Traitement du signal
TD 4
Linear Predictive Coding (LPC)

C. Dossal

Octobre 2007

1 Introduction

Le but de ce TP est d'utiliser le modèle LPC de la parole pour rédiger des programmes qui analyse et synthétise la parole et aussi pour effectuer des tâches simples de reconnaissance vocale. Comme dans le TP précédent sur le vocoder, le son sera fenêtré proprement, analysé et synthétisé sur des périodes de l'ordre de 30 ms.
Nous utiliserons la commande LPC directement contenue dans la toolbox signal de matlab qui utilise l'algorithme de Levinson Durbin pour calculer les coefficients LPC. Nous rappelons que cette commande renvoie $p+1$ coefficients dont le premier est toujours 1.
Nous utiliserons dans un premier temps le ficheier aeio.wav. Vous utiliserez ensuite vos propres enregistrements.

2 Analyse par LPC

2.1 Calcul et visualisation des coefficients LPC

1. Ecrire une fonction

   `function Coefs=CalculCoefsLPC(name,p)`

   qui prend pour entrée un fichier et un entier p et qui renvoie le tableau des coefficients LPC associés à chaque fenêtre de 40 ms. On utilisera un fenêtrage de Hamming pour l'analyse et un recouvrement de moitié, c'est à dire que par chaque point passe deux fenêtres. Le Tableau de sortie comportera $p+1$ colonnes et $nfenetres$ lignes.

2. Ecrire une fonction

   `function Visu(Coefs,nfenetres)`

   qui prend en entrée un jeu de $p+1$ coefficients et une taille de fenêtre et qui affiche en db le spectre du filtre associé, c'est à dire le logarithme de la valeur absolue de la fft du filtre associé.
   Faites des tests sur différents jeux de coefficients extraits de différentes parties du tableau. Comparez des jeux de coefficients issus d'une même partie du son et de parties différentes en affichants plusieurs courbes sur la même figure.
   Qu'oberservez vous ?

2.2 Ecriture d'un programme d'analyse par LPC

Nous allons construire un programme d'analyse et de synthèse par LPC.

```
Srec=AnalyseSyntheseLPC(name,p)
```

La première partie de ce programme consiste à analyser fenêtre par fenêtre, les coeffficients LPC et l'excitation. Nous allons modéliser l'excitation par un train de Dirac ou un bruit blanc selon que le son sera voisé ou non. Ainsi pour une fenêtre, on doit déterminer
- Les coefficients LPC,
- La nature de l'excitation, voisée ou non,
- Le gain, c'est à dire l'amplitude de l'excitation ou du son,
- Si le signal est voisé, le pitch ou fréquence d'excitation.

Nous allons découper cette analyse en plusieurs fonctions qui agissent sur une fenêtre du signal et qui constitueront la fonction AnalyseSyntheseLPC.

3. Rédigez une fonction

   ```
   E=CalculExcitation(Z,coefs,N,p)
   ```

   qui prend pour entrée un signal fenêtré Z de taille N, un jeu de p coefficients LPC $coefs$ et qui renvoie le vecteur excitation associé au filtre. Cette excitation correspond à la différence entre la prédiction linéaire et le vecteur réel. L'excitation est ainsi nulle pour les p premiers coefficients.

4. En utilisant la fonction *xcorr* de matlab qui calcule l'autocorrelation d'un signal, rédigez une fonction

   ```
   v=Voisement(cor);
   ```

   qui renvoie $v = 1$ si la fenêtre est voisé et $v = 0$ si elle ne l'est pas. Sélectionner préalablement deux portions du signal, l'une voisée, l'autre non-voisée et visualisez les autocorrelations associées pour comprendre comment déterminer si le son est voisé ou non.

5. On pourra dans un premier temps estimer le gain (l'amplitude de l'excitation) par la commande

   ```
   gain(i)=sqrt(var(E));
   ```

   où i représente ici l'indice de la fenêtre.

6. Rédigez une fonction

   ```
   pitch(i)=EstimationPitch(cor);
   ```

   qui estime le pitch ou fréquence d'excitation associée à la fenêtre.

7. Construire une fonction

   ```
   [vois,pitch]=CorrectionVoisement(vois,pitch,nfenetres);
   ```

   qui corrige le vecteur vois et le vecteur pitch en utilisant les valeurs obtenues pour les fenêtres précédentes. En effet, il se peut que le voisement ou la fréquence soit difficile à estimer localement mais qu'ils puissent être estimés a posteriori par le contexte. Cette étape de à correction à partir des fenêtre voisine est souvent nécessaire et permet de corriger à bon nombres d'erreurs.

2.3 Synthèse

Maintenant qu'on a estimé les coeffcients LPC et la nature de l'excitation, on peut effectuer une resynthèse du son à partir de ces paramètres. La reconstruction s'effectuera aussi par un recouvrement de deux fenêtres, en utilisant une fenêtre de Hanning (cosinus carré) afin de conserver l'énergie. Si veut effectuer une modification du son, on peut maintenant agir sur

- Les coefficients LPC si on veut modifier le locuteur ou ce qu'il prononce.
- La longueur des fenêtres si on veut modifier la durée du son.
- Le pitch si on veut modifier la hauteur du son.

