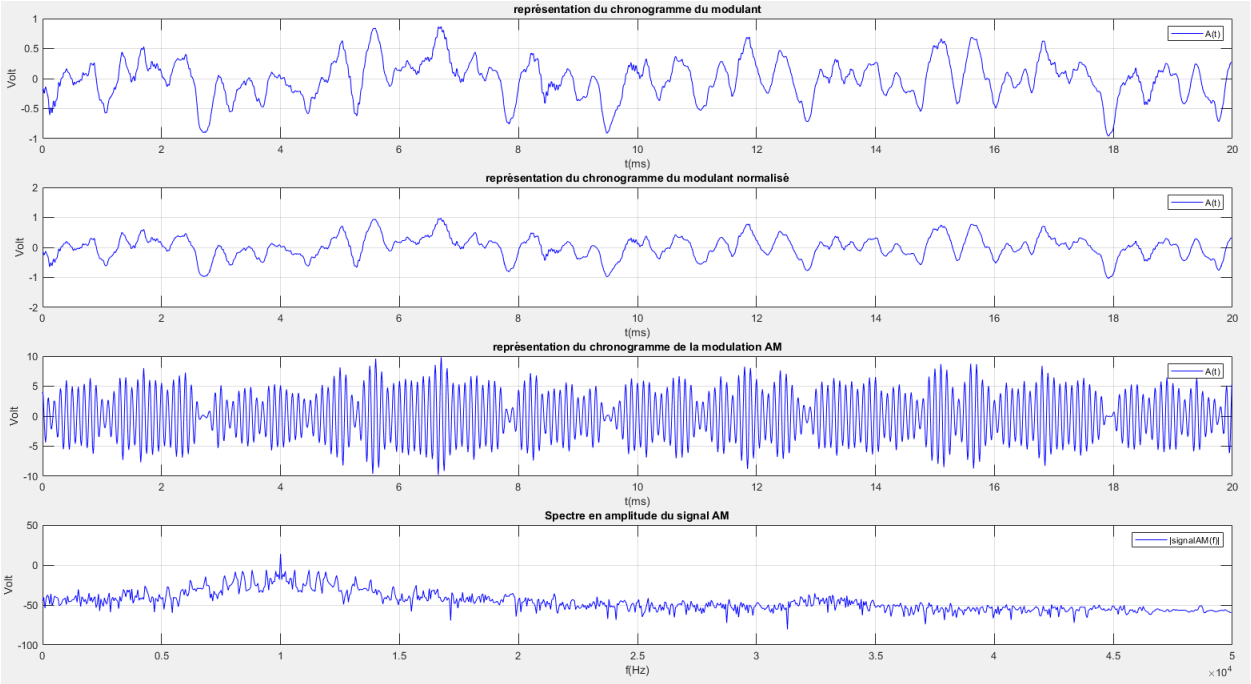
La première permet de centrer le signal vers 0, la deuxième à obtenir l'amplitude crête-crête du signal et la dernière de le normaliser en multipliant le modulant par $\frac{2}{Amplitude\ crête-crête}$.

On génère le signal AM avec la ligne suivante :

```
signalAM = (1 + m * modulant_norm') .* porteuse
```

Cette ligne correspond à la formule $s(t) = P_o\big(1 + k \times m(t)\big) \times \cos(\omega pt)$.

On a donc les chronogrammes et le spectre suivant :

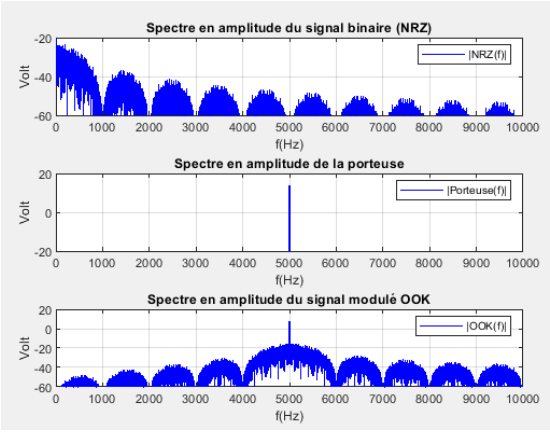


On voit bien que quand le modulant est bas sur le chronogramme, l'amplitude est faible sur le signal modulé et inversement quand le modulant est fort. Sur le spectre on a un pic à 1kHz qui est la fréquence porteuse que l'on a initialisé au début du programme. La fréquence ne changeant pas, il n'y a pas d'autre pic sur le spectre.

IV / Modulation ASK, PSK et FSK

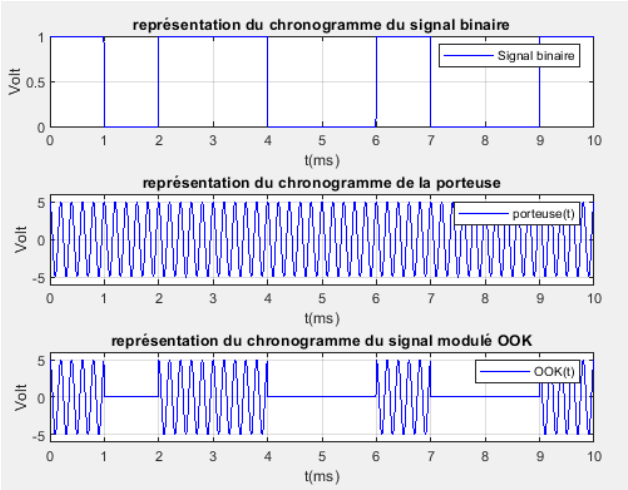
1) Modulation ASK

Pour la modulation ASK, on a les résultats suivants :



On voit que le spectre OOK contient le spectre du signal NRZ de chaque côté de la porteuse.

```
ASK=signal_NRZ.*porteuse;
```



On voit que quand le signal NRZ est à 0, on n'a pas de signal sur le chronogramme du OOK et quand il y a le signal est à 1, on a la porteuse qui est envoyé sur le signal modulé OOK.

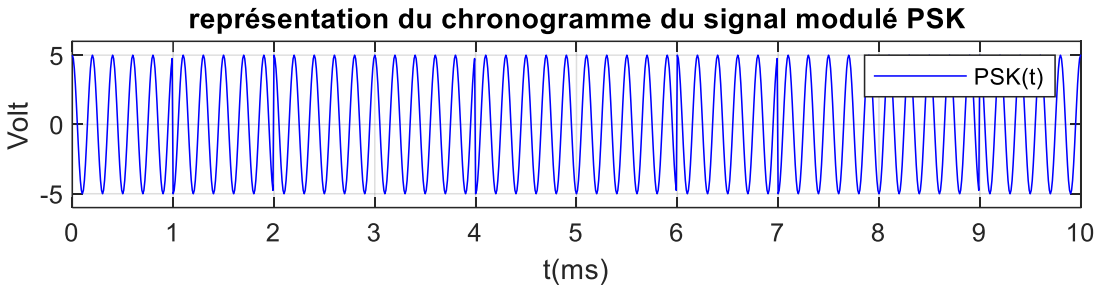
2) Modulation PSK

On change la ligne de modulation par la ligne suivante :

```
PSK= -(porteuse .* (1 - 2 * signal_NRZ));
```

On a donc 1 – 2 fois le signal ce qui fait qu'on obtient 1 quand le signal NRZ est à 0 et -1 quand le signal NRZ est à 1. Or ici nous voulons avoir l'inverse de ce résultat, le moins devant la multiplication entre la porteuse et la modulation nous permet d'obtenir le résultat escompté.

On a le chronogramme suivant :



On voit donc bien des sauts de phase lors des fronts du signal NRZ.

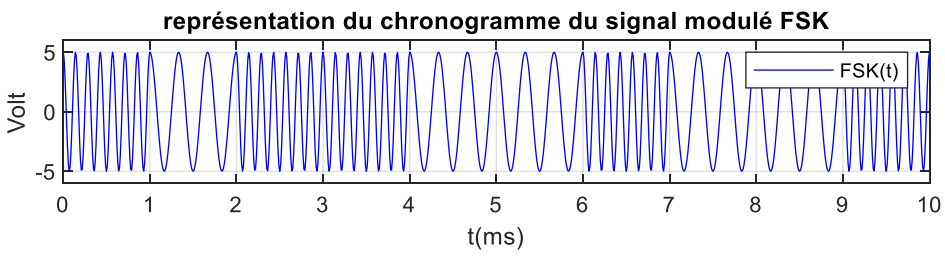
3) Modulation FSK

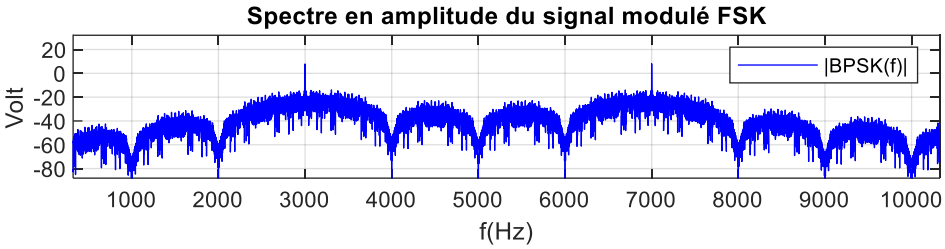
Pour faire une modulation FSK, on met la ligne suivante dans le programme :

```
FSK = 5 * cos(2*pi*((fp - deltaf)*(1 - signal_NRZ) + (fp + deltaf)*signal_NRZ).*t);
```

Elle permet d'avoir $fp - Df$ quand le signal NRZ est à 0 avec `(fp - deltaf)*(1 - signal_NRZ)` et d'avoir $fp + Df$ quand le signal NRZ est à 1 avec `(fp + deltaf)*signal_NRZ.`

On a le chronogramme et le spectre du signal modulé :





On voit bien qu'il y a une différence de fréquence entre les moments où le signal NRZ est à 1 et quand il est à 0. Sur le spectre on voit qu'il y a une raie à 3000 Hz et à 7000Hz. Soit à $fp \mp Df$ comme prévu initialement.

R&T Semestre 3 2022/2023

SAE31 : Etude et mise en œuvre d'un système de transmission

Transmission sur Adalm Pluto

I / Présentation de la SDR ADALM-PLUTO

La radio définie par logicielle SDR « Software Defined Radio » est un équipement qui permet de concevoir puis de tester des émetteurs / récepteur RF. Pour cela la SDR comprend :

- Un transceiver RF (l'AD9363). Il s'agit principalement d'un modulateur/démodulateur IQ.
- Un SoC « System On Chip » comportant un DSP/FPGA permettant d'implémenter les algorithmes de traitement du signal pour l'émission/réception RF
- Une interface pour la connexion avec un PC qui permettra de tester ces algorithmes sur un logiciel sur le PC avant son implantation dans le DSP/FPGA.

