Contents

1 Introduction

Designing filters comes with many tradeoffs; among the parameters to take into account are the cutoff frequencies, transition band, ripple, processing power available, and whether linear phase is necessary. The purpose of this assignment is to compare six types of bandstop filters with the same cutoff frequencies, transition band, and ripple.

The bandstop should allow frequencies on the range of [0, 1400] and [1900, 5,512.5] Hz with a tolerance of 0.5dB. The stopband is a region from [1600, 1700] Hz with a maximum gain of -100dB. The filters are then applied to a song with noise in the stopband region; the filtered song can then be heard. For each filter, a brief description, the order, add/multiple operations per input sample, magnitude/phase response, group delay, pole-zero diagram, and impulse response are included.

2 Filter Design & Implementation

Many factors determine the quality of a filter design. Obviously, the filter must attenuate a signal by the prescribed amount. However, an ideal order is as low as possible; larger order requires more processing power and time. This time can be approximated by examining the number of additions and multiplications required for each input signal sample in the filter convolution: $2n + 1$, where n is the filter order.

Also important is the group delay, the derivative of phase. A non-uniform group delay can cause distortion in the output signal, as some frequencies are delayed more than others.

The MATLAB code for each filter is provided in section 5, the appendix.

2.1 Butterworth: Order 18

Butterworth filters are designed to be maximally flat; when compared to other filters such as Chebyshev and Elliptic, it has a slower roll off, but no ripple near the corner frequencies. This roll off occurs at -20N dB per decade, where N is the order of the filter.

Figure 1 displays the magnitude and frequency response. Note how there is no ripple in the magnitude transition, just a smooth decay. Also note the rapid change in the frequency response; this causes the group delay in Figure 2 to have a large spike, non-ideal spike in the stop-band region. The pole-zero diagram in Figure 3 illustrates how the Butterworth had 9 zeros on the unit circle at 1650 Hz surrounded by a semicircle of nine poles. Each zero and pole is accompanied by its complex conjugate mirrored across the real (x) axis. Finally, the impulse response is included in Figure 4.
With a relatively low order, the Butterworth design only needs 37 add/multiply operations per input sample.

