Practical Filter Design

FINAL PROJECT

Michael Eller | ECEN 4632 | December 17, 2015

Contents

Introduction

Digital filter design is an ongoing field, with countless uses in this Internet of Things age. If a device has almost any sort of friendly user-interface, odds are there's some digital filtering under the hood.

The purpose of this assignment was to introduce us to a practical design of digital filters. We are given a very noisy signal and told to remove the noise that exists between 1600 and 1700 Hz in order to recover the underlying song. This would effectively remove the majority of the white noise from our song. For each filter implementation there is a brief description of the filter, the order, the number of add/multiply operations per input sample required, the magnitude and phase response, group delay, pole-zero diagram, and impulse response.

Filter Design and Implementation

Filter design is usually not as trivial as "design a stopband filter". Many factors, including quality of the filter, order of the filter, linear phase response, group delay, and others, must be accounted for when designing a digital filter.

The minimum requirements for our stopband filter were to have two passbands (0-1400 Hz & 1900-5512.5 Hz) and set a stopband (1600-1700 Hz). The passbands would allow a ripple of at most -±0.5 dB, while the stopband would attenuate at least 100 dB. Ideally, we would like to achieve this with the lowest order filter possible, while still having a uniform group delay.

The required filters were designed using MATLAB's *fdatool* (as seen in Figure 1 below). This tool gives the user an interactive environment to specify filter parameters and design the filters. The tool also provides easy navigation to several useful analyses, such as: magnitude response, phase response, group delay, impulse response, and pole/zero plots. The *fdatool* also allows the designer to easily generate corresponding MATLAB code to implement their filters they created.

The generated MATLAB code for each filter is included in the Appendix.

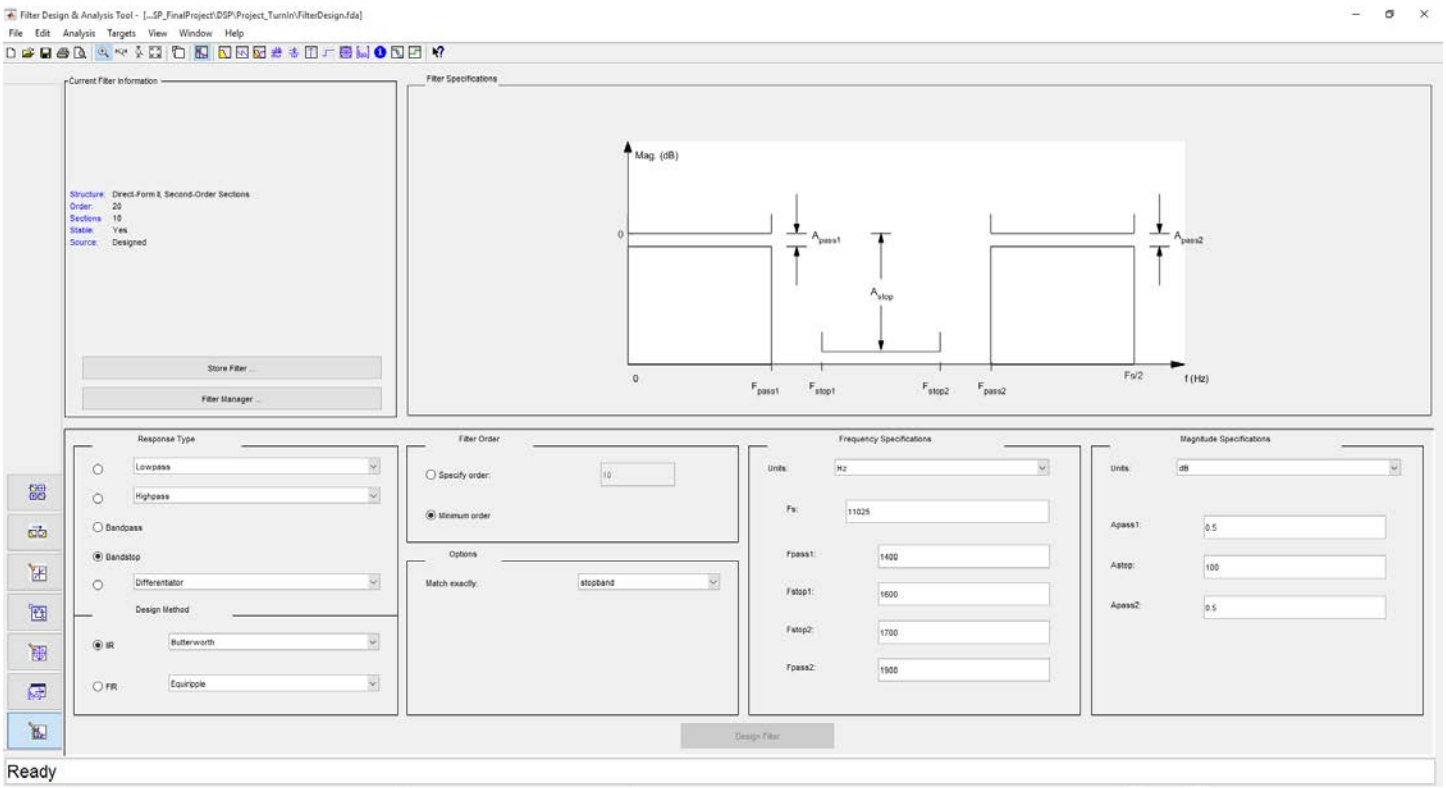

Figure 1: MATLAB *fdatool*

BUTTERWORTH: ORDER 20

First described in 1930 by British engineer and physicist Stephen Butterworth, the Butterworth filter is designed to have as flat a frequency response as possible in the passband. Compared with a Chebyshev Type I/Type II filter or an elliptic filter, the Butterworth filter has a slower roll-off, and thus requires a higher order to implement a stopband. However, the Butterworth filter has a more linear phase response in the passband than compared to the Chebyshev Type I/Type II or elliptic filters.

Figure 2 displays the magnitude and phase response of our Butterworth stopband filter. Observe that there are no ripples in the passband (as further illustrated by Figure 3 below) and the near-linear phase that the Butterworth achieves in the passbands. Unfortunately, because of the Butterworth's rapid change in phase during the stopband, we have a large, non-ideal spike. This result can be seen in Figure 4 below. The pole/zero diagram in Figure 5 below shows the Butterworth's 10 zeros surrounded by a semicircle of 10 poles. These 10 zeros and 10 poles are accompanied by their complex conjugates, mirrored across the real axis. The impulse response as seen in Figure 6 shows how the first few samples will always be much larger in magnitude than later samples. When filtering using the Butterworth, I chose to zero the first 15 samples, as they were much larger in amplitude compared to others in the song. Practically, not accounting for the large initial response would result in a song that stayed soft after the first few samples. The Butterworth design only needs 40 add/multiply operations per input sample, making it appealing for cheap computations.

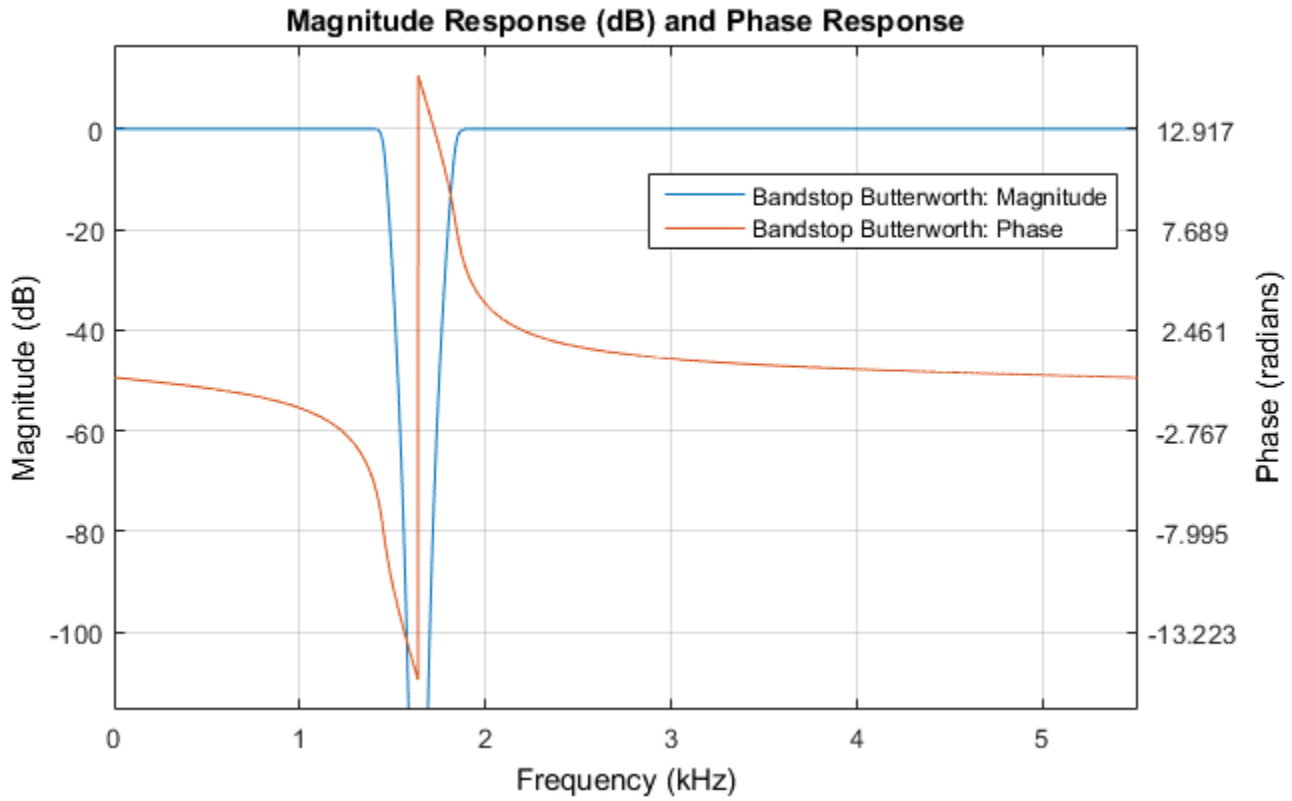

Figure 2: Butterworth Magnitude & Frequency Response

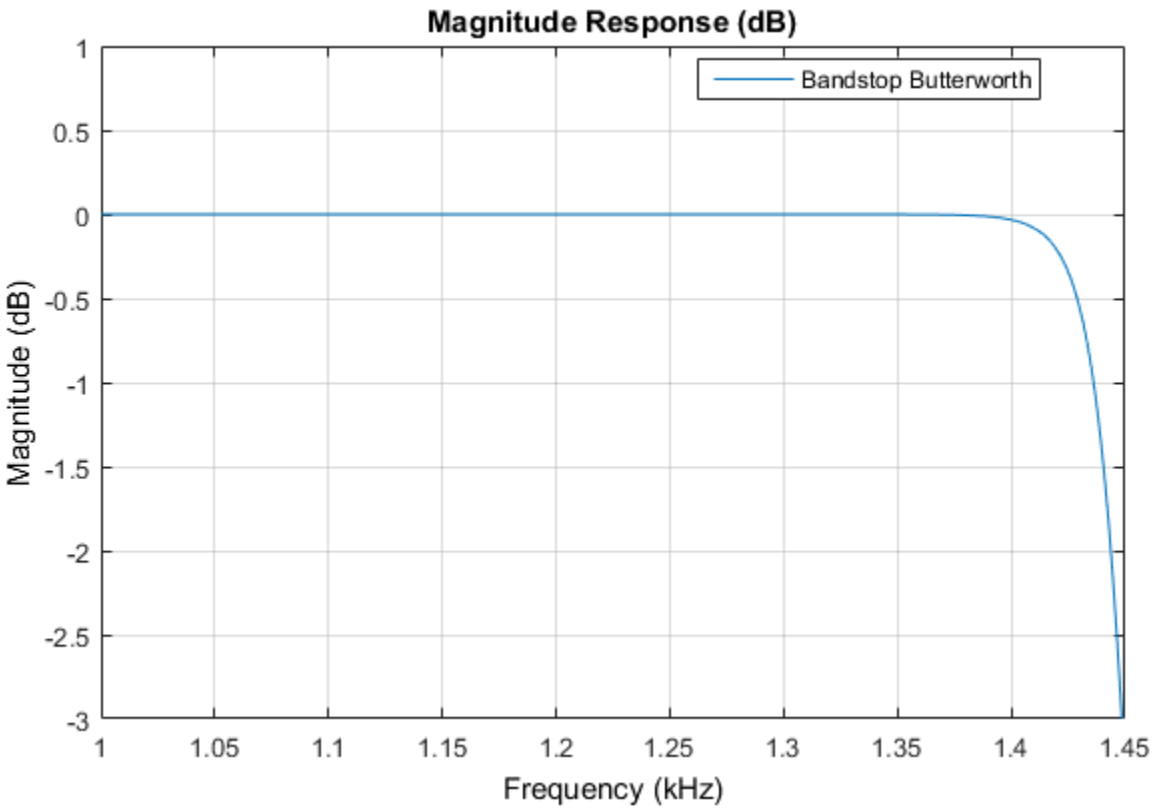

Figure 3: Butterworth Magnitude Response (zoom)

