TRAVAUX PRATIQUES DE TRAITEMENT DU SIGNAL (2)

1 • PRÉALABLES.

Cette séance doit vous familiariser avec l'utilisation des processus linéaires pour traiter des signaux aléatoires. Nous vous proposons de supprimer (ou en tout cas fortement atténuer) un bruit de type craquement dans un fichier son, Votre travail est relié à votre cours de traitement du signal. Il vous est demandé de le préparer sérieusement et de réaliser un travail expérimental de qualité. Vous disposez, pour cette séance, d'un fichier contenant un enregistrement sonore dont vous devez supprimer les craquements.

2 • SUPPRESSION D'UN BRUIT DE TYPE CRAQUEMENT.

2.1 • Le fichier.

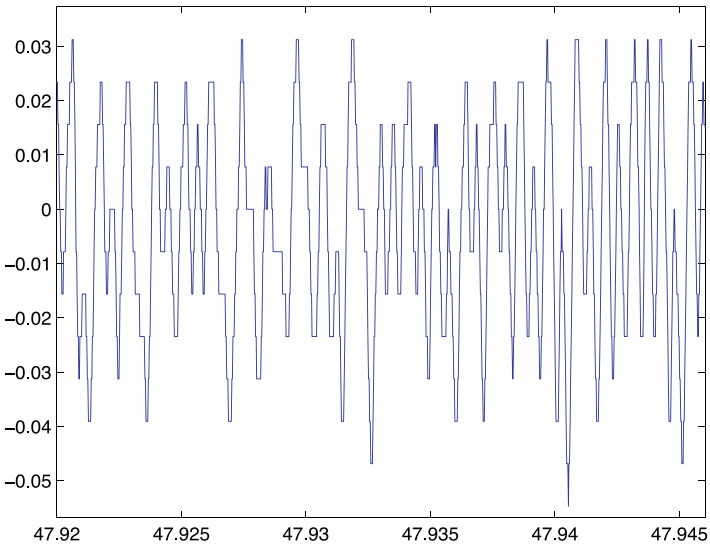
Vous disposez d'un fichier son sous le nom de `MusiqueBruit.wav` que vous chargerez sous l'environnement Matlab par la commande :

```
>> [signal_audio, Fs] = audioread('MusiqueBruit.wav') ;
```

Visualisez ce signal audio :

```
>> delta_t = 1/Fs ; taille_signal = length(signal_audio) ;
>> temps = delta_t * (0:(taille_signal-1)) ;
>> figure(22) ; plot(temps, signal_audio) ;
>> axis([0, max(temps), -1, 1 ]) ; title('signal audio original') ;
```

Vous voyez apparaître des pics disposés aléatoirement sur le son. C'est une affection classique qu'on appelle "bruit de craquement", qui apparaissent sur des sons numérisés issus de disques vinyls mais aussi sur des signaux sonores numériques après transmission à cause des erreurs de transmission. En zoomant sur le signal, vous pouvez aussi voir clairement la quantification du signal sonore, celui-ci évoluant par palliers.



Vous pouvez par ailleurs écouter le morceau de musique enregistré dans ce fichier son en cliquant dessus. Il s'agit d'un enregistrement altéré d'une oeuvre connue d'un compositeur du 20ème siècle. Vous pouvez entendre les craquements (qui ne sont pas prévus par le compositeur initialement).

2.2 • Principe de la méthode

Pour retirer ces craquements, du fichier son, on pourrait utiliser un filtre passe-bas (justifiez ce point de vue). Vous pouvez d'ailleurs essayer cette solution en utilisant un filtre de Butteworth par exemple (commande `butter` pour définir le filtre et `filter`

pour filtrer). Un exemple de code vous est donné ci-dessous.

```
>> [B,A] = butter(7,0.02) ;
>> signal_audio_filtre = filter(B,A,signal_audio) ;
>> audiowrite('MusiqueFiltre.wav', signal_audio_filtre, Fs) ;
>> figure(55) ; plot(temps,signal_audio_filtre) ;
>> axis([0, max(temps), -1, 1 ]) ; title('signal audio filtre') ;
```

Que constatez vous ? Quelle est la fréquence de coupure du filtre ? Qu'est-ce qui pourrait vous guider pour changer cette fréquence de coupure ? Que savez-vous des filtres numériques ?

