REALSIMPLE Basic Virtual Acoustic Guitar Lab

Nelson Lee and Julius O. Smith III

RealSimple Project*
Center for Computer Research in Music and Acoustics (CCRMA)
Department of Music, Stanford University
Stanford, California 94305

May 13, 2016

Abstract

This is the REALSIMPLE lab on estimating parameters for a physics-based guitar model from measurements from your own guitar! We discuss various components of a physics-based model, and methods for measuring these components and estimating parameters for corresponding model components.

1 Introduction

As discussed in previous labs, there are three components to a physics-based guitar model: the excitation, the string, and the body. The string model, as reviewed in Electric Guitar Lecture: Simple Strings, is based on the Digital Waveguide: a digital implementation of the solution to the Wave Equation. The body of the guitar is implemented as a filter, which can be measured by striking the bridge of your guitar with an impulse-force hammer. Lastly, we present simple methods for computing excitation signals to drive your string-model.

2 Setting Up

In this laboratory you will estimate parameters for a physical-model guitar model based on measurements of a real acoustic guitar.

Below are the steps to be taken, along with some illustrations of the kind of results you should expect:[1]

2.1 Required Software

1. Matlab: Student versions of Matlab are available; the Signal Processing Toolbox is required.
2. CCRMA STK (C++ signal toolkit): the STK is free for download from the CCRMA homepage.

*Work supported by the Wallenberg Global Learning Network

[1]The figures in this laboratory were generated by Stanford Ph.D. student Nelson Lee, using software developed in the course of his thesis research.

2.2 Required Hardware

1. Your guitar!
2. The guitar we used for this lab is a Selmer-Maccaferri copy, similar to the guitar Django Reinhardt used throughout his career.
3. Our recording setup at CCRMA consisted of a Digidesign M-box 2, with two XLR input jacks, and Digidesign's recording software, Pro Tools 7.0. The guitar was equipped with a Schertler DYN-G contact pickup that connected straight into the M-box.

3 Record a Note From Your Guitar

Record a note on your guitar. Figure 1 shows what your note might look like when plotted in the time domain (highest open note on an acoustic steel-string manouche guitar).

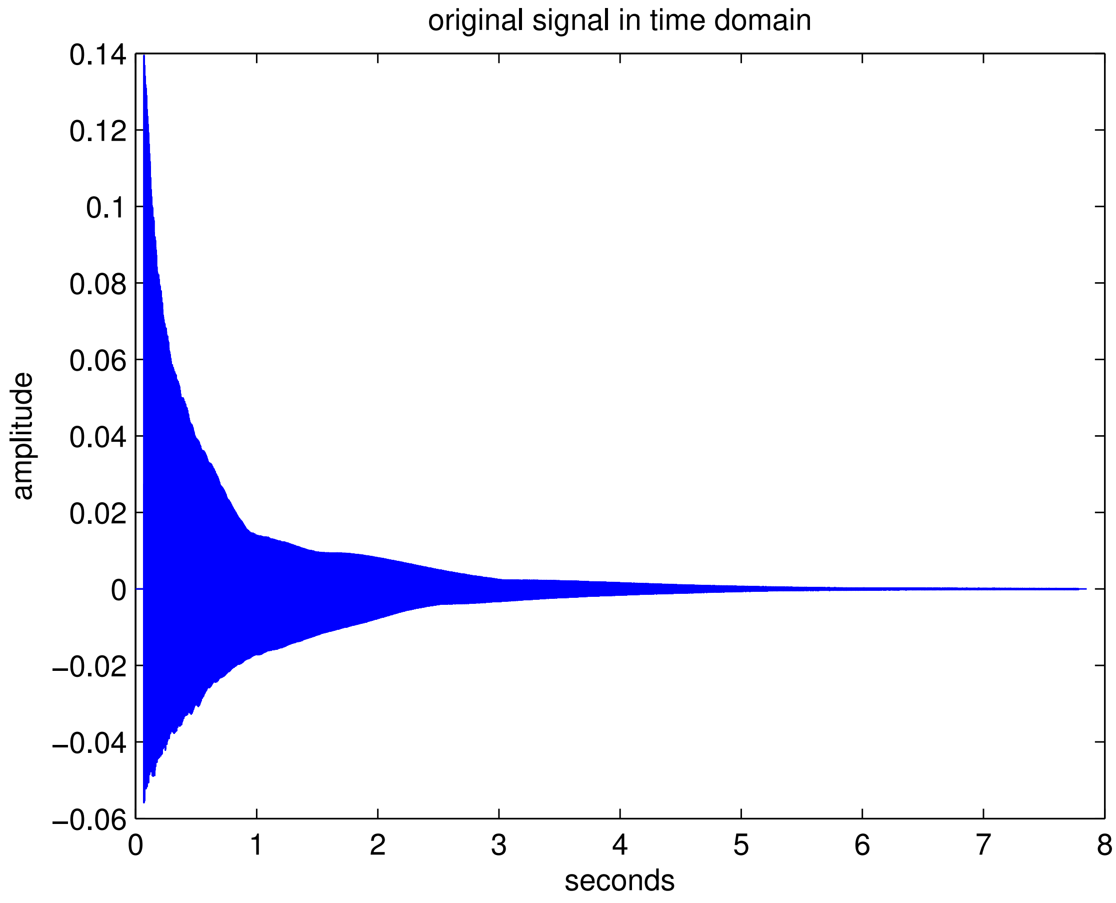

Figure 1: Original signal in the time domain.

The following code segments shows basic usage of reading a file into Matlab and plotting the data to obtain the results in Figure 1.

```
fileName = 'hard_long_1_16b.wav';
[y,fs,bits]=wavread(fileName);

%% fs is the sampling rate of the recording
%% here we have fs = 44100
t=(0:length(y)-1)*1/fs;
figure;plot(t,y)
xlabel('seconds');
ylabel('amplitude');
title('original signal in time domain');
```

4 The String Model

In this section, we will use a metric commonly used in reverberation studies for estimating the decay-rates/loop-filter parameters of your string model. The metric, Energy Decay Relief, shows how the energy of a signal decays over time for different frequencies. In this section we will review the Energy Decay Relief and then estimate parameters for your loop filter for your string-model.

4.1 Using the Energy Decay Relief (EDR)

Compute the Energy Decay Relief (EDR) [1] of the original signal. Here we use Matlab's Signal Processing Toolkit. The function spectrogram computes the Short-Time Fourier Transform (STFT) of the signal. From the STFT, we compute the EDR. Figure 2 and Fig. 3 show EDRs for the recorded note.

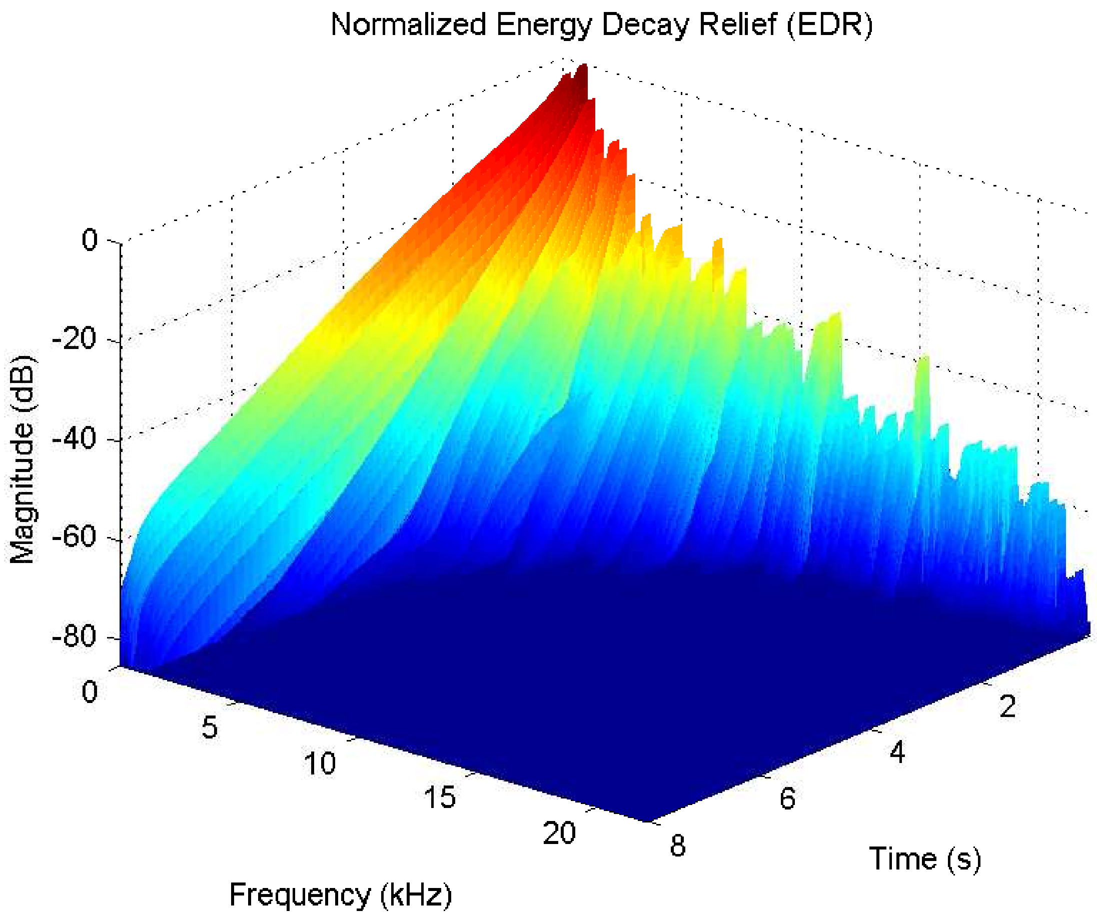

Figure 2: EDR of the signal truncated at -85 dB

We've provided the following code segments to demonstrate usage of the Signal Processing Toolbox in Matlab, as well as three-dimensional plotting.

```
frameSizeMS = 30; % minimum frame length, in ms
```