Nous utiliserons la SDR ADALM-PLUTO d'Analog Devices qui est dédié pour l'enseignement et qui a un coût très faible (une centaine d'Euros) avec en contrepartie un FPGA de faible capacité dans lequel nous ne pourrons pas implanter les algorithmes créés.

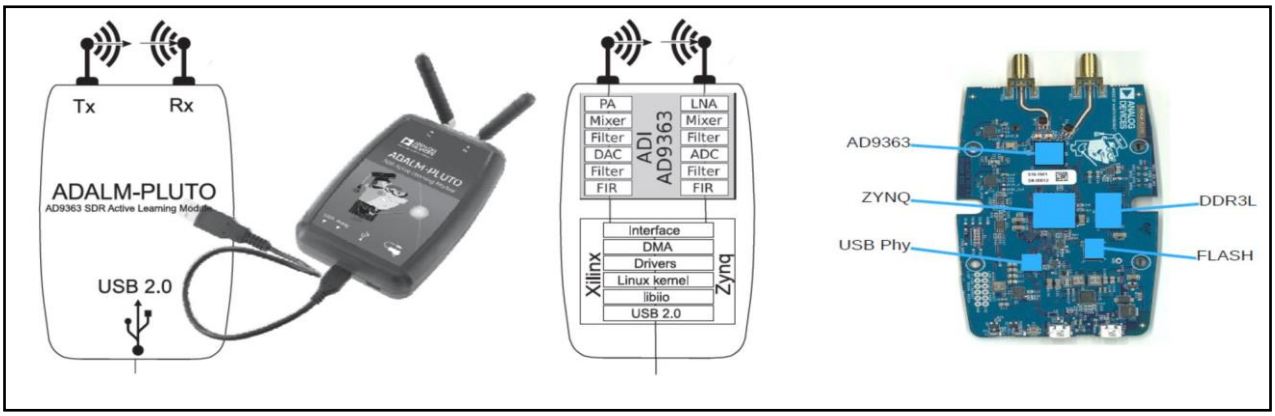


Figure 1: ADALM Pluto

Power	
DC Input (USB)	4.5 V to 5.5 V
Conversion Performance and Clocks	
ADC and DAC Sample Rate	65.2 kSPS to 61.44 MSPS
ADC and DAC Resolution	12 bits
Frequency Accuracy	±25 ppm
RF Performance	
Tuning Range	325 MHz to 3800 MHz
Tx Power Output	7 dBm
Rx Noise Figure	<3.5 dB
Rx and Tx Modulation Accuracy (EVM)	–34 dB (2%)
RF Shielding	None
Digital	
USB	2.0 On-the-Go
Core	Single ARM Cortex®-A9 @ 667 MHz
FPGA Logic Cells	28k
DSP Slices	80
DDR3L	4 Gb (512 MB)
QSPI Flash	256 Mb (32 MB)
Physical	
Dimensions	117 mm × 79 mm × 24 mm 4.62" × 3.11" × 0.95"
Weight	114 g
Temperature	10°C to 40°C

Figure 2: caractéristiques de l'ADALM Pluto

Nous configurerons cette SDR avec le logiciel **Matlab** qui comprend une bibliothèque intégrant l'ensemble des algorithmes nécessaires à la réalisation des émetteurs / récepteurs RF.

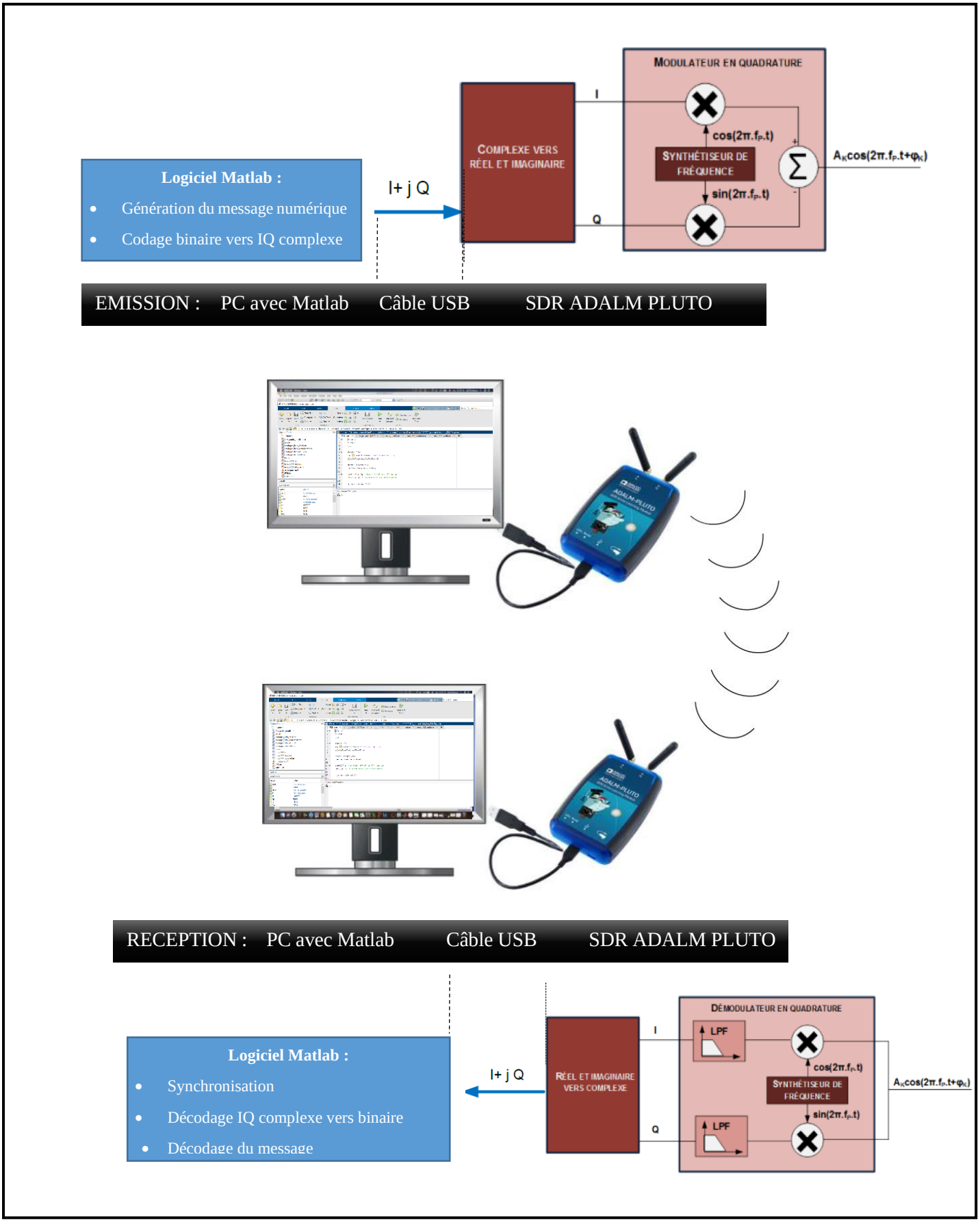


Figure 3: schéma de principe d'une transmission utilisant Matlab et l'Adalm Pluto

II / Emission de la porteuse avec un Adalm Pluto

1) Objectif

Il s'agit ici de réaliser l'émission de la porteuse de fréquence f_p a l'aide de l'Adalm Pluto.

La fréquence porteuse sera choisie dans la bande libre ISM allant de 2,4 à 2,5GHz. Comme vous serez plusieurs à émettre simultanément, vous ne devez pas tous avoir la même fréquence porteuse.

Vous devez affecter à chacun d'entre vous un numéro de canal : n. Votre fréquence de porteuse vaudra alors $\boldsymbol{f_p = 2400 + n * 2\ MHz}$.

2) Principe de l'émission

Le schéma de principe est le suivant :

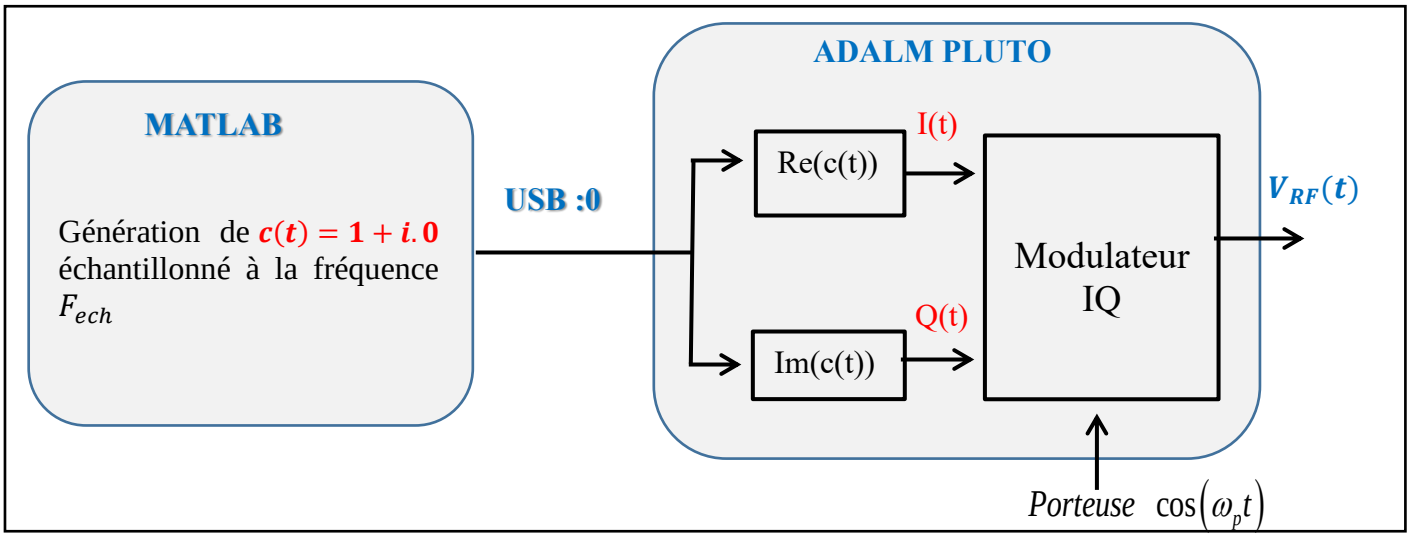


Figure 4 : principe de l'émission de la porteuse

On rappelle que : $V_{RF}(t) = I.cos(\omega_p t) - Q.sin(\omega_p t)$ (1).