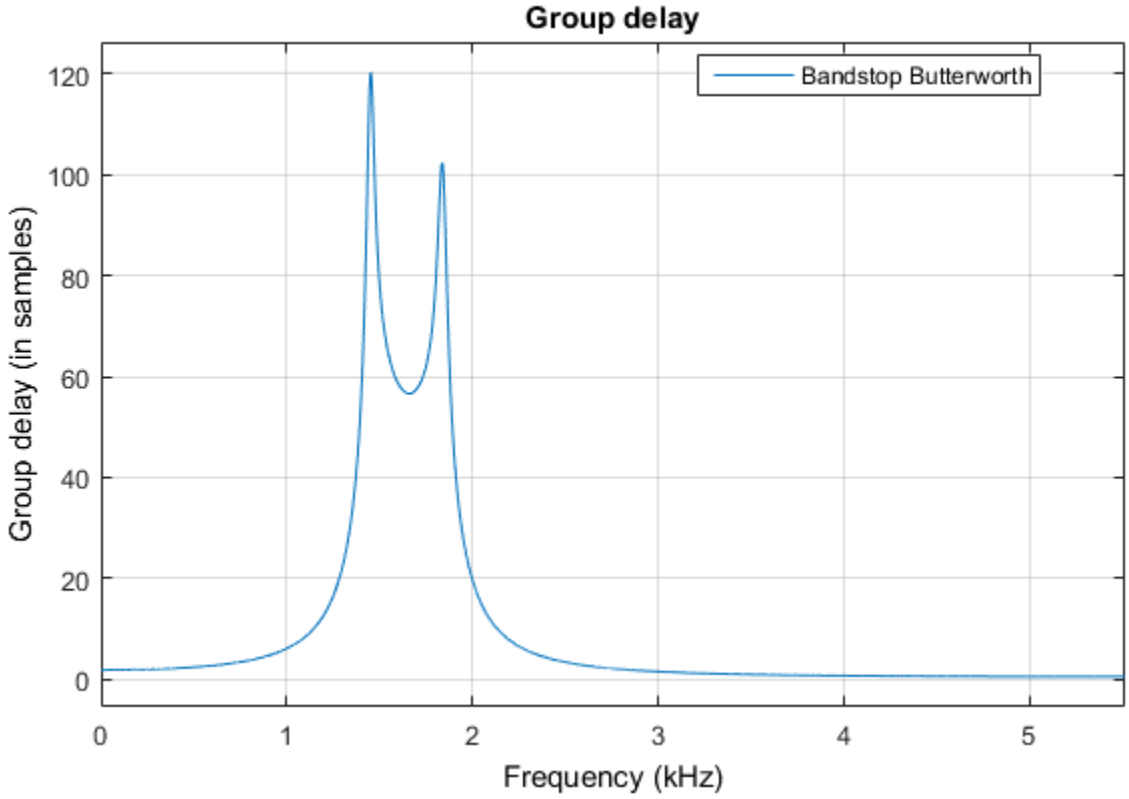

Figure 4: Butterworth Group Delay

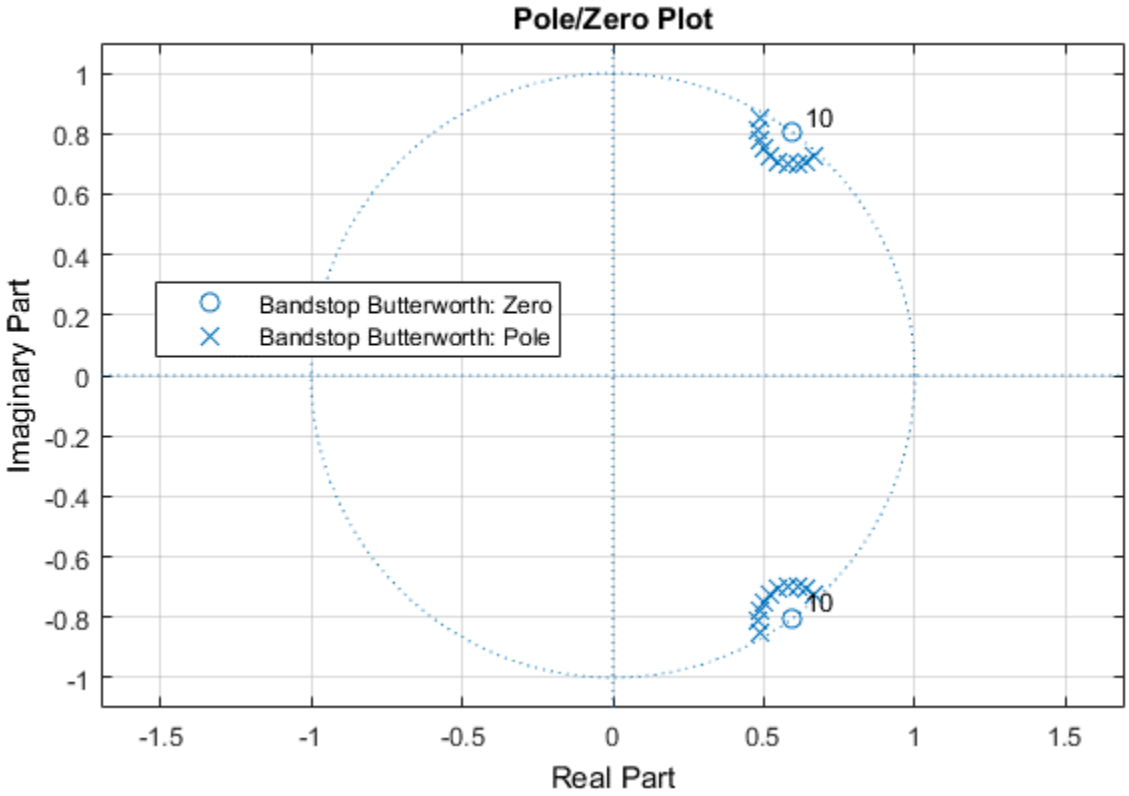

Figure 5: Butterworth Pole/Zero Plot

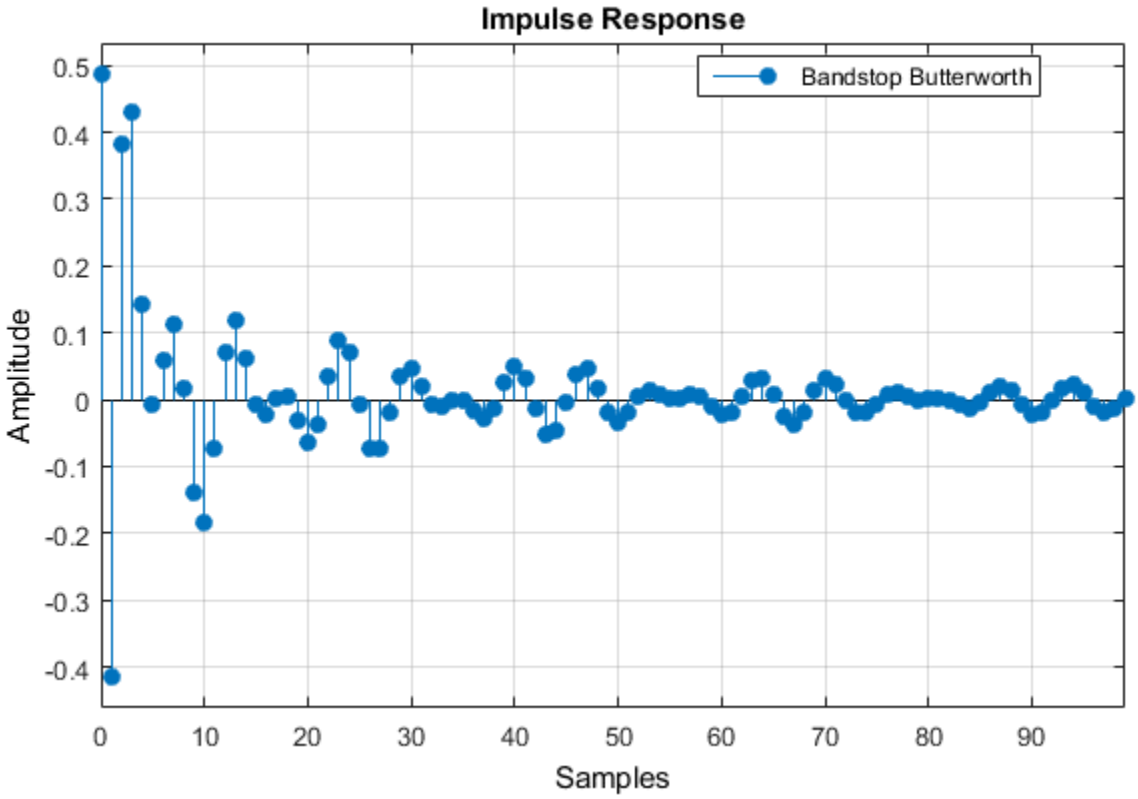

Figure 6: Butterworth Impulse Response

CHEBYSHEV TYPE I: ORDER 14

Named after Pafnuty Chebyshev for his derivation of Chebyshev polynomials, Chebyshev filters are characterized by having a much steeper roll-off than Butterworth filters. Despite having a steeper roll-off than the Butterworth, Chebyshev filters have ripple in the passband (where the Butterworth does not).

The slight ripple in the passband can be more easily seen in Figure 8. It is useful to note the differences between Figure 8 and Figure 3. While the Chebyshev Type I has a slight ripple in the passband, its faster roll-off more than makes up for that. With only an order of 14, the Chebyshev Type I is much more effective than the Butterworth. However, it suffers the same issue with its impulse response. A similar approach is taken here and the first 15 samples are simply thrown away. Another alternative would have been to pad the signal to be filtered with zeros so that no samples would actually be destroyed. To implement this Chebyshev Type I filter would require 28 add/multiply operations.

