Hugo HELBLING
Mathilde PICHON

TD 5 MATLAB TRAITEMENT DU SIGNAL

L'objectif de ce TD est de savoir échantillonner un signal et élaborer son spectre par FFT. Savoir analyser et filtrer un signal numérique à partir des méthodes classiques.

A) Échantillonnage d'un signal porte

Soit x(t) le signal porte d'amplitude 1 et de largeur (-0,5s à +0,5s).

1. Détermination de la fréquence d'échantillonnage Fe en appliquant Shannon

Le théorème de Shannon nous dit qu'il faut échantillonner à une cadence au moins deux fois plus grande que la fréquence maximale contenue dans x(t).

→ **Fe > 2.Fo** On constate aisément que **Fe = 8 Hz** convient. On a donc **Te = 1/Fe**

Enfin, notre fréquence d'échantillonnage de 8 Hz indique qu'on aura **8 échantillons par seconde,** et sachant que notre FFT ira de -Fe/2 à +Fe/2, on aura :

N = 64 = Nombre de points total dans le signal. Enfin, on a une résolution numérique

Rf = 0,0125 et la relation Rf = N / Fe.

2. Élaboration du signal numérique x(n) obtenu par échantillonnage de x(t)

On crée donc le script suivant sur Matlab :

```
Fe = 8;
Rf = 0.125 ;
N = Fe / Rf ;
Te = 1 / Fe ;
t = (-N/2)*Te:Te:(N/2)*Te-Te;
p = zeros(1,N);
p(N/2-3:N/2+4)=ones(1,8);
y = heaviside(t+0.5)-heaviside(t-0.5);
plot(t,y,'*')
plot(t,p,'o')
```

On obtient donc la fonction Porte de 2 façon différentes :

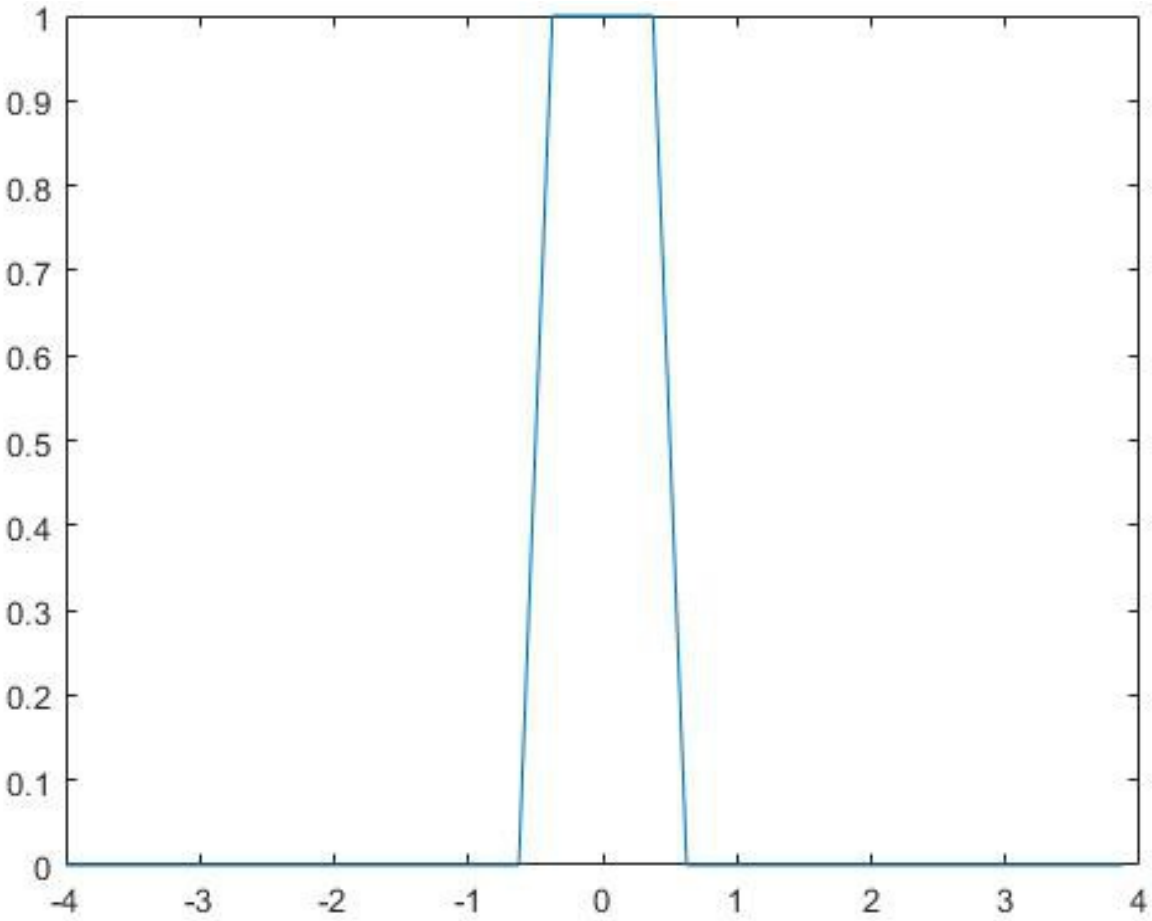


Figure 1: affichage de la fonction porte en continu

On observe la fonction porte par la méthode ONES :