Pour émettre la porteuse, il suffit de générer I(t)=1 et Q(t)=0. Le programme Matlab devra donc créer le signal complexe : c(t)=1+i.0

3) Mise en œuvre

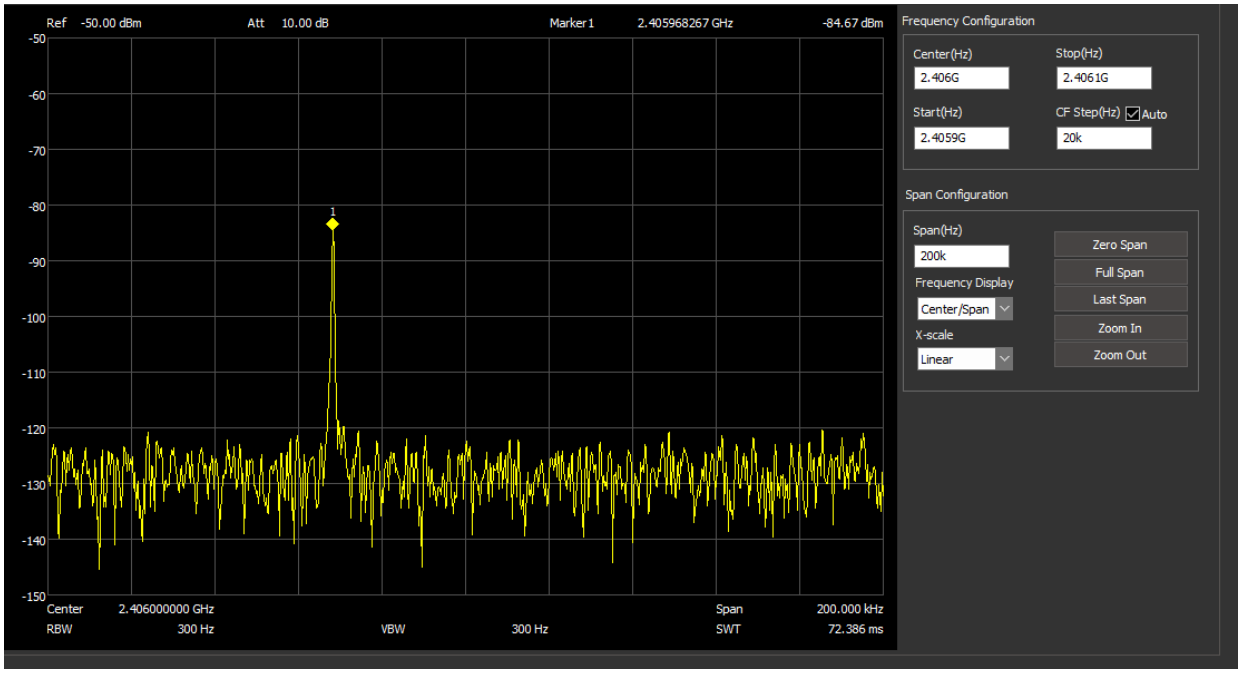
Nous avons le poste 3, par conséquent on aura une fréquence porteuse de 2,406GHz.

```
%% Calcul des symboles complexes c à transmettre au modulateur IQ de l'Adalm Pluto
f0 = 50000;
c=complex(exp(1j * 2*pi*f0*t)); % pour transmettre la porteuse C=1+0.j  (I=1 et Q=0)
```

La dernière ligne indique que l'on va créer les symboles complexes avec la partie réelle I à 1 et la partie imaginaire Q à 0, on aura donc : $c = 1 + 0 \times j$. On crée ensuite une variable tx qui utilise la fonction sdrtx, cette dernière permet de créer un « Objet système » d'émetteur pour l'Adalm Pluto avec en paramètre la fréquence porteuse et la fréquence d'échantillonnage.

4) Validation

Nous allons utiliser l'analyseur de spectre SSA 3050X-R de Siglent pour cette étude.



On peut donc voir qu'il y a une erreur puisque la fréquence de la porteuse relevée est de 2.405968GHz, soit une différence de – 32 kHz par rapport à ce qui est attendu.

On exprime donc en ppm l'erreur relative,

$$Err(ppm) = \frac{(Ft - Fp)}{Fp} \cdot 10^6 = \frac{(2.406 \cdot 10^9 - 2.405968 \cdot 10^9)}{2.405968 \cdot 10^9} \cdot 10^6 = 13.3ppm$$

On peut donc voir que l'erreur en ppm est inférieure à 25 ce qui est conforme à la documentation de l'appareil.

> *Conclusion : la fréquence porteuse de l'Adalm Pluto n'est pas extrêmement précise. Ceci peut être gênant lors d'une transmission. Le paragraphe suivant donnera une méthode afin de corriger cette erreur.*

III / Transmission entre 2 Adalm Pluto – décalage fréquentiel

On veut maintenant réaliser une transmission entre 2 Adalm Pluto. On pourra se reporter à la Figure 3 et à la Figure 4.

1) Emission

En émission on va maintenant générer $c(t) = e^{j2\pi f_0 t}$ avec $f_0 = 50kHz$.

On a : $cos(a + b) = \cos a \times \cos b - \sin a \times \sin b$

D'après le cours : $I(t) = \cos(2\pi f_0 t)$ et $Q(t) = \sin(2\pi f_0 t)$

On part de : $V_{RF}(t) = I.\cos(\omega_p t) - Q.\sin(\omega_p t)$

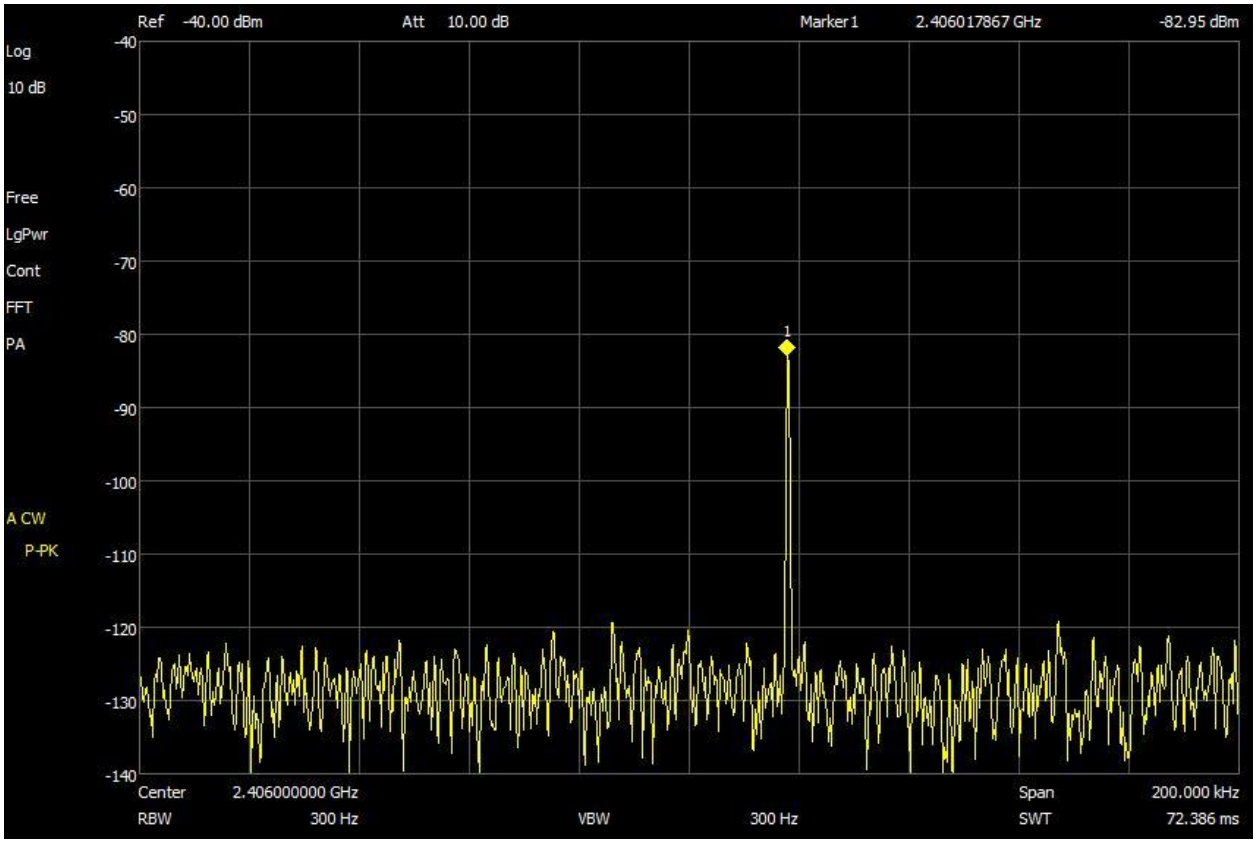
Donc : $V_{RF}(t) = \cos(2\pi f_0 t).\cos(2\pi f_p t) - \sin(2\pi f_0 t).\sin(2\pi f_p t)$

Donc : $V_{RF}(t) = \cos(2\pi f_0 t + 2\pi f_p t)$

Enfin : $V_{RF}(t) = \cos(2\pi(f_0 + f_p)t)$

Par conséquent, on a bien un décalage de la fréquence porteuse de $f_0 = 50kHz$.

Sur l'analyseur de spectre, on a donc le résultat suivant :



On a donc une raie à 2,406017 GHz. On a donc une différence de $2{,}406017 - 2{,}405968 \approx 50 \times 10^6 Hz\ soit\ 50kHz$.