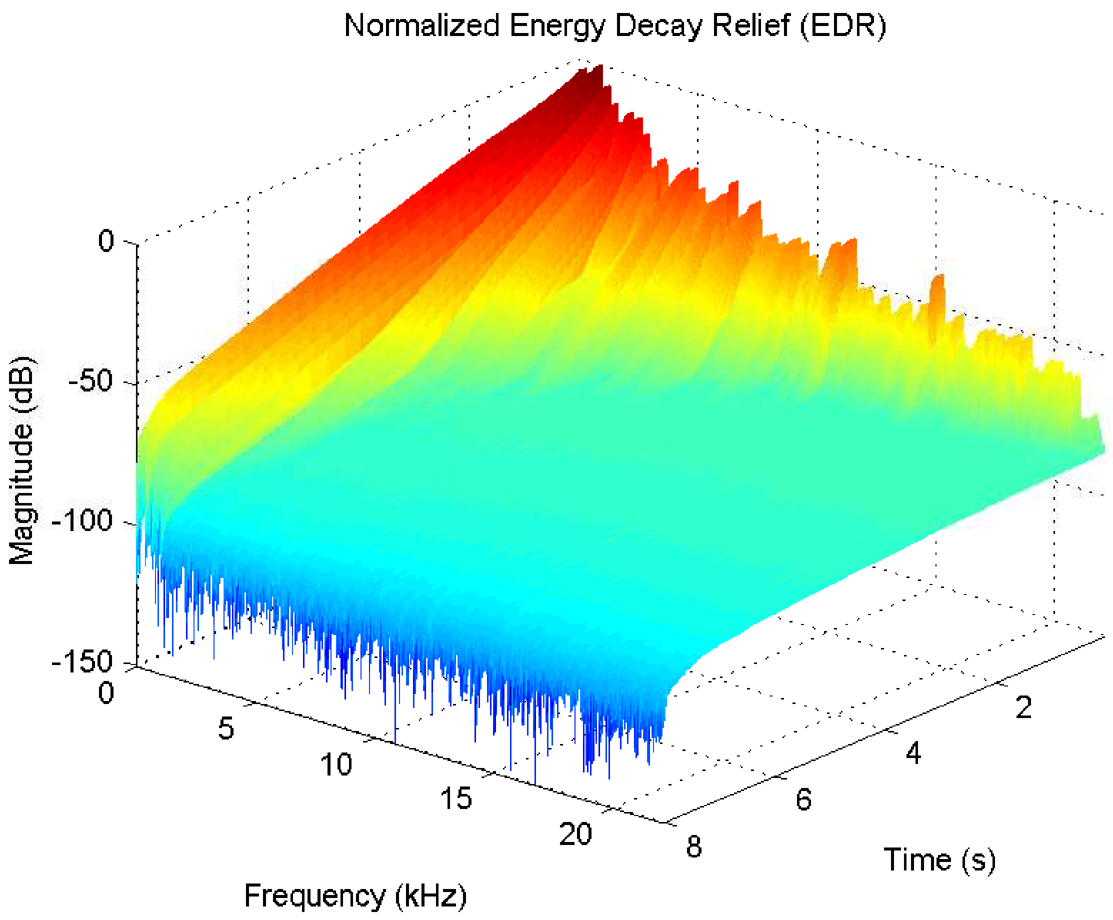

Figure 3: EDR of the signal without truncation

```
overlap = 0.75; % fraction of frame overlapping
windowType = 'hann'; % type of windowing used for each frame

[signal,fs,bits] = wavread(fileName);

% calculate STFT frames
minFrameLen = fs*frameSizeMS/1000;
frameLenPow = nextpow2(minFrameLen);
frameLen = 2^frameLenPow; % frame length = fft size
eval(['frameWindow = ' windowType '(frameLen);']);
[B,F,T] = spectrogram(signal,frameWindow,overlap*frameLen,2*STFT_Npt,fs);

[nBins,nFrames] = size(B);

B_energy = B.*conj(B);
B_EDR = zeros(nBins,nFrames);
for i=1:nBins
    B_EDR(i,:) = fliplr(cumsum(fliplr(B_energy(i,:))));
end
B_EDRdb = 10*log10(abs(B_EDR));

% normalize EDR to 0 dB and truncate the plot below a given dB threshold
offset = max(max(B_EDRdb));
B_EDRdbN = B_EDRdb-offset;

B_EDRdbN_trunc = B_EDRdbN;
for i=1:nFrames
  I = find(B_EDRdbN(:,i) < minPlotDB);
  if (I)
    B_EDRdbN_trunc(I,i) = minPlotDB;
  end
end

figure(gcf);clf;
mesh(T,F/1000,B_EDRdbN_trunc);
view(130,30);
title('Normalized Energy Decay Relief (EDR)');
xlabel('Time (s)');ylabel('Frequency (kHz)');zlabel('Magnitude (dB)');
axis tight;zoom on;
```

4.2 Designing the loop filter for your guitar model

In this section we review Matlab calls for fitting filters to wanted magnitude responses. As using such functions are not only useful for computing the loop-filter of your string-model, it should be kept in mind that such methods are useful for general applications requiring filter-fitting.

4.2.1 Fitting Filters in Matlab

1. Measure the decay rate of each partial overtone, and calculate the desired amplitude response for the loop filter in a digital waveguide model. Take the EDR, and using the rate of decay of the signal, compute the necessary gains for the harmonic series of the signal. We will have code samples for fitting a line to the dB decay of one frequency in a later section.

   Figure 4 shows the resulting desired magnitude response of our loop filter as calculated from the EDR..

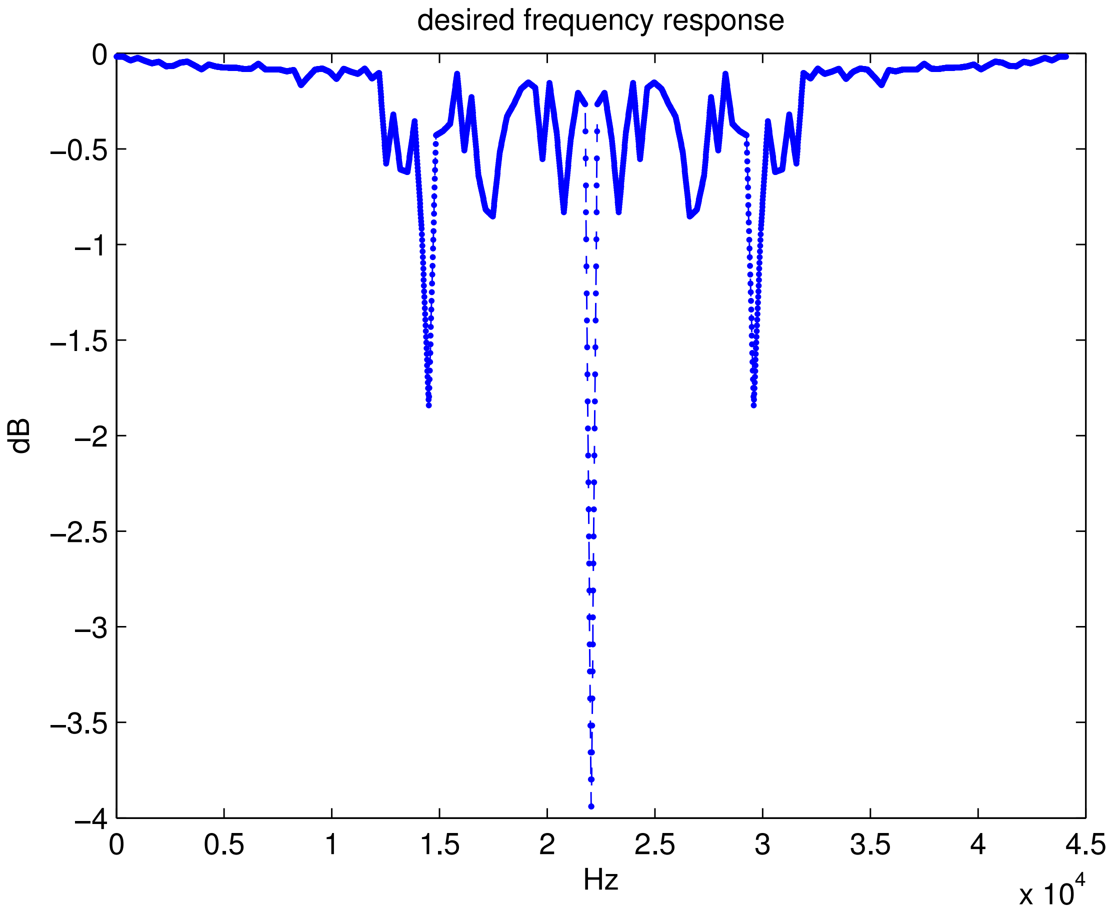

Figure 4: Desired magnitude response of our loop filter.

2. Convert the desired amplitude response to minimum-phase form.[2] This simplifies complex behavior involving the phase of the excitation signal with the fitted filter. Figure 5 shows the magnitude response after converting to minimum phase. Figure 6 shows the resulting

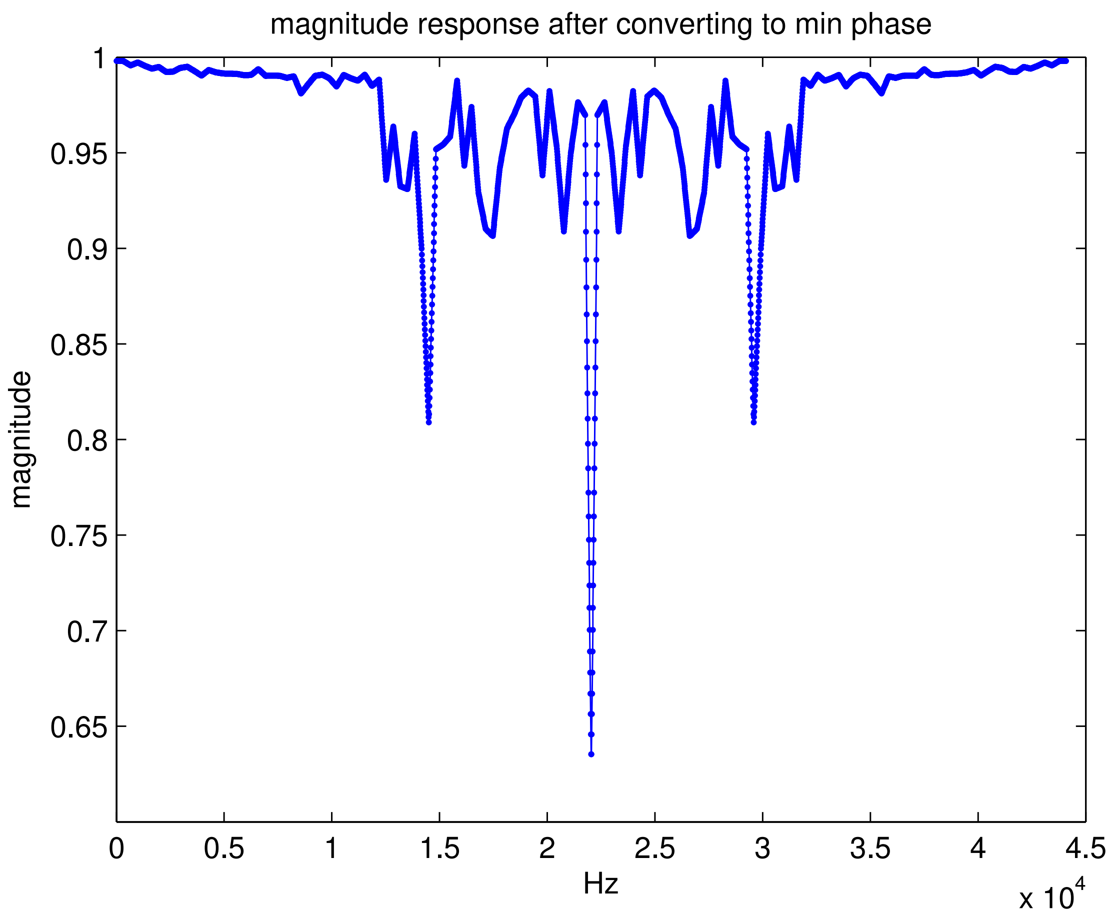

Figure 5: The desired magnitude response after converting to minimum phase

minimum-phase response.

