Traitement numérique du signal

Travaux Pratiques (7h)

Synthèse, analyse et filtrage d'un signal audio numérique

Université Paris 13, Institut Galilée, Ecole d'ingénieurs Sup Galilée
Parcours Informatique et Réseaux Alternance - 1ère année

2017-2018

Consignes

- Récupérer le fichier `TP.zip` sur le site

  `http://www.laurentoudre.fr/tns.html`

- Ouvrir MATLAB et créer un répertoire de travail. Dézipper le fichier `TP.zip` dans ce répertoire.
- A la fin de la séance, récupérer les scripts que vous avez écrits et les envoyer par e-mail au chargé de TP ainsi qu'à vous même afin de les conserver pour la prochaine séance.

Rendu

- Sept scripts : `TP_Part1.m`, `TP_Part2.m`, `TP_Part3.m`, `TP_Part4.m`, `TP_Part5.m`, `TP_Part6.m` et `TP_Part7.m` et deux fonctions : `create_note.m` et `create_melody.m` Chaque fichier doit contenir votre nom, votre prénom et la date.
- Compte-rendu à rendre avant le 01/03/2018, contenant les observations, commentaires et réponses aux questions. Le compte rendu doit contenir votre nom et votre prénom.

Plan de l'étude

1 Introduction

1.1 But de l'étude

Le but de ce TP est de synthétiser sous MATLAB des signaux audio numériques, d'analyser certaines de leurs propriétés temporelles et fréquentielles, et de réaliser un filtrage élémentaire pour effectuer une séparation de sources. Ces signaux seront définis de façon analytique sous la forme d'une concaténation de sinusoïdes pures échantillonnées et associées chacune à une fréquence fondamentale f_0. Ce modèle, bien que grossier, permet de générer des signaux musicaux idéaux, qui peuvent être étudiés et filtrés de façon simple par les techniques usuelles de traitement numérique du signal.

1.2 Modèle

On considère le signal continu suivant :

$$x(t) = \sin(2\pi f_0 t) \tag{1}$$

Il s'agit d'une sinusoïde pure de fréquence fondamentale f_0, d'amplitude 1 et de phase égale à 0. Le signal obtenu correspond à une note de musique. La note jouée est liée à fréquence fondamentale de la sinusoïde : plus elle est élevée, plus le son est aigu. Par exemple, une sinusoïde de fréquence fondamentale $f_0 = 440$ Hz correspond à la note *la* : il s'agit de la note que l'on entend en utilisant un diapason ou en décrochant son téléphone fixe.

Ce signal analogique n'est pas étudiable avec MATLAB (il contient une infinité d'échantillons et ses valeurs ne sont pas quantifiées), nous allons donc travailler sur une version échantillonnée et quantifiée de ce signal. On définit donc le signal numérique x_n échantillonné avec une fréquence d'échantillonnage F_s et contenant N échantillons :

$$x_n = \sin(2\pi f_0 t_n) \tag{2}$$

Les temps t_n sont définis de la façon suivante :

$$t_n = \frac{n}{F_s} \text{ pour } 0 \leq n \leq N-1 \tag{3}$$

Par défaut, MATLAB quantifie toutes les valeurs sur 64 bits et sauf mention contraire, on travaillera dans cette partie avec un signal échantillonné à $F_s = 8000$ Hz, de fréquence fondamentale $f_0 = 800$ Hz et d'une durée $d = 2$ secondes.

1.3 Simulations sous MATLAB

On veut synthétiser sous MATLAB un premier signal numérique selon le modèle décrit ci-dessus

1. Se placer dans le répertoire de travail et créer sous MATLAB un script vide nommé `TP_part1.m`

On écrira en haut de chaque script le nom et le prénom de l'auteur, la date, ainsi que les commandes suivantes qui permettent de nettoyer tout l'espace de travail à chaque fois que le script est lancé :

```
% NOM Prenom
% Date
clear all   % Supprime toutes les variables de l'espace de travail
close all   % Ferme toutes les figures courantes
clc         % Nettoie l'historique des commandes
```

2. Fixer les valeurs de F_s, d et f_0. Comment doit-on choisir N en fonction de d et F_s ? Que devient cette expression si d n'est plus un entier ? Calculer N en fonction de F_s et d.