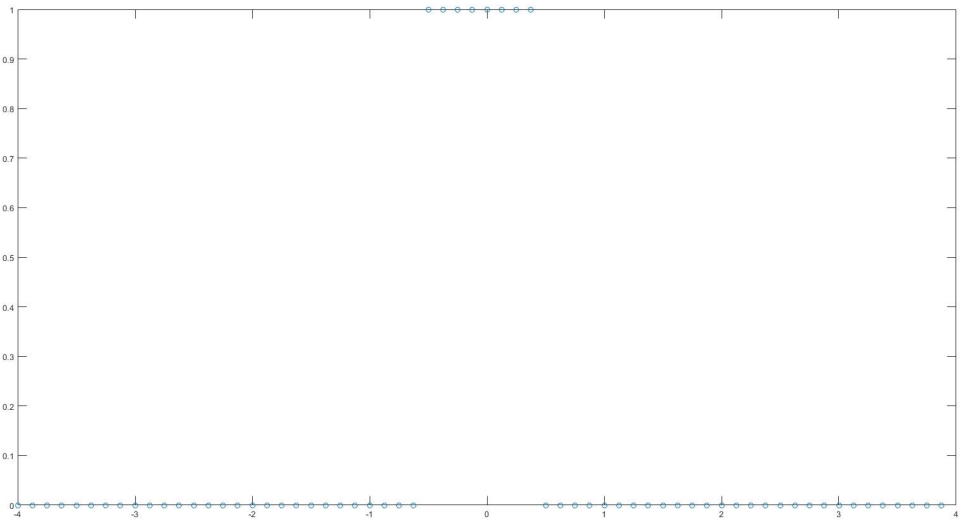


Figure 2. Affichage de p

Puis, on observe la fonction porte avec la méthode FORUM

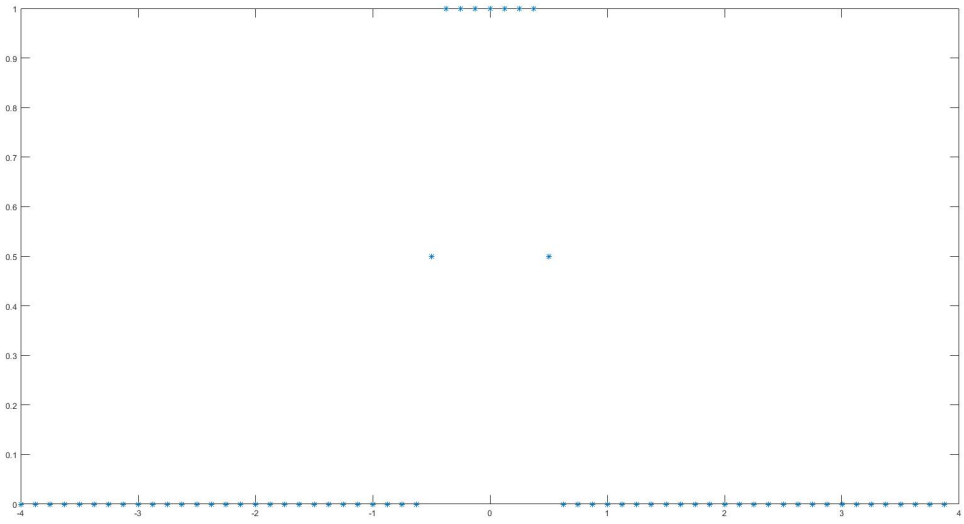


Figure 3. Affichage de y

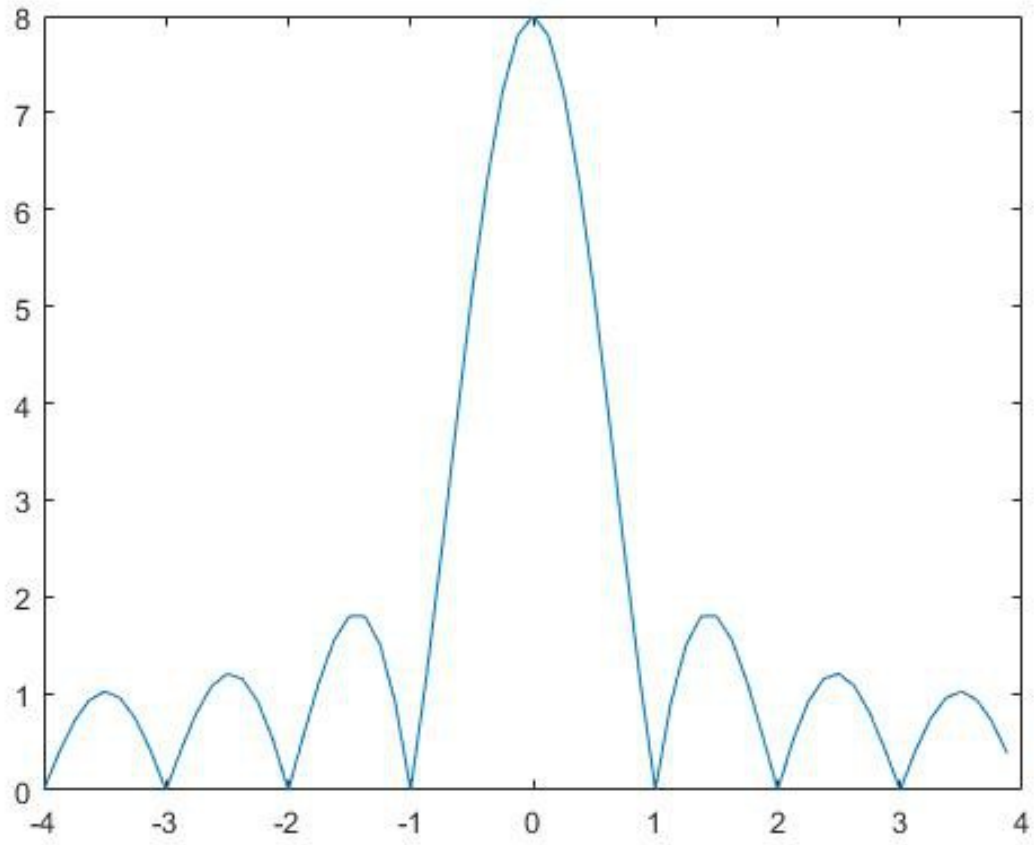
Observations : Si on affiche la fonction en continu, on observe aucune différence. Cependant, en discontinue, on remarque bien que la fonction est construite différement. On verra par la suite que cela a une influence sur le spectre.

