Tutoriel transformée de Fourier discrète avec Scilab

1. Introduction

Ce document présente une initiation à la transformée de Fourier discrète et propose d'en explorer quelques aspects de manière concrète au moyen du logiciel Scilab[1].

La transformée de Fourier discrète est un outil puissant d'analyse, et au besoin de retouche, des propriétés spectrales d'une fonction f de variable la réelle t. Pour le comprendre, il est plus simple de raisonner d'abord sur une fonction f périodique, de période T. Dans ce cas, les étudiants voient dans les autres disciplines que f présente une signature spectrale faite de raies aux fréquences discrètes $0, \frac{1}{T}, \frac{2}{T}, \frac{3}{T}, \ldots -\frac{1}{T}, -\frac{2}{T}, -\frac{3}{T}, \ldots$. Ces raies ont pour amplitudes complexes les coefficients de Fourier dits exponentiels :

$$c_k = \frac{1}{T}\int_0^T f(t)e^{-\mathrm{i}2\pi k\frac{t}{T}}dt$$

où $k \in \mathbb{Z}$. En pratique, on calcule rarement ces intégrales. On les approche par la formule des rectangles. On découpe uniformément l'intervalle $[0;T]$ en n petits intervalles de taille $T_e = \frac{T}{n}$, sur lesquels on remplace la quantité à intégrer par sa valeur en l'extrémité gauche. Si bien que :

$$c_k \approx \frac{T_e}{T}\sum_{l=0}^{n-1} f(lT_e)\, e^{-\mathrm{i}2\pi k\frac{lT_e}{T}}$$

ou encore :

$$c_k \approx \frac{1}{n}\sum_{l=0}^{n-1} f(lT_e)\, \omega^{-k.l} \text{ avec } \omega = e^{\mathrm{i}\frac{2\pi}{n}}$$

Autrement dit, et au facteur multiplicatif $\frac{1}{n}$ près, c_k est à peu près le k −ième coefficient de la transformée de Fourier dicrète (TFD) $(F_0, F_1, \ldots, F_{n-1})$ de la séquence $(f_0, f_1, \ldots, f_{n-1}) = \big(f(0.T_e), f(1.T_e), \ldots, f\big((n-1).T_e\big)\big)$. La précision de la méthode est d'autant plus grande que l'exponentielle varie lentement sur la durée T_e, donc que la période $\frac{T}{k}$ de cette exponentielle est grande devant T_e, donc que $\frac{n}{k}$ est grand.

En bref, les premières valeurs $F_0, F_1, F_2, \ldots$ de la TFD approchent convenablement les premiers coefficients de Fourier d'indices positifs $c_0, c_1, c_2, \ldots$. Les dernières valeurs de la TFD sont tout aussi précieuses : comme $\omega^{-k.l} = \omega^{-(n-k).l}$, les valeurs $F_{n-1}, F_{n-2}, F_{n-3}, \ldots$ approchent $c_{-1}, c_{-2}, c_{-3}, \ldots$. Plus n est grand, plus l'observation du spectre est large.

Quand la fonction f n'est pas périodique, son spectre est un « continuum » de raies. La TFD approche celles aux fréquences $0, \frac{1}{T}, \frac{2}{T}, \frac{3}{T}, \ldots -\frac{1}{T}, -\frac{2}{T}, -\frac{3}{T}$ où T ne désigne plus une période, mais un temps d'observation.

[1] Voir ici : http://www.scilab.org

2. Échantillonnage

Il est usuel de nommer « signal » une fonction de variable réelle, cette variable étant assimilée au temps (mesuré en secondes), la valeur de la fonction à l'instant t désignant la grandeur physique à laquelle on s'intéresse (elle peut être de diverses natures : tension, intensité, pression, etc.).

Les signaux traités peuvent être très réguliers ou au contraire assez chaotiques ou « bruités », et ils peuvent être périodiques ou non. Il est cependant commode de considérer tout signal comme périodique, quitte à tronquer le signal sur une durée limitée[2] et à considérer que le signal se répète au-delà de cette durée.

Dans les systèmes numériques (au contraire des systèmes analogiques), le signal n'est pas mesuré en continu mais seulement à intervalles réguliers : c'est ce qu'on appelle l'échantillonnage du signal. L'intervalle de temps entre deux prises de mesure successives s'appelle le **pas d'échantillonnage**. Après calibrage, les mesures sont ensuite disposées sur une échelle discrète et converties en nombres entiers. Cette opération, dite de **quantification,** n'est pas prise en compte dans ce présent document mais elle ferait partie intégrante d'un processus de numérisation d'un signal.

