Phase analysis of behaviuoral data

Leonardo Lancia

Laboratoire Parole et Langage

CNRS / Aix Marseille Univesité

leonardo.lancia@cnrs.fr

This tutorial is divided in two parts: **Extracting the phase** and Denoising algorithm be warned that the Denoising algorithm may be slow depending on the machine. If so, and if you want to check if it works, reduce the number of randomization by reducing the parameter **nRandNoiseRep** (line 43).

Ectracting the phase

```
%just adding the subdirs containing used functions to the path and shutting
%off warnings
clear all
addpath(genpath(['./utils']));
addpath('./get_phase');
warning('off')
```

Let's load the data, the variable sigIn will contain a matrix whose columns contain articulator positions on different dimensions (see dataIn.descriptor) sampled at dataIn.samplerate Hz. We can also build a vector with the time stamps to use in plots.

```
pathIn='.\data\cl_pata_2.mat';
dataIn=load(pathIn); %load data
sigIn=dataIn.data(1:end,:);
sr=dataIn.samplerate;% sampling rate
sigT=[1:length(sigIn)]./sr;% time steps
```

We will use TTIP heigth (column 16, 'lang-aY' in dataIn.descriptor) which we willz-score after application of a low pass FIR filter with cut-off at 20 Hz and filter order equal to 1000.

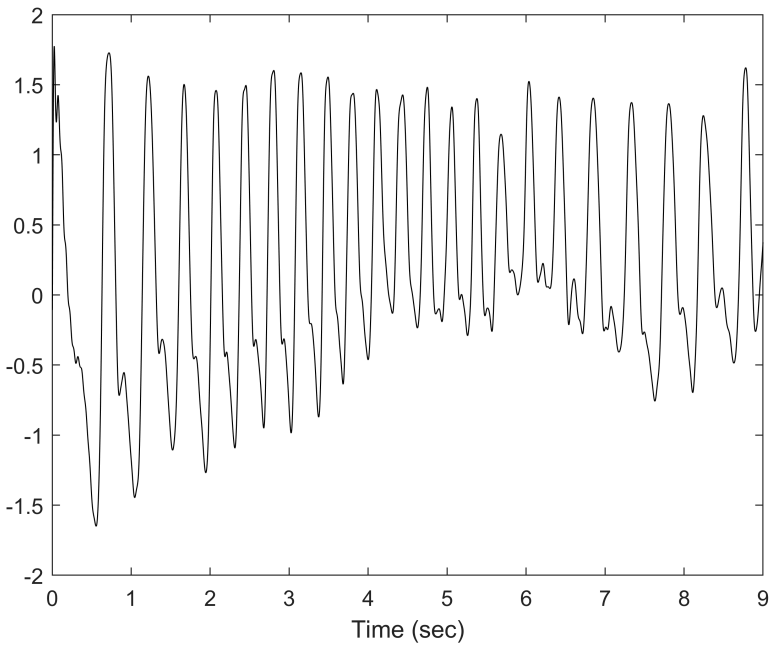
```
artDimN=16; %select TTIP (cf. strmatch('lang-aY',dataIn.descriptor,'exact') )

%low pass filter parameters
hiCut=20; %Max allowed frequency
filtLen=1000;%filter order
% %%here we lowpass filter the signal

hiCutNorm=hiCut./( sr/2 ); %normalized cut off for the low-pass filter
b1=fir1(filtLen,hiCutNorm,'low'); %filter coeffs
filtSig=filter(b1,1,sigIn(:,artDimN)); %apply filter to input sig.
filtSig=zscore(filtSig);%at least now it oscillates around 0 like a real sinusoid.
myData=filtSig(filtLen/2+1:end); %correct filter delay
```

```
myDataT=sigT(1:length(myData)); %remove exceding samples' time points

figure;
plot(myDataT,myData,'color','k');
xlabel('Time (sec)')
```