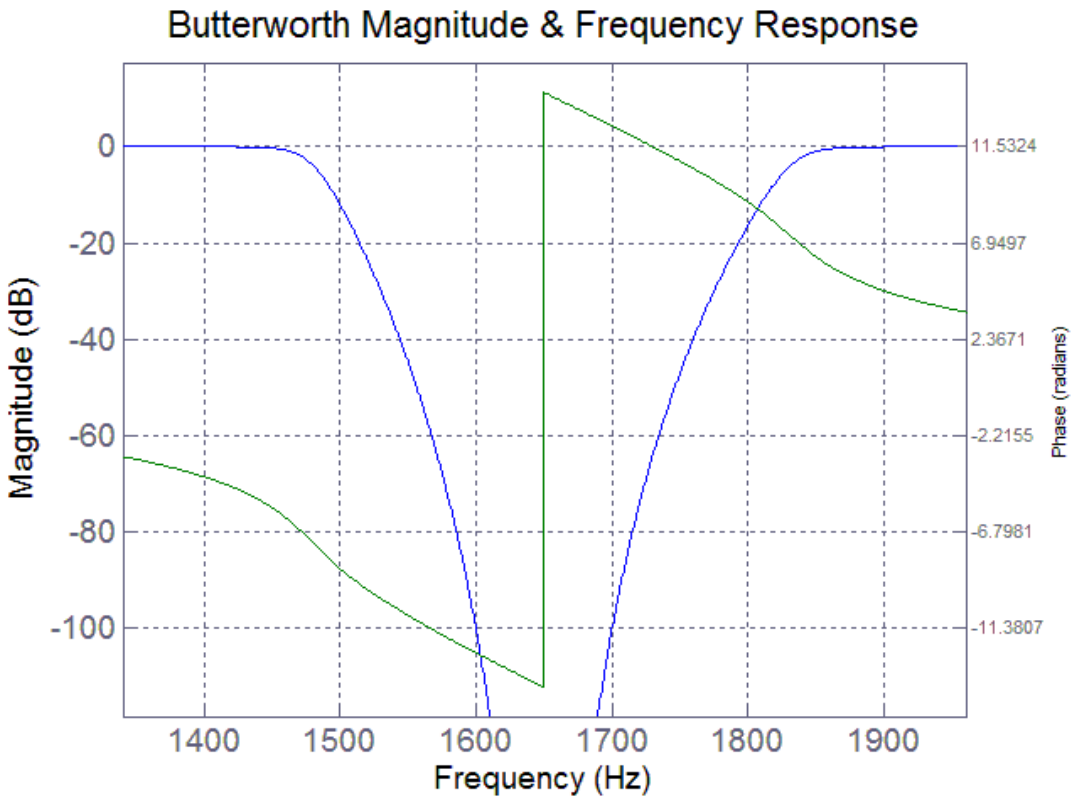

Figure 1: Butterworth Magnitude & Frequency Response

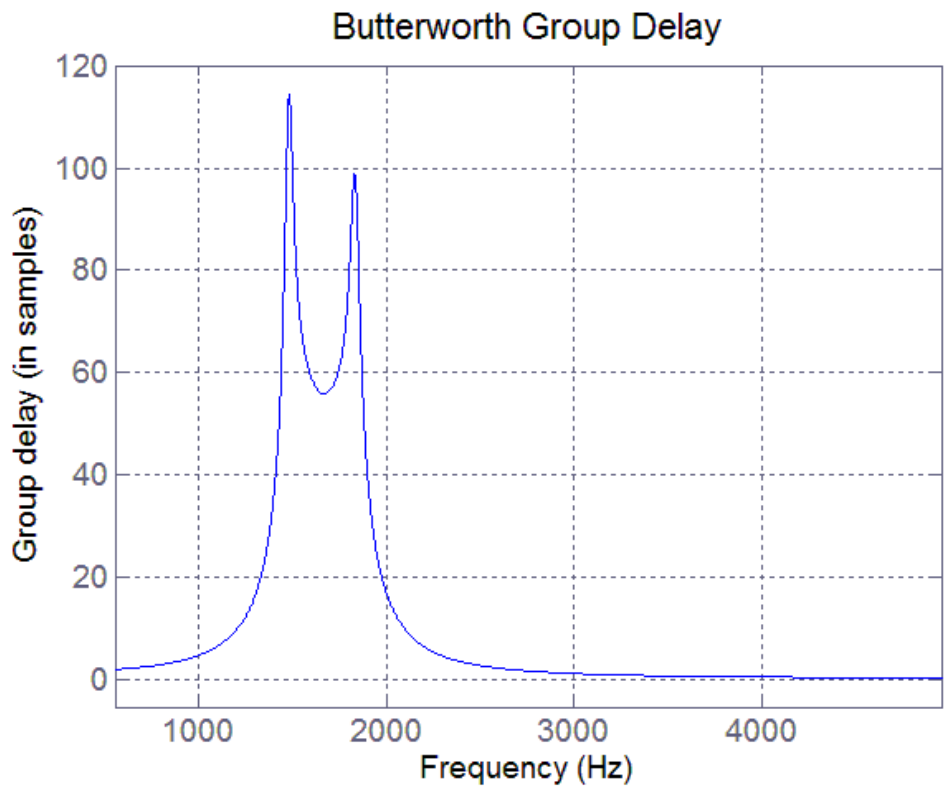

Figure 2: Butterworth Group Delay

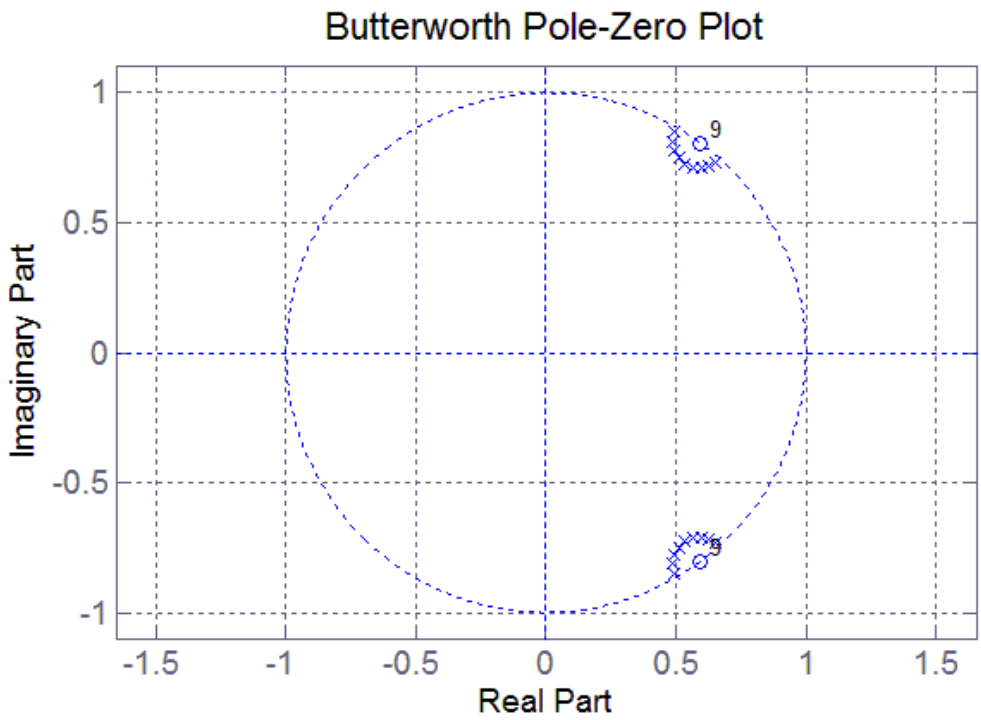

Figure 3: Butterworth Pole-Zero Plot

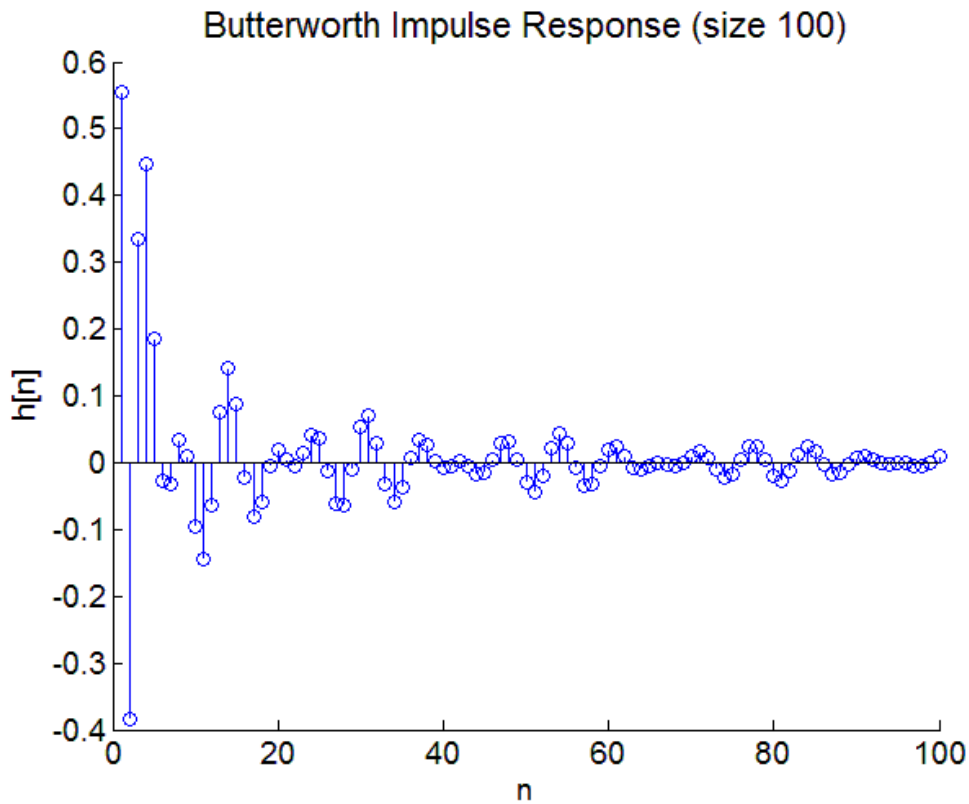

Figure 4: Butterworth Impulse Response

2.2 Chebyshev Type I: Order 12

Chebyshev filters are based on a concept called Chebyshev polynomials; put simply, these coefficients describe polynomials that are bounded to a one by one box centered at the origin and diverge elsewhere. Though these filters have a steeper roll off than Butterworth, they have ripple in the passband.

The ripple is difficult to see in Figure 5, the magnitude as phase plot, so it is the focus of Figure 6, a plot of the magnitude zoomed in on the passband ripple. Note the difference between Figure 5 and Figure 1, the magnitude responses of the Chebyshev Type I and Butterworth filters, respectively. Though Chebyshev filters have a faster roll off, the rate of attenuation change seems similar here. This is because the order of the Butterworth is 150% that of the Chebyshev. The phase response and group delay, found in Figure 7, are very similar to the Butterworth design. The pole-zero diagram, Figure 8, is different than that of the Butterworth, but has the same idea: surround *n* zeros locally with *n* poles. The impulse response is shown in Figure 9 and is fairly similar in shape to the Butterworth. 25 add/multiply operations are required.