Ici, l'échantillonnage d'un signal sur une certaine durée produira donc une liste de nombres réels. Cette liste peut se manipuler comme un vecteur (ou tableau à une seule ligne).

Quelques commandes Scilab utiles pour créer et manipuler des vecteurs :

Attention, dans Scilab les vecteurs sont indexés à partir du rang 1 (et non 0) !
On peut créer un vecteur dans Scilab en donnant la liste de ses composantes ; ainsi :
`t=[1,3,8,17]`
crée un vecteur à quatre composantes, la dernière étant `t(4)` , valant 17 (ne pas essayer de former `t[4]`, cela ne marche pas).
Il est également possible de créer rapidement un vecteur (ou liste) ayant des composantes régulièrement espacées, c'est idéal pour représenter la liste des instants de prise de mesure :
`t=0.1:0.5:3`
permet de créer une liste de valeurs démarrant avec la valeur 0,1, avançant de 0,5 en 0,5 et ne dépassant pas 3 (la dernière est donc ici égale à 2,6).
Une fois que des vecteurs sont créés, on peut aisément leur appliquer des calculs. Par exemple,
`v=0.002*t`
donne un vecteur `v` dont les composantes sont égales à celles de `t` multipliées par 0,002[3]. Dans la même logique, l'addition des vecteurs fonctionne :
`w=v+t`
donne un vecteur dont les composantes résultent de l'addition de celles de `v` et de celles de `t` de même rang (mathématiquement parlant, on obtiendra ici $w = 1{,}002t$).
On peut aussi extraire un sous-vecteur en spécifiant les indices de début et de fin de tranche :

```
-->w(2:5)
 ans  =
    0.6012    1.1022    1.6032    2.1042
```

Pour la multiplication, il faut distinguer le produit scalaire, qui se forme ainsi : `w*t'` (pour des vecteurs en ligne), du produit composante par composante, qui s'écrit : `w.*t` , et redonne un vecteur.
On peut enfin « appliquer » une fonction sur un vecteur. Par exemple, la fonction carré :

```
-->function y=carre(x);y=x.*x;endfunction
-->carre(t)
 ans  =
    0.01    0.36    1.21    2.56    4.41    6.76
```

[2] Mathématiquement parlant, c'est une restriction (par exemple à un intervalle d'amplitude $N \times Te$).
[3] C'est l'opération de multiplication d'un vecteur par un scalaire dans un espace vectoriel.

Exemples de création et d'échantillonnage d'un signal

Une bonne base pour créer des signaux est fournie par les fonctions sinusoïdales (une alternative intéressante étant fournie par les fonctions en « créneaux » ou en « dents de scie »).

La période (ou pas) d'échantillonnage est tée T_e. On en déduit la fréquence d'échantillonnage, $F_e = \frac{1}{T_e}$, qui se mesure en Hz, ainsi que la pulsation d'échantillonnage $\omega_e = 2.\pi.F_e$, qui se mesure en rad.s^{-1}.

On commence à mesurer le signal à l'instant $Nmin \times T_e$ et on termine à l'instant $Nmax \times T_e$, le nombre de mesures étant ainsi $Nmax - Nmin + 1$ et le nombre d'intervalles $Nmax - Nmin$.

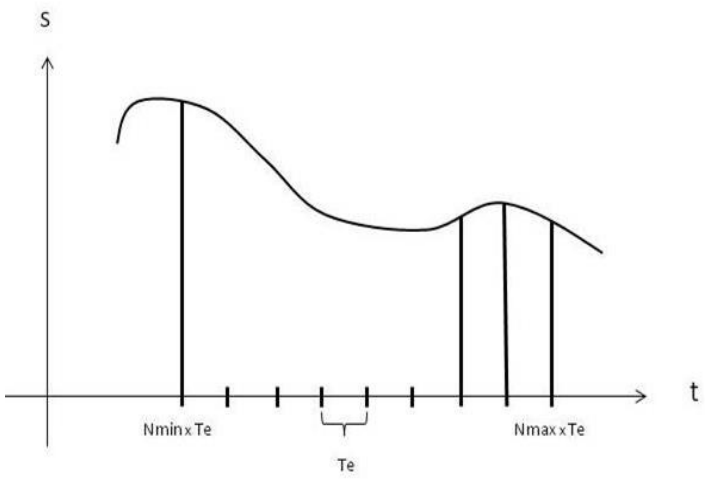


Figure 1 : Graphique pour l'échantillonnage temporel (amplitude s du signal en fonction du temps)