We now set the parameters of our algorithm.

phaseMethod is a cell of strings, each string is a label of a different method for the extraction of the quadrature signal: 'h' stands for Hilbert and 'q' for direct quadrature extraction. The first label is used to define the method used in the first part of the algorithm. The second label is used in the second part of the algorithm.

```
phaseMethod={'h','h'};
```

The thershold ε (used int the normalization of the cycles' amplitudes) must be a vector with two values, one for each part of the algorithm. Setting them to NaN, makes use of the default value of 1e-10

```
threshs=[5e-10,5e-10];% equivalent to threshs=[NaN,NaN]
```

m, and **n** are the length and the order of the Savitsky Golay differentiator

```
m = 16;      % number of filtered points (FIR order)
n = 5;      % approximation polynomial order
```

nMasks indicates the number of mask signals used in Masked Sifting

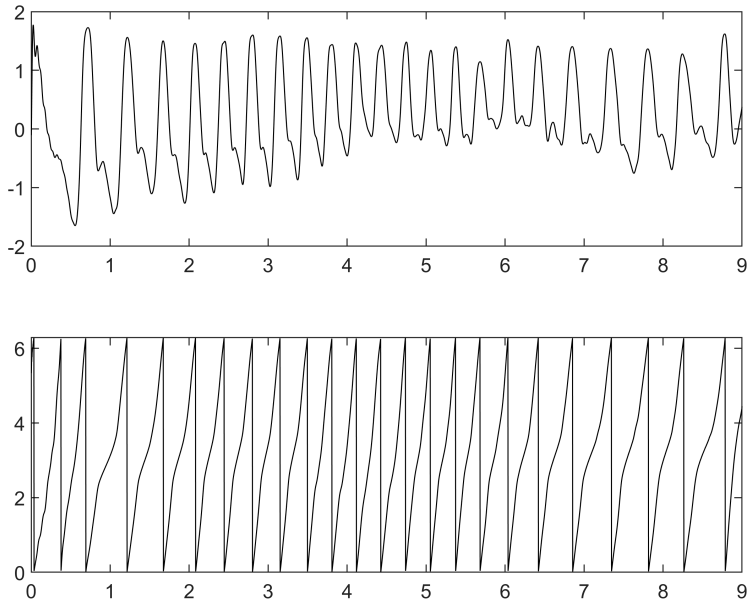
```
nMasks=22;% number of mask signal for Sifting
```

ampCoeff determines the amplitude of the masks as a proportion of $4 \times \mathrm{std}(r)$ where *r* is the signal submitted to masked sifting. The value $4 \times \mathrm{std}(r)$ represents a rough approximation of the range of a normal distribution given its standard deviation.

```
ampCoeff=2;% amplitude coefficient for masked sifting
```

The algorithm can now be launched. The first output is the final estimation of the phase, the second output is the signal obtained from masked sifting; the third output is the phase estimated from the centered signal in the first part of the algorithm; the fourth output is the centered signal used to estimate phase and frequency in the first part and the fifth output is the masking signal used in masked sifting.

```
[PHI,siftedSig,PHI0,centeredSig,mask]=getPHImask(myData,sr,m,n,nMasks,ampCoeff,phaseMethod,threshs);
figure;
pl(1)=subplot(211);
plot(myDataT,myData(:,:),'color','k');
pl(2)=subplot(212);
plot(myDataT,PHI,'color','k');
linkaxes(pl,'x')
```



Denoising algorithm

We may want to process our data in a more adaptive way, which is without setting a low-pass cut-off frequency for the filter.