[2] https://ccrma.stanford.edu/~jos/filters/Conversion_Minimum_Phase.html
https://www.dsprelated.com/showarticle/110.php

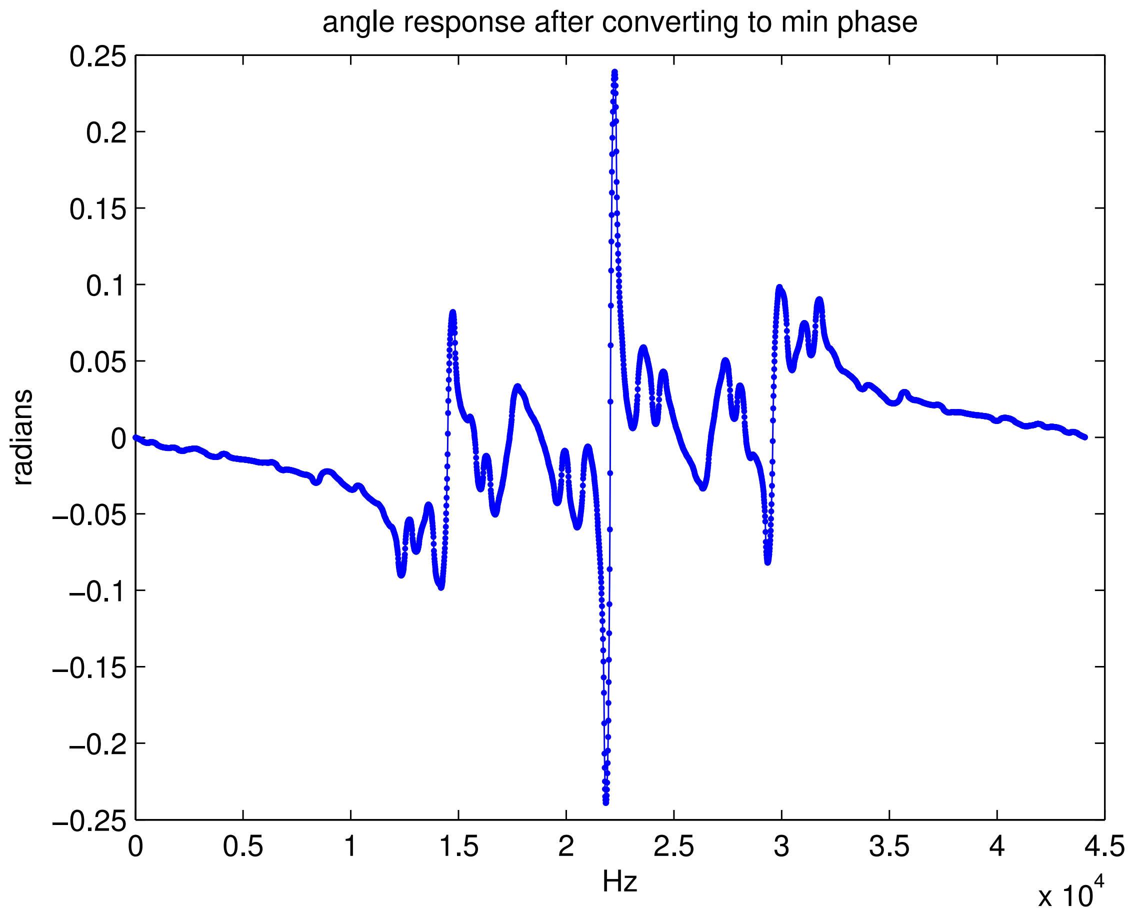

Figure 6: The phase response after converting to minimum phase.

3. Design the loop filter to approximate the minimum-phase frequency response. Figure 7 shows the result obtained for this example using `invfreqz` in Matlab (or Octave Forge).

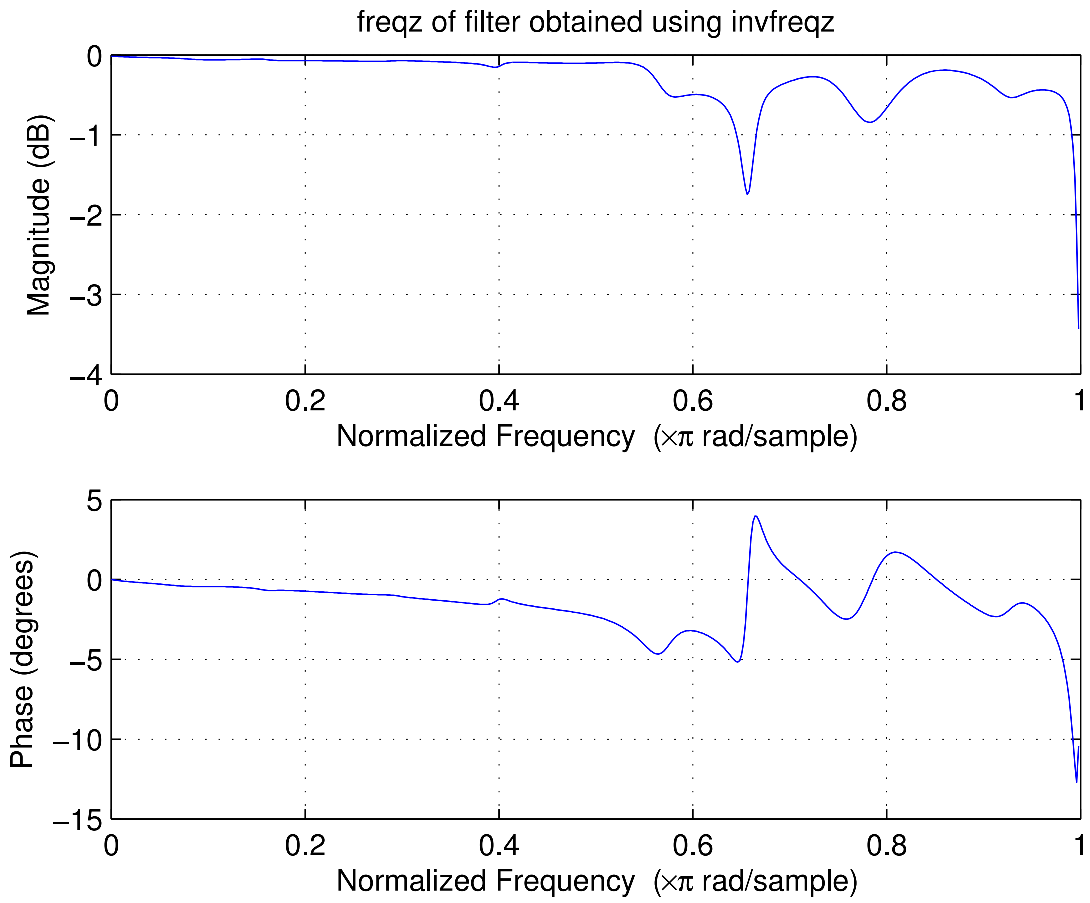

Figure 7: Magnitude and Phase responses of the fitted filter using `invfreqz`

The code segments demonstrates how we fitted a filter using `invfreqz` and `stmcb`.

```
% Note that Hmin corresponds to the response of the minimum-phase filter
% that we want to fit to obtain filter parameters using invfreqz
wH = (0:Npt/2)*2*pi/Npt; % sampled normalized radian frequency axis
wt = 1 ./ [wH(2),wH(2:end)]; % weight fn: boost low freqs, avoid 0
[B,A] = invfreqz(Hmin(1:Npt/2+1),wH,25,25,wt);
figure;freqz(B,A)
title('freqz of filter obtained using invfreqz');

% % Obtain filter parameters using stmcb
hdesired = real(ifft(Hmin));
figure;plot((0:length(hdesired)-1)/fs,(hdesired));
title('impulse response of min phase filter')
```

```
xlabel('seconds');
ylabel('amplitude');
[stmb,stma] = stmcb(hdesired,40,40);
figure;freqz(stmb,stma)
title('freqz of filter obtained using stmcb');
```

Figure 8 shows the result obtained for this example using **`stmcb`** from the Matlab Signal Processing Tool Box.

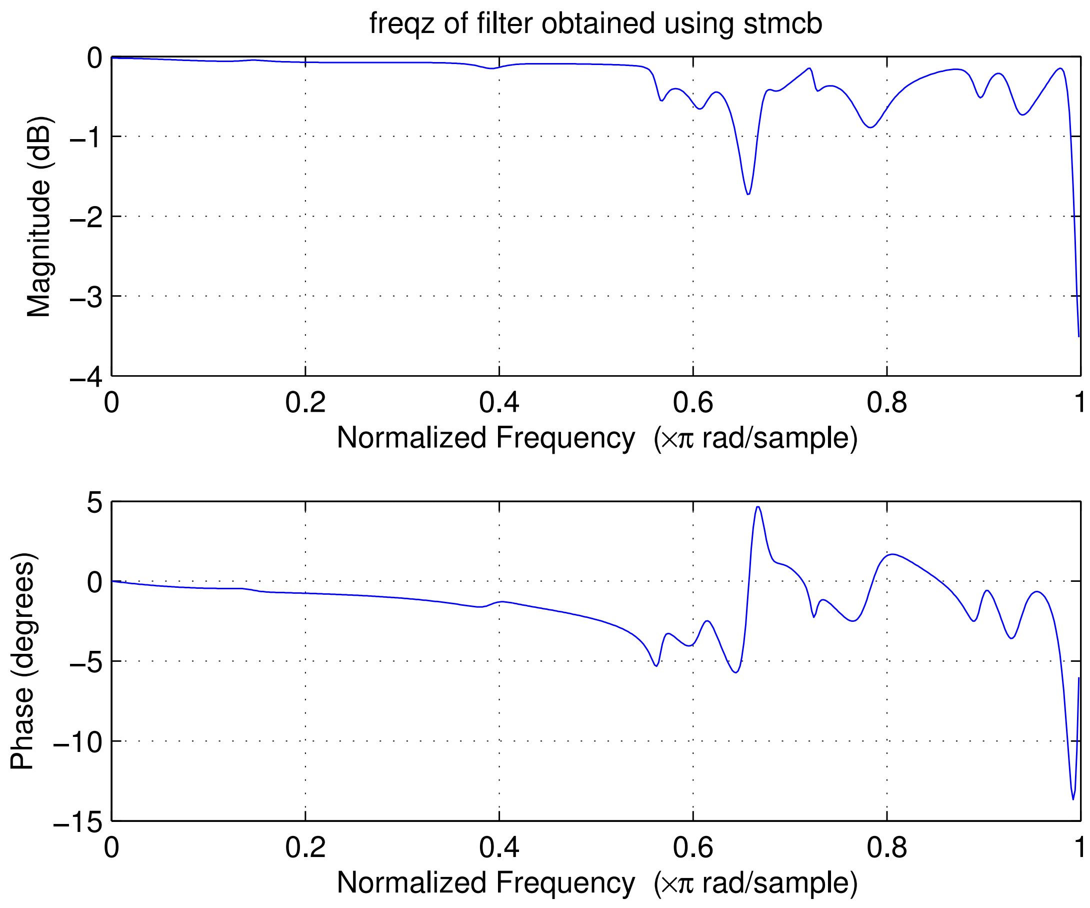

Figure 8: Magnitude and phase response of the filter

Plotting the results in Matlab is simple, as the following code segments shows. Note that Matlab's **`freqz`** function takes as arguments the coefficients of a filter plots the magnitude and phase responses of the described filter.

```
figure;freqz(B,A)
title('freqz of filter obtained using invfreqz');

figure;freqz(stmb,stma)
```

```
title('freqz of filter obtained using stmcb');
```

4.2.2 Viewing Magnitude Responses of the Fitted Filters and the Original Signal

Figure 9 shows an overlay of the original, `invfreqz`, and `stmcb` amplitude responses.

