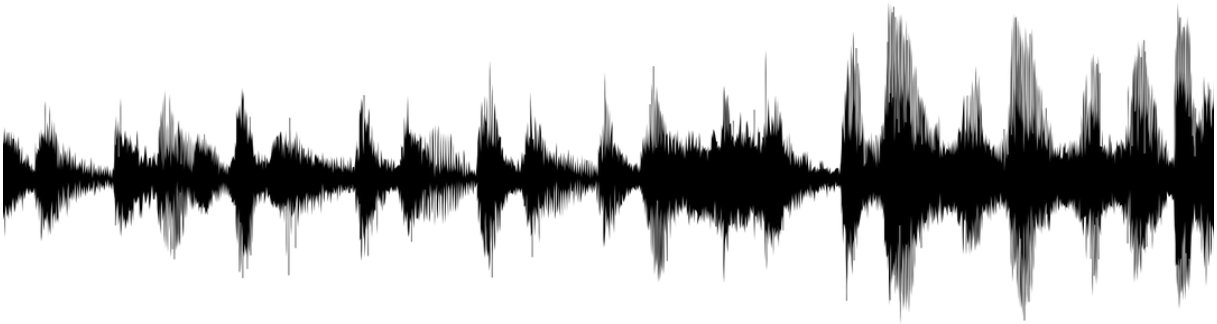
Compte Rendu de TP 02

Travaux pratiques de reconnaissance du son



Nom : BOUARICHE

Prénom : Iheb

Année : 2023/2024

Spécialité : Instrumentation an2

Séance 6 : Expérimentations et mise en place d'un prétraitement sur un son

b. Séquence de mots :

On réalise une fonction qui convertit un vecteur composé de chiffres à valeurs dans {1, 2, 3} en une séquence de mots, qui dictent la séquence de chiffres en utilisant la fonction suivante :

```
% I.c)
function synthetiseur_vocal3(v)
horiz=@(u)u(:)';
  if ~all(size(v)==[1 numel(v)])
    error('param2es invalides'),
  end
if ~all(ismember(v,[1 2 3]))
    error('param2es invalides'),
end
if isempty(v)
    error('param2es invalides'),
end
y=[];
for k=1:numel(v)
  if v(k) == 1
      a  = lireSon('donnees/un/',0);
      a = interp(a,2);
      y = [y,a'];

  elseif v(k) == 2
      a = lireSon('donnees/deux/',0);
      a = interp(a,2);
      y = [y,a'];

  elseif v(k) == 3
      a = lireSon('donnees/trois/',0);
      a = interp(a,2);
      y = [y,a'];

  end
 sound(y,44100);
end
```

c. Rendre le son plus aigu ou plus grave :

Question 3 : On doit montrer expérimentalement comment en modulant la fréquence d'échantillonnage, il est possible de rendre un son plus aigu ou plus grave et pourquoi la durée du signal est-elle modifiée.

Si Fe est grande -> le nombre de points est plus grand -> le vecteur est plus grand -> la durée du signal est plus longue.

Conclusion :

- Pour rendre un son aigu, il faut diminuer Fe et vis-versa.

- La durée du signal dépend de la fréquence d'échantillonnage.

Question 4 : On doit montrer comment grâce à l'interpolation, il est possible de rendre un son plus aigu ou plus grave, mais cette fois sans changer la fréquence d'échantillonnage.

Pour rendre le son plus aigu : on peut réduire la distance entre les points échantillonnés dans chaque trame en ajoutant des points supplémentaires. Cela augmente la densité des échantillons et peut donner l'impression que le signal varie plus rapidement dans le temps.

Pour rendre le son plus grave : on peut augmenter la distance entre les points échantillonnés en espaçant davantage les points supplémentaires. Cela peut donner l'impression que le signal varie plus lentement dans le temps.

Question 5 :

Mathématiquement la multiplication des deux sinusoïdes ça nous permet d'avoir une sommation de deux sinusoïdes avec une fréquence de 590Hz et l'autre avec une fréquence de 290Hz.

$$x_s(nT_e)x_m(nT_e) = \frac{1}{2}\sin\left(2\pi\frac{f_s+f_m}{f_e}n\right) + \frac{1}{2}\sin\left(2\pi\frac{f_s-f_m}{f_e}n\right)$$

Apres, le filtre passe-haut est utilisé pour supprimer les composantes de fréquence plus basses que la fréquence de coupure du filtre. Donc si on veut garder la composante à 590 Hz, on peut concevoir un filtre passe-haut avec une fréquence de coupure légèrement inférieure à 590 Hz. Cela permettra de laisser passer la composante à 590 Hz tout en atténuant les composantes de fréquence plus basses.

Le script :

```
duree = 2;
fe = 44100;
t = 0:1/fe:(duree-1/fe);
fs = 440;
xs = sin(2*pi*fs*t);
sound(xs, fe);

y1 = 0.5*sin(2*pi*(fs + 150)*t) + 0.5*sin(2*pi*(fs - 150)*t);

N = 1000;
fc = 440;

B = fir1(N, fc / (fe/2), 'high');
A = 1;

y2 = filter(B, A, y1);
sound(y2, fe);
```

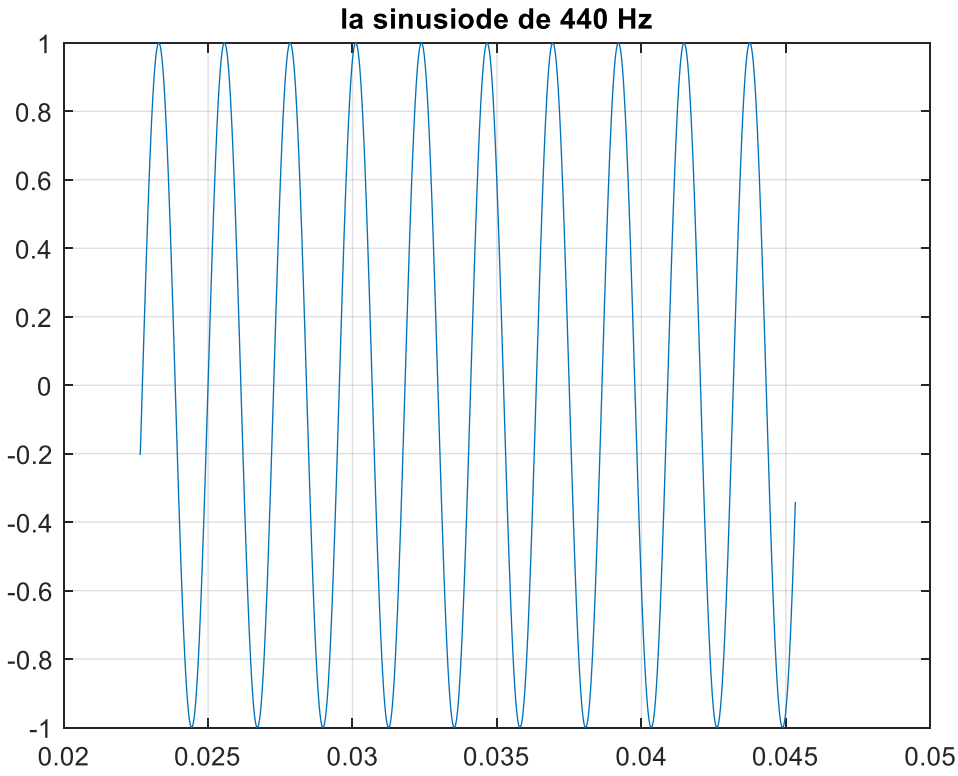


Figure : la sinusoïde à 440Hz.

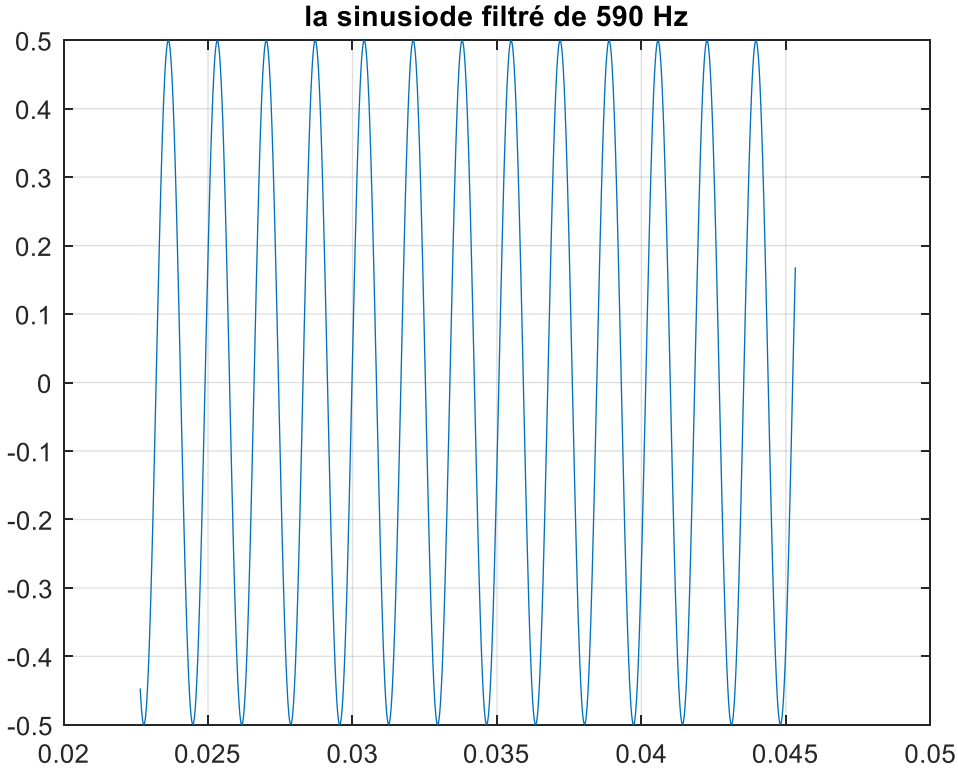


Figure : la sinusoïde filtrée à 590Hz.