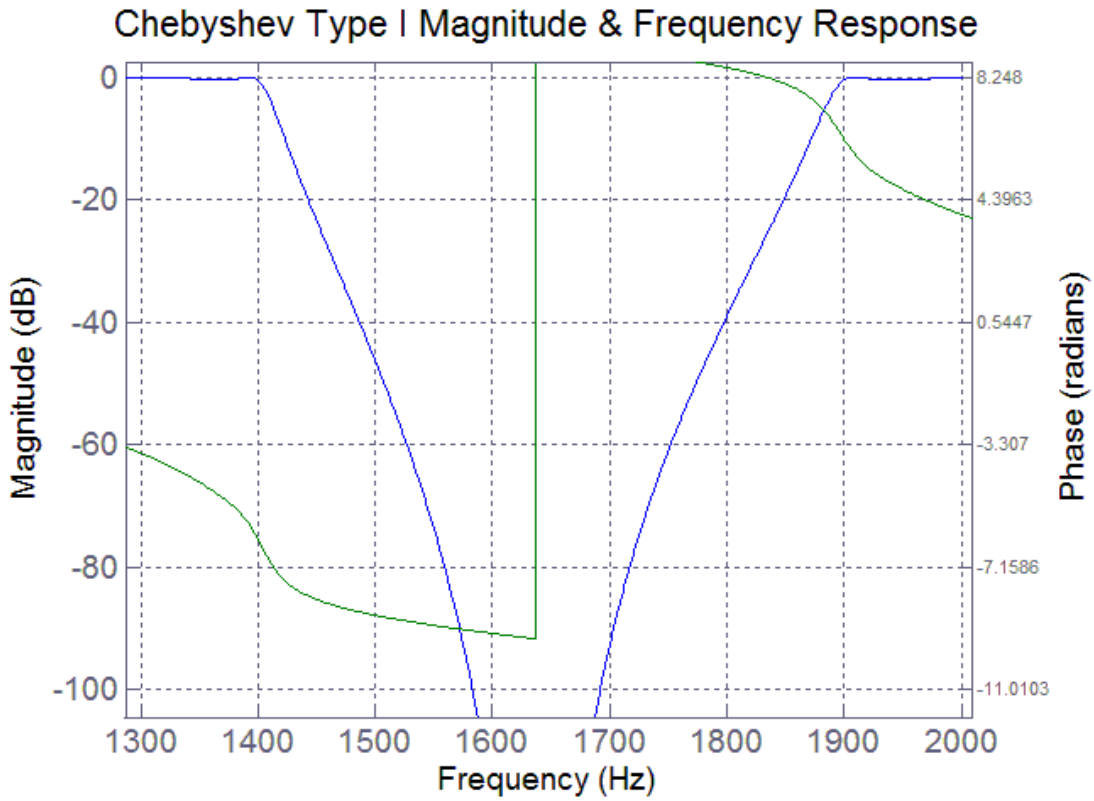

Figure 5: Chebyshev Type I Magnitude & Frequency Response

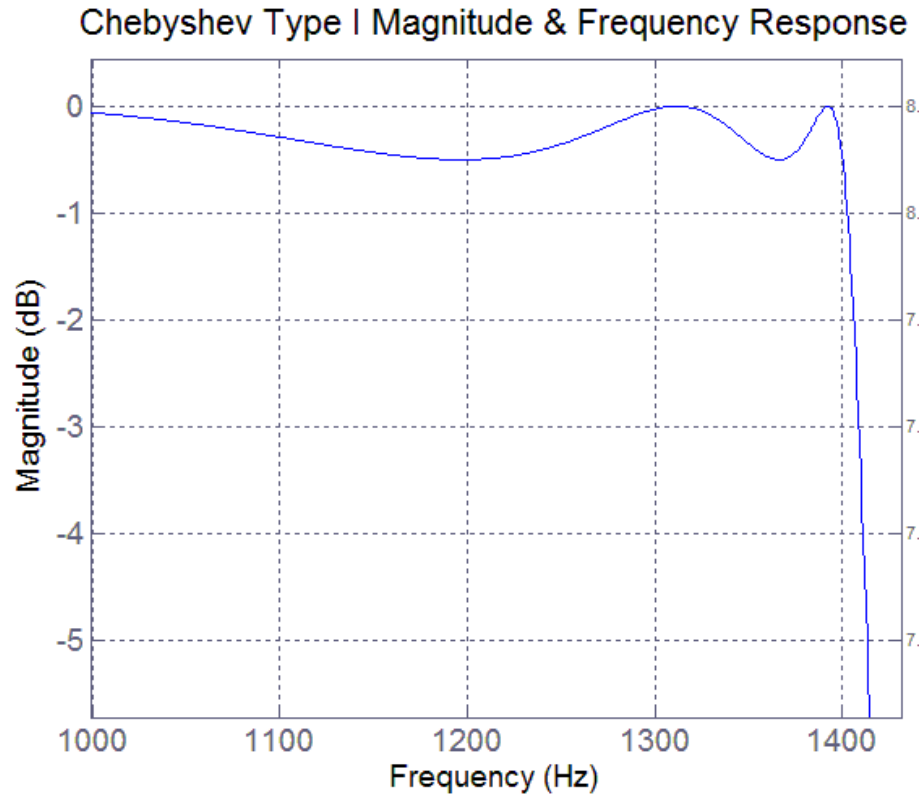

Figure 6: Chebyshev Type I magnitude, zoomed to show ripples

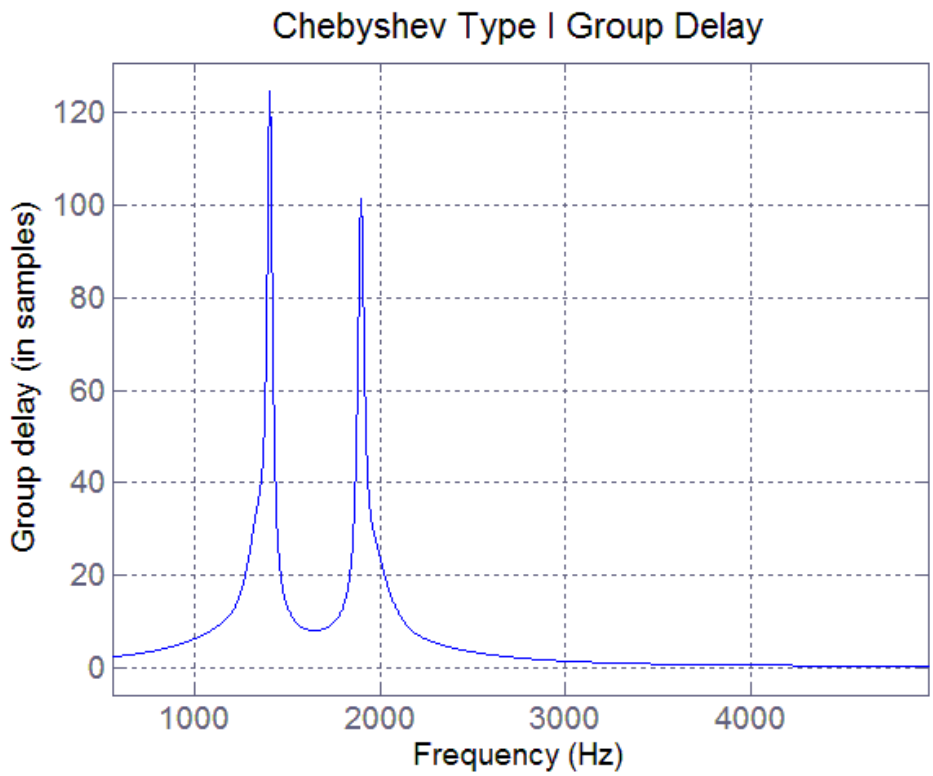

Figure 7: Chebyshev Type I Group Delay

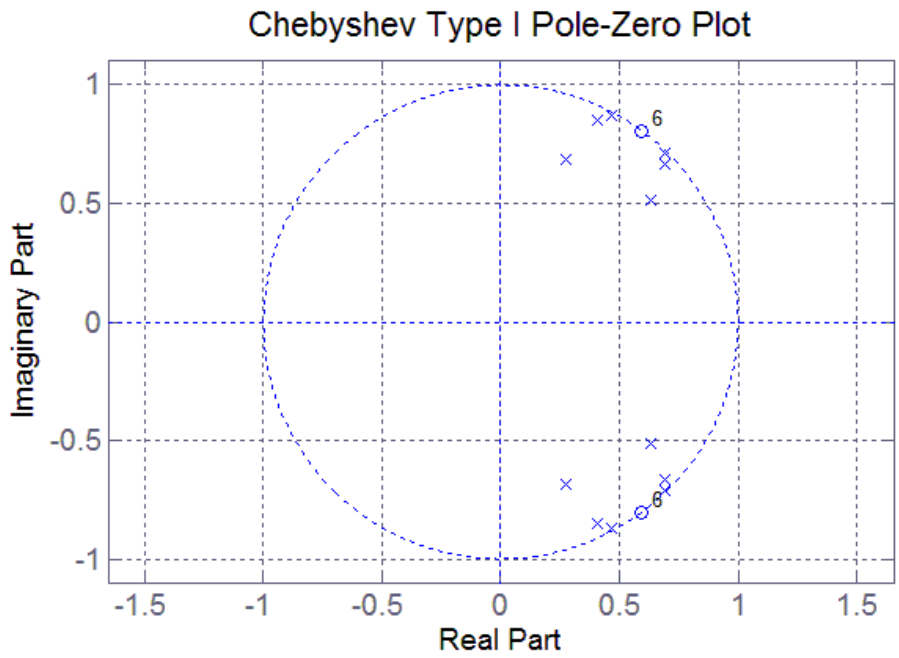

Figure 8: Chebyshev Type I Pole-Zero Plot

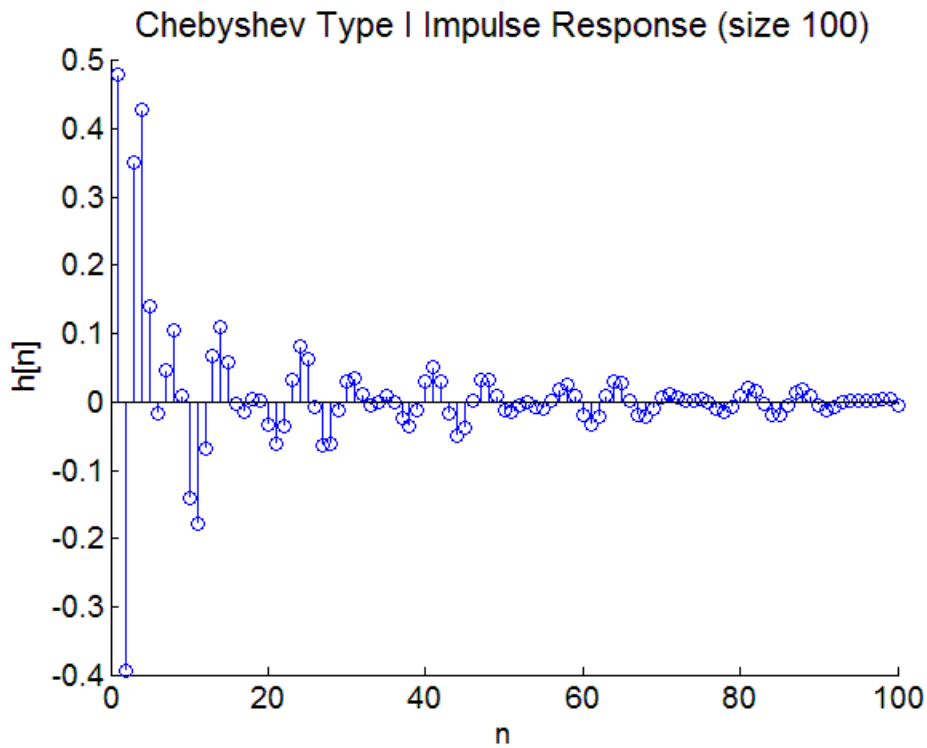

Figure 9: Chebyshev Type I Impulse Response

2.3 Chebyshev Type II: Order 12

The Chebyshev Type II filter is very similar to the Type I in section 2.2. However, instead of ripple in the passband, it occurs in the stopband. Figure 10, the magnitude and phase response, shows how the Type II filter rolls off smoothly (with no ripple), then ripples between 1600 and 1700 Hz before smoothly returning to the passband. The phase response, also in Figure 10, is slightly different from Butterworth and Chebyshev Type I in that there is a slight saw-tooth effect in the phase transition. However, while the group delay in Figure 11 appears very similar in shape to the previous two filters, it is around half in magnitude. The pole-zero diagram in Figure 12 more closely resembles the Butterworth; the zeros are not in one location, but the poles surround the zeros in a semi-circle fashion. The impulse response in Figure 13 takes a similar form to the previous two filters with different coefficient magnitudes. The order of this filter is the same as Chebyshev Type I, 12, and therefore requires 25 add/multiply operations.