Sous Matlab il existe trois façon d'arrondir un nombre :

```
y=round(x);     % Arrondit a l'entier le plus proche
y=floor(x);     % Arrondit a l'entier inferieur (partie entiere)
y=ceil(x);      % Arrondit a l'entier superieur
```

3. Construire sous MATLAB le vecteur temps `t` de taille N et contenant les différentes valeurs t_n. **Attention : dans tous ces Travaux Pratiques, on travaillera toujours avec des vecteurs colonnes !**

Pour créer un vecteur `y` ligne contenant toutes les valeurs entre `debut` et `fin` avec un pas de `pas`, on peut utiliser

```
y = debut:pas:fin;
% Exemples d'utilisation :
x1 = 0:0.1:1;   % Vecteur ligne 0, 0.1, 0.2, ...
x2 = 3:8;       % Vecteur ligne 3, 4, 5, ...
x3 = (0:5)'/2;  % Vecteur colonne 0 1/2 1, ...
```

Si la valeur de `pas` n'est pas spécifiée, MATLAB la fixe automatiquement à 1.

4. Construire le signal `x` contenant les différentes valeurs x_n

Etant donné un vecteur `y`, on peut construire un nouveau vecteur `z` de la façon suivante:

```
z = y+2;    % On rajoute 2 a toutes les valeurs du vecteur y
z = 5*y;    % On multiplie toutes les valeurs du vecteur y par 5
z = y.^3;   % On met toutes la valeurs du vecteur y a la puissance 3
z = log(y);     % On prend le logarithme de toutes les valeurs du vecteur y
z = abs(y);     % On prend la valeur absolue de toutes les valeurs du vecteur y
z = cos(y/7);   % On divise toutes les valeurs du vecteur y par 7, puis on en prend
                % le cosinus
```

5. Ecouter le son avec vos écouteurs : est-il grave ? aigu ? Essayer différentes valeurs pour f_0 (entre 300 et 1200 Hz) et commenter.

Sous MATLAB, un son est défini comme un vecteur ayant des valeurs comprises entre -1 et 1 et associé à une fréquence d'échantillonnage. Pour jouer un son `x` échantillonné à `Fs`, on utilise la commande:

```
sound(x,Fs)     % Joue le son brut OU
soundsc(x,Fs)   % Joue le son apres renormalisation
                % (renormalise toutes les valeurs du vecteur entre -1 et 1)
```

Il vaut mieux lancer cette instruction dans la *Command Window*, car sinon le son se jouera à chaque fois qu'on exécutera le script !

6. Quelle est la période fondamentale T_0 du signal $x(t)$ avant échantillonnage ? Afficher le signal échantillonné x_n en fonction du temps sur l'intervalle $[0\ ,\ 4T_0]$ et annoter correctement la figure. Essayer différentes valeurs pour f_0 (entre 300 et 1200 Hz) et commenter.

Voici quelques commandes utiles pour faire une figure sous MATLAB:

```
figure          % Ouvre une nouvelle figure
plot(t,x)       % Affiche le vecteur x en fonction du vecteur t
title('Ma Figure')  % Donne un titre a la figure
xlabel('Temps (en secondes)')    % Donne un nom a l'axe des abscisses
ylabel('Amplitude') % Donne un nom a l'axe des ordonnees
xlim([0 1])     % Restreint la figure pour les abscisses comprises entre 0 et 1
ylim([-1 1])    % Restreint la figure pour les ordonnees comprises entre -1 et 1
```

2 Etude dans le domaine temporel

2.1 Propriétés du signal

Nous allons maintenant étudier les propriétés de notre signal : périodicité, énergie, puissance moyenne.

1. Créer sous MATLAB un script vide nommé `TP_part2.m` dans lequel vous définissez les valeurs de F_s, d, f_0 et N. On utilisera $F_s = 8000$ Hz, $f_0 = 800$ Hz et $d = 2$ secondes.

2. Le signal x_n est-il périodique ? Si oui, quelle est sa période fondamentale M_0 ? Confirmer ceci sous MATLAB en calculant la distance euclidienne entre deux périodes du signal.

- Pour sélectionner uniquement certaines valeurs d'un vecteur `y`, on peut utiliser la commande :

```
z = y(1:4);     % z contient les 4 premieres valeurs de y
z = y(3:2:9);   % z contient les valeurs y(3), y(5), y(7), y(9)
```

- Pour calculer la distance euclidienne entre deux vecteurs `x` et `y` de même taille on peut calculer :

```
D = sqrt(sum((x-y).^2));     % sqrt : racine carree
```

3. Calculer (sur papier) l'énergie totale et la puissance moyenne totale du signal $x(t)$ avant échantillonnage.

4. En synthétisant un signal de $d = 1000$ secondes (qui se rapproche du cas $N \to +\infty$), évaluer sous MATLAB l'énergie totale E_x, ainsi que la puissance moyenne totale P_x du signal échantillonné x_n. Commenter. Utiliser les deux définitions de la puissance moyenne totale (celle donnée pour un signal quelconque et celle donnée pour un signal périodique) et vérifier que l'on obtient la même chose.