Commentaire : on remarque qu'on a obtenu le même signal mais avec une fréquence plus grande (égale à 550 Hz).

d. Sous-échantillonner le signal :

Pour l'application considérée, la reconnaissance de mots, les informations au-delà de 4kHz ne sont pas jugées prioritaires. Aussi pour simplifier, les signaux sont sous-échantillonnés à 8kHz. Nous construisons une fonction qui réalise ce sous-échantillonnage avec la script suivant :

```
function [y2, fe2] = sous_ech(y1, fe1)

    fe2 = 8000;
    N = 1000;
    cutoff_frequency = 4000 / (fe1 / 2);
    B = fir1(N, cutoff_frequency);
    A = 1;
    z1 = filter(B, A, y1);

    t1 = 0:1/fe1:(length(y1)-1)/fe1;
    t2 = 0:1/fe2:t1(end);
    y2 = zeros(size(t2));

    for t2_ = 1:length(t2)
        t1_ = find(t1 >= t2(t2_), 1, 'first');
        y2(t2_) = z1(t1_);
    end
end
```

Dans ce code, j'ai utilisé la fonction fir1 pour concevoir un filtre passe-bas de 1000 termes avec une fréquence de coupure normalisée à 4000 Hz (la moitié de la fréquence d'échantillonnage cible de 8000 Hz). Les coefficients du filtre (B et A) sont utilisés pour filtrer le signal d'entrée y1 à l'aide de la fonction filtre. Ensuite, le signal filtré est sous-échantillonné en utilisant une interpolation linéaire pour obtenir le signal de sortie y2 à la fréquence d'échantillonnage cible de 8000 Hz.

e. Découper le son en une succession de trames :

```
clear all
clc
close all

fs = 10;
duree=1;
fe = 1e3;
t=0:1/fe:(duree-1/fe);
x = sin(2*pi*fs*t);
figure()
plot(x)
grid on
y = zeros(10,100);
for i=1:1:10
    for k=1:1:100
    y(i,k) = x(k*i);
    end
end

duree_trame = 0.1;
```

```
% On utilise les fonction
[xk,ech_trames,centres_trames] = decoupe_signal(x,1e3,duree_trame,0);
visualisation(x,fe,duree_trame,xk,ech_trames,centres_trames)

% Pour un signal de fe = 8Khz
fs = 10;
duree=1;
fe = 8e3;
t=0:1/fe:(duree-1/fe);
x = sin(2*pi*fs*t);
duree_trame = 0.1
[xk,ech_trames,centres_trames] = decoupe_signal(x,8e3,duree_trame,0);
visualisation(x,fe,duree_trame,xk,ech_trames,centres_trames)
```

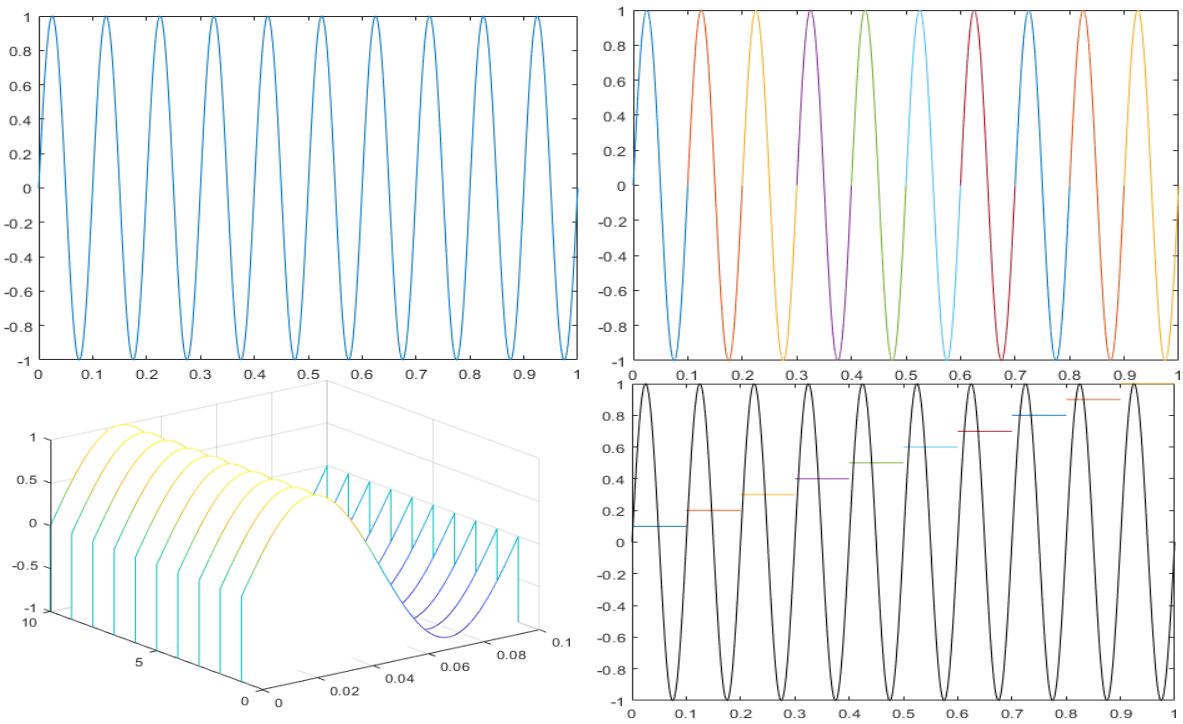


Figure : En haut à gauche : signal x en fonction du temps. En haut à droite : succession des trames affichées avec des couleurs différentes en fonction du temps. En bas à gauche : succession des trames représentées cette fois en fonction d'une base de temps propre à la trame. En bas à droite : signal x en noir en fonction du temps et indication colorée des instants associés à chaque trame.

Commentaire : on remarque qu'on a bien découpé le signal en 10 trames avec 100 ms pour chaque trame.

Nous découpons maintenant le signal x en trames de 100 ms avec 25% de chevauchement, et nous donnons les quatre visualisations (le résultat avec chevauchement est représenté sur la figure ci- dessous).

Script :

```
% avec un chevauchement de 25%
[xk,ech_trames,centres_trames] = decoupe_signal(x,1e3,duree_trame,0.25);
visualisation(x,fe,duree_trame,xk,ech_trames,centres_trames)
```

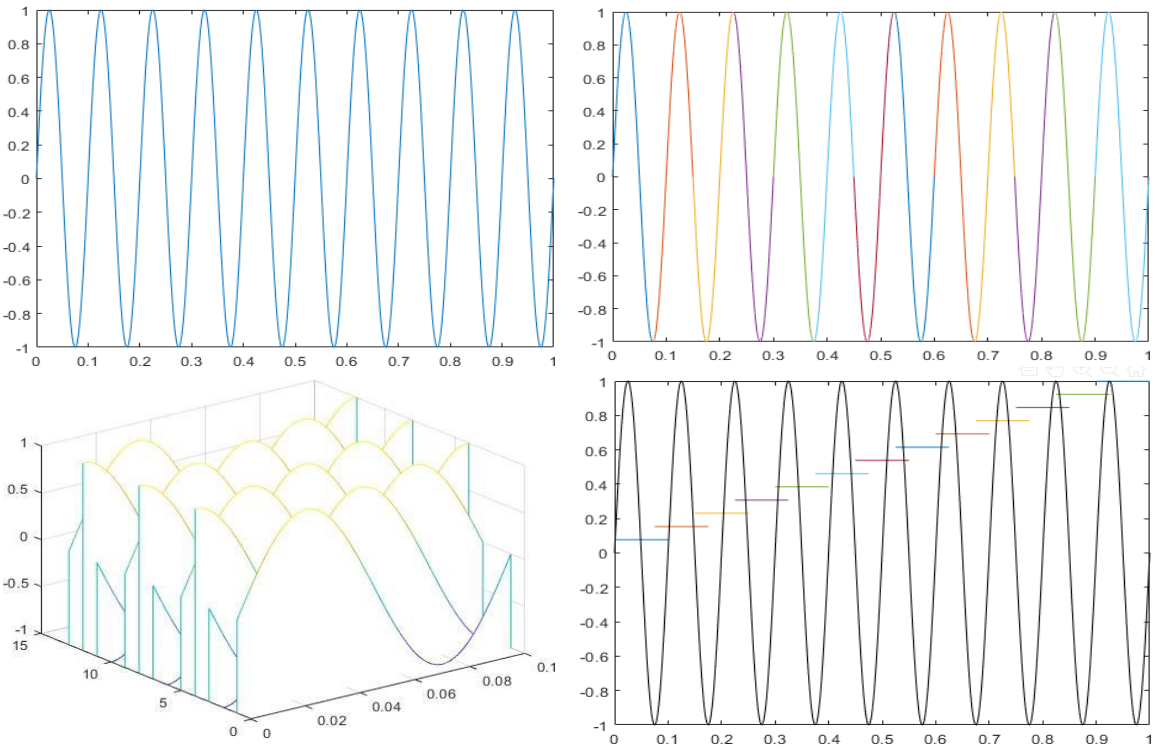


Figure : En haut à gauche : signal x en fonction du temps. En haut à droite : succession des trames affichées avec des couleurs différentes en fonction du temps. En bas à gauche : succession des trames représentées cette fois en fonction d'une base de temps propre à la trame. En bas à droite : signal x en noir en fonction du temps et indication colorée des instants associés à chaque trame.

Commentaire : Dans cette partie, le signal a été découpé en 10 trames de 100 ms avec un chevauchement. Cette caractéristique est clairement visible dans les quatre figures. Pour la figure en bas à droite, on peut observer le décalage des trames les unes par rapport aux autres, correspondant à un chevauchement de 25% de leur durée. Cette observation justifie la présence du chevauchement et démontre son impact sur la représentation graphique du signal.

La figure ci-dessous représente différentes trames découpées sans chevauchement. La fréquence d'échantillonnage utilisée est de 8kHz.

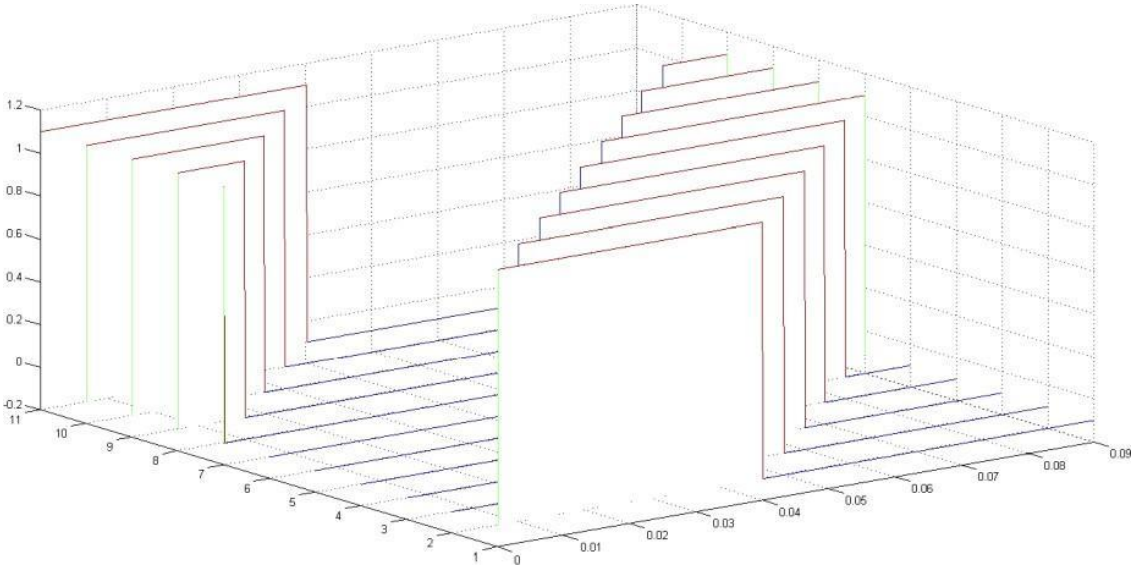


Figure : Figure relative à la question I.10. Succession des trames représentées cette fois en fonction d'une base de temps propre à la trame.