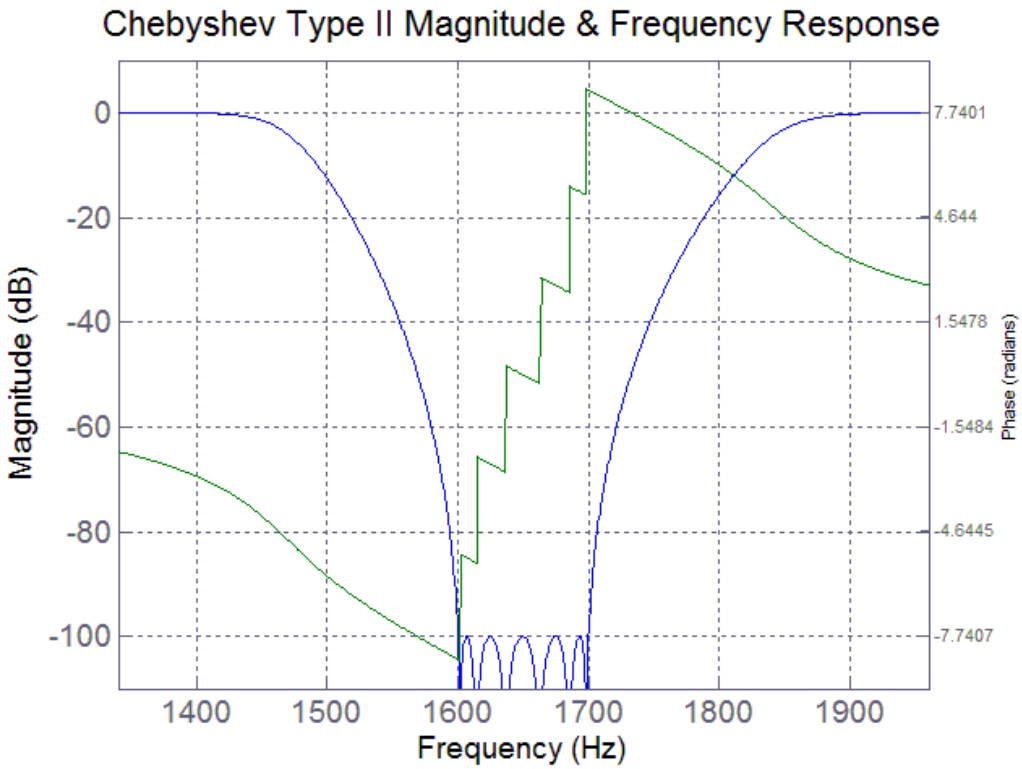

Figure 10: Chebyshev Type II Magnitude & Frequency Response

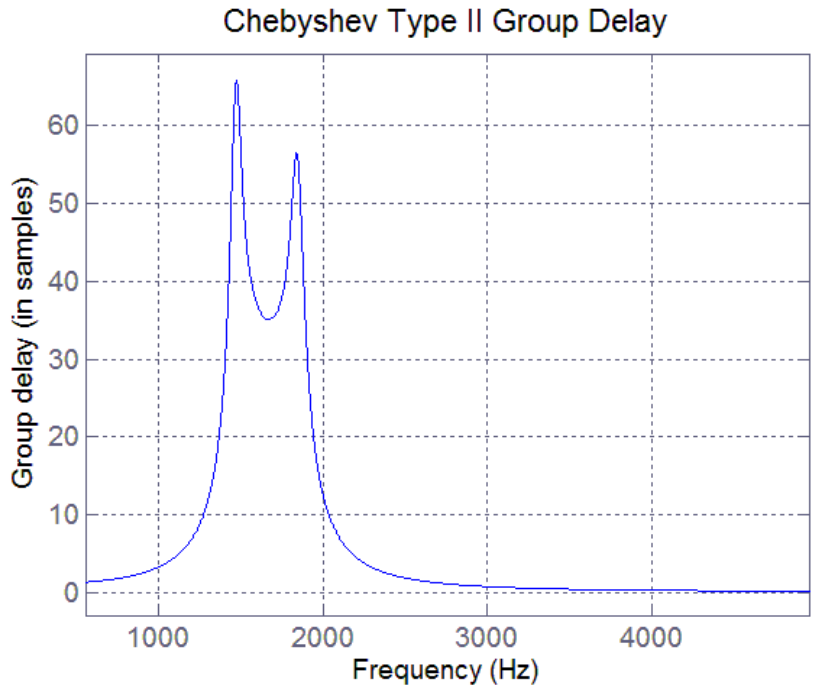

Figure 11: Chebyshev Type II Group Delay

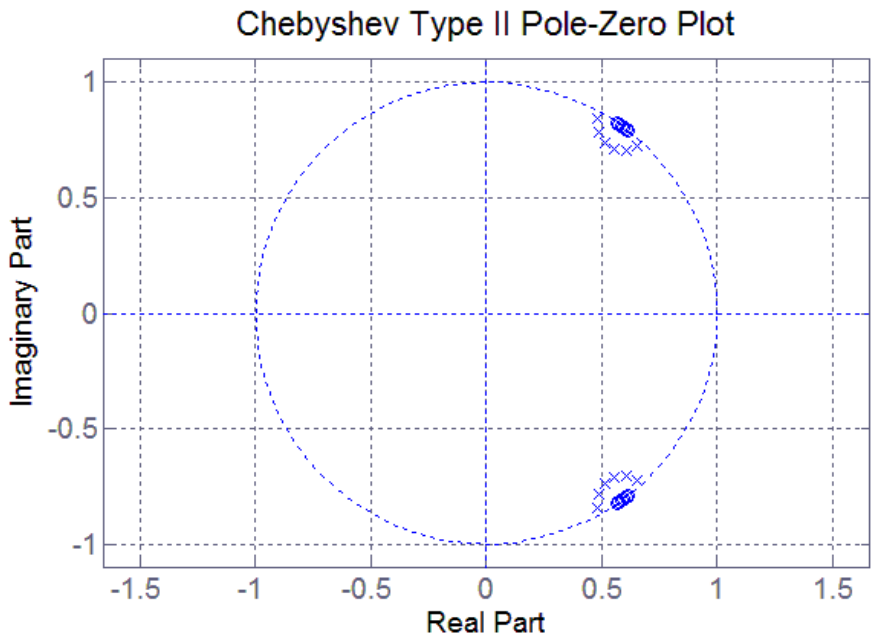

Figure 12: Chebyshev Type II Pole-Zero Plot

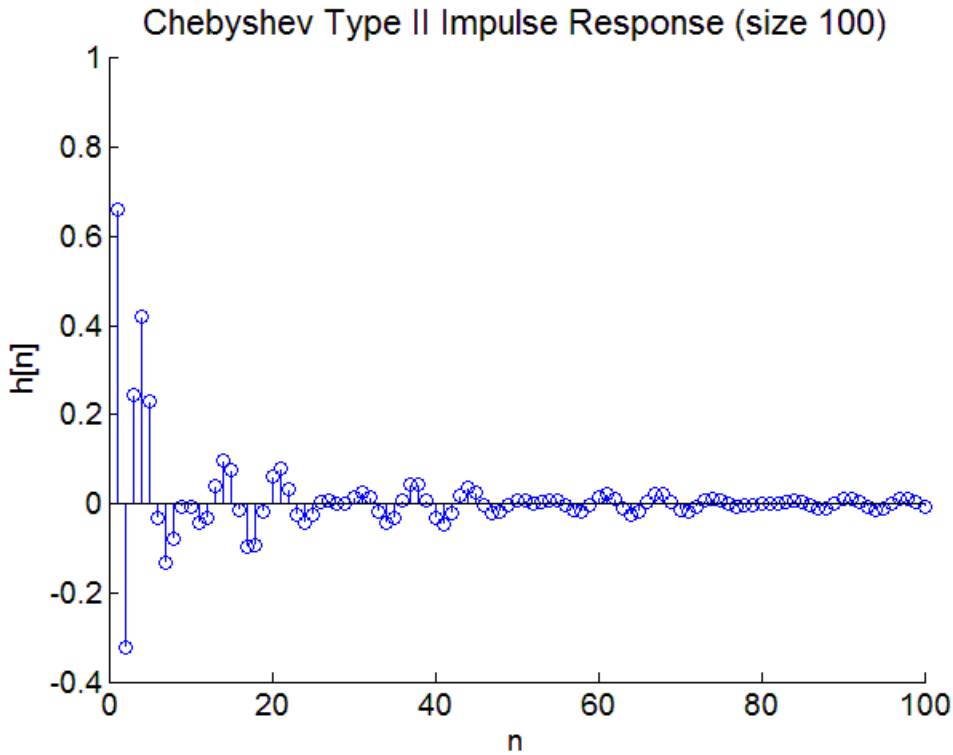

Figure 13: Chebyshev Type II Impulse Response

2.4 Elliptic: Order 12

The elliptic filter combines the two properties of the Chebyshev Type I and II filters; it leverages equalized ripple in the both passband and stopband. Figure 14 confirms this; it appears to be the hybrid of the previous two filters ripple bands. Figure 15, the close up of the passband ripple, can be compared to Figure 6, the same figure for Chebyshev Type I. The only difference is a slight compaction of the elliptic passband ripple response towards the right. The phase has a similar localized "stairway" or sawtooth response as Chebyshev Type II, likely caused by the ripples in stopband.

The group delay, found in Figure 16, resembles the previous three filters in shape, but only Butterworth and Chebyshev Type I in magnitude. Figure 17 shows the pole-zero diagram; this is analogous to the Chebyshev Type I plot, Figure 8, in that the poles surround the zeros in a pattern unlike the semicircle of the Butterworth and Chebyshev Type II. The impulse response in Figure 18 also resembles the three previous filters with different magnitude. With an order of 12, the elliptic filter has the name number of add/multiply operations as both Chebyshevs: 25.