Pour obtenir un scalaire `z` à partir d'un vecteur `y`, on pourra utiliser :

```
z = sum(y);      % somme des elements du vecteur y
z = mean(y);     % moyenne des elements du vecteur y
z = max(y);      % maximum des elements du vecteur y
```

2.2 Echantillonnage et critère de Nyquist

Nous allons maintenant nous intéresser à l'échantillonnage et illustrer le critère de Nyquist. On considère les paramètres suivants : $F_s = 8000$ Hz, $f_0 = 800$ Hz et $d = 2$ secondes.

1. D'après Nyquist, quelle est la valeur minimale de la fréquence d'échantillonnage que nous pouvons utiliser sans dégrader de façon irréversible le signal $x(t)$?

2. Illustrons ceci de façon expérimentale : choisir différentes valeurs pour F_s, certaines au dessus de cette valeur minimale, d'autres en dessous et écouter les signaux obtenus. Conclure.

3. On suppose à nouveau que la fréquence F_s est égale à 8000 Hz. Montrer par le calcul que les sinusoïdes de fréquences avec $f_0 = 800$ Hz et $f_0 = 7200$ Hz ont la même fréquence apparente. Vérifier ceci en écoutant les signaux obtenus avec ces deux fréquences fondamentales.

3 Synthèse d'une note et d'une mélodie

Nous allons dans cette partie étudier non plus une seule sinusoïde mais plusieurs, qui, mises bout à bout, vont créer une mélodie.

3.1 Synthèse d'une note

Pour le moment nous avons considéré une sinusoïde de fréquence fondamentale f_0 quelconque, sans savoir si cette fréquence correspondaient à une note produite effectivement par un instrument de musique. En réalité, si l'on considère par exemple un piano, il y a seulement un ensemble fini de notes (donc de fréquences) qu'il peut produire. Dans notre cas, nous allons donner à chaque note du piano un numéro, et calculer la fréquence fondamentale associée grâce à la formule suivante:

$$f_0^{note} = 440 \times 2^{\frac{note-69}{12}} \tag{4}$$

où $note$ est le numéro de la note que l'on veut jouer. On peut observer que la note $note = 69$ correspond justement à la fréquence $f_0 = 440$ Hz (c'est la note *la*) dont on parlait dans l'introduction. Voici un tableau qui présente un extrait de la correspondance entre numéros de notes, notes musicales et fréquences fondamentales.

octave	do	do ♯	ré	ré ♯	mi	fa	fa ♯	sol	sol ♯	la	la ♯	si
1	24 32.7	25 34.65	26 36.71	27 38.89	28 41.2	29 43.65	30 46.25	31 49	32 51.91	33 55	34 58.27	35 61.74
2	36 65.41	37 69.3	38 73.42	39 77.78	40 82.41	41 87.31	42 92.5	43 98	44 103.83	45 110	46 116.54	47 123.47
3	48 130.81	49 138.59	50 146.83	51 155.56	52 164.81	53 174.61	54 185	55 196	56 207.65	57 220	58 233.08	59 246.94
4	60 261.63	61 277.18	62 293.66	63 311.13	64 329.63	65 349.23	66 369.99	67 392	68 415.3	69 440	70 466.16	71 493.88
5	72 523.25	73 554.37	74 587.33	75 622.25	76 659.26	77 698.46	78 739.99	79 783.99	80 830.61	81 880	82 932.33	83 987.77
6	84 1046.5	85 1108.73	86 1174.66	87 1244.51	88 1318.51	89 1396.91	90 1479.98	91 1567.98	92 1661.22	93 1760	94 1864.66	95 1975.53
7	96 2093	97 2217.46	98 2349.32	99 2489.02	100 2637.02	101 2793.83	102 2959.96	103 3135.96	104 3322.44	105 3520	106 3729.31	107 3951.07

1. Créer sous MATLAB un script vide nommé `TP_part3.m` dans lequel vous définissez les valeurs de F_s, d et $note$. Vous choisirez les valeurs de d et $note$, et on utilisera $F_s = 8000$ Hz.