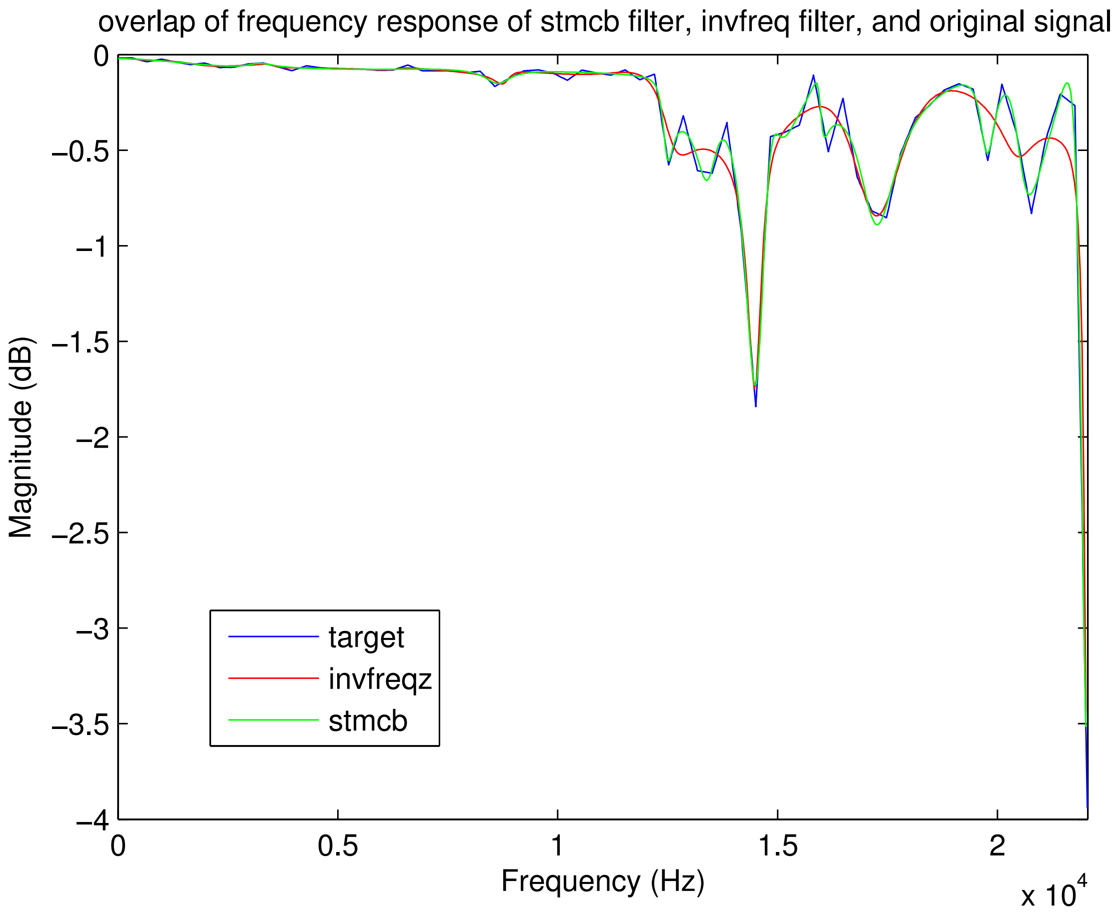

Figure 9: Overlap of the fitted filter magnitude responses and the original signal

Figure 10 shows a close-up of the overlay of the previous figure.

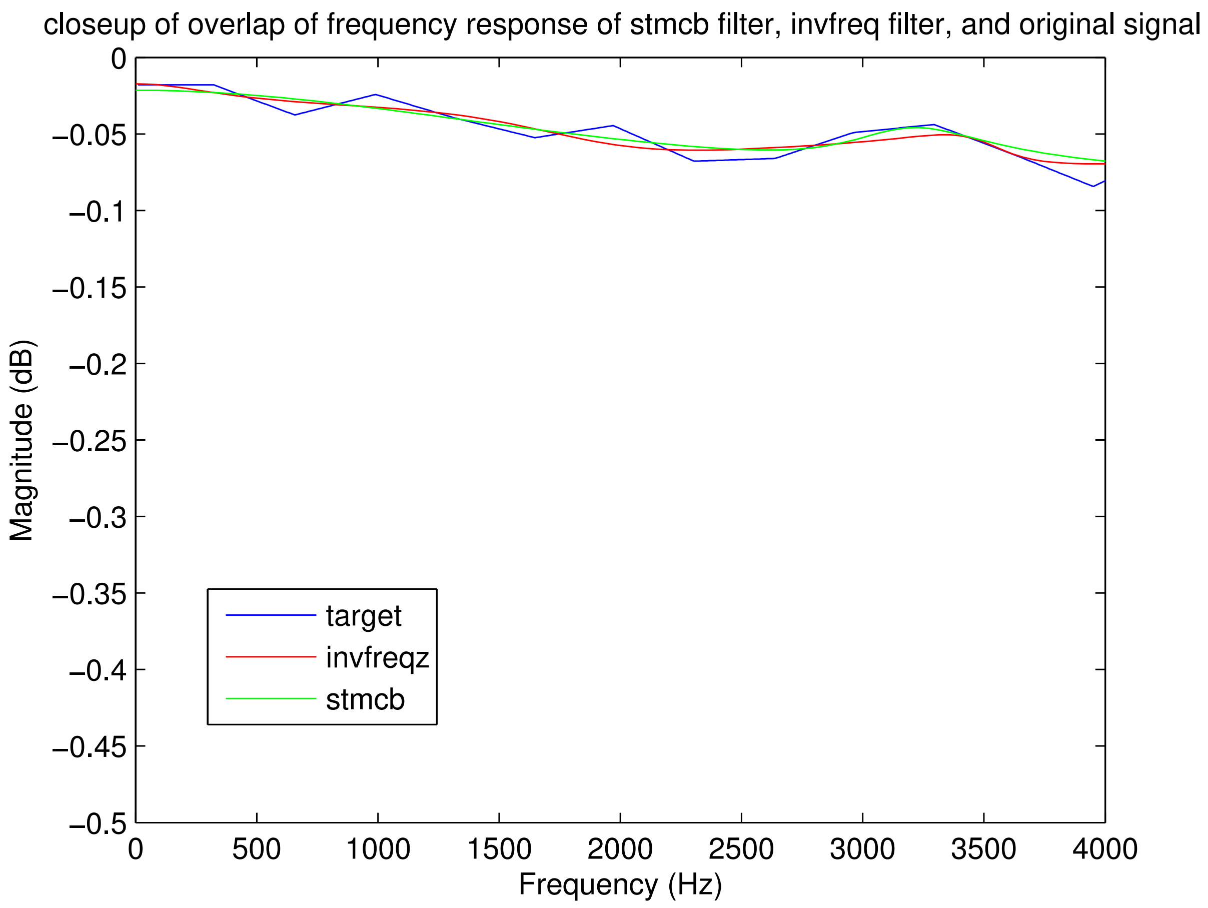

Figure 10: Closeup of the overlap of the fitted filter magnitude responses and the original signal

The following code segment was used to generate the above two plots:

```
figure;
plot(wH*fs/(2*pi),20*log10(abs(Hmin(1:Npt/2+1))));
hold on
plot(Finvf,20*log10(abs(Hinvf)),'r')
plot(Fstm,20*log10(abs(Hstm)),'g')
xlabel('Frequency (Hz)')
ylabel('Magnitude (dB)')
legend('target','invfreqz','stmcb')
title(['closeup of overlap of frequency response of stmcb filter,',...
       ' invfreq filter, and original signal]');
axis([0 4000 -0.5 0])
```

5 Computing the Excitation Signal

Now that we have a string-model for our guitar model, we can use it along with the original recorded guitar tone to compute the excitation signal using inverse-filtering.

1. Inverse filter to obtain the excitation signal. Here we assume that the excitation signal, when fed into a filtered-delay-loop, will match the original signal. Therefore, to obtain the excitation signal, we delay the original signal by one period, run it through our filter, and subtract the resulting signal from the original to obtain the excitation signal. Figure 11 shows the result obtained for this example.

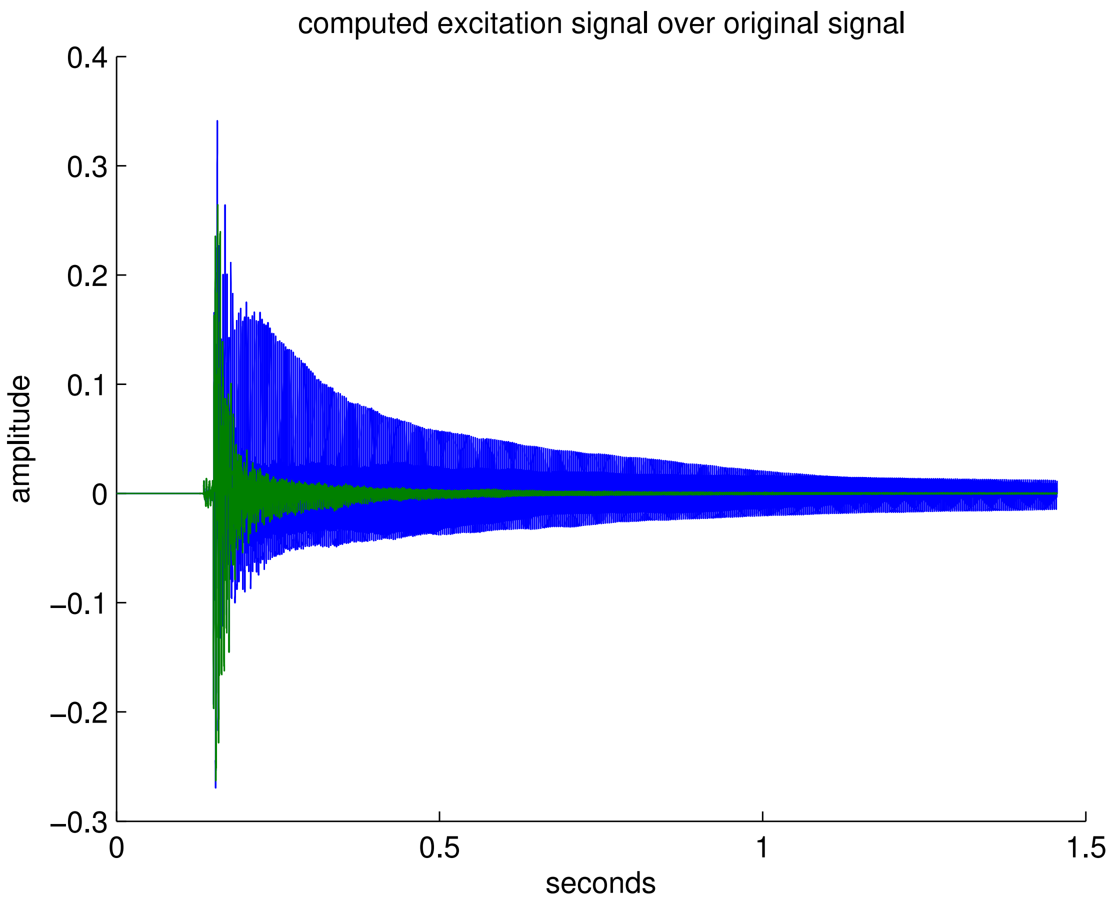

Figure 11: Plot of the computed excitation signal over the original signal

The following code segment computes the excitation signal and writes it out to disk.

```
% find the excitation signal.
% FUND_F corresponds to the fundamental freq
N = round(fs/FUND);
```

```
del_y = [zeros(N,1);y(1:end-N)];
filt_y = filter(stmb,stma,del_y);
% filt_y = filter(B,A,del_y);

e_sig = y-filt_y;
wavwrite(e_sig,fs,bits,'../sound_files/e_sig.wav');
```

