EE 301

Term Project (TP18)

"Structural Modeling of Pinna related Transfer Function"

Submitted by :

Aditya Jain (11037), Anirudh Agrawal (11098), Ishendra Agarwal (11324), Sohum Datta (11724)

Department of Electrical Engineering
IIT Kanpur

JAN-APR 2014

Table of Contents

1		**Introduction**	**3**
2		**Some Fundamental Concepts in PRTF analysis**	**4**
	2.1	Pre Processing	4
	2.2	Resonances	4
	2.3	Notches	5
	2.4	Reflections and Anthropometry	5
3		**A Structural Model of Pinna**	**6**
	3.1	Resonance Block	6
	3.2	Reflection Block	7
	3.3	Stages of Notch and Resonance Analysis	7
	3.3.1	Notch Extraction	8
	3.3.2	Resonance Extraction	12
4		**Results**	**14**
5		**Conclusion**	**25**
6		**Future Scope**	**25**
7		**Code Listing**	**26**
	7.1	script_tp18.m	26
	7.2	Algorithm_1.m	31
	7.3	Algorithm_2.m	40
	7.4	lmin.m	48
	7.5	get_CIPIC_HRIR.m	50
	7.6	get_CIPIC_HRIR_onset.m	53
	7.7	find_bandwidth.m	55
	7.8	refine_notch.m	57
	7.9	refine_resonance.m	60
	7.10	CIPIC_database_path.m	63
	7.11	Readme.txt	64

1. INTRODUCTION

There exists a number of ways to render HRTF-based spatial audio. Approximations based on low-order rational functions and series expansions of HRTFs were proposed, resulting in simple yet valuable tools for HRTF modeling. On the other hand the complexity of filter coefficients and weights, respectively, makes both techniques unsuitable for real-time applications. Conversely, structural modeling seems nowadays to be an attractive alternative approach for all those scenarios where fast computation is needed: within this characterization, the contribution of the listener's head, ears and torso to the HRTF are isolated in several subcomponents, each accounting for some well-defined physical phenomenon.

Our work focuses on the contribution of the pinna to the HRTF. Although perceptually dominated by head motion cues, pinna effects on incident sound waves are of great importance in sound spatialization. Several experiments have shown that, contrarily to azimuth effects which are dominated by diffraction around the listener's head and may be reduced to simple and intuitive binaural quantities, elevation cues are basically monaural and heavily depend on the listener's pinna shape, being the result of a superposition of scattering waves influenced by a number of resonant modes inside pinna cavities. Within this framework, it is crucial to find a suitable model for representing the pinna contribution to the HRTF, whose transfer function we commonly refer to as Pinna-Related Transfer Function (PRTF). Once such model is available, cascading it to a simple Head-and-Torso (HAT) model would yield a complete structural HRTF representation.

In this paper we exploited an iterative approach which aims at separating resonance effects from pinna reflections in experimentally measured PRTFs, a method for extracting the frequencies of the most important notches is here sketched, followed by a discussion on the possible relation between notch frequencies and anthropometry. Finally, a structural model of the pinna is proposed.

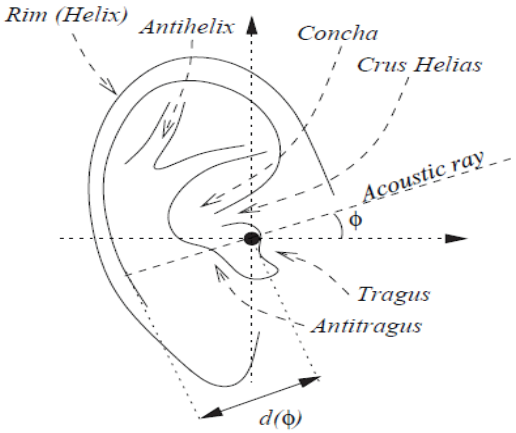


Figure 4. *Anatomy of the pinna.*

2. SOME FUNDAMENTAL CONCEPTS IN PRTF ANALYSIS

2.1 Pre processing

For purpose of analysis we focus on HRIRs sampled on the median plane, with elevation varying from −45 to 90. The pre-processing step we apply to obtain a raw estimate of the PRTF is windowing the corresponding HRIR using a 1.0 ms Hann window. In this way, spectral effects due to reflections caused by shoulders and torso are removed from the PRTF estimate. In order to isolate the spectral notches in the so built PRTFs we exploit an ad-hoc designed algorithm that returns an estimate of the separated resonant and reflective components. Once convergence is reached, say at iteration n, the PRTF spectrum bodies the resonant component alone, while a combination of the n multi-notch filters provides the reflective component.

2.2 Resonances

We can identify two resonances, centred around 4 kHz, appears to be very similar amongst subjects since it spans all elevations. The resonance's band width appears to increase with elevation. Still it is most prominent at low elevations between 12 and 18 kHz. Note that the higher resonance may be perceptually irrelevant since it lies near the upper limit of the audible range. In addition, since the resonances at 12 and 7 kHz are excited in mutually exclusive elevation ranges, we may look forward to a double-resonance filter design.

2.3 Notches

We now discuss the PRTF features identified by the decomposition carried out through the separation algorithm. We use the "notch tracking" algorithm to track the notches. The notch detection step simply locates all of the local minima in the reflective component's spectrum. Since it is preferable to restrict our attention to the frequency range where reflections due to the pinna alone are most likely seen, and ignore notches which are overall feeble, two post processing steps are performed on the obtained tracks:

- Delete the tracks which are born and die outside the range 4 − 14 kHz;
- Delete the tracks that do not present a notch deeper than 5 dB.

But certain subjects do not have notch deeper than 5db, so we took 0 as minimum notch depth

2.4 Reflections and anthropometry

We now move towards the definition of a realistic mapping between notch frequencies and reflection points over the pinna, by relating each major notch at frequency f0 to a different reflection. Reflection models typically assume that all reflection coefficients are

positive. In such case, in order for destructive interference to occur , the extra distance travelled by the reflected wave with respect to the direct wave must be equal to half a wavelength:

td = 1/2f0.

However, since the impedance of the pinna is greater than that of air, there may be a boundary created by an impedance discontinuity which could produce its own reflection and ultimately reverse the phase of the wave. In this case, a delay of half a wavelength would not produce notches in the spectrum any more. Instead, destructive interference would appear for full-wavelength delays only:

td =1/f0

We choose to use this last assumption, and relate notches to pinna geometry through a simple ray-tracing procedure similar to the one The distance of each reflection point with respect to the entrance of the ear canal is calculated through the following equation,

d = c* td /2 = c/2f0
where f0 represents the frequency of the current notch at particular elevation and c is the speed of sound .

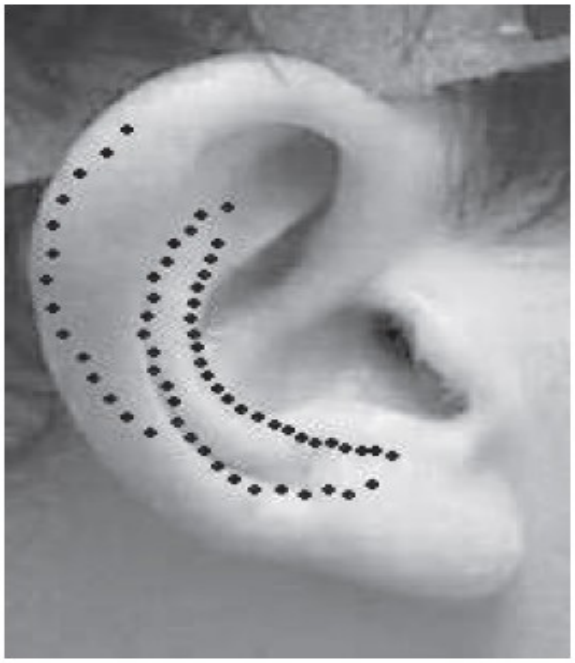


Figure 5. *Reflection points on Subject 134's right pinna.*

3. A STRUCTURAL MODEL OF THE PINNA

The information gathered from the outputs of the decomposition and notch tracking algorithms allows to model the PRTF with two resonances and three spectral notches. Our final aim is to design two distinct filter blocks, one accounting for resonances and one for reflections. Clearly, in order to reach complete control of the filter parameters, full parameterization of the model on anthropometrical measurements is needed.

3.1 Resonance Block

Approximate filter design for the resonance block can be obtained by:

- Deducing centre frequency fC
- Magnitude G of each resonance
- Fixing bandwidth fB = 5 Khz

and directly using the so found parameters to design two second-order peak filters with fixed bandwidth as shown below:

$$H_{res}(z) = \frac{V_0(1-h)(1-z^{-2})}{1+2dhz^{-1}+(2h-1)z^{-2}},$$

where

$$h = \frac{1}{1+\tan(\pi\frac{f_B}{f_s})},$$

$$d = -\cos(2\pi\frac{f_C}{f_s}),$$

$$V_0 = 10^{\frac{G}{20}},$$

A posteriori analysis of the synthesized resonances has revealed that PRTFs for high elevations only need the first resonance to be synthesized, being the second very close to it. We thus choose to bypass the second resonant filter when $\phi \geq 20$.

3.2 Reflection Block

Similarly for what concerns the reflection block, we feed:

- Centre frequency fC
- Notch Depth G
- Bandwidth fB coming from the notch tracking algorithm to second order notch filters of the form.

$$H_{refl}(z) = \frac{1+(1+k)\frac{H_0}{2}+d(1-k)z^{-1}+(-k-(1+k)\frac{H_0}{2})z^{-2}}{1+d(1-k)z^{-1}-kz^{-2}},$$

where the parameters are calculated as follows :

$$V_0 = 10^{-\frac{G}{20}},$$

$$H_0 = V_0 - 1,$$

$$k = \frac{\tan(\pi\frac{f_B}{f_s}) - V_0}{\tan(\pi\frac{f_B}{f_s}) + V_0},$$

After calculating both the resonating and notch filters we cascade them to make the overall filter according to the following block diagram:

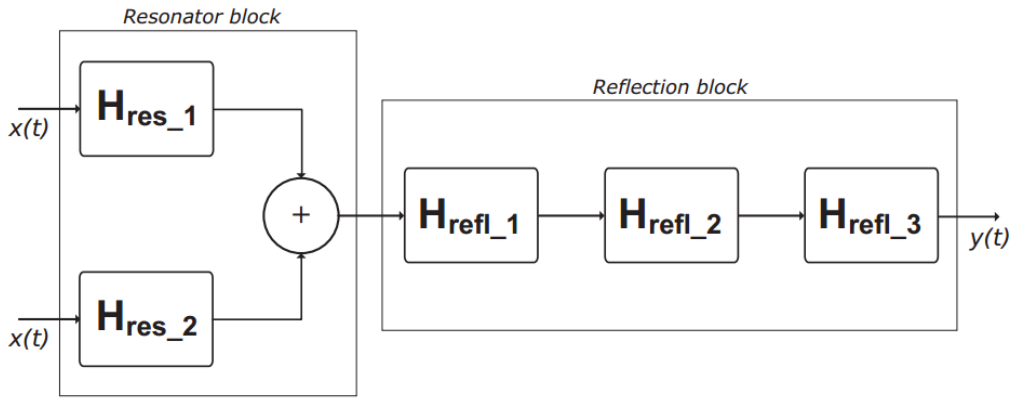


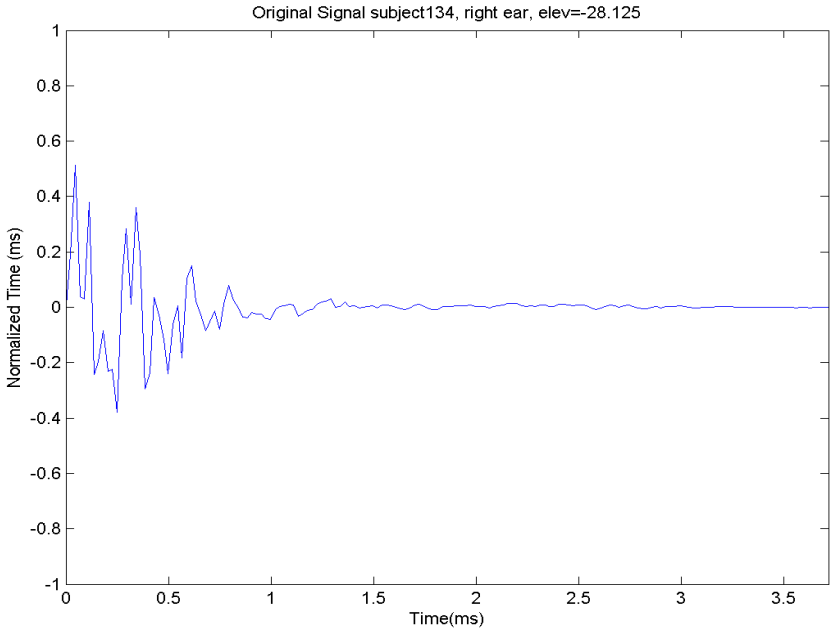
Figure 6. *General model for the reconstruction of PRTFs.*

3.3 Stages of Extraction of Notches and Resonances

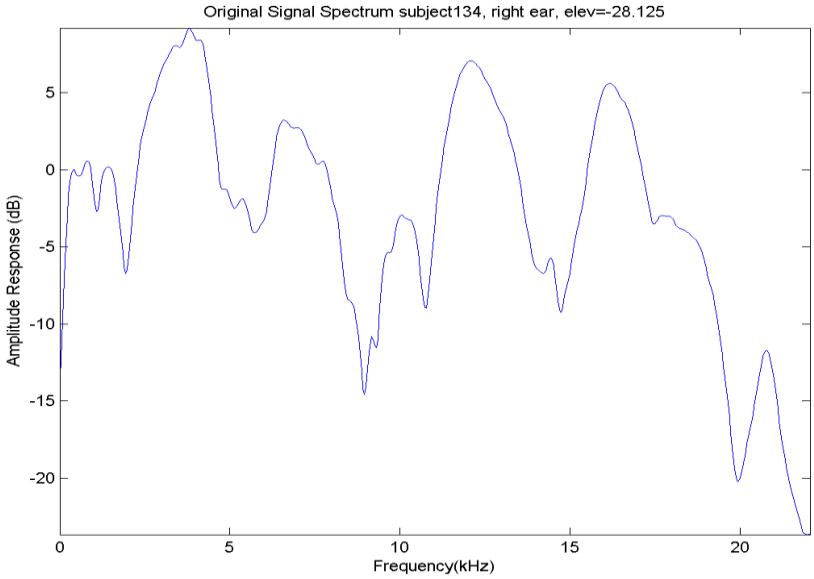
Here we demonstrate the different stages of extraction of notches and resonances. We have chosen Subject 134 Right Ear, elevation -28.125 as an example for our demonstration.

NOTCH EXTRACTION

Original HRIR

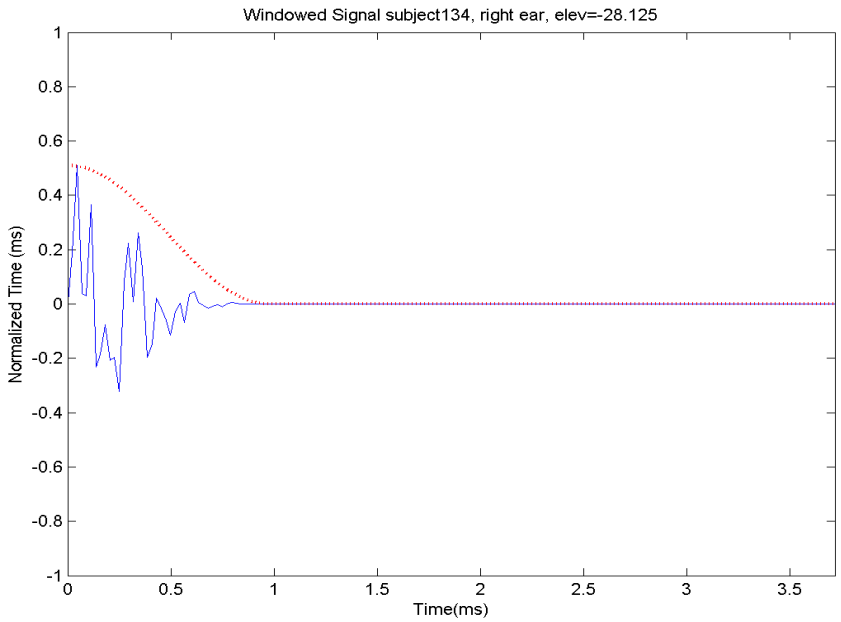


Original HRTF

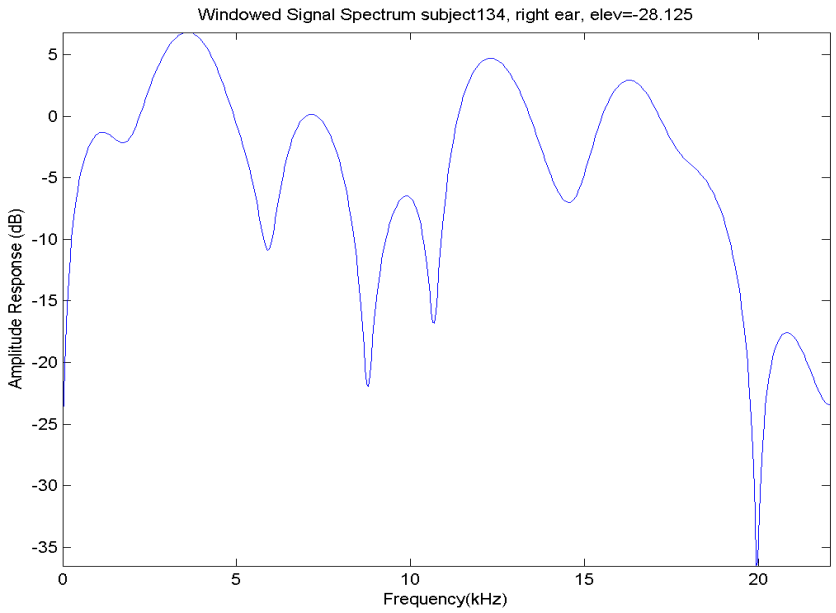


Notice the presence of various disturbances in HRTF. The HRIR needs to be windowed (i.e. remove Onset time and apply Hann Window) to get PRTF.