3. Calculer le spectre de x(n) par et tracer son spectre

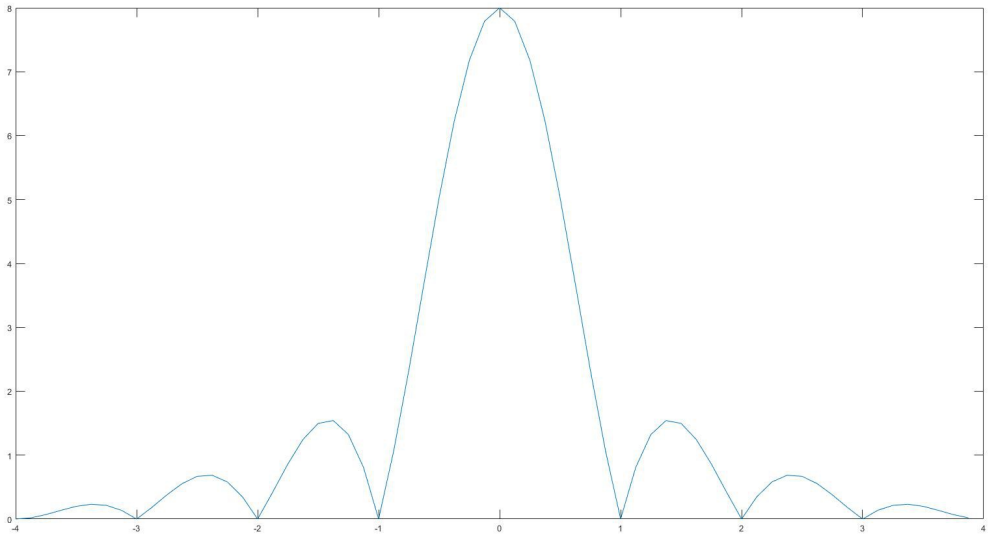
On veut maintenant tracer le spectre du signal, on utilise donc les fonctions fft et fftshift pour réorganiser les valeurs :

```
Fe = 8;
Rf = 0.125 ;
N = Fe / Rf ;
Te = 1 / Fe ;
t = (-N/2)*Te:Te:(N/2)*Te-Te;
p = zeros(1,N);
p(N/2-3:N/2+4)=ones(1,8);
y = heaviside(t+0.5)-heaviside(t-0.5);
plot(t,y,'*')
plot(t,p,'o')
fp = fftshift(abs(fft(p)));
fd = fftshift(abs(fft(y)))*Te;
plot(t,fp)
plot(t,fd)
```

On observe ainsi la fft de la fonction ONES :



On observe ensuite la fft de la fonction FORUM :



La fonction p est créé avec la fonction Ones. La fonction Ones est la fonction qu'on crée nous même.

La fonction y avec Heaviside.

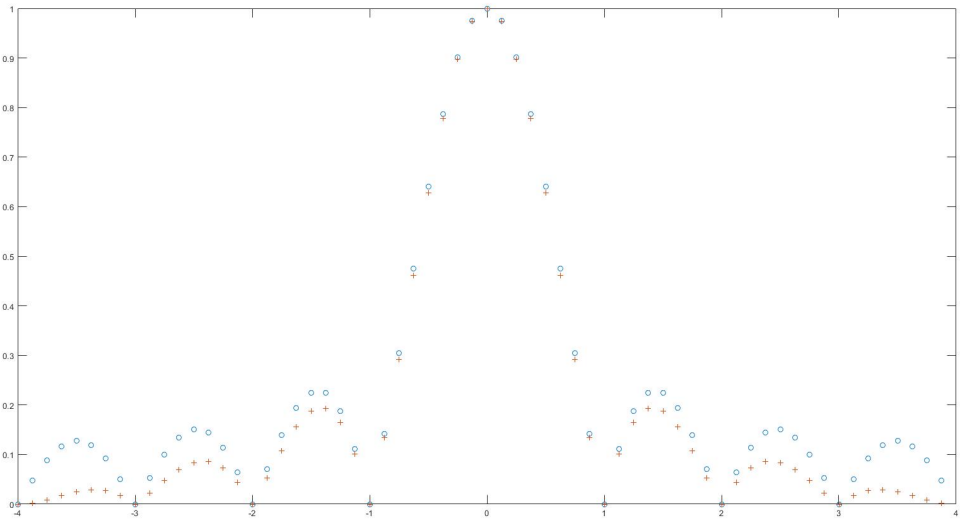
On se rend compte que Heaviside est moins efficace que Ones d'après les deux courbes précédentes donc il est preferable de construire la fonction de nous même.
On se rend compte que le théorème de Shannon ne peut être respecté car la fréquence d'un sinus cardinal est infinie dûe aux oscillations.