In this case we can use the function mEMDdeNoise.m

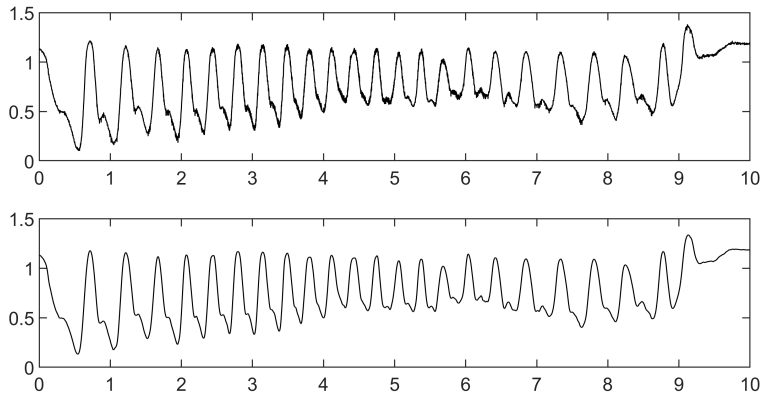
for that we need to define the value of the parameter **nRandNoiseRep** determining the number of simulated random processes and the parameter **alphaVal**, determining the significance threshold.

The first output of the function represents the filtered signal, the second output represents the IMFs, the fourth output corresponds to their their frequencies, the fifth output represents the indexes of the IMFs that added produce the filtered signal and the last output represents the standard deviation of the estimated random component.

```
% set additional parameters required by denoising
nRandNoiseRep=100;
alphaVal=0.05;
%apply denoising
[filtereds,imf,imfF,filteredidx,noiseStd]=mEMDdeNoise(sigIn(:,artDimN),sr,nMasks,ampCoeff,alphaVal,nRand
```

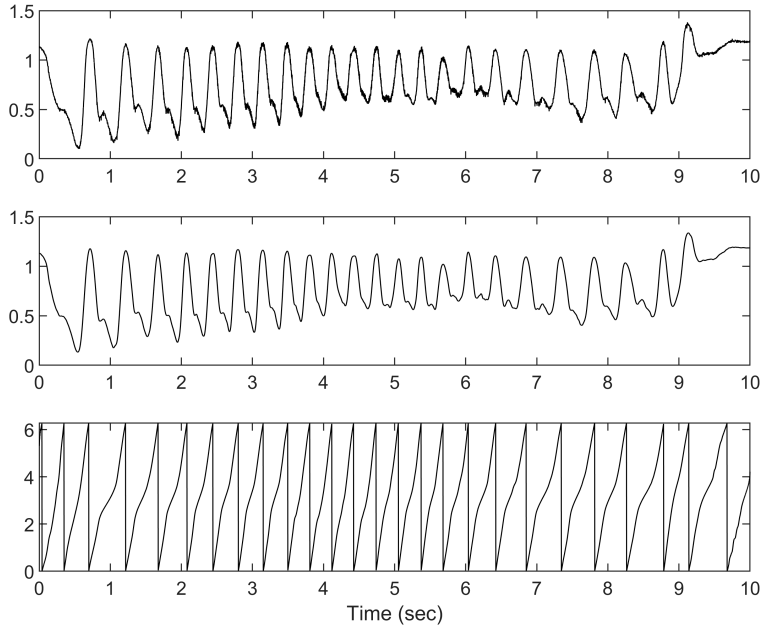
We can plot the clean signal

```
figure
pll(1)=subplot(311);
plot(sigT,sigIn(:,artDimN),'k')
pll(2)=subplot(312);
plot(sigT,filtereds,'k')
linkaxes(pll,'x')
```




```
[PHI,~,~,~,~]=getPHImask(filtereds,sr,m,n,nMasks,ampCoeff,phaseMethod,threshs);

pll(3)=subplot(313);
plot(sigT,PHI,'k')
linkaxes(pll,'x')
xlabel('Time (sec)')
```


1.5
1
0.5
0
0 1 2 3 4 5 6 7 8 9 10
1.5
1
0.5
0
0 1 2 3 4 5 6 7 8 9 10
6
4
2
0
0 1 2 3 4 5 6 7 8 9 10
Time (sec)