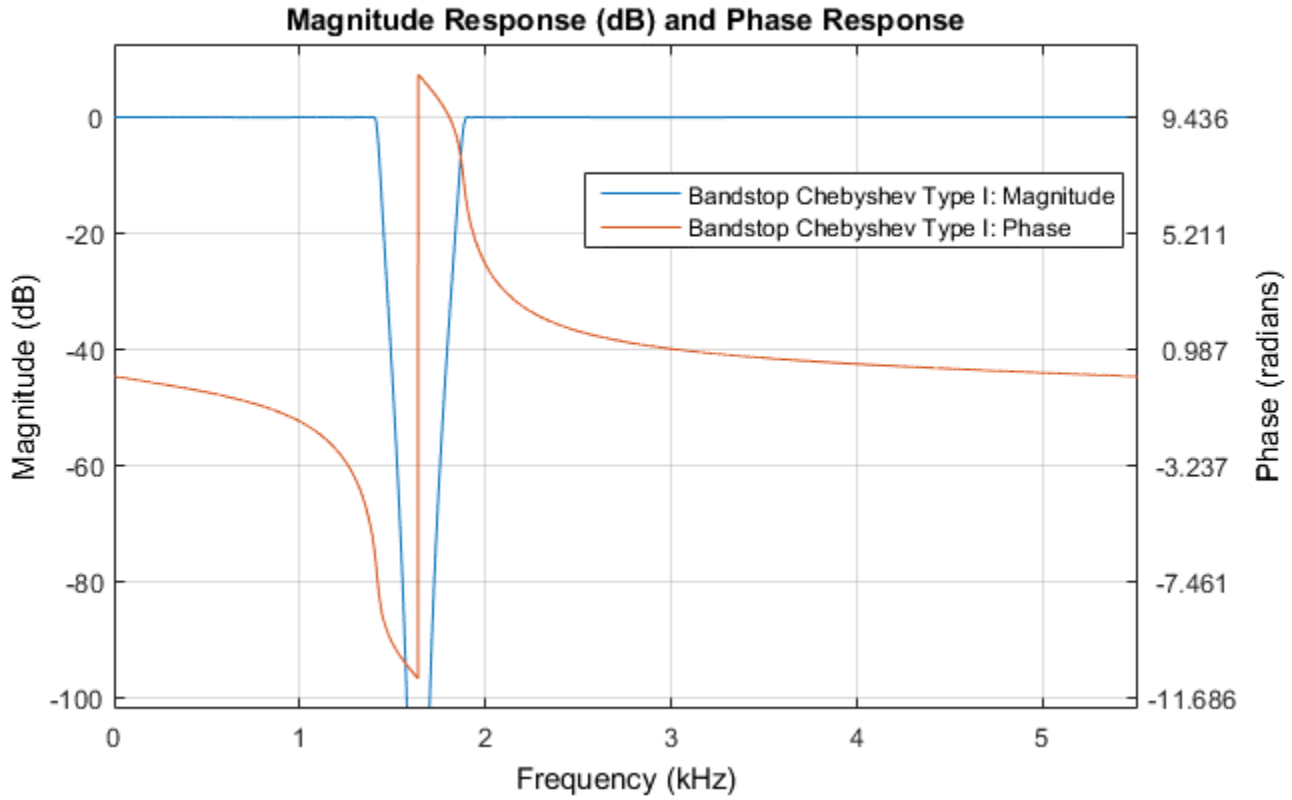

Figure 7: Chebyshev Type I Magnitude & Phase Response

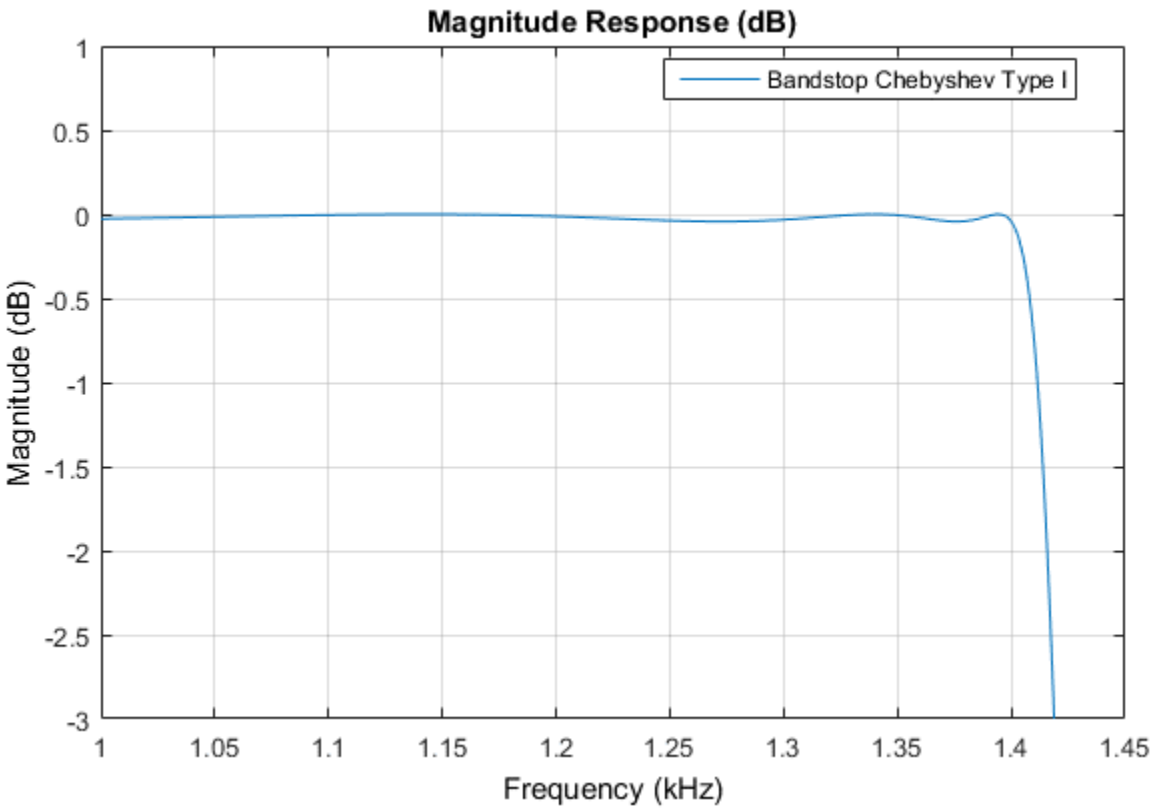

Figure 8: Chebyshev Type I Magnitude Response (zoom)

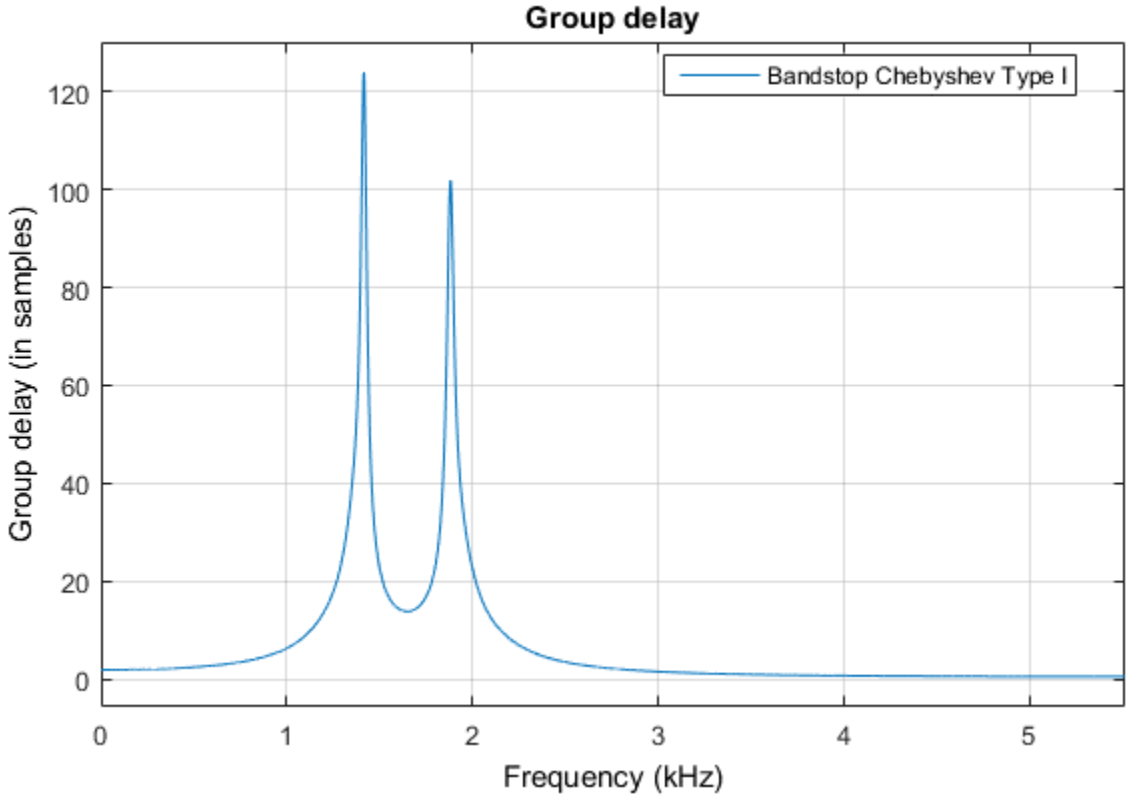

Figure 9: Chebyshev Type I Group Delay

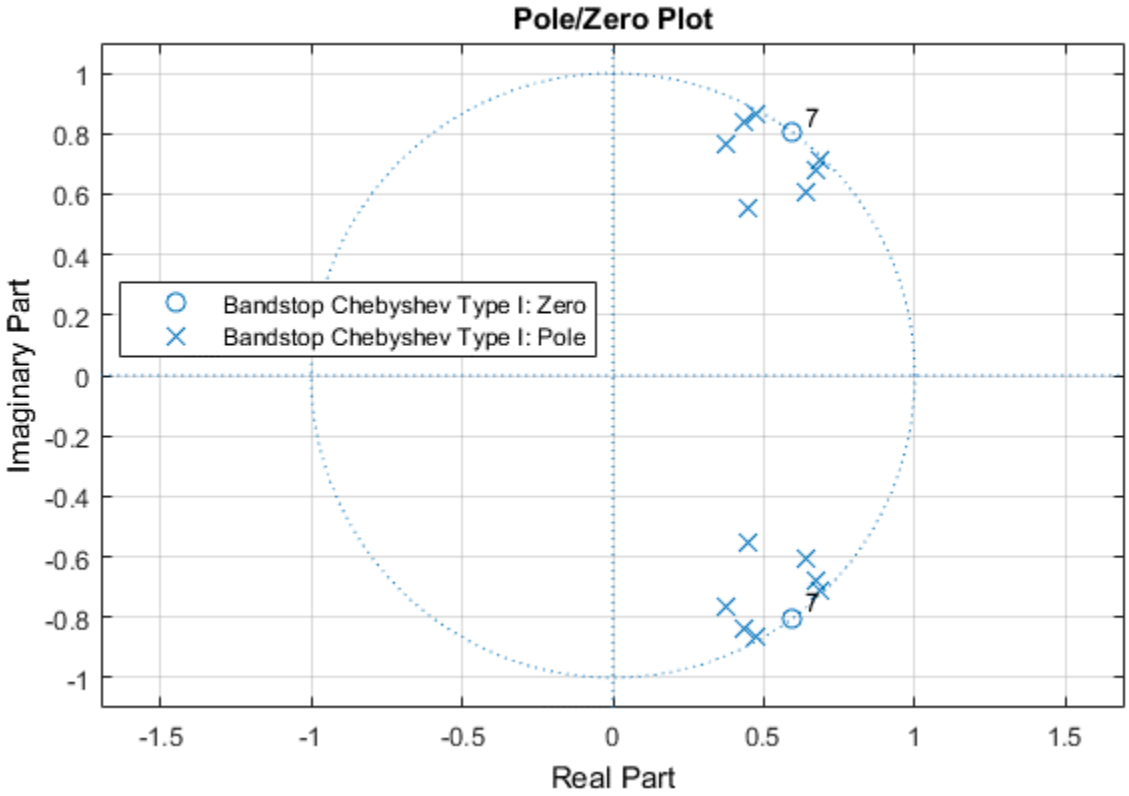

Figure 10: Chebyshev Type I Pole/Zero Plot

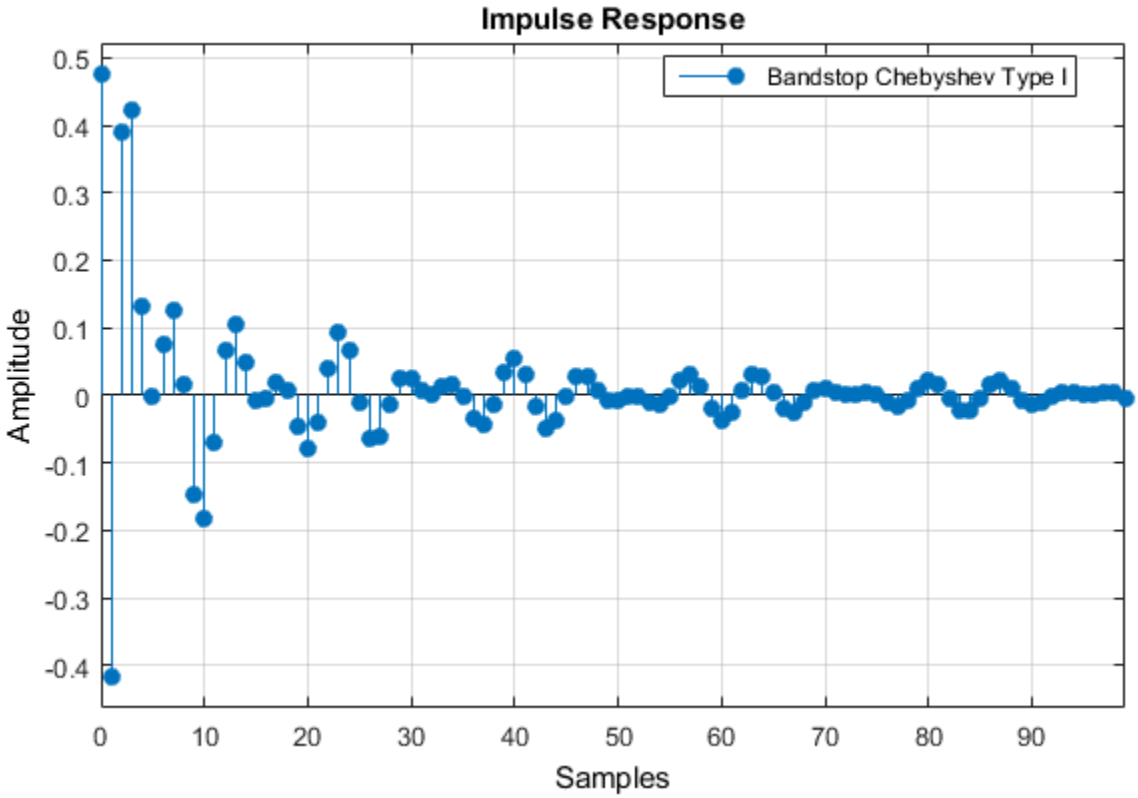

Figure 11: Chebyshev Type I Impulse Response

CHEBYSHEV TYPE II: ORDER 14

The second filter in the Chebyshev family is the Chebyshev Type II filter. Similar to the Type I, the Type II shares a steep roll-off but has more stopband ripple than the Butterworth filter. The phase response of the Type II (Figure 12) is slightly different from Type I. While the overall shape of the phase in the stopband is the same, the Type II filter has a saw-tooth effect in the phase transition. This causes a similar spike in the group delay (Figure 14) as was seen in the Chebyshev Type I (Figure 9). Another point of note is that the Chebyshev Type II filter has its zeros dispersed around one location (Figure 15) instead of directly overlapping as was the case with Type I (Figure 10). On a practical note, similar to the Type I Chebyshev filter, the first 28 samples of the Type II filter are thrown out to properly normalize the signal. The impulse response settles down enough after the 28th sample though (Figure 16). Just as the Type I did, the Type II filter only requires 28 add/multiply operations per input sample. This would supposedly make the Type II seem as viable an option as the type I, but the filtered signal from the Type II filter sounds much worse compared to the Type I.