2. The resulting excitation signal is ploted in Figure 12.

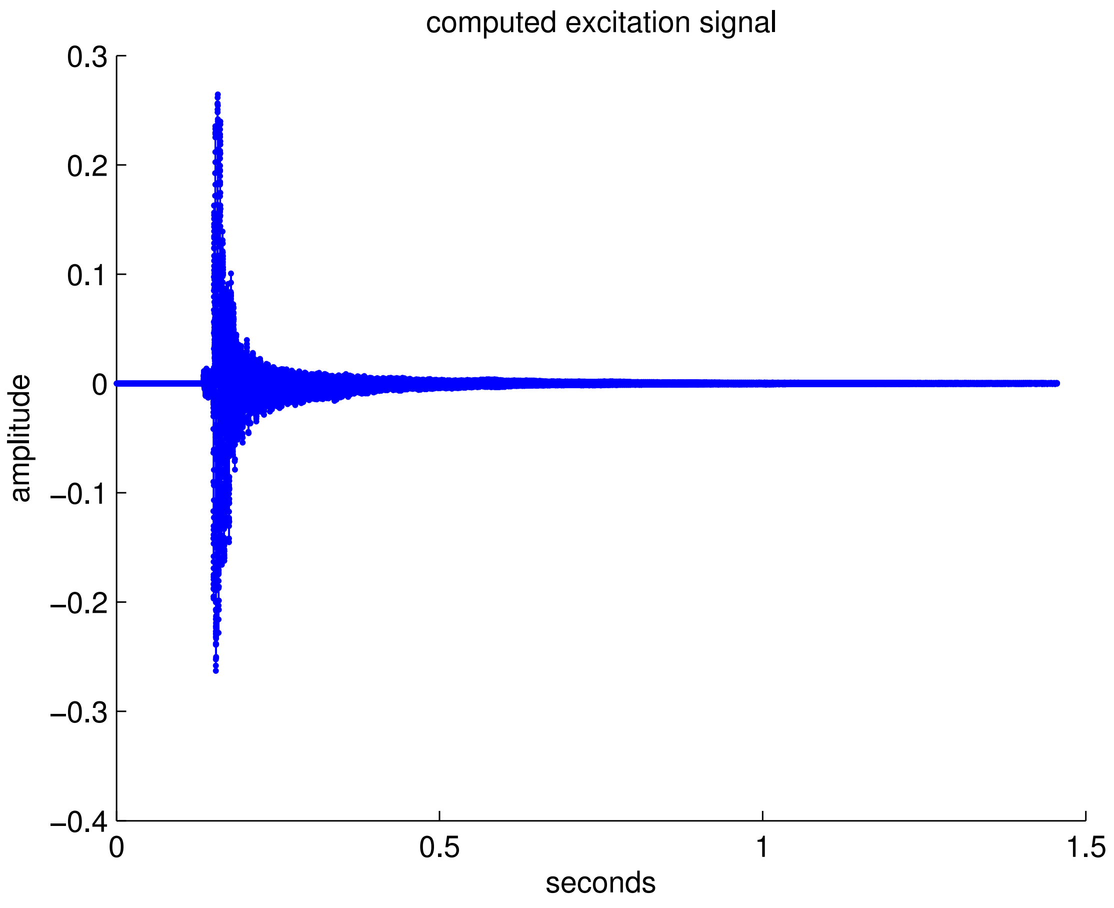

Figure 12: Computed excitation signal

3. Figure 13 shows the resulting excitation signal in dB.

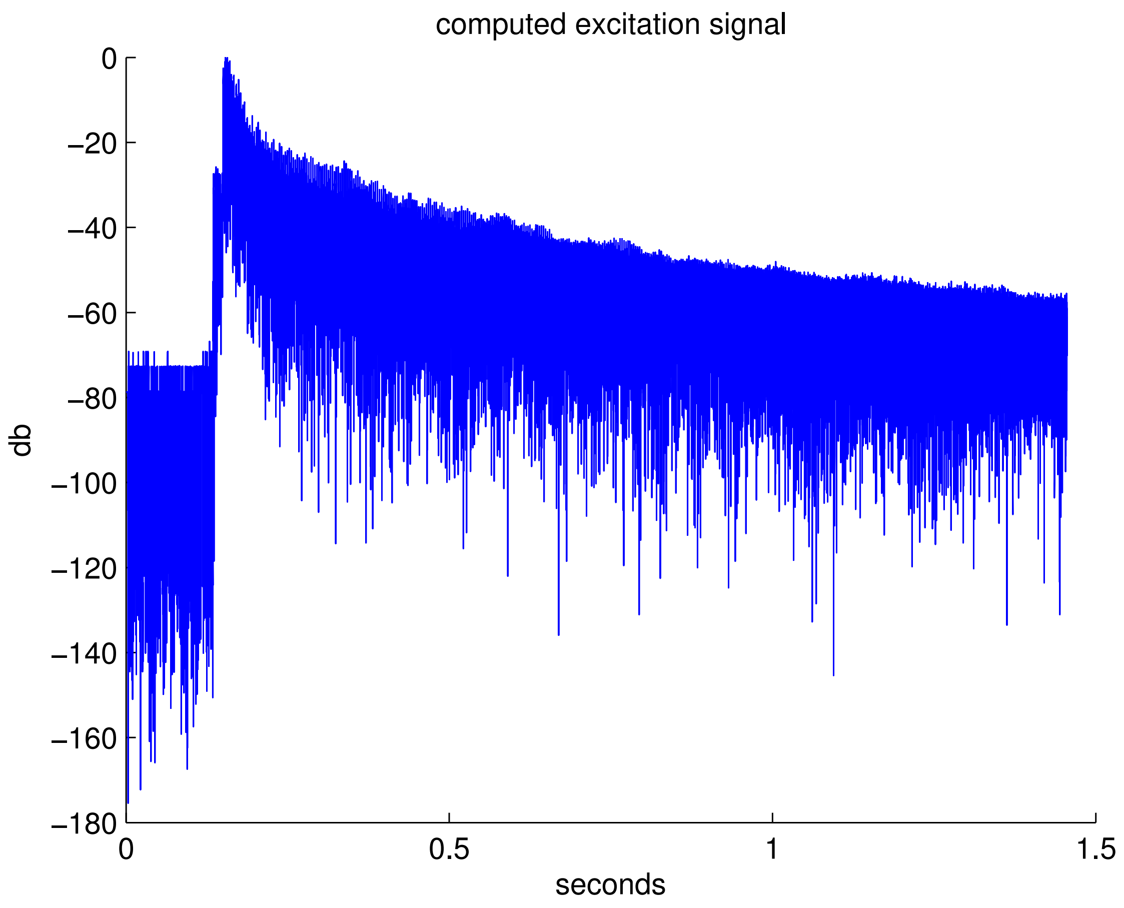

Figure 13: Computed excitation signal in dB

6 Finding the Desired Gains of the Loop Filter for Individual Frequencies

In 4, we had a graph of a filter with corresponding gains that we had estimated from the EDR. In this section, we go into detail as to how those gains were computed for filter-fitting from the EDR.

6.1 The Fundamental Frequency

Here we will elaborate on how to use the results of the EDR for each frequency to compute the wanted gain at that frequency of our loop filter. For the fundamental, we want to match a straight line to the linear decay (in dB) for the plot. As shown, the segment we highlighted in green marks a good region for a linear fit. Since we know how many dBs the signal decays on this segment, we can compute the desired gain. We have provided the code segment in our Matlab program that does this:

```
%% here i corresponds to the index into our arrays for the fundamental freq (328.38Hz)
curr_vals = B_EDRdbN(i,:);
segmentFrames = startFrame:endFrame;
curr_region = curr_vals(segmentFrames);

P = polyfit(segmentFrames,curr_region,1);
slopes(i) = P(1);
offsets(i) = P(2);

RT(i) = -60*overlapSamp/(slopes(i)*fs);
g = 10.^(-3./(RT(i)*(FUND)));
gdb = 20*log10(g);

%% here gdb is in the format necessary for Matlab to fit a filter
%% to a desired  magnitude response
```

Figure 14 shows the line fit to the decay of the original signal at the fundamental.

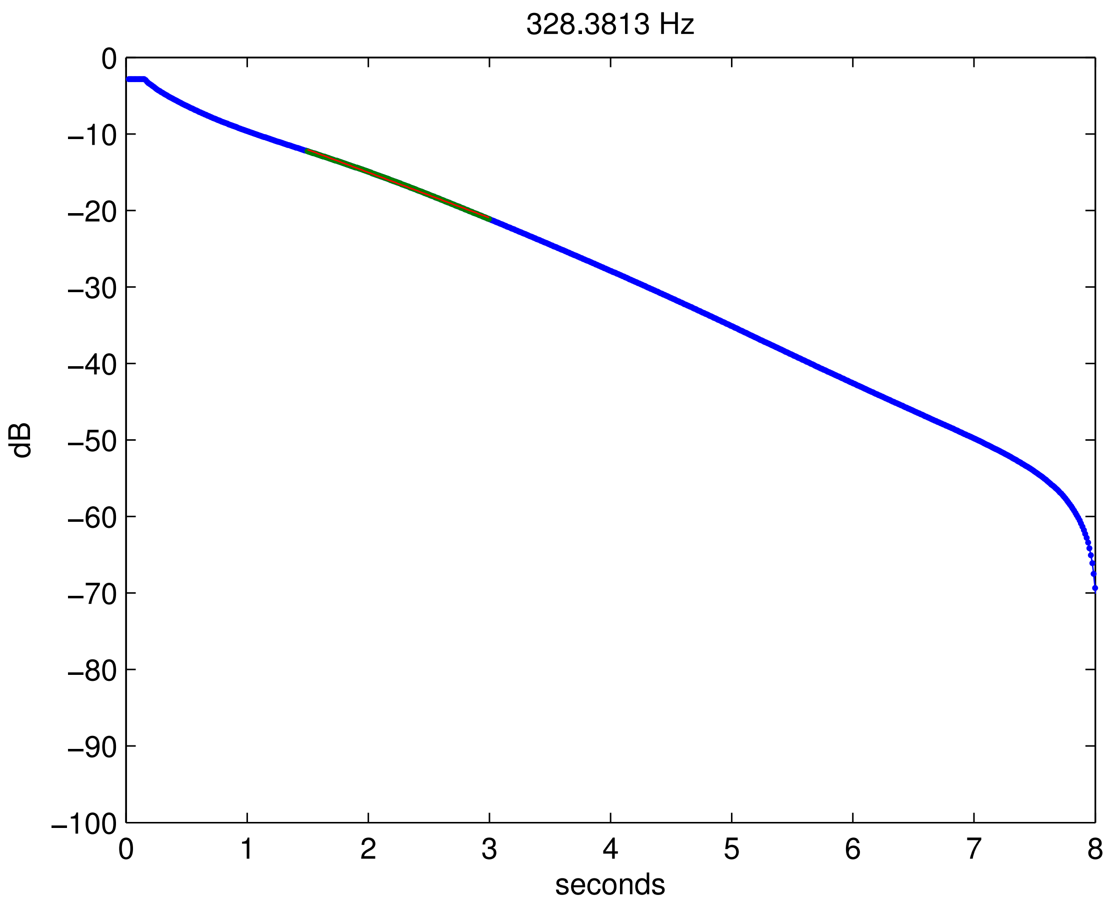

Figure 14: Overlay of the fitted straight line with the decay in energy at the fundamental

Figure 15 shows the extrapolated delay beginning at 0dB to -100dB at the fundamental frequency.

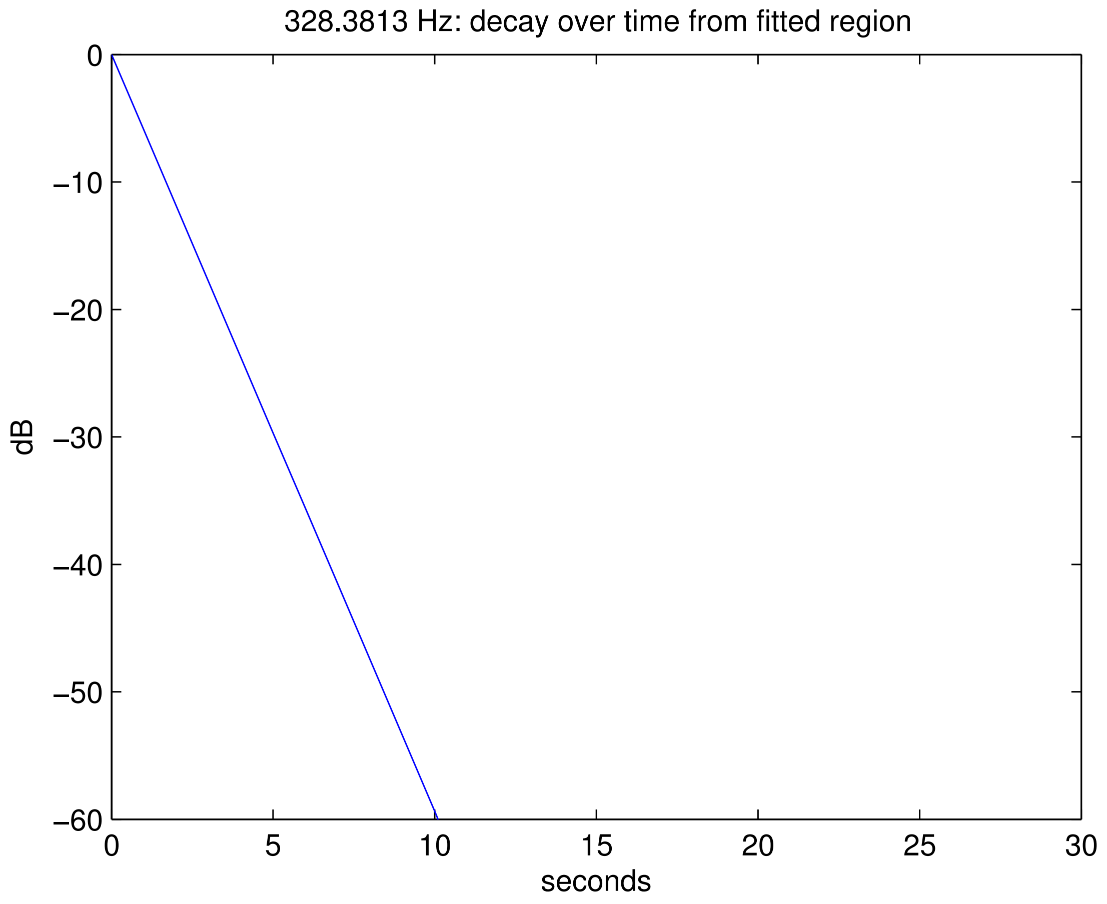

Figure 15: Extrapolation of our fitted decay at the fundamental frequency

6.2 Other harmonics

This section shows overlays of our linear fits to higher overtones, as well as their extrapolated decays. Note that in some cases such as the first, there is a two-stage decay which often occurs due to the coupling of nearly identical resonances. We want to find the gain relative to the slope of the greater initial decay, rather than the second-stage decay.