La méthode que nous vous proposons d'utiliser doit mettre en évidence les craquements en vue de leur détéction. Cette mise en évidence consiste à filtrer le signal grâce à un filtre prédictif adapté au signal puis de regarder l'écart entre le signal prédit et le signal courant. Une telle opération de comparaison peut être bien sûr réalisée en une fois. En effet, en fréquentiel, si $F(\omega)$ est le filtre passe-bas à utiliser, $F(\omega)E(\omega)$ est la transformée de Fourier du signal filtré et $F(\omega)E(\omega) - E(\omega) = (F(\omega) - 1)E(\omega)$ la transformée de Fourier de la comparaison du signal filtré et du signal original. On pose $H(\omega) = (F(\omega) - 1)$ le filtre que l'on doit utiliser pour mettre en évidence les craquements. Pour que la localisation des craquements soit la plus fiable possible, ce filtre doit être symétrique (expliquez pourquoi).
Une façon de programmer un filtre symétrique est de mettre en cascade deux filtres, l'un causal $h(t)$ et l'autre anti-causal $g(t)$. Dans ce TP, nous allons supposer que ces deux filtres sont de type moyenne adaptative (MA).
Une étude théorique du problème a montré que le filtre causal de réponse impulsionnelle $h(t)$ qui sera le mieux adapté à notre problème (et qui mettra donc en évidence les craquements) est l'inverse du filtre adapté au signal à filtrer (de réponse impulsionnelle $d(t)$). $g(t)$ n'est autre que la réponse impulsionnelle de sa version anti-causale, c'est à dire que $h(t) = g(-t)$.
Donc, si $D(\omega)$ est la fonction de transfert du filtre adapté au signal dont on veut détécter des anomalies, on a la relation : $H(\omega) = \frac{1}{D(\omega)}$.
Nous nous occupons de signaux discrets, le filtre MA en question est donc modélisable par sa fonction de transfert échantillonnée : $H(z) = 1 + h_1 z^{-1} + \ldots + h_p z^{-p}$, dont l'algorithme associé est : $s_k = (e_k + h_1 e_{k-1} + \ldots + h_p e_{k-p})$, s_k étant la sortie du filtre et e_k l'entrée. Le filtre anti-causal s'écrit : $G(z) = 1 + h_1 z^1 + \ldots + h_p z^p$ puisque $g_{-k} = h_k$. Dans ce cas la sortie du filtre anti-causal s'écrit :
$s_k = (e_k + h_1 e_{k+1} + \ldots + h_p e_{k+p})$ (s_k dépend donc uniquement des échantillons futurs).
Expliquez la relation entre $H(z)$, $D(z)$ et $G(z)$.

Vous devez donc :

- identifier le filtre adapté au signal à filtrer, en déduire les coefficients $(h_1, \ldots, h_p)$,
- filtrer le signal avec le filtre causal, puis avec le filtre anti-causal
- comparer la valeur absolue de ce signal filtré à un seuil prédéfini.

Le seuil en question doit bien sûr dépendre de la puissance du signal : $P = \frac{1}{N}\sum_k s_k^2$.

3 • MANIPULATIONS

3.1 • Identification du filtre adapté

Si vous regardez un fichier son, sans l'écouter, vous pourriez penser que celui-ci n'est autre qu'un fichier de bruit. C'est ce que nous allons supposer ici : nous allons supposer que le morceaux de musique (signal) et le bruit de craquements sont deux signaux aléatoires ayant des propriétés statistiques différentes. Le signal étant majoritaire, c'est lui qui sera identifié.

On va supposer que le signal est issus du filtrage d'un bruit blanc de puissance unitaire par un filtre AR (voir cours). Dans ce cas, on retrouve la réponse impulsionnelle du filtre grâce à la fonction d'auto-covariance R_{xx} (en appelant x votre signal). On ne dispose bien sûr pas de cette fonction d'auto-covariance, on l'estime donc (abusivement ici) par la fonction d'auto-corrélation C_{xx}. Nous supposerons que le filtre AR à identifier est de rang p=20 (vous pourrez changer cette valeur par la suite pour voir comment le filtre se comporte). Nous allons essayer de trouver le filtre AR qui, si on lui mets en entrée un bruit blanc, le transforme en un signal qui aurait les mêmes propriétés statistiques (supposées) que le morceau de musique, c'est à dire la même fonction de corrélation (ou de covariance).

Pour identifier le filtre AR, vous devez calculer la fonction d'auto-corrélation du signal audio (`signal_audio`) sur un intervalles de valeurs [-M, M] tel que M≥20 (voir TP précédent). Prenez M=100 pour visualiser cette auto-corrélation. Quelle reflexion vous induit cette visualisation ? Utilisez votre remarque pour choisir p convenablement.