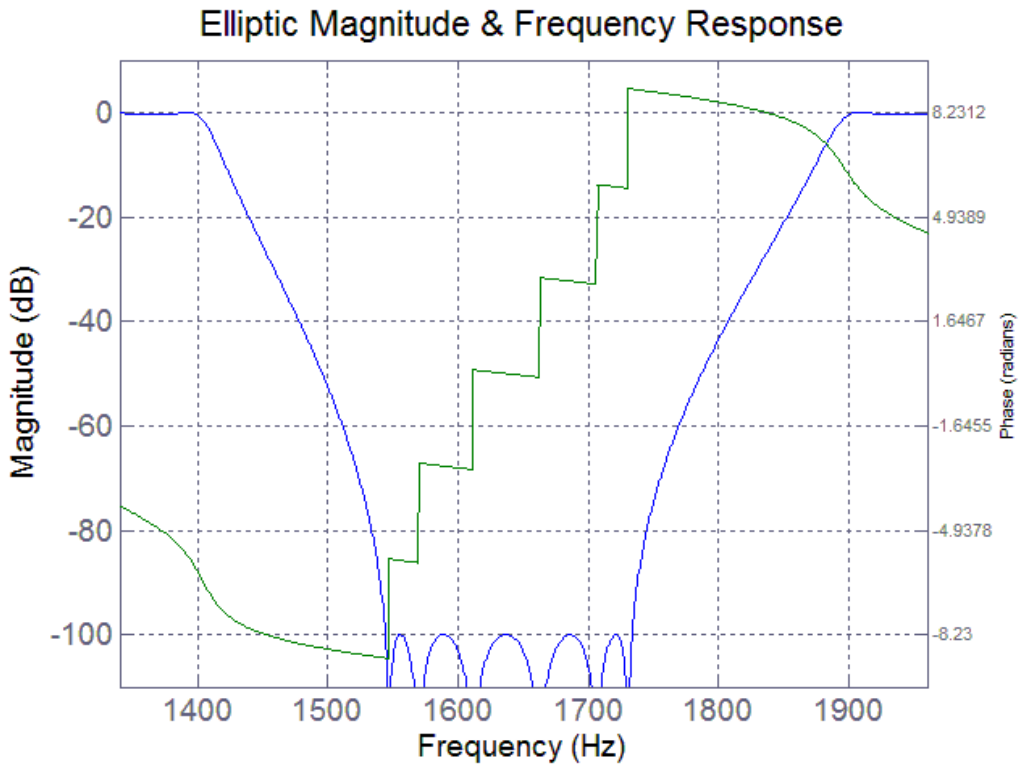

Figure 14: Elliptic Magnitude & Frequency Response

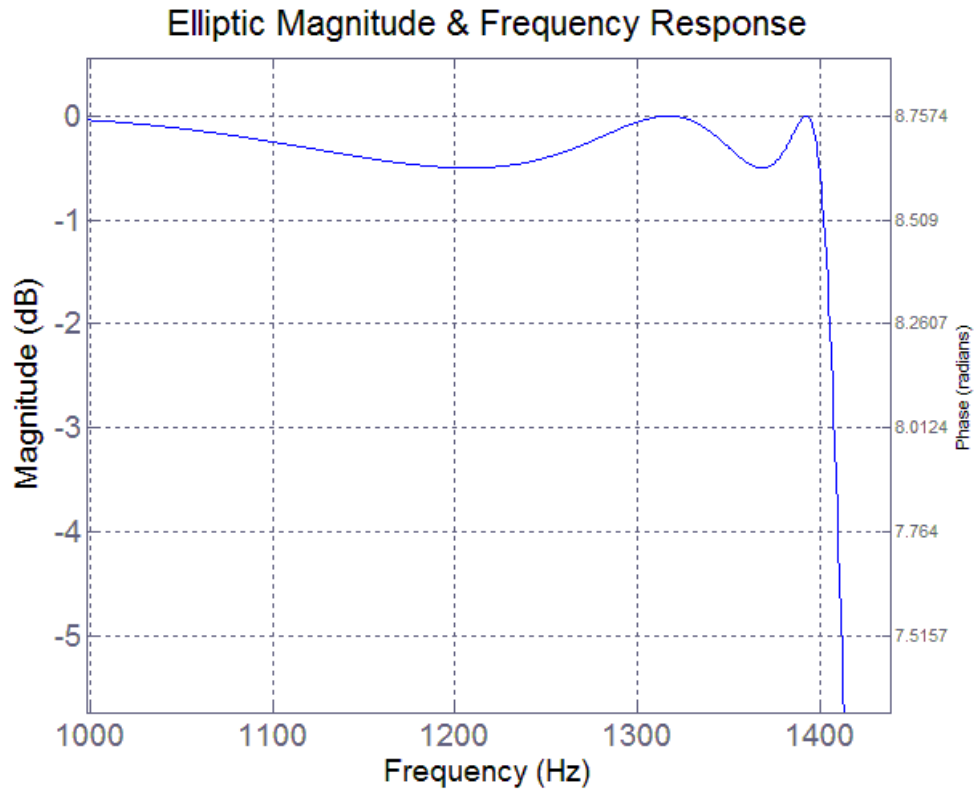

Figure 15: Elliptic magnitude, zoomed to show ripples

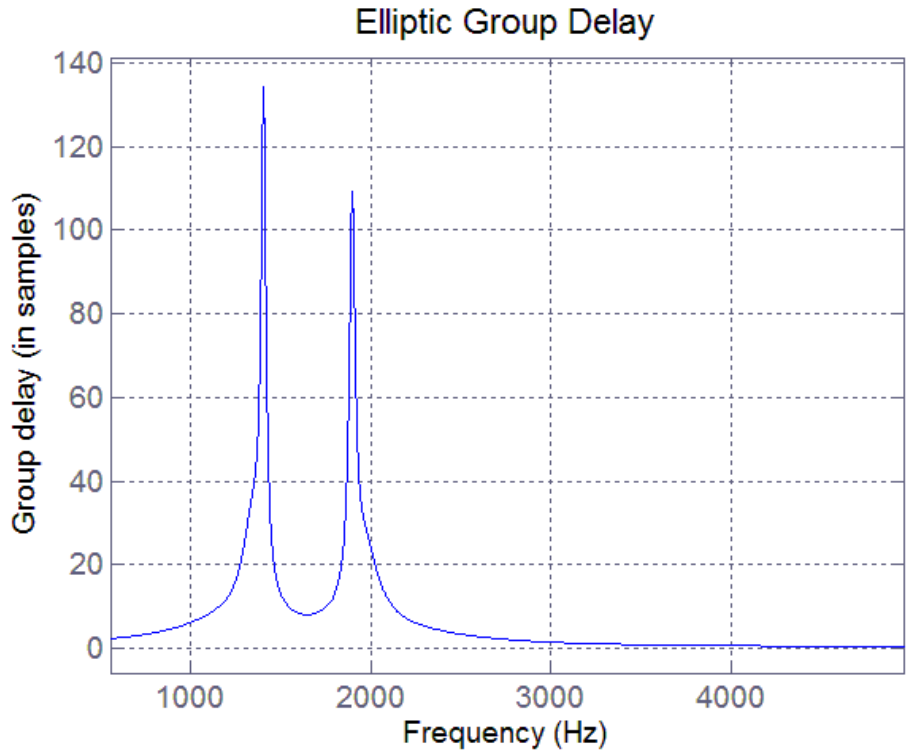

Figure 16: Elliptic Group Delay

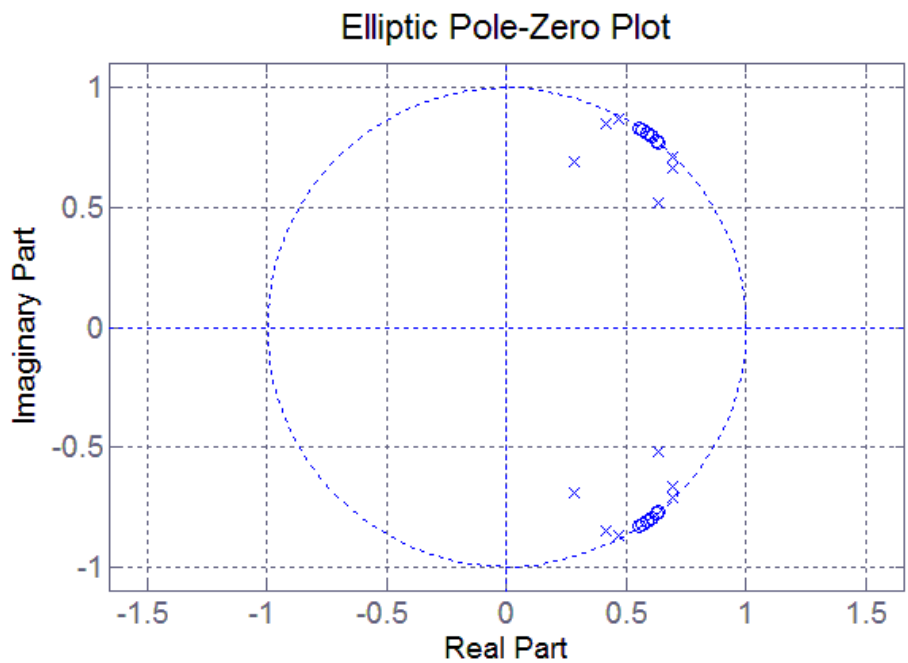

Figure 17: Elliptic Pole-Zero Plot

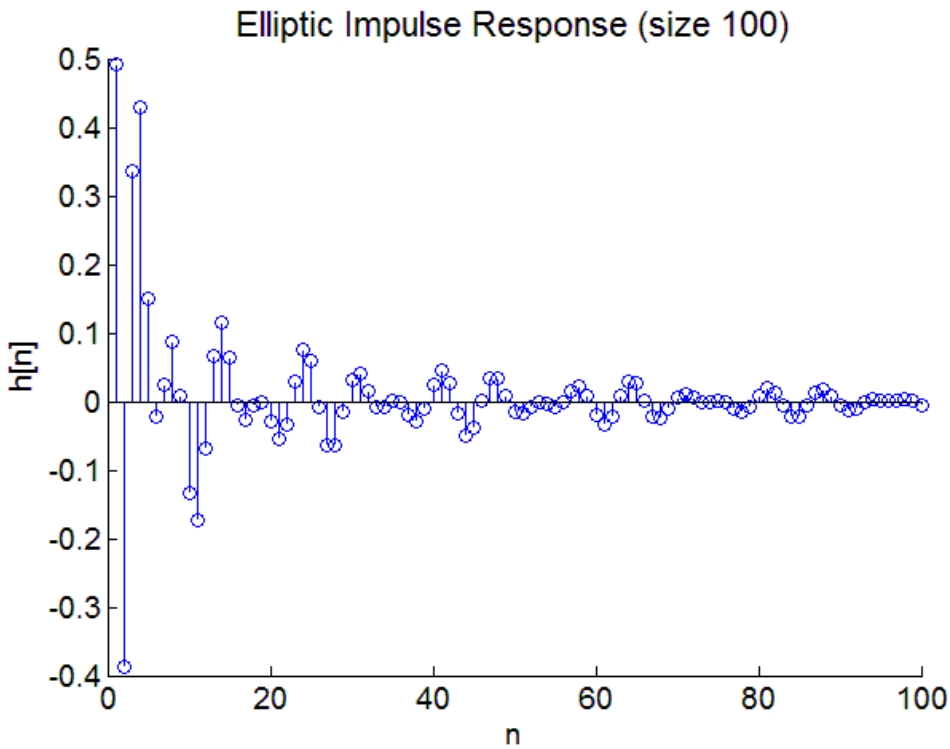

Figure 18: Elliptic Impulse Response

2.5 Kaiser: Order 354

The Kaiser filter is the first of this set that is a window function, it is only "on" for a certain time $0 < n <$ M. The criteria are to minimize the main lobe of the window, as well as minimize the energy in the sidelobes. The coefficients are based off a modified Bessel function. Though the order is much higher than the previous four, this filter has linear phase, meaning all frequencies are delayed an equal amount of time. This can be seen in the group delay plot in Figure 20; note how the group delay τ = M (the order) / 2.

The magnitude response, seen in Figure 19 is not significantly different from the Chebyshev Type II filter, with a slow roll-off and ripple in the stopband. However, the stopband's ripple is not uniform. Figure 19's lowest displayed magnitude is -130dB, where as the four previous filters' are -110dB. The phase is most notable in this figure as it has a constant magnitude of slope. The pole-zero plot, found in Figure 21, also takes a different approach than all the previous filters. Since it is finite impulse response (FIR) and linear phase, all 354 poles are at the origin. The zeros are spread throughout, surrounding unit circle and focusing in on unit circle in the stopband.

Following the familiar trend, the impulse response in Figure 22 is not similar to those previously explored. The response is centered around the group delay and has a length of 354; an impulse length 100 simply did not convey the true impulse response.

As the order is very high, 709 add/multiply operations are required.