On veut comparer par superposition avec le spectre obtenu en continu :

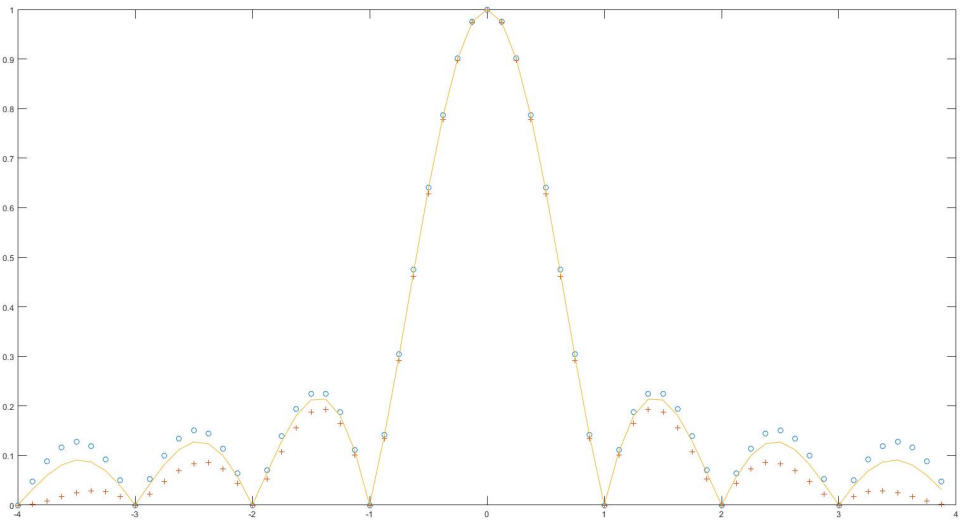
clear

```
% Fe = 8;
% Rf = 0.125 ;
% N = Fe / Rf ;
% Te = 1 / Fe ;
% t = (-N/2)*Te:Te:(N/2)*Te-Te;
% f = -Fe/2:Rf:Fe/2-Rf;
% p = zeros(1,N);
% p(N/2-3:N/2+4)=ones(1,8);
% y = heaviside(t+0.5)-heaviside(t-0.5);
% plot(t,y,'*')
% plot(t,p,'o')
% fp = fftshift(abs(fft(p)))*Te;
% fd = fftshift(abs(fft(y)))*Te;
% a = abs(sinc(f));
% plot(f,a)
% plot(t,fp,'o',t,fd,'+',f,a)
```

On obtient :



On superpose ensuite nos deux spectres des fonctions (Ones et Heaviside) avec le spectre continu :



On obtient bien un sinus cardinal pour la transformée de Fourrier de la fonction porte.

En changeant la fréquence d'échantillonnage, on change ainsi le nombre de points de x(n). On se rend compte que si on augmente Fe, on augmente la précision du spectre mais si on diminue Fe, on obtient qu'une partie du signal et donc une perte d'informations.

Ensuite, en prenant Fe à une valeur constante, on augmente le nombre de points. On peut ainsi observer l'effet du zéro-padding, c'est à dire l'augmentation de la durée du signal par injection de valeurs nulles avant et après.

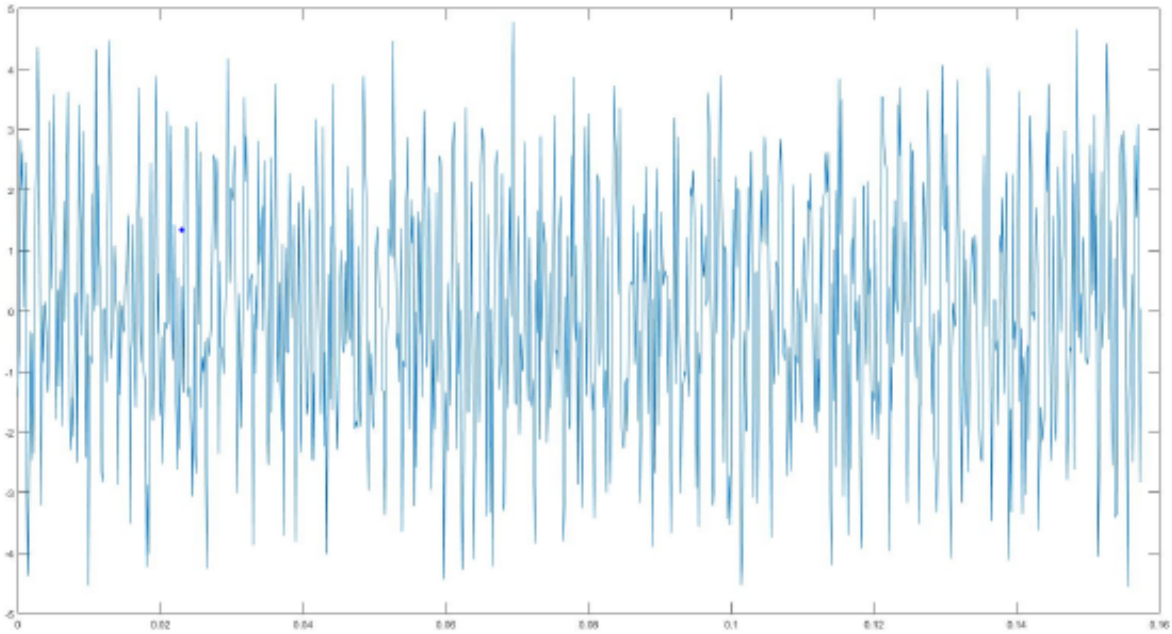
B) Traitement numérique d'un signal par filtrage IIR et FIR

On importe tout d'abord le fichier *mesure,* comportant un vecteur de taille 2x1024. La première ligne donne le temps, la seconde les valeurs correspondantes du signal échantillonné.