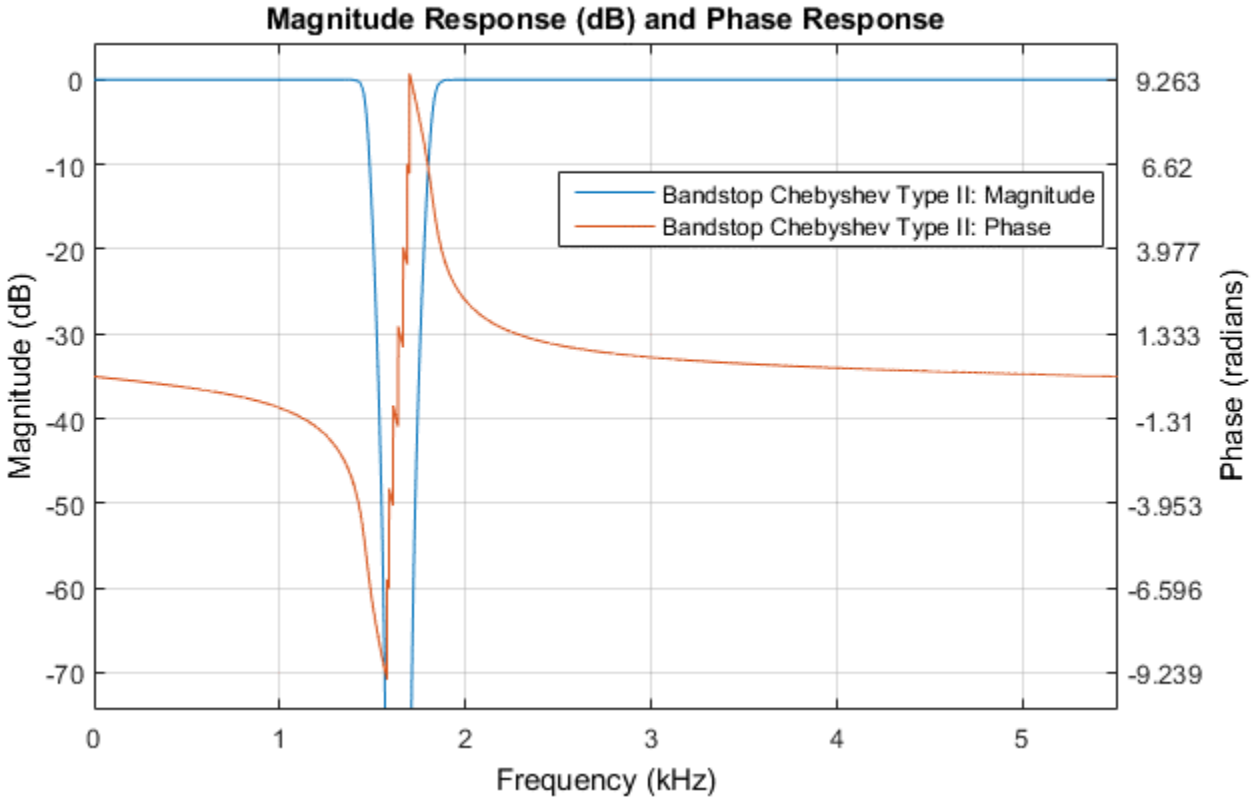

Figure 12: Chebyshev Type II Magnitude & Phase Response

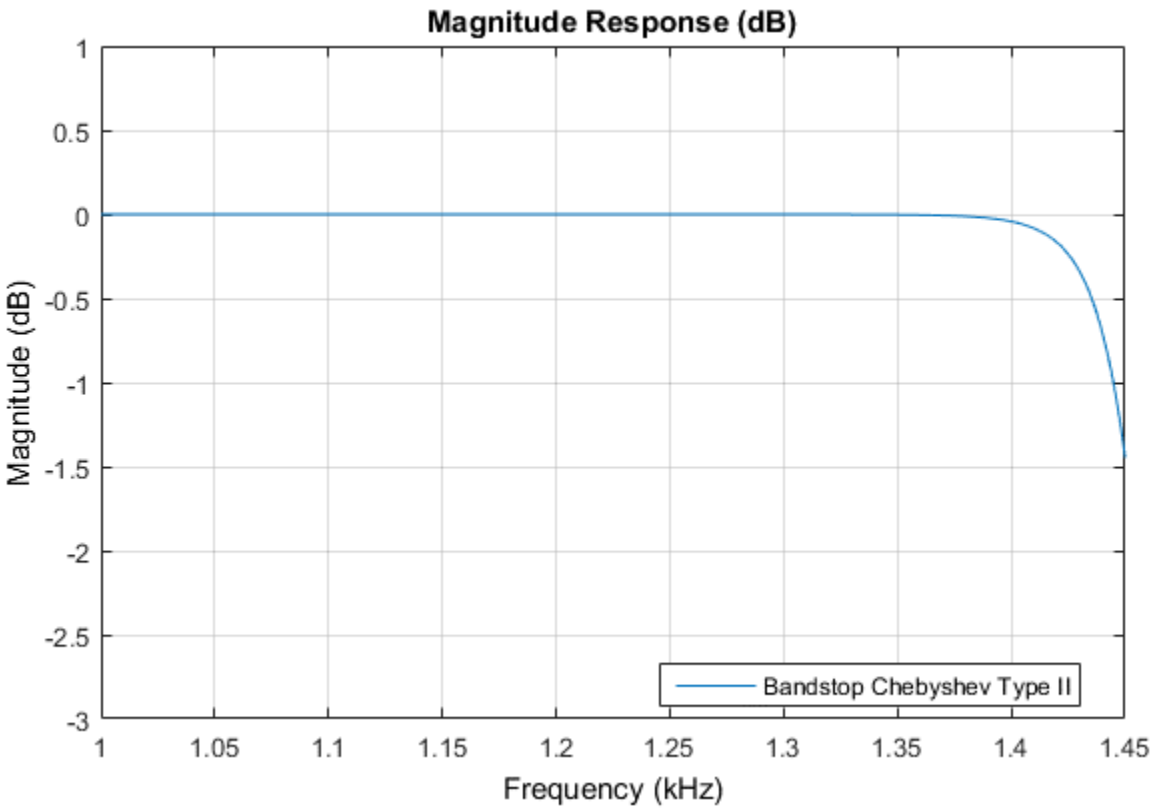

Figure 13: Chebyshev Type II Magnitude Response (zoom)