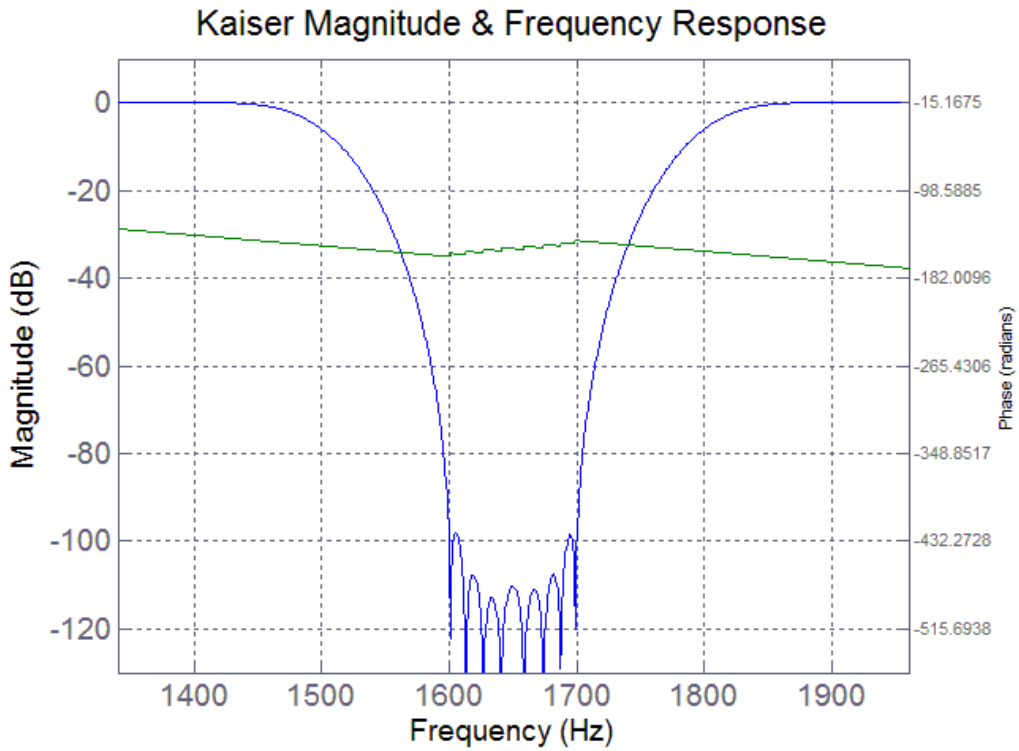

Figure 19: Kaiser Magnitude & Frequency Response

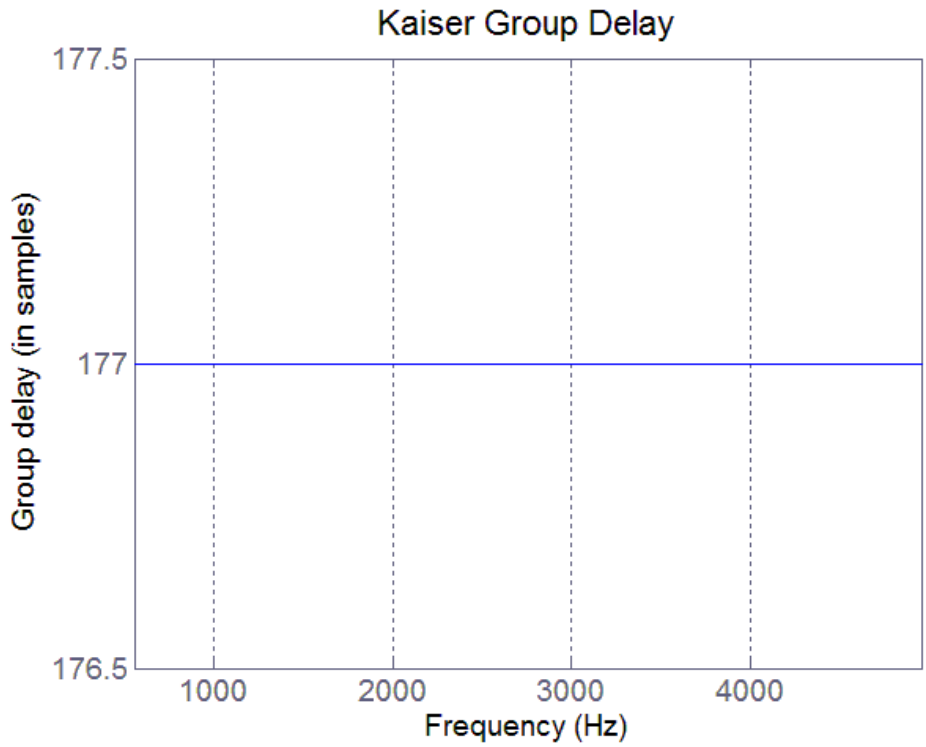

Figure 20: Kaiser Group Delay

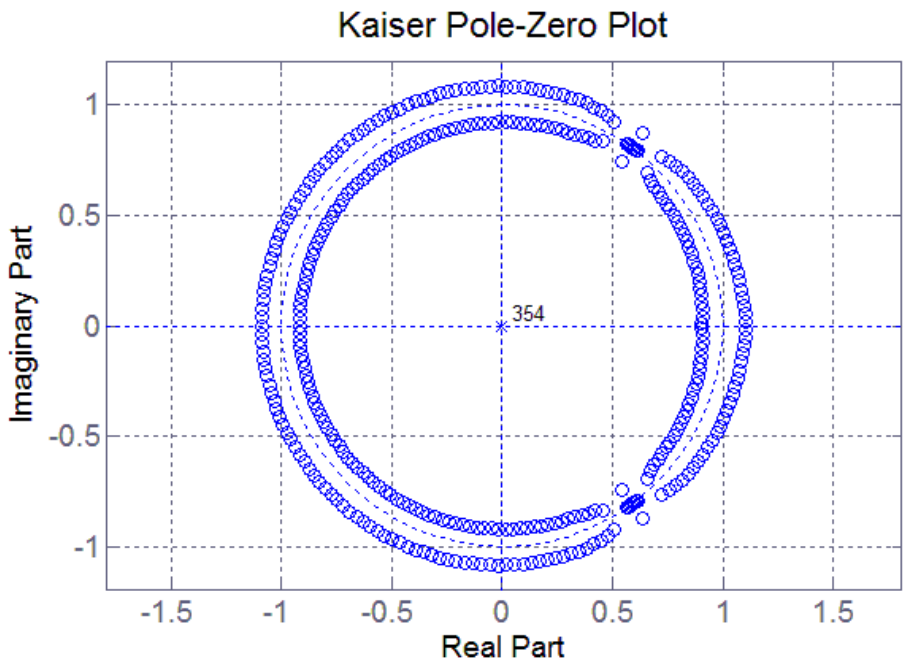

Figure 21: Kaiser Pole-Zero Plot

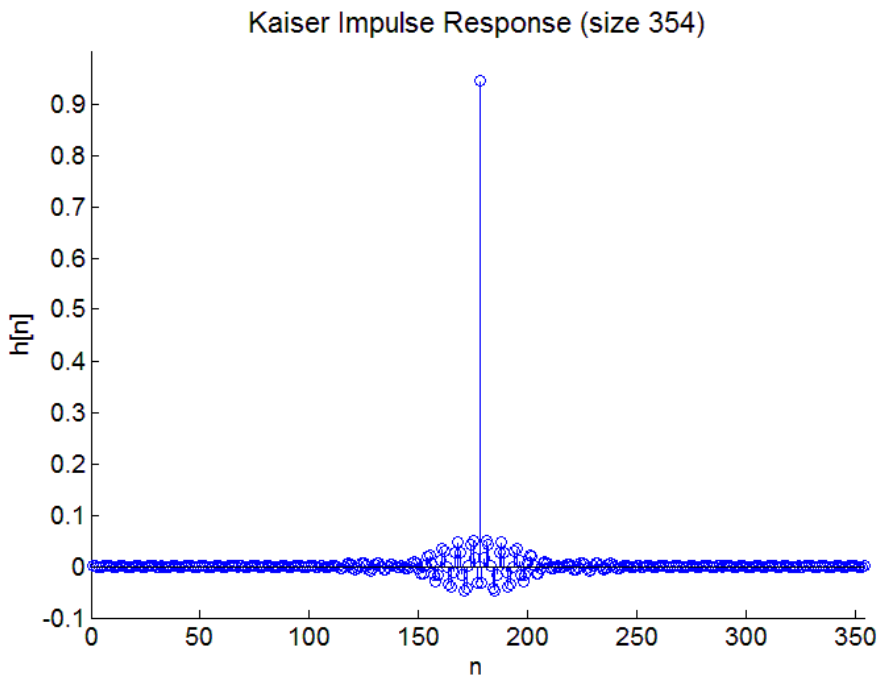

Figure 22: Kaiser Impulse Response

2.6 Parks-McClellan: Order 184

The Parks-McClellan filter is very similar to the Kaiser in section 2.5. However, instead of a single formula, it is an algorithm to find the correct order and filter type to match the parameters. The general concept is that a polynomial can be fit to the parameters of the filter such that it touches the bounding box, alternating strictly between overshoot and undershoot.

It has the same basic frequency results as Kaiser, though there are fewer and more even ripples in the stopband, which can be seen in Figure 23. The phase is again linear, which is reflected in the group delay of Figure 24. The group delay is a constant, half of the filter order. The pole-zero plot in Figure 25 is again similar to Kaiser, though there are fewer poles spread further apart and a zero out at around 2.6. This zero is the inverse of the one at around 0.4; linear phase implies that if z_0 exists, so does z_0^{-1}. The impulse response in Figure 26 is size 184, not 100, to show how it is centered at the group delay ($\tau = 92$). 369 add/multiplies are required. This filter requires 369 add/multiplies per input sample