1. Analyse du signal

Tout d'abord, on affiche la courbe donnée par le fichier *mesure.*
Pour cela, on utilise le script suivant :

```
tt = mesure(1,:); % axe des abscisse avec la première ligne du vecteur
m = mesure(2,:); % axe des ordonnées correspondants aux valeurs du signal
stest = sin(2*pi*50*tt); % cette fonction servira de test pour vérifier nos fft
plot(tt,m) % on affiche la courbe de m en fonction de tt
```



Signal de sortie récupéré à partir du fichier *mesure.dat*

Ensuite, on cherche à analyser le signal par FFT. Pour cela, on utilise le script suivant, où les commentaires associés permettent d'expliquer les lignes de codes :

```
Tee=tt(1,2)-tt(1,1); % période d'échantillonnage
Fee = 1/Tee; % fréquence d'échantillonnage
NN = length(m); % nombres de points échantillonnés, à savoir 1024
Rff = Fee/NN; % résolution numérique
basef=-Fee/2:Rff:Fee/2-Rff; % création de l'axe des abscisses
fo_me = fftshift(abs(fft(m)))*(1/NN); % calcul de la fft qu'on normalise
fft_test = fftshift(abs(fft(stest)))*(1/NN); % fft de la fonction test
plot(basef,fo_me) % affichage de la fft du signal issu de mesure.dat
plot (tt,stest) % affichage de la fonction test
plot(basef,fft_test) % affichage de la fft de la fonction test
```

Remarques :

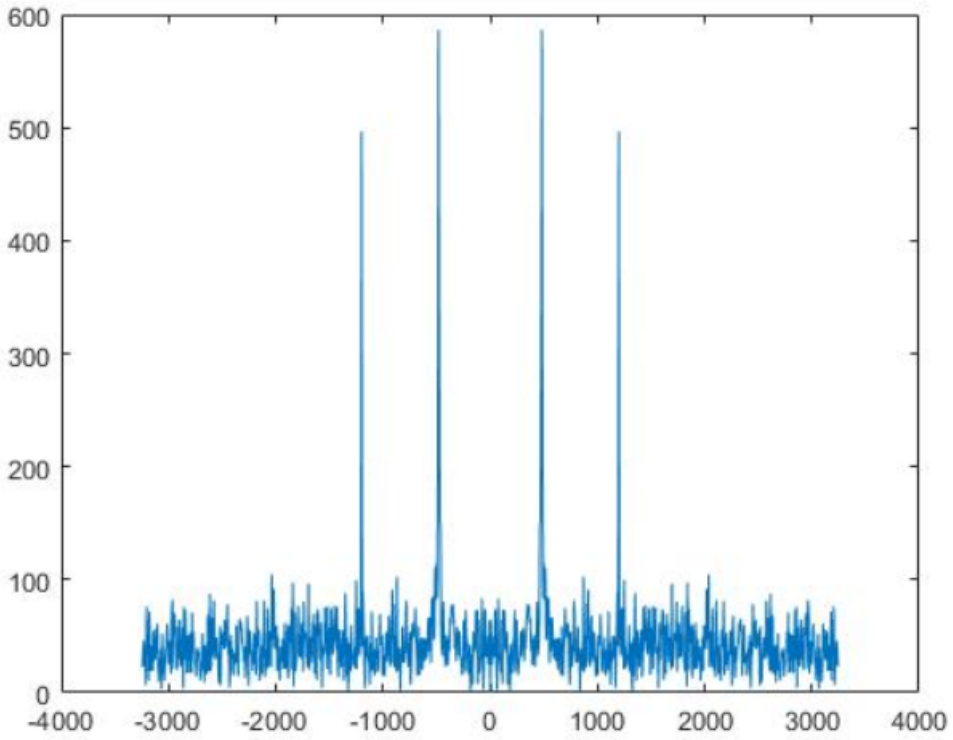
_ On vérifie que les vecteurs *basef,fo_me…* ont bien une taille de 1024 sinon… il y a une erreur. C'est pour ça, par exemple, que l'on va de -Fee/2 à Fee/2 – Rff.

_ La fonction test est une fonction dont on connaît la transformée de Fourier. Cela nous permet de vérifier la justesse de notre script.