On essaie de retrouver les paramètres et le signal de départ utilisés pour obtenir cette figure. On remarque qu'on a des trames avec des signaux carré et avec une amplitude égale a 1.

Script :

```
t=0:1/8000:1-1/8000);

x=square(2*pi*t*10.44);

plot(t,x);
```

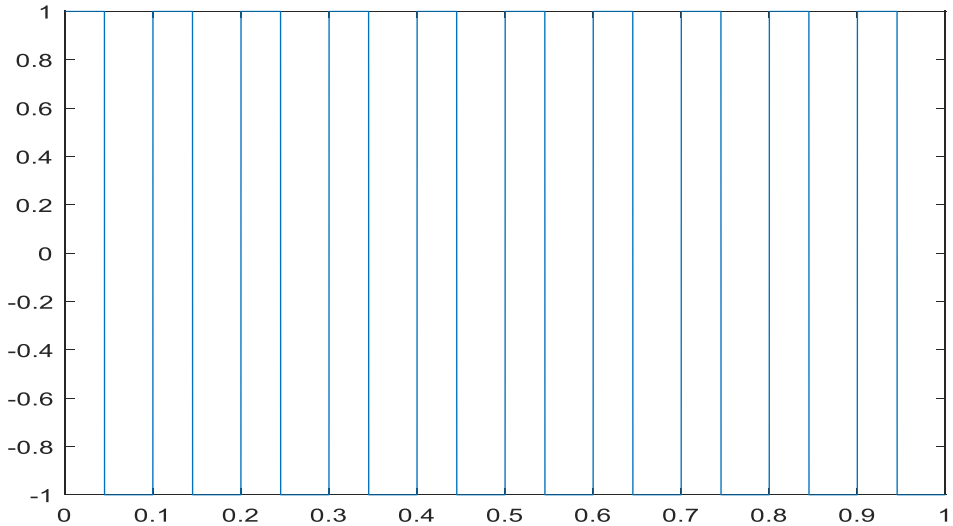


Figure : le signal originale pour avoir la succession des trames de la figure précédente.

f. Détection du début et de la fin d'un son :

On considère maintenant pour x un des sons enregistrés. Nous découpons x en trames de 30ms avec 0.25 de chevauchement. Ensuite, nous observons les quatre visualisations et en particulier l'importance du silence.

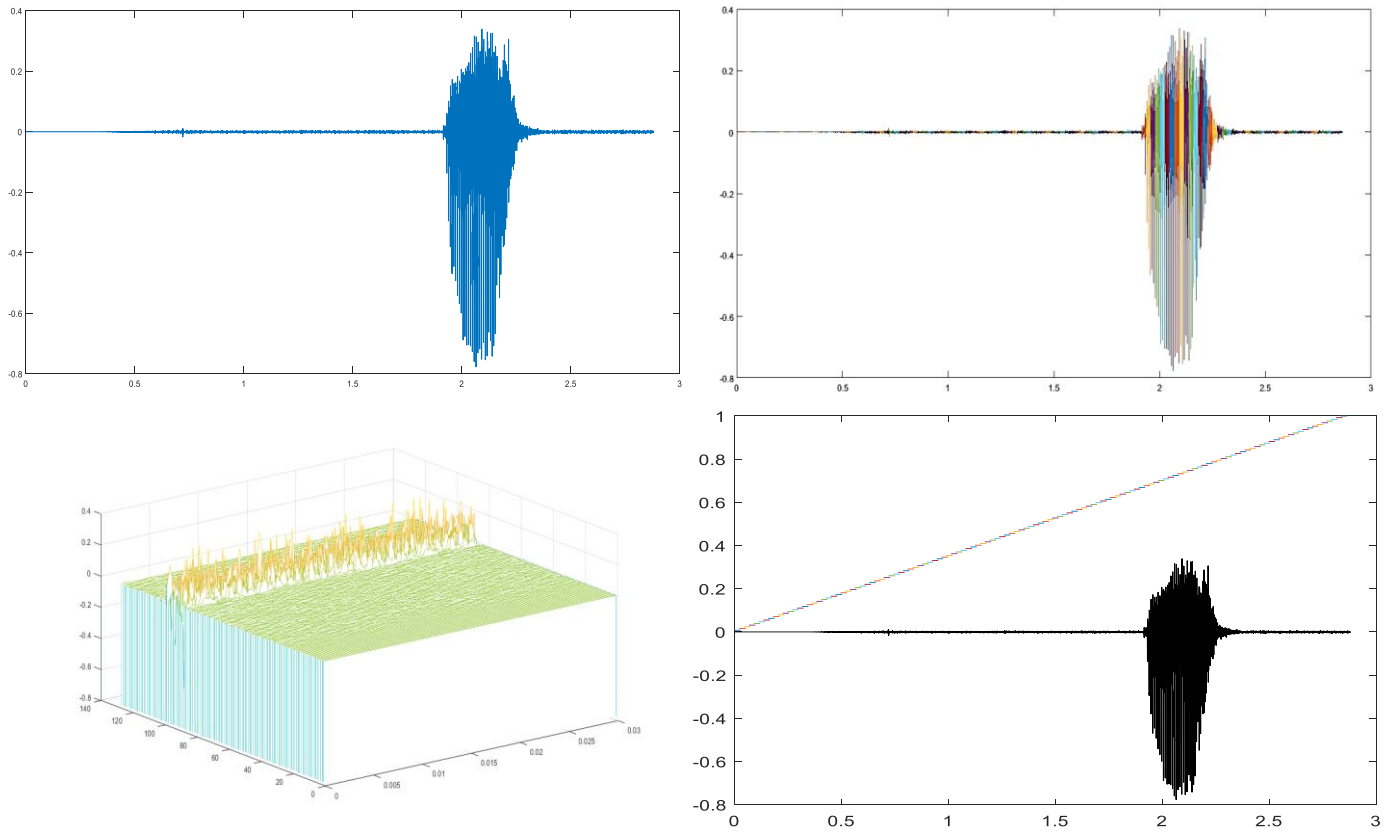


Figure : Figures relatives à la question I.12.

Nous calculons pour chaque trame PMCT. Ensuite, nous représentons sur un graphe le signal superposé à PMCT en multipliant PMCT par un facteur de sorte que le maximum de PMCT coïncide avec le maximum de la valeur absolue du signal x.

Nous créons une fonction c_PMCT qui calcule PMCT à partir du signal découpé suivant le script suivant :

```
function PMCT=c_PMCT(xk,x)
PMCT=s_c_PMCT(xk);
t1=0:3/(length(x)):3-1/length(x);
t2=0:3/(length(PMCT)):3-1/length(PMCT);
figure();
plot(t1,abs(x));
hold on;
plot(t2,max(x)*100*PMCT);
hold off;
```

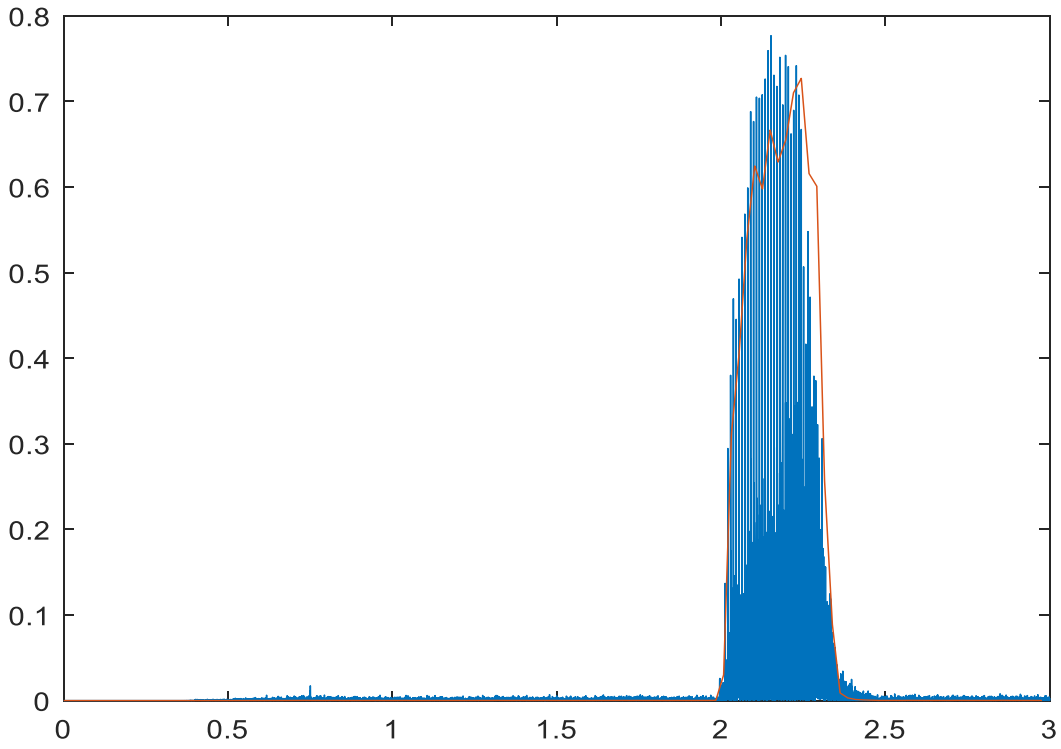


Figure : Figure relative à la question I.13.

Remarque : Nous observons que nous pouvons considérer PMCT comme une approximation de l'enveloppe énergétique de x.