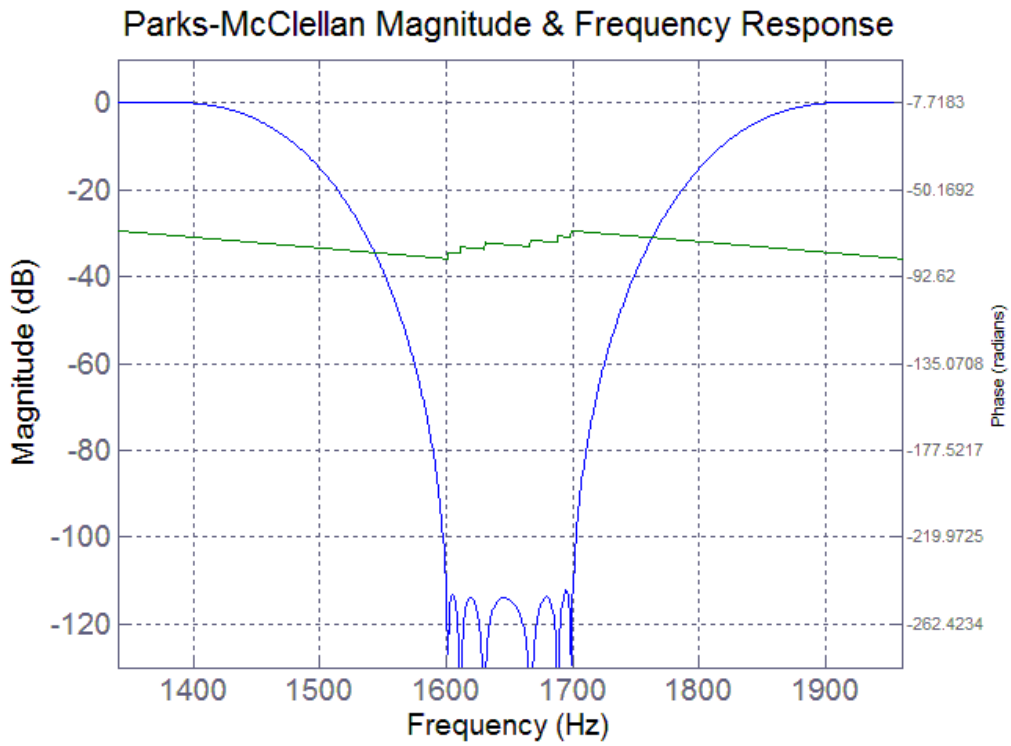

Figure 23: Park-McClellan Magnitude & Frequency Response

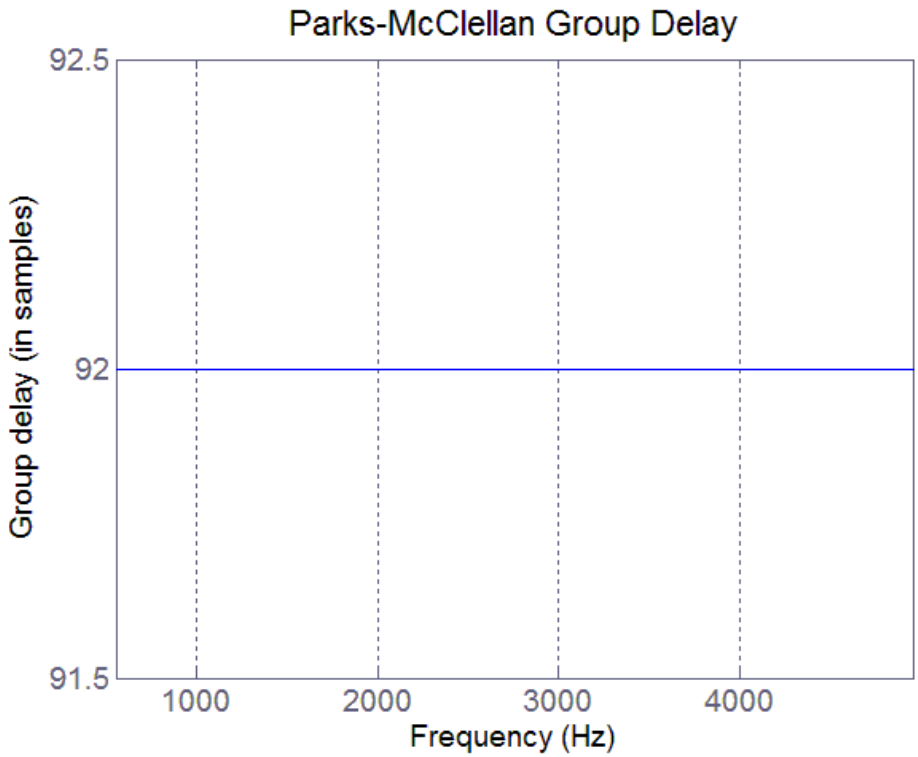

Figure 24: Parks-McClellan Group Delay

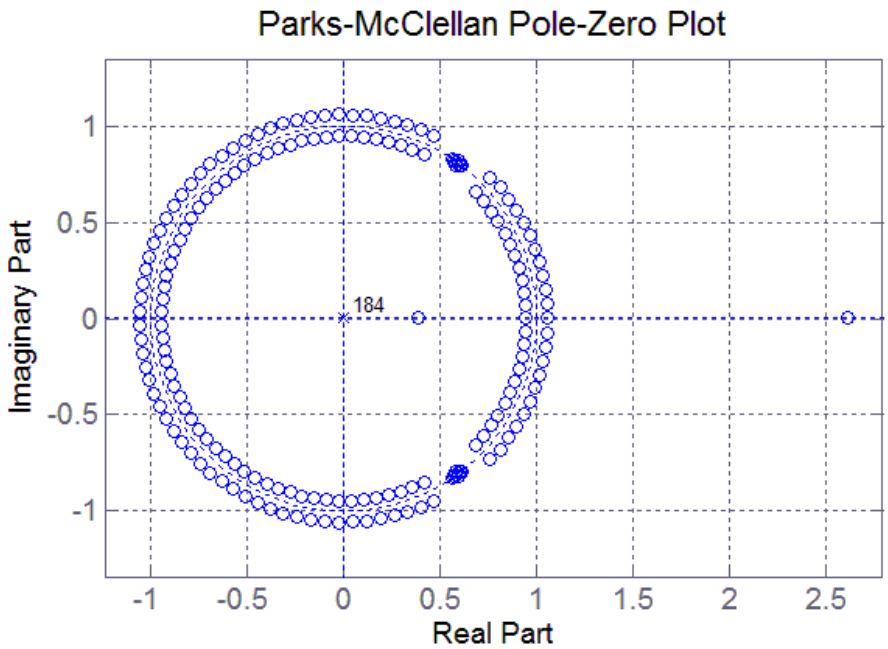

Figure 25: Parks-McClellan Pole-Zero Plot

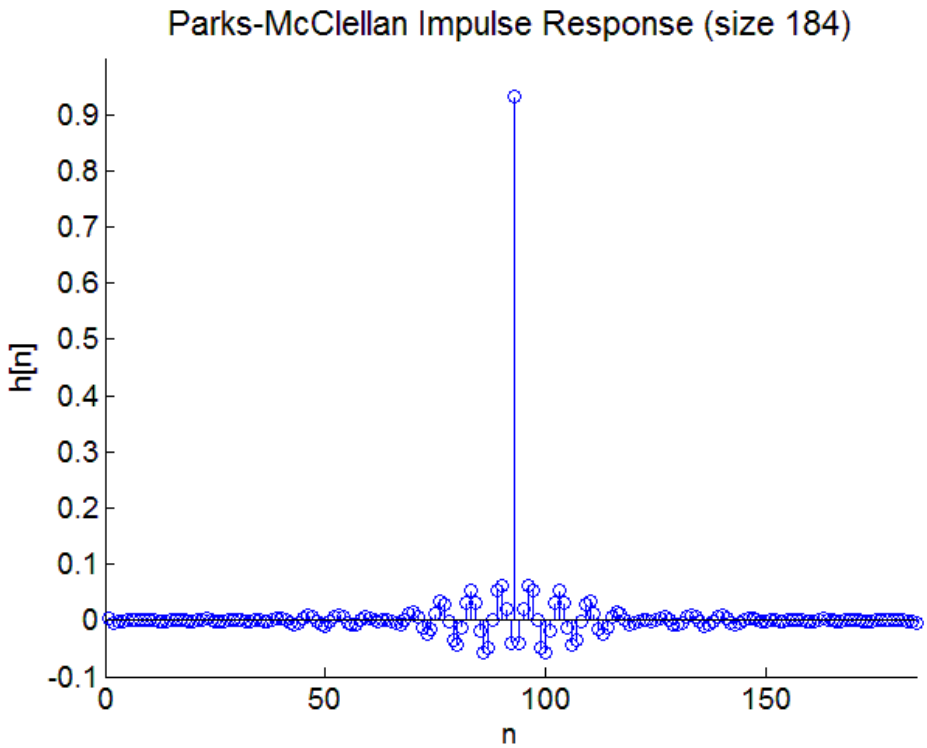

Figure 26: Parks-McClellan Impulse Response

3 Filter Performance

Filter	Order	Linear Phase?	Quality
Butterworth	18	No	Sound quality ok, slight ringing in background
Chebyshev Type I	12	No	Better than Butterworth, no background ringing
Chebyshev Type II	12	No	Similar to Butterworth with background ringing
Elliptic	12	No	Similar to Chebyshev Type I, no ringing
Kaiser	354	Yes	Worse ringing than Butterworth
Parks-McClellan	184	Yes	Slight ringing

Table 1: Comparison of filter types

As can be seen from Table 1, the filters that were noticeably better were the Chebyshev Type I and Elliptic. This was due to the fact they didn't have a high frequency ringing in the background, which was likely due to the filter not destroying all the noise. Though the two filters with linear phase probably had good sound quality because of it, it was overshadowed by the ringing. The reason behind this can be clearly seen in Figure 27, the frequency domain representation of three signals.

The black signal, mostly hidden behind the other two signals, is the noise; it can predominantly be seen in the stopband region of the filters. The next layer is the Kaiser filter; while it eliminates the majority of the noise in the stopband, there are two regions on either side that the Elliptic filter (blue) removes that the

Kaiser does not. This is the reason for the "ringing." Examining the magnitude responses of the two filters, Figure 19 and Figure 14, at 1500Hz, the elliptic filter attenuates the signal by -50dB, while the Kaiser attenuates it by only about -5dB.

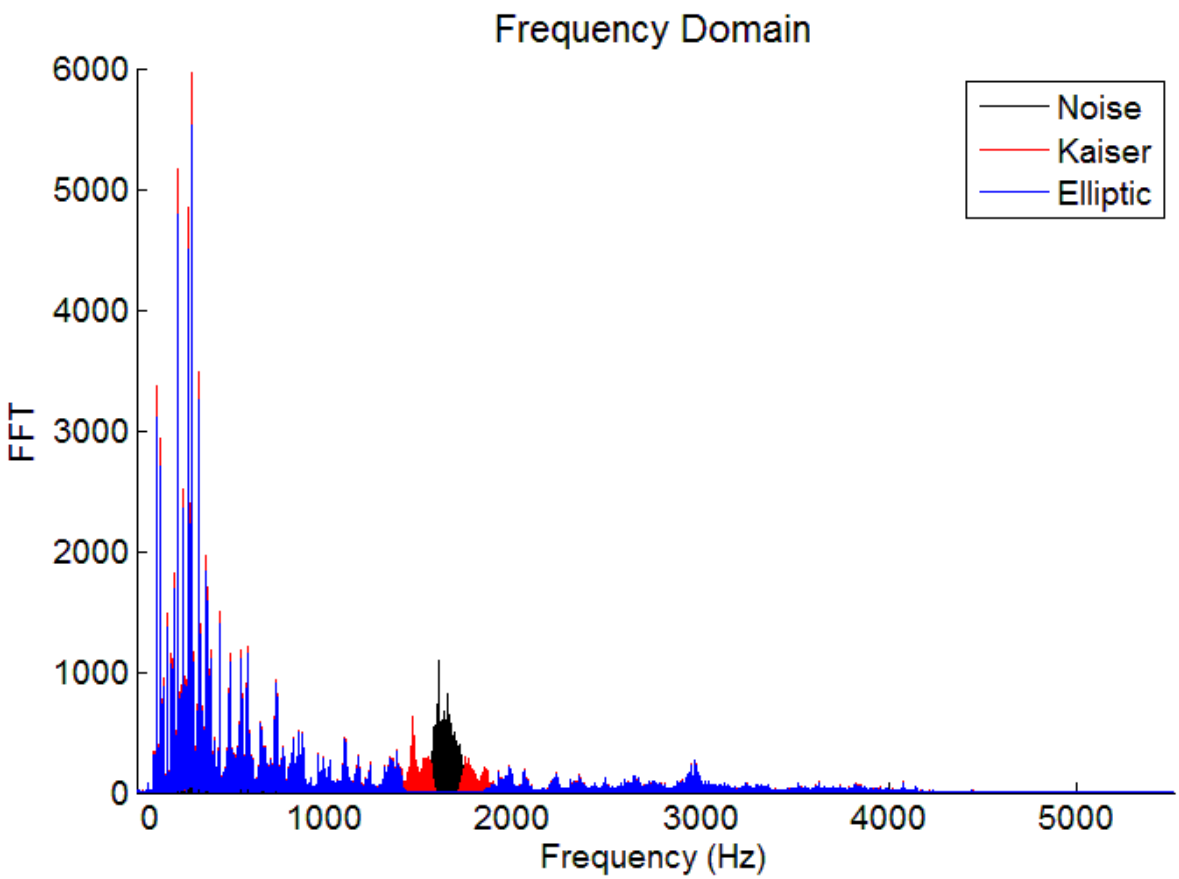

Figure 27: Frequency Domain

The best filter in terms of sound quality was the elliptic. However, even though it does have a low order, it is still an infinite impulse response (IIR) filter without linear phase. The Parks-McClellan or Kaiser would likely provide a better response if they had a slightly wider stopband.