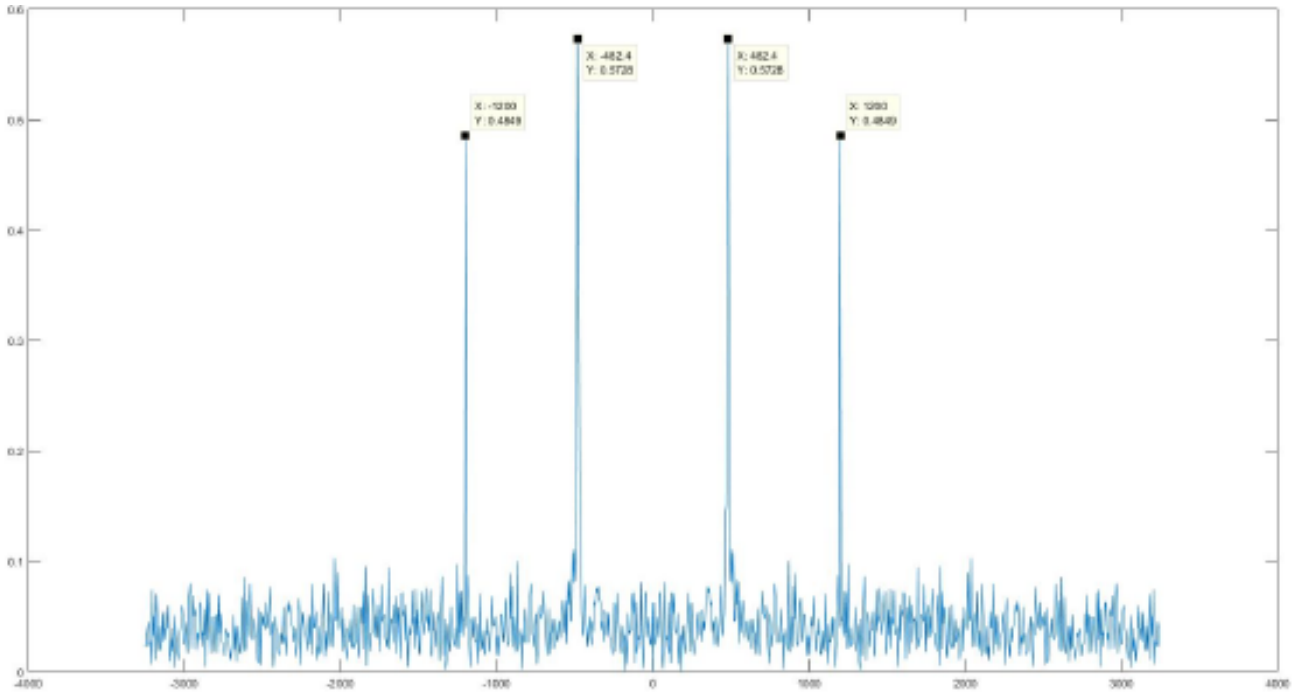
_ Nous avons fait la partie une et la partie deux du TD dans le même script, d'où les noms des variables un peu farfelues afin de ne pas répéter celles de la partie une.

_ Le terme en (1/NN) permet de normaliser la transformée de Fourier.

On obtient la transformée de Fourier du vecteur *mesure* suivante :



Transformée de Fourier non normalisée



Transformée de Fourier normalisée

Analyse de la transformé de Fourier

La forme de la transformé de Fourier est la suivante :

A_1 * sin(2*pi*f_1*t) + A_2 * sin(2*pi*f_2*t) + B(t)

Les 4 pics correspondent aux deux sinus, respectivement aux fréquences 482,4Hz et 1200Hz. Comme on peut le voir sur la courbe.

De plus, le terme B(t) correspondant au bruit se voit sur la courbe au niveau des oscillations qu'on observe un peu au-dessus de 0, et qui dure tout au long du signal.

On ne souhaite que garder les hautes fréquences, les plus basses fréquences sont perturbées. On va donc mettre en œuvre un filtre passe-haut avec deux méthodes, un filtre RII et un filtre RIF.

2. Filtrage du signal.

2.1 Technique RII (Réponse Impulsionnelle Infinie).

Principe : passage continu-numérique par transformation bilinéaire (Tustin). On prend un filtre de type Butterworth.

On a :

$$\textbf{Forme RII}: C(z) = \frac{\sum_{j=0}^{M} b_j z^{-j}}{\sum_{i=0}^{N} a_i z^{-i}} \quad \text{fraction rationnelle avec } N \geq M \text{ (causalité) et } a_0 = 1$$

On souhaite conserver la fréquence à 1200 Hz, on utilise donc une bande passante avec une **fréquence de basse coupure *fbc* telle que :**

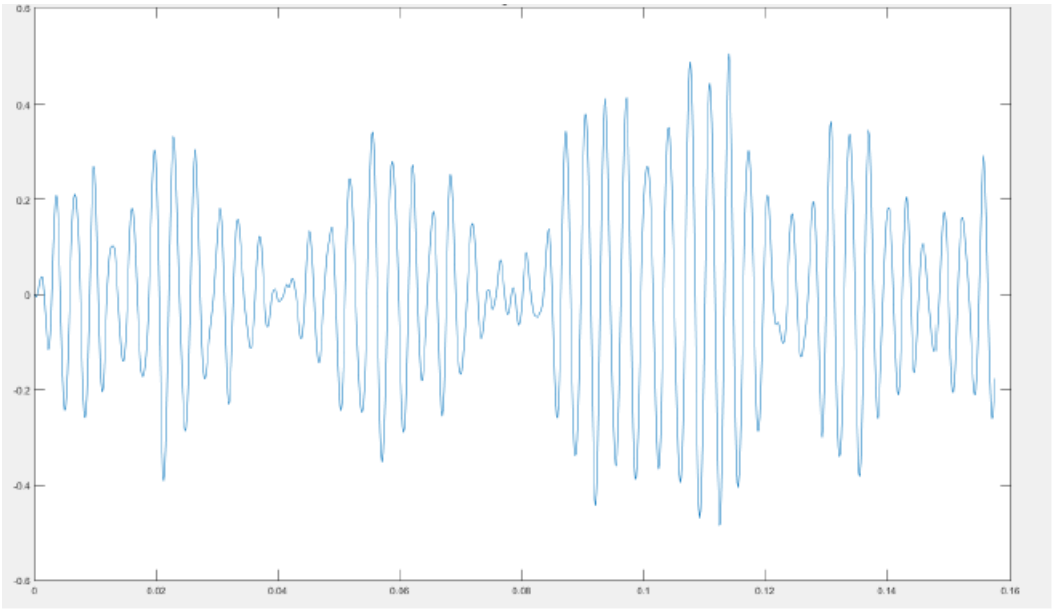
fbc = 1150 Hz

Et une fréquence de haute coupure *fhc* telle que :