Question 15 :

Le calcul des instants de début et de fin se fait par le script suivant :

Script :

```
indices = find(signal > seuil * max(signal));
t_debut=t(indices(1))
t_fin=t(indices(length(indices)))
```

Le résultat obtenu :

```
t_debut =

    2.0315


t_fin =

    2.3386
```

L'affichage de résultat sur un graphe se fait par le script suivant :

```
signal = s_c_PMCT(xk);
PMCT = signal;
t=0:3/(length(PMCT)):3-1/length(PMCT)
seuil = 0.1

indices = find(signal > seuil * max(signal));

figure;
plot(t, signal, 'b', 'LineWidth', 2);
hold on;
plot(t,max(signal)*seuil*ones(1,length(t)), 'r', 'LineWidth', 2);
scatter([t(indices(1)),t(indices(length(indices)))],
[signal(indices(1)),signal(indices(length(indices)))], 'g', 'filled');
title('Représentation du signal avec des instants de debut et de fin');
legend('Signal', 'Seuil', 'Instants où le signal dépasse 10% de la valeur
maximale');
xlabel('Temps');
ylabel('Amplitude');
grid on;
hold off;
```

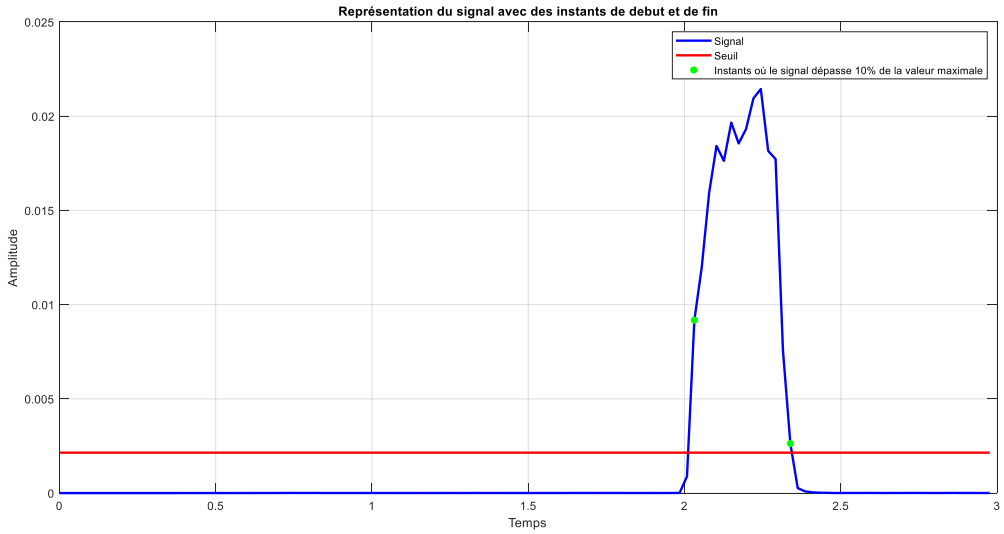


Figure : Représentation du signal avec des instants de débuts et de fin

Les points verts indiquent les instants ou le signal de PMCT est supérieur au seuil de 10%.

Question 16 :

On cherche une fonction qui modifie le signal découpé en supprimant les trames de silence :

Script :

```
function[xk,ech_trames,centres_trames]=supprime_silence(xk,ech_trames,centr
es_trames)
signal = s_c_PMCT(xk);
PMCT = signal;
t=0:3/(length(PMCT)):3-1/length(PMCT);
t_=0:3/(length(xk)):3-1/length(xk);
seuil = 0.1
```

```
% Trouver les indices où le signal dépasse le seuil
indices = find(signal > seuil * max(signal));

length(t)
length(t_)

figure;
round(indices(1)*1323/127)
round(indices(end)*1323/127)
xk = xk(round(indices(1)*1323/127):round(indices(end)*1323/127))

plot(t_(round(indices(1)*1323/127):round(indices(end)*1323/127)),xk, 'b',
'LineWidth', 2);
grid on
end
```

Script pour l'exécution :

```
duree_trame = 0.03;
[xk,ech_trames,centres_trames] = decoupe_signal(x,Fe,duree_trame,0.25);
supprime_silence(xk,ech_trames,centres_trames);
```

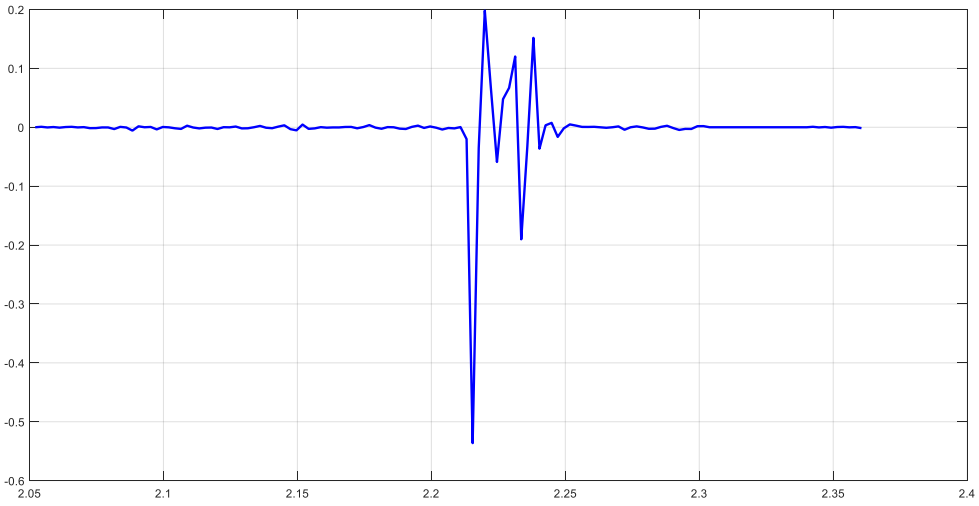


Figure : Signal découpé sans le silence.

Commentaire : on peut voir qu'on a obtenu une partie de signal ou son PMCT est supérieur à un seuil de 10%.

Donc a partir de PMCT on peut supprimer le silence et avoir les instants de début et de fin de son.

Séance 07 : Obtenir une classification grâce à des descripteurs.

a. Obtenir un descripteur par son à partir d'un descripteur par trame :

Question 18 :

On génère deux descripteurs par son en utilisant la moyenne et l'écart-type. On représente sur un graphique 2D l'ensemble des données. La position de chaque son apparaît sous la forme d'un point dont l'ordonnée est déterminée par l'estimation de l'écart-type de PMCT et l'abscisse est déterminée par la moyenne de PMCT. Et ça se fait on exécute le script suivant :

Script :

```
repertoire = 'donnees/un/';

fichiers_audio = dir(fullfile(repertoire, '*.m4a'));
mean_1 = [];
std_1 = [];
for i = 1:length(fichiers_audio)
    chemin_fichier = fullfile(repertoire, fichiers_audio(i).name);

    [x,Fe] = audioread(chemin_fichier);
    [xk,ech_trames,centres_trames]=preparation(x,Fe,duree_trame,0.25);
    PMCT = s_c_PMCT(xk);
    mean_1 = [mean_1 , mean(PMCT)]
    std_1  = [std_1 , std(PMCT)]

    fprintf('Lecture du fichier audio: %s\n', fichiers_audio(i).name);

end



repertoire = 'donnees/deux/';

fichiers_audio = dir(fullfile(repertoire, '*.m4a'));
mean_2 = [];
std_2 = [];
for i = 1:length(fichiers_audio)
    chemin_fichier = fullfile(repertoire, fichiers_audio(i).name);

    [x,Fe] = audioread(chemin_fichier);
    [xk,ech_trames,centres_trames]=preparation(x,Fe,duree_trame,0.25);
    PMCT = s_c_PMCT(xk);
    mean_2 = [mean_2 , mean(PMCT)]
    std_2  = [std_2 , std(PMCT)]

    fprintf('Lecture du fichier audio: %s\n', fichiers_audio(i).name);

end


repertoire = 'donnees/trois/';
```

```
fichiers_audio = dir(fullfile(repertoire, '*.m4a'));
mean_3 = [];
std_3 = [];
for i = 1:length(fichiers_audio)
    chemin_fichier = fullfile(repertoire, fichiers_audio(i).name);

    [x,Fe] = audioread(chemin_fichier);
    [xk,ech_trames,centres_trames]=preparation(x,Fe,duree_trame,0.25);
    PMCT = s_c_PMCT(xk);
    mean_3 = [mean_3 , mean(PMCT)]
    std_3  = [std_3 , std(PMCT)]

    fprintf('Lecture du fichier audio: %s\n', fichiers_audio(i).name);

end


scatter(mean_1,std_1, 10, 'r', 'filled');
hold on;

scatter(mean_2,std_2,  10, 'g', 'filled');

scatter( mean_3,std_3, 10, 'b', 'filled');

% Add labels and legend
xlabel('Moyenne');
ylabel('Ecart-type');
title('representation des descripteurs des trois sons');
legend('son "UN"', 'son "Deux"', 'son "Trois"');

hold off;
```

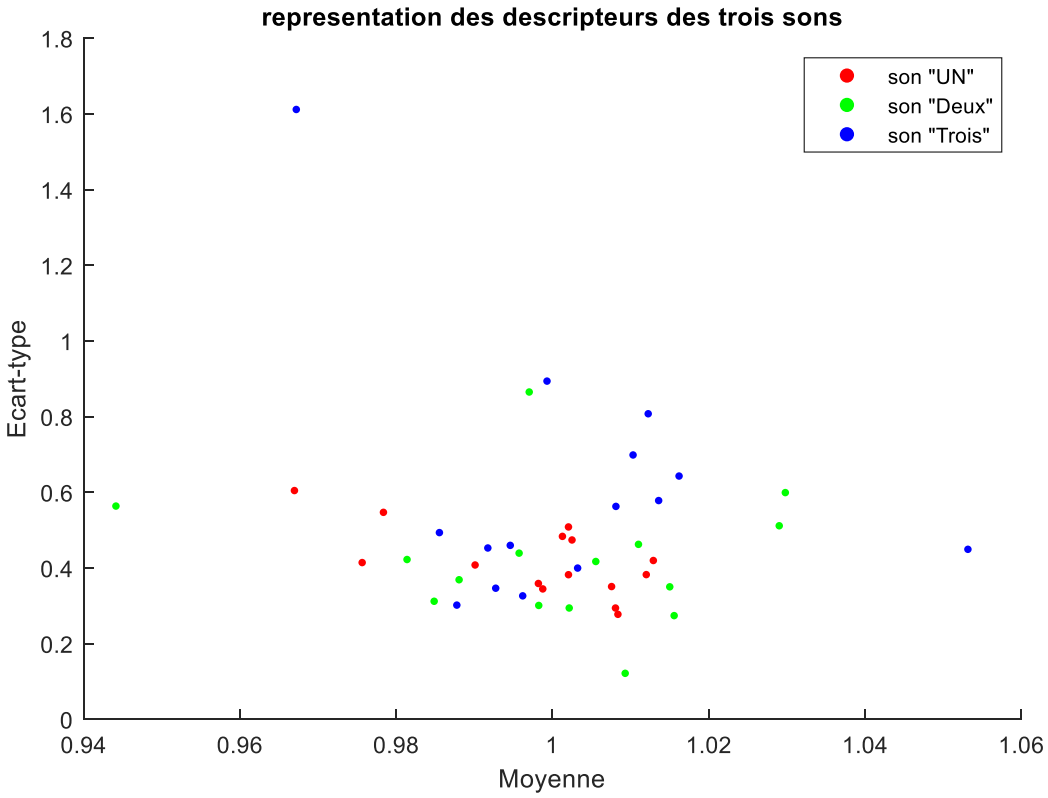


Figure : Graphe relatif à la question II.18. Chaque point correspond à un son coloré respectivement en rouge, vert et bleu, en fonction de ce que le son correspond à un, deux ou trois. La position du son est déterminée en abscisse par la moyenne de PMCT et en ordonnée à par l'écart-type de PMCT.

Commentaire : on remarque que c'est possible de définir un son avec des descripteurs, ce qui minimise la dimension de données et facilite l'interprétation et la classification. Et c'est ça se qu'on va faire par la suite.

Questions 19 : Une fonction qui calcule la distance euclidienne entre deux descripteurs.