2) Réception

On remplace les parties servant à l'émission dans le programme par les lignes suivantes :

```
%% configuration de l'ADALM PLUTO récepteur
rx = sdrrx('Pluto', 'RadioID', 'usb:1', 'CenterFrequency', fp, ...
        'BasebandSampleRate', Fech, 'SamplesPerFrame', Nech, ...
        'OutputDataType', 'double', 'ShowAdvancedProperties', true);


%% Réception des échantillons
reception= rx(); % réception d'une trame de Nech échantillons
reception=reception';   % on prend la transposée pour avoir un vecteur colonne...
```

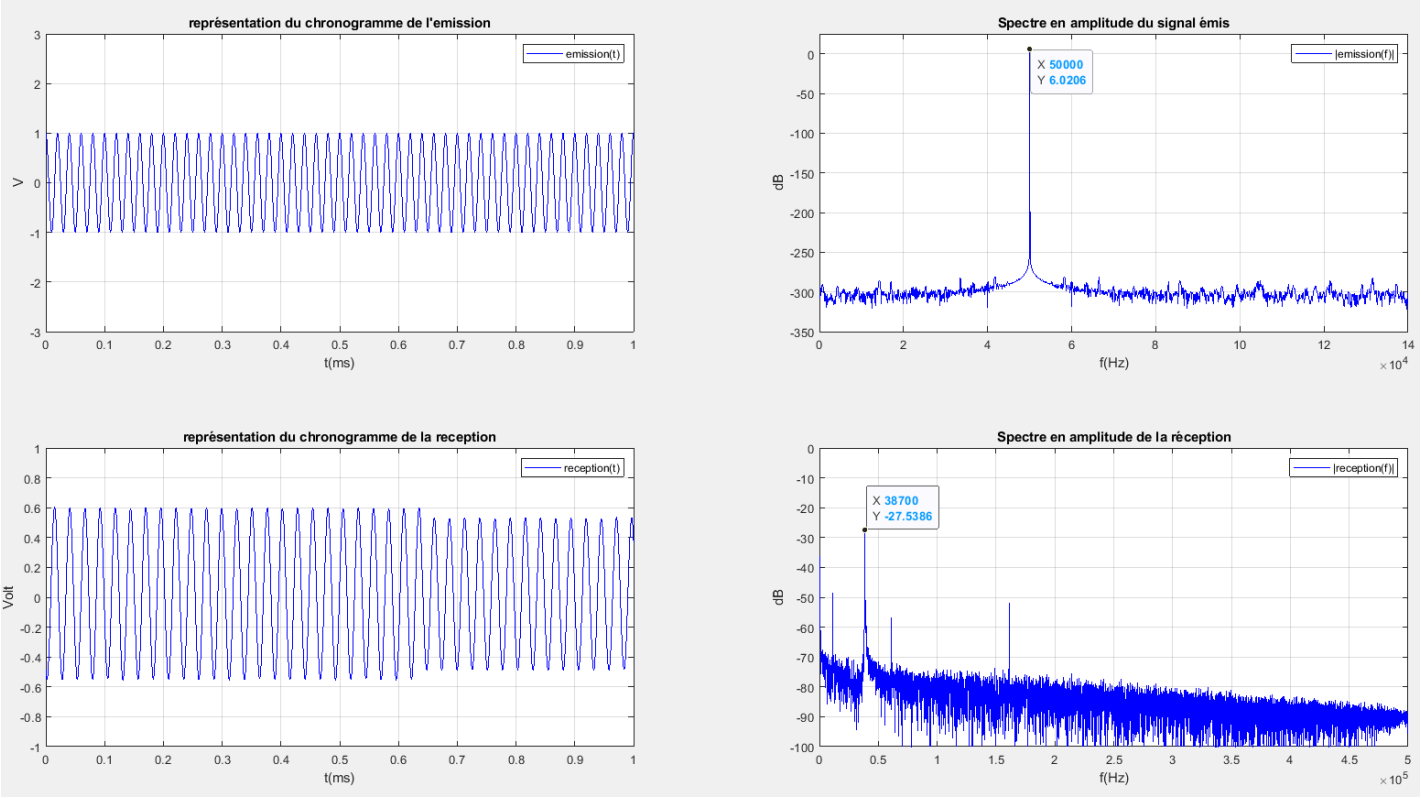
3) Exploitation des résultats

Si la transmission était parfaite, on devrait avoir $reception(t) = c(t) = e^{j2\pi f_0 t}$.

- En fait, on va avoir 2 modifications :
    - Le signal reçu ne sera pas de la même amplitude que le signal émis : le signal est atténué et il y a des amplis en réception.
    - Le signal reçu n'aura pas la même fréquence que le signal émis : ceci est dû à l'imprécision des horloges des Adalm Pluto (25ppm, cf. Figure 2). On notera f_1 la fréquence de réception.

On s'attend donc à avoir : $reception(t) = ke^{j2\pi f_1 t}$.

- On notera $\Delta f = |f_1 - f_0|$ la déviation de fréquence en Hertz soit en ppm : $dev_{ppm} = \frac{\Delta f}{f_p} * 10^6$
- On considère que :
    - Le signal émis est $e(t) = Re(c(t))$
    - Le signal reçu est $r(t) = Re(reception(t))$



La déviation en fréquence est donc de 11,3kHz comme observé ici entre le 50kHz d'envoyé le 38,700kHz obtenu en réception

$$Err(ppm) = \frac{\Delta f}{Fp} \cdot 10^6 = \frac{(50000 - 38700)}{2.4 \times 10^9} \cdot 10^6 = 4{,}7\ ppm$$

R&T Semestre 3 2022/2023

SAE31 : Etude et mise en œuvre d'un système de transmission

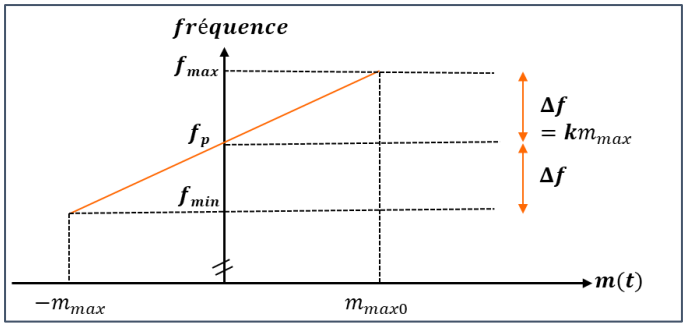
Transmission FM (*)

I / Principe théorique de l'émission FM

1) Rappel sur la modulation FM

- Soit m(t) un signal modulant à transmettre pouvant varier entre - m_{max} et + m_{max}.
- Une porteuse $p(t) = cos(2\pi f_p t)$.
- Le signal modulé FM a pour expression : $\boldsymbol{FM(t) = cos(\Phi(t))}$ **(1)** où $\Phi(t)$ est la phase instantanée. Sa **fréquence instantanée** est définie par : $\boldsymbol{f(t) = \frac{1}{2\pi}\frac{d\phi(t)}{dt}}$ **(2)**
- La fréquence instantanée du signal modulé FM a pour expression : .

$$\boldsymbol{f(t) = f_p + k.m(t)} \ \textbf{(3)}$$



- On définit **l'excursion de fréquence** par : $\Delta f = k.m_{max}$. On a donc $k = \frac{\Delta f}{m_{max}}$

Si on prend en compte les équations (2) et (3), on a donc : $\boldsymbol{\frac{1}{2\pi}\frac{d\Phi(t)}{dt} = f_p + k.m(t)}$ **(4)**

Et donc : $\Phi(t) = 2\pi \int_0^t (f_p + k.m(t))dt = 2\pi f_p t + 2\pi k \int_0^t m(t)dt$

Finalement l'expression du signal modulé en fréquence est :

$$\boldsymbol{FM(t) = cos\left(2\pi f_p t + 2\pi \frac{\Delta f}{m_{max}} \int_0^t m(t)dt\right)} \quad \textbf{(5)}$$

2) Réalisation d'une modulation FM à l'aide d'un modulateur IQ

Le signal en sortie de l'Adalm Pluto doit donc être identique à celui de l'équation (5).

On rappelle que le programme Matlab est censé générer le signal complexe C=I+j.Q en entrée du modulateur IQ de l'Adalm Pluto (cf. Figure 3).

C peut être mis sous forme polaire : $\boldsymbol{C = I + jQ = \rho . e^{j\theta}}$ **(6)**

Avec: $\boldsymbol{I = \rho . cos(\theta)\ et\ Q = \rho . sin(\theta)}$

On a donc en sortie du modulateur IQ de l'Adalm Pluto :

$V_{RF}(t) = I\cos(\omega t) - Q\sin(\omega t) = \rho . cos(\theta)\cos(\omega t) - \rho . sin(\theta)\sin(\omega t)$

Donc : $V_{RF}(t) = \rho . (cos(\theta)\cos(\omega t) - sin(\theta)\sin(\omega t))$

On rappelle la formule de trigo : $\cos(a+b) = cosa.cosb - sina.sinb$

Soit : $\boldsymbol{V_{RF}(t) = \rho(t).cos(\omega_p t + \theta(t))}$ **(7)**

Conclusion partielle : pour réaliser une modulation FM, les équations (5) et (7) doivent donc être identiques. Il faut donc prendre :

$$\begin{cases} \rho(t) = 1 \\ \theta(t) = 2\pi \dfrac{\Delta f}{m_{max}} \displaystyle\int_0^t m(t)dt \end{cases}$$

Conclusion finale :

Pour réaliser une modulation FM du signal m(t) avec une excursion de fréquence Δf, Matlab devra générer le signal complexe

$$\boldsymbol{c(t) = e^{i\theta(t)}\ avec\ \theta(t) = 2\pi \frac{\Delta f}{m_{max}} \int_0^t m(t)dt} \quad \textbf{(8)}$$

II / Réalisation de l'émission FM

- Récupérer le signal audio nommé 'musique.mp4' et disponible sur l'ENT à l'aide de la commande 'audioread'. Ce signal sera par la suite nommé $m(t)$.

  Remarque : la commande audioread permet de récupérer aussi la fréquence d'échantillonnage de ce signal audio: f_{ech}.

- Limiter le signal m(t) à ses Nech=10^6 premiers échantillons afin de ne pas demander trop de ressources à Matlab.

