Estimating the phase of Spectral flow signals

Leonardo Lancia

Laboratoire Parole et Langage

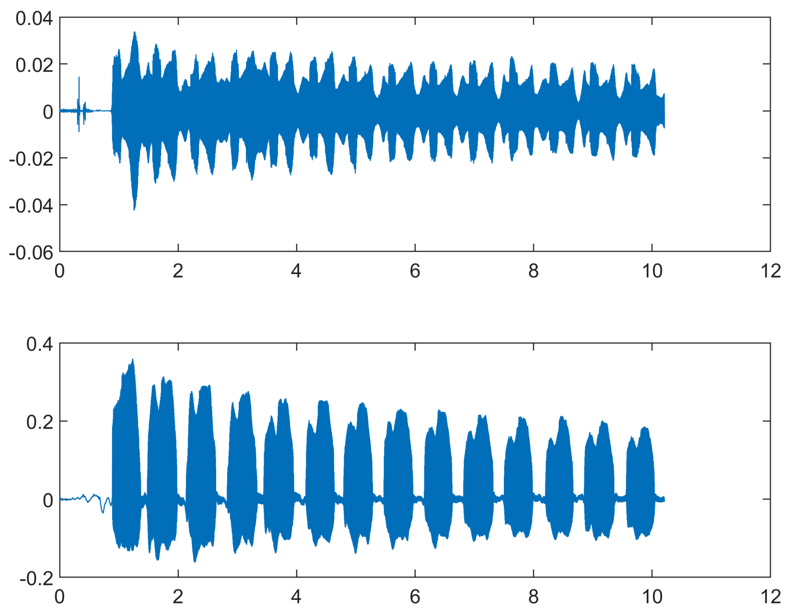
CNRS / Aix Marseille Univesité

leonardo.lancia@cnrs.fr

The data we are going to load and plot represent the repeated production of the phrase 'ma mine' by a female speaker of french. The first channel contains the audio, whereas the second the EGG signal.

```
clear all
pathIn='./data/mamine_cont_2.wav';
addpath(genpath(['./utils']));
addpath('./get_phase');
[x,sr]=audioread(pathIn);

figure;
ff(1)=subplot(211);
plot([1:length(x)]./sr,x(:,1))
ff(2)=subplot(212);
plot([1:length(x)]./sr,x(:,2))
```



Spectral flow computation

We will work on the acoustic waveform (first channel) that we submit to the computation of the spectral flow. For that we use L. Goldstein function audmod.m (downlowaded here: https://sail.usc.edu/~lgoldste/ArtPhon/Code/audmod.zip). The function has been modified in two ways:

1) it can take as first argument also a cell structure containing a waveform and its sampling rate;

2) the computations of statistics and plottings can be skipped by setting the last argument to 1.

The first argument of the function is the input signal, this can be a path to a file or a cell structure containing a waveform in the first cell and its sampling rate in the second cell;

the second argument (**MFCCrate**) is the sampling rate of the computed modulation function;

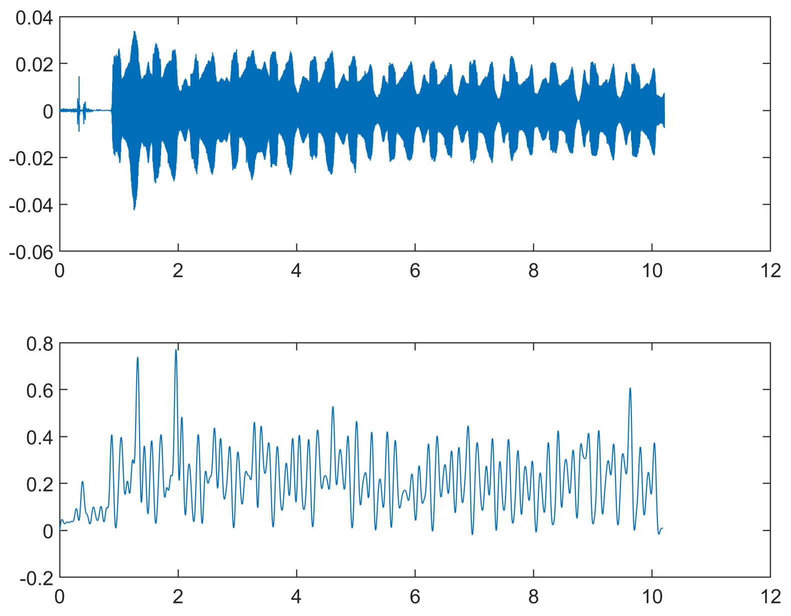
the third argument (**cutOff**) is the cut-off frequency of the low pass filters applied to the MFCCs and to their summed derivative;

the last argument (**full**) makes the function compute statistics and plot the result if different from 0.

```
% set parameters
MFCCrate=1000;
cutOff=12;
full=0;

modF=audmod({x(:,1),sr}, MFCCrate,cutOff,full);
modT=[1:length(modF)]./MFCCrate;

figure;
bb(1)=subplot(211);
plot([1:length(x)]./sr,x(:,1))
bb(2)=subplot(212);
plot(modT,modF)
linkaxes(bb,'x')
```