1. The 2nd Harmonic

   Figure 16 shows the linear fit and the region used for the 2nd harmonic.

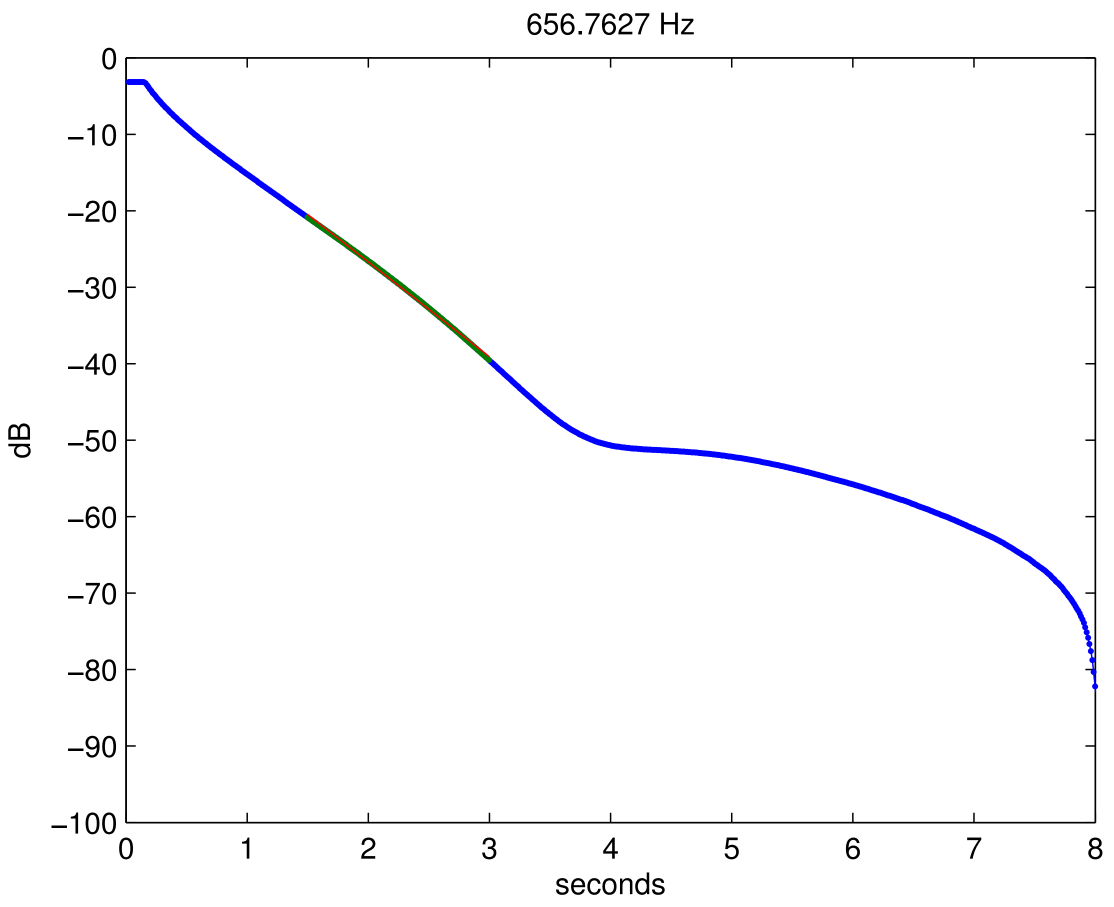

Figure 16: Linear fit and the region used for the 2nd harmonic

Figure 17 shows the extrapolated decay for the 2nd harmonic.

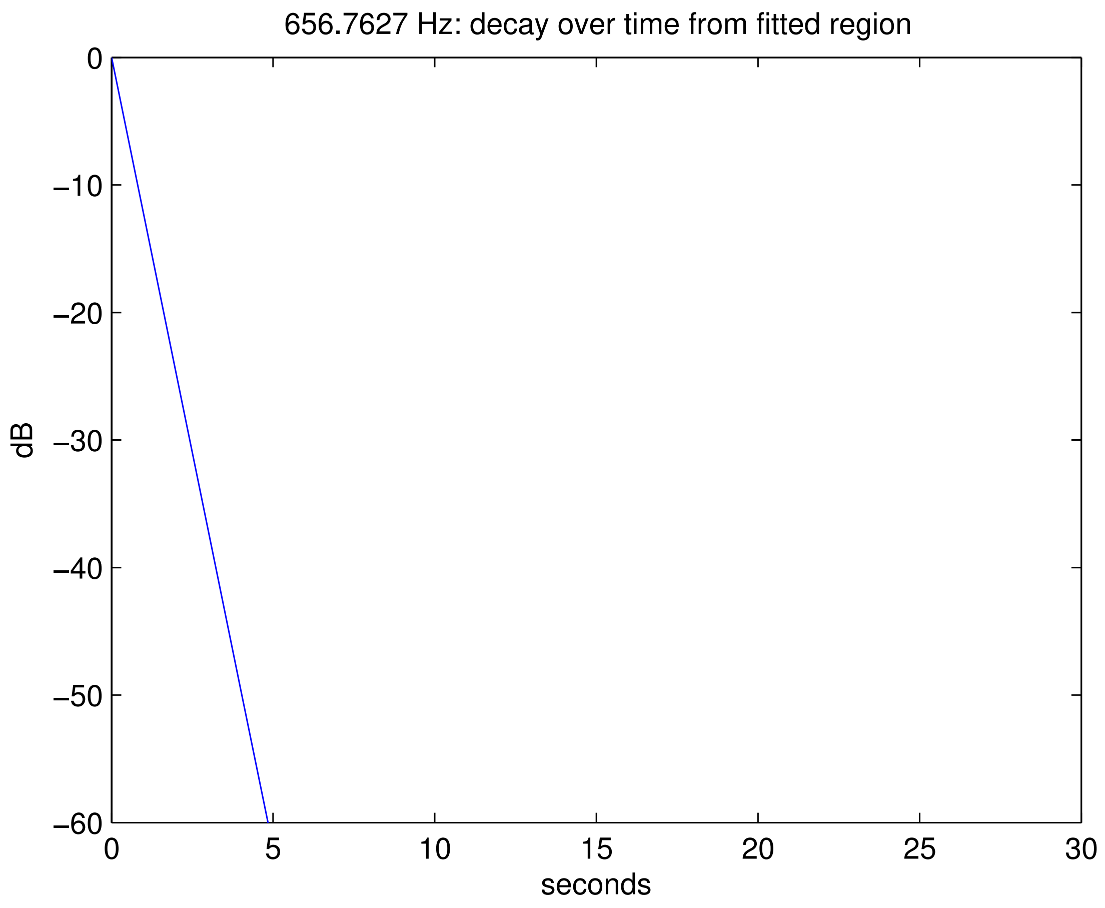

Figure 17: Extrapolated decay for the 2nd harmonic

2. The 3rd Harmonic

   Figure 18 shows the linear fit and the region used for the 3rd harmonic.

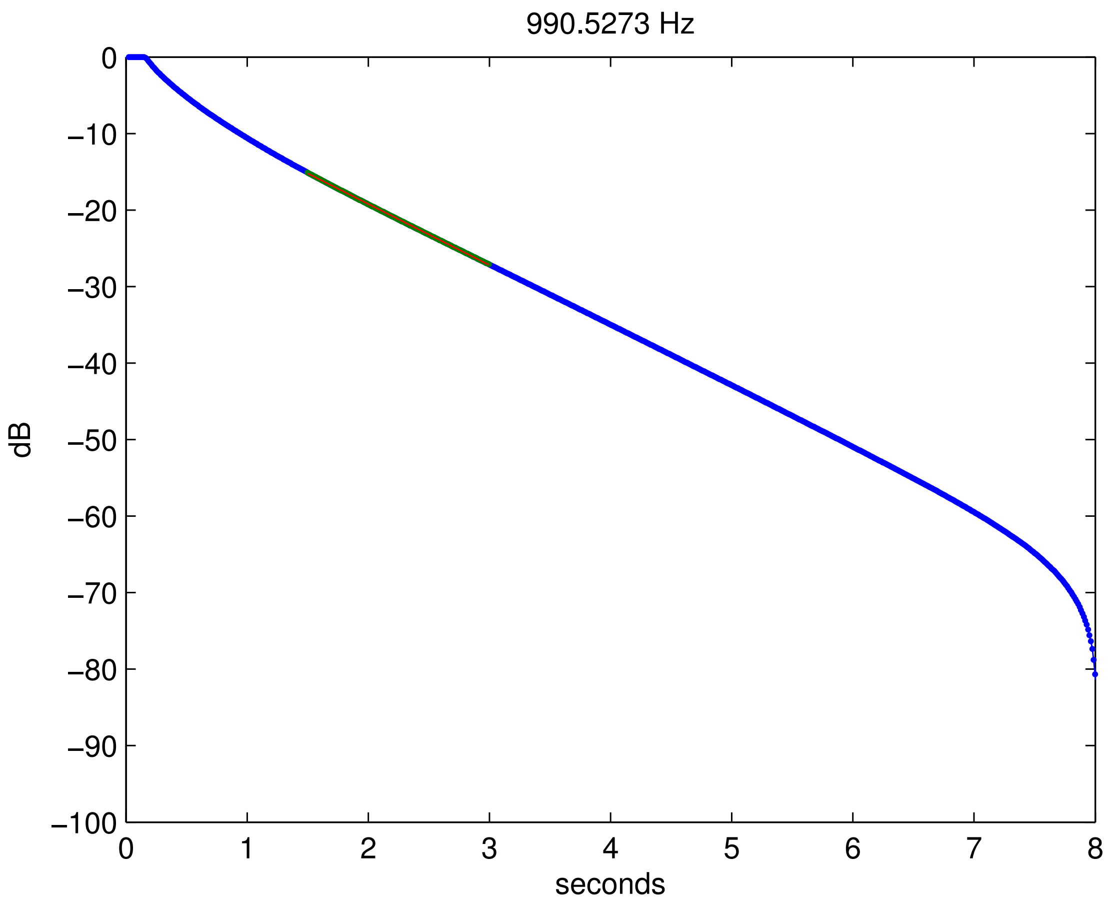

Figure 18: Linear fit and the region used for the 3rd harmonic

Figure 19 shows the extrapolated decay for the 3rd harmonic.