On doit donc maintenant générer le signal c(t) donné dans l'équation (8) :

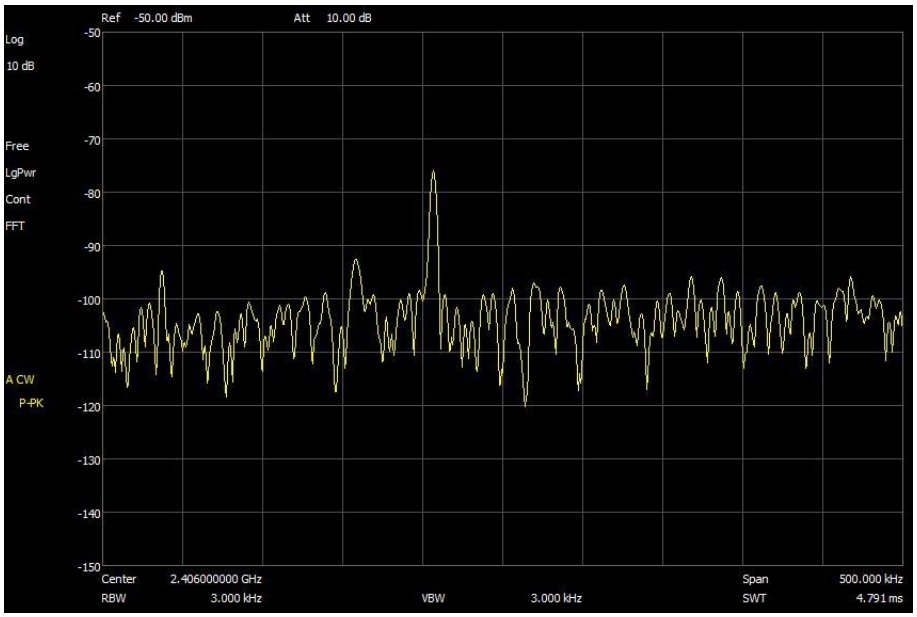
- En utilisation la fonction ‘cumtrapz’ (intégration par la méthode des trapèzes), générer le signal $\text{integ}(t) = \int_0^t m(t)dt$.
- Calculer la valeur maximale du signal m(t) : m_{max} .
- Calculer $\theta(t) = \frac{2\pi\Delta f}{m_{max}}.\text{integ}(t)$
- En déduire le signal complexe : $c(t) = e^{i\theta(t)}$.
- Transmettre c(t) à l’adalm pluto grâce au ligne suivantes :

```
% Set up the transmitter
fe=1e6;
tx = sdrtx('Pluto', 'RadioID', 'usb:0', 'CenterFrequency', fp, 'BasebandSampleRate',
fe, 'Gain',0,'ShowAdvancedProperties', true,'BISTLoopbackMode','Disabled');
% Send signals
release(tx)
transmitRepeat(tx, c);
```

Remarque : la fréquence d’échantillonnage de l’Adalm Pluto ($fe = 1MHz$) n’est pas la même que celle du fichier audio (fech)…

- Si un signal audio dure 3 minutes. Compte tenu de ces valeurs de fe et de fech, combien de temps durera sa transmission ?

On a le spectre suivant sur l’analyseur de spectre :



On ne peut pas le voir sur une seule image mais la raie bouge sur le spectre.

R&T Semestre 3 2022/2023

SAE31 : Etude et mise en œuvre d'un système de transmission

Transmission FSK d'un texte

I / Principe de la FSK

- On considère pour la suite :
  - Un message numérique, m= [1 0 1 0 1 1 0 1 0] à transmettre.
  - N_b=9 : nombre de bits du message
  - D=20kbits/s le débit binaire et T_b=1/D la durée d'un bit.
  - f_p=2,4GHz la fréquence porteuse (à ajuster selon votre numéro de canal).
  - Δf=100kHz la déviation de fréquence.
  - On se réfère toujours à la Figure 4 page 11.

- Pour transmettre le k^{ième} bit du message, m_k, le principe de la FSK est le suivant :
  - Si m_k=1 : on doit transmettre un signal sinusoïdal de fréquence $f_p + Df$. On veut donc avoir $V_{RF}(t) = cos(2\pi(f_p + \Delta f)t)$. Ce signal devra être transmis pendant la durée T_b de ce bit.
  - Si m_k=0 : on doit transmettre un signal sinusoïdal de fréquence $f_p - Df$. On veut donc avoir $V_{RF}(t) = cos(2\pi(f_p - \Delta f)t)$. Ce signal devra être transmis pendant la durée T_b de ce bit.

Pour un « 1 », l'antenne devra émettre $c(t) = e^{j2\pi\Delta ft}$ car avec cette valeur on a :

$V_{FSK}(t) = \cos(2\pi\Delta ft) + j\sin(2\pi\Delta ft)$,

De plus on utilise un modulateur IQ donc : $V_{FSK}(t) = I.\cos(2\pi f_p t) - Q.\sin(2\pi f_p t)$

Enfin, on a vu dans la partie III/1) de la transmission avec l'ADALM Pluto que cela nous donnait $V_{FSK}(t) = cos(2\pi(f_p + \Delta f)t)$ avec $I(t) = \cos(2\pi\Delta ft)$ et $Q(t) = \sin(2\pi\Delta ft)$.

Pour un « 0 », l'antenne devra émettre $c(t) = e^{-j2\pi\Delta ft}$ car avec cette valeur on a :

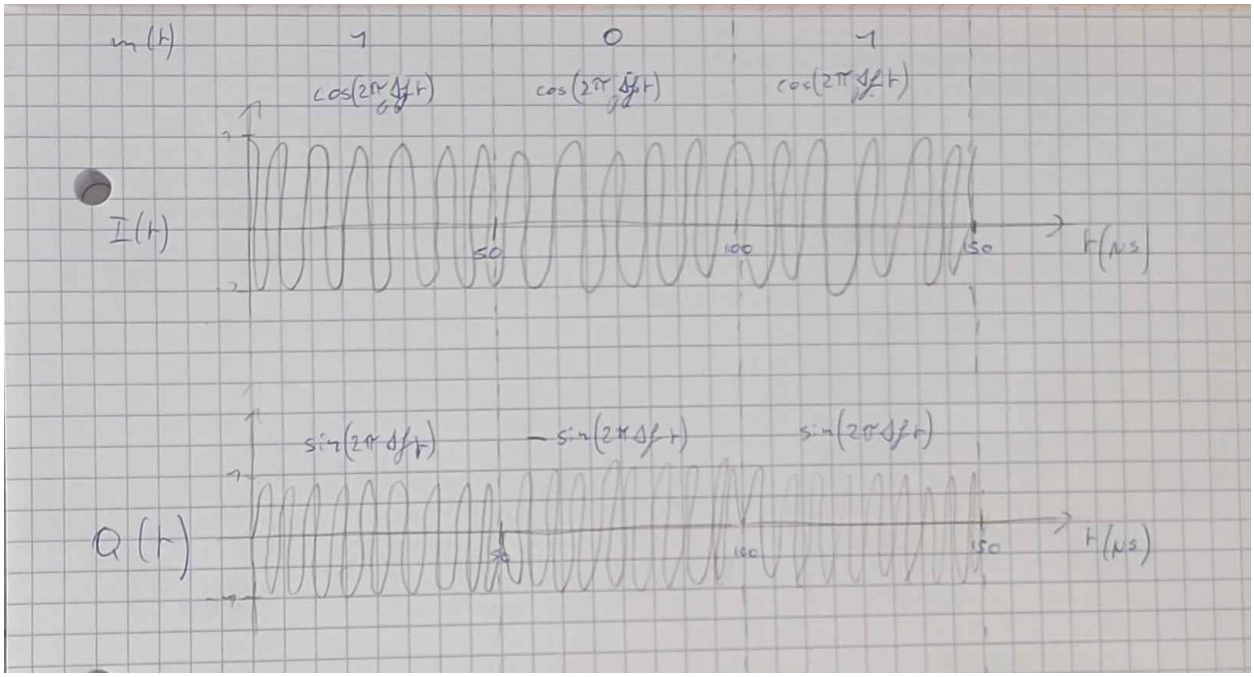
$V_{FSK}(t) = \cos(2\pi\Delta ft) - j\sin(2\pi\Delta ft)$,

De plus on utilise un modulateur IQ donc : $V_{FSK}(t) = I.\cos(2\pi f_p t) - Q.\sin(2\pi f_p t)$

Enfin, avec ce qu'on a vu plus tôt, on en déduit que $V_{FSK}(t) = cos(2\pi(f_p - \Delta f)t)$ avec $I(t) = \cos(2\pi\Delta ft)$ et $Q(t) = -\sin(2\pi\Delta ft)$.

On a : $D = 20kbits/s$ donc $Tb = \frac{1}{20000} = 50\mu s$ donc $Ts = 50\mu s$ car on a une valence de 2.

On a l'allure des chronogrammes pour les 3 premiers bits suivants :



II / Emission d'un message simple en FSK

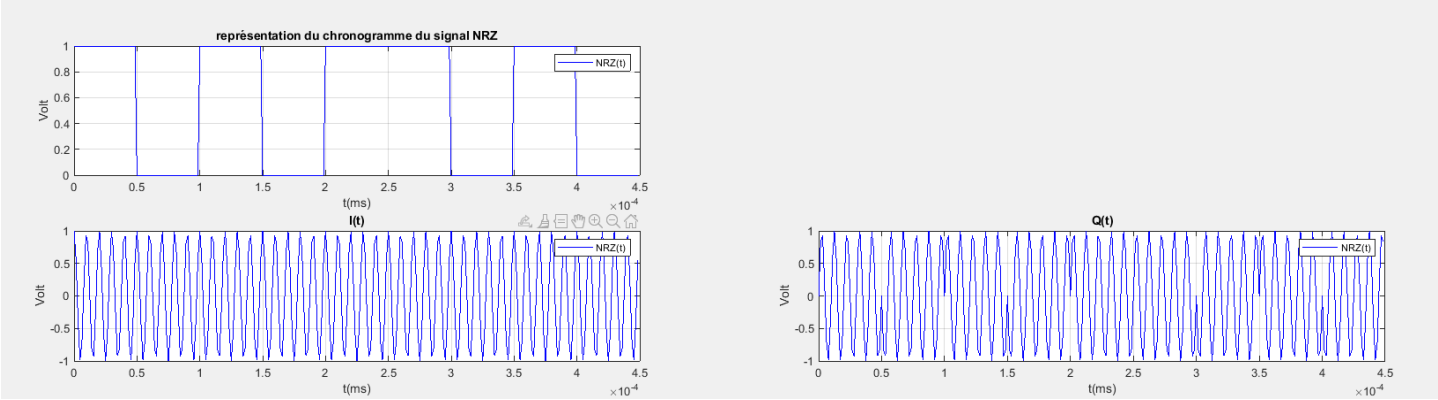
On se propose d'émettre en FSK le message m(t) suivant les caractéristiques données dans le paragraphe précédent.

On doit donc générer sous Matlab le signal c(t) correspondant à cette FSK. En fait, ce signal est échantillonné à une fréquence f_{ech}.