Just a couple of considerations on the signal obtained. Taking the difference between successive MFFC values is equivalent to applying a high pass filter to the data. In the computation of the modulation function the summed MFCCs differences are then low pass filterd. Since we apply both a low pass and a high pass filters, we obtain a quite narrow-banded signal (altough centred on a value different from zero). In principle, centring the cycles around zero and normalizing their amplitude should permit a principled computation of the phase. This correspond to the operations conducted in the first part of the algorithm described in the companion paper.

Phase extraction

The next step consists in running the algorithm for phase extraction.

The first two arguments are the signal and its sampling rate.

The third argument (**m**) is the length of the Savitzsky-Golay differentiator filter, while the fourth (**n**) is its order (chunks of signals of length m are approximated by polynomials of order n).

The fifth argument (**nMasks**) is the number of masking signals used in MAsked sifting.

the sixth argument (**ampCoeff**) is the coefficient that determine the amplitude of the mask signals as a proportion of a rough estimate of the signal range (which assuming that the signal is normally distributed should be given by $4 \times \text{std}(r)$).

The seventh argument (**phaseMethod**) is a cell containing two strings indicating the method to compute the phase in the first and in the second part of the algorithm.

The last argument (**threshs**) is a vector containing the values of the threshold used in the applications of refined amplitude normalization in the two parts of the algorithm. leaving them to NaN selects the default value of 5×10^{-10}`for both applications.

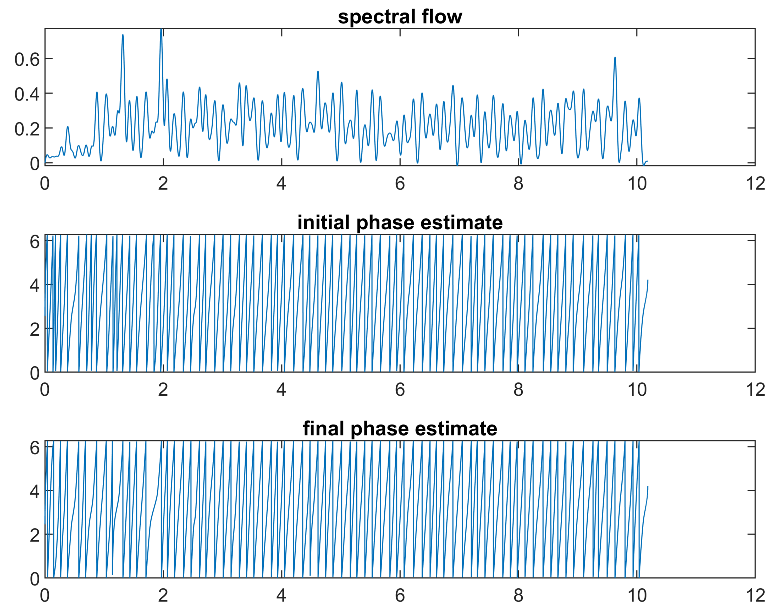
```
%set parameters

m=16;
n=5;
nMasks=22;
phaseMethod={'h','h'};
threshs=[NaN,NaN];
ampCoeff=1.2;

[PHI,siftedSig,PHI0,centeredSig,mask]=getPHImask(modF,MFCCrate,m,n,nMasks,ampCoeff,phaseMethod,threshs);
```

The function outputs the computed phase signal (PHI); the output of masked sifting applied to the input signal (siftedSig); the phase computed from the input signal submitted to one iteration of sifting and to refined amplitude normalization (PHI0); the centred input signal (centeredSig); and the masking signal used for masked sifting (mask).

```
figure;
bb(1)=subplot(311);
plot(modT,modF)
title('spectral flow')
bb(2)=subplot(312);
plot(modT,PHI0)
title('initial phase estimate')
bb(3)=subplot(313);
plot(modT,PHI)
title('final phase estimate')
linkaxes(bb,'x')
```