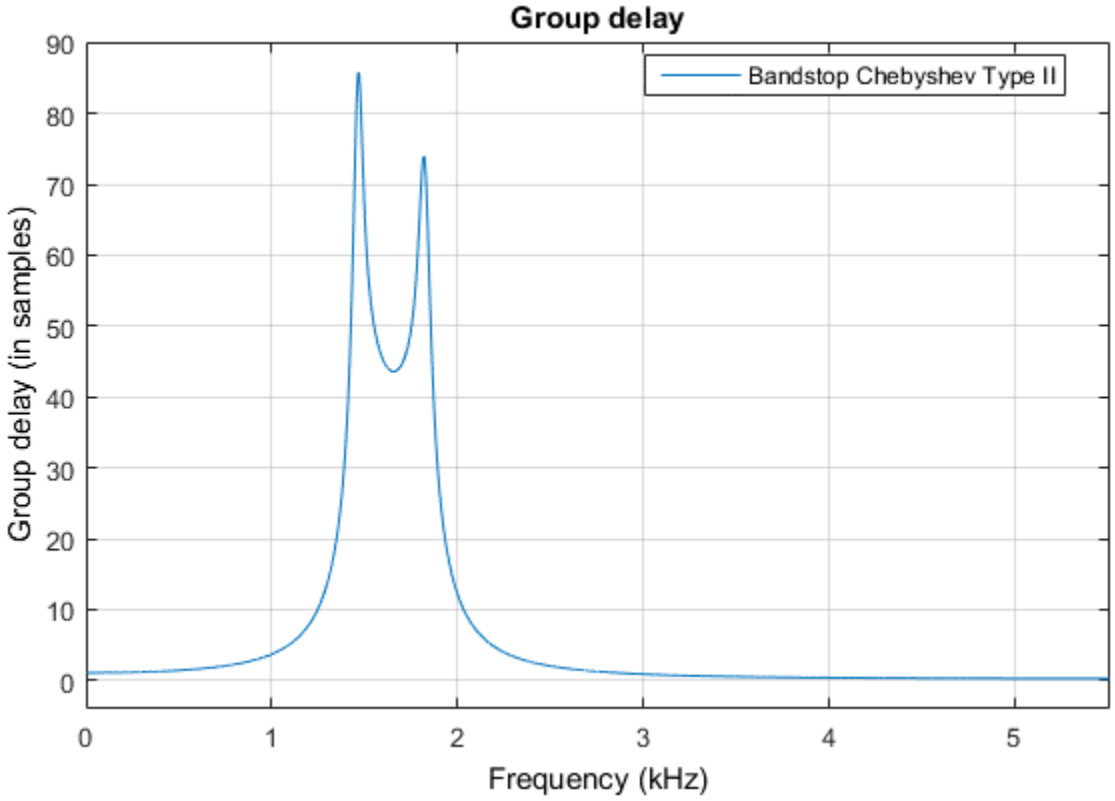

Figure 14: Chebyshev Type II Group Delay

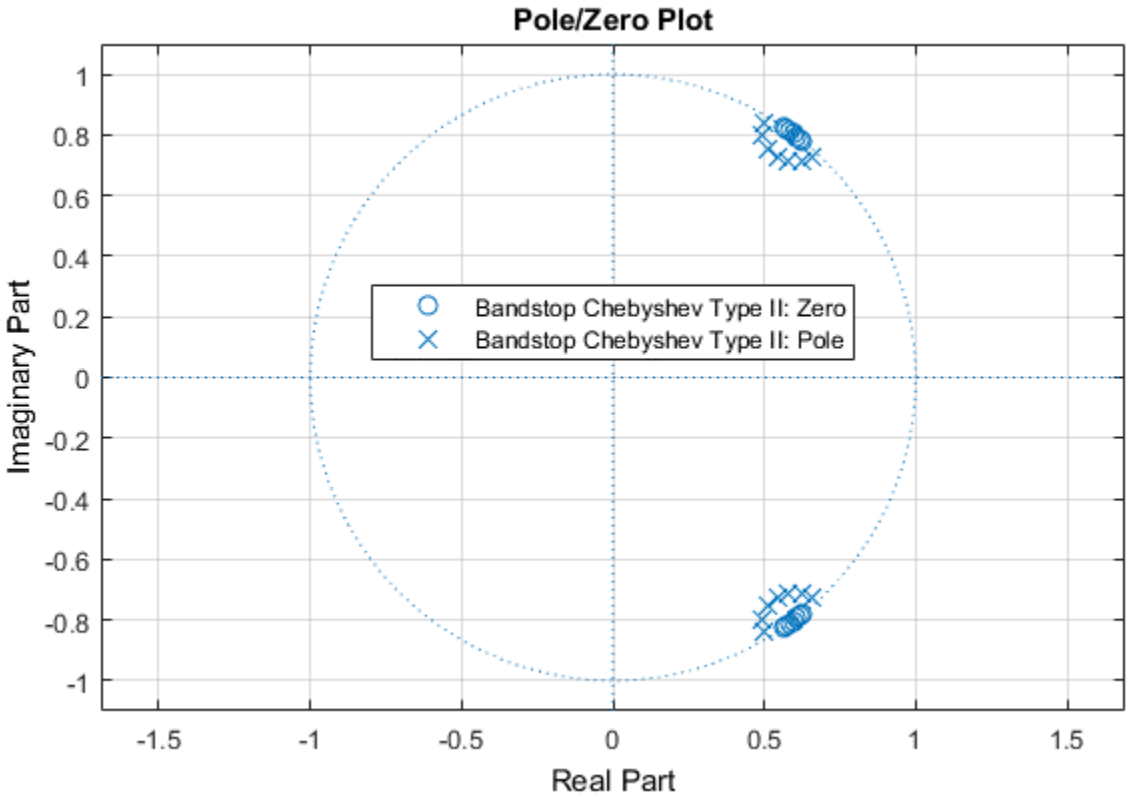

Figure 15: Chebyshev Type II Pole/Zero Plot

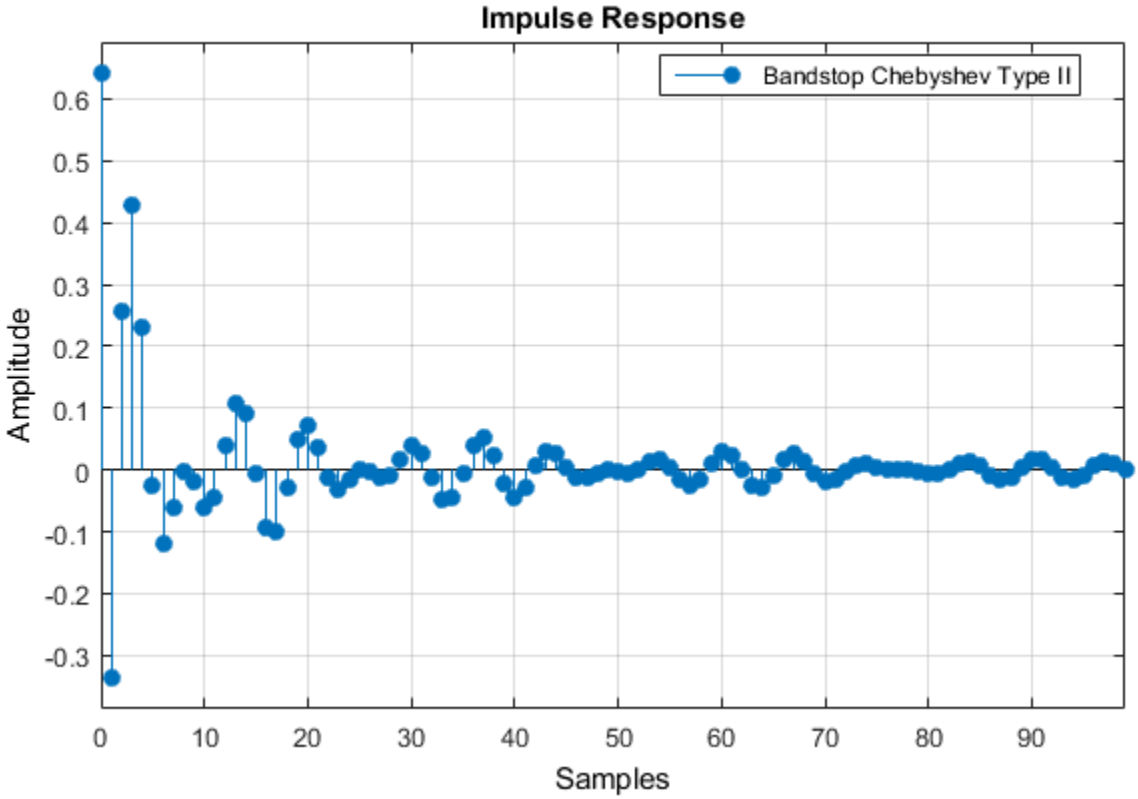

Figure 16: Chebyshev Type II Impulse Response

ELIPTIC: ORDER 12

The elliptic (sometimes known as a Cauer filter, named after Wilhelm Cauer, or as a Zolotarev filter, named after Yegor Zolotarev) is best known for its equiripple behavior and its incredibly fast transition between passband and stopband. As the ripple in the stopband approaches zero, the filter becomes a Type I Chebyshev filter. As the ripple in the passband approaches zero, the filter becomes a Type II Chebyshev filter. And as the ripple in the stopband and the passband approach zero, the filter becomes a Butterworth filter. Figure 18 shows the elliptic filter having a similar passband ripple as of the Chebyshev Type I (Figure 8) and a similar phase as the Chebyshev Type II (Figure 12).

The group delay (Figure 19) resembles the Butterworth and Type I/II Chebyshev filters. The pole/zero plot (Figure 20) of the elliptic filter is also very similar to the pole/zero plot of the Chebyshev Type I (Figure 10) filter. And the impulse response (Figure 21) of the elliptic filter is very similar to the previous three filters. Only requiring 24 add/multiply operations, it may seem surprising that the elliptic filter produces the best sounding output compared to the other five filters.

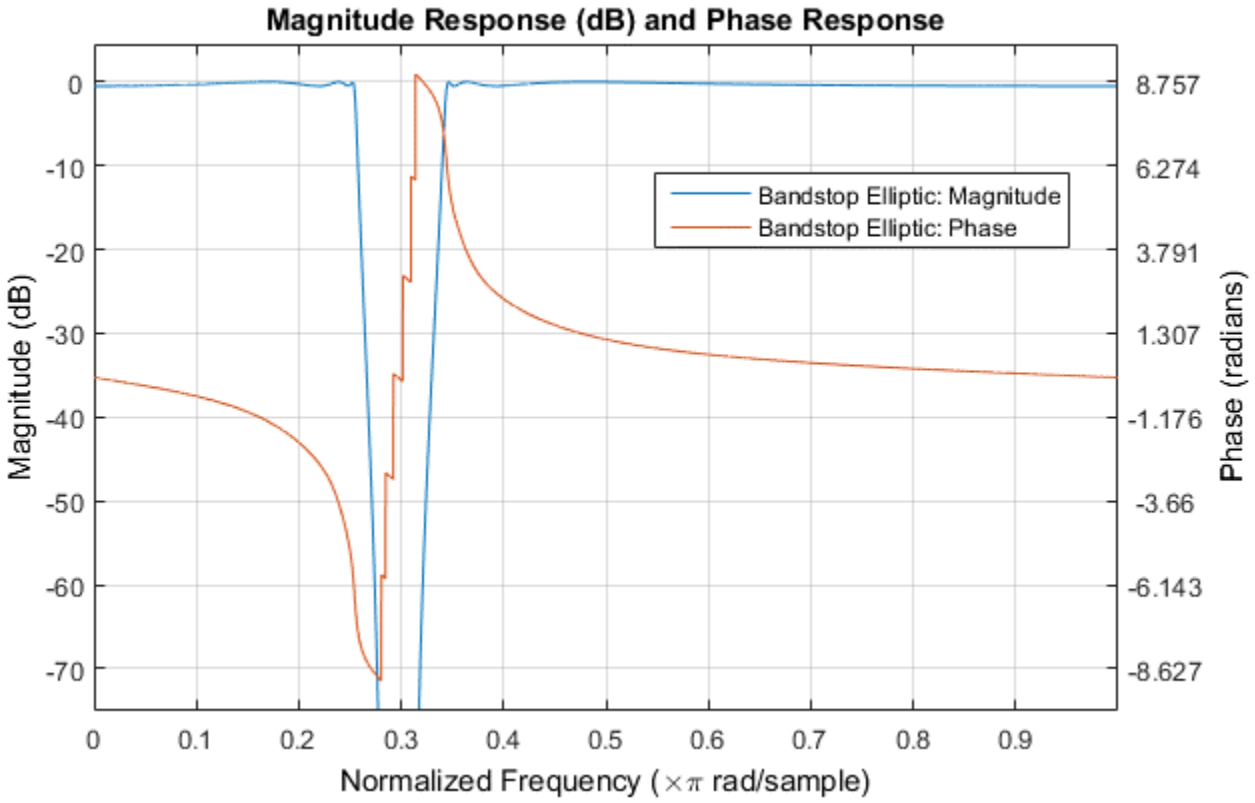

Figure 17: Elliptic Magnitude & Phase Response

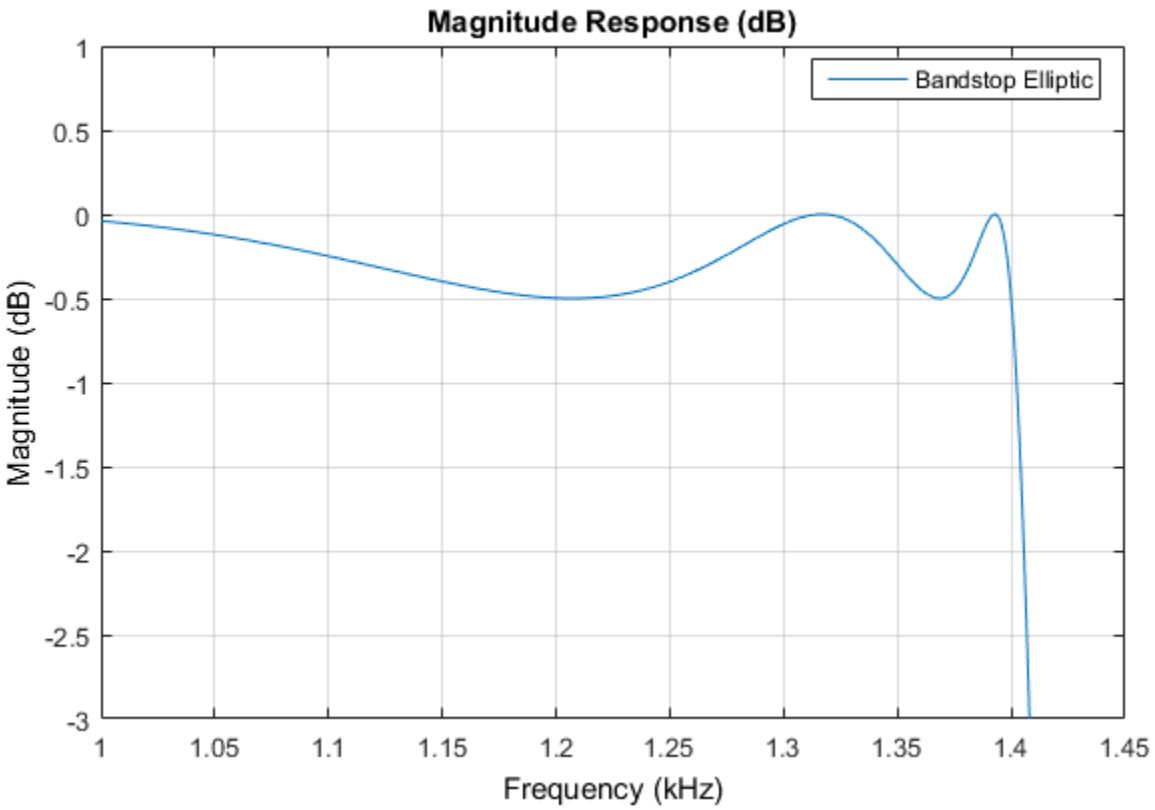

Figure 18: Elliptic Magnitude Response (zoom)