On choisira $\boldsymbol{f_{ech} = N_{ech_{symb}}.D}$, c'est-à-dire $T_{ech} = \frac{T_b}{N_{ech_{symb}}}$. Avec : $\boldsymbol{N_{ech_{symb}} = 32}$.

On aura ainsi 32 échantillons par symbole (bit).

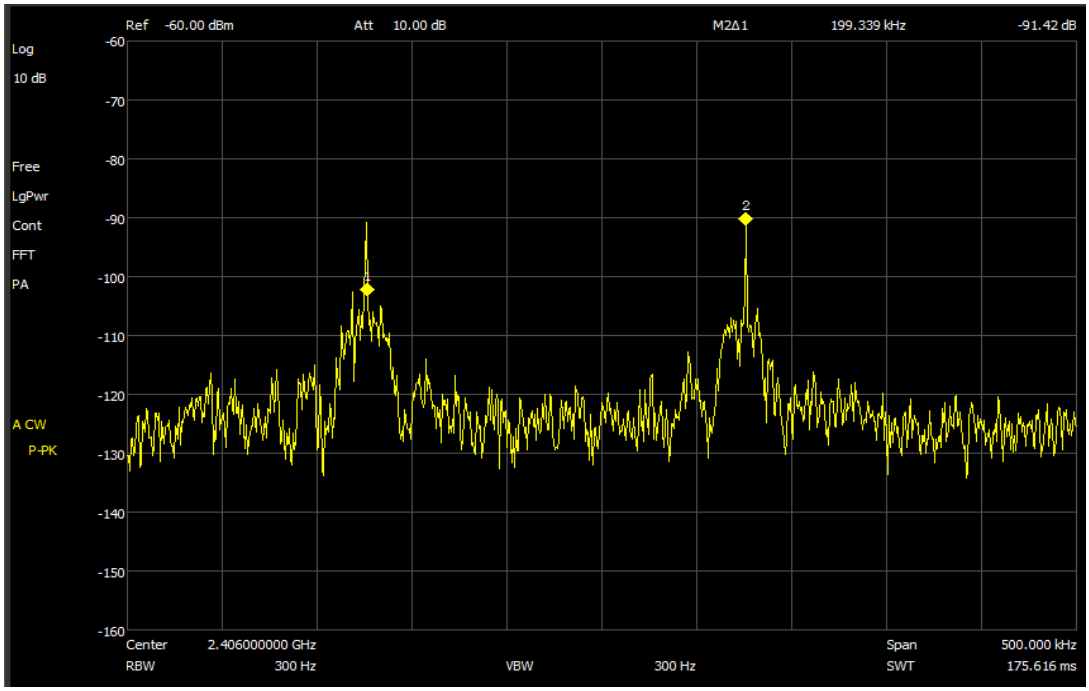
Au total on aura donc $N_{ech} = \boldsymbol{N_{ech\_symb} * N_b}$ échantillons à transmettre, où N_b est le nombre de bits du message.



On voit donc les deux signaux sinusoïdaux. On voit que I(t) est constant pendant toute la transmission et reste à cos $(2\pi\Delta ft)$. On voit aussi qu'il y a des « sauts » dans le chronogramme de Q(t) car ce dernier passe de sin $(2\pi\Delta ft)$ à $-$sin $(2\pi\Delta ft)$ en fonction du signal NRZ.

```
alea=ones(1,10000); %genère un tableau de une valeur qui prend soit 1 soit 0 sur 10000 colonnes
m=[m alea]; %ajoute 10000 bits aléatoires après la séquence utilisateur
```

On émet le signal est on a le spectre suivant :



On voit donc clairement les 2 pics qui ont un delta de 200 kHz entre eux.

III / Démodulation FSK

On ajoute la ligne suivante pour permettre la démodulation du signal :

```
data_demod = fskdemod(reception,2,df,Nech_symbol,fe);
```

On démodule donc le signal reçu (reception), avec une valence de 2, la différence entre les 2 fréquence est de 200kHz qui ici est df, le nombre de symbole pour un échantillon est de 32 (ici Nech_symbol), enfin on ajoute la fréquence d'échantillonnage pour faire en sorte que la démodulation respecte le théorème de Shannon.

On constate un problème pour trouver le message que l'on veut envoyer. Ce dernier est difficile à trouver car on ne commence pas forcément la réception au début de la transmission.

IV / Constitution simplifiée d'une trame

Afin de résoudre le problème constaté précédemment, les données seront transmises dans une trame ayant la constitution suivante :

- 16 bits de synchro : 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
- Un champ de préambule de 13 bits : 1 1 1 1 1 0 0 1 1 0 1 0 1
- Le message m(t)

On crée la trame avec la ligne suivante :

```
msg_final = [synchro preambule m] ;
```

On met d'abord les bits de synchronisation, puis les bits de préambule et enfin le message m.

Une séquence de préambule comme celle-ci permet de synchroniser le receveur et l'émetteur et de réduire les erreurs dû à la distorsion. On a ici la séquence de Barker la plus longue avec 13 bits,

Pour permettre de détecter le préambule dans les données démodulés, on ajoute les deux lignes suivantes :

```
detecteur = comm.PreambleDetector(Preambule, 'Input', 'Bit');
indice_preambule = detecteur(data_demod);
```

On crée d'abord le détecteur avec `comm.PreambleDetector` puis on utilise ce détecteur dans les données démodulées et on met les indices de la fin de ces préambules dans la variable `indice_preambule.`

V / Transmission de texte

L'objectif est maintenant de transmettre du texte saisi par :

```
Message=input('Entrez le message à transmettre:','s');
```

On convertit la chaîne de caractère en suite binaire avec la boucle suivante :

```
for n=1:length(msg_ascii)
    caractere_dec = dec2bin(msg_ascii(n));
    caractere_dec = double(caractere_dec) - double('0');
    msg_dec = [msg_dec caractere_dec];
end
```

Dans cette boucle, on transforme un caractère du message ASCII en binaire. On retire ensuite la valeur numérique associé à « 0 » pour permettre d'avoir un tableau. On met ensuite ce tableau dans la variable msg_dec qui va contenir le message à transmettre en binaire.

On ajoute ensuite la partie contenant le nombre de bit du message :

```
longueur_message = length(msg_dec);              % longueur du message en nombre de bits
longueur_binaire = dec2bin(longueur_message, 16);  % conversion en binaire sur 16 bits
longueur_binaire = double(longueur_binaire) - double('0'); % conversion en tableau

msg_final = [synchro preambule longueur_binaire msg_dec] ;
```

On récupère la longueur du message quand il est codé en binaire. On code ensuite cette valeur en binaire que l'on convertit en tableau car c'est un string à la base et on crée ensuite la trame avec la synchro, le préambule, la longueur du message en binaire et enfin le message en binaire.

Enfin, on crée le CRC de notre trame :

```
%création du CRC
CRC_polynomial = [1 0 0 1];  % polynome choisi avec la documentation de comm.CRCGenerator
crcGen = comm.CRCGenerator(CRC_polynomial, 'ChecksumsPerFrame', 1);

msg_final_crc = crcGen(msg_final.');
msg_final_crc = msg_final_crc.';
 % Polygone de CRC choisi ici comme exemple, vous pouvez choisir un autre.
 CRC_polynomial = [1 0 0 1];  % Correspond au polynôme x^3 + 1
 crcGen = comm.CRCGenerator(CRC_polynomial, 'ChecksumsPerFrame', 1);

 msg_final_crc = crcGen(msg_final.');
 msg_final_crc = msg_final_crc.';
```

On crée un polynôme qui permet d'avoir un CRC plus ou moins précis. On crée ensuite le générateur de CRC avec le polynôme créé précédemment. On applique ensuite ce générateur à notre trame msg_final pour ajouter le CRC à la fin du message.

On la transforme ensuite en vecteur ligne pour permettre la transmission.

Réception du message :

Comme précédemment on configure l'adam pluto en récepteur :

```
rx = sdrrx('Pluto', 'RadioID', 'usb:0', 'CenterFrequency', fp, ...
        'BasebandSampleRate', Fech, 'SamplesPerFrame', Nech * 2, ...
        'OutputDataType', 'double', 'ShowAdvancedProperties', true);
```

On définit la fréquence porteuse à laquelle on veut recevoir le signal, la fréquence d'échantillonnage ainsi que le nombre d'échantillons ainsi que le type de donnée en réception.

```
%% Réception des échantillons
reception= rx(); % réception d'une trame de Nech échantillons
reception= reception.';   % on prend la transposée pour avoir un vecteur colonne.
```

On a maintenant dans la variable réception un vecteur colonne avec les données modulées.

On utilise la fonction intégrée à matlab pour démoduler et obtenir nos données en binaire.

```
%% démodumation

data_demod = fskdemod(reception,2,df,Nech_symbol,Fech);

data_demod = data_demod.';
```

Par la suite on détecte le préambule intégré aux données et on note chaque indice du tableau de donnée data demod où ils se trouvent :

```
%% détection préambule

detecteur = comm.PreambleDetector(Preamble, 'Input', 'Bit'); indice_preambule = detecteur(data_demod);
```

On récupère ensuite le message en trouvant la longueur du message transmis sur 16 bits, on prend à partir du 2 ème indice du préambule (2eme fois où on le trouve dans les données) afin d'être sûr d'avoir la trame en entier. Dans m on récupère donc les bits du message en enlevant les 16bits de la longueur du message.

```
%% récupération du message

nombre_bitsmsg = data_demod(indice_preambule(2)+1: indice_preambule(2)+16) nombre_bitsmsg = nombre_bitsmsg.'; nombre_bitsmsg =num2str(nombre_bitsmsg); nombre_bitsmsg_dec = bin2dec(nombre_bitsmsg);

m = data_demod(indice_preambule(2)+17: indice_preambule(2)+16+nombre_bitsmsg_dec);
```

De la même manière que pour la transmission du message on fait l'inverse pour re transformer en chaîne de caractère :

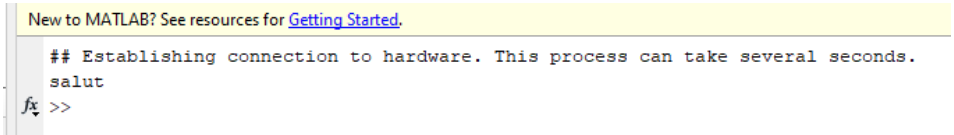
```
m_reshape = reshape(m, 7, []); m_final = ''; for col = 1:size(m_reshape, 2)
bit_sequence = m_reshape(:, col);

bit_str = num2str(bit_sequence', '%d');
bit_str = strrep(bit_str, ' ', ''); %enleve les espaces
lettre = char(bin2dec(bit_str));
m_final = [m_final lettre];


end
```

On peut ensuite envoyer notre message et on obtient bien le message voulu en réception.