Il est très intéressant de travailler sur des données sonores, parce que Scilab permet de les traiter commodément. Pour s'en tenir à un standard, le son peut être échantillonné à 22050 Hz[4]. On peut proposer comme **activités :**

1. **créer un son d'une demi-seconde simulant une note LA 440 (fréquence 440 Hz) de différentes « couleurs » (son plus ou moins pur), et en représenter graphiquement le premier centième de seconde (comme courbe ou comme série de points isolés) ;**
2. **charger un son à partir d'un fichier, et le tronquer ou en extraire une section d'un quart de seconde ;**
3. **charger un son et transformer sa « hauteur » en doublant la fréquence de base.**

Exercice 1. Le plus simple pour créer le vecteur des instants d'échantillonnage est d'employer la fonction `soundsec` :

```
t=soundsec(0.5,22050);
```

Il revient exactement au même de définir la fréquence et le pas d'échantillonnage puis le vecteur :

```
Fe=22050;Te=1/Fe;t1=0:Te:0.5-Te ;
```

Nous pouvons maintenant créer le signal sinusoïdal (son pur) de fréquence 440 Hz :

```
s=sin(440*2*%pi*t);
```

Un centième de seconde, à 22050 Hz, représente 220 points d'échantillonnage :

```
plot(s(1:220))
```

On obtient la « courbe » ci-dessous (ce n'est pas - en toute rigueur - la « courbe » de la fonction sinus d'origine, mais une ligne brisée passant par les points issus de l'échantillonnage).

Pour « voir » les points d'échantillonnage on peut imposer un affichage de « marqueurs », par exemple :

```
plot(s(1:220),'r-d')
```

[4]Un son de bonne qualité (CD) devait plutôt être échantillonné à 44100Hz. Un son de qualité téléphonique peut être échantillonné à 7000Hz.

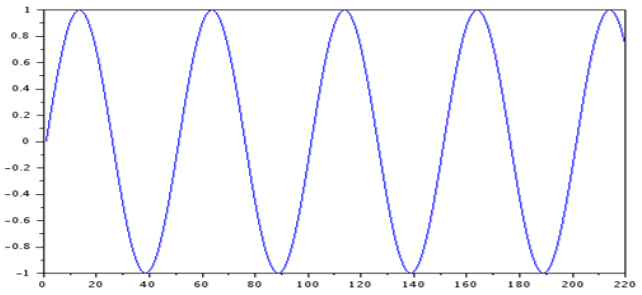


Figure 2 : Un centième de seconde d'un son sinusoïdal à 440 Hz

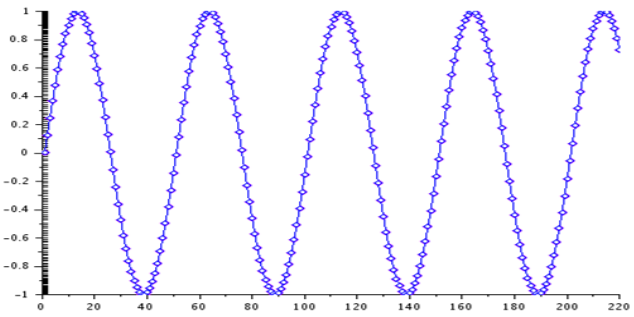


Figure 3 : Encore un centième de seconde (avec l'échantillonnage)

On verra dans l'exercice 2 (à l'écoute) que ce son n'est pas très mélodieux : c'est un son de diapason, dont le timbre est pauvre. Les musiciens préfèrent souvent des sons basés sur une fonction en « dents de scie », qu'il est facile de créer à partir du sinus :

```
s1=asin(sin(440*2*%pi*t));
```

(la fonction arcsinus, employée ici, tend à « redresser » le sinus puisqu'elle est réciproque de celle-ci entre $-\frac{\pi}{2}$ et $\frac{\pi}{2}$).