Original PRIR

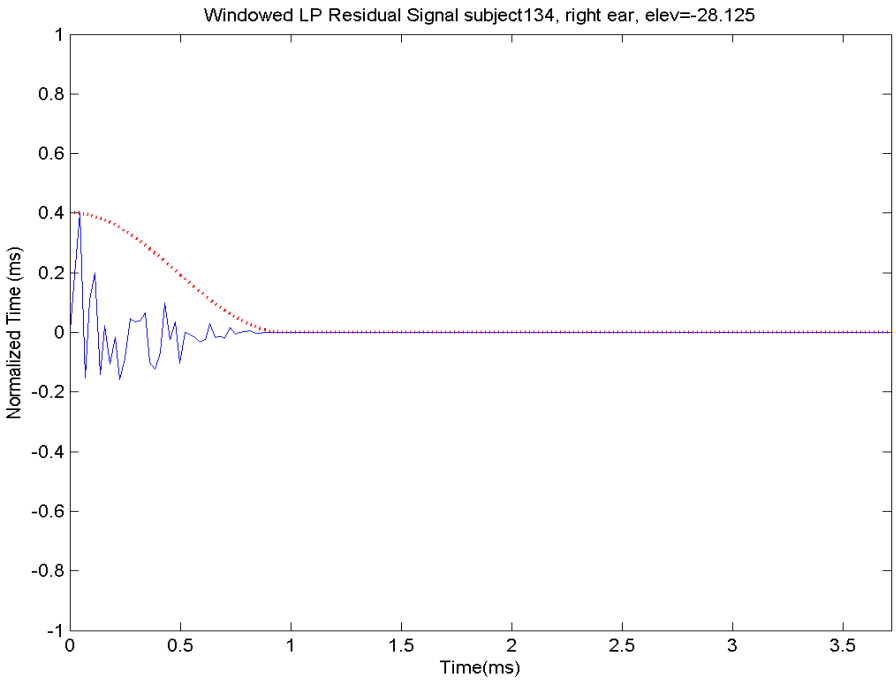


Original PRTF

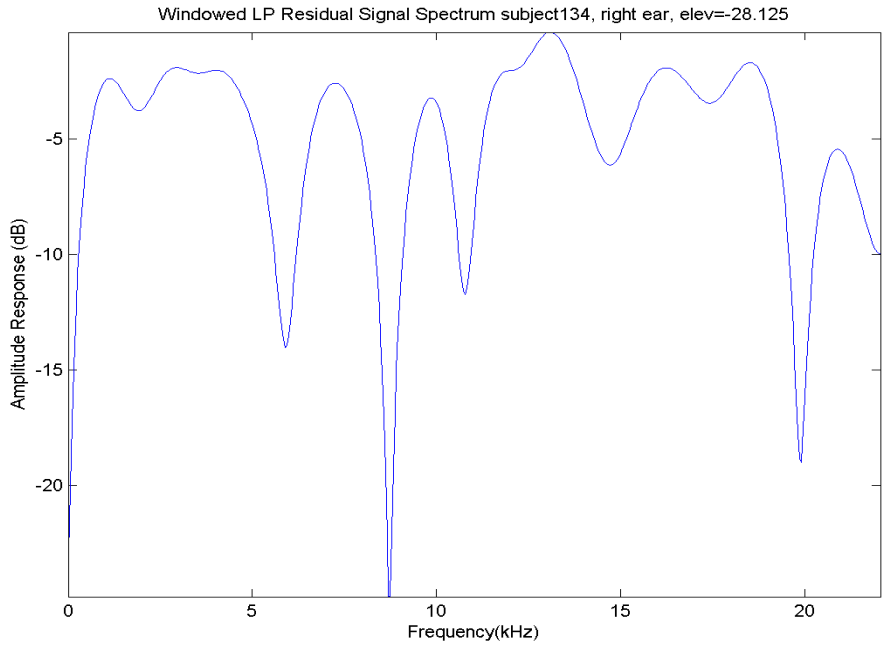


The Hann Window of 1ms is taken because if we consider longer than that then responses due to shoulders, knees and torso will show up.

LP Residual of PRIR

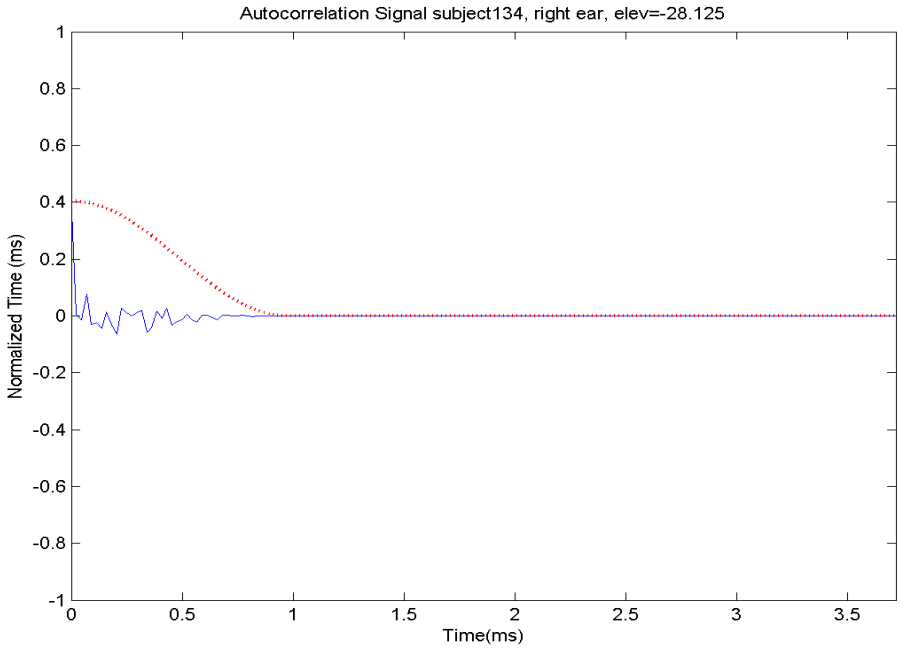


LP Residual of PRTF

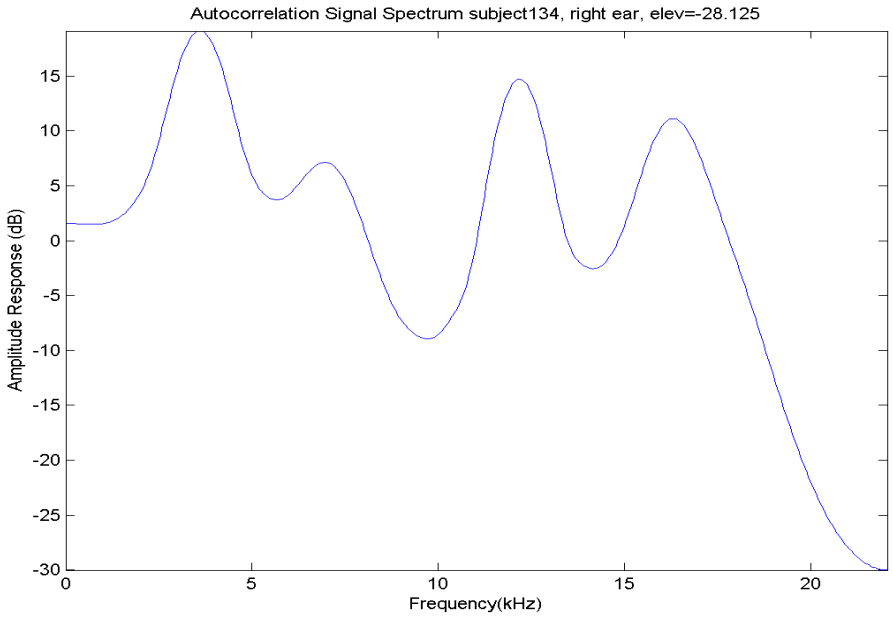


Passing through Linear Predictive(LP) residual filter removes the poles of PRTF (by approximately multiplying the PRTF with its denominator). Hence, resonance is absent.

Autocorrelation of LP residual

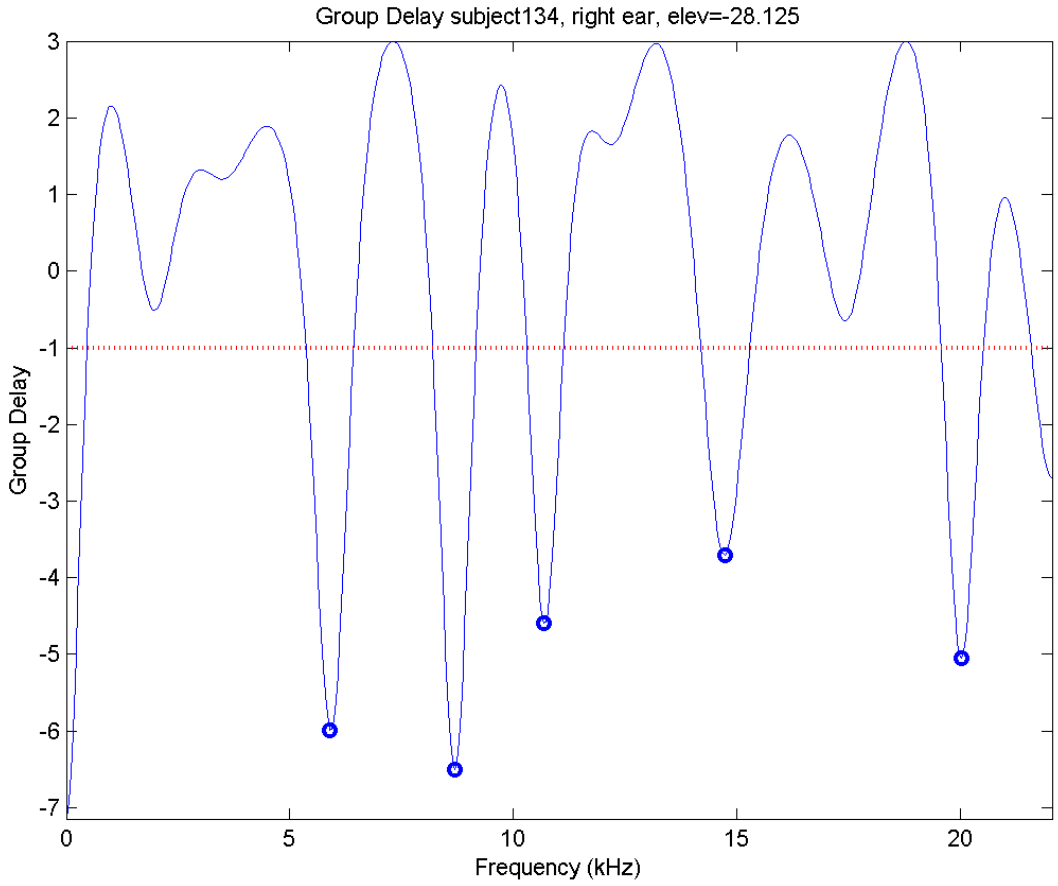


Spectrum of Autocorrelation of LP residual



The autocorrelation function of the PRIR is evaluated. This is done to make the notches sharper so that we can predict the notch frequencies accurately.

Group Delay Plot

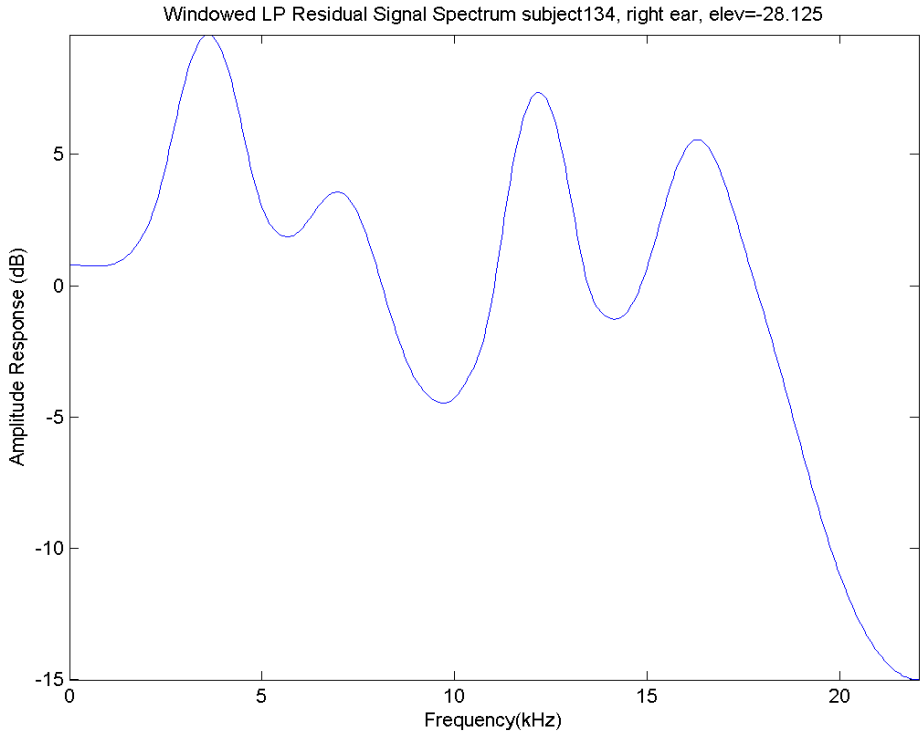


Group Delay plots separates two closely-occurring notches. Without Group delay, such notches will coalesce and extracting their central frequency will be very difficult. The frequencies where local minima occur are notch frequencies.

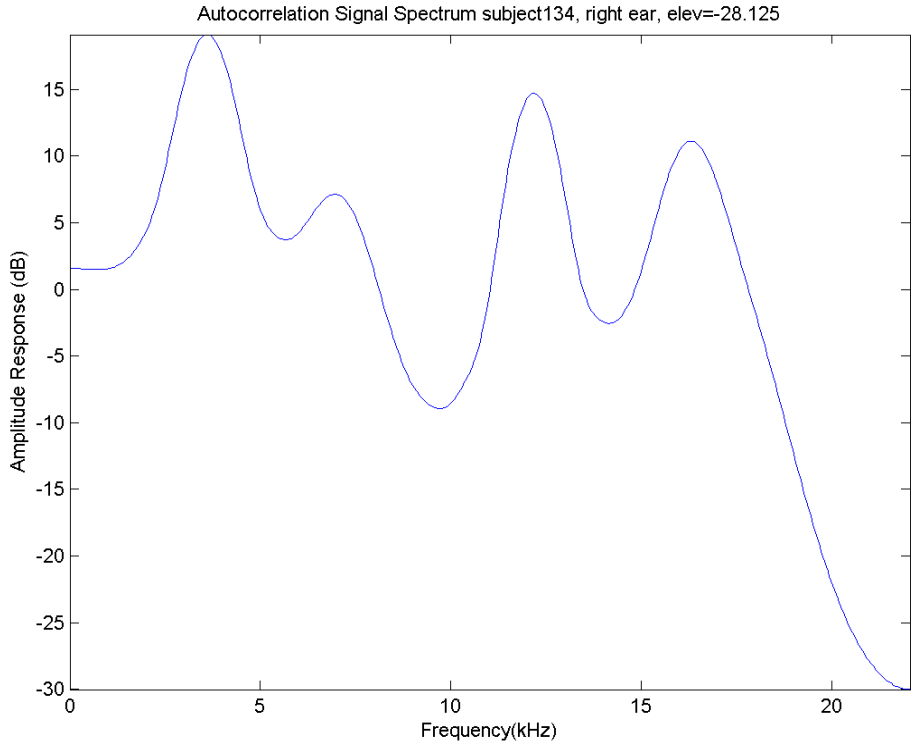
RESONANCE EXTRACTION

To extract the resonance frequencies, we obtain the coefficients of the LP residual of the PRIR. These coefficients form the transfer function which is approximately the denominator of the PRTF (i.e. resonances). To remove notches, therefore, we just obtain the impulse response of the reciprocal of the filter made from LP residual coefficients. The usual process of obtaining autocorrelation, local maxima in the group delay response, etc is carried out. To avoid repetition, only frequency response of LP residual and group delay is shown.