2. Créer une fonction `create_note.m` prenant en paramètres d'entrée une durée `d`, un numéro de note `note` et une fréquence d'échantillonnage `Fs` et renvoyant un signal `x` (correspondant à une sinusoïde de durée d et de fréquence fondamentale f_0^{note} comme défini ci-dessus) associé à un vecteur temps `t`. Dans le cas où `note = -1`, on renverra un signal nul de durée `d` (cela revient à créer une sinusoïde de fréquence $f_0 = 0$!)

   - Pour créer une fonction sous MATLAB, il faut créer un fichier .m dont le nom est exactement le nom de la fonction qu'on veut définir. L'entête du fichier doit être le suivant :

   ```
   function [out1,out2] = nomFonction(in1,in2,in3)
   % nomFonction : nom de la fonction : le script doit s'appeler nomFonction.m
   % in1, in2, in3 : arguments d'entree de la fonction
   % out1, out2 : arguments de sortie de la fonction
   ```

   - Pour tester une fonction, on ne peut pas l'exécuter telle quelle : il faut créer un script où l'on définit les entrées, où l'on appelle la fonction et où l'on récupère les sorties de la fonction.
   - Pour tester une condition sous MATLAB, il faut utiliser une structure du type :

   ```
   if i==0     % Si i est egal a 0
         i=i+1;
   else        % Sinon
         i=i-1;
   end
   ```

3. Tester la fonction au sein du script `TP_part3.m` en synthétisant et en écoutant plusieurs notes de différentes hauteurs et différentes durées. Tester le cas `note = -1`.

3.2 Synthèse d'une mélodie

Une mélodie ce n'est pas seulement une note mais une succession de notes ! Nous allons donc utiliser non plus une seule durée et une seule note, mais des vecteurs contenant plusieurs durées et plusieurs notes, afin de former une mélodie.

1. En s'inspirant de la fonction `create_note.m`, créer une nouvelle fonction `create_melody.m` prenant en paramètres d'entrée un vecteur de duree `d_vect`, un vecteur de numéros de note `note_vect` et une fréquence d'échantillonnage `Fs` et renvoyant un signal `x` associé à un vecteur temps `t`. Le signal `x` correspond à la concaténation de plusieurs sinusoïdes dont les durées et hauteurs sont respectivement définies dans les vecteurs `d_vect` et `note_vect`

   - Quelques opérations sur les vecteurs en MATLAB :

     ```
     y = [];      % Cree un vecteur vide
     y = [y_1 y_2];  % Concatenation des vecteurs y_1 et y_2 (vecteurs lignes)
     y = [y_1 ; y_2];% Concatenation des vecteurs y_1 et y_2 (vecteurs colonnes)
     N=length(y);    % Taille du vecteur y
     y=zeros(N,1);   % Cree un vecteur colonne de taille N contenant des 0
     y=zeros(1,N);   % Cree un vecteur ligne de taille N contenant des 0
     ```

   - La syntaxe d'une boucle `for` sous MATLAB est la suivante :

     ```
     j=0;
     for i=1:3   % Tous les i de 1 a 3
         j=j+i;
     end
     ```

   - On commencera par générer le vecteur `x`, puis on génèrera le vecteur temps `t`

2. Tester la fonction au sein du script `TP_part3.m` en choisissant des vecteurs `d_vect` et `note_vect`, puis en synthétisant et en écoutant la mélodie associée.

3. Tester la fonction en chargeant les vecteurs `d_vect` et `note_vect` stockés dans la variable `melody1.mat`. Synthétiser et écouter la mélodie associée.

   Pour charger toutes les variables enregistrées dans un fichier .mat, il faut utiliser la commande :

   ```
   load('data.mat');
   ```

4. Comment modifier cette mélodie pour qu'elle soit jouée une octave en dessous ? Et pour qu'elle aille 50% plus vite ?

4 Etude dans le domaine fréquentiel

Les mélodies que nous avons générées sont difficilement étudiables dans le domaine temporel : en effet, nous avons vu qu'il était par exemple impossible de visualiser le signal de façon correcte sauf en le regardant sur un temps très court. Nous allons donc continuer notre étude dans le domaine spectral, en observant la transformée de Fourier discrète de notre signal. Nous allons voir que cette représentation nous permet d'identifier par exemple en un seul coup d'oeil l'ensemble des notes jouées durant le morceau.