On peut même envisager des sons basés sur une fonction créneau, dont un rapport cyclique modulera la largeur des impulsions.

```
alpha = 0.35;
s2= sign(s+alpha);
```

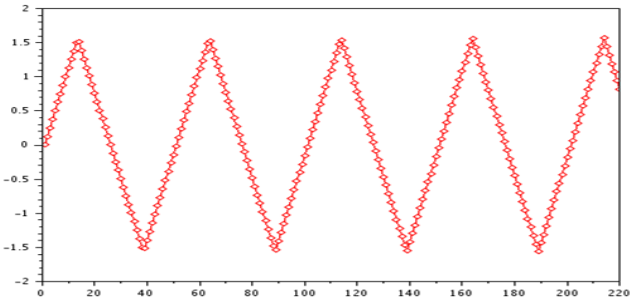


Figure 4 : Un signal en « dents de scie »

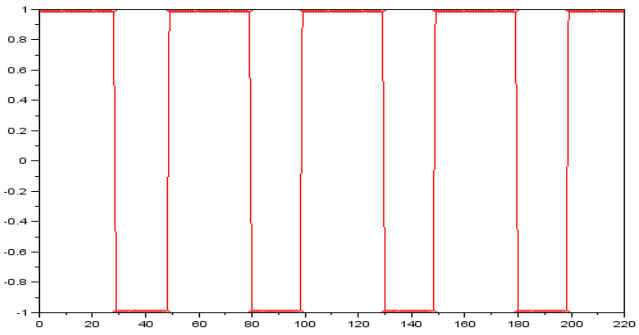


Figure 5 : Un signal « rectangulaire »

Exercice 2. L'aide de Scilab indique comment procéder :

```
y=loadwave("SCI/modules/sound/demos/chimes.wav");
```

La variable y qui en résulte est une matrice à deux lignes (une pour le canal gauche, l'autre pour le droit) et autant de colonnes que d'échantillons. En supposant que le son est enregistré à la fréquence de 22050 Hz, un quart de seconde contient 5512 échantillons ; on peut donc extraire le son du premier canal ainsi :

```
s=y(1,1:5512);
```

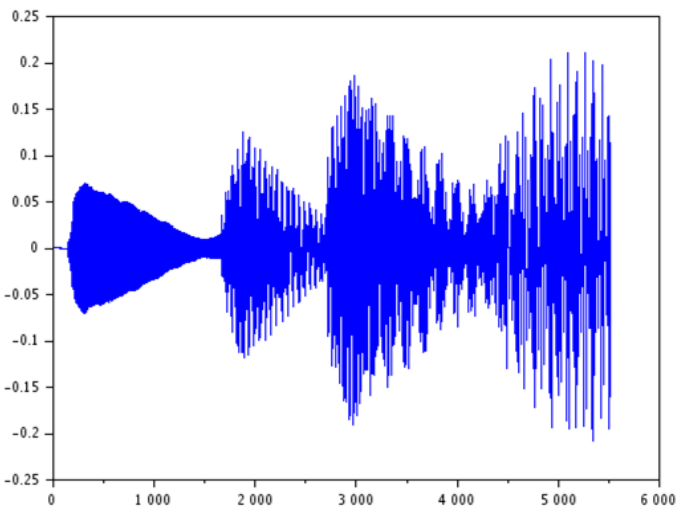


Figure 6 : un signal musical tronqué

Exercice 3. On peut par exemple poursuivre à partir du fichier sonore obtenu dans l'exercice 2. Une manière très rudimentaire – mais inexacte - de « doubler » la hauteur du son consiste à éliminer un échantillon sur deux, en compressant le tout (la sinusoïde se resserre) : la durée est donc moitié moindre[5] :

```
z(i)=y(1,2,$);
playsnd(z)
```

Une autre manière, sans perte d'information, consiste à doubler la fréquence d'échantillonnage de restitution du son (par défaut, 22050Hz) :

```
playsnd(z,44100)
```

3. Transformée de Fourier discrète

Les notations utilisées :

Le signal continu initial présente un spectre de fréquences positives occupant une bande limitée (dite monolatérale) comprise entre f_{min} et f_{max}. Le théorème de Nyquist-Shannon impose une fréquence d'échantillonnage F_e (nombre d'échantillons par seconde) au moins égale à $2.f_{max}$.

La période d'échantillonnage est notée T_e (et on a : $F_e = \frac{1}{T_e}$). On choisit le même nombre d'échantillons N pour le signal et pour son spectre[6]. L'échantillonnage de la transformée de Fourier discrète ramène celle-ci à N fréquences distinctes : $0 = 0 \times \frac{F_e}{N}$, $\frac{F_e}{N}$, … , $(N-1) \times \frac{F_e}{N}$. Noter que la résolution spectrale est l'inverse de la durée d'observation : $\frac{1}{NT_e} = \frac{F_e}{N}$.