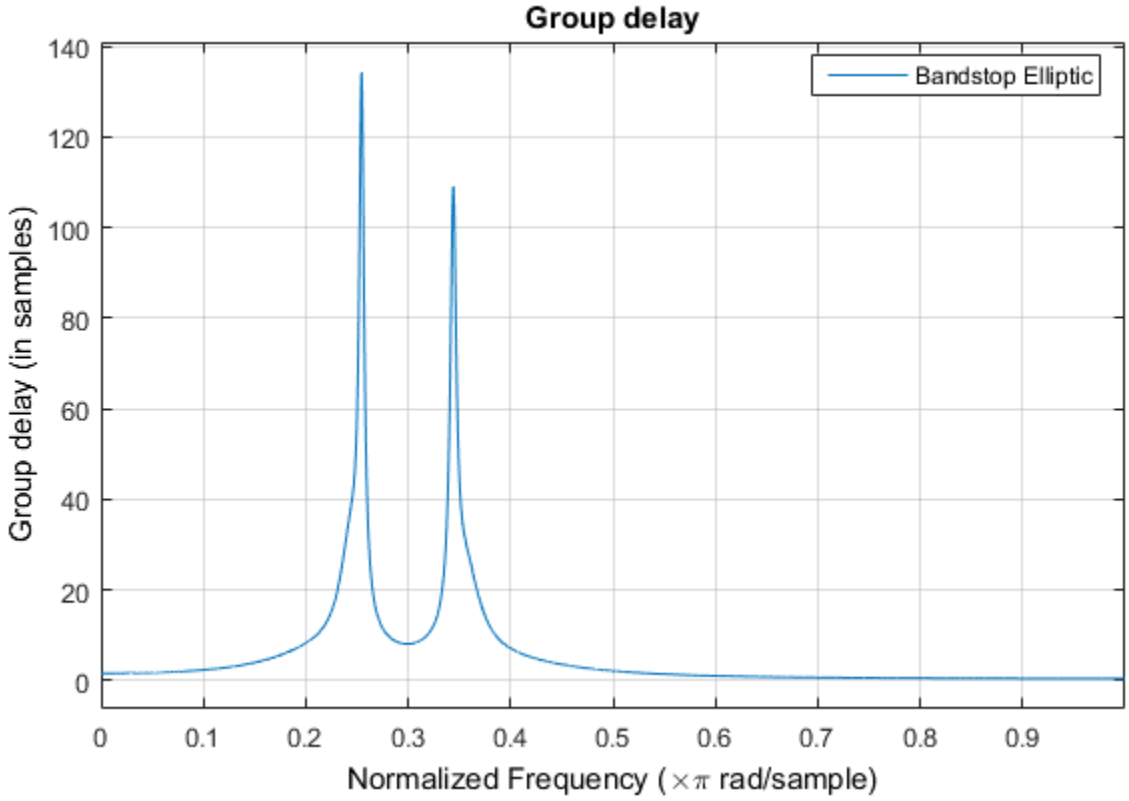

Figure 19: Elliptic Group Delay

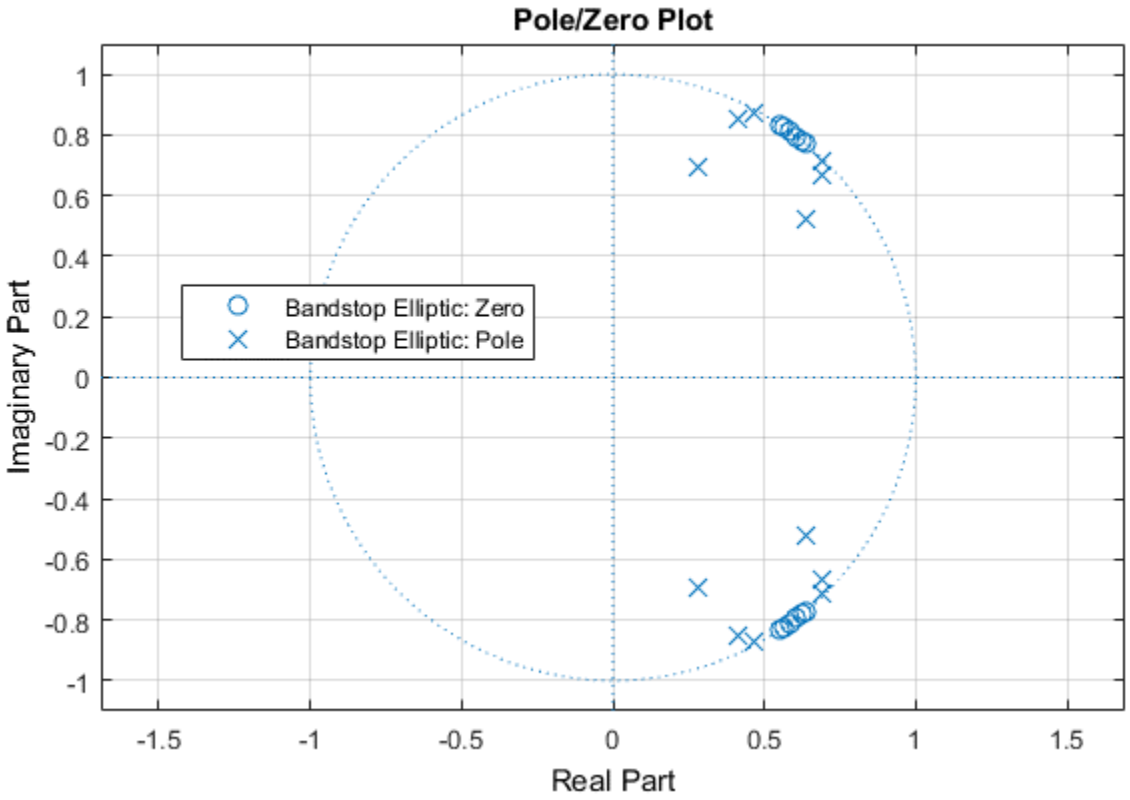

Figure 20: Elliptic Pole/Zero Plot

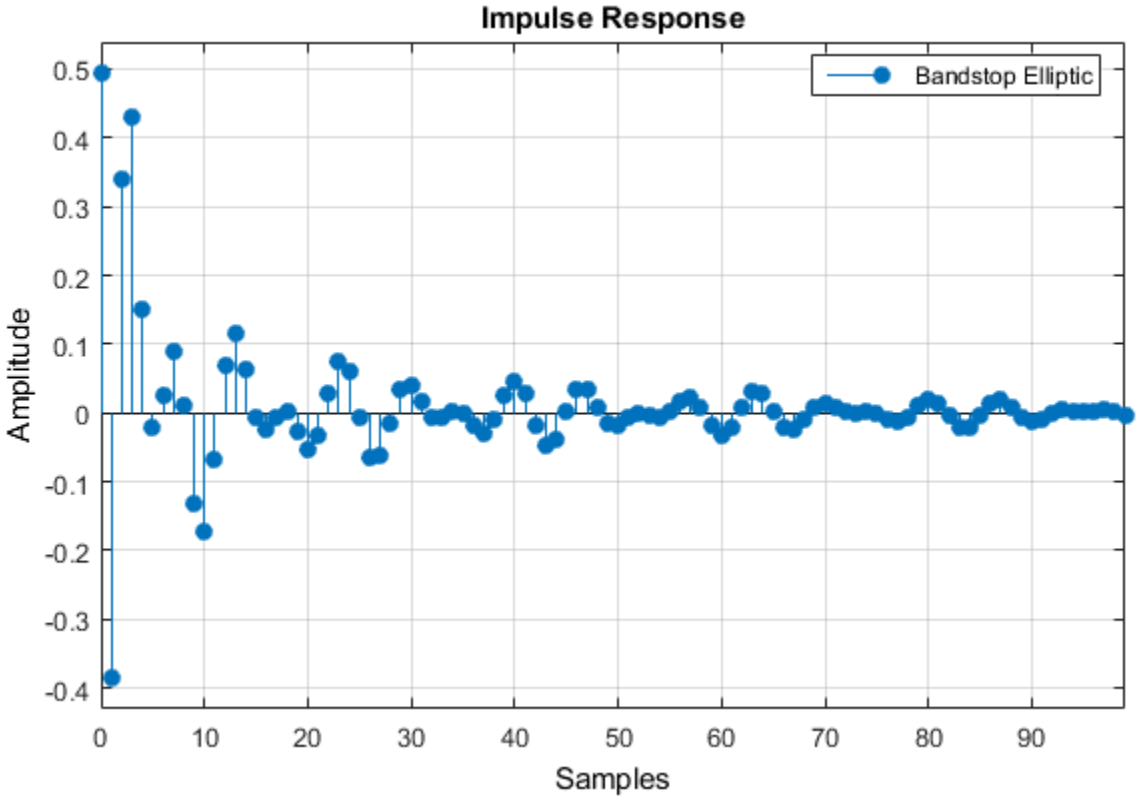

Figure 21: Elliptic Impulse Response

KAISER WINDOW: ORDER 354

Developed by James Kaiser at Bell Laboratories, the Kaiser Window is our first window function in this assignment. The order of this filter is much higher than any of the previous filters seen so far. The advantage that this filter design has is that it has linear phase, which can be clearly seen in Figure 24. The group delay is half the order of our filter.

The magnitude response (Figure 22) is very similar to the Chebyshev Type II filter (Figure 12) with a similar slow roll-off in the stopband. What is not similar to any of the filters discussed so far is the pole/zero plot (Figure 25). Since this is a finite impulse response (FIR) and linear phase filter, all of the poles are at the origin and every zero has an inverse on the other side of the unit circle. A positive aspect was its mostly uniform impulse response (Figure 26). Except for sample 177, the impulse response was easy to work with. Unfortunately this is a very costly filter, requiring 355 add/multiply operations. Despite this being a very costly filter and having linear phase, the Kaiser windows produced one of the worst outputs out of the six filters.

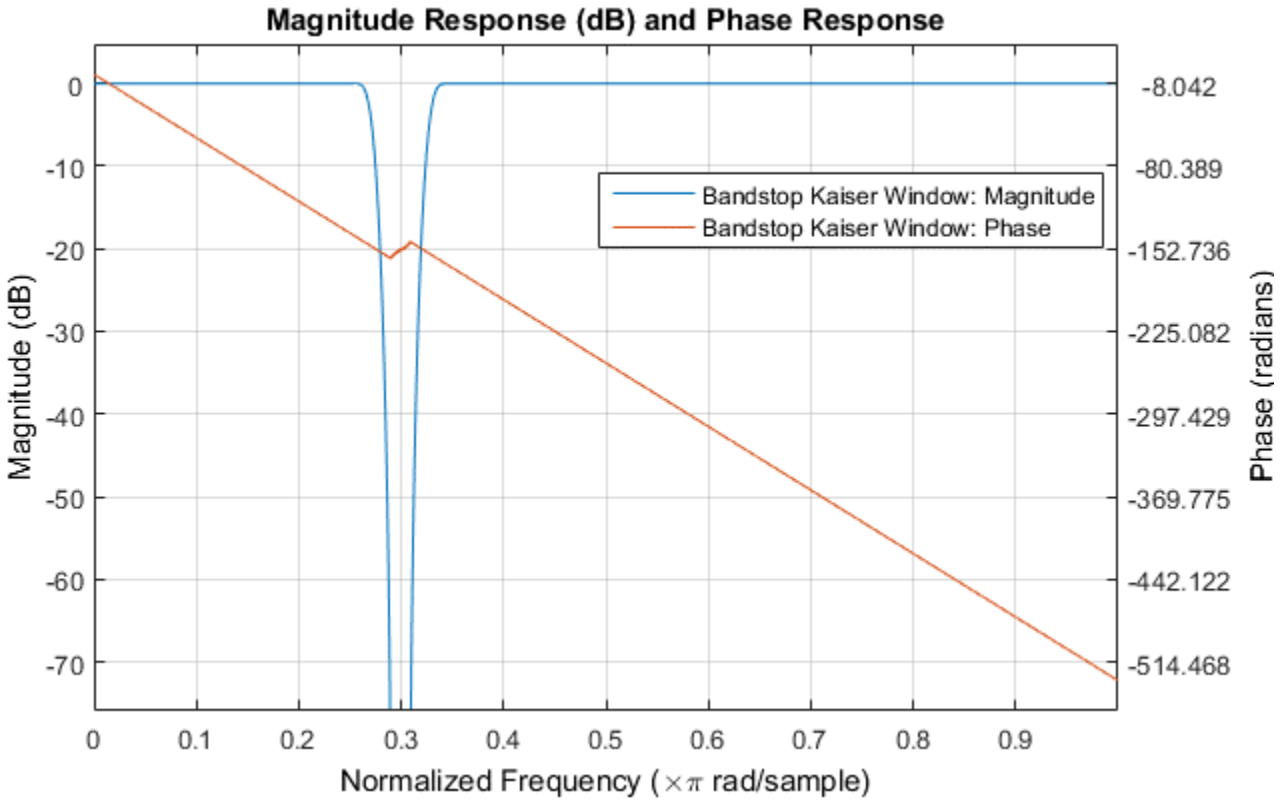

Figure 22: Kaiser Window Magnitude & Phase Response

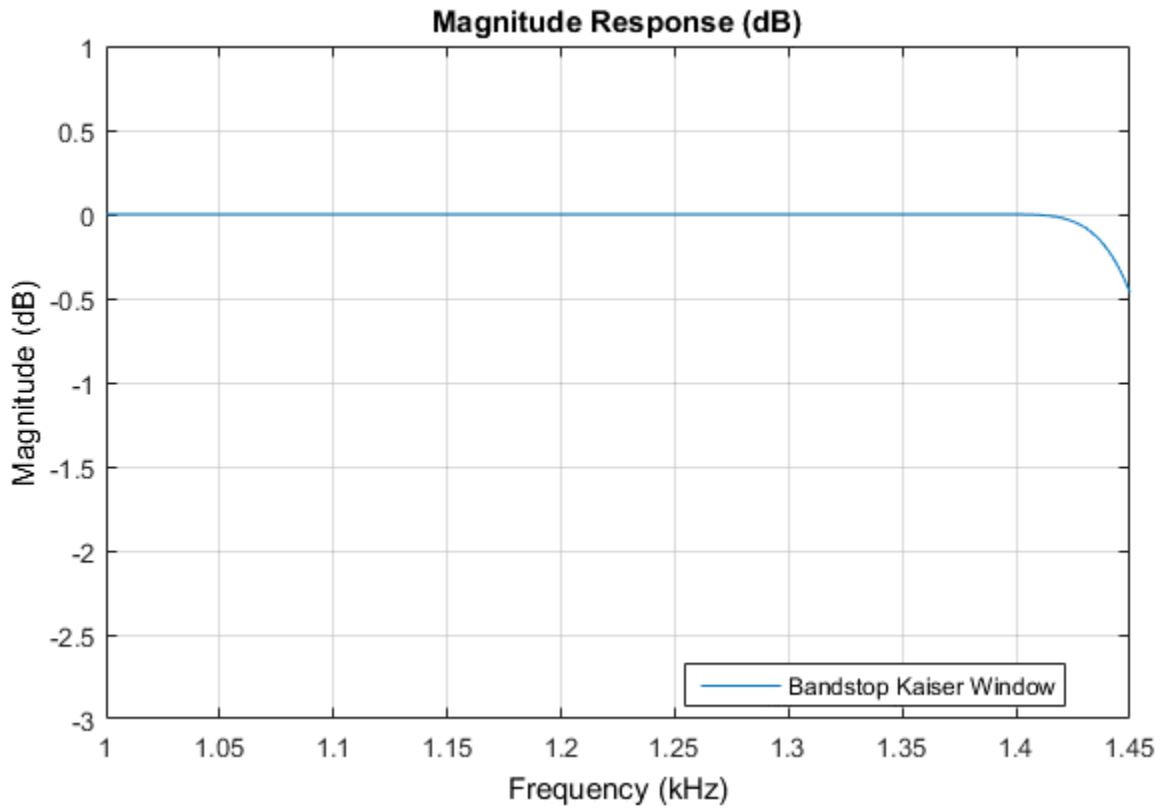

Figure 23: Kaiser Window Magnitude Response (zoom)