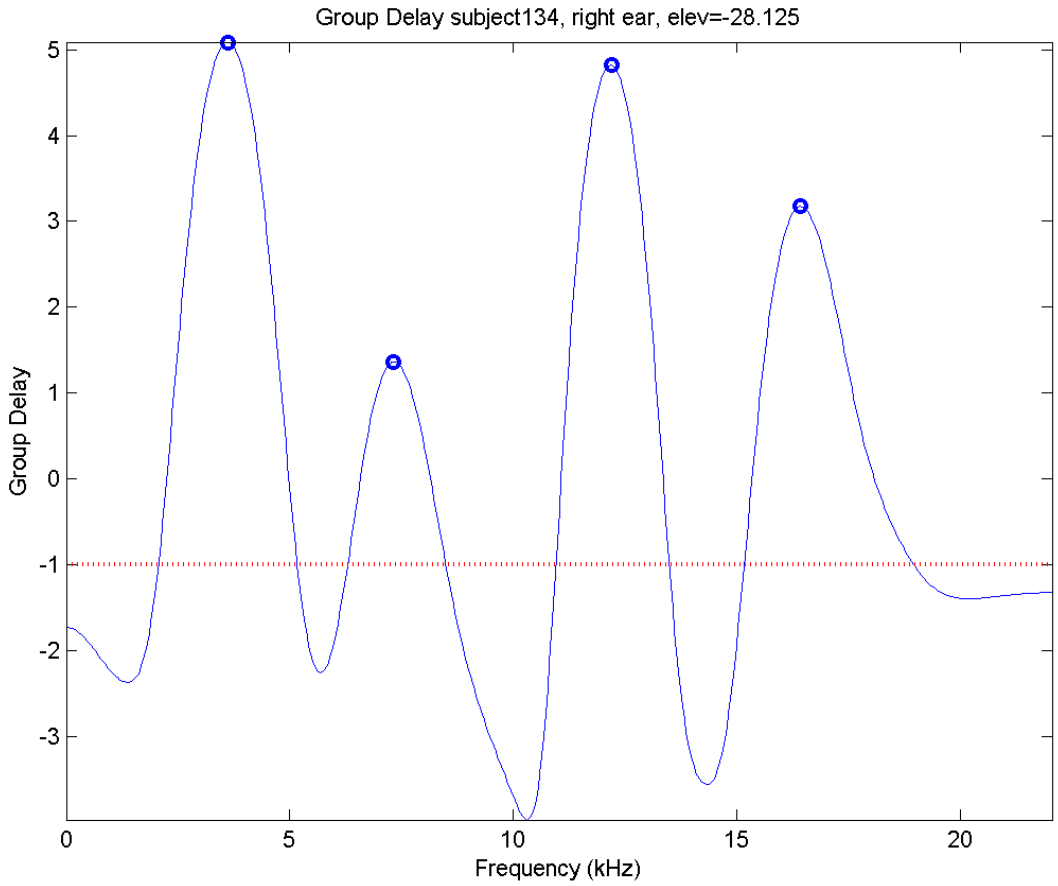
LP Residual of PRTF Resonance Plot



Spectrum of Autocorrelation of LP residual



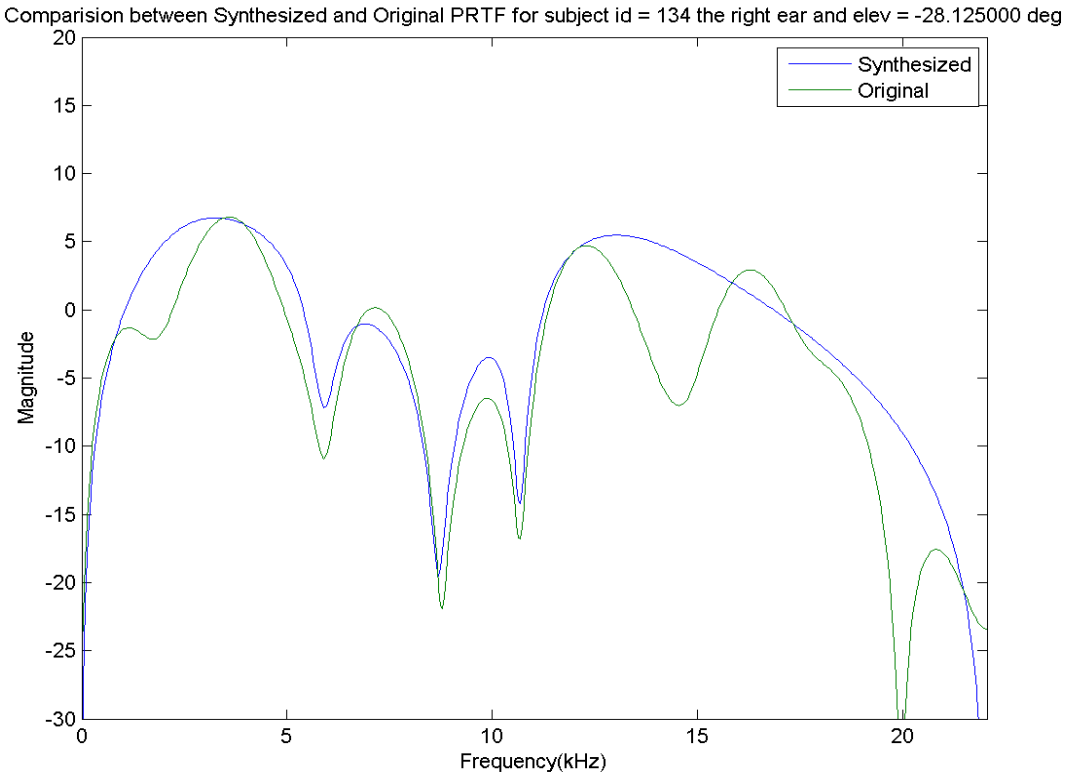
Group Delay Plot

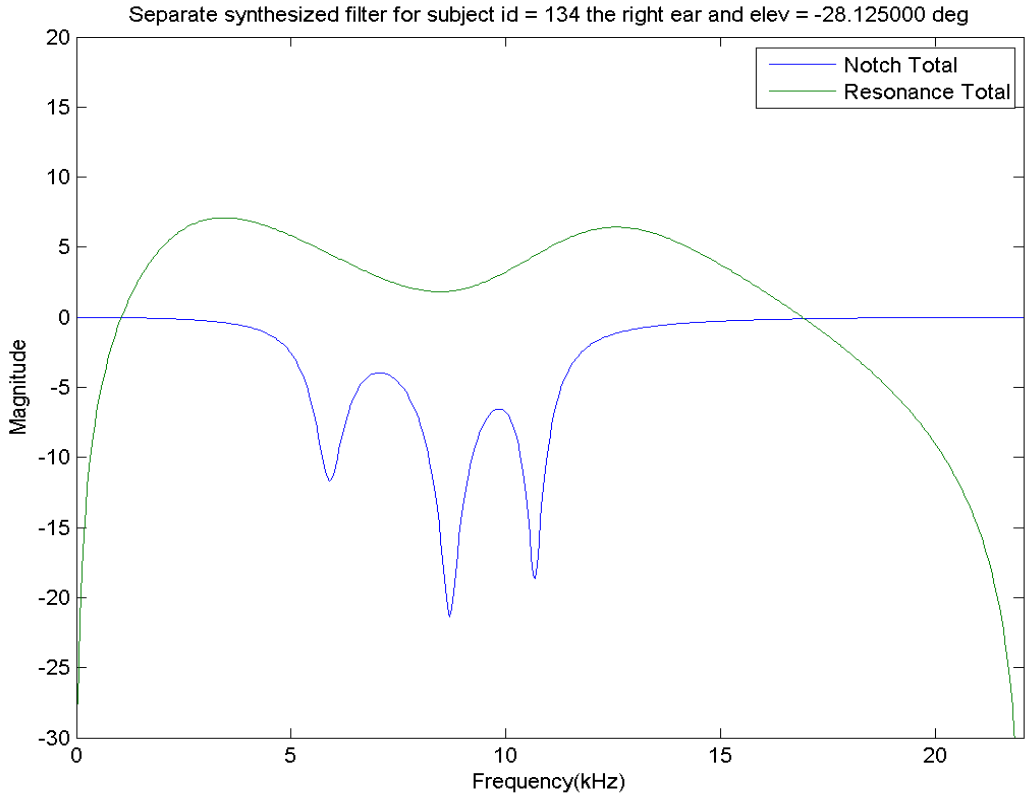


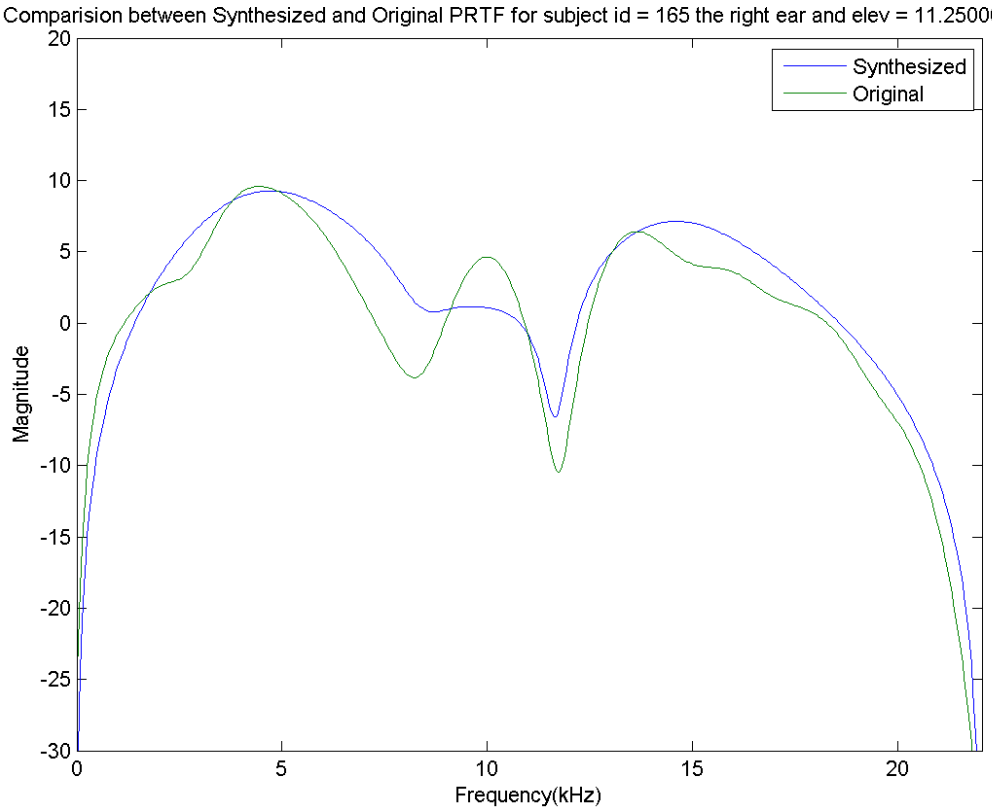
The frequencies where local maxima occur are great estimates of resonance frequencies. However, a better methods like ARMA modelling can be employed to get more reliable results.

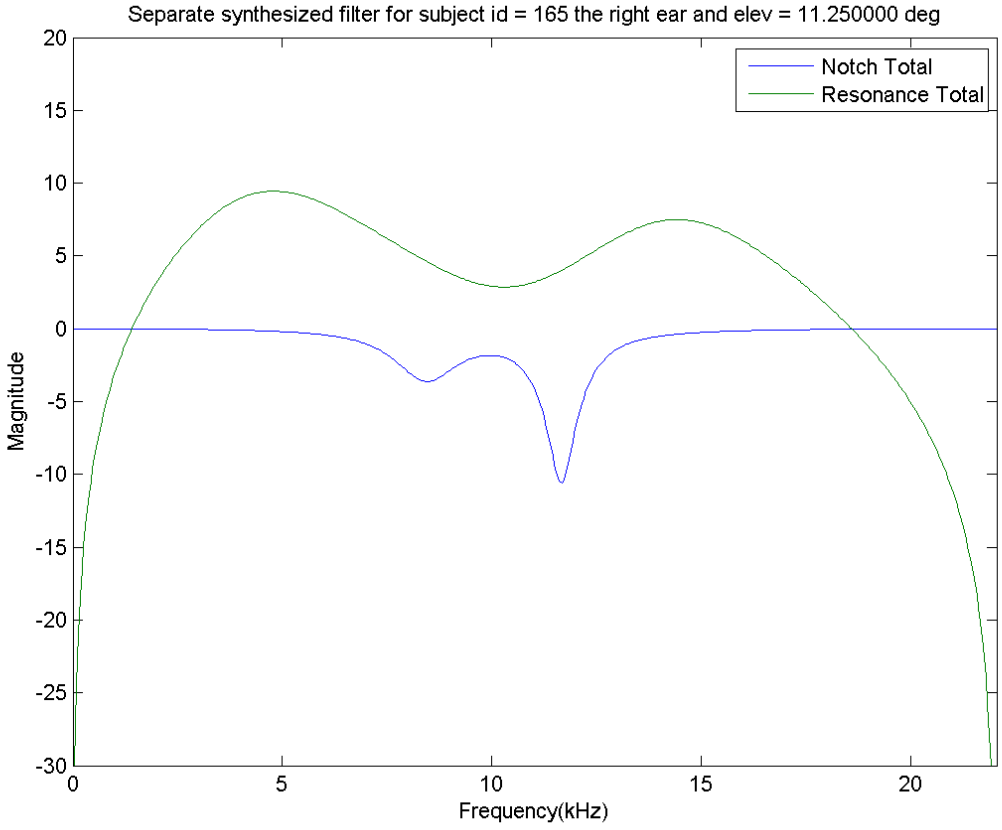
4. RESULTS

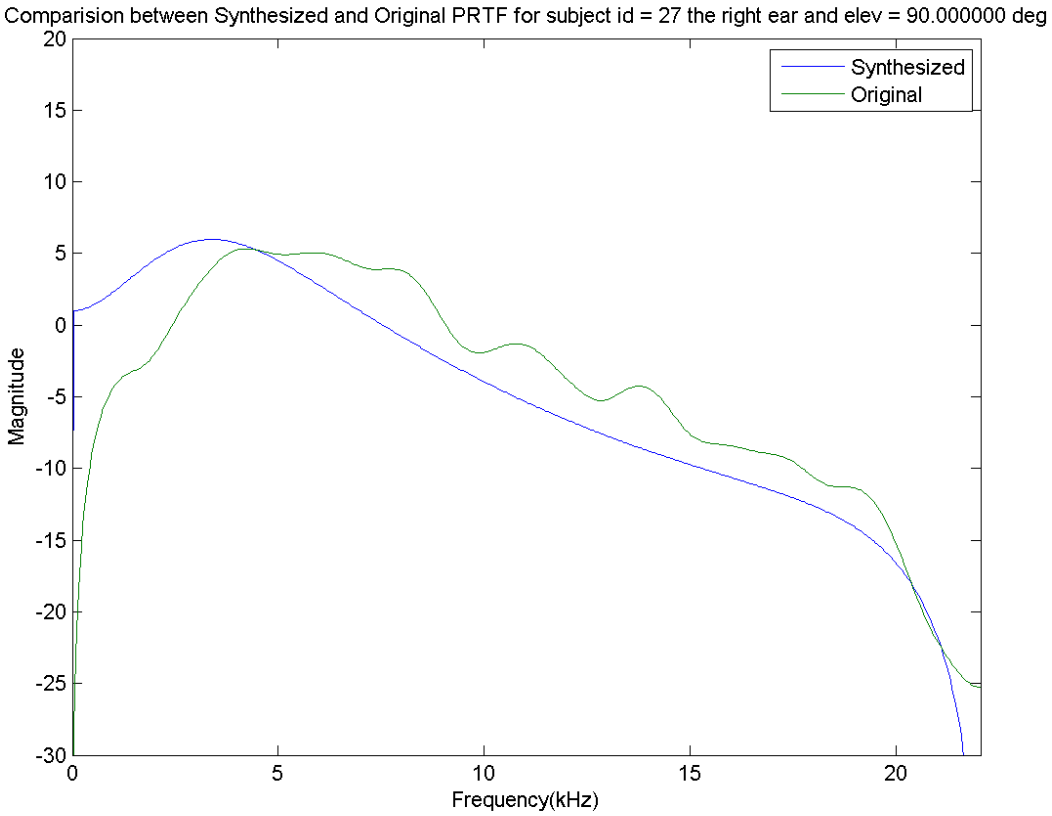
Comparison between original and re-synthesized PRTF magnitudes for 10 distinct subjects, each at a different elevation.

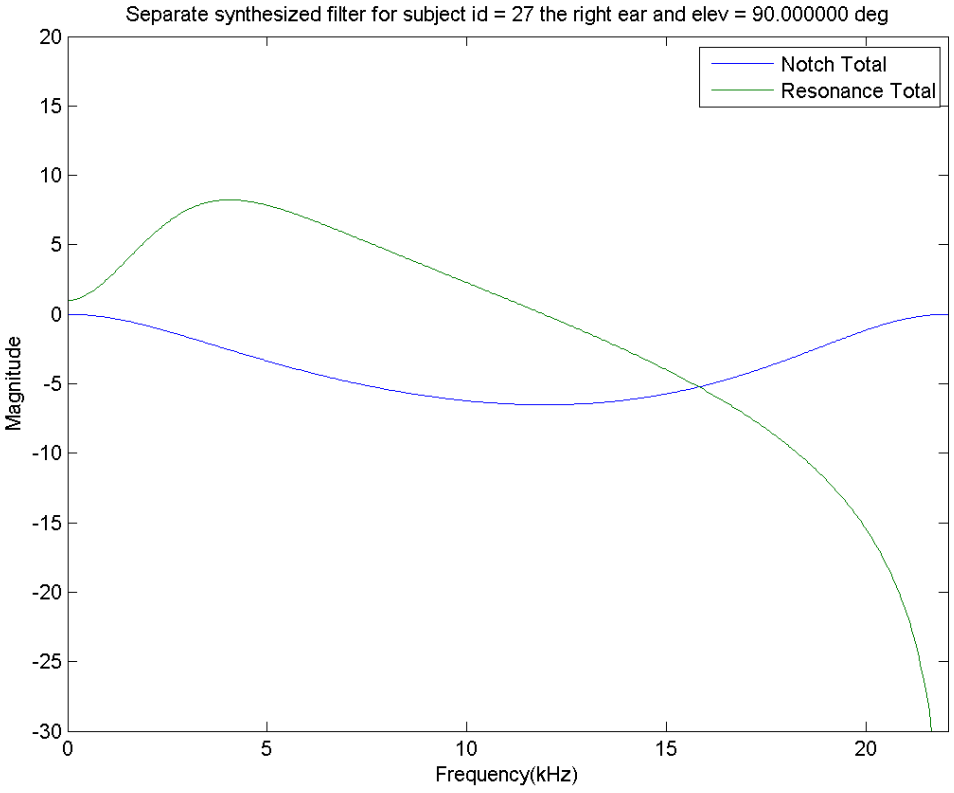

Comparision between Synthesized and Original PRTF for subject id = 134 the right ear and elev = -28.125000 deg
Synthesized
Original
Magnitude
Frequency(kHz)
20
15
10
5
0
-5
-10
-15
-20
-25
-30
0
5
10
15
20

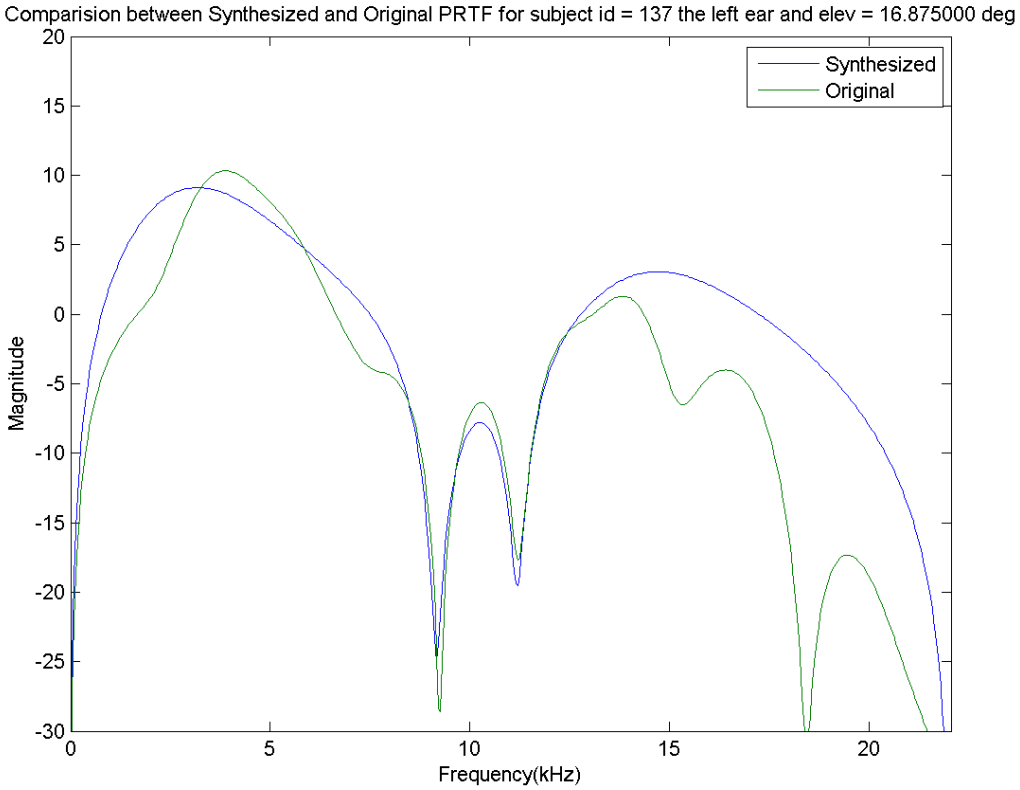

Separate synthesized filter for subject id = 134 the right ear and elev = -28.125000 deg
Notch Total
Resonance Total
Magnitude
Frequency(kHz)
20
15
10
5
0
-5
-10
-15
-20
-25
-30
0
5
10
15
20

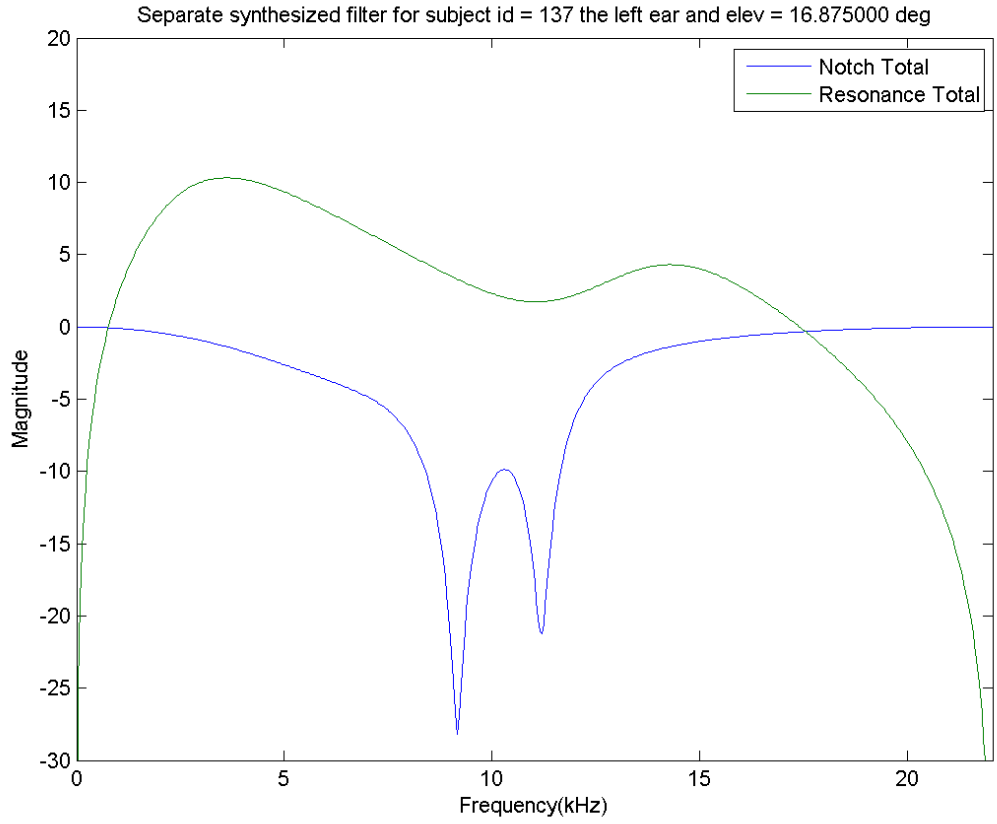

Comparision between Synthesized and Original PRTF for subject id = 165 the right ear and elev = 11.250000 deg
Synthesized
Original
Magnitude
Frequency(kHz)
20
15
10
5
0
-5
-10
-15
-20
-25
-30
0
5
10
15
20