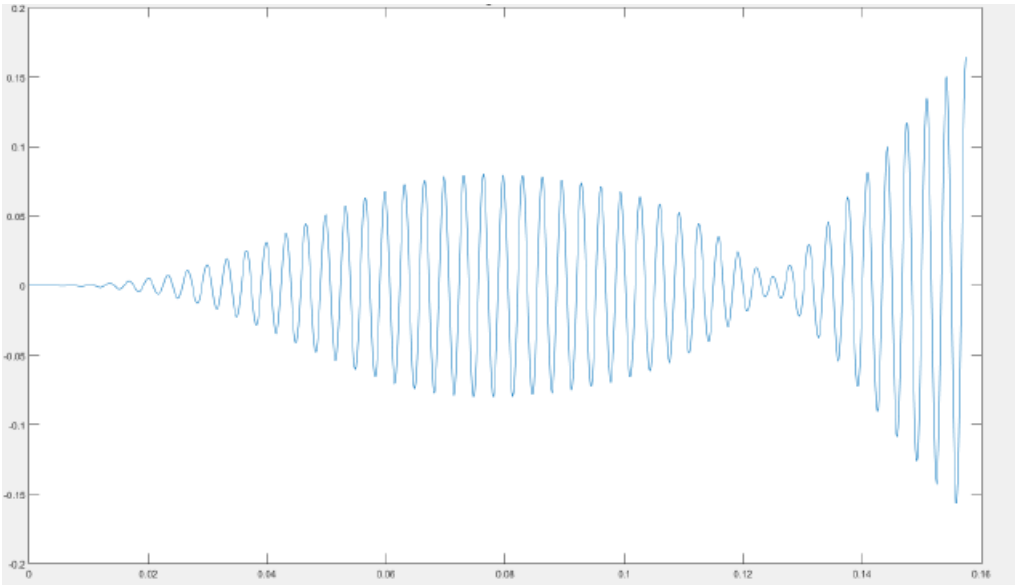
fhc = 1250 Hz

On utilise alors le script suivant :

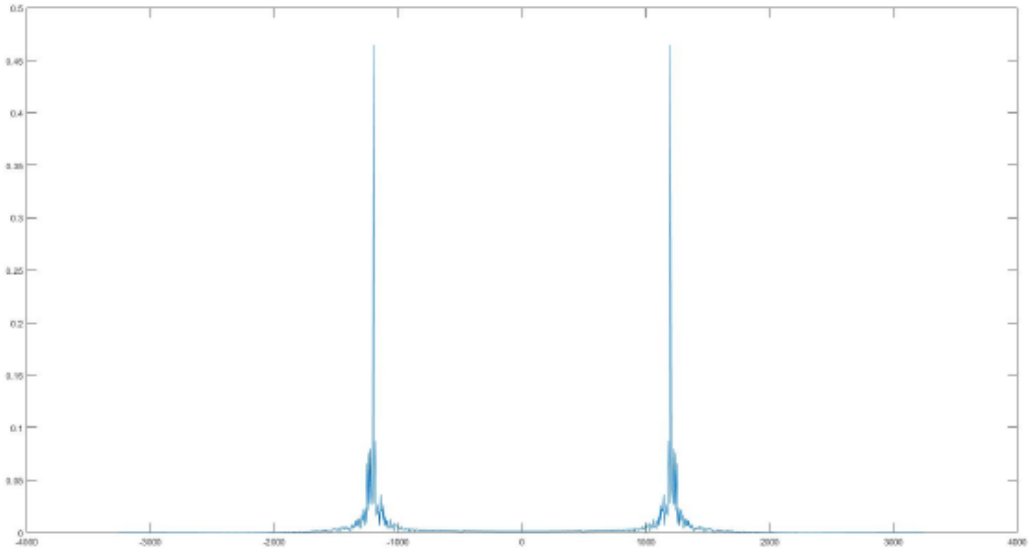
```
fbc = 1150 ; % fréquence de basse coupure
fhc = 1250 ; % fréquence de haute coupure
ordre = 2 ; % ordre du filtre de butterworth
wn = [ fbc fhc ] * 2/Fee ; % bande passante normalisée (entre 0 et 1)
[b,c] = butter (ordre,wn); % coeff de fonction transfert filtre butterworth
nf = filter(b,c,m); % filtrage du signal de sortie mesure
fft_nf = fftshift(abs(fft(nf)*1/NN); % calcul de la fft du signal filtré
fvtool(b,c); % affiche la réponse du filtre : gain, phase, pôles...
figure
plot(basef,fft_nf) % affichage de la fft du signal filtré
figure
plot(tt,nf) % affichage du signal filtré
```