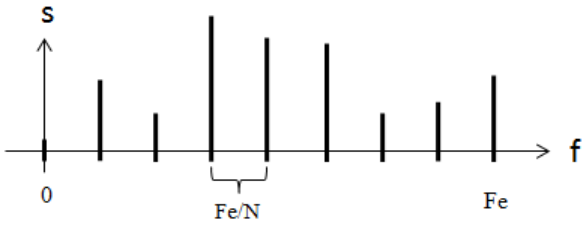


Figure 7 : échantillonnage fréquentiel (amplitude S des fréquences du signal en fonction de la fréquence f)

[5] L'opération visant à « diviser par deux » la fréquence est aussi envisageable, en dupliquant tous les échantillons (au prix d'une piètre qualité).

[6] Sauf à faire du « zéro padding », c'est-à-dire à border la séquence temporelle d'échantillons de valeurs nulles.

Quelques commandes Scilab utiles :

`fft( )` : transformée de Fourier rapide (fast Fourier transform)[7], à appliquer à un vecteur ; retourne un vecteur complexe (utiliser la commande `abs( )` pour en avoir le module) de même taille.

`ifft( )` : transformée de Fourier inverse

Exemple : reprenons le son sinusoïdal à 440 Hz, employé ci-dessus

```
Fe=22050; Te=1/Fe;
t=0:Te:0.5-Te // vecteur des instants
s=sin(440*2*%pi*t); // vecteur
échantillon
N=size(t,'c'); //vérifier la taille
du vecteur

// échantillonnage de la TFD
fr=(0:1:(N-1))*Fe/N;
// calcul de la TFD
y=abs(fft(s));
// représentation graphique
plot(fr,y);
```

Le graphique fait apparaître deux « barres » et (apparemment) rien de plus[8] ; les deux « pics » correspondent aux deux exponentielles qui composent le sinus. La « raie » à 440 Hz est bien visible. Celle à 21610 Hz est en fait l'image périodisée de 22050 Hz (fréquence d'échantillonnage) de celle à -440 Hz (non visible sur l'écran).

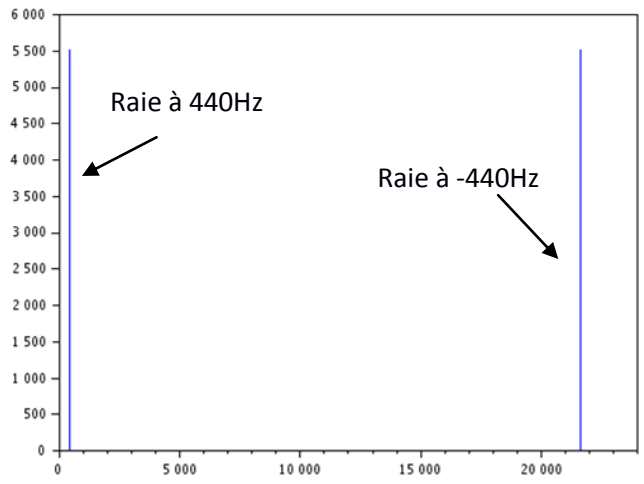


Figure 8 : Une transformée de Fourier discrète très simple

Activités.

1. **On propose de créer (ou plutôt simuler) un signal de 0,6s, échantillonné à 1000 Hz et contenant deux fréquences pures à 40 Hz et 90 Hz d'amplitude 1, déphasées, puis un signal « bruité » déduit du précédent en ajoutant un bruit aléatoire gaussien centré de variance 1, et de comparer les transformées de Fourier discrètes des deux signaux.**
2. **Analyser l'influence de la durée d'observation.**
3. **Proposer une méthode simple de débruitage fondée sur la TFD.**

Nombres aléatoires dans Scilab : la commande `grand(1, N, "nor",0,1)` renvoie un vecteur de N valeurs pseudo-aléatoires suivant une loi normale de moyenne nulle et d'écart-type 1.

[7] La FFT implémente un algorithme de calcul rapide de la TFD, optimial quand N est une puissance de 2.