Émission :

```
message = 'salut'; %ne pas mettre d'espace sivouplé
```

Réception :



On contrôle ensuite le CRC pour ça on a besoin de la trame en entier d'où les –28 (16 de synchro et 13 de préambule -1 car on est déjà au dernier bit du CRC) et +3 de CRC à la fin.

```
m_finalCRC = data_demod(indice_preambule(2)-28: indice_preambule(2)+16+nombre_bitsmsg_dec+3);
```

On ajoute ensuite une erreur dans le CRC pour tester son fonctionnement. Pour cela on ajoute une ligne dans le fichier de réception qui va regarder si le CRC obtenu est bien celui attendu.

```
%%verif crc
CRC_Polynome =[1 0 0 1];
crcDetection = comm.CRCDetector(CRC_Polynome, "ChecksumsPerFrame", 1);

[~, err] = crcDetection(m_finalCRC);

if err == 0
     disp("Le message est GOOOODD")
else
    disp("c po bon")
end
```

On crée un CRCDetector avec le même polynôme que pour l'émission. On applique ce détecteur au message obtenu en réception. On obtient alors un booléen qui nous donne s'il y a une erreur ou non, donc si la variable err est à 0 cela signifie qu'il n'y a pas d'erreur et si la variable est à 1 il y a une erreur dans la réception. Par exemple si on ajoute une erreur dans le message, on a le résultat suivant pour le message 'salut' :

R&T Semestre 3 2022/2023

SAE31 : Etude et mise en œuvre d'un système de transmission

Emission en M-PSK

Cette partie est volontairement peu guidée. Vous ferez le programme et les mesures afin de remplir le cahier des charges suivant :

Cahier des charges : l'objectif est de simuler, étudier, puis d'émettre à l'aide d'un Adalm Pluto un signal binaire en M-PSK.

- Réalisation de l'émission
  - Données émises : signal binaire aléatoire de Nb=40320bits. Débit de D=20kbits/s.
  - Modulation M-PSK. On prendra **M=4 au début**.
  - Nombre d'échantillons par symbole : Nech_symb=32.
  - Filtre de Nyquist en cosinus raidi de coefficient α=0,4.
  - **La transmission sur l'ADALM Pluto ne devra être réalisée qu'après avoir effectué toutes les simulations.**
- Vous devrez visualiser et analyser :
  - Les chronogrammes de I et de Q
  - Le spectre du signal modulé.
  - La constellation.
  - A l'aide d'un analyseur de spectre, le relevé du spectre du signal émis par l'Adalm Pluto (question à traiter à la fin).
- Vous devrez réaliser :
  - L'étude du l'effet du filtre de Nyquist
  - L'étude de l'effet d'un bruit de transmission.
  - L'étude de l'effet d'un décalage en fréquence et en phase au niveau de la transmission.
- Recommencer l'étude pour différentes valeurs de M (8, 16, …)

On crée d'abord un signal binaire aléatoire avec un débit de 20kbits/s. Le nombre de bits Nb a été diminué pour permettre une exécution plus rapide du programme :

```
Nb=1024;   % nombre de bits à transmettre
D=20000;  %débit en bits/s
data = randi([0 M-1],Nb,1); % création des nombres aléatoires
data = rectpulse(data, Nech_symb);
```

La dernière ligne permet de répéter chaque bits 32 fois pour avoir 32 échantillons.

On module ensuite en MPSK avec M = 4 pour l'instant :

```
M = 4;
data_mod = pskmod(data, M, pi/M); %modulation en mpsk
```

On ajoute ensuite le filtre de Nyquist avec la fonction comm.RaisedCosineTransmitFilter :

```
Nyquist = 0.4;

filtreNyquist = comm.RaisedCosineTransmitFilter('Shape','Normal','RolloffFactor',
Nyquist,'OutputSamplesPerSymbol',Nech_symb, 'Gain', sqrt(Nech_symb));

signal_Nyquist = filtreNyquist(data_mod).';
```

On commente ces dernières lignes pour tout d'abord analyser le signal sans l'effet du filtre de Nyquist. I est la partie réelle du signal complexe tandis que Q est la partie imaginaire de ce signal. On crée donc les chronogrammes des signaux I et Q avec les 2 lignes suivantes :

```
%% Création des signaux I et Q
I = round(real(signal_utile),3);
Q = round(imag(signal_utile),3);
```

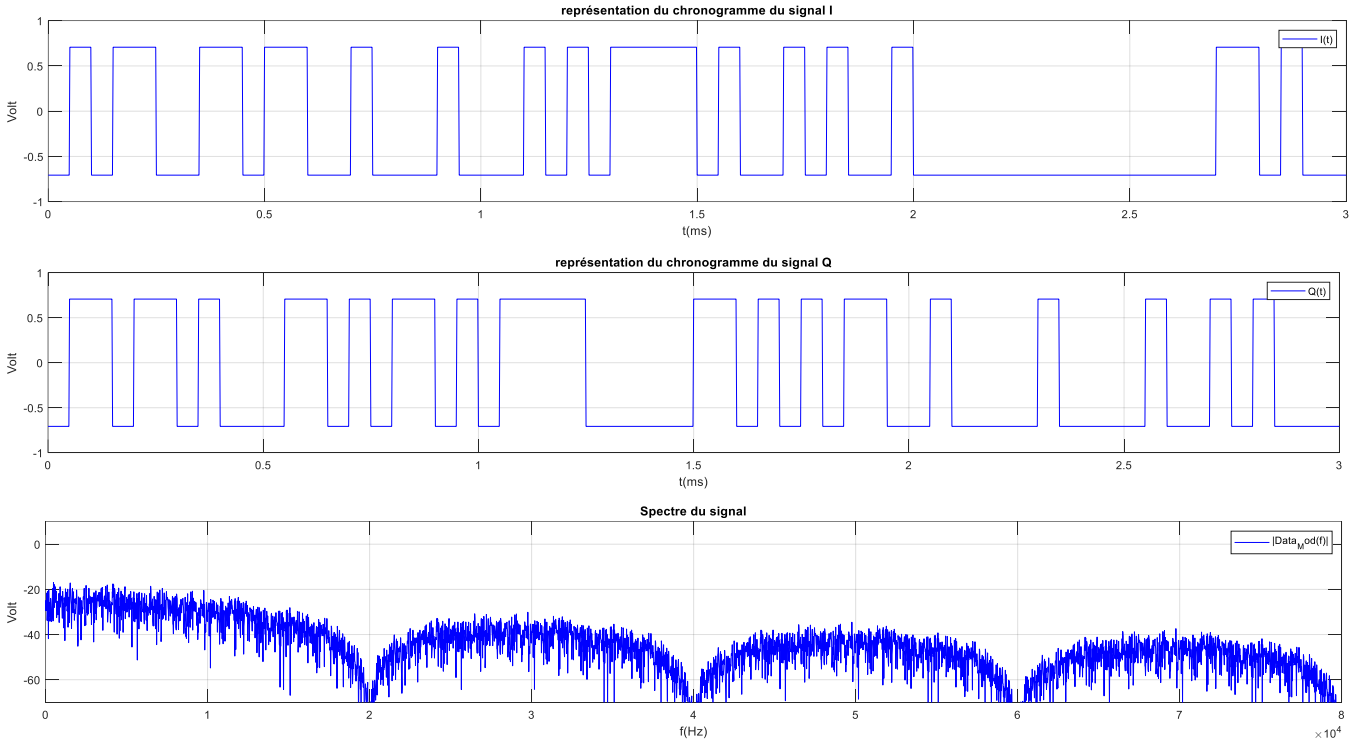
On obtient le spectre du signal modulé en utilisant le programme spectre.m :

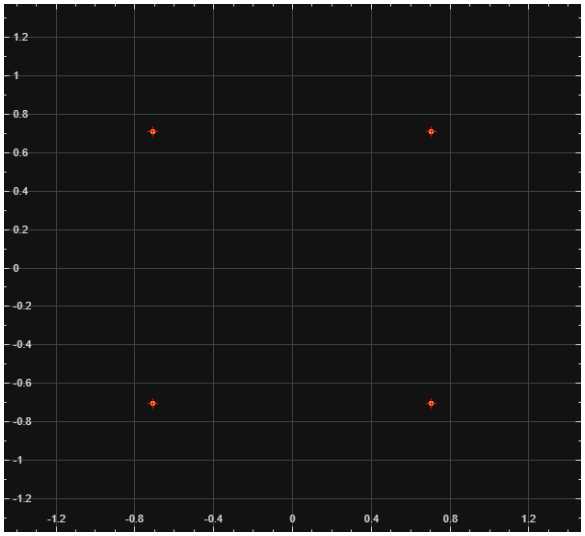
```
[X f]=spectre(data_mod.',fe,Nech);
```

Enfin on crée une constellation grâce à la fonction comm.ConstellationDiagram :

```
constellation = comm.ConstellationDiagram;
constellation(signal_utile);
```