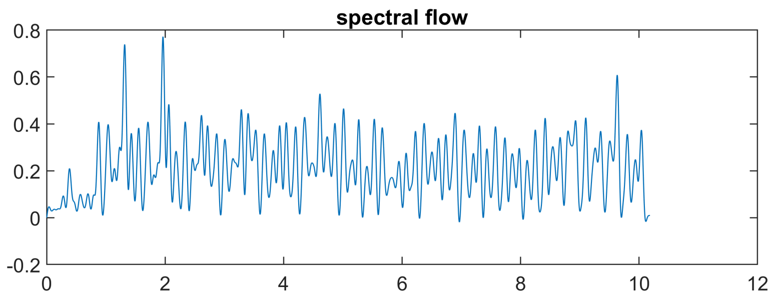


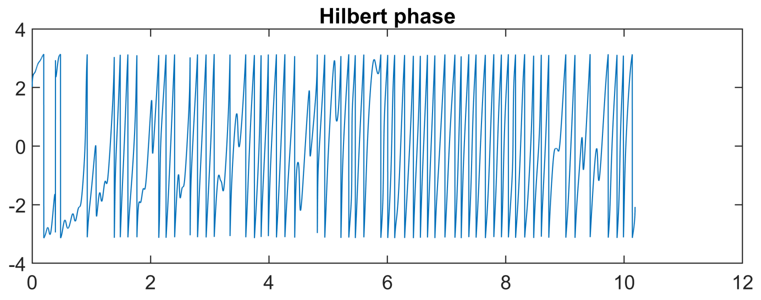
As observed above, in the case of such a relatively narrow banded signal, both phase estimates make a relatively good job. The two estimates diverge in cases of small peaks or elbows of clear peaks.

Failure of simpler approaches

Note that applying the Hilbert transform to the modulation function gives poor results

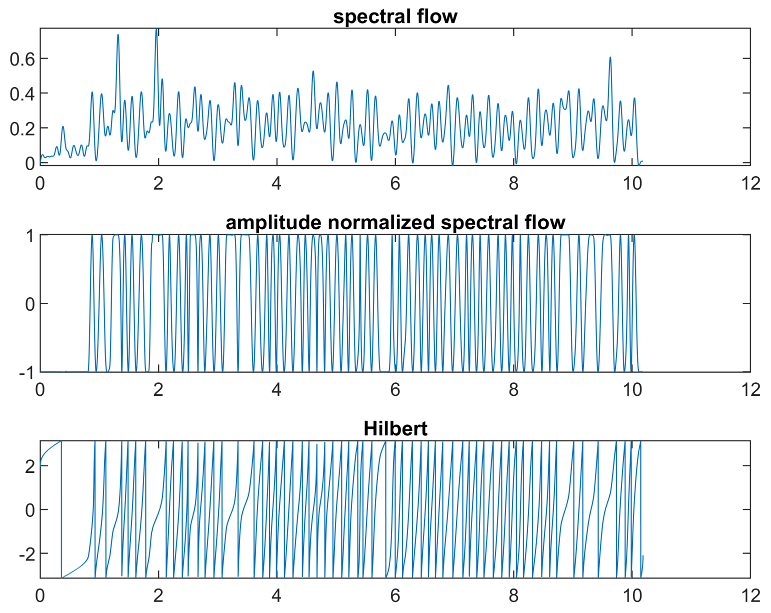
```
figure;
bb(1)=subplot(211);
plot(modT,modF)
title('spectral flow')
bb(2)=subplot(212);
hPhase=angle(hilbert(zscore(modF)));
plot(modT,hPhase)
title('Hilbert phase')
```





Even when the modulation function is submitted to refined amplitude notrmalization results are less robust than those obtained with our algorithm. This happens because the amplitude normalization operations tends to discard small cycles that are not centered on zero.

```
figure;
bb(1)=subplot(311);
plot(modT,modF)
title('spectral flow')
bb(2)=subplot(312);
normSig=normalize_cycle_amp(zscore(modF)');%normalize cycle's amplitudes
plot(modT,normSig)
title('amplitude normalized spectral flow')
bb(3)=subplot(313);
hPhase=angle(hilbert(normalize_cycle_amp(zscore(modF)')));%get hilbert phase
plot(modT,hPhase)
title('Hilbert')
```


spectral flow
0.6
0.4
0.2
0
0
2
4
6
8
10
12
amplitude normalized spectral flow
1
0
-1
0
2
4
6
8
10
12
Hilbert
2
0
-2
0
2
4
6
8
10
12