Lab 05: Linear Phase, Critical Band Smoothing, Warped Filter Design

Ron Guglielmone

PROVIDED IMPULSE RESPONSES

Two guitar cabinet impulse responses were provided; both from a Jensen speaker through an SM57 microphone. One was mic'd on-axis and the other off-axis (Figure 1).

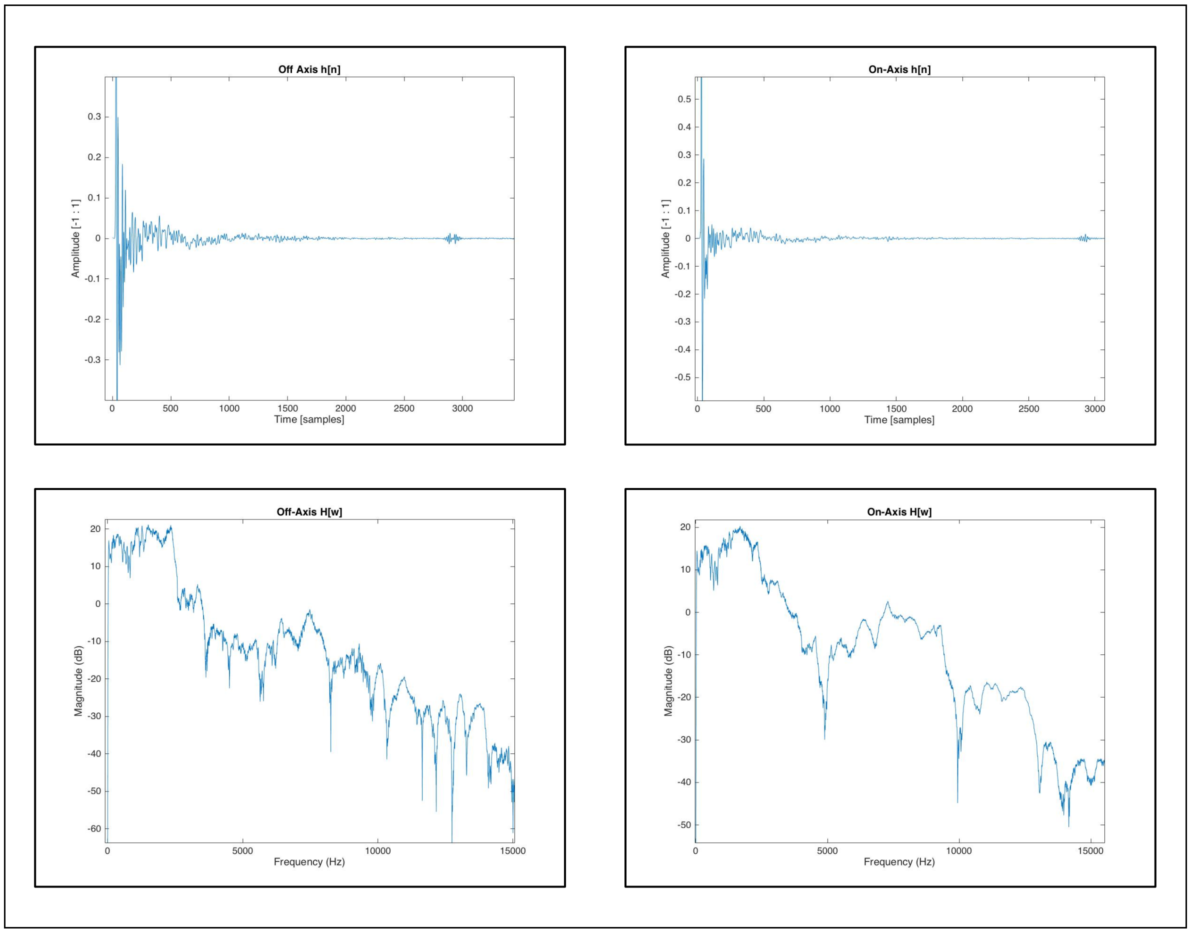

Figure 1, off-axis (left) and on-axis (right) impulse responses and spectra as provided.

For the duration of this assignment, I will be using the on-axis file ('Jen57On.wav').