4 Mitigating True Signal Loss

As can be seen from Figure 27, the elliptic filter has a fair amount of frequency information in 2,000 to 3,000 Hz range but very little in the 1,400 to 1,900 Hz range once the filter has been applied. A possible solution to this would be to remove the noise as above, and then subtract the filtered signal from the original signal. What is left is the noise and some of the original song that occurs in the stopband range.

If the noise is uniform, it can be averaged; that average noise can then be subtracted from the original signal, leaving some noise but mostly the signal itself. This "average" can get smarter, possibly running over parts of the noise band instead of the entirety of it in the case that the noise is concentrated in one a main lobe, but exists to the sides as well.

5 Appendix

Each section was a cell in a MATLAB file. Section 5.1, setup, must be run before all else. It need only be run once per session. Sections 5.2 – 5.7 generate the system *Hd*, which can then be plotted by Section 5.8. Section 5.9 generates Figure 27, the frequency domain, of the noise, elliptic, and Kaiser filters.

5.1 Setup

```
%% Setup

nyq = 11025 / 2; % Define nyquist frequency
Wp = [1400 1900] / nyq; % Passband ranges
Ws = [1600 1700] / nyq; % Stopband ranges
Rp = .5; % Ripple in passband
Rs = 100; % Ripple in stopband

fcuts = [1400 1600 1700 1900]; % Cutoff frequencies
mags = [1 0 1]; % Magnitudes (stopband)
devs = [10^(1/40)-1 10^-5 10^(1/40)-1]; % ripples

[noise, Fs] = wavread('noisy.wav'); % Read the noise file
```

5.2 Butterworth

```
%% Butterworth

type = 'Butterworth';

[n, Wn] = buttord(Wp, Ws, Rp, Rs); % Calculate the order
disp([type ' order: ' num2str(2*n)]); % Display the order
[z,p,k] = butter(n,Wn,'stop'); % Create the filter

[sos,g] = zp2sos(z,p,k); % Convert to an sos system
Hd = dfilt.df2tsos(sos,g); % Create the dfilt obj
```

5.3 Chebyshev Type I

```
%% Chebyshev Type I

type = 'Chebyshev Type I';

[n, Wn] = cheb1ord(Wp, Ws, Rp, Rs); % Calculate the order
disp([type ' order: ' num2str(2*n)]);

[z,p,k] = cheby1(n,Rp,Wn, 'stop'); % Create the filter
[sos,g] = zp2sos(z,p,k); % Convert to an sos system
Hd = dfilt.df2tsos(sos,g); % Create the dfilt obj
```

5.4 Chebyshev Type II

```
type = 'Chebyshev Type II';

[n, Wn] = cheb2ord(Wp, Ws, Rp, Rs); % Calculate the order
disp([type ' order: ' num2str(2*n)]); % Display the order

[z, p, k] = cheby2(n,Rs,Wn, 'stop');% Create the filter
[sos,g] = zp2sos(z,p,k); % Convert to an sos system
Hd = dfilt.df2tsos(sos,g); % Create the dfilt obj
```

5.5 Elliptic

```
%% Elliptic

type = 'Elliptic';

[n, Wn] = ellipord(Wp, Ws, Rp, Rs); % Calculate the order
disp([type ' order: ' num2str(2*n)]); % Display the order
[z, p, k] = ellip(n,Rp, Rs, Wn, 'stop'); % Create the filter

[sos,g] = zp2sos(z,p,k); % Convert to an sos system
Hd = dfilt.df2tsos(sos,g); % Create the dfilt obj
```

5.6 Kaiser

```
%% Kaiser

type = 'Kaiser';
[n, Wn, beta, ftype] = kaiserord(fcuts, mags, devs, nyq * 2); % Calculate
order
disp([type ' order: ' num2str(n)]); % Display the order

n = n + rem(n,2);
hh = fir1(n, Wn, ftype, kaiser(n+1,beta), 'noscale'); % Create the filter

Hd = dfilt.df2t(hh); % Create the dfilt obj
```

5.7 Parks-McClennan

```
type = 'Parks-McClellan';

[n, fo, ao, w] = firpmord(fcuts, mags, devs, nyq * 2); % Calculate order

disp([type ' order: ' num2str(n)]); % Display the order

b = firpm(n, fo, ao, w); % Create the filter
Hd = dfilt.df2t(b); % Create the dfilt obj
```

5.8 Plotting the filters

```
%% Plot the filter

xtick = (cumsum(ones(1,51)) - 1) * 100 * 2 / 11025; % Define freqency in hz

h = fvtool(Hd); % Display the magnitude
set(h,'analysis','freq'); % Set analysis to frequency
set(gca, 'Fontsize', 14); % Set fontsize
set(gca, 'XTick', xtick); % Set xaxis to evenly space frequency
set(gca, 'XTickLabel', num2str(xtick' * 11025 / 2)); % Set xaxis to hz
xlabel('Frequency (Hz)', 'FontSize', 14); % Label xaxis
set(get(gca, 'YLabel'), 'FontSize', 14); % Set yaxis fontsize
title([type ' Magnitude & Frequency Response'], 'FontSize', 16); % Title
xlim([.2431 .3556]); % Set x bounds, 1340 to 1960 hz
ylim([-110 10]); % set y bounds

% Group Delay plot
grpdelay(Hd); % Plot group delay
set(gca, 'Fontsize', 14); % Set fontsize
set(gca, 'XTick', xtick(1:10:end)); % Set xaxis to evenly space frequenc
set(gca, 'XTickLabel', num2str(xtick(1:10:end)' * 11025 / 2)); % Set xaxis to
hz
xlabel('Frequency (Hz)', 'FontSize', 14); % Label xaxis
set(get(gca, 'YLabel'), 'FontSize', 14); % Set yaxis fontsize
title([type ' Group Delay'], 'FontSize', 16); % Title
xlim([.1 .9]); % Set x axis bounds

% Pole-Zero plot
zplane(Hd, 'off'); % Plot the pole-zero diagram
set(gca, 'FontSize', 14); % Scale font size
title([type ' Pole-Zero Plot'], 'FontSize', 16); % Title
set(get(gca, 'XLabel'), 'FontSize', 14); % X label font size
set(get(gca, 'YLabel'), 'FontSize', 14); % Y label font size

% Impulse Response
figure; % new figure
hold on;
set(gca, 'FontSize', 14); % set font size
title([type ' Impulse Response (size 100)'], 'FontSize', 16); % title
xlabel('n', 'FontSize', 14); % x label
ylabel('h[n]', 'FontSize', 14); % y label
y = filter(Hd,[1 zeros(1,99)]); % Convolve an impulse with the filter
stem(1:100, y); % Plot the impulse response

% Sound output
output = filter(Hd, noise); % Convolve the signal with the filter
output = output / max(output); % Normalize so the max is 1
sound(output, Fs); % Play the output sound
```

5.9 Frequency Domain

```
% Kaiser
[n, Wn, beta, ftype] = kaiserord(fcuts, mags, devs, nyq * 2); % Calculate
order
n = n + rem(n,2); % Order
hh = fir1(n, Wn, ftype, kaiser(n+1,beta), 'noscale'); % Create the filter
Hd = dfilt.df2t(hh); % Create the dfilt obj
output_kaiser = filter(Hd, noise); % Convolve the signal with the filter
output_kaiser = output_kaiser / max(output_kaiser); % Normalize so the max is
1

% Elliptic
[n, Wn] = ellipord(Wp, Ws, Rp, Rs); % Calculate the order
[z, p, k] = ellip(n,Rp, Rs, Wn, 'stop'); % Create the filter
[sos,g] = zp2sos(z,p,k); % Convert to an sos system
Hd = dfilt.df2tsos(sos,g); % Create the dfilt obj
output_elliptic = filter(Hd, noise); % Convolve the signal with the filter
output_elliptic = output_elliptic / max(output_elliptic); % Normalize so the
max is 1

% FFTs
fft_noise = fft(noise);
fft_kaiser = fft(output_kaiser);
fft_elliptic = fft(output_elliptic);

len = length(fft_noise) / 2;

% Plots
figure;
hold on;
plot(1:len, abs(fft_noise(1:len)), 'k'); % Noise
plot(1:len, abs(fft_kaiser(1:len)), 'r'); % Kaiser
plot(1:len, abs(fft_elliptic(1:len)), 'b'); % Elliptic
xtick = (cumsum(ones(1,6)) - 1) * 1000 * 2 * len / 11025; % Define freqency
in hz
set(gca, 'XTick', xtick); % Set xaxis to evenly space frequency
set(gca, 'XTickLabel', num2str(xtick' * 11025 / 2 / len)); % Set xaxis to hz
set(gca, 'FontSize', 14); % set font size
title('Frequency Domain', 'FontSize', 16); % title
xlabel('Frequency (Hz)', 'FontSize', 14); % x label
ylabel('FFT', 'FontSize', 14); % y label
legend('Noise', 'Kaiser', 'Elliptic');
xlim([0 len]);
```