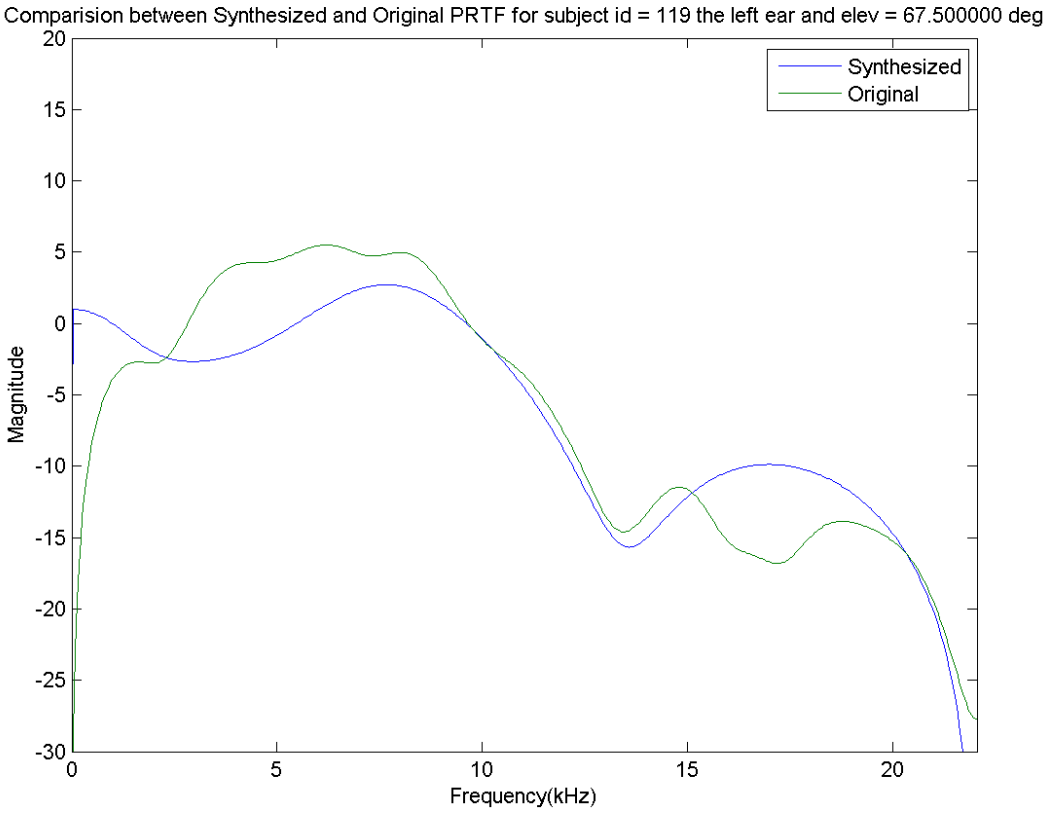

Separate synthesized filter for subject id = 165 the right ear and elev = 11.250000 deg
Notch Total
Resonance Total
Magnitude
Frequency(kHz)
20
15
10
5
0
-5
-10
-15
-20
-25
-30
0
5
10
15
20

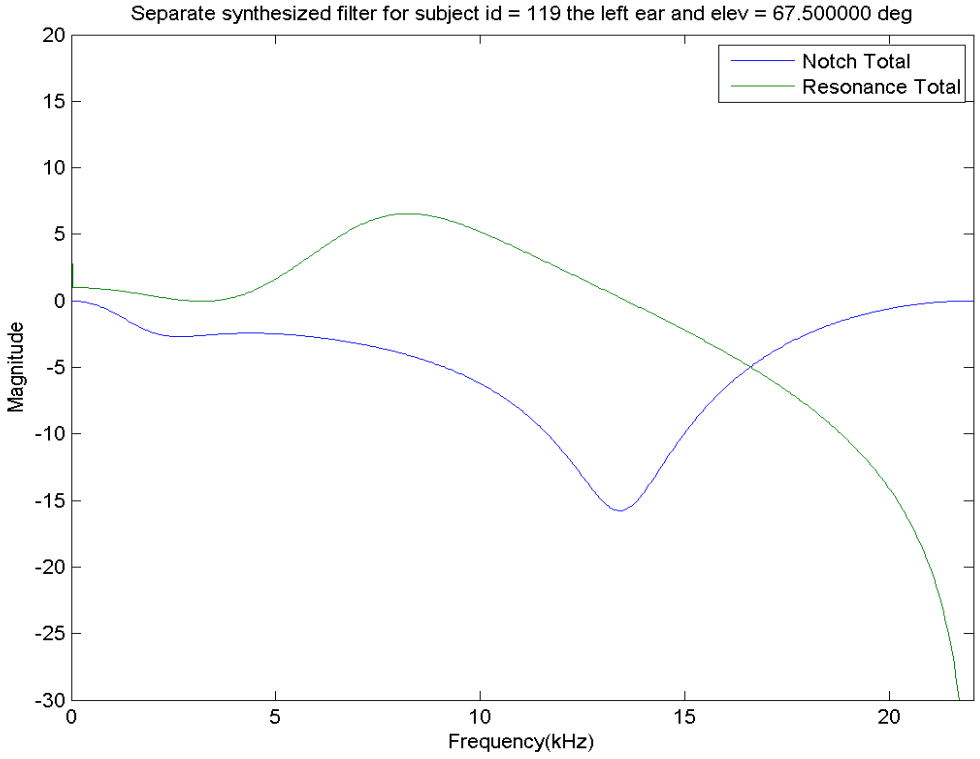

Comparision between Synthesized and Original PRTF for subject id = 27 the right ear and elev = 90.000000 deg
Synthesized
Original
Magnitude
Frequency(kHz)
20
15
10
5
0
-5
-10
-15
-20
-25
-30
0
5
10
15
20

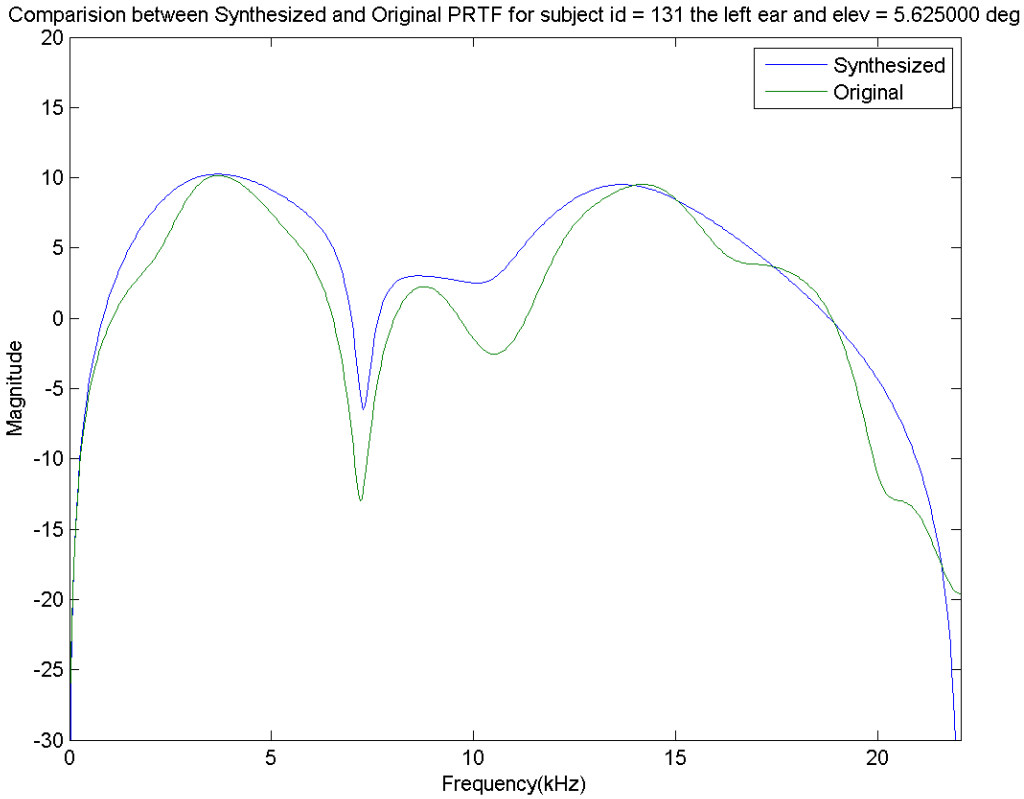

Separate synthesized filter for subject id = 27 the right ear and elev = 90.000000 deg
Notch Total
Resonance Total
Magnitude
Frequency(kHz)
20
15
10
5
0
-5
-10
-15
-20
-25
-30
0
5
10
15
20

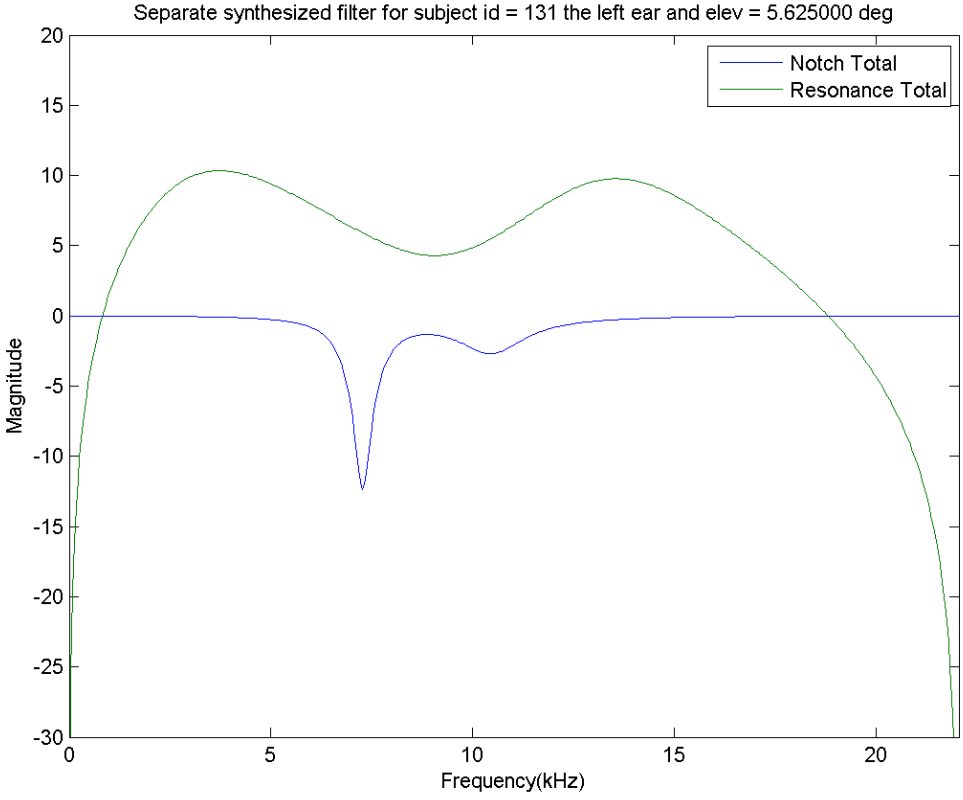

Comparision between Synthesized and Original PRTF for subject id = 137 the left ear and elev = 16.875000 deg
Synthesized
Original
Magnitude
Frequency(kHz)
20
15
10
5
0
-5
-10
-15
-20
-25
-30
0
5
10
15
20

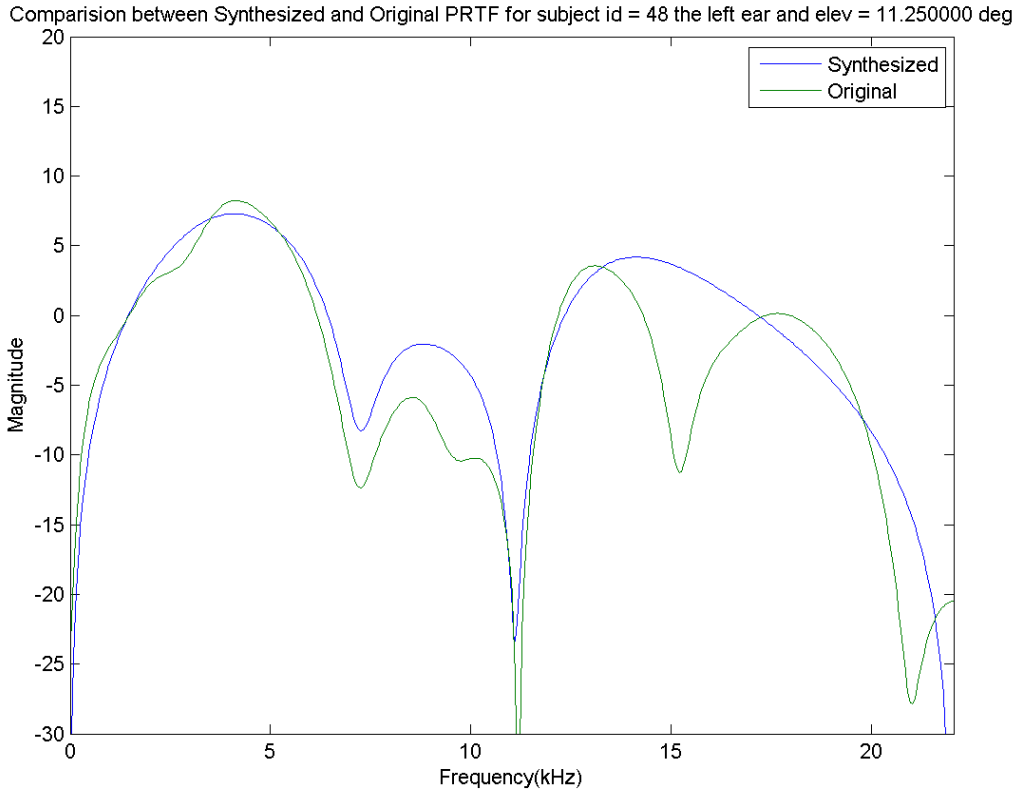

Separate synthesized filter for subject id = 137 the left ear and elev = 16.875000 deg
Notch Total
Resonance Total
Magnitude
Frequency(kHz)
20
15
10
5
0
-5
-10
-15
-20
-25
-30
0
5
10
15
20

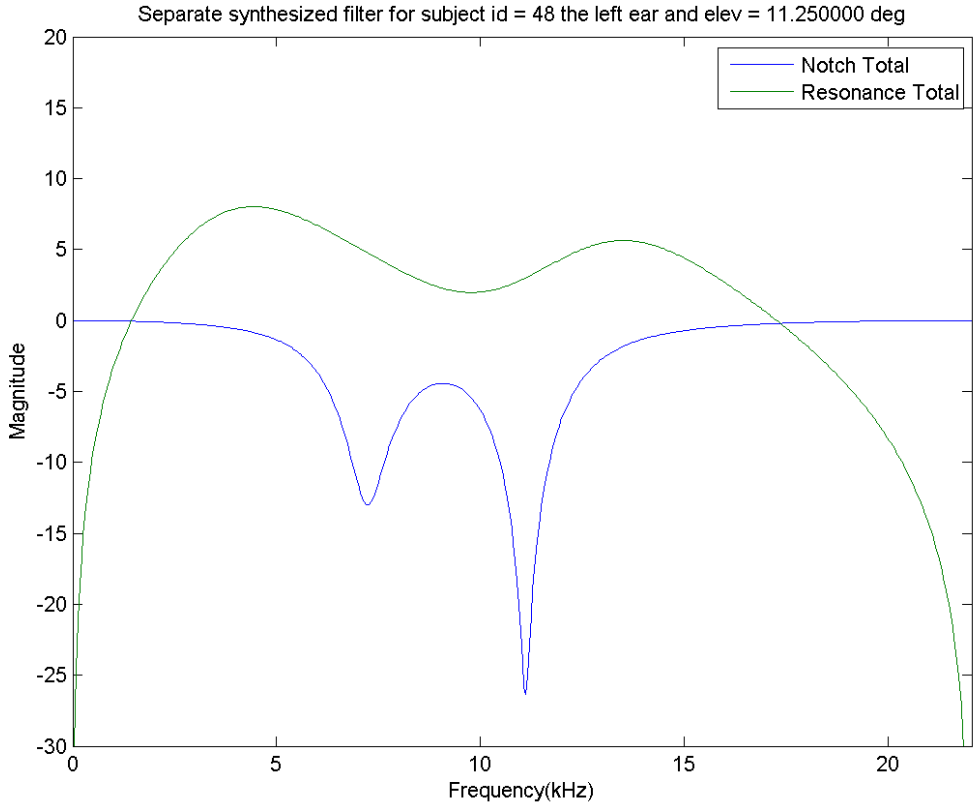

Comparision between Synthesized and Original PRTF for subject id = 119 the left ear and elev = 67.500000 deg
Synthesized
Original
Magnitude
Frequency(kHz)
20
15
10
5
0
-5
-10
-15
-20
-25
-30
0
5
10
15
20

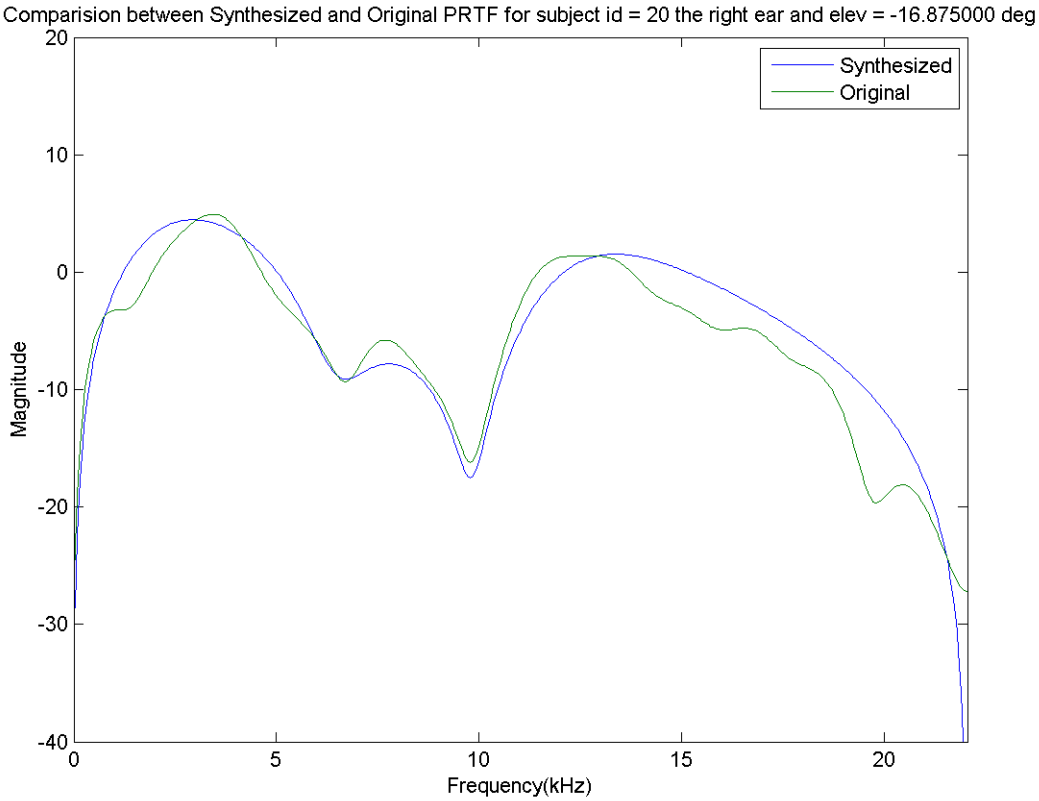

Separate synthesized filter for subject id = 119 the left ear and elev = 67.500000 deg
Notch Total
Resonance Total
Magnitude
Frequency(kHz)
20
15
10
5
0
-5
-10
-15
-20
-25
-30
0
5
10
15
20