Script:

```
function d=distance1(rep1,rep2)
[x1,fe1] = audioread(rep1);
[x2,fe2] = audioread(rep2);
duree_trame = 0.3;
chevauchement = 0.25;
[xk1,ech_trames1,centres_trames1]=preparation(x1,fe1,duree_trame,chevauchem
ent);
[xk2,ech_trames2,centres_trames2]=preparation(x2,fe2,duree_trame,chevauchem
ent);
PMCT1 = s_c_PMCT(xk1);
PMCT2 = s_c_PMCT(xk2);

d = sqrt((mean(PMCT1)-mean(PMCT2))^(2)+(std(PMCT1)-std(PMCT2))^(2));
end
```

Cette fonction prend comme descripteurs la moyenne et l'écart-type et calcule la distance entre ces deux descripteurs. Cette distance peut être utilisé pour la classification et reconnaissance de son.

b. Comment tester la performance d'un algorithme de classification à partir d'une distance entre sons :

Question 20 : On cherche à calculer la distance définie dans la question précédente pour les différents sons avec un son de « Un ». Et on a tracé les différentes distances dans un graphe.

Script :

```
repertoire = 'donnees/un/';

fichiers_audio = dir(fullfile(repertoire, '*.m4a'));
chemin_fichier = fullfile(repertoire, fichiers_audio(1).name);
[x_test,Fe] = audioread(chemin_fichier);
distances1 = [];
for i = 1:length(fichiers_audio)
    chemin_fichier = fullfile(repertoire, fichiers_audio(i).name);

    [x,Fe] = audioread(chemin_fichier);
    PMCT1 = s_c_PMCT(x);
    PMCT2 = s_c_PMCT(x_test);

    d = sqrt((mean(PMCT1)-mean(PMCT2))^(2)+(std(PMCT1)-std(PMCT2))^(2));
    distances1 = [distances1 d];
end


repertoire = 'donnees/deux/';
fichiers_audio = dir(fullfile(repertoire, '*.m4a'));
distances2 = [];
for i = 1:length(fichiers_audio)
    chemin_fichier = fullfile(repertoire, fichiers_audio(i).name);

    [x,Fe] = audioread(chemin_fichier);
    PMCT1 = s_c_PMCT(x);
    PMCT2 = s_c_PMCT(x_test);

    d = sqrt((mean(PMCT1)-mean(PMCT2))^(2)+(std(PMCT1)-std(PMCT2))^(2));
    distances2 = [distances2 d];
end


repertoire = 'donnees/trois/';

fichiers_audio = dir(fullfile(repertoire, '*.m4a'));
distances3 = [];
for i = 1:length(fichiers_audio)
    chemin_fichier = fullfile(repertoire, fichiers_audio(i).name);

    [x,Fe] = audioread(chemin_fichier);
    PMCT1 = s_c_PMCT(x);
    PMCT2 = s_c_PMCT(x_test);

    d = sqrt((mean(PMCT1)-mean(PMCT2))^(2)+(std(PMCT1)-std(PMCT2))^(2));
    distances3 = [distances3 d];
end

plot(distances1)
hold on
```

```
plot(distances2)
plot(distances3)

% Add labels and legend
xlabel('Les sons');
ylabel('la distance');
title('les distances avec differents groupe de sons');
legend('distance avec le groupe des sons "Un"','distance avec le groupe des
sons "deux"','distance avec le groupe des sons "Trois"');
```

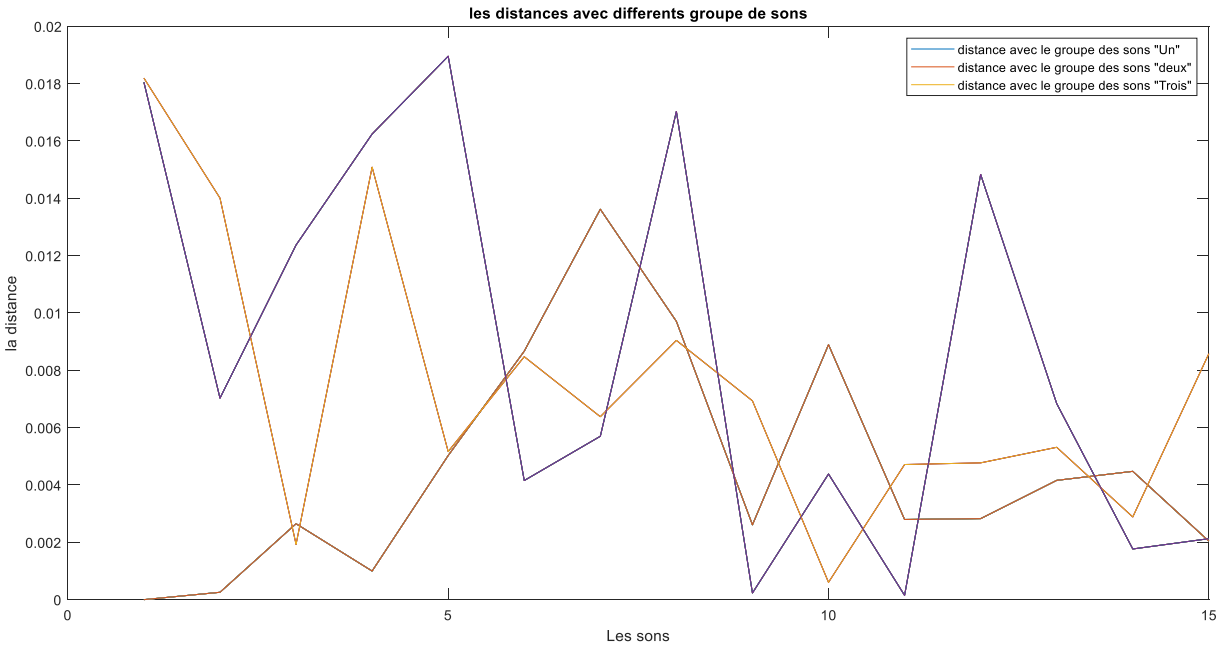


Figure : La distance entre un son de « Un » et les autres sons.

On peut remarquer que la distance change d'un son a l'autre et d'une classe à l'autre aussi.

Question 21 : On commence par la création des répertoires pour l'apprentissage.

Script :

```
rep_1={'C:\Users\IHEB.Home\Desktop\Nouveau dossier
(2)\donnees\un\','C:\Users\IHEB.Home\Desktop\Nouveau dossier
(2)\donnees\deux\','C:\Users\IHEB.Home\Desktop\Nouveau dossier
(2)\donnees\trois\'};
rep_simulation='C:\Users\IHEB.Home\Desktop\Nouveau dossier
(2)\simulation\';
rep_requete='C:\Users\IHEB.Home\Desktop\Nouveau dossier
(2)\simulation\requete'
rep_apprend_1={'C:\Users\IHEB.Home\Desktop\Nouveau dossier
(2)\simulation\un\','C:\Users\IHEB.Home\Desktop\Nouveau dossier
(2)\simulation\deux\','C:\Users\IHEB.Home\Desktop\Nouveau dossier
(2)\simulation\trois\ '};
i_ref=genere_un_pb_classification(rep_1,rep_simulation,rep_requete,rep_appr
end_1);
```

Et puis on va crier une fonction qui s'appelle prédit qui va nous donner la distance avec le point le plus proche de chaque groupe de donnée, le script est le suivant :

Script :

```
function mini=predit(rep_requete,rep_apprend_l)

fichiers_audio = dir(fullfile(rep_requete, '*.mat'))
chemin_fichier_requete = fullfile(rep_requete, fichiers_audio(1).name)
x1 = load(chemin_fichier_requete);
for k = 1:3
    fichiers_audio = dir(fullfile(rep_apprend_l{k}, '*.mat'))
for i=1:length(fichiers_audio)
    chemin_fichier = fullfile(rep_apprend_l{k}, fichiers_audio(i).name)
    x2 = load(chemin_fichier);
    duree_trame = 0.3;
    chevauchement = 0.25;
    fe = load(chemin_fichier).fe;

[xk1,ech_trames1,centres_trames1]=preparation(x1.y,fe,duree_trame,chevauche
ment);

[xk2,ech_trames2,centres_trames2]=preparation(x2.y,fe,duree_trame,chevauche
ment);
    PMCT1 = s_c_PMCT(xk1);
    PMCT2 = s_c_PMCT(xk2);

    d = sqrt((mean(PMCT1)-mean(PMCT2))^(2)+(std(PMCT1)-std(PMCT2))^(2));

    y(i,k) = d

    [mini, colIndex] = min(y + (y == 0) * max(y(:)))
end
end
end
```

Et puis le résultat si on applique cette fonction sur notre donnée de simulation est le suivant :

ans =

0.1090 0.0688 0.0906

Commentaire :

On voit bien que la distance avec le point deux est la plus petite, ce qui implique que le son d'entrée est « deux ».

Question 22 : On cherche maintenant à calculer la sensibilité globale pour la méthode de validation croisée avec K=5.

Script :

```
rep_l={'..\donnees\un\','..\donnees\deux\','..\donnees\trois\'};
rep_simulation='..\simulation\';
rep_requete='..\simulation\requete\';
rep_apprend_l={'..\simulation\un\','..\simulation\deux\','..\simulation\tro
is\'};
K=5;
genere_pb=genere_validation_croisee(rep_l,rep_simulation,rep_requete,rep_ap
prend_l,K);
D = 0;
N = 0;
for k=1:K
```

```
 y=genere_pb(k);
 Mr=length(y);
 D=D+Mr;
 yp=zeros(size(y));
 for mr=1:Mr
   yp(mr)=s_predit(rep_requete,mr,rep_apprend_l,@s_distance1);
 end
 N=N+sum(yp==y);
end
OA=N/D;
disp(['OA=',num2str(OA)])
```

```
>>OA=0.4
```

Commentaire : On obtient une valeur de 0.4 de sensibilité, ce qui n'est pas vraiment bon. Car on a 40% des prédiction qui sont juste, ce qui veut dire que notre prédicteur se trompe dans 60% des cas.

Question 23 : en utilisant le script de la question précédente pour réaliser une fonction qui calcule la sensibilité globale.