PROBLEM 1, PART A: LINEAR PHASE FUNCTION

A small MATLAB script was written to convert the impulse response to a minimum-phase version. The analysis from page 281 of our course reader was used as a guide:

$$h_{\text{lin}}(n) = \mathcal{F}^{-1}\left\{e^{-j\tau\omega} \cdot |H(\omega)|\right\}.$$

This resulted in the following MATLAB code (Figure 2).

```
% Problem 1 - Part A

function LinPhaseIR = tf2LinPhaseIR(tf, L)

% tf is a magnitude response |H(z)|
% L is the length of its impulse response h[n]

w = [ 0 : L-1 ]*2*pi/L;
tau = L/16;
LinPhaseIR = real(ifft(exp(-1i*tau*w).*abs(tf)));
LinPhaseIR = LinPhaseIR( 1 : L/2 );

end
```

Figure 2, MATLAB code for problem 1(a).

PROBLEM 1, PART B: LINEAR PHASE CONVERSION

The on-axis impulse response ('Jen57On.wav') was converted to a linear-phase version using the function above. This resulted in the following MATLAB code (Figure 3).

```
% Problem 1 - Part B

% Read file, get length:
[h, fs] = audioread('Jen57On.wav');
L = length(h);

% Convert to mag:
h = h/max(abs(h));
H = fft(h,L);

% Convert to linear phase:
linPhase = tf2LinPhaseIR(H,L);

% Get new mag:
linPhase = linPhase /max(abs(linPhase));
H1 = fft(linPhase);
```

Figure 3, MATLAB for 1(b).

The resulting impulse response and magnitude spectrum were plotted (Figure 4).

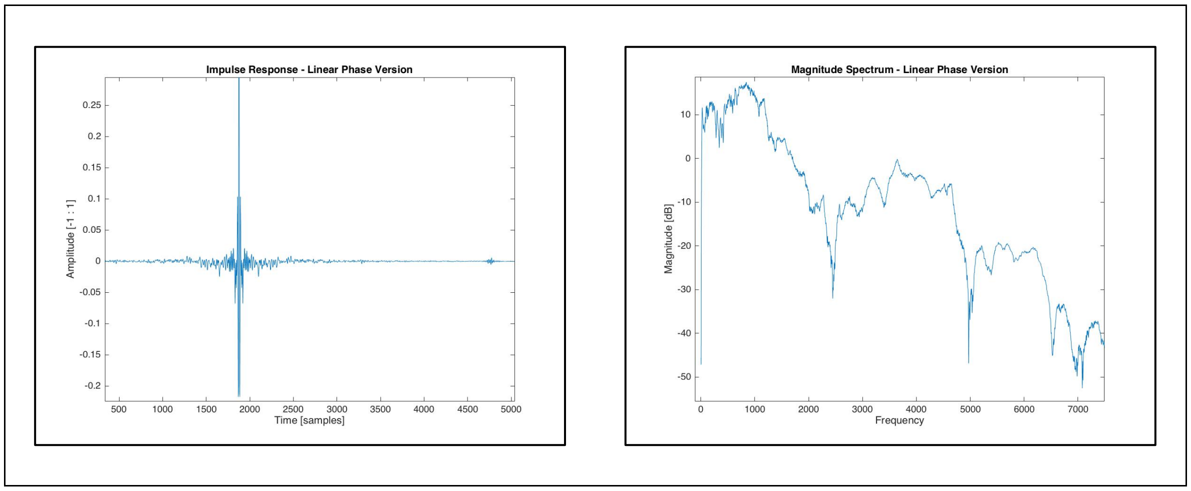

Figure 4, impulse response and magnitude spectrum for linear phase version of h[n].

Comparing these to Figure 1, we can see that the magnitude spectrum has stayed the same (as desired), but the impulse response has become more symmetric, and has a shape similar to a shifted non-causal filter.

PROBLEM 1, PART C: DIFFERENCE BETWEEN FILTERS

The linear phase version sounds fuzzier in the midrange frequencies. There is more roundness to the tone, whereas the original filter sounds comparably crisp and direct. These differences must be due to phase, as the frequency responses match.

The code used to implement this filtering is shown in Figure 5.

```
% Read files:
[h, fs] = audioread('Jen57On.wav');
h = h/max(abs(h));
[x, fs] = audioread('ElecGtr-Preflex-Dirty-dry.wav');
x = x/max(abs(x));

% Convert to mag:
h = h/max(abs(h));
L = length(h);
H = fft(h,L);

% Convert to LP:
linH = tf2LinPhaseIR(H,L);

% Filter the input:
% y = fftfilt(linH,x);
y = fftfilt(h,x);
y = y/max(abs(y));
```

Figure 5, code for problem 1(c).