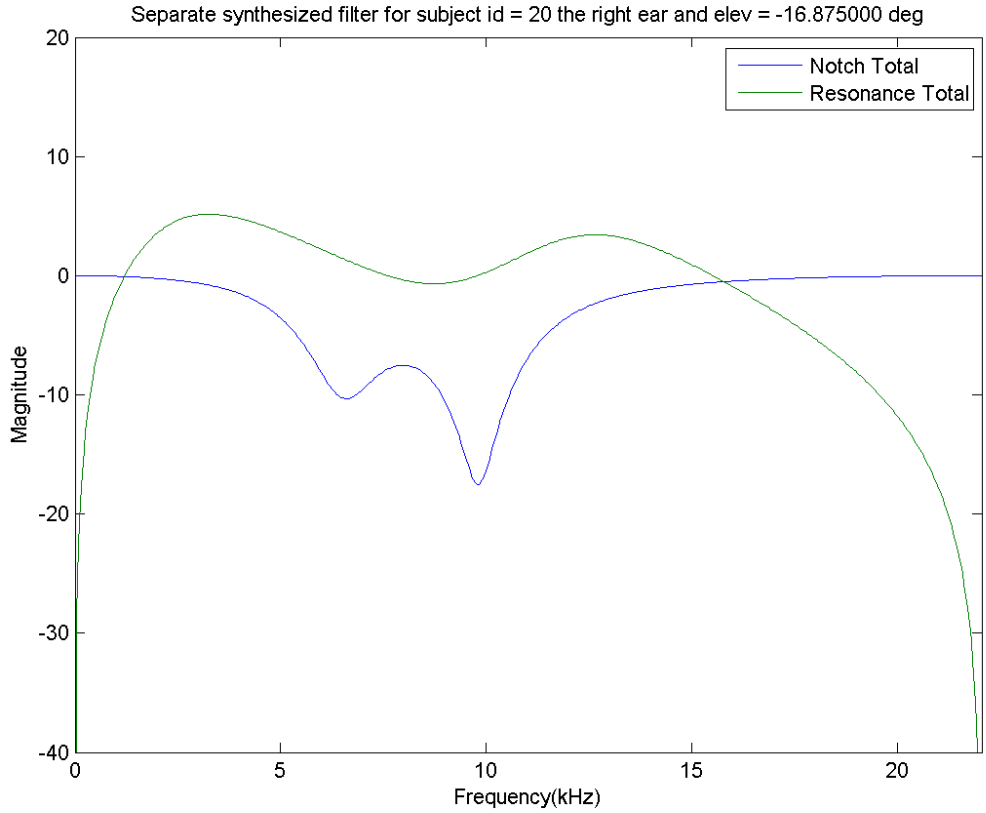

Comparision between Synthesized and Original PRTF for subject id = 131 the left ear and elev = 5.625000 deg
Synthesized
Original
Magnitude
Frequency(kHz)
20
15
10
5
0
-5
-10
-15
-20
-25
-30
0
5
10
15
20

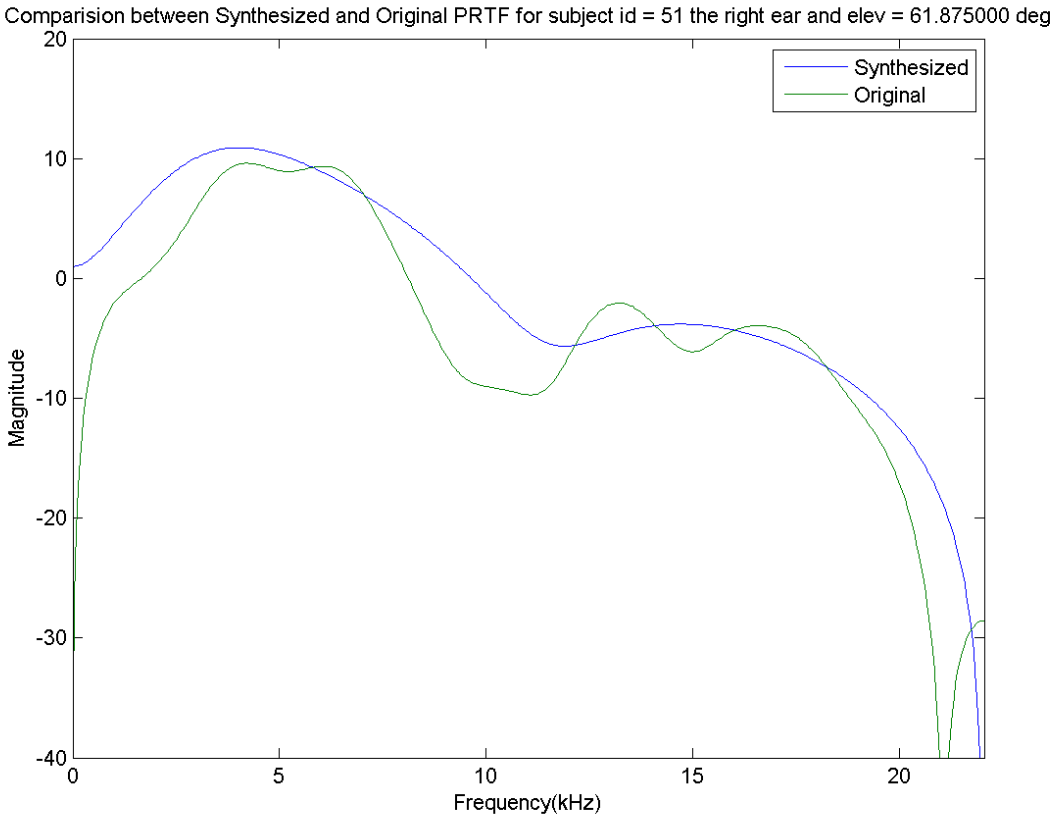

Separate synthesized filter for subject id = 131 the left ear and elev = 5.625000 deg
Notch Total
Resonance Total
Magnitude
Frequency(kHz)
20
15
10
5
0
-5
-10
-15
-20
-25
-30
0
5
10
15
20

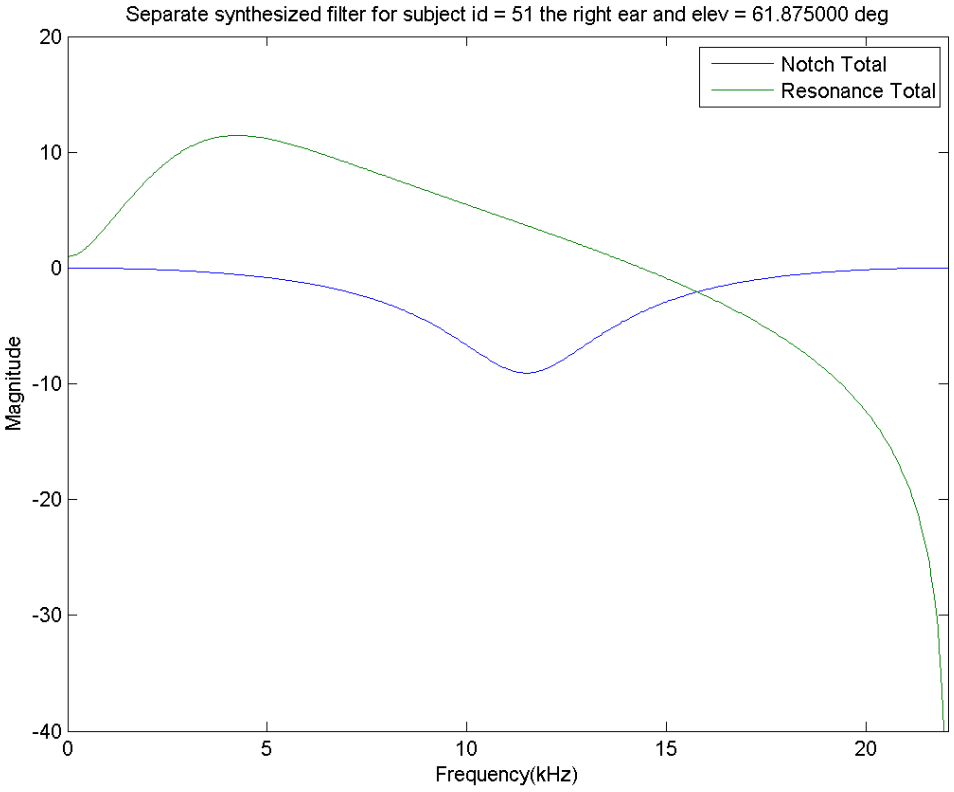

Comparision between Synthesized and Original PRTF for subject id = 48 the left ear and elev = 11.250000 deg
Synthesized
Original
Magnitude
Frequency(kHz)
20
15
10
5
0
-5
-10
-15
-20
-25
-30
0
5
10
15
20

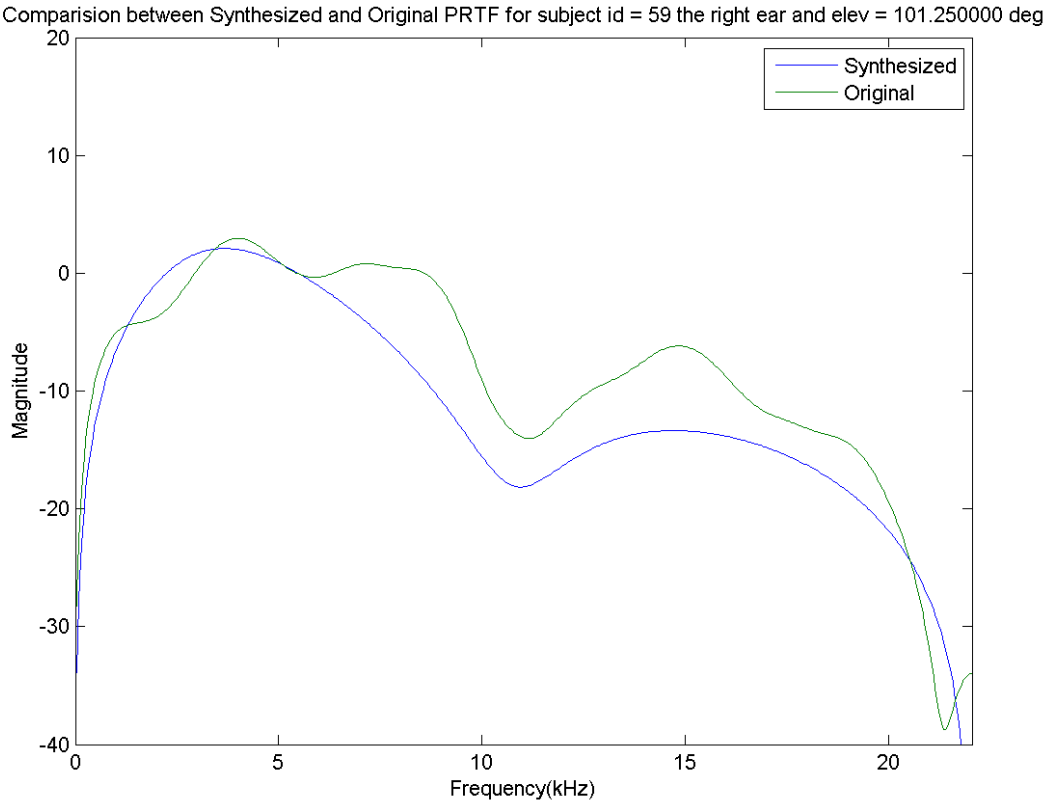

Separate synthesized filter for subject id = 48 the left ear and elev = 11.250000 deg
Notch Total
Resonance Total
Magnitude
Frequency(kHz)
20
15
10
5
0
-5
-10
-15
-20
-25
-30
0
5
10
15
20

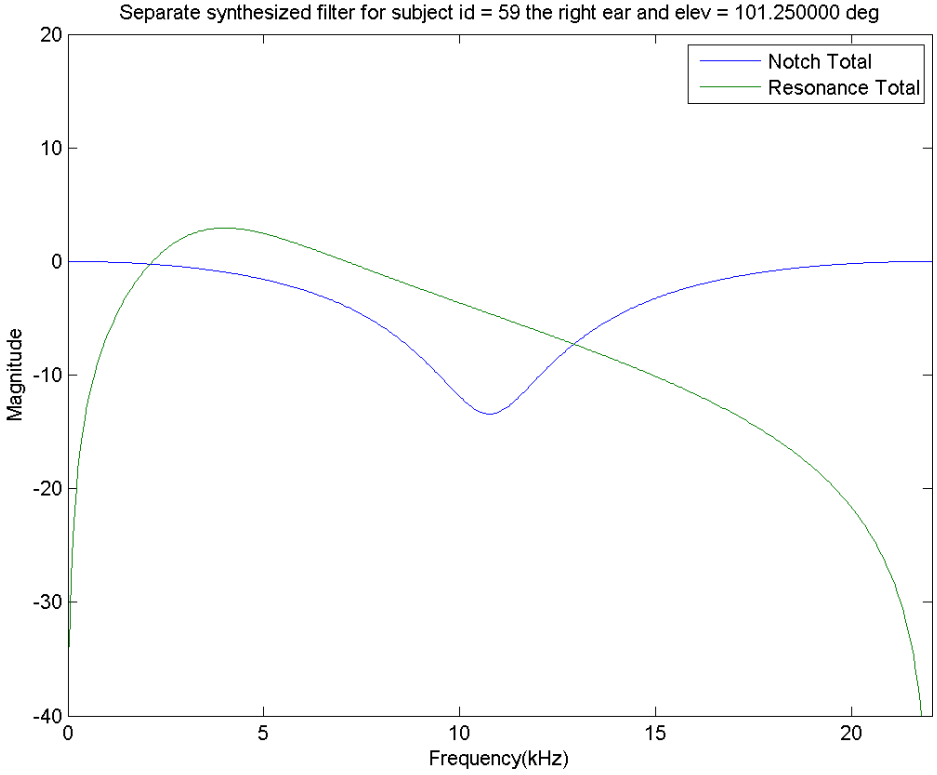

Comparision between Synthesized and Original PRTF for subject id = 20 the right ear and elev = -16.875000 deg
Synthesized
Original
Magnitude
Frequency(kHz)
20
10
0
-10
-20
-30
-40
0
5
10
15
20


Separate synthesized filter for subject id = 20 the right ear and elev = -16.875000 deg
Notch Total
Resonance Total
Magnitude
Frequency(kHz)
20
10
0
-10
-20
-30
-40
0
5
10
15
20


Comparision between Synthesized and Original PRTF for subject id = 51 the right ear and elev = 61.875000 deg
Synthesized
Original
Magnitude
Frequency(kHz)
20
10
0
-10
-20
-30
-40
0
5
10
15
20


Separate synthesized filter for subject id = 51 the right ear and elev = 61.875000 deg
Notch Total
Resonance Total
Magnitude
Frequency(kHz)
20
10
0
-10
-20
-30
-40
0
5
10
15
20


Comparision between Synthesized and Original PRTF for subject id = 59 the right ear and elev = 101.250000 deg
Synthesized
Original
Magnitude
Frequency(kHz)
20
10
0
-10
-20
-30
-40
0
5
10
15
20


Separate synthesized filter for subject id = 59 the right ear and elev = 101.250000 deg
Notch Total
Resonance Total
Magnitude
Frequency(kHz)
20
10
0
-10
-20
-30
-40
0
5
10
15
20

5. CONCLUSION

An approach for structural Pinna Related Transfer Modelling, which exploits an algorithm that separates the resonant and reflective parts of the PRTF spectrum was presented. Decomposition to re-synthesize the original PRTF through a low-order filter model, whose results show an overall suitable approximation was used. In a parallel manner, explanation of the scattering process resulting in the most important spectral notches in the PRTF provided visually convincing results was showed.

6. FUTURE SCOPE

1. Improving the synthesis step by using better methods to determine the resonance frequencies as in some cases the derived frequencies are not suitably accurate