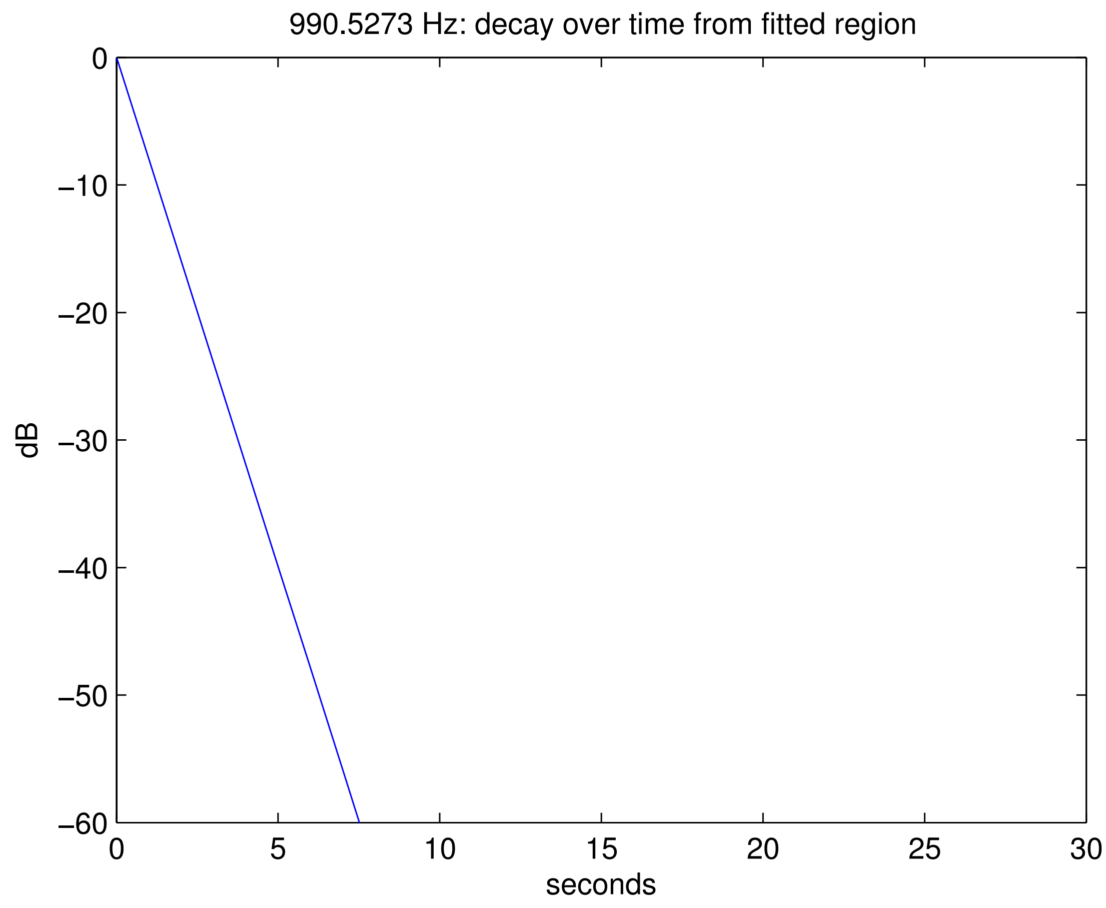

Figure 19: Extrapolated decay for the 3rd harmonic

3. At 3628Hz

   Figure 20 shows the linear fit and the region used at 3628Hz.

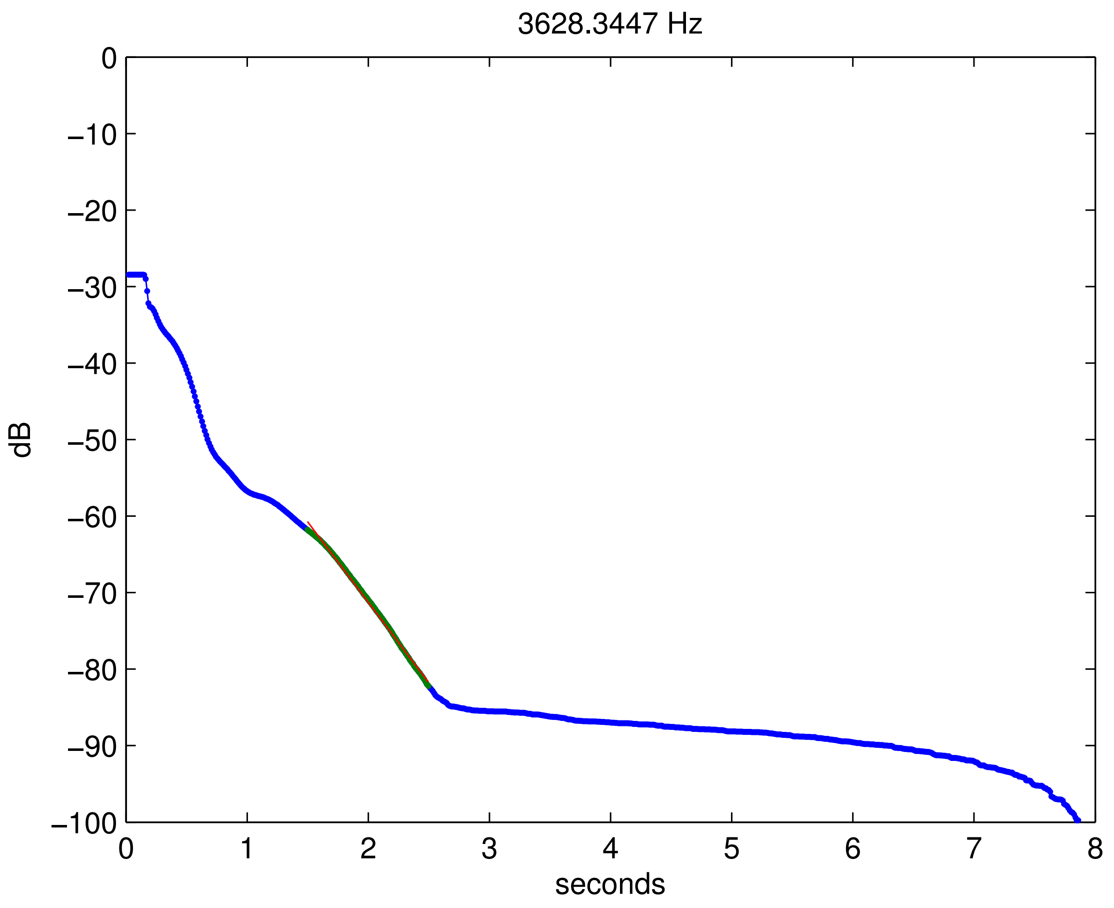

Figure 20: Linear fit and the region used 3628Hz

Figure 21 shows the extrapolated decay at 3628Hz.

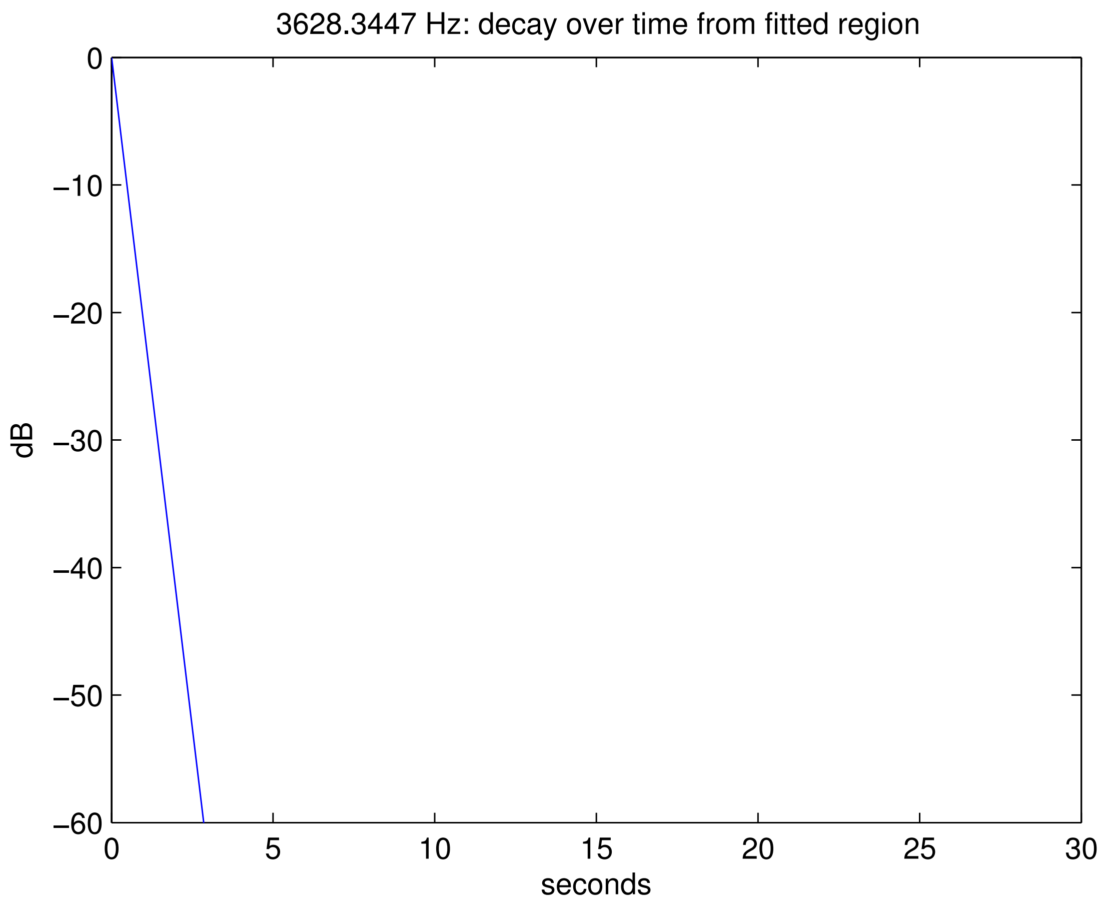

Figure 21: Extrapolated decay at 3628Hz

4. At 12527Hz

   Figure 22 shows the linear fit and the region used at 12527Hz.

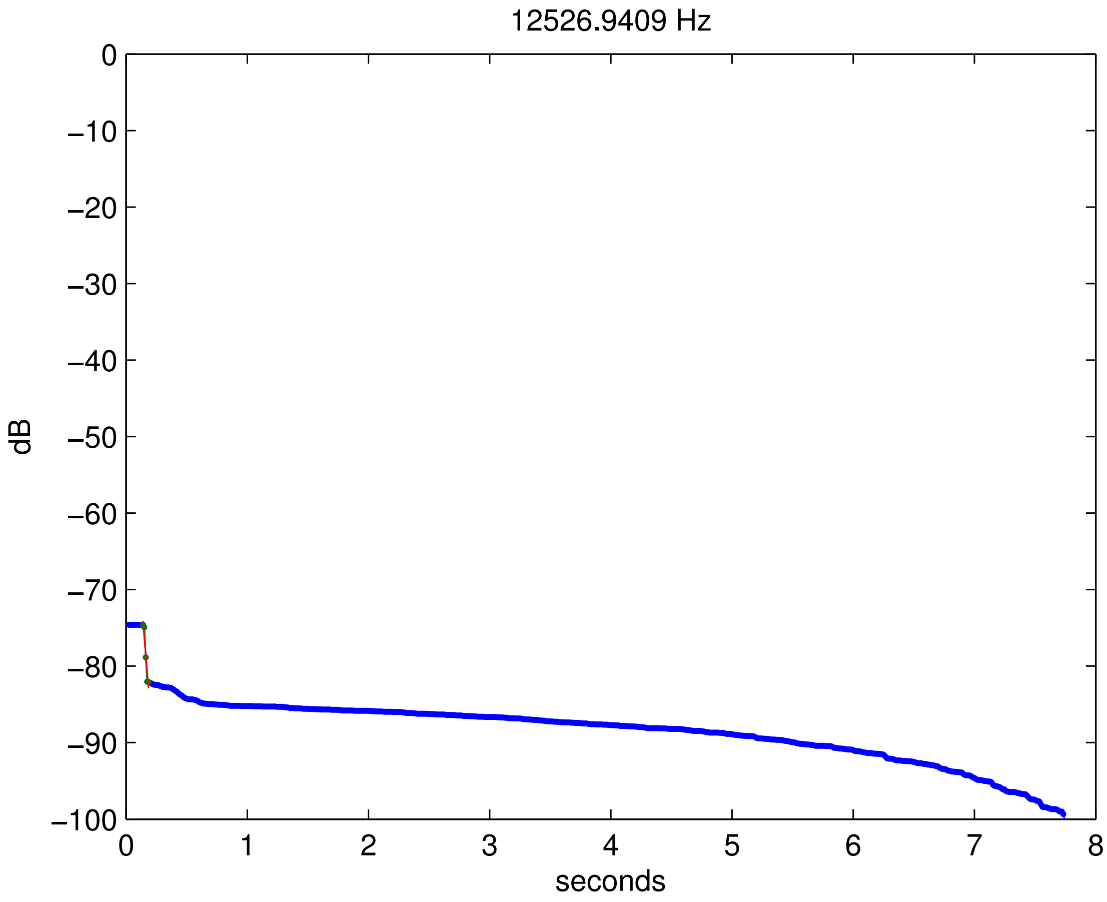

Figure 22: Linear fit and the region used 12527Hz

Figure 23 shows the extrapolated decay at 12527Hz.