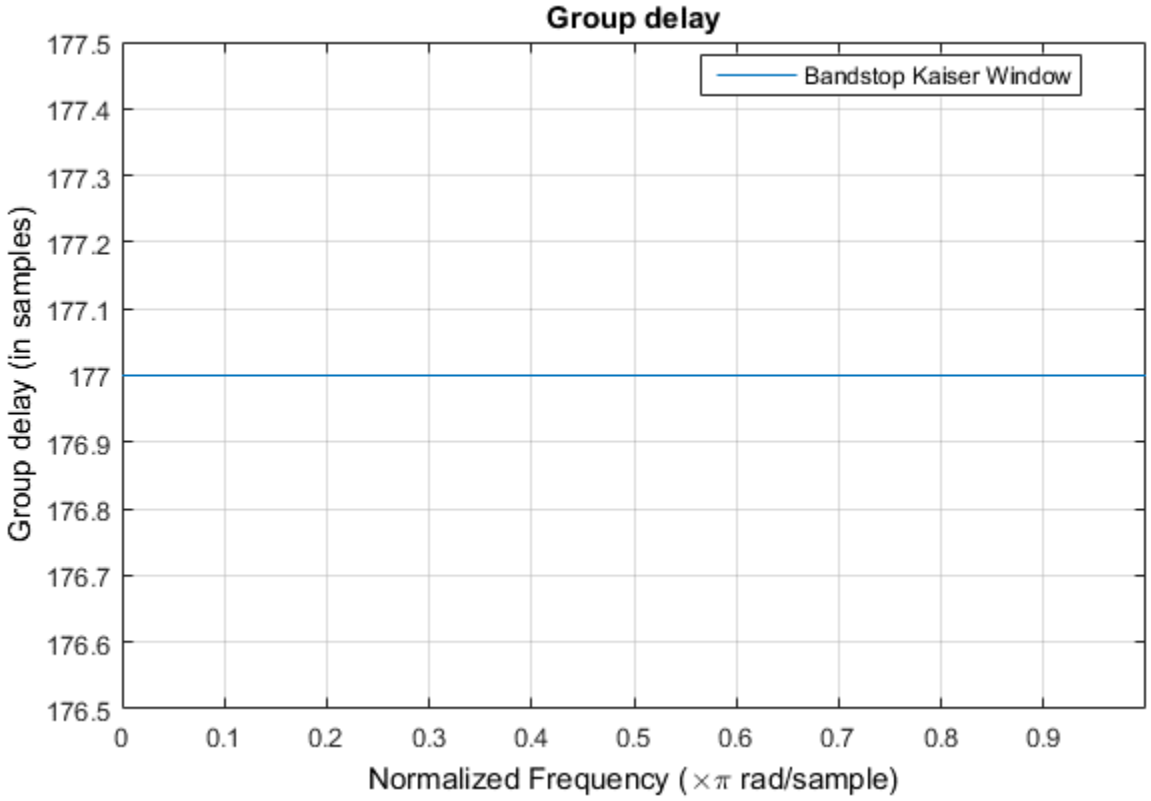

Figure 24: Kaiser Window Group Delay

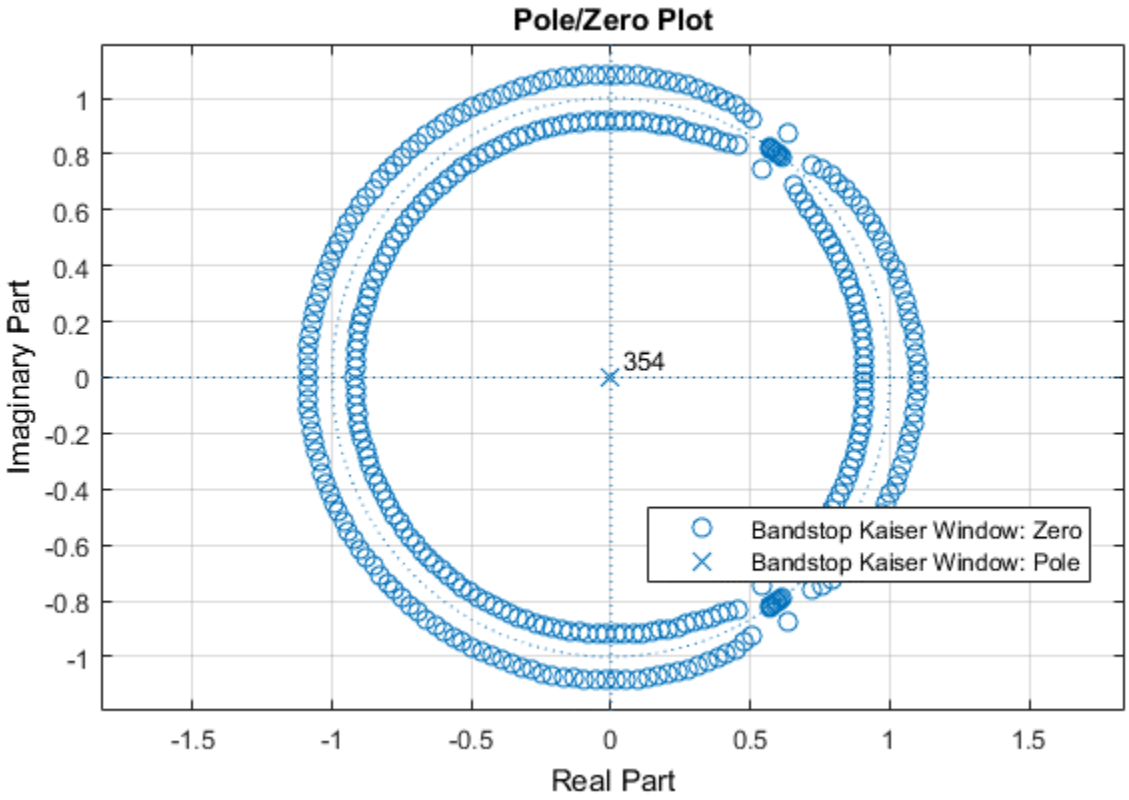

Figure 25: Kaiser Pole/Zero Plot

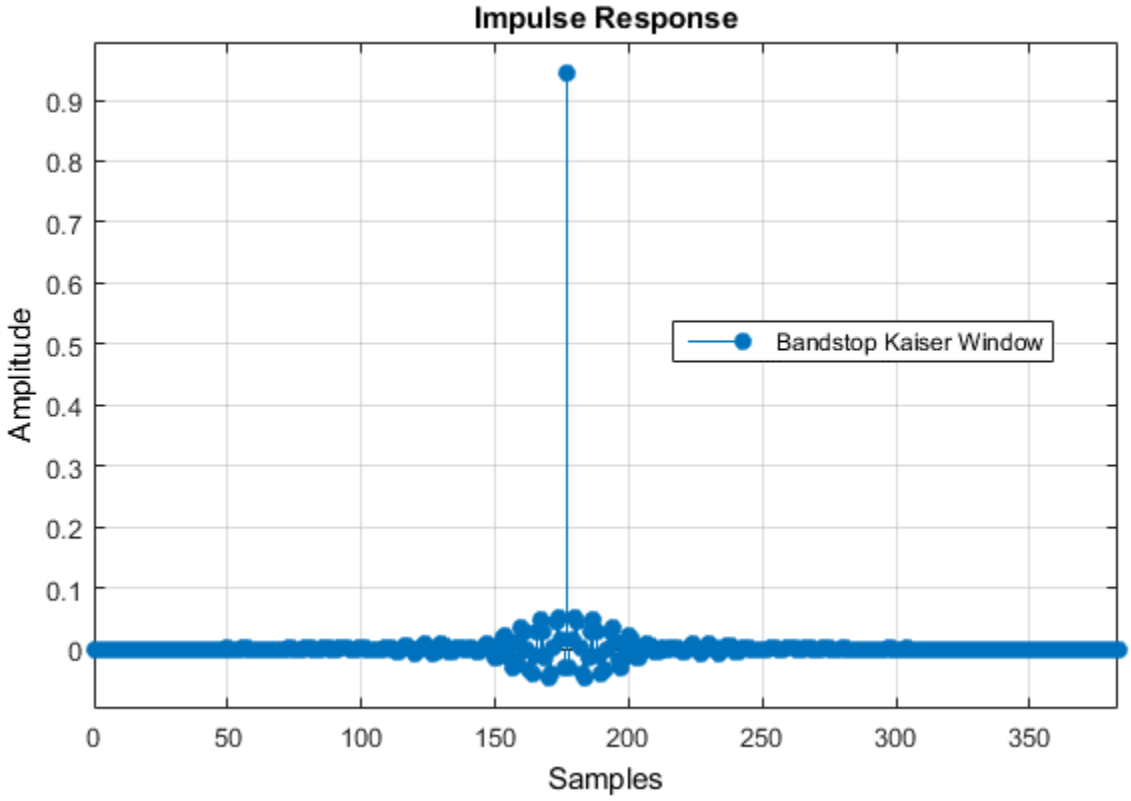

Figure 26: Kaiser Window Impulse response

PARKS-MCCLELLAN: ORDER 196

Published by James McClellan and Thomas Parks in 1972, the Parks-McClellan algorithm is an iterative algorithm for finding the optimal Chebyshev finite impulse response. The Parks-McClellan algorithm is specifically a variation of the Remez exchange algorithm, with the change that it is specifically designed for FIR filters. It is an industry standard for FIR filter design.

The Parks-McClellan algorithm has basically the same magnitude response (Figure 27) as the Kaiser window (Figure 22). The ripple in the passband for the Parks-McClellan filter (Figure 28) is greater than the same section on the Kaiser window (Figure 23). Just as was the case with the Kaiser window (Figure 24), the phase is linear for the Parks-McClellan filter (Figure 29). The pole/zero plot (Figure 30) for the Parks-McClellan filter is very similar to the pole/zero plot (Figure 25) for the Kaiser window, except for the zero at 0.04 and its inverse at 22. The Parks-McClellan filter also has much less zeros. Just like the Kaiser window (Figure 26), the Parks-McClellan filter has an unusually high impulse response (Figure 31) at sample 98, half the filter's order. Despite being a somewhat better filter than the Kaiser window, and having linear phase, the cost of 197 add/multiply operations does not justify the poor output quality that it produced.