PROBLEM 2, PART A: MINIMUM PHASE FUNCTION

Page 292 of the course reader provides an outline for implementing minimum phase conversion as follows:

$$h_{\min} = \mathcal{F}^{-1}\left\{\exp\left[\log|H(\omega)| - j\mathcal{H}\{\log|H(\omega)|\}\right]\right\}.$$

This resulted in the following MATLAB function (Figure 6).

```
function MinPhaseIR = tf2MinPhaseIR(tf, L)

% tf is a magnitude response |H(z)|
% L is the length of its impulse response h[n]

MinPhaseIR = real(ifft(conj(exp(hilbert(log(abs(tf')))))));
MinPhaseIR = MinPhaseIR(1:L);

end
```

Figure 6, MATLAB code for problem 2(a).

PROBLEM 2, PART B: MINIMUM PHASE CONVERSION

The following MATLAB script uses the function above to convert the measured impulse response to minimum phase (Figure 7).

```
% Read file:
[h, fs] = audioread('Jen57On.wav');

% Get length:
L = length(h);

% Convert to mag:
h = h/max(abs(h));
H = fft(h,L);

% Convert to LP:
minPhase = tf2MinPhaseIR(H,L);

% Get new mag:
minPhase = minPhase /max(abs(minPhase));
H1 = fft(minPhase);
```

Figure 7, MATLAB code for problem 2(b).

The resulting minimum phase impulse response and its spectrum are plotted in Figure 8.

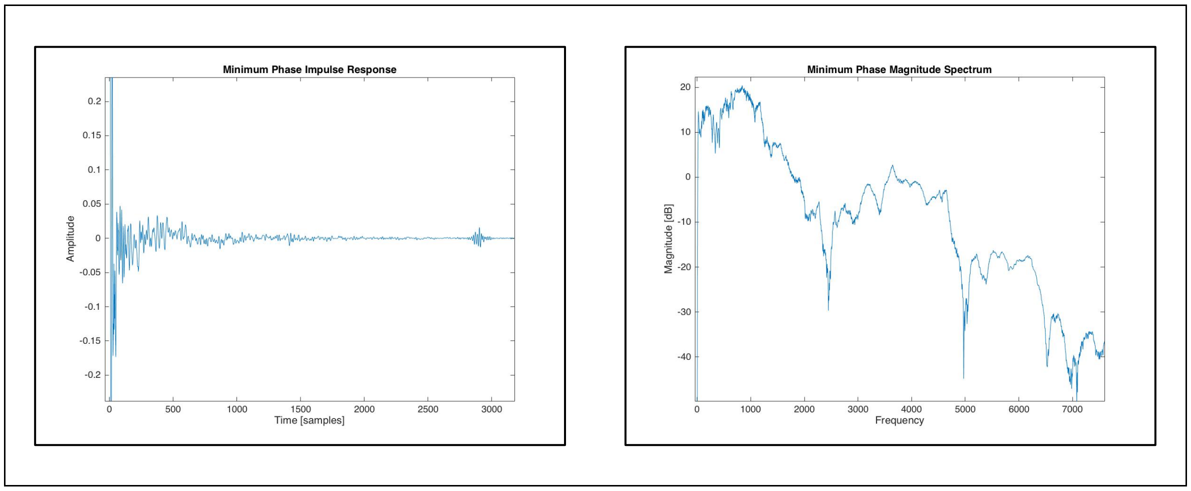

Figure 8, impulse response and spectrum for minimum phase filter.

Comparing these to the original filter, we notice differences in the impulse response, but a perfect match in the magnitude response (Figure 9).

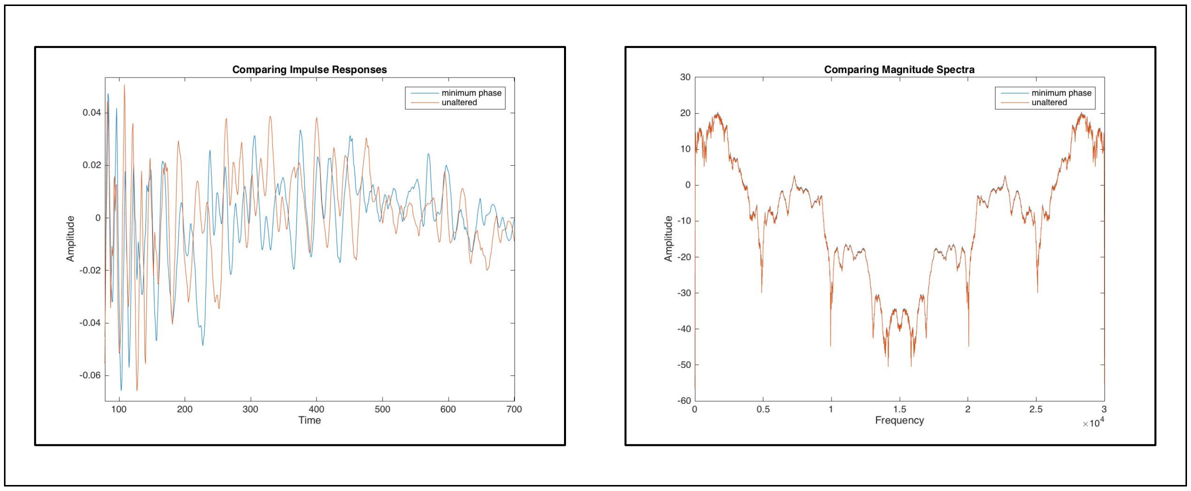

Figure 9, comparing the original filter to the minimum phase version.