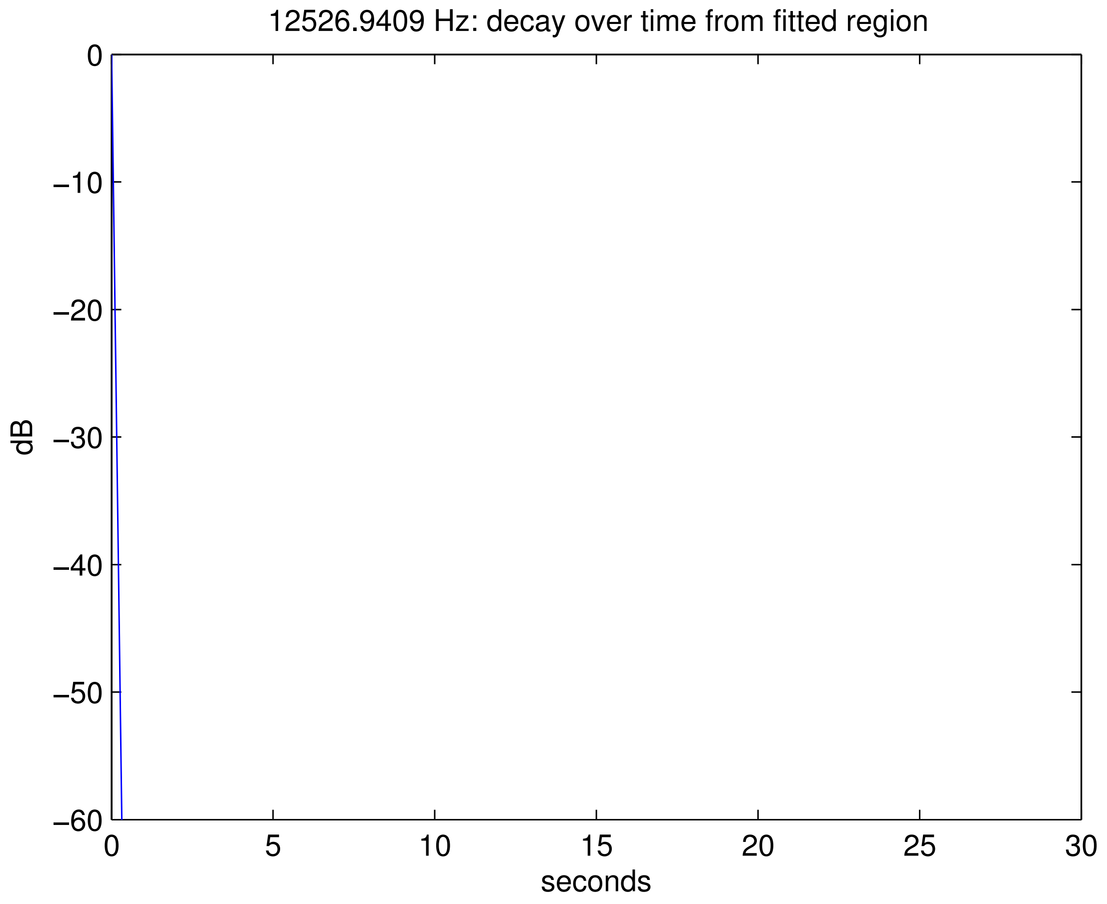

Figure 23: Extrapolated decay at 12527Hz

5. At 21759Hz

   Figure 24 shows the linear fit and the region used at 21759Hz.

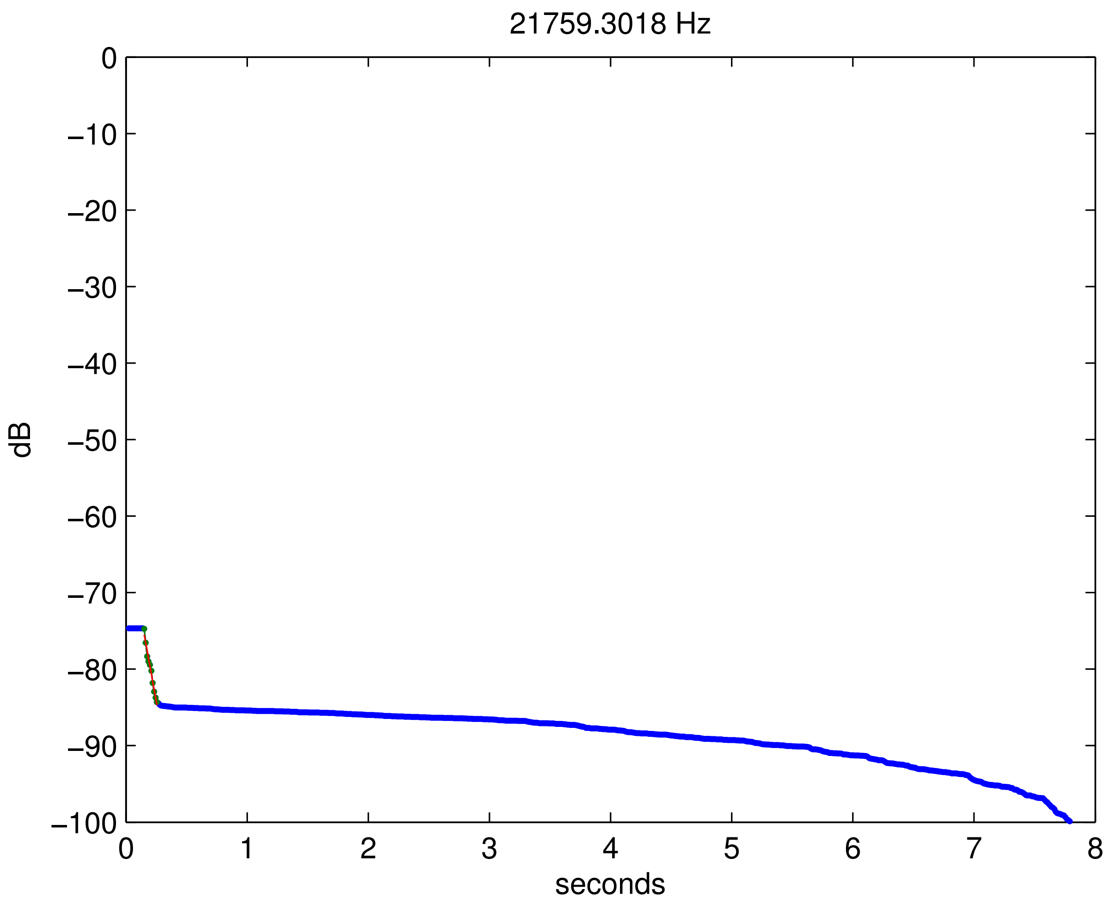

Figure 24: Linear fit and the region used 21759Hz

Figure 25 shows the extrapolated decay at 21759Hz.

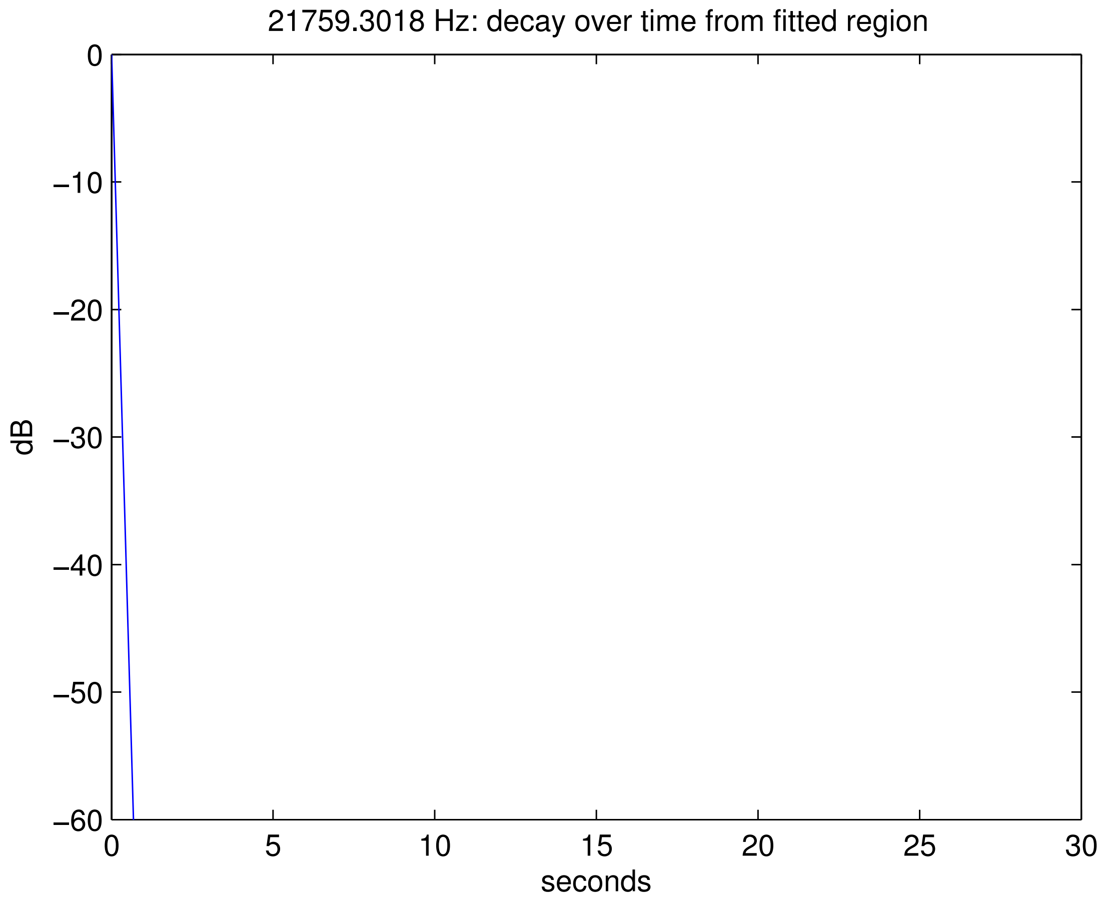

Figure 25: Extrapolated decay at 21759Hz

7 Conclusions

In this laboratory, we discussed methods for estimating parameters of a physics-based guitar model from a recorded guitar tone. Using the Energy Decay Relief, we computed the gains of our loop filter for our string model. With the string-model estimated, we used inverse-filtering with the original recorded guitar tone for estimating the excitation signal for our model.

References

[1] J. O. Smith, *Physical Audio Signal Processing*, `http://ccrma.stanford.edu/~jos/pasp/`, Aug. 2007, online book.