4.1 Cas d'une note

1. Créer sous MATLAB un script vide nommé `TP_part4.m` dans lequel vous définissez les valeurs de F_s, d et $note$. On utilisera $F_s = 8000$ Hz, $note = 69$ et $d = 2$ secondes.

2. Synthétiser le signal sonore associé grâce à la fonction `create_note.m` et tracer le module au carré de la transformée de Fourier discrète du signal grâce à la fonction `my_FFT.m` fournie sur le site du cours. Annoter correctement la figure !

> La fonction fournie `[Y,f]=my_FFT(y,Fs)` prend en entrée un signal `y` échantillonné à `Fs` Hz, et renvoie la transformée de Fourier discrète `Y` associée à un vecteur de fréquences `f`. Le spectre est représenté ici sous sa forme réarrangé, c'est à dire qu'il est centré sur la fréquence nulle.

3. Calculer (sur papier) le module au carré de la transformée de Fourier continue du signal $x(t)$. Comparer avec la figure obtenue et commenter.

4. Quelle est la résolution en fréquence de la TFD ? Parmi les notes proposées dans le tableau présenté précédemment, lesquelles seront observables de façon correcte sur la TFD ?

5. Générer le signal associé à une note observable et un signal associé à une note non observable. Comparer les spectres des deux signaux et commenter.

6. Montrer qu'il est néanmoins possible de retrouver quelle note a été jouée en regardant uniquement le module au carré de la transformée de Fourier.

4.2 Cas d'une mélodie

1. Créer à la suite du script `TP_part4.m` un signal sonore correspondant à une mélodie contenant 4 notes (dont vous choisirez les hauteurs et les durées) et échantillonné à $F_s = 8000$ Hz.

2. Tracer le module au carré de la transformée de Fourier discrète du signal et montrer qu'on peut retrouver quelles notes ont été jouées en regardant uniquement la transformée de Fourier.

5 Filtrage et séparation de sources

Un des grands domaines du traitement du signal est la séparation de sources. L'idée étant, à partir d'un mélange de différents signaux provenant de plusieurs sources (par exemple plusieurs personnes qui parlent en même temps), de parvenir à reconstruire le message associé à chacune des sources (par exemple retrouver ce qu'a dit chaque personne). C'est un problème courant en acoustique donc, mais aussi en télécommunications, en bio ingéniérie etc...

Dans notre cas, nous allons tenter d'additionner deux signaux sonores, et de reconstruire chacun d'entre eux grâce à un filtrage linéaire adapté. Nous avons vu que nous pouvions visualiser les notes de la mélodie en observant la transformée de Fourier. Nous allons voir que nous pouvons aussi modifier la mélodie en travaillant directement sur cette transformée de Fourier.

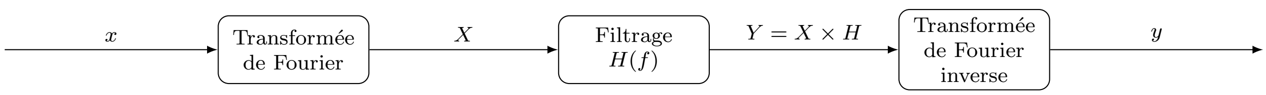
Pour cela, nous allons utiliser au choix un filtre passe-bas idéal (si l'on veut supprimer les notes aigues), passe-haut idéal (si l'on veut supprimer les notes graves), ou passe-bande idéal (si l'on veut supprimer toutes les notes aigues et graves et ne garder que les médiums). Les fonctions de transfert de ces filtres sont définies de la façon suivante et dépendent d'une ou plusieurs fréquences de coupure (selon le filtrage que l'on souhaite effectuer) :

$$\text{Passe-bas} \quad H_{LP}(f) = \begin{cases} 1 & \text{si } |f| < f_c \\ 0 & \text{sinon} \end{cases}$$

$$\text{Passe-haut} \quad H_{HP}(f) = \begin{cases} 0 & \text{si } |f| < f_c \\ 1 & \text{sinon} \end{cases}$$

$$\text{Passe-bande} \quad H_{BP}(f) = \begin{cases} 1 & \text{si } f_{c1} < |f| < f_{c2} \\ 0 & \text{sinon} \end{cases}$$

La suite des opérations à réaliser est présentée ci-dessous :



Il s'agit ici d'un filtrage linéaire vu dans le domaine fréquentiel, correspondant donc à une multiplication de la transformée du signal avec la fonction de transfert du filtre.