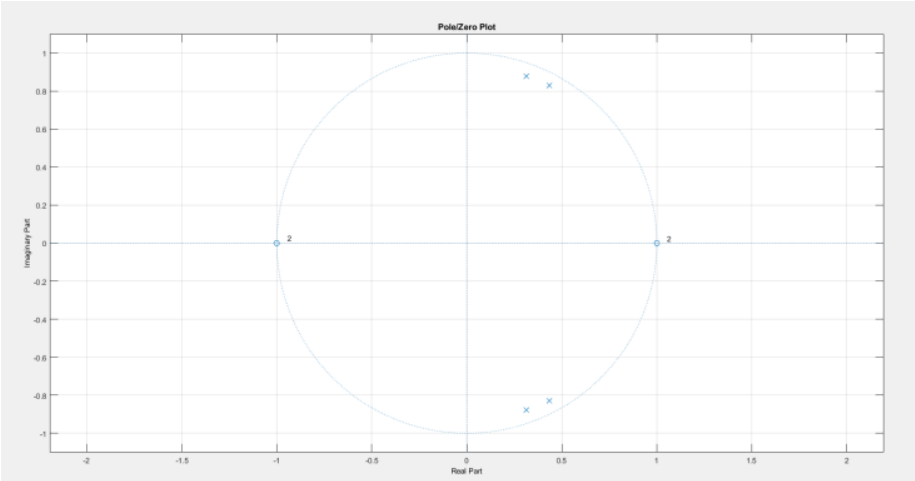
Signal filtré par Butterworth RII (ordre 2)



Signal filtré par Butterworth RII (ordre 4)



Transformée de fourier du signal filtré avec un filtrage de type RIF



Affichage des zéros pour un RII de bande passante (1100,1300)

Interprétations.

- Plus la largeur de la bande passante est réduite, **plus la qualité du filtrage** sera grande. Toutefois, il ne faut pas prendre une bande passante **trop réduite,** sous peine de perdre trop d'informations.

- Plus l'ordre du filtre **est grand, plus la qualité du filtrage sera grande.** Toutefois, le **retard** sera également plus important.

- Avec un filtre RII, on peut à la fois **tirer le signal à zéro** et **jouer sur la résonance.** De plus, l'ordre du filtre RII correspond à l'ordre du **dénominateur.**

2.2. Technique RIF.

Principe : technique dite par fenêtrage et troncature. On spécifie la réponse fréquentielle désirée H(exp(jw)) comme étant celle d'un filtre idéal (gain unitaire dans la BP, nul ailleurs.) Par définition, cette fonction est continue et périodique. On calcule par Fourier inverse les coefficients h(n) qui sont identiques à ceux de la série (**infinie !)** de Fourier. Enfin, troncature et régularisation sur N points par une fonction fenêtre w(n) (type Hamming par défaut) tel que b(n)=h(n)*w(n).
Noter que sous la forme RIF, les coefficients h_n sont les mêmes que ceux du filtre H(z).

On a :

$$H(z^{-1}) = a(0) + a(1) * z^{-1} + \ldots$$

Ici, les paramètres de réglages sont **la longueur de la réponse impulsionnelle et la bande passante.**

On a donc le script suivant :

```
fbc = 1150 ; % fréquence de basse coupure
fhc = 1250 ; % fréquence de haute coupure
```

```
wn = [ fbc fhc ] * 2/Fee ; % bande passante normalisée (entre 0 et 1)
ordre1 = 40 ; % longueur de la réponse impulsionnelle finie
b1 = fir1(ordre1,wn);% creation du filtre RIF avec un fenêtre de Hamming
no = filter(b,1,m);% filtrage du signal de sortie mesure
fft2 = fftshift(abs(fft(no))*1/NN);% calcul de la fft du signal filré
fvtool(b1);% comme pour le RII sauf que c'est sur le RIF
figure
plot(basef,fft2) % affichage de la fft du signal filtré
```

Notons bien la différence avec le filtre RII : **ici, l'ordre du filtre correspond aux nombres de 0. Par conséquent, avec un filtre RIF, on ne peut pas jouer sur la résonance comme pour le filtre RII.**

Malheureusement, nous avons perdu la courbe sur notre clé usb et lorsqu'on relance le programme, Matlab refuse de reconnaître le fichier mesure ou du moins, on n'a pas su lui faire reconnaître. On a tout de même noté les observations suivantes :

Observations :

- Plus le nombre de zéros (donc, l'ordre du filtre) est grand, plus le filtrage sera bon mais le retard sera également plus **important.** Le tout est donc de trouver un bon compromis entre retard et filtrage convenable. En filtrant, il y a de toute façon une perte d'informations, l'essentiel étant de trouver les paramètres cohérents selon ce que l'on veut.

- Un mauvais choix d'ordre s'avère plus grave que pour un filtre RII car si l'ordre est trop petit, on ne met pas à zéro toutes les parties du signal que l'on souhaite ne pas laisser passer.

CONCLUSION

Ce TD nous a montré l'importance du choix de la fréquence d'échantillonnage, de la construction même du signal, car en construisant un même signal différemment on a des soucis de transformé de Fourier.
On a également vu les différences entre filtre RII et RIF : **le filtre RIF est plus intuitif, on met à zéros les parties du signal qu'on ne veut pas prendre en compte sans pouvoir jouer sur la résonance, tandis qu'avec un filtre RII, on peut jouer sur la résonance.**
Ceci étant dit, il existe des points communs à ces deux méthodes :
Tout d'abord, si l'on gagne en précision, on perd en rapidité et inversement. De plus, plus la bande passante est petite, plus la qualité du signal filtré sera bonne. Il faut toutefois une bande passant suffisamment grande pour avoir l'information souhaité.

De manière général, on a pu comprendre que :

- **Pour traiter un signal, il faut clairement savoir ce que l'on souhaite garder et ce qu'on souhaite éliminer.**

- Traiter un signal, c'est faire des concessions. On ne peut pas être gagnant sur tous les points : par exemple en rapidité et en précision. Pour bien traiter un signal il faut trouver le bon compromis, nous permettant d'avoir un signal filtré qui garde sa cohérence et qui est satisfaisant avec ce que nous voulions en théorie.