2. Finding a relation between the resonant component derived, to the anthropometry

7. CODE LISTING

7.1 script_tp18.m

```
% DESCRIPTION :
%-------------------------------------------------------------------------------
---------------------
% Main script for EE301A Term Project TP18 (Indian Institute of Technology
Kanpur)
% Session: 2013-14
% Group 18
%
% Title of reeference paper:
%   "Structural Modelling of Pinna-Related Transfer Functions
% Authors (of Ref. paper): S. Spagnol, M. Geronazzo, F. Avanzini
%-------------------------------------------------------------------------------
---------------------
% Author :      Aditya Jain (11037), Anirudh Agrawal (11098), Ishendra Agarwal
(11324),
%               Sohum Datta (11724)
%
%
%*******************************************************************************
*********************
clc;
display('EE301A TERM PROJECT TP18 (INDIAN INSTITUTE OF TECHNOLOGY KANPUR)');
display('Session: 2013-14, Group 18');
display('-----------------------------------------------------------------');
clear all;
close all;

% -------------------------------------------------------------------------
% ASK VARIOUS INPUTS FROM USER
% -------------------------------------------------------------------------
% ask the user of these values
ids=[3,8,9,10,11,12,15,17,18,19,20,21,27,28,33,40,44,48,50,51,58,59,60,61,65,11
9,124,126,127,131,133,134,135,137,147,148,152,153,154,155,156,158,162,163,165]
;
azimuth=[-80 -65 -55 -45:5:45 55 65 80];
elevation=-45+5.625*(0:49);


% Input Subject Id
disp('Valid Subject Ids:');
disp(ids);
subject_id = input('\nEnter Subject id: ');
while isempty(find(ids==subject_id)) == 1
    subject_id = input('Invalid Input. Enter Subject id again: ');
end

pinna = input('Enter which ear - "left" or "right": ', 's');
while strcmp(pinna,'left')==0 && strcmp(pinna,'right')==0
    pinna = input('Enter a valid ear - "left" or "right": ', 's');
end

% Input Elevation Angle
disp('Valid Elevations:');
```

```
disp(elevation);
elev = input('Enter elevation (the program will take the nearest value): ');

% Round of elevation to nearest valid value if not found
if isempty(find(elevation==elev)) == 1
    temp = round((elev + 45)/5.625 + 1);
    elev = -45 + 5.625*(temp - 1);
    if(isempty(find(elevation==elev)) == 0)
        display(sprintf('Nearest Elevation Taken: %f', elev))
    else
        display('Error in Estimation of nearest elevation. Exiting...');
        quit; %CHECK THIS SHOULD NOT HAPPEN
    end
end

% END of Input from User

azim=0;
freq_kHz=22.05;
log_enable=1;
plot_enable = 0;
diff_enable=0;
FFT_LENGTH=1024;

% Load the HRIR from the data base.
[HRIR,HRTF_mag,HRTF_phase,time,frequency]=...
    get_CIPIC_HRIR(subject_id,...
              pinna,...
              elev,...
              azim,...
              freq_kHz,...
              FFT_LENGTH,...
              log_enable,...
              diff_enable,...
              plot_enable);

% Compute the onset time.
[onset]=get_CIPIC_HRIR_onset(subject_id,pinna,elev,azim,freq_kHz);

% Use the HRIR from the onset time onwards.
signal=HRIR(onset:end);

% Extract the pinna spectral notch frequencies
time_cutoff_ms=1.0;
acor_time_cutoff_ms=1.0;
lp_flag=1;
LP_ORDER=12;
threshold=-1;           %Threshold is set to -4 dB
log_flag=1;
fs=2*freq_kHz;          %freq_kHz is set to 22.05 kHz

% Ask user if he wants step-by-step plots for Notch Analysis
flag = input('\nDo you want Plots for Notch Extraction? (y/n): ','s');
if strcmp(flag,'y') == 1
    plot_flag = 1;
else
    plot_flag = 0;
```

```
end

% Analyzing notches
[zero_frequencies_kHz,zero_frequencies_index,zero_gain,plots] = ...
    Algorithm_1(signal,...
                time_cutoff_ms,...
                acor_time_cutoff_ms,...
                lp_flag,...
                LP_ORDER,...
                threshold,...
                fs,...
                FFT_LENGTH,...
                log_flag,...
                plot_flag);

display('Notch Analysis Done.');

% Ask user if he wants step-by-step plots for Resonance Analysis
flag = input('\nDo you want Plots for Resonance analysis? (y/n): ','s');
if strcmp(flag,'y') == 1
    plot_flag = 1;
else
    plot_flag = 0;
end

% Analyzing resonances
[reso_frequencies_kHz,reso_frequencies_index,reso_gain,reso_plots] = ...
    Algorithm_2(signal,...
                time_cutoff_ms,...
                acor_time_cutoff_ms,...
                lp_flag,...
                LP_ORDER,...
                threshold,...
                fs,...
                FFT_LENGTH,...
                log_flag,...
                plot_flag);
display('Resonance Analysis Done.');

%Extract Notches from Notch analysis
[notch_ind, notch_gain, notch_bandwidth, notch_length] =
refine_notch(zero_frequencies_index,  zero_frequencies_kHz, zero_gain,
frequency, plots.win_signal_spectrum);

%Extract Resonances from Resonance analysis
% NOTE: use function 'refine_resonance' for double track(specify 2 frequency
ranges),
% 'refine_resonance_single' for single track (specify only 1 frequency range).
% Double track recommended.
[resonance_ind, resonance_gain, resonance_length] =
refine_resonance(reso_frequencies_index, reso_frequencies_kHz, reso_gain);


%-------------------------------------------------------------------------------
-------------------------
%   Synthesizing NUMERATOR and DENOMINATOR transfer functions
```

```
%-------------------------------------------------------------------------
------------------------

% NOTCH numerator transfer functions
notch_num = zeros(notch_length,3);     %numerator vector of each notches; row i
contains numerator for notch_ind(i)
% NOTCH denominator transfer functions
notch_den = zeros(notch_length,3);     %denominator vector of each notches; row
i contains numerator for notch_ind(i)

for i = 1:notch_length             %notch number
    d = - cos(2 * pi * frequency(notch_ind(i))/ fs);
    V = 10^(notch_gain(i)/20);
    H = V - 1;
    k = (tan(pi * notch_bandwidth(i)/ fs) - V)/(tan(pi * notch_bandwidth(i)/
fs) + V);
    % numerator coefficients
    notch_num(i,1) = 1 + (1 + k)*0.5*H;
    notch_num(i,2) = d*(1-k);
    notch_num(i,3) = (-k) - 0.5*(k + 1)*H;
    % denominator coefficients
    notch_den(i,1) = 1;
    notch_den(i,2) = d*(1-k);
    notch_den(i,3) = -k;
end

% RESONANCE numerator transfer functions
resonance_num = zeros(resonance_length, 3);     %numerator vector of each
resonance; row i contains numerator for resonance_ind(i)
% RESONANCE denominator transfer functions
resonance_den = zeros(resonance_length,3);     %denominator vector of each
resonance; row i contains numerator for resonance_ind(i)

for i = 1:resonance_length
    h = 1/(1 + tan(pi * 5/fs));
    d = - cos(2 * pi * frequency(resonance_ind(i))/ fs);
    V = 10^(resonance_gain(i)/20);
    % numerator coefficients
    resonance_num(i,1) = V*(1-h);
    resonance_num(i,3) = V*(h-1);
    % denominator coefficients
    resonance_den(i,1) = 1;
    resonance_den(i,2) = 2*d*h;
    resonance_den(i,3) = 2*h-1;
end


% -------------------------------------------------------------------------
% Construct The Predicted PRTF
% -------------------------------------------------------------------------

% construct a series cascade of resonances.
if(notch_length > 0)
    tf_notch_num = notch_num(1,:);
    tf_notch_den = notch_den(1,:);
    for i=2:notch_length
        tf_notch_num = conv(tf_notch_num, notch_num(i,:));
```

```
        tf_notch_den = conv(tf_notch_den, notch_den(i,:));
    end
else    %if no notches found
    tf_notch_num = 1;
    tf_notch_den = 1;
end

% construct a parallel additive cascade of resonances.
if(resonance_length > 0)
    tf_reso_num = resonance_num(1,:);
    tf_reso_den = resonance_den(1,:);
    for i=2:resonance_length
        tf_reso_num = conv(tf_reso_num, resonance_den(i,:)) + conv(tf_reso_den,
resonance_num(i,:));
        tf_reso_den = conv(tf_reso_den, resonance_den(i,:));
    end
else    %if no resonances found
    tf_reso_num = 1;
    tf_reso_den = 1;
end

% Cascade the resonance and notch blocks
tf_final_num = conv(tf_notch_num, tf_reso_num);
tf_final_den = conv(tf_notch_den, tf_reso_den);

% ------------------------------------------------------------------------
%  Plot Predicted PRTF and Actual PRTF
% ------------------------------------------------------------------------


w = linspace(0, 2*pi, FFT_LENGTH);
w_exp = exp(1i*w);
val_num = polyval(tf_final_num, w_exp);
val_den = polyval(tf_final_den, w_exp);
val_num = val_num(1:FFT_LENGTH/2 + 1);
val_den = val_den(1:FFT_LENGTH/2 + 1);

% plot original and Predicted PRTF
figure;
plot(frequency, 20*log10(abs(val_num./val_den)),frequency,
plots.win_signal_spectrum);
axis([0 22.05 -30 20]); xlabel('Frequency(kHz)');ylabel('Magnitude');
legend('Synthesized PRTF','Original PRTF');
title(sprintf('Comparision between Synthesized and Original PRTF for subject id
= %d the %s ear and elev = %f deg',subject_id,pinna,elev));


%plot the notch and resonance part separately
figure;
val_reso_num = polyval(tf_reso_num, w_exp);
val_reso_den = polyval(tf_reso_den, w_exp);
val_reso_num = val_reso_num(1:FFT_LENGTH/2 + 1);
val_reso_den = val_reso_den(1:FFT_LENGTH/2 + 1);

val_notch_num = polyval(tf_notch_num, w_exp);
val_notch_den = polyval(tf_notch_den, w_exp);
val_notch_num = val_notch_num(1:FFT_LENGTH/2 + 1);
```

```
val_notch_den = val_notch_den(1:FFT_LENGTH/2 + 1);
plot(frequency, 20*log10(abs(val_notch_num./val_notch_den)),frequency,
20*log10(abs(val_reso_num./val_reso_den)));
axis([0 22.05 -30 20]); xlabel('Frequency(kHz)');ylabel('Magnitude');
legend('Notch Total','Resonance Total');
title(sprintf('Separate synthesized filter for subject id = %d the %s ear and
elev = %f deg',subject_id,pinna,elev));
display('----END----');
```

7.2 Algorithm_1.m

```
function [zero_frequencies_kHz,zero_frequencies_index,zero_gain,plots] = ...
    Algorithm_1(signal,...
                time_cutoff_ms,...
                acor_time_cutoff_ms,...
                lp_flag,...
                LP_ORDER,...
                threshold,...
                fs,...
                FFT_LENGTH,...
                log_flag,...
                plot_flag)

%-------------------------------------------------------------------------------
---------------------
%This function returns the frequencies of the pinna spectral notches.
%-------------------------------------------------------------------------------
---------------------
% Author     :     Vikas.C.Raykar
% Date       :     25 May 2004
% Contact    :     vikas@umiacs.umd.edu
%-------------------------------------------------------------------------------
---------------------
%REFERENCE :
%-------------------------------------------------------------------------------
---------------------
% Extracting frequencies of the pinna spectral notches in measured head related
impulse responses
% Vikas C. Raykar, Ramani Duraiswami, and B. Yegnanarayana, University of
Maryland CollegePark,
% Department of Computer Science Technical Report, CS-TR-4609, July 2004
% (also published as UMIACS-TR-2004-51).
%-------------------------------------------------------------------------------
---------------------
% EXAMPLE USAGE : See the script Algorithm_1_driver.m
%-------------------------------------------------------------------------------
---------------------

%-------------------------------------------------------------------------------
---------------------
% INPUT :
%-------------------------------------------------------------------------------
---------------------
% signal                : The time domain HRIR signal.
%
%   Determine the initial onset of the HRIR and use the HRIR from that instant.
%   The onset is available for the CIPIC database. Use the function
%   get_CIPIC_HRIR_onset.m to retrive the onset time.
%   [onset]=get_CIPIC_HRIR_onset(subject_id,pinna,elev,azim,freq_kHz);
%   signal=HRIR(onset:end)
%
% time_cutoff_ms        : Size of the Hann window in ms.
%
%   The torso and the knee reflections are cutoff by this window. A value of
%   1.0 ms is sufficient if the HRIR is used from the initial onset. Else
%   use a value of initial onset+1.0 ms.
%
```

```
% acor_time_cutoff_ms  : Size of the window for windowing the autocorrelation
fucntion.
%
%   Use a value of around 1.0-2.0 ms.
%
% lp_flag              : if 1 the analysis is done on the LP residual
% LP_ORDER             : Order for LP analysis (10-12)
%
%   The script has the option of not using LP analysis. However LP analysis
%   is recommended.
%
% threshold            : threshold for the group delay function (0 to -1)
%
%   Using the zero threshold for the group delay function, all valleys below
the zero value
%   are marked as relevant notches and their frequencies are noted. In practice
a slightly
%   lower threshold of -1 was found to give good results and eliminate any
spurious nulls
%   by windowing.
%
% fs                   : Sampling frequency in kHz
% FFT_LENGTH           : length of the FFT in samples
% log_flag             : if 1 the spectrum magnitude is in dB
% plot_flag            : if 1 the results are plotted
%
%-------------------------------------------------------------------------------
---------------------
% OUTPUT :
%-------------------------------------------------------------------------------
---------------------
% zero_frequencies_kHz   : Frequencies of the pinna spectral notches in kHz.
%
% zero_frequencies_index : The corresponding  frequency index.
%
%   frequency=[0:fs/FFT_LENGTH:fs/2];
%   zero_frequencies_kHz=frequency(zero_frequencies_index);
%
% plots                  : A structure containing various intermediate results.
%-------------------------------------------------------------------------------
---------------------
% ALGORITHM
%-------------------------------------------------------------------------------
---------------------
% 1. Get residual signal from Linear Prediction.
% 2. Window the signal using a Hann window.
% 3. Find the autocorrelation fucntion of the windowed signal.
% 4. Window the autocorrealtion function.
% 5. Find the the group-delay of the windowed autocorrelation signal.
% 6. Find local minima in the thresholded group delay.
%-------------------------------------------------------------------------------
---------------------




%-----------------------------------------------------------------------
% Frequency and time axis
%-----------------------------------------------------------------------
```

```
N=length(signal);
frequency=[0:fs/FFT_LENGTH:fs/2];
time=[0:1/(fs):(N-1)/(fs)];

%-------------------------------------------------------------------------
% Make a copy of the original signal for plotting purposes
%-------------------------------------------------------------------------

original_signal=signal;

%-------------------------------------------------------------------------
% Spectrum of the original signal
%-------------------------------------------------------------------------

original_signal_spectrum=abs(fft(signal,FFT_LENGTH));
original_signal_spectrum=original_signal_spectrum(1:(FFT_LENGTH/2)+1);

if log_flag == 1
    original_signal_spectrum=20*log10(original_signal_spectrum);
end

%-------------------------------------------------------------------------
% Hanning Window
%-------------------------------------------------------------------------

M=round(time_cutoff_ms*fs);
win=zeros(2*N-1,1);
win(N-M+1:N+M-1)=window(@hann,2*M-1);
win=win(N:end);

%-------------------------------------------------------------------------
%   NOTE: CODE ADDED BY SOHUM DATTA
%-------------------------------------------------------------------------

win_signal=signal.*win; %Windowed signal (without LP analysis done)

%spectrum of the above signal
windowed_signal_spectrum=abs(fft(win_signal,FFT_LENGTH));
windowed_signal_spectrum=windowed_signal_spectrum(1:(FFT_LENGTH/2)+1);
windowed_signal_spectrum=20*log10(windowed_signal_spectrum);    %converting to
log
%-------------------------------------------------------------------------
% Linear Prediction Aanalysis
%-------------------------------------------------------------------------

if lp_flag == 1

    LP_coefficients=lpc(signal,LP_ORDER);
    LP_residual=real(filter(LP_coefficients',1,signal));
    signal=LP_residual;

end

%-------------------------------------------------------------------------
% Spectrum of the signal
```

```
%-------------------------------------------------------------------------

signal_spectrum=abs(fft(signal,FFT_LENGTH));
signal_spectrum=signal_spectrum(1:(FFT_LENGTH/2)+1);

if log_flag == 1
    signal_spectrum=20*log10(signal_spectrum);
end

%-------------------------------------------------------------------------
% Window the signal
%-------------------------------------------------------------------------

win_signal=signal.*win;

%-------------------------------------------------------------------------
%Spectrum of the windowed signal
%-------------------------------------------------------------------------

win_signal_spectrum=abs(fft(win_signal,FFT_LENGTH));
win_signal_spectrum=win_signal_spectrum(1:(FFT_LENGTH/2)+1);

if log_flag == 1
    win_signal_spectrum=20*log10(win_signal_spectrum);
end

%-------------------------------------------------------------------------
% Autocorrelation of the windowed signal
%-------------------------------------------------------------------------

win_signal_acor=xcorr(win_signal);

%-------------------------------------------------------------------------
% Spectrum of the  autocorrelation signal
%-------------------------------------------------------------------------

win_signal_acor_spectrum=abs(fft(win_signal_acor,FFT_LENGTH));
win_signal_acor_spectrum=win_signal_acor_spectrum(1:(FFT_LENGTH/2)+1);

if log_flag == 1
   win_signal_acor_spectrum=20*log10(win_signal_acor_spectrum);
end

%-------------------------------------------------------------------------
% Autocorrelation window
%-------------------------------------------------------------------------

M=round(acor_time_cutoff_ms*fs);
awin=zeros(2*N-1,1);
awin(N-M+1:N+M-1)=window(@hann,2*M-1);

%-------------------------------------------------------------------------
% Window the autocorrelation fucntion
%-------------------------------------------------------------------------

win_signal_acor_win=(win_signal_acor.*awin);
```

```
awin=awin(N:end);

%-------------------------------------------------------------------------
% Spectrum of the windowed autocorrelation signal
%-------------------------------------------------------------------------

win_signal_acor_win_spectrum=abs(fft(win_signal_acor_win,FFT_LENGTH));
win_signal_acor_win_spectrum=win_signal_acor_win_spectrum(1:(FFT_LENGTH/2)+1);

if log_flag == 1
   win_signal_acor_win_spectrum=20*log10(win_signal_acor_win_spectrum);
end

%-------------------------------------------------------------------------
% Group delay of the windowed autocorrelation signal
%-------------------------------------------------------------------------

win_signal_acor_win=win_signal_acor_win(N:end);
win_signal_acor=win_signal_acor(N:end);

grp_delay=grpdelay(win_signal_acor_win,1,FFT_LENGTH,'whole');
grp_delay=grp_delay(1:(FFT_LENGTH/2)+1);

%-------------------------------------------------------------------------
% Zero threshold the group delay
%-------------------------------------------------------------------------
zero_threshold_grp_delay=grp_delay;
zero_threshold_grp_delay(find(zero_threshold_grp_delay > threshold))=0;

%-------------------------------------------------------------------------
% Find the local minima in the group delay function
%-------------------------------------------------------------------------

[lmval,indd]=lmin(zero_threshold_grp_delay,2);      %NOTE: here argument 2 of
lmin was changed from 0 to 2

zero_frequencies_kHz=frequency(indd);
zero_frequencies_index=indd;

% This should be whose gain?? CHECK THIS PART
zero_gain = windowed_signal_spectrum(indd);     %gain from  windowed and LP
analysis)

%-------------------------------------------------------------------------
% Return the intermediate results to the structure plots
%-------------------------------------------------------------------------

plots.frequency=frequency;
plots.time=time;
plots.original_signal=original_signal;
plots.original_signal_spectrum=original_signal_spectrum;
plots.signal=signal;
plots.signal_spectrum=signal_spectrum;
plots.win=win;
plots.win_signal=win_signal;
```

```
plots.win_signal_spectrum=windowed_signal_spectrum;          %NOTE Here:
windowed LP spectrum for evaluation of bandwidth
plots.win_signal_acor=win_signal_acor;
plots.win_signal_acor_spectrum=win_signal_acor_spectrum;
plots.awin=awin;
plots.win_signal_acor_win=win_signal_acor_win;
plots.win_signal_acor_win_spectrum=win_signal_acor_win_spectrum;
plots.grp_delay=grp_delay;
plots.zero_threshold_grp_delay=zero_threshold_grp_delay;


if plot_flag==1

    figure;
    K=6;
    L=2;

    %----------------------------------------

    subplot(K,L,1);
    plot(time,original_signal);
    axis([time(1) time(end) -1 1]);
    h=ylabel('(a)');
    set(h,'rotation',0);
    legend('Signal');


   %----------------------------------------

    subplot(K,L,3);
    plot(time,signal);
    axis([time(1) time(end) -1 1]);
   hold on;
    plot(time,win*max(signal),'r:');
    h=ylabel('(b)');
    set(h,'rotation',0);
    legend('LP residual');


    %----------------------------------------

    subplot(K,L,5);
    plot(time,win_signal);
    axis([time(1) time(end) -1 1]);
    h=ylabel('(c)');
    set(h,'rotation',0);
    legend('Windowed LP residual');


    %----------------------------------------

    subplot(K,L,7);
    plot(time,win_signal_acor);
    axis([time(1) time(end) -1 1]);
    hold on;
    plot(time,awin*max(win_signal_acor),'r:');
```

```
        h=ylabel('(d)');
       set(h,'rotation',0);
           legend('Autocorrelation');


    %----------------------------------------

    subplot(K,L,9);
    plot(time,win_signal_acor_win);
    axis([time(1) time(end) -1 1]);
    xlabel('Time (ms)');
        h=ylabel('(e)');
       set(h,'rotation',0);
    legend('Windowed Autocorrelation');


    %----------------------------------------

    subplot(K,L,2);
    plot(frequency,original_signal_spectrum);
    axis([frequency(1) frequency(end) min(original_signal_spectrum)
max(original_signal_spectrum)]);
        h=ylabel('(f)');
       set(h,'rotation',0);
    title('Corresponding Spectrum Magnitude');

    %----------------------------------------

    subplot(K,L,4);
    plot(frequency,signal_spectrum);
    axis([frequency(1) frequency(end) min(signal_spectrum)
max(signal_spectrum)]);
        h=ylabel('(g)');
       set(h,'rotation',0);


    %----------------------------------------

    subplot(K,L,6);
    plot(frequency,windowed_signal_spectrum);
    axis([frequency(1) frequency(end) min(windowed_signal_spectrum)
max(windowed_signal_spectrum)]);
        h=ylabel('(h)');
       set(h,'rotation',0);


    %----------------------------------------

    subplot(K,L,8);
    plot(frequency,win_signal_acor_spectrum);
   axis([frequency(1) frequency(end) min(win_signal_acor_spectrum)
max(win_signal_acor_spectrum)]);
       h=ylabel('(i)');
       set(h,'rotation',0);
```

```
    %----------------------------------------

    subplot(K,L,10);
    plot(frequency,win_signal_acor_win_spectrum);
    h=ylabel('dB');
    axis([frequency(1) frequency(end) min(win_signal_acor_win_spectrum)
max(win_signal_acor_win_spectrum)]);
        h=ylabel('(j)');
            set(h,'rotation',0);


    %----------------------------------------
    hold on;
    subplot(K,L,12);
    plot(frequency,grp_delay);
    xlabel('Frequency (kHz)');
    hold on;
    axis([frequency(1) frequency(end) min(grp_delay) max(grp_delay) ]);
    hold on;
    plot(zero_frequencies_kHz,grp_delay(zero_frequencies_index),'r.');
    h=ylabel('(k)');
    hold on;
    plot(frequency,threshold*ones(size(frequency)),'b:');
    set(h,'rotation',0);
    title('Group Delay');

end
```