5.1 Cas de deux notes successives

Nous allons considérer un signal composé de deux notes successives, et tenter de supprimer l'une des deux.

1. Créer dans un nouveau script `TP_part5.m` un signal sonore correspondant à une mélodie contenant 2 notes : une aïgue et une grave (dont vous choisirez les hauteurs et les durées) et échantillonné à $F_s = 8000$ Hz. Tracer le module au carré de la transformée de Fourier discrète du signal et proposer une fréquence de coupure f_c permettant de séparer les deux notes.

2. Créer un vecteur `H_LP` et un vecteur `H_HP` correspondant aux fonctions de transfert du filtre passe-bas et passe-haut idéaux. Pour cela, il faudra créer des vecteurs de même taille que le vecteur de fréquences `f` renvoyé par la transformée de Fourier qui seront composés de 0 et de 1.

   Pour modifier des valeurs d'un vecteur grâce à une condition, on peut utiliser les commandes suivantes :

   ```
   y(y<0) = 0;          % Met a 0 toutes les valeurs strictement negatives de y
   z(y<3 & y>0) = 1;    % Met a 1 toutes les valeurs du vecteur z correspondant aux indices
                        % ou y est compris entre 0 et 3
   ```

3. Appliquer le filtre passe-bas `H_LP` sur la transformée de Fourier discrète calculée. Faire une transformée de Fourier inverse (grâce à la fonction `my_FFTinv.m` fournie) pour retrouver le signal sonore filtré. Ecouter et commenter. Observer le module au carré de la transformée de Fourier discrète du signal ainsi filtré et

commenter.

Pour multiplier deux vecteur entre eux terme à terme, on utilisera

```
z= y1.*y2;  % Multiplie les vecteur y_1 et y_2 terme a terme
z= y1./y2;  % Divise les vecteur y_1 et y_2 terme a terme
```

4. Faire la même chose avec le filtre passe-haut `H_HP`.

5.2 Séparation de sources supervisée

Nous allons maintenant créer deux mélodies, les additionner et les séparer grâce au filtrage que nous avons utilisé dans la partie précédente. La méthode que nous allons utiliser rentre dans la catégorie des méthodes supervisées, c'est à dire que l'on dispose d'une information a priori sur les signaux que l'on doit reconstruire (par exemple ici, on sait qu'une des sources ne contient que des notes aigues, et l'autre que des notes graves)

1. Créer dans un script `TP_part6.m` un signal `x1` d'une durée de 10 secondes, échantillonné à $F_s = 8000$ Hz et contenant des notes (au moins 4) comprises entre 50 et 71. Ecouter le signal obtenu.
2. Créer dans le même script un signal `x2` d'une durée de 10 secondes, échantillonné à $F_s = 8000$ Hz et contenant des notes (au moins 4) comprises entre 72 et 90. Ecouter le signal obtenu.
3. Créer le signal `x = x1 + x2`. Quelle fréquence de coupure f_c peut-on utiliser pour séparer les deux signaux `x1` et `x2` ?
4. En utilisant les filtres précédemment définis, reconstruire à partir de `x` les signaux `y1` et `y2` estimant respectivement `x1` et `x2`.
5. Ecouter les signaux obtenus et les comparer aux sources réelles. Calculer la distance euclidienne entre `x1` et `y1` et celle entre `x2` et `y2`.
6. Observer les spectres des signaux `x1` et `y1` puis ceux des signaux `x2` et `y2`
7. Tester d'autres fréquences de coupure f_c et commenter.

5.3 Séparation de sources aveugle

Cette fois-ci, il n'y a plus d'information a priori. Il va falloir établir une stratégie afin de séparer la ligne mélodique de l'accompagnement, mais de façon aveugle.

1. Créer un script `TP_Part7.m` Ouvrir le fichier `melody2.wav` grâce à MATLAB et l'écouter. A partir du travail réalisé précédemment, et en observant la transformée de Fourier, proposer et appliquer une stratégie pour obtenir uniquement la ligne mélodique.

Pour ouvrir un fichier son sous MATLAB, on utilisera la commande

```
[y,Fs]=wavread('test.wav'); % y : vecteur son, Fs : frequence d'echantillonnage
```

2. Tester plusieurs stratégies. Décrire et évaluer de façon objective vos expériences (par exemple en calculant la distance euclidenne entre le signal retrouvé renormalisé et la vérité terrain stockée dans le fichier `melody3.wav`)