Script :

```
function OA=mesure_sensibilite(distance,K)
rep_1={'..\donnees\un', '..\donnees\deux\','..\donnees\trois\'} ;
rep_simulation='..\simulation\' ;
rep_requete='..\simulation\requete\';
rep_apprend_l={'..\simulation\un\','..\simulation\deux\','..\simulation\tro
is\ '} ;
genere_pb=genere_validation_croisee(rep_1,rep_simulation,rep_requete,rep_ap
pre nd_l,K);
N=0; D=0;
for k=1:K
disp(["K=",N2str(K),"k=",N2str(k)])
y=genere_pb(k) ;
if rand(1)<0.05
    v_genere_pb(rep_1,rep_simulation,rep_requete,rep_apprend_l,y) ;
end
Mr=length(y) ;
D=D+Mr ;
yp=zeros(size(y)) ;
for mr=1 :Mr
yp(mr)=s_predit(rep_requete,mr,rep_apprend_l,distance) ;
end
N=N+sum(yp==y) ;
end
OA=N/D ;
end
```

c. Utilisation d'une distance entre trames :

Le script :

```
duree_trame=30e-3; chevauchement=0.25;
M=lireSon('..\donnees\un\',-1);
M_vect=randperm(M);
m1=M_vect(1);
m2=M_vect(2);
[x,fe,~]=lireSon('..\donnees\un\',m1);
```

```
[xk1,ech_trames,centres_trames1]=preparation(x,fe,duree_trame,chevauchement
);
[x,fe2,~]=lireSon('..\donnees\un\',m2);
[xk2,ech_trames,centres_trames2]=preparation(x,fe,duree_trame,chevauchement
);
[x,fe3,~]=lireSon('..\donnees\deux\',m2);
[xk3,ech_trames,centres_trames3]=preparation(x,fe,duree_trame,chevauchement
);

PMCT1=s_c_PMCT(xk1);
PMCT2=s_c_PMCT(xk2);
PMCT3=s_c_PMCT(xk3);
m = min([length(PMCT1),length(PMCT2),length(PMCT3)]);
distancesA=abs(PMCT1(1:m)-PMCT2(1:m));
distancesB=abs(PMCT1(1:m)-PMCT3(1:m));

disp(['moy distA=',num2str(mean(distancesA)),'moy
distB=',num2str(mean(distancesB))])
figure(1);
plot(centres_trames1(1:m),distancesA,'b',centres_trames3(1:m),distancesB,'r
-');
```

Commentaire : Cette expérimentation montre que la variabilité entre des sons similaires est aussi importante que la variabilité entre des sons différents. Mais le problème est dans le nombre de trames dans chaque son et le changement de durée qui rendre cette expérience difficile pour la comparaison.

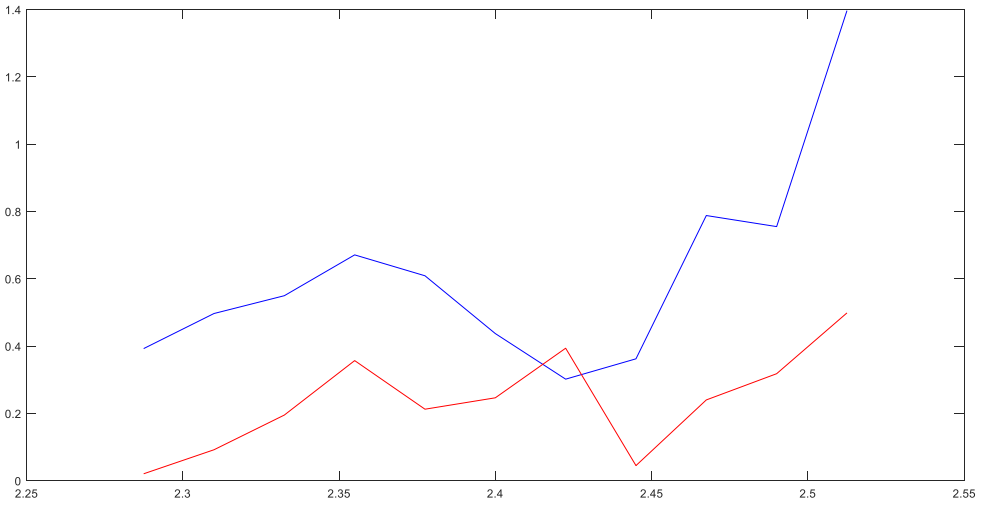


Figure : représentation de la distance en fonction du centre de la trame. (Rouge est la distance entre deux sons de même groupe et le bleu représente la distance de deux signaux de groupes différents)

Commentaire : on peut faire la différence avec un nombre minimal de trames, ce qui rend le traitement pas vraiment précis. Et pour ce problème, on peut utiliser la déformation temporelle.

d. Obtenir une distance entre sons à partir d'une distance entre trames :

Question 25 : Pour vérifier que toutes les valeurs sont positives, on peut comparer chaque instant avec l'instant suivante et on doit trouver que des valeurs positives, voici le script qui fais ça :

```
xn = 0:19;
```

```
tn = 0:19;
tpn = tn + randn(size(tn));

% Vérifier si t'_n+1 >= t'_n
differences = diff(tpn);
verification = all(differences >= 0);

% Afficher les résultats
disp(['Toutes les différences sont positives : ', num2str(verification)]);

% Tracer les signaux
figure;
plot(tn, xn, 'o-', tn, tpn, 'x-');
legend('t_n', 't''_n');
xlabel('n');
ylabel('Valeurs');
title('Comparaison entre t_n et t''_n');
grid on;
```

Résultat :

>> Toutes les différences sont positives : 0

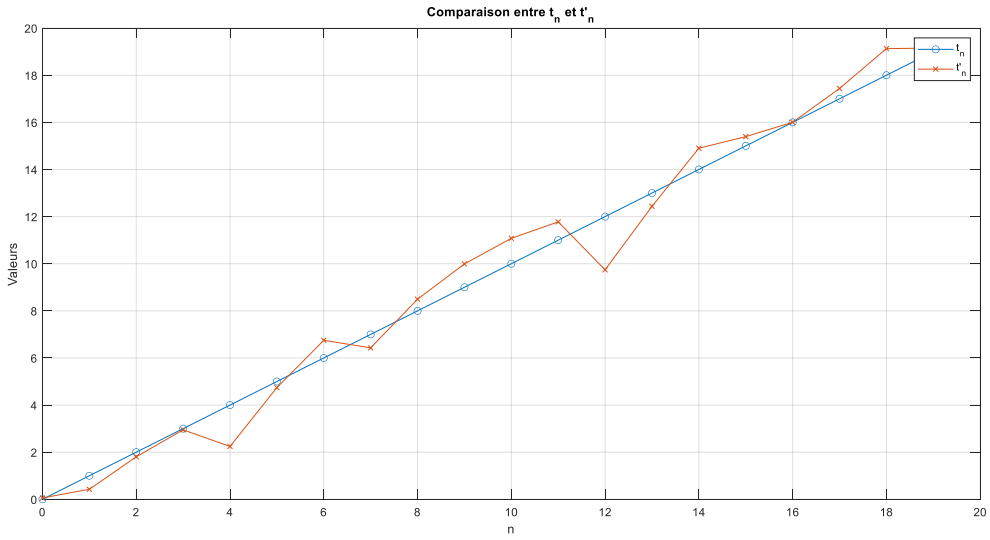
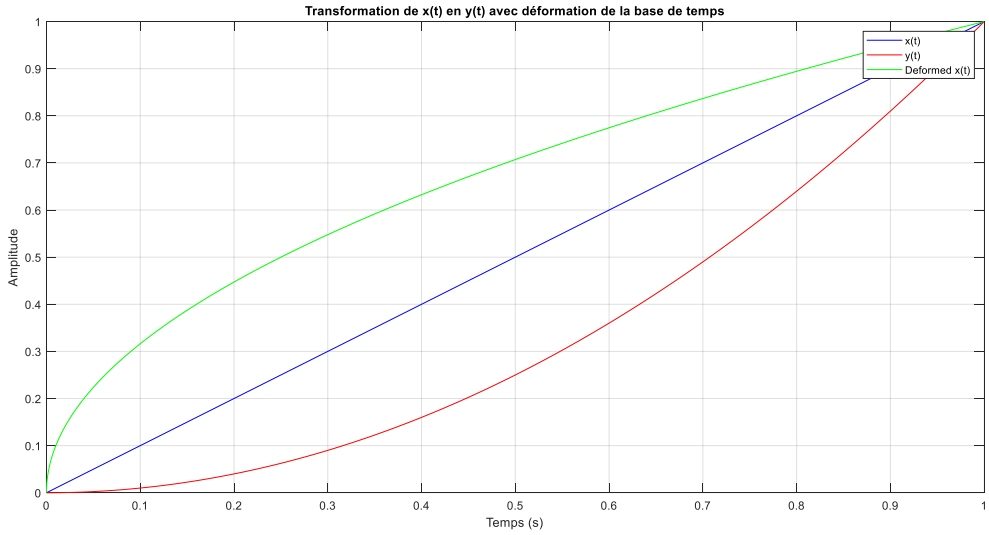


Figure : figure de comparaison.

Commentaire : on peut voir que la différence est n'est pas toujours supérieur à zéro, et on peut vérifier ça avec le résultat obtenu du script et le graph obtenu.

Question 26 :



Commentaire : Le graphique résultant montre les signaux x(t) (en bleu), y(t) (en rouge), et le signal x(t) déformé (en vert) en fonction du temps. On peut observer visuellement la transformation de la base de temps.

d.2 Déformation au moyen d'une table d'indexation :

Le script crée trois signaux :

- (t_n, x_n) : Signal bleu ('b') représenté par les variables tn et xn.
- (t'_n, y_n) : Signal vert ('g') représenté par les variables tpn et yn.
- $(t'_{\Phi y}[n], x_{\Phi x}[n])$: Signal rouge ('r-') représenté par les indices Φx et Φy appliqués aux signaux tn et xn.

Les lignes de commandes vérifient certaines propriétés définies dans l'équation (23) :

- $\Phi x[n] \leq \Phi x[n+1]\, et\, \Phi x[0] = 0, \Phi x[N\Phi - 1] = Nx - 1$
- $\Phi y[n] \leq \Phi y[n+1]\, et\, \Phi y[0] = 0, \Phi y[N\Phi - 1] = Ny - 1$
- $|\Phi x[n] - \Phi y[n]| \leq \delta$

La boucle ***while*** génère des indices Φx et Φy. Et la figure 6 résultante est affichée avec les trois signaux superposés. Les couleurs sont attribuées comme suit :

- Signal original (t_n, x_n) : Bleu ('b')
- Signal d'origine déformé (t'_n, y_n) : Vert ('g')
- Signal transformé $(t'_{\Phi y}[n], x_{\Phi x}[n])$: Rouge ('r-')

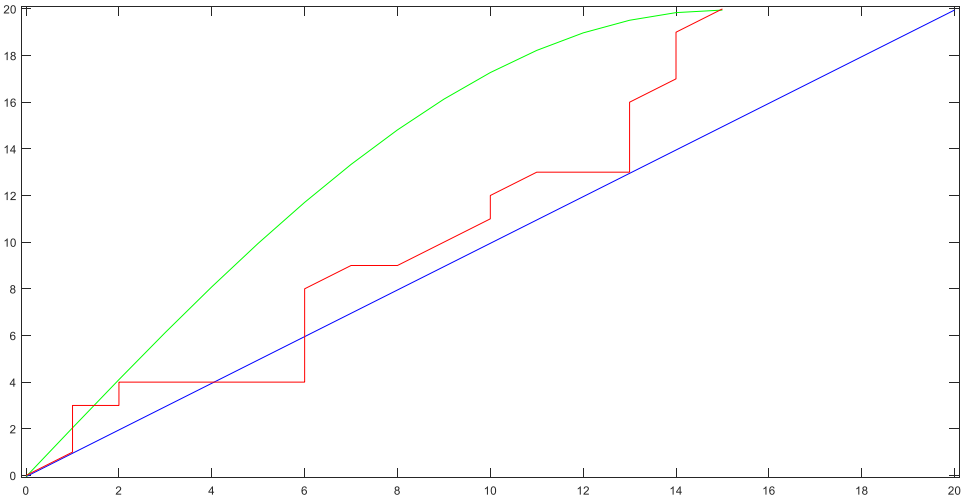


Figure 6 : $(t_n, x_n), (t'_n, y_n)$ et un signal transformé $(t'_{\Phi y}[n], x_{\Phi x}[n])$.