[8] En fait, la transformée de Fourier discrète y n'est pas tout à fait nulle en-dehors des deux « pics » parce que la fréquence propre du signal (440 Hz) n'est pas un diviseur exact de la fréquence d'échantillonnage (22050 Hz). Ce point sera discuté plus loin.

Exercice 1. On commence par le signal « déterministe ».

```
//Composantes fréquentielles d'un signal
Fe=1000.0; Te=1/Fe;
//observer 0,6 s, échantillonner à Fe
t=0.0:Te:0.6-Te;
N=size(t,'c');  //nombre d'échantillons
s=sin(2*%pi*40*t)+sin(2*%pi*90*t+%pi/4);
//création d'un signal échantillonné.
y=abs(fft(s));
//graphique avec les pics de fréquence
plot(y)
//compte tenu de la symétrie de y, on ne
garde que N/2 points,
// et on légende l'axe des fréquences avec
les bonnes graduations
N1=floor(N/2);
fr=(0:1:N1-1)*Fe/N;
plot(fr(1:N1),y(1:N1));
```

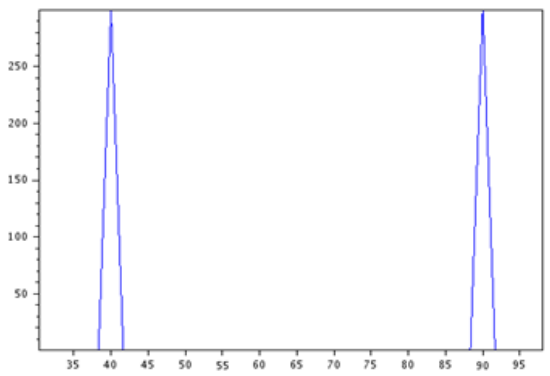
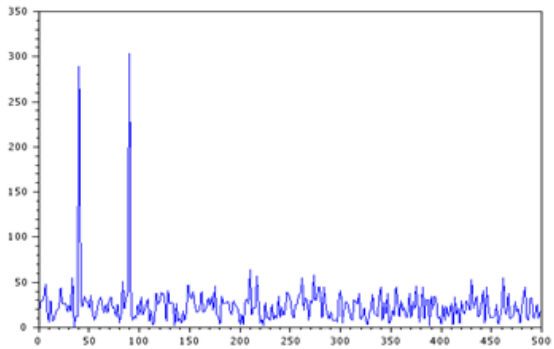


Figure 9 : TFD d'un signal composite à 40Hz et 90Hz

Le signal bruité s'obtient par superposition (ou addition).

```
R=grand(1,N,"nor",0,1);
s_bruit=s+R;
y_bruit=abs(fft(s_bruit));
plot(fr(1:N1),y_bruit(1:N1))
```

On voit nettement se dégager les « raies » à 40 Hz et 90 Hz.



Exercice 2. De manière générale, plus la durée d'observation du signal est grande, plus la résolution fréquentielle est fine. Un cas d'école est celui du signal périodique : la TFD renvoie le spectre exact aux fréquences d'échantillonnage (et, donc, les coefficients de Fourier) si le signal est observé sur un multiple entier de sa période, et si la fenêtre d'observation du signal est elle-même un multiple entier du pas d'échantillonnage[9]; sinon le spectre n'est qu'approché. Il gagne néanmoins en netteté à mesure que la fenêtre d'étude s'allonge.

Ici, 0,6s est multiple de la période du signal composite (non bruité) puisque 40x0,6 et 90x0,6 sont des entiers. La durée d'observation est un multiple entier du pas d'échantillonnage puisque 1000x0,6 est entier. La TFD a donc retourné 4 raies pures, correspondant aux fréquences 40 Hz, 90 Hz, 910 Hz[10], 960 Hz[11].

Observons maintenant le signal pendant 0,533s ; 1,11s ; 5,333s, et affichons la TFD correspondant à la plage de fréquences 0 – 150 Hz. Nous constatons les effets annoncés, sous la forme d'une « robe » élargissant les pics.

```
Fe=1000.0; Te=1/Fe;
t1=0.0:Te:0.533-Te; t2=0.0:Te:1.11-Te; t3=0.0:Te:5.333-Te;
s1=sin(2*%pi*40*t1)+sin(2*%pi*90*t1+%pi/4);
s2=sin(2*%pi*40*t2)+sin(2*%pi*90*t2+%pi/4);
s3=sin(2*%pi*40*t3)+sin(2*%pi*90*t3+%pi/4);
N1=size(t1,'c'); N2=size(t2,'c');N3=size(t3,'c');
fr1=(0:1:(150*N1/Fe))*Fe/N1; fr2=(0:1:(150*N2/Fe))*Fe/N2;
fr3=(0:1:(150*N3/Fe))*Fe/N3;
y1=abs(fft(s1)); y2=abs(fft(s2)); y3=abs(fft(s3));
plot(fr1,y1(1:1+(150*N1/Fe)));
```