8. Rédiger une fonction

   ```
   E=ConstructionExcitation(gain,pitch,vois,taillefenetre)
   ```

   qui reconstruit une excitation en fonction d'un gain, d'un indicateur de voisement "vois", d'un pitch et d'une taille de fenêtre.
   Je conseille de faire la reconstruction par demi-fenêtre où l'on calculera un pitch moyen et un gain qui peut évoluer de manière continue si les deux fenêtres associées à la demi fenêtre sont voisées . Si une des fenêtres est non voisé, il faut faire attention ...

9. Rédigez une fonction

   ```
   Srec=SyntheseLPC(E,Coefs,taillefenetre,p);
   ```

   qui reconstruit un son, demi fenêtre par demi fenêtre à partir de l'excitation E et des coeffcients LPC Coefs .

10. Utilisez les fonctions précédentes pour construire le programe AnalyseSynthèseLPC.
11. Ajouter une fonctionnalité à AnalyseSynthèseLPC qui permet de modifier la longueur du son en jouant sur la taille des fenêtres de reconstruction.
12. Testez le programme sur différents sons.

3 Reconnaisance Vocale

L'objet de cette section est de voir comment les coeffcients LPC peuvent caractériser un son et un locuteur. Le modèle LPC est assez rudimentaire mais permet quand même de reconnaitre des sons voyelles les uns des autres et de distinguer un locuteur d'un autre. Nous allons réutiliser la fonction CalculCoefsLPC pour le calcul des coefficients.
Pour effectuer la reconnaissance, on propose de générer un dictionnaire de coefficients LPC associé à un son et à une personne. Ces coeffcients vont être calculés à partir d'un échantilon de voix et stockés dans un tableau. Pour reconnaitre le son ou le locuteur, on comparera les coefficients estimés du son que l'on étudie au dictionnaire.
Extraire les coefficients est aisé mais effectuer la comparaison est plus problématique. Deux questions se posent :

- Comment estimer une distance entre deux jeux de coefficients ?
- A partir des distances aux éléments du dictionnaire, comment déterminer le son le plus proche ?

En effet, pour plus de robustesse, on associera plusieurs jeux de coefficients, par exemple 10 ou 20 pour un son donné. Il exite des moyens plus robustes que d'identifier un son en considérant le jeux de coefficients LPC le plus proche de celui observé.
Nous proposons une certaine distance et une approche par "plus proches voisins", mais il existe mille autres manières de procéder. La distance que nous proposons repose sur la

distance entre les pics de fréquences des filtres. En effet, les coeffcients LPC des filtres ne sont pas significatifs, ce sont les fréquences que ces filtres amplifient ou aténuent et la manière dont ils les modifient qui sont signifinants. Nous proposons donc d'utiliser de calculer les positions et l'amplitude des pics de fréquences associés aux filtres LPC.

13. Rédigez une fonction

    `[Positions,Values]=ParametresFiltres(Coef,N,n)`

    qui prend pour entrée un jeux de coefficients de filtre LPC, une taille N de fenêtre et un nombre n de pics et qui renvoie dans les vecteurs Positions et Values les positions et les amplitudes des n premiers pics (au sens des pics de basses fréquence).

14. Rédigez une fonction qui calcule la distance entre un jeu de paramètres de position P1 et d'amplitude de pics V1 et un jeu de référence appartenant à un dictionnaire.

    `d=ComparaisonCoefs(P1,V1,P2,V2)`

    Nous proposons de calculer pour chacun des pics de (P1,V1) de rechercher le pic de (P2,V2) qui est le plus proche en norme 1 et de sommer les distances des 4 pics les plus proches de ceux du jeu (P2,V2). Il arrive que des pics apparaissant dans les éléments du dictionnaire ne soient pas présents dans le son, il faut éviter des erreurs d'appariement dû à l'absence d'un pic. Tout autre solution qui marche est la bienvenue.

15. Rédigez une fonction qui va stocker dans un tableau le dictionnaire des coefficients LPC associés aux différents sons. Ce tableau peut être construit à partir de différents fichier .wav ou d'un seul en utilisant la fonction CalculCoefsLPC.

    `[Pos,Val]=Dictionnaire(name1,name2,...)`

    Dans Pos on stoke différentes positions de pics et dans Val les amplitudes associées. On pourra commencer par 10 par sons par exemple. Donc si on considère 4 sons et une analyse sur les 5 premiers pics. Pos et Val auront chacun $4\times10=40$ lignes et 5 colonnes.

16. Rédigez une fonction qui estime le son de la fenêtre d'étude en comparaison avec le dictionnaire.

    `Son=DeterminationSon(Pos1,Val1,Pos2,Val2,nb)`

    où Pos1 et Val1 correspondent au son étudié et Pos2 et Val2 sont les tableaux obtenus par la fonction dictionnaire. nb rappelle le nombre de sons étudiés contenus dans le dictionnaire.

17. Créez ensuite une fonction

    `Son=EstimationSon(name,Pos2,Val2)`

    qui estime le son dans chaque fenêtre.

18. On terminera par insérer dans la fonction précédente un algorithme de correction en utilisant les fenêtre avoisinante. Comme précédemment on peut considérer que les sons de fenêtres voisines sont liés et on peut donc essayer de corriger une approche brutale en exploitant les informations apportées par le contexte.

    `Soncorrige=CorrectionSon(Son)`