Ici on réalise une approximation très grossière de la minimisation en tirant aléatoirement 200 tables d'indexations Φx et Φy et en considérant les tables qui réalisent cette minimisation.

On modifie les lignes de code de façon à trouver une approximation de cette distance pour δ = 6.

Le script :

```
clear ; xn=0 :20 ;
I=length(xn) ; tn=0 :20 ;
tpn=0 :15 ; yn=sin(tpn/15*pi/2)*20 ; J=length(yn) ;
delta=6 ; d_min=Inf ;
e=@()ceil(rand(1)*2)-1;
for exp=1 :200
ok=0 ;
while(~ok)
    Phi_x=[1] ;
    Phi_y=[1] ;
while(1)
Phi_x=[Phi_x(:)' Phi_x(end)+e()];
Phi_y=[Phi_y(:)' Phi_y(end)+e()];
if Phi_x(end)>=length(xn)
    break;
end
if Phi_y(end)>=length(tpn)
    break ;
end
end
ok=1 ;
if ~(Phi_x(end)==length(xn))
    ok=0;
end
if ~(Phi_y(end)==length(tpn))
    ok=0;
end
if ~(max(abs(Phi_y-Phi_x))<=delta)
    ok=0;
end
end
d=sum((yn(Phi_y)-xn(Phi_x)).^2)/(I+J) ;
```

```
if d<d_min
d_min=d ;
disp(['exp=',num2str(esp), 'd_min=',num2str(d_min)])
figure();
plot(tn,xn-0.05,'b',tpn,yn-0.05,'g',tpn(Phi_y),xn(Phi_x),'r-') ;
axis([-0.1 max(max(tpn),max(tn))+0.1 -0.1 max(max(xn),max(yn))+0.1])
end
end
```

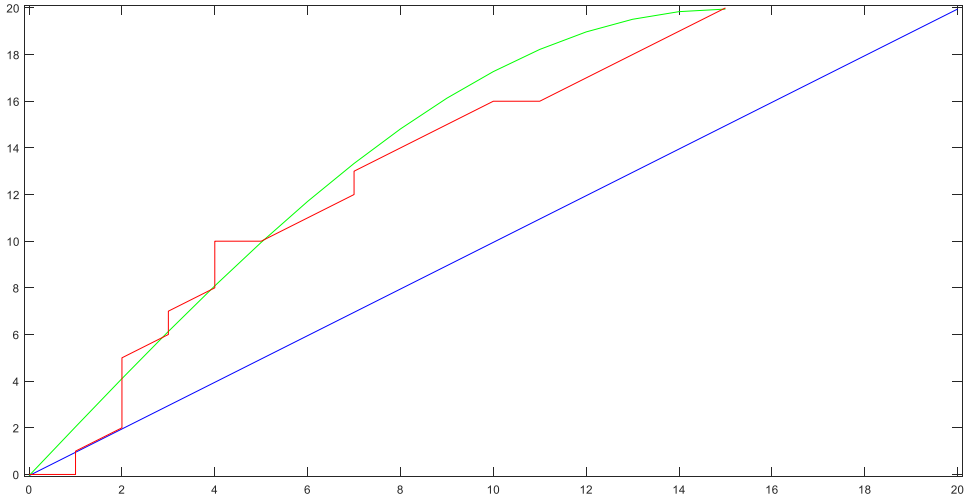


Figure : (t_n, x_n), (t'_n, y_n) et un signal transformé $(t'_{\Phi y}[n], x_{\Phi x}[n])$.

Le résultat obtenu :

>>

exp=1d_min=7.0204

exp=4d_min=6.7487

exp=6d_min=2.6434

exp=111d_min=2.3229

exp=136d_min=1.0975

Commentaire :

Dans cette partie on a effectué une approximation de la distance minimale entre deux signaux déformés dans le temps (xn et yn). La déformation temporelle est déterminée par des tables d'indexations Φx et Φy qui sont tirées aléatoirement.

Voici une explication détaillée du script :

- Ce script génère deux signaux xn et yn, où xn est un signal de longueur 21 (0:20) et yn est généré en déformant temporellement un signal sinusoïdal.

- Il définit un certain nombre de paramètres, dont la distance maximale autorisée (delta = 6), le nombre d'itérations pour la recherche aléatoire (200), et initialise une variable pour la distance minimale trouvée (d_min = Inf).

- Il utilise une boucle principale (for $\boldsymbol{exp = 1:200}$) pour effectuer 200 expériences (c'est-à-dire tirer aléatoirement 200 ensembles de tables d'indexations Φx et Φy).

- À l'intérieur de chaque expérience, une boucle $\boldsymbol{while}$ est utilisée pour générer aléatoirement Φx et Φy tout en respectant certaines conditions, comme spécifié dans les propriétés (23).

- Une fois les tables d'indexations Φx et Φy générées, la distance entre les signaux déformés est calculée selon l'équation (24).

- Si la distance calculée est inférieure à la distance minimale précédemment trouvée (d_min), la distance minimale est mise à jour.

- La boucle se répète jusqu'à ce que les 200 expériences soient terminées.

En fin de compte, le script affiche le minimum de la distance trouvée sur l'ensemble des expériences. Et on sait que L'objectif est d'obtenir une approximation de la distance minimale entre les deux signaux déformés en prenant en compte la déformation temporelle.

d.3. Heuristique permettant de trouver rapidement une distance tenant compte d'une déformation temporelle dynamique :

Le script :

```
clear
xn=0:5;
tn=0:5;
tpn=0:3;
yn=[0 2 4 5];
delta=5;
Phi_x =[ 1 2 3 4 5 6];
Phi_y =[ 1 1 1 2 3 4];
ok=1; %ok=0 pour regénérer une nouvelle solution
e_=[1 0;1 1; 0 1];
e=@()e_(ceil(rand(1)*3),:);
while(~ok)
Phi_x=[1];
Phi_y=[1];
while(1)
e_v=e();
Phi_x=[Phi_x(:)' Phi_x(end)+e_v(1)];
Phi_y=[Phi_y(:)' Phi_y(end)+e_v(2)];
if Phi_x(end)>=length(xn) break; end
if Phi_y(end)>=length(tpn) break; end
end
ok=1;
if ~(Phi_x(end)==length(xn)) ok=0; end
if ~(Phi_y(end)==length(tpn)) ok=0; end
if ~(max(abs(Phi_y-Phi_x))<=delta) ok=0; end
end
figure(1); plot(tn,xn-0.05,'bx-',tpn,yn-0.05,'gx-
',tpn(Phi_y),xn(Phi_x),'rx-','Linewidth',2);
axis([-0.1 max(max(tpn),max(tn))+0.1 -0.1 max(max(xn),max(yn))+0.1])
xlabel('t_n'),
text(0.5*(tn(4)+tn(5)),0.5*(xn(4)+xn(5))-0.1,'\leftarrow
(t_n,x_n)','FontSize',14);
text(0.5*(tn(2)+tn(3)),0.5*(yn(2)+yn(3))-0.1,'\leftarrow
(t_n,y_n)','FontSize',14);
text(0.5*(tpn(Phi_y(4))+tpn(Phi_y(5))),0.5*(xn(Phi_x(4))+xn(Phi_x(5)))-
0.1,'\leftarrow (t_{\Phi,y}Axis=[-1.1
```

```
max(max(length(tpn)),max(length(tn)))+1.1.-1.1
max(max(length(xn)),max(length(yn)))+1.1]';
figure(2); plot(Phi_x-1,Phi_y-1,'o-',...
[Axis(1), Axis(2)],delta+[Axis(1), Axis(2)],'k-',...
[Axis(1), Axis(2)],delta+[Axis(1), Axis(2)]+0.05,'k:',...
[Axis(1), Axis(2)],delta+[Axis(1), Axis(2)]+0.1,'k:',...
[Axis(1), Axis(2)],-delta+[Axis(1), Axis(2)],'k-',...
[Axis(1), Axis(2)],-delta+[Axis(1), Axis(2)]-0.05,'k:',...);
[Axis(1), Axis(2)],-delta+[Axis(1), Axis(2)]-0.1,'k:',...
'LineWidth',2);
axis(Axis)
for k=1:length(Phi_x)
dk=(yn(Phi_y(k))-xn(Phi_x(k))).^2;
text(Phi_x(k)-0.4-1,Phi_y(k)+0.4-1,num2str(dk,'%2.1f'));
end
text(3,5,'Lieu permis'),
xlabel('\Phi_x[n]'), ylabel('\Phi_y[n]'),
```

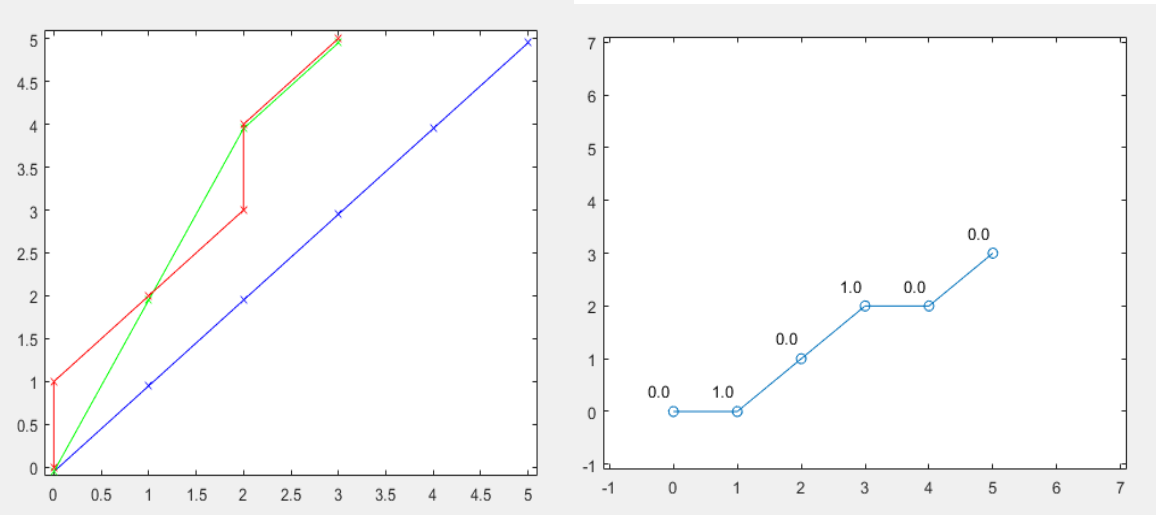


Figure : Figures relatives à la question II.29. A droite, les tables d'indexations sont représentées avec en ordonné $\Phi y[n]$ et en abscisse $\Phi x[n]$. Il y a six ronds ce qui correspond à six valeurs de n de 0 à 5 identiques au nombre de croix sur la courbe rouge à gauche. Le premier nombre 0.0 indique que le premier plus rouge en bas à gauche coïncide avec le premier plus de la courbe verte. Le deuxième nombre 1.0 traduit le fait que pour n=1, il y a une différence de 1 entre le deuxième plus de la courbe rouge qui s'est éloigné du deuxième plus de la courbe verte dont la deuxième croix s'est aussi éloignée. La différence de 1 une fois au carré reste égale à 1.