7.3 Algorithm_2.m

```
function [reso_frequencies_kHz,reso_frequencies_index,reso_gain,plots] = ...
    Algorithm_2(signal,...
                time_cutoff_ms,...
                acor_time_cutoff_ms,...
                lp_flag,...
                LP_ORDER,...
                threshold,...
                fs,...
                FFT_LENGTH,...
                log_flag,...
                plot_flag)

%-------------------------------------------------------------------------------
---------------------
%This function returns the frequencies of the pinna spectral RESONANCES.
%-------------------------------------------------------------------------------
---------------------
% Author     :     Vikas.C.Raykar (Sohum datta modified this)
% Date       :     25 May 2004
% Contact    :     vikas@umiacs.umd.edu
%-------------------------------------------------------------------------------
---------------------
%REFERENCE :
%-------------------------------------------------------------------------------
---------------------
% Extracting frequencies of the pinna spectral notches in measured head related
impulse responses
% Vikas C. Raykar, Ramani Duraiswami, and B. Yegnanarayana, University of
Maryland CollegePark,
% Department of Computer Science Technical Report, CS-TR-4609, July 2004
% (also published as UMIACS-TR-2004-51).
%-------------------------------------------------------------------------------
---------------------
% EXAMPLE USAGE : See the script Algorithm_1_driver.m
%-------------------------------------------------------------------------------
---------------------

%-------------------------------------------------------------------------------
---------------------
% INPUT :
%-------------------------------------------------------------------------------
---------------------
% signal                : The time domain HRIR signal.
%
%   Determine the initial onset of the HRIR and use the HRIR from that instant.
%   The onset is available for the CIPIC database. Use the function
%   get_CIPIC_HRIR_onset.m to retrive the onset time.
%   [onset]=get_CIPIC_HRIR_onset(subject_id,pinna,elev,azim,freq_kHz);
%   signal=HRIR(onset:end)
%
% time_cutoff_ms        : Size of the Hann window in ms.
%
%   The torso and the knee reflections are cutoff by this window. A value of
%   1.0 ms is sufficient if the HRIR is used from the initial onset. Else
%   use a value of initial onset+1.0 ms.
```

```
%
% acor_time_cutoff_ms  : Size of the window for windowing the autocorrelation
fucntion.
%
%   Use a value of around 1.0-2.0 ms.
%
% lp_flag              : if 1 the analysis is done on the LP residual
% LP_ORDER             : Order for LP analysis (10-12)
%
%   The script has the option of not using LP analysis. However LP analysis
%   is recommended.
%
% threshold            : threshold for the group delay function (0 to -1)
%
%   Using the zero threshold for the group delay function, all valleys below
the zero value
%   are marked as relevant notches and their frequencies are noted. In practice
a slightly
%   lower threshold of -1 was found to give good results and eliminate any
spurious nulls
%   by windowing.
%
% fs                   : Sampling frequency in kHz
% FFT_LENGTH           : length of the FFT in samples
% log_flag             : if 1 the spectrum magnitude is in dB
% plot_flag            : if 1 the results are plotted
%
%-------------------------------------------------------------------------------
---------------------
% OUTPUT :
%-------------------------------------------------------------------------------
---------------------
% reso_frequencies_kHz   : Frequencies of the pinna spectral resonance in kHz.
%
% reso_frequencies_index : The corresponding  frequency index.
%
%   frequency=[0:fs/FFT_LENGTH:fs/2];
%   reso_frequencies_kHz=frequency(reso_frequencies_index);
%
% plots                  : A structure containing various intermediate results.
%-------------------------------------------------------------------------------
---------------------
% ALGORITHM
%-------------------------------------------------------------------------------
---------------------
% 1. Get residual signal from Linear Prediction.
% 2. Window the signal using a Hann window.
% 3. Find the autocorrelation fucntion of the windowed signal.
% 4. Window the autocorrealtion function.
% 5. Find the the group-delay of the windowed autocorrelation signal.
% 6. Find local minima in the thresholded group delay.
%-------------------------------------------------------------------------------
---------------------



%-----------------------------------------------------------------------
% Frequency and time axis
```

```
%-----------------------------------------------------------------------

N=length(signal);
frequency=[0:fs/FFT_LENGTH:fs/2];
time=[0:1/(fs):(N-1)/(fs)];

%-----------------------------------------------------------------------
% Make a copy of the original signal for plotting purposes
%-----------------------------------------------------------------------

original_signal=signal;

%-----------------------------------------------------------------------
% Spectrum of the original signal
%-----------------------------------------------------------------------

original_signal_spectrum=abs(fft(signal,FFT_LENGTH));
original_signal_spectrum=original_signal_spectrum(1:(FFT_LENGTH/2)+1);

if log_flag == 1
    original_signal_spectrum=20*log10(original_signal_spectrum);
end

%-----------------------------------------------------------------------
% Hanning Window
%-----------------------------------------------------------------------

M=round(time_cutoff_ms*fs);
win=zeros(2*N-1,1);
win(N-M+1:N+M-1)=window(@hann,2*M-1);
win=win(N:end);

%-----------------------------------------------------------------------
%   NOTE: CODE ADDED BY SOHUM DATTA
%-----------------------------------------------------------------------

win_signal=signal.*win; %Windowed signal (without LP analysis done)

%spectrum of the above signal
windowed_signal_spectrum=abs(fft(win_signal,FFT_LENGTH));
windowed_signal_spectrum=windowed_signal_spectrum(1:(FFT_LENGTH/2)+1);
windowed_signal_spectrum=20*log10(windowed_signal_spectrum);    %converting to
log



%-----------------------------------------------------------------------
% Linear Prediction Analysis
%-----------------------------------------------------------------------

if lp_flag == 1

    LP_coefficients=lpc(signal,LP_ORDER);
    LP_residual=real(filter(1,LP_coefficients',[1; zeros(N-1,1)]));
    signal=LP_residual;

end
```

```
%-------------------------------------------------------------------------
% Spectrum of the signal
%-------------------------------------------------------------------------

signal_spectrum=abs(fft(signal,FFT_LENGTH));
signal_spectrum=signal_spectrum(1:(FFT_LENGTH/2)+1);
%plot(frequency, signal_spectrum);                  %NOTE: UNCOMMENT THIS LINE
FOR PLOT
if log_flag == 1
    signal_spectrum=20*log10(signal_spectrum);
end

%-------------------------------------------------------------------------
% Window the signal
%-------------------------------------------------------------------------

win_signal=signal.*win;

%-------------------------------------------------------------------------
%Spectrum of the windowed signal
%-------------------------------------------------------------------------

win_signal_spectrum=abs(fft(win_signal,FFT_LENGTH));
win_signal_spectrum=win_signal_spectrum(1:(FFT_LENGTH/2)+1);

if log_flag == 1
    win_signal_spectrum=20*log10(win_signal_spectrum);
end

%-------------------------------------------------------------------------
% Autocorrelation of the windowed signal
%-------------------------------------------------------------------------

win_signal_acor=xcorr(win_signal);

%-------------------------------------------------------------------------
% Spectrum of the  autocorrelation signal
%-------------------------------------------------------------------------

win_signal_acor_spectrum=abs(fft(win_signal_acor,FFT_LENGTH));
win_signal_acor_spectrum=win_signal_acor_spectrum(1:(FFT_LENGTH/2)+1);

if log_flag == 1
   win_signal_acor_spectrum=20*log10(win_signal_acor_spectrum);
end

%-------------------------------------------------------------------------
% Autocorrelation window
%-------------------------------------------------------------------------

M=round(acor_time_cutoff_ms*fs);
awin=zeros(2*N-1,1);
awin(N-M+1:N+M-1)=window(@hann,2*M-1);

%-------------------------------------------------------------------------
```

```
% Window the autocorrelation fucntion
%-------------------------------------------------------------------------

win_signal_acor_win=(win_signal_acor.*awin);
awin=awin(N:end);

%-------------------------------------------------------------------------
% Spectrum of the windowed autocorrelation signal
%-------------------------------------------------------------------------

win_signal_acor_win_spectrum=abs(fft(win_signal_acor_win,FFT_LENGTH));
win_signal_acor_win_spectrum=win_signal_acor_win_spectrum(1:(FFT_LENGTH/2)+1);

if log_flag == 1
   win_signal_acor_win_spectrum=20*log10(win_signal_acor_win_spectrum);
end

%-------------------------------------------------------------------------
% Group delay of the windowed autocorrelation signal
%-------------------------------------------------------------------------

win_signal_acor_win=win_signal_acor_win(N:end);
win_signal_acor=win_signal_acor(N:end);

grp_delay=grpdelay(win_signal_acor_win,1,FFT_LENGTH,'whole');
grp_delay=grp_delay(1:(FFT_LENGTH/2)+1);

%-------------------------------------------------------------------------
% Zero threshold the group delay
%-------------------------------------------------------------------------
reso_threshold_grp_delay=grp_delay;
reso_threshold_grp_delay(find(reso_threshold_grp_delay < threshold))=0;

%-------------------------------------------------------------------------
% Find the local maxima in the group delay function
%-------------------------------------------------------------------------
temp = reso_threshold_grp_delay * (-1); %NOTE: code addition here: temporary
array as lmax dosent exist
[lmval,indd]=lmin(temp,0);              %NOTE: here argument 2 of lmin was
changed from 0 to 2

reso_frequencies_kHz=frequency(indd);
reso_frequencies_index=indd;

% This should be whose gain?? CHECK THIS PART
reso_gain = windowed_signal_spectrum(indd);     %gain from windowed and LP
analysis)

%-------------------------------------------------------------------------
% Return the intermediate results to the structure plots
%-------------------------------------------------------------------------

plots.frequency=frequency;
plots.time=time;
plots.original_signal=original_signal;
plots.original_signal_spectrum=original_signal_spectrum;
```

```
plots.signal=signal;
plots.signal_spectrum=signal_spectrum;
plots.win=win;
plots.win_signal=win_signal;
plots.win_signal_spectrum=windowed_signal_spectrum;          % NOTE: change
'windowed' to 'win'
plots.win_signal_acor=win_signal_acor;
plots.win_signal_acor_spectrum=win_signal_acor_spectrum;
plots.awin=awin;
plots.win_signal_acor_win=win_signal_acor_win;
plots.win_signal_acor_win_spectrum=win_signal_acor_win_spectrum;
plots.grp_delay=grp_delay;
plots.reso_threshold_grp_delay=reso_threshold_grp_delay;


if plot_flag==1

    figure;
    K=6;
    L=2;

    %----------------------------------------

    subplot(K,L,1);
    plot(time,original_signal);
    axis([time(1) time(end) -1 1]);
    h=ylabel('(a)');
    set(h,'rotation',0);
    legend('Signal');


   %----------------------------------------

    subplot(K,L,3);
    plot(time,signal);
    axis([time(1) time(end) -1 1]);
   hold on;
    plot(time,win*max(signal),'r:');
    h=ylabel('(b)');
    set(h,'rotation',0);
    legend('LP residual');


    %----------------------------------------

    subplot(K,L,5);
    plot(time,win_signal);
    axis([time(1) time(end) -1 1]);
    h=ylabel('(c)');
    set(h,'rotation',0);
    legend('Windowed LP residual');


    %----------------------------------------

    subplot(K,L,7);
```

```
    plot(time,win_signal_acor);
    axis([time(1) time(end) -1 1]);
    hold on;
    plot(time,awin*max(win_signal_acor),'r:');
        h=ylabel('(d)');
       set(h,'rotation',0);
           legend('Autocorrelation');


    %----------------------------------------

    subplot(K,L,9);
    plot(time,win_signal_acor_win);
    axis([time(1) time(end) -1 1]);
    xlabel('Time (ms)');
        h=ylabel('(e)');
       set(h,'rotation',0);
    legend('Windowed Autocorrelation');


    %----------------------------------------

    subplot(K,L,2);
    plot(frequency,original_signal_spectrum);
    axis([frequency(1) frequency(end) min(original_signal_spectrum)
max(original_signal_spectrum)]);
        h=ylabel('(f)');
       set(h,'rotation',0);
    title('Corresponding Spectrum Magnitude');

    %----------------------------------------

    subplot(K,L,4);
    plot(frequency,signal_spectrum);
    axis([frequency(1) frequency(end) min(signal_spectrum)
max(signal_spectrum)]);
        h=ylabel('(g)');
       set(h,'rotation',0);


    %----------------------------------------

    subplot(K,L,6);
    plot(frequency,windowed_signal_spectrum);           %NOTE: CHANGE
'windowed' to 'win'
    axis([frequency(1) frequency(end) min(windowed_signal_spectrum)
max(windowed_signal_spectrum)]);
        h=ylabel('(h)');
       set(h,'rotation',0);


    %----------------------------------------

    subplot(K,L,8);
    plot(frequency,win_signal_acor_spectrum);
```

```
   axis([frequency(1) frequency(end) min(win_signal_acor_spectrum)
max(win_signal_acor_spectrum)]);
       h=ylabel('(i)');
       set(h,'rotation',0);


    %----------------------------------------

    subplot(K,L,10);
    plot(frequency,win_signal_acor_win_spectrum);
    h=ylabel('dB');
    axis([frequency(1) frequency(end) min(win_signal_acor_win_spectrum)
max(win_signal_acor_win_spectrum)]);
        h=ylabel('(j)');
             set(h,'rotation',0);


    %----------------------------------------
    hold on;
    subplot(K,L,12);
    plot(frequency,grp_delay);
    xlabel('Frequency (kHz)');
    hold on;
    axis([frequency(1) frequency(end) min(grp_delay) max(grp_delay) ]);
    hold on;
    plot(reso_frequencies_kHz,grp_delay(reso_frequencies_index),'r.');
    h=ylabel('(k)');
    hold on;
    plot(frequency,threshold*ones(size(frequency)),'b:');
    set(h,'rotation',0);
    title('Group Delay');

end
```

7.4 lmin.m

```
function [lmval,indd]=lmin(xx,filt)

%LMIN   function [lmval,indd]=lmin(x,filt)
%   Find local minima in vector X, where LMVAL is the output
%   vector with minima values, INDD is the corresponding indeces
%   FILT is the number of passes of the small running average filter
%   in order to get rid of small peaks.  Default value FILT =0 (no
%   filtering). FILT in the range from 1 to 3 is usially sufficient to
%   remove most of a small peaks
%   Examples:
%   xx=0:0.01:35; y=sin(xx) + cos(xx ./3);
%   plot(xx,y); grid; hold on;
%   [a b]=lmin(y,2)
%    plot(xx(a),y(a),'r+')
%   see also LMAX, MAX, MIN
%
%*************************************************|
%   Serge Koptenko, Guigne International Ltd.,     |
%   phone (709)895-3819, fax (709)895-3822         |
%--------------06/03/97----------------------------|

x=xx;
len_x = length(x);
    fltr=[1 1 1]/3;
  if nargin <2, filt=0;
    else
x1=x(1); x2=x(len_x);

    for jj=1:filt,
    c=conv(fltr,x);
    x=c(2:len_x+1);
    x(1)=x1;
        x(len_x)=x2;
    end
  end

lmval=[];
indd=[];
i=2;        % start at second data point in time series

    while i < len_x-1,
    if x(i) < x(i-1)
       if x(i) < x(i+1) % definite min
lmval =[lmval x(i)];
indd = [ indd i];

       elseif x(i)==x(i+1)&x(i)==x(i+2) % 'long' flat spot
%lmval =[lmval x(i)];   %1   comment these two lines for strict case
%indd = [ indd i];  %2 when only  definite min included
i = i + 2;          % skip 2 points

       elseif x(i)==x(i+1)  % 'short' flat spot
%lmval =[lmval x(i)];   %1   comment these two lines for strict case
%indd = [ indd i];  %2 when only  definite min included
```

```
i = i + 1;      % skip one point
        end
    end
    i = i + 1;
    end

if filt>0 & ~isempty(indd),
    if (indd(1)<= 3)|(indd(length(indd))+2>length(xx)),
       rng=1;   %check if index too close to the edge
    else rng=2;
    end

       for ii=1:length(indd),
        [val(ii) iind(ii)] = min(xx(indd(ii) -rng:indd(ii) +rng));
        iind(ii)=indd(ii) + iind(ii)  -rng-1;
       end
  indd=iind; lmval=val;
else
end

[lmval,index]=sort(lmval);
indd=indd(index);
```