Pour identifier le filtre, il vous faut demander à Matlab de résoudre l'équation :

$$\begin{bmatrix} C_{xx}(0) & \ldots & C_{xx}(p) \\ C_{xx}(-1) & \ldots & C_{xx}(p-1) \\ \ldots & \ldots & \ldots \\ C_{xx}(-p) & \ldots & C_{xx}(0) \end{bmatrix} \begin{bmatrix} \varphi_0 \\ \varphi_1 \\ \ldots \\ \varphi_p \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ \ldots \\ 0 \end{bmatrix}$$

. Pour inverser la matrice $\begin{bmatrix} C_{xx}(0) & \ldots & C_{xx}(p) \\ \ldots & \ldots & \ldots \\ C_{xx}(-p) & \ldots & C_{xx}(0) \end{bmatrix}$, préférez la commande `pinv` (calcul de pseudo inverse) à la command `inv` car il se peut que la matrice soit singulière (pourquoi ?). On retrouve les coefficients h_k recherchés en posant : $h_k = \varphi_k / \varphi_0$ pour $k = 0, \ldots, p$. Visualisez la réponse impulsionnelle obtenue.

3.2 • Filtrage du signal audio

Filtrez le signal `signal_audio` avec le filtre causal, puis filtrez le résultat avec le filtre anticausal. Faites très attention aux bornes d'intégration. Pensez à utiliser des variables :

```
>> taille_signal = length(signal_audio) ;
>> taille_reponse = length(reponse_impuls) ;
>> % Filtre causal
>> entree = signal_audio ; sortie = entree ;
>> for k= (taille_reponse+1):(taille_signal-taille_reponse)
>>  sortie(k) = % ici votre algorithme causal
>> end
>> signal_audio_filtre = sortie ;
>> % Filtre anti-causal
>> entree = signal_audio_filtre ; sortie = entree ;
>> for k= (taille_reponse+1):(taille_signal-taille_reponse)
>>  sortie(k) = % ici votre algorithme anti-causal
>> end
>> signal_audio_filtre = sortie ;
```

Puis prenez-en la valeur absolue et divisez-la par la puissance du signal entrant :

```
>> Puissance = sum(signal_audio.^2) / taille_signal ;
>> signal_audio_filtre = abs(signal_audio_filtre) / Puissance ;
```

Visualisez ce signal filtré. Les points de craquements doivent être très visibles. Choisissez un seuil qui vous semble correct (entre 5 et 10 habituellement) et créez un le vecteur des points où le signal est bruité :

```
>> seuil = 8 ;
>> indice_craquement = ( signal_audio_filtre > seuil ) ;
```

Cet `indice_craquement` vaudra 1 à tous les points identifiés comme un craquement. Puis visualisez si cette identification est correcte :

```
>> figure(22) ; clf ; hold on ; plot(temps, signal_audio) ;
>> for k=1:taille_signal
>>  if(indice_craquement(k)) plot(temps(k), signal_audio(k),'ro') ; end
>> end
```

Changez le seuil pour en voir l'influence.

3.3 • Filtrage du signal

Vous allez maintenant retirer le bruit impulsionnel de votre signal. Un des meilleurs filtres pour le bruit impulsionnel est le filtre médian. Le filtre médian remplace le $k^{ième}$ échantillon du signal par la médiane d'un voisinage de cet échantillon. Appliqué à tout le signal, ce filtre aurait le même impact sur le signal qu'un passe-bas. Appliqué aux seuls échantillons bruités, il améliore considérablement le rapport signal-sur-bruit.

Vous pouvez essayez plusieurs voisinages. Voici un exemple :

```
>> horizon = 5 ;
>>  for (k=(horizon+1):(taille_signal-horizon))
>>   if indice_craquement(k)
>>    Echantillon = signal_audio_bruite((k-horizon):k-1) ;
>>    Echantillon = [Echantillon ; signal_audio_bruite(k+1:(k+horizon))] ;
>>    signal_reconstruit(k) = median(Echantillon) ;
>>   end
>>  end
```

Sauvegardez votre signal reconstruit :

```
>> audiowrite('MusiqueSansBruit.wav', signal_reconstruit, Fs) ;
```

Il ne vous reste plus qu'à écouter le son obtenu (utilisez Audacity par exemple) et essayer cet algorithme pour différentes valeurs de p, d'horizon du filtre médian, de seuil, etc. Vous pouvez aussi essayer d'autre types de filtrage (moyenne, moyenne pondérée, ...), ainsi bien-sûr que d'autres passe-bas.