PROBLEM 2, PART C: DIFFERENCE BETWEEN FILTERS

Although they are very similar, the phase-altered filter exhibits a bit less clarity in the mid-range frequencies. It has a somewhat fuzzier, thicker, less crisply defined presence.

PROBLEM 3, PART A: CRITICAL-BAND SMOOTHING FUNCTION

A script was written to implement critical-band smoothing as described on page 277 of the course reader. First, each bark bandwidth is calculated using the provided function khz2erb( ), then lower and upper boundaries are calculated for the band using erb2khz ( ). Finally, an average of the magnitude response over that bandwidth is calculated.

$$P(\omega;\beta) = \sum_{\varphi=f(b(\omega)-\beta/2)}^{f(b(\omega)+\beta/2)} |H(\varphi)|^2.$$

The MATLAB function as implemented is included in Figure 10 below.

```
function smoothed_out = cbsmooth(tf,beta,N)

% Constants:
fs = 44100;
halfFs = fs/2;
HsQ = abs(tf(1:(N/2) + 1)).^2;

% Pre-allocate:
Out = zeros(size(HsQ)+1);

% Main averaging loop:
for i = 1 : (N/2) + 1

  % Center of the process block:
  centerHz = i * halfFs / (N/2) / 1000;
  % Convert to Erb:
  centerBark = khz2erb(centerHz);
  % Define lower bound:
  lowBound = erb2khz(centerBark - beta/2);
  % Round lower bound:
  floor = max(1, round(lowBound*1000*(N/2)/halfFs));
  % Define upper bound:
  highBound = erb2khz(centerBark + beta/2);
  % Round upper bound:
  ceiling = min(N/2, round(highBound*1000*(N/2)/halfFs));
  % Average between two bounds:
  Out(i) = mean(HsQ(floor:ceiling));

end

% Return output:
smoothed_out = Out;

end
```

Figure 10, MATLAB code for problem 3(a).

PROBLEM 3, PART B: CRITICAL-BAND SMOOTHING CONVERSION

The following MATLAB script (Figure 11) was written to smooth the impulse response with values of beta = [0.5, 1, 2, 5] using the function cbsmooth( ).

```
% Constants:
beta = [0.5 1.0 2.0 5.0];
N = 8192;

% Read file:
[h, fs] = audioread('Jen57On.wav');

% Get length:
L = length(h);

% Convert to mag:
h = h/max(abs(h));
H = fft(h,N);
figure(1);
subplot(5,1,1);
plot(h);
figure(2);
subplot(5,1,1);
plot(20*log10(abs(H)));

for i = 1 : length(beta)

% Convert to LP:
smoothedH = cbsmooth(H,beta(i),N);

% Get new mag:
smoothedH = smoothedH /max(abs(smoothedH));
H1 = fft(smoothedH);

% Plot:
figure(1);
subplot(5,1,i+1);
plot(smoothedH);
xlabel(['Beta = ' num2str(beta(i))]);
figure(2);
subplot(5,1,i+1);
plot(20*log10(abs(H1)));
xlabel(['Beta = ' num2str(beta(i))]);

hold all

end
```

Figure 11, MATLAB for problem 3(b).

The following plots represent the different levels of smoothing with impulse responses on the top, and frequency magnitude responses below (Figure 12).