Commentaire : On peut voir les deux tables d'indexations comme un chemin particulier permettant de relier le point $(0, 0)$ au point $(Nx - 1, Ny - 1)$. Ce chemin est composé d'arêtes horizontales, verticales et en diagonales et à chaque arrêt, il y a une valeur correspondant à $d\Phi x, \Phi y\ (x, y, n) = |x\Phi x[n] - y\Phi y[n]|$.

En sommant les carrés de ces différentes valeurs et avec une normalisation nous obtenons une distance associée à un chemin $d\Phi x, \Phi y\ (x, y)$.

Question30 : En exécutant à de multiples reprises les lignes de codes de la question précédente, combien y a-t-il de nœuds possibles et combien d'arêtes possibles. Observez qu'il n'y a qu'une seule valeur possible par arête.

En plus de la racine et du puit, il y a 11 nœuds et 23 arêtes possibles :

- 2 nœuds avec une valeur de 0.
- 4 nœuds avec une valeur de 1.
- 4 nœuds avec une valeur de 4.
- 1 nœuds avec une valeur de 9.

Les chemins acceptables sont uniquement ceux passant par les nœuds que nous avons pu visualiser en appliquant plusieurs fois les lignes de commandes. Ces chemins ne doivent être composés que d'arêtes allant vers la droite, en diagonale montante et vers la droite et en allant verticalement vers le haut et ce dans les trois cas en se déplaçant que d'un pas à la fois. Les trois types d'arêtes sont respectivement notés H, D, V, de manière à ce qu'un chemin puisse être codé sous la forme d'une succession de ces symboles. Nous observons que les chemins ne sont pas tous de la même longueur.

Question 31 : Proposez un chemin le plus court, le plus long, celui dont la somme des valeurs associées est la plus faible et la plus élevée.

- Chemin le plus court : D,D,H,H,D.
- Chemin le plus long : H,H,V,V,H,H,D.
- Chemin dont la sommes des valeurs associées à chaque nœud traversé est la plus faible : D,H,D,H,D. Cela vaut 2.
- Chemin dont la somme des valeurs associées à chaque nœud traversé est la plus élevée : V,D,H,H,H,D. Cela vaut 18.

II.d.4 Utilisation de cette distance en simulation numérique :

De façon un peu similaire à la question II.19, nous créons une fonction distance2 permettant de comparer deux sons avec prise en compte de la déformation temporelle dynamique en l'utilisant non pas sur les valeurs des deux signaux mais sur leur puissance moyenne à court terme sur chaque trame. La durée de chaque trame est de 30ms et le chevauchement est de 0.25. Pour δ, nous utiliserons un nombre de trame égale à un dixième du nombre de trames du signal le plus court.

Script :

```
Functiond=distance2(rep1,m1,rep2,m2)
[x1,fe]=lireSon(rep1,m1) ;
[x2,fe]=lireSon(rep2,m2) ;
duree_trame=30e-3 ;
chevauchement=0.25 ;
xk1=preparation(x1,fe,duree_trame,chevauchement) ;
xk2=preparation(x2,fe,duree_trame,chevauchement) ;
PMCT1=s_c=PMCT(xk1) ;
PMCT2=s_c_PMCT(xk2) ;
delta=ceil(min(length(PMCT1),length(PMCT2))/10) ;
d=dist_DTD(PMCT1,PMCT2,delta) ;
end
```

Enfin, nous utilisons la question II.23 pour calculer la sensibilité globale (OA) par une validation croisée et nous comparons les performances entre celles obtenues avec distance1 et distance2.

Script :

OA1=s_mesure_sensibilite(@s_distance1,5) ;

OA2=s_mesure_sensibilite(@s_distance2,5) ;

Les résultats :

>> OA1=0.25

>> OA2=0.43

Séance 08 : Descripteurs de trames

a.1 Éstimation de la fréquence fondamentale, F0_ZCR :

Question 37 :

Script :

```
t=0:1e-3:5;

x1=sin(2*pi*t)+1/2*sin(2*pi*5*t);

x2=sin(2*pi*t+pi/20);

figure(1);

plot(t,x1,t,x2,t,0,'k-')

grid on ;
```

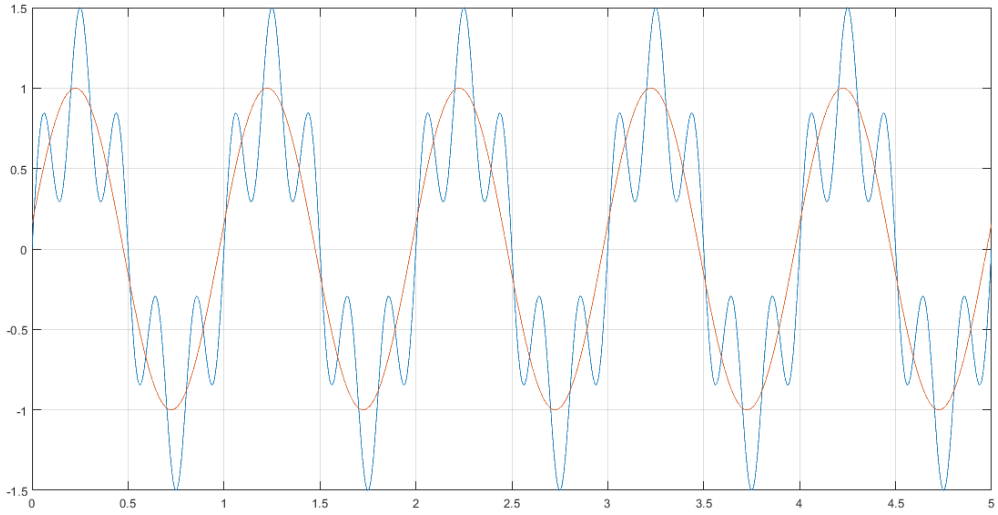


Figure : Les courbes bleues et vertes ont le même nombre de passages à zéro. Mais la courbe bleue est une sinusoïde de fréquence 1Hz, tandis que la courbe verte est une somme de sinusoïde dont la fréquence la plus basse est aussi 1Hz.

Commentaire : On peut bien voire que l'intersection de signal avec l'axe des abscisses se fait pour les fréquences les plus basse.

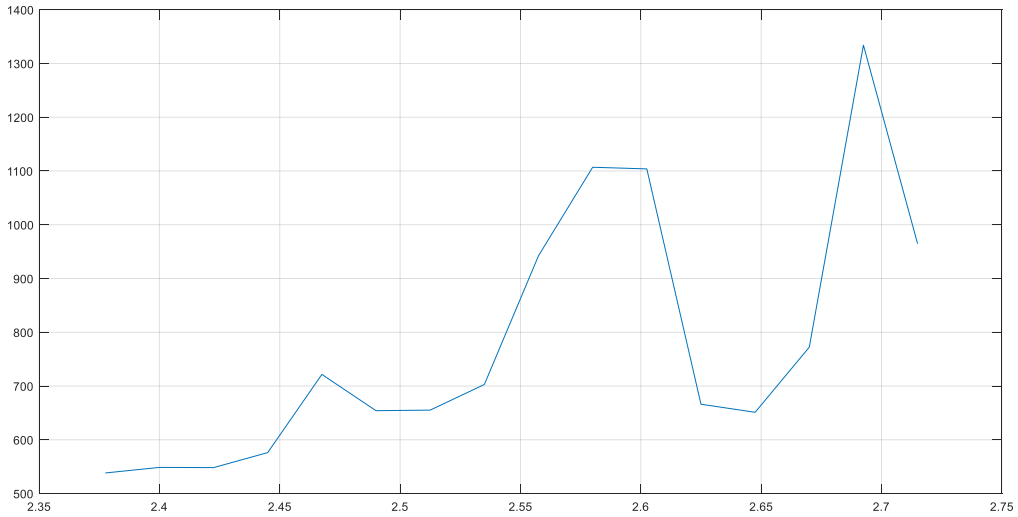


Figure : l'évolution de descripteur en fonction du temps.

Commentaire : on constate un changement de descripteur (fréquence fondamental) en fonction du temps (pour chaque trame).

Question 38 :

Le script pour la fonction qui calcule la fréquence fondamentale :

```
function F0_ZCR=c_F0_ZCR(xk,fe)
[xk,ech_trames,centres_trames]=preparation(xk,fe,30e-3,0.25);
K=size(xk,1);
F0_ZCR=zeros(K,1);
fenetre=window(@hamming,length(xk(1,:)))';
for trame=1:K
Z=abs(sign(xk(trame,2:end))-sign(xk(trame,1:end-1)))/2;
F0_ZCR(trame)=mean(Z.*fenetre(1:end-1))*fe/2/mean(fenetre(1:end-1));
end
end
```

Question 39 :

Le script pour la fonction qui calcule la distance en prenant en considération la difformation temporelle :

```
function distance3 = distance3(xk1,xk2,fe)
zk1 = c_F0_ZCR(xk1,fe);
zk2 = c_F0_ZCR(xk2,fe);
distance3 = distance2(zk1,zk2)
end
```

Commentaire : Dans cette fonction on a utilisé les fonctions pour le calcul de la fréquence fondamentale de chaque trame pour chaque son et puis on calculer la distance entre les deux descripteurs en prenant en considération la déformation temporelle.

Conclusion :

En conclusion, ce TP nous a permis d'acquérir des compétences importantes dans la manipulation des signaux sonores, caractérisés par des vecteurs de valeurs considérablement étendus. Nous avons appris à représenter ces vecteurs à l'aide de descripteurs tels que la moyenne, l'écart-type, la puissance, et d'autres paramètres pertinents.

De plus, nous avons exploré la déformation temporelle des signaux sonores, ajustant leur durée pour les rendre comparables à d'autres sons. Cette étape est cruciale pour harmoniser les données et faciliter la comparaison.

Enfin, nous avons abordé la comparaison de deux sons, ainsi que la classification et la reconnaissance des modèles sonores. Ces aspects sont essentiels dans le domaine du traitement du signal audio, permettant d'identifier et de catégoriser efficacement les différentes caractéristiques des sons.

Dans l'ensemble, ce travail pratique a fourni une compréhension approfondie des méthodes de manipulation, de comparaison et de classification des signaux sonores, enrichissant ainsi nos connaissances dans le domaine du traitement du signal audio.