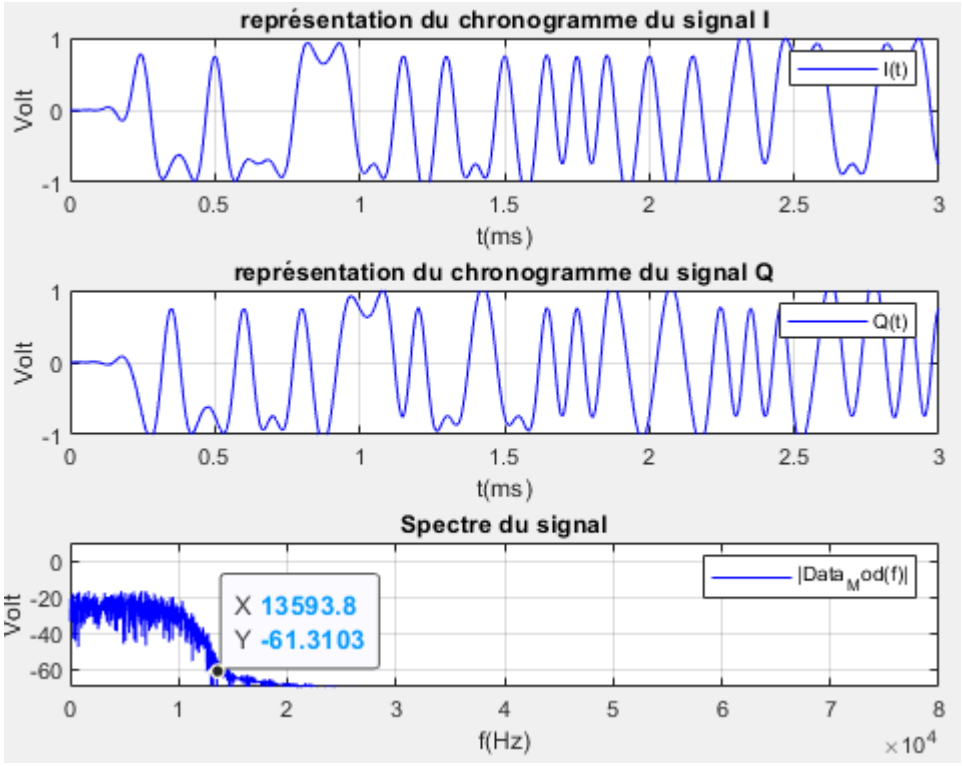
En exécutant le programme on obtient les chronogrammes, le spectre et la constellation suivante :





On a donc bien des chronogrammes qui vont entre 0,707 et -0,707. Ces signaux étant codé en NRZ, on a un spectre pour qui son premier lobe s'arrête à 20kHz soit D. Enfin on a bien une constellation avec 4 points qui sont aux points où ils sont sensés être. L'absence de bruit donne des points parfaitement placés, ce qui n'est pas possible dans une transmission réelle.

On va maintenant voir l'effet d'un filtre de Nyquist sur la transmission. Quand on décommente les lignes qui crée le filtre de Nyquist, on a les chronogrammes et le spectre suivant :

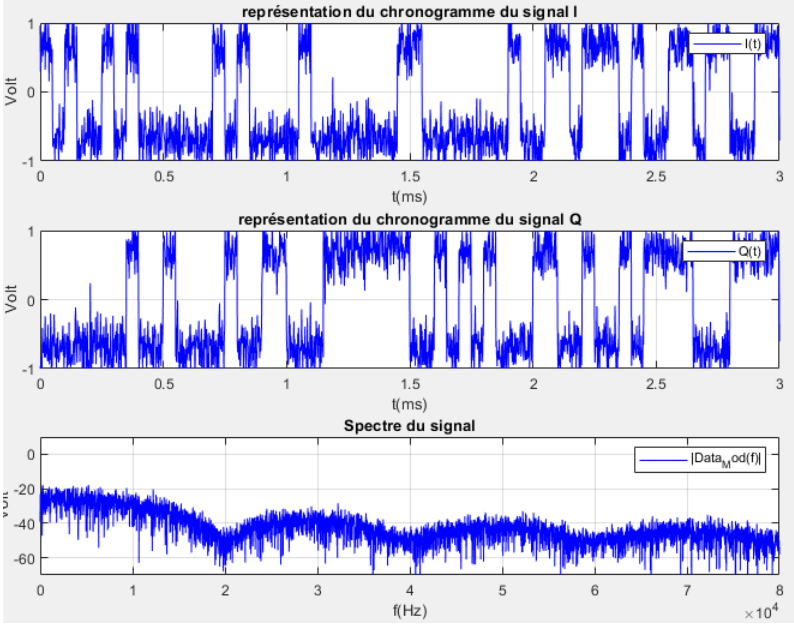


Les chronogrammes sont plus lisses et le spectre s'arrête à environ 14kHz. Cette valeur s'explique par la formule $B = (1 + \alpha) \times \frac{R}{2}$. Ici on a donc : $B = (1 + 0{,}4) \times \frac{20 \times 10^3}{2} = 14000$.

On teste ensuite l'effet du bruit sur l'émission. On ajoute du bruit avec les lignes suivantes :

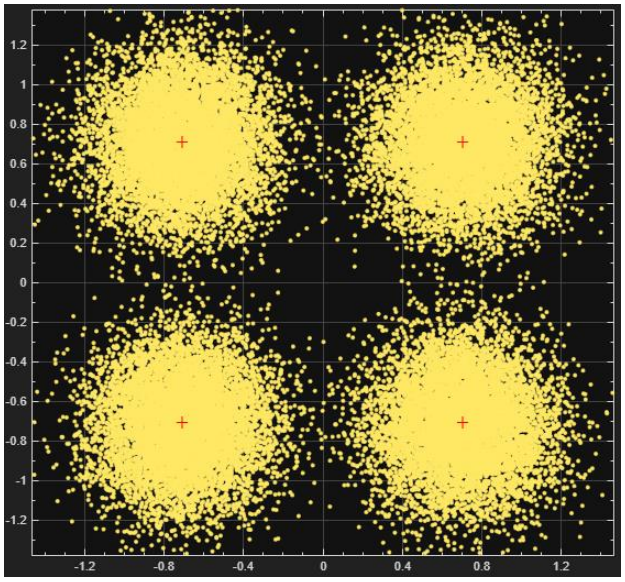
```
Bruit = comm.AWGNChannel;
signal_bruit = Bruit(data_mod);
```

On exécute ensuite le script en faisant attention à transposer les vecteurs à certains endroits. On obtient alors les chronogrammes et le spectre suivant :



On voit clairement que les chronogrammes ne sont plus lisses et sont très bruité. Le spectre est lui aussi très différent, les lobes sont moins prononcés que sur les captures précédentes.

On obtient aussi la constellation suivante :



Les clusters de points sont beaucoup plus larges que sur l'autre constellation vu plus haut. Il est donc plus difficile de savoir ce que signifient certains points.

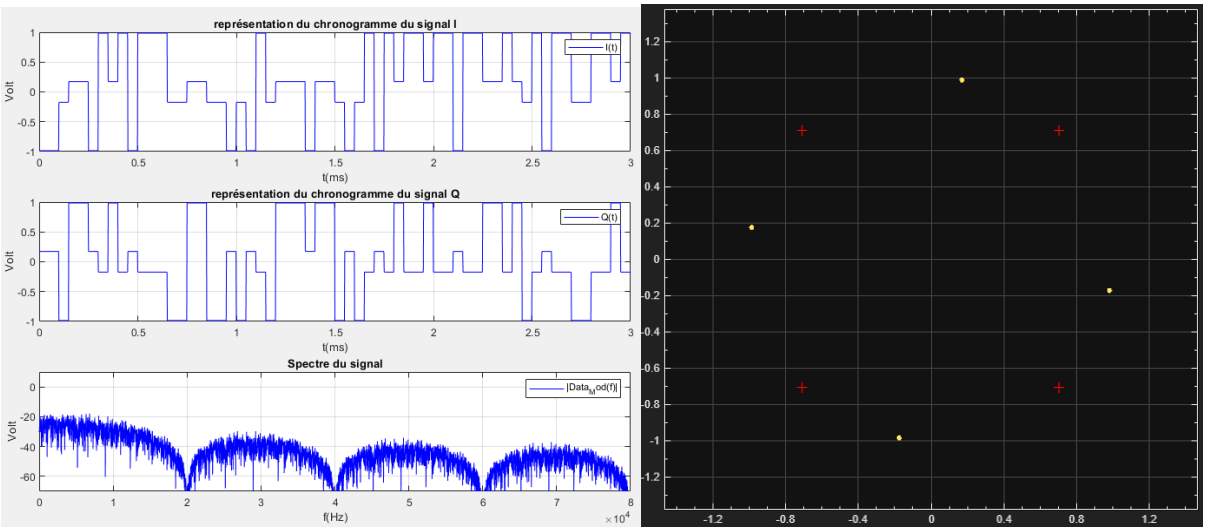
Enfin on exécute le programme avec un décalage de fréquence et de phase. Pour cela on ajoute les lignes suivantes au programme :

```
ModulPhase = comm.PhaseFrequencyOffset("PhaseOffset", 35);
signal_module = ModulPhase(data_mod);
```

Si on veut ajouter un décalage de fréquence peut mettre les lignes suivantes :

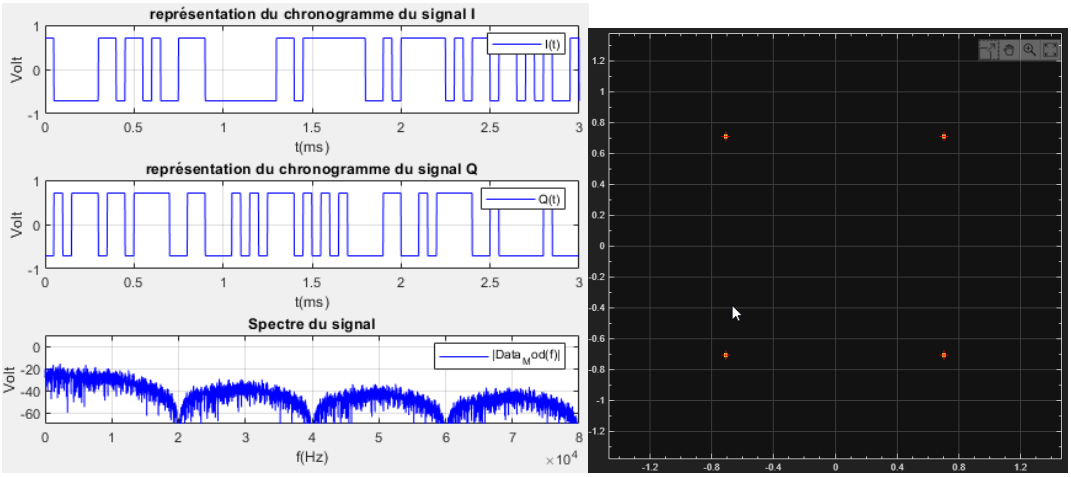
```
ModulPhase = comm.PhaseFrequencyOffset("FrequencyOffset", 1000);
signal_module = ModulPhase(data_mod);
```

Pour un décalage de phase on obtient les figures suivantes :



On voit que les points de la constellation ne sont plus situés sur les croix rouges. Ils sont décalés de 35° comme on l'a spécifié dans le programme.

Avec un changement de fréquence on obtient les figures suivantes :



Un décalage de fréquence n'a aucun effet car nous modulons notre signal avec la phase.