[9] Cette deuxième condition est, en pratique (c'est-à-dire sur un montage matériel), toujours réalisée
[10] 910 = -90 [1000]
[11] 960 = -40 [1000]

```
plot(fr2,y2(1:1+(150*N2/Fe)));
plot(fr3,y3(1:1+(150*N3/Fe)));
```

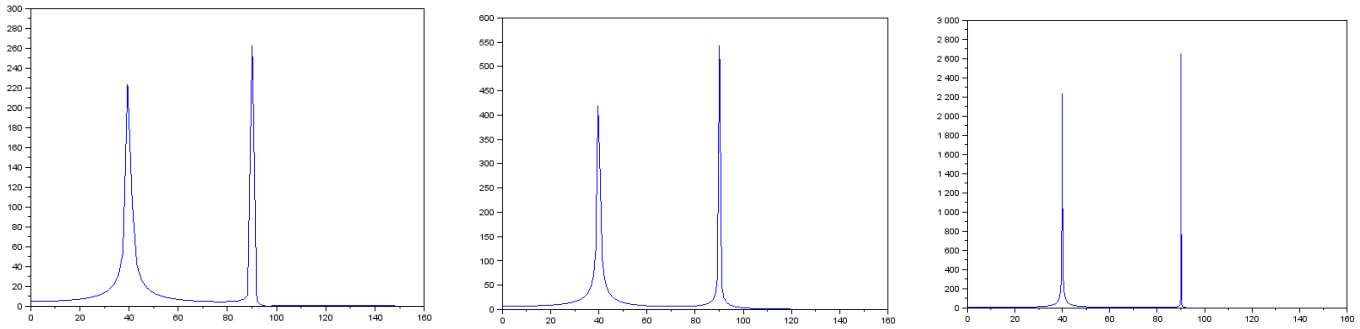


Figure 11 : Influence de la fenêtre d'étude sur la TFD

Exercice 3. Travaillons sur la séquence bruitée. Voici ce que l'on obtient en enlevant les fréquences de faible amplitude : au-dessous d'un certain seuil choisi, ces fréquences ont une amplitude ramenée à 0 (fenêtre temporelle de durée 0,6s).

```
y_debruite=fft(s_bruit).*(abs(fft(s_bruit))>55);
plot(fr(1:N1),abs(y_debruite(1:N1)));



s_debruite=ifft(y_debruite);
```

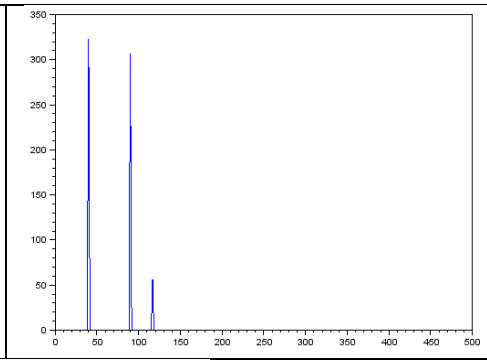


Figure 12 : Débruitage par un procédé rudimentaire

En appliquant la transformée de Fourier inverse, le signal est filtré. Il demeurera toujours une petite erreur sur `s_debruite`. La faute au bruit entachant les deux pics, qui ne peut être supprimé facilement : le bruit a lui aussi une composante spectrale aux fréquences 40 Hz et 90 Hz, qui s'est indissociablement mêlée à celle des sinusoïdes.

Cette méthode est rudimentaire, et peut être source d'erreurs : quand la durée d'observation n'est pas multiple de la période du signal, ou quand le signal est non périodique, toutes les composantes spectrales peuvent être porteuses d'information utile.

L'équipe qui a rédigé ce document ressource était composée de :

Cabane Robert, IGEN, groupe des mathématiques
Dubouloz Georges, IA-IPR de mathématiques, académie de Grenoble
Fleurant Sandrine, IA-IPR de mathématiques, académie de Nantes
Zayana Karim, IGEN, groupe des mathématiques