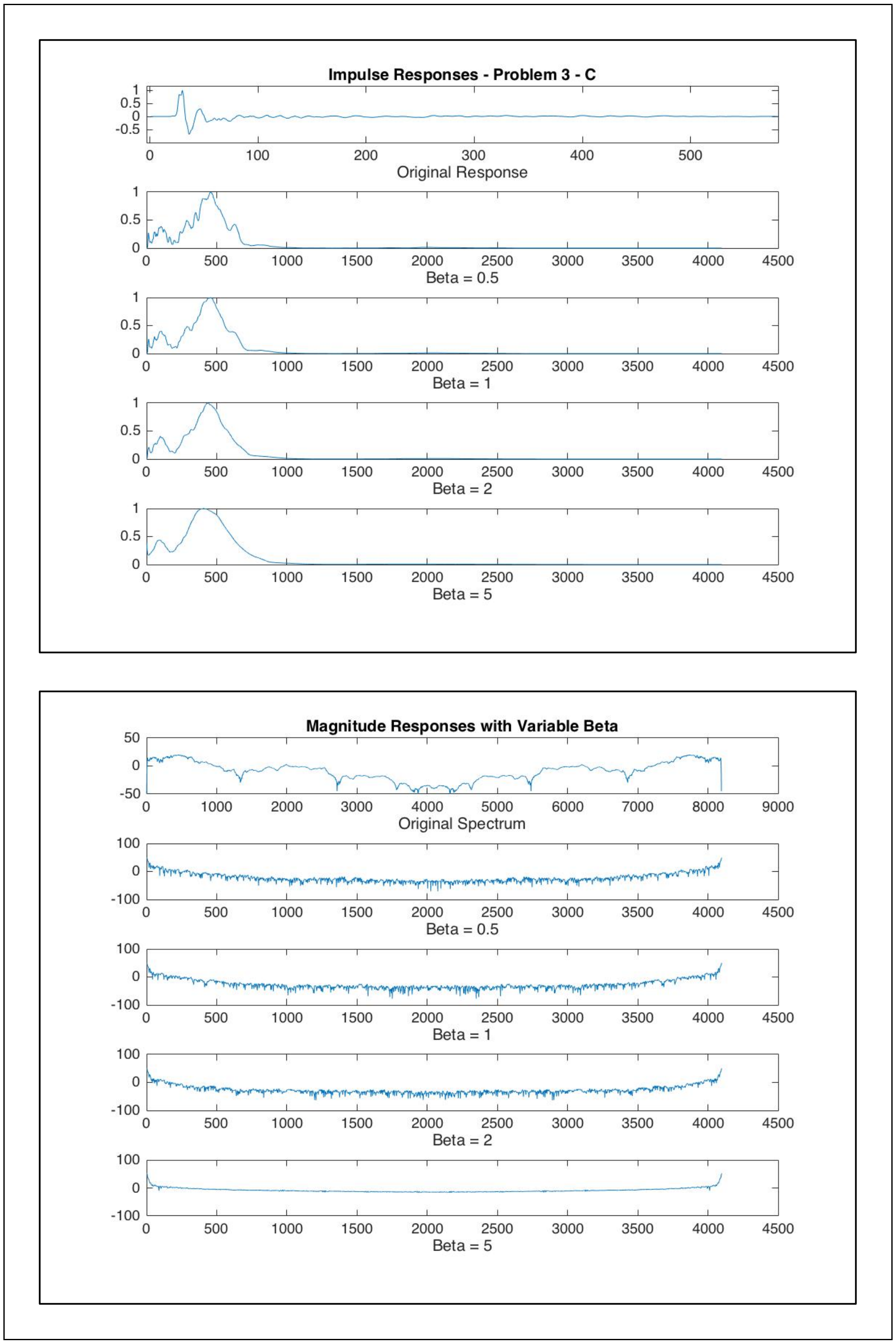

Figure 12, plots for question 3(b).

PROBLEM 3, PART C: CRITICAL-BAND SMOOTHING CONVERSION

The following MATLAB script was used to test different amounts of smoothing as determined by the 'beta' parameter (Figure 13).

```
% Ron Guglielmone
% MUSIC 424, CCRMA, Stanford University
% May 13, 2017
%
% Problem 3 - Part C

clear all;
close all;

% Constants:
N = 8192;

% Read file:
[h, fs] = audioread('Jen57On.wav');
h = h/max(abs(h));
[x, fs] = audioread('ElecGtr-Preflex-Dirty-dry.wav');
x = x/max(abs(x));

% Get length:
L = length(h);

% Convert to mag:
h = h/max(abs(h));
H = fft(h,N);

% Convert to LP:
smoothedH = cbsmooth(H,beta,N);

% Filter the input:
y = fftfilt(smoothedH,x);
y = y/max(abs(y));

% Listen:
sound(y,fs)
```

Figure 13, code for problem 3(c).

Maybe I implemented the smoothing incorrectly, because no values of beta were acceptable in my opinion. All smoothed filters sounded very muddy and flat... Lower values of beta did sound better than higher values, but no value of beta produced an acceptable result aesthetically.

PROBLEM 4, PART A: PRONY'S METHOD

I ran out of time on this assignment. Hopefully I can ask about this at a later date and catch up on some of the missed concepts!