7.5 get_CIPIC_HRIR.m

```
function [HRIR,HRTF_mag,HRTF_phase,time,frequency]=...
    get_CIPIC_HRIR(subject_id,...
             pinna,...
             elev,...
             azim,...
             freq_kHz,...
             FFT_LENGTH,...
             log_enable,...
             diff_enable,...
             plot_enable)

%-------------------------------------------------------------------------------
---------------------
% This function returns the Head Related Impulse Response (HRIR) and the
% Head Related Transfer Function (HRTF) of a particular pinna for a given
% elevation and azimuth in the CIPIC HRIR database.
%-------------------------------------------------------------------------------
---------------------
% EXAMPLE USAGE:
%
%
[HRIR,HRTF_mag,HRTF_phase,time,frequency]=get_HRTF(10,'right',0,0,22.05,1024,1,
0,1);
%-------------------------------------------------------------------------------
---------------------
% INPUT : subject_id  : subject id number
%         pinna       : 'left' or 'right'
%         elev        : elevation in degrees   [ 72 elevations (-90:5:265) ]
%         azim        : azimuth in degrees     [ 19 azimuths (0:5:90) ]
%         freq_kHz    : Upper frequency cutoff [ Upper limit is 22.05 kHz]
%         FFT_LENGTH  : length of the FFT in samples
%         log_enable  : if 1 the HRTF magnitude is in dB
%         diff_enable : if 1 diff operation is enabled (removes any DC if
present)
%         plot_enable : if 1 the HRIR and the HRTF are plotted
%-------------------------------------------------------------------------------
---------------------
% OUTPUT : HRIR       : Pinna Related Impulse Response
%          HRTF_mag   : Pinna Related Transfer Function Magnitude
%          HRTF_phase : Pinna Related Transfer Function Phase
%          time       : time axis in milliseconds
%          frequency  : frequency axis in kHz
%-------------------------------------------------------------------------------
---------------------
% Author     :     Vikas.C.Raykar
% Date       :     25 May 2004
% Contact    :     vikas@umiacs.umd.edu
%-------------------------------------------------------------------------------
---------------------
% NOTE: Make sure you set the path to the CIPIC database in
CIPIC_database_path. The CIPIC database
%       can be downloaded from
%       http://interface.cipic.ucdavis.edu/CIL_html/CIL_HRTF_database.htm
%-------------------------------------------------------------------------------
---------------------
```

```
ids=[3,8,9,10,11,12,15,17,18,19,20,21,27,28,33,40,44,48,50,51,58,59,60,61,65,11
9,124,126,127,131,133,134,135,137,147,148,152,153,154,155,156,158,162,163,165]
;
azimuth=[-80 -65 -55 -45:5:45 55 65 80];
elevation=-45+5.625*(0:49);
fs=44100; %Sampling rate

%Check whether the id is valid or not
if isempty(find(ids==subject_id)) == 1
    disp('NOT A VALID ID');
    disp('FOLLOWING ARE THE VALID IDS');
    disp(ids);
    return;
end

%Check whether it is a valid azimuth or not
azimuth_index=find(azimuth==azim);

if isempty(azimuth_index) == 1
    disp('NOT A VALID AZIMUTH');
    disp('FOLLOWING ARE THE VALID AZIMUTHS');
    disp(azimuth);
    return;
end

%Check whether it is a valid elevation or not
elevation_index=find(elevation==elev);

if isempty(elevation_index) == 1
    disp('NOT A VALID ELEVATION');
    disp('FOLLOWING ARE THE VALID ELEVATIONS');
    disp(elevation);
    return;
end

%Load the HRIR for given subject

database_path=CIPIC_database_path;
pinna_path=sprintf('standard_hrir_database\\subject_%03d\\hrir_final.mat',subje
ct_id);
filename=strcat(database_path,pinna_path);
load(filename);

%load the HRIR for the given elevation and azimuth
if strcmp(pinna,'left')==1
    HRIR=squeeze(hrir_l(azimuth_index,elevation_index,:));
    %temp=round(OnL(azimuth_index,elevation_index));
    %HRIR=[HRIR(temp+67:end)];
end

if strcmp(pinna,'right')==1
    HRIR=squeeze(hrir_r(azimuth_index,elevation_index,:));
end

% Downsample

HRIR=resample(HRIR,round(2*freq_kHz*1000),fs);
```

```
if diff_enable==1
   HRIR=[diff(HRIR); 0];
end

%Compute the HRTF magnitude
HRTF_mag=abs(fft(HRIR,FFT_LENGTH));

if log_enable==1
    HRTF_mag =20*log10(HRTF_mag);
end

%phase
HRTF_phase=unwrap(angle(fft(HRIR,FFT_LENGTH)'));

%Return only  from 0 to pi
HRTF_mag=HRTF_mag(1:(FFT_LENGTH/2)+1);
HRTF_phase=HRTF_phase(1:(FFT_LENGTH/2)+1)';

frequency=[0:(freq_kHz*2)/FFT_LENGTH:freq_kHz];
time=[0:1/(2*freq_kHz):(max(size(HRIR))-1)/(2*freq_kHz)];

if plot_enable==1

    figure;
    subplot(3,1,1);
    plot(time,HRIR);
    hold on;
    title(sprintf('HRIR and HRTF for Subject %d %s pinna elevation %f azimuth
%f',subject_id,pinna,elev,azim));
    xlabel('time in ms');
    ylabel('HRIR');
    grid on;

    subplot(3,1,2);
    plot(frequency,HRTF_mag);
    xlabel('frequency in kHz');
    if log_enable==1
        ylabel('HRTF (dB)');
    else
        ylabel('HRTF Magnitude');
    end
    grid on;

    subplot(3,1,3);
    plot(frequency,HRTF_phase);
    xlabel('frequency in kHz');
    ylabel('Phase (radians)');
    grid on;
end
```

7.6 get_CIPIC_HRIR_onset.m

```
function [onset]=get_CIPIC_HRIR_onset(subject_id,pinna,elev,azim,freq_kHz)

%---------------------------------------------------------------------------
---------------------
% This function returns the onset time of the Head Related Impulse Response
(HRIR) of a particular
% pinna for a given elevation and azimuth in the CIPIC HRIR database.
%---------------------------------------------------------------------------
---------------------
% EXAMPLE USAGE:
%
% [onset]=get_CIPIC_HRIR_onset(10,'right',0,0,22.05)
%---------------------------------------------------------------------------
---------------------
% INPUT : subject_id  : subject id number
%         pinna       : 'left' or 'right'
%         elev        : elevation in degrees   [ 72 elevations (-90:5:265) ]
%         azim        : azimuth in degrees     [ 19 azimuths (0:5:90) ]
%         freq_kHz    : Upper frequency cutoff [ Upper limit is 22.05 kHz]
%---------------------------------------------------------------------------
---------------------
% OUTPUT : onset      : onset time in number of samples [use HRIR(onset:end)]
%---------------------------------------------------------------------------
---------------------
% Author     :     Vikas.C.Raykar
% Date       :     25 May 2004
% Contact    :     vikas@umiacs.umd.edu
%---------------------------------------------------------------------------
---------------------
% NOTE: Make sure you set the path to the CIPIC database in
CIPIC_database_path. The CIPIC database
%       can be downloaded from
%       http://interface.cipic.ucdavis.edu/CIL_html/CIL_HRTF_database.htm
%---------------------------------------------------------------------------
---------------------


ids=[3,8,9,10,11,12,15,17,18,19,20,21,27,28,33,40,44,48,50,51,58,59,60,61,65,11
9,124,126,127,131,133,134,135,137,147,148,152,153,154,155,156,158,162,163,165]
;
azimuth=[-80 -65 -55 -45:5:45 55 65 80];
elevation=-45+5.625*(0:49);

%Check whether the id is valid or not
if isempty(find(ids==subject_id)) == 1
    disp('NOT A VALID ID');
    disp('FOLLOWING ARE THE VALID IDS');
    disp(ids);
    return;
end

%Check whether it is a valid azimuth or not
azimuth_index=find(azimuth==azim);

if isempty(azimuth_index) == 1
```

```
    disp('NOT A VALID AZIMUTH');
    disp('FOLLOWING ARE THE VALID AZIMUTHS');
    disp(azimuth);
    return;
end

%Check whether it is a valid elevation or not
elevation_index=find(elevation==elev);

if isempty(elevation_index) == 1
    disp('NOT A VALID ELEVATION');
    disp('FOLLOWING ARE THE VALID ELEVATIONS');
    disp(elevation);
    return;
end

%Load the HRIR for given subject

database_path=CIPIC_database_path;
pinna_path=sprintf('standard_hrir_database\\subject_%03d\\hrir_final.mat',subje
ct_id);
filename=strcat(database_path,pinna_path);
load(filename);

if strcmp(pinna,'left')==1
    onset=round(OnL(azimuth_index,elevation_index)*freq_kHz*2/44.1);
end

if strcmp(pinna,'right')==1
    onset=round(OnR(azimuth_index,elevation_index)*freq_kHz*2/44.1);
end
```

7.7 find_bandwidth.m

```
function bandwidth = find_bandwidth(freq_ind, freq, spectrum, flag);
% DESCRIPTION :
%-------------------------------------------------------------------------------
---------------------
% This function returns 3dB bandwidth of a given index from a given array of
frequencies and
% corresponding amplitude spectrum.
%-------------------------------------------------------------------------------
---------------------
% Author     :     Sohum Datta
%
%-------------------------------------------------------------------------------
---------------------
% INPUTS :
% 1. freq_ind:
%       A frequency index from array 'freq' whose bandwidth to be returned.
%
% 2. freq:
%       array of frequencies (data).
%
% 3. spectrum:
%       spectrum corresponding to array freq (in dB).
%
% 4. flag:
%       flag == 0, bandwidth of notch (flag == 1, then that of a resonance).
%% OUTPUT :
%-------------------------------------------------------------------------------
---------------------
% 1. bandwidth:
%       the 3dB bandwidth of the frequency corresponding to 'freq_ind' in array
'freq'.
%
%*******************************************************************************
********************


N = length(freq);
i = freq_ind;
if flag == 0        %i.e. bandwidth of notch
    threshold = spectrum(i) + 3;
    while ((i > 1) && (spectrum(i) <= threshold))
        i = i - 1;
    end
    low = freq(i);  % lower end of Bandwidth
    j = freq_ind;
    while ((j < N) && (spectrum(j) <= threshold))
        j = j + 1;
    end
    high = freq(j);
end
if flag == 1                %i.e. bandwidth of resonance
    threshold = spectrum(i) - 3;
    while ((i > 1) && (spectrum(i) >= threshold))
        i = i - 1;
    end
    low = freq(i);  % lower end of Bandwidth
```

```
    j = freq_ind;
    while ((j < N) && (spectrum(j) >= threshold))
        j = j + 1;
    end
    high = freq(j);
end
bandwidth = high - low;
```

7.8 refine_notch.m

```
function [notch_ind, notch_gain, notch_bandwidth, notch_length] =
refine_notch(zero_frequencies_index,  zero_frequencies_kHz, zero_gain,
frequency, spectrum);
% DESCRIPTION :
%-------------------------------------------------------------------------
---------------------
% This function returns the indices, gains and number of pinna spectral notchs
found in a given
% frequency range. Specifically, best (at-most NUM1) notches between LOW1 -
HIGH1 kHz, and
% (at-most NUM2) notches between LOW2 - HIGH2 kHz (variables LOW1, LOW2, HIGH1,
HIGH2 defined later)
% are returned.
%-------------------------------------------------------------------------
---------------------
% Author     :     Sohum Datta
%
% INPUTS :
%-------------------------------------------------------------------------
---------------------
% 1. zero_frequencies_index:
%       The indices of notch frequencies from array 'frequency' (see later)
%
% 2. zero_frequencies_kHz:
%       The frequencies (in KHz) of the notches to be checked
%
% 3. zero_gain:
%       The gain (dB) of the notch frequencies
%
%
%-------------------------------------------------------------------------
---------------------
% OUTPUTS :
% 1. notch_ind:
%       Stores the Index (in 'zero_frequencies_kHz') of the notches found to be
in required range.
%
% 2. notch_gain:
%       Stores the corresponding Gain(in dB)
%
% 3. notch_bandwidth:
%       Returns the 3dB Bandwidth of the corresponding notches.
%
% 4. notch_length:
%       Denotes the number of notches that lie in the desired range.
%
%*************************************************************************
*******************



% ------------------------------------------------------------------------
% SET THE DESIRED RANGE
% ------------------------------------------------------------------------
% WARNING: Ensure HIGH >= LOW > 0, NUM > 0 for proper functioning.

LOW = 4;
```

```
HIGH = 14;
NUM = 3;

% ------------------------------------------------------------------------
% Computing Number of notches present in desired range
% ------------------------------------------------------------------------

%if no notches found, end
if(sum([zero_gain <= 0]) == 0)
    display('Bad Data: No notches Found. Terminating...');
    quit;   % End current session
end

%Sort Gain array in ascending order.
sorted_gain = sort(zero_gain, 'ascend');    %check this line if its working

% save the indices of frequencies of 'zero_frequencies_kHz'
% corresponding to sorted_gain.
sorted_ind = zeros(1, length(sorted_gain));
for i = 1: length(sorted_gain)
    sorted_ind(i) = find(zero_gain == sorted_gain(i));
end

% For each position in zero_frequencies_kHz, the corresponding position in
freq_inrange is 1 iff
% that frequency lies in range and its gain is non-negative.
freq_inrange = ([zero_frequencies_kHz <= HIGH] & [LOW <= zero_frequencies_kHz]
& [zero_gain < 0]');
notch_length = sum(freq_inrange);       % this is the no. of notches found in
required range

% if no notches found in range, warn the users
if(notch_length == 0)
    display('CAUTION: No notches found in the required ranges. Skipping the
notch block entirely.');
    notch_ind = 0;
    notch_gain = 0;
    notch_bandwidth = 0;
    return;     %return to the main script
end

% Take best NUM notches if more than NUM found.
if(notch_length > NUM)
    notch_length = NUM;
end

% ------------------------------------------------------------------------
% Generating parameters for each notch found in range
% ------------------------------------------------------------------------

% initialize the outputs arrays to size
notch_ind = zeros(1, notch_length);
notch_gain = zeros(1, notch_length);
notch_bandwidth = zeros(1, notch_length);

j = 1;      %iterator for the next position of o/p arrays to be filled
```

```
% iterate over all notch frequencies, and increment j if it belongs
% in range. Put the corresponding parameters in the output array.
for i=1:length(sorted_gain)
    if(freq_inrange(sorted_ind(i)))
        notch_ind(j) = zero_frequencies_index(sorted_ind(i));
        notch_gain(j) = zero_gain(sorted_ind(i));
        notch_bandwidth(j) = find_bandwidth(notch_ind(j),frequency, spectrum,
0);
        j = j + 1;
    end
end


%-------------------------------------------------------------------------------
-------------------------
%   End of Notch Extraction Block
%-------------------------------------------------------------------------------
-------------------------
```

7.9 refine_resonance.m

```
function [resonance_ind, resonance_gain, resonance_length] =
refine_resonance(reso_frequencies_index,  reso_frequencies_kHz, reso_gain);
% DESCRIPTION :
%-------------------------------------------------------------------------------
---------------------
% This function returns the indices, gains and number of pinna spectral
resonances found in a given
% frequency range. Specifically, best (at-most NUM1) resonances between LOW1 -
HIGH1 kHz, and
% (at-most NUM2) resonances between LOW2 - HIGH2 kHz (variables LOW1, LOW2,
HIGH1, HIGH2 defined later)
% are returned.
%-------------------------------------------------------------------------------
---------------------
% Author     :     Sohum Datta
%
% INPUTS :
%-------------------------------------------------------------------------------
---------------------
% 1. reso_frequencies_index:
%       The indices of resonance frequencies from array 'frequency' (see later)
%
% 2. reso_frequencies_kHz:
%       The frequencies (in KHz) of the notches to be checked
%
% 3. reso_gain:
%       The gain (dB) of the notch frequencies
%
%
%-------------------------------------------------------------------------------
---------------------
% OUTPUTS :
% 1. resonance_ind:
%       Stores the Index (in 'reso_frequencies_kHz') of the resonances found to
be in required range.
%
% 2. resonance_gain:
%       Stores the corresponding Gain(in dB)
%
% 3. resonance_length:
%       Denotes the number of resonances that lie in the desired range.
%
%*******************************************************************************
*********************



% ------------------------------------------------------------------------
% SET THE DESIRED RANGE
% ------------------------------------------------------------------------
% WARNING: Ensure HIGH2 >= LOW2 >= HIGH1 >= LOW1 > 0; NUM1, NUM2 > 0 for
% proper functioning.

%range1
LOW1 = 3;
HIGH1 = 8;
```

```
NUM1 = 2;

%range2
LOW2 = 12;
HIGH2 = 18;
NUM2 = 2;

% ------------------------------------------------------------------------
% Computing Number of resonances present in desired range
% ------------------------------------------------------------------------

%if no resonances found, end
if(sum([reso_gain >= 0]) == 0)
    display('Bad Data: No resonances Found. Terminating...');
    quit;   % End current session
end

%Sort Gain array in descending order.
sorted_gain = sort(reso_gain, 'descend');   %check this line if its working

% save the indices of frequencies of 'reso_frequencies_kHz'
% corresponding to sorted_gain.
sorted_ind = zeros(1, length(sorted_gain));
for i = 1: length(sorted_gain)
    sorted_ind(i) = find(reso_gain == sorted_gain(i));
end

% For each position in reso_frequencies_kHz, the corresponding position in
freq_inrange1 is 1 iff
% that frequency lies in range and its gain is non-negative.
freq_inrange1 = ([reso_frequencies_kHz <= HIGH1] & [LOW1 <=
reso_frequencies_kHz]) & [reso_gain >= 0]' ;
resonance_length1 = sum(freq_inrange1);     % this is the no. of resonances
found in required range1

%same as above for range2
freq_inrange2 = ([reso_frequencies_kHz <= HIGH2] & [LOW2 <=
reso_frequencies_kHz]) & [reso_gain >= 0]' ;
resonance_length2 = sum(freq_inrange2);     % this is the no. of resonancees
found in required range2

% Take the max. no. of resonances possible if more than required found.
if(resonance_length1 > NUM1)
    resonance_length1 = NUM1;
end
if(resonance_length2 > NUM2)
    resonance_length2 = NUM2;
end

% total no. of resonances found in required range
resonance_length = resonance_length1 + resonance_length2;

% if no resonances found in range1, warn the users
if(resonance_length == 0)
    display('CAUTION: No resonances found in the required ranges. Skipping the
resonance block entirely.');
    resonance_ind = 0; resonance_gain = 0;
```

```
    return;     %return to the main script
end


% ------------------------------------------------------------------------
% Generating parameters for each resonance found in range
% ------------------------------------------------------------------------

% initialize the outputs arrays to size
resonance_ind = zeros(1, resonance_length);
resonance_gain = zeros(1, resonance_length);

j = 1;  %iterator for the next position of o/p arrays to be filled
k1 = 1; %iterator for no. of notches encountered in range1
k2 = 1; %iterator for no. of notches encountered in range2

% iterate over all resonance frequencies, and increment k1 or k2
% depending on which range it belongs. Put the corresponding parameters
% in the output array.

for i=1:length(reso_gain)
    if(freq_inrange1(sorted_ind(i))) && (k1 <= resonance_length1)
        resonance_ind(j) = reso_frequencies_index(sorted_ind(i));
        resonance_gain(j) = reso_gain(sorted_ind(i));
        j = j + 1;
        k1 = k1 + 1;
    end
    if(freq_inrange2(sorted_ind(i))) && (k2 <= resonance_length2)
        resonance_ind(j) = reso_frequencies_index(sorted_ind(i));
        resonance_gain(j) = reso_gain(sorted_ind(i));
        j = j + 1;
        k2 = k2 + 1;
    end
end


display('Resonances extracted successfully.');
%-------------------------------------------------------------------------
------------------------
%   End of Resonance Extraction Block
%-------------------------------------------------------------------------
-------------------------
```

7.10 CIPIC_database_path.m

```
function [database_path] = CIPIC_database_path()

%-------------------------------------------------------------------------------
---------------------
% This is the diectory which contains the CIPIC HRIR database.
% Modify this to point to your database.
% The CIPIC database can be downloaded from
% http://interface.cipic.ucdavis.edu/CIL_html/CIL_HRTF_database.htm
%-------------------------------------------------------------------------------
---------------------
% Author     :     Vikas.C.Raykar
% Date       :     29 July 2004
% Contact    :     vikas@umiacs.umd.edu
%-------------------------------------------------------------------------------
---------------------


database_path=sprintf('CIPIC_hrtf_database\\');
```

7.11 readme.txt

```
*****README FIRST*****
EE301A Term Project TP18 (Indian Institute of Technology Kanpur)
Session: 2013-14
Group 18

Author     :   Aditya Jain (11037), Anirudh Agrawal (11098)
          Ishendra Agarwal (11324), Sohum Datta (11724)


-----------------------------------------------------------------
 REFERENCE :
-----------------------------------------------------------------

 Title of reeference paper:
     "Structural Modelling of Pinna-Related Transfer Functions
 Authors (of Ref. paper): S. Spagnol, M. Geronazzo, F. Avanzini

-----------------------------------------------------------------
 DIRECTORY CONTENTS :
-----------------------------------------------------------------

*Main Script to open in MATLAB: script_tp18.m*

readme.txt
script_tp18.m
Algorithm_1.m
Algorithm_2.m
lmin.m
CIPIC_database_path.m
get_CIPIC_HRIR.m
get_CIPIC_HRIR_onset.m
CIPIC_hrtf_database [Folder containing *ENTIRE* CIPIC database]
find_bandwidth.m
refine_notch.m
refine_resonance.m
```