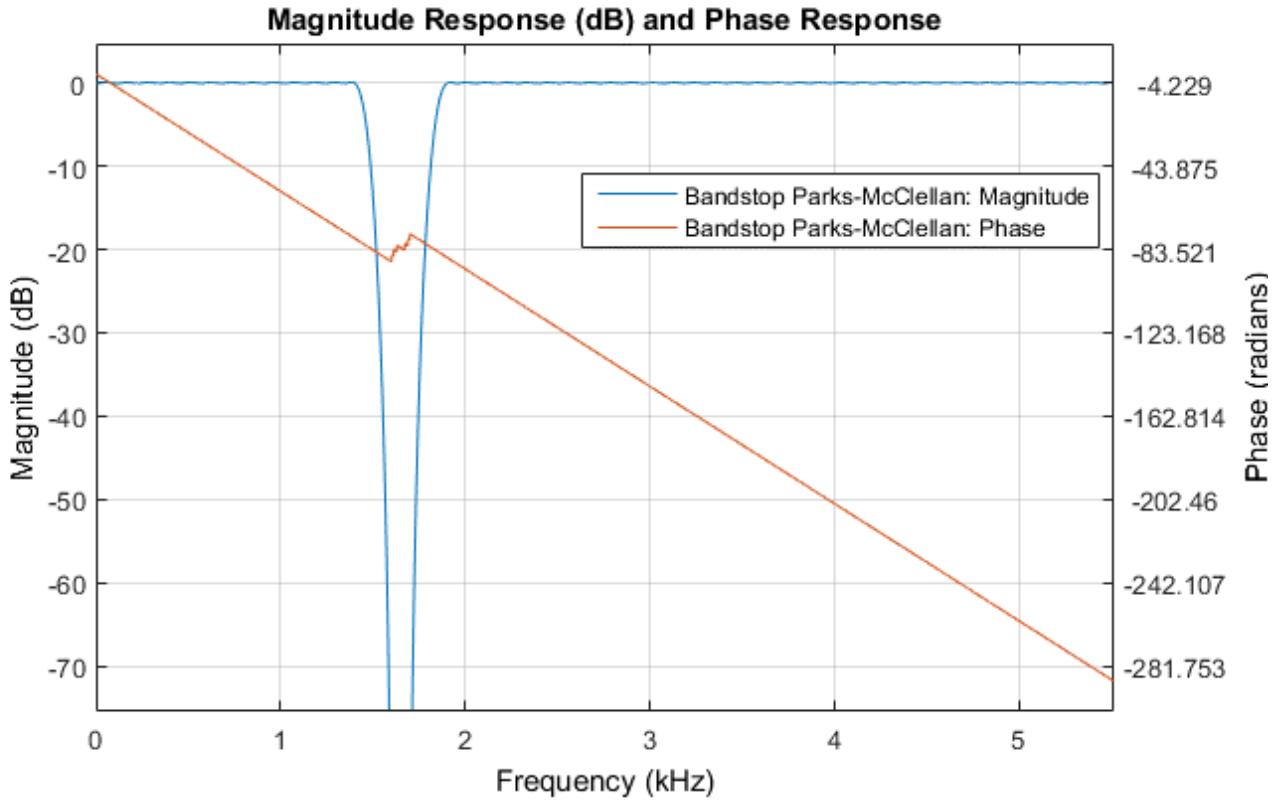

Figure 27: Parks-McClellan Magnitude & Phase Response

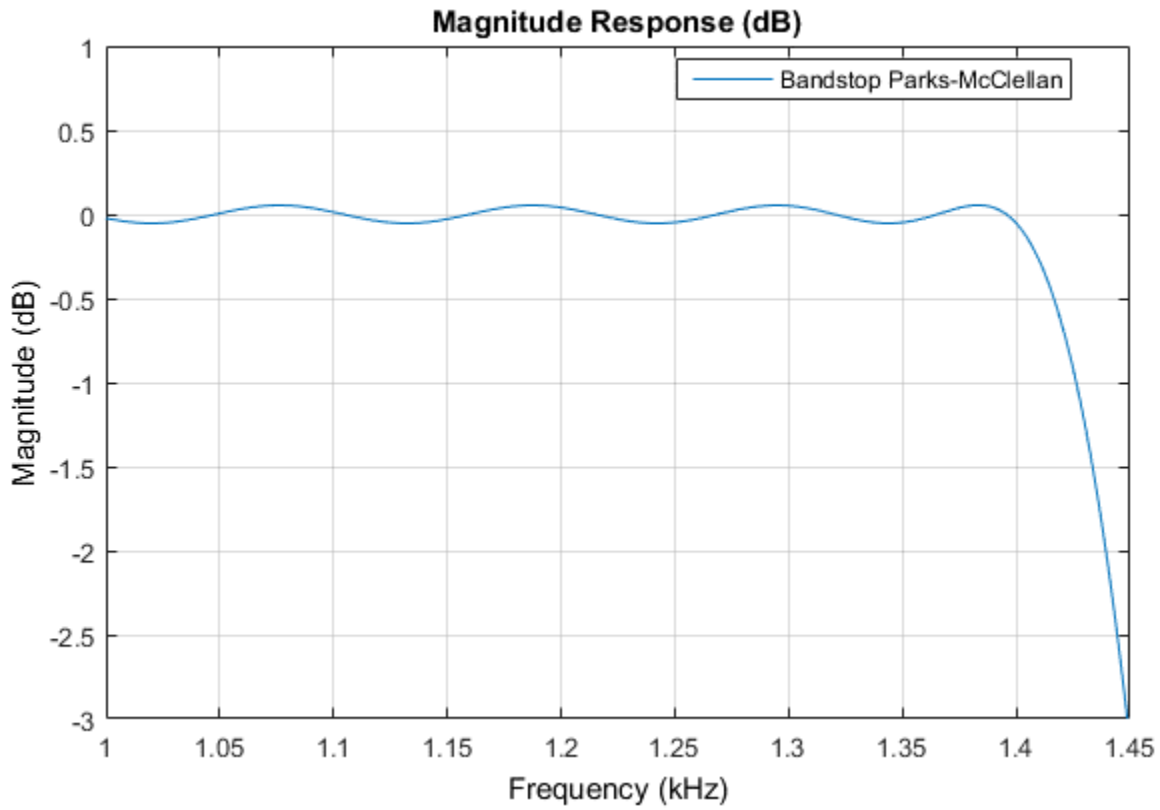

Figure 28: Parks-McClellan Magnitude Response (zoom)

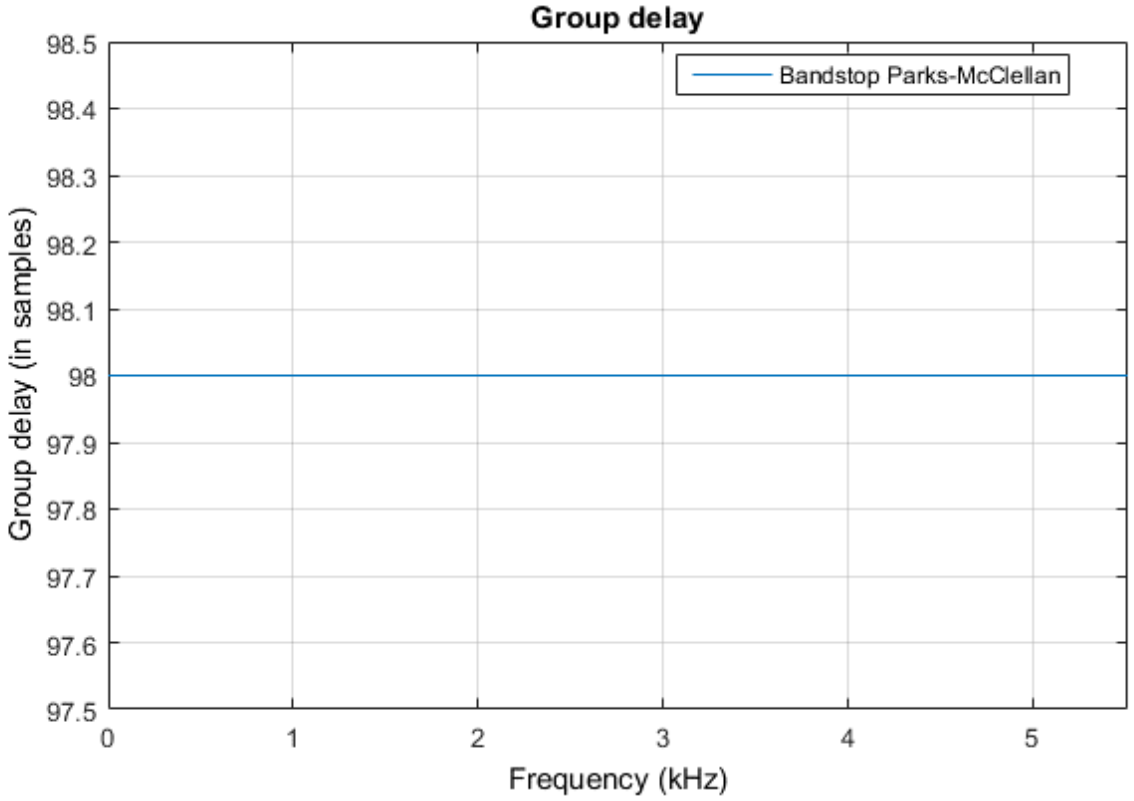

Figure 29: Parks-McClellan Group Delay

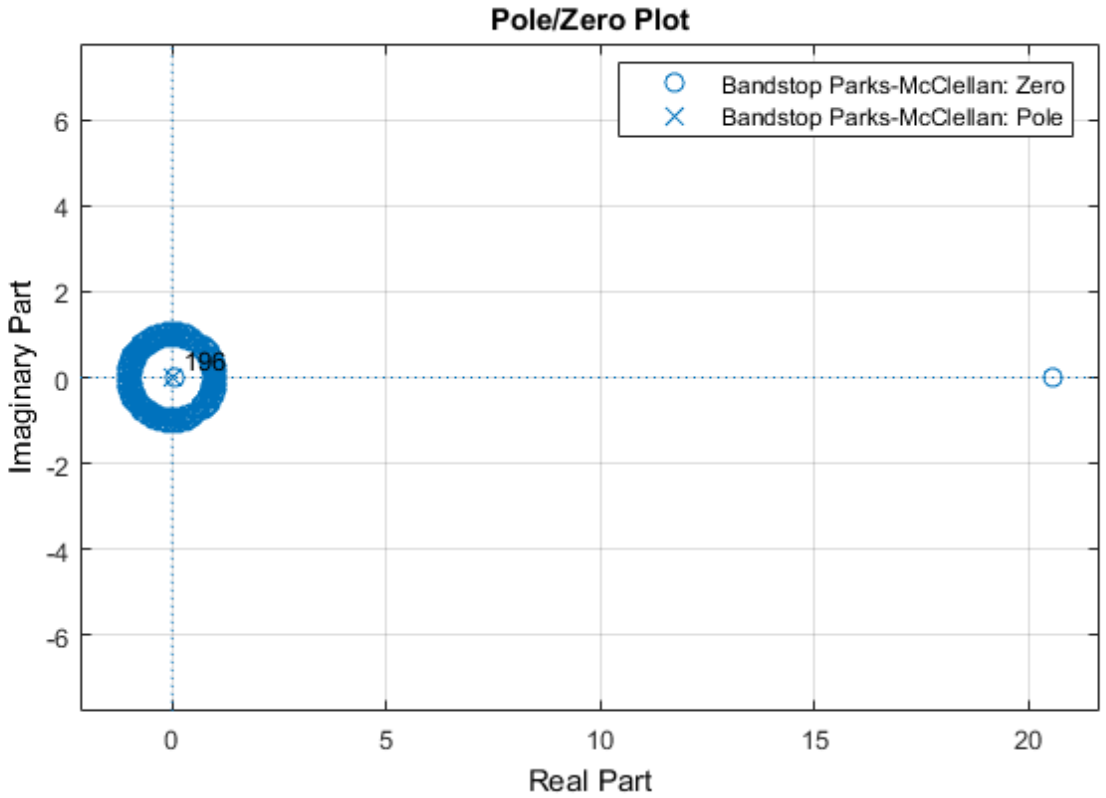

Figure 30: Parks-McClellan Pole/Zero Plot

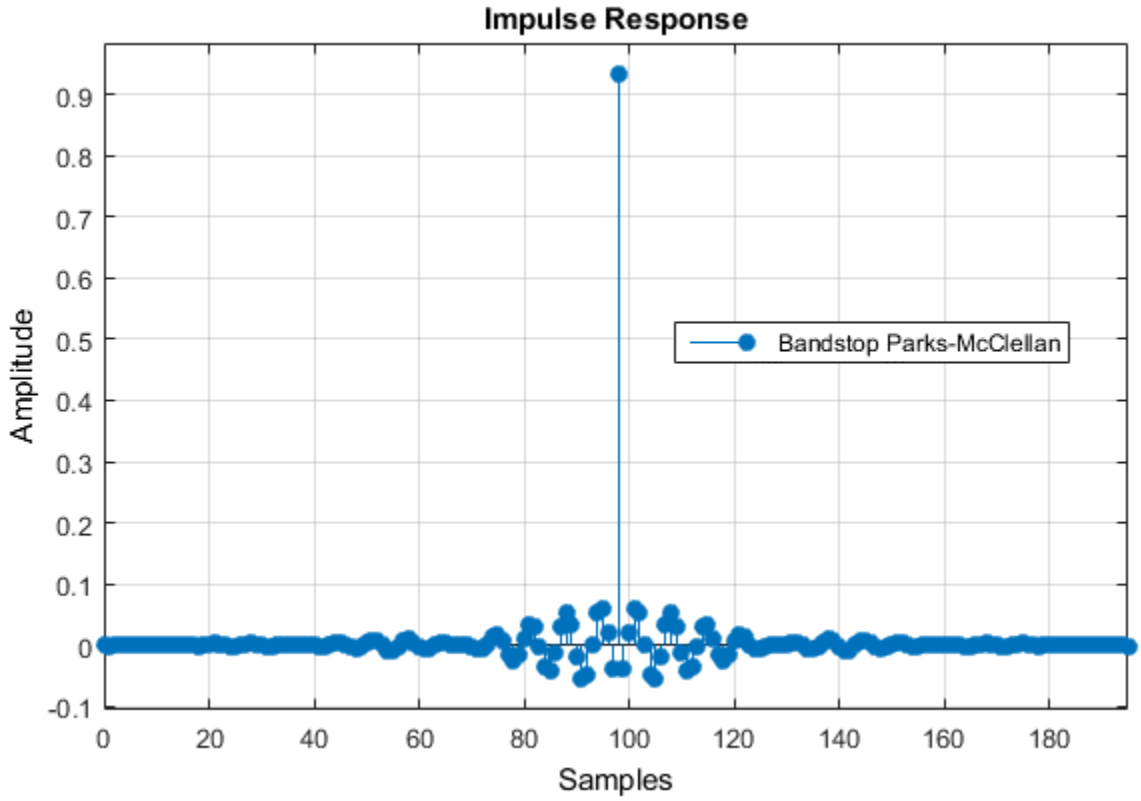

Figure 31: Parks-McClellan Impulse Response

Filter Performance

Filter	Order	Linear Phase?	Quality
Butterworth	20	No	Slight ringing
Chebyshev Type I	14	No	Background ringing barely discernable
Chebyshev Type II	14	No	About the same as Butterworth
Elliptic	**12**	**No**	**No discernable ringing**
Kaiser	354	Yes	Some ringing still present
Parks-McClellan	184	Yes	Some ringing still present, less than Kaiser

Table 1: Filter Comparison

On a purely subjective basis, the elliptic and Chebyshev Type I filters were noticeably better than the other candidates. The elliptic and Chebyshev Type I filters removed the high frequency ringing that most likely existed in the transition bands. With their high roll-offs, the Chebyshev Type I and elliptic filters made the filtered song pleasant to listen to.

On a more objective basis, the Chebyshev Type I and elliptic filters successfully filtered out the noise in the 1700-1900 Hz band. What made the elliptic filter slightly better was its sharper transition which removed the last of the noise from the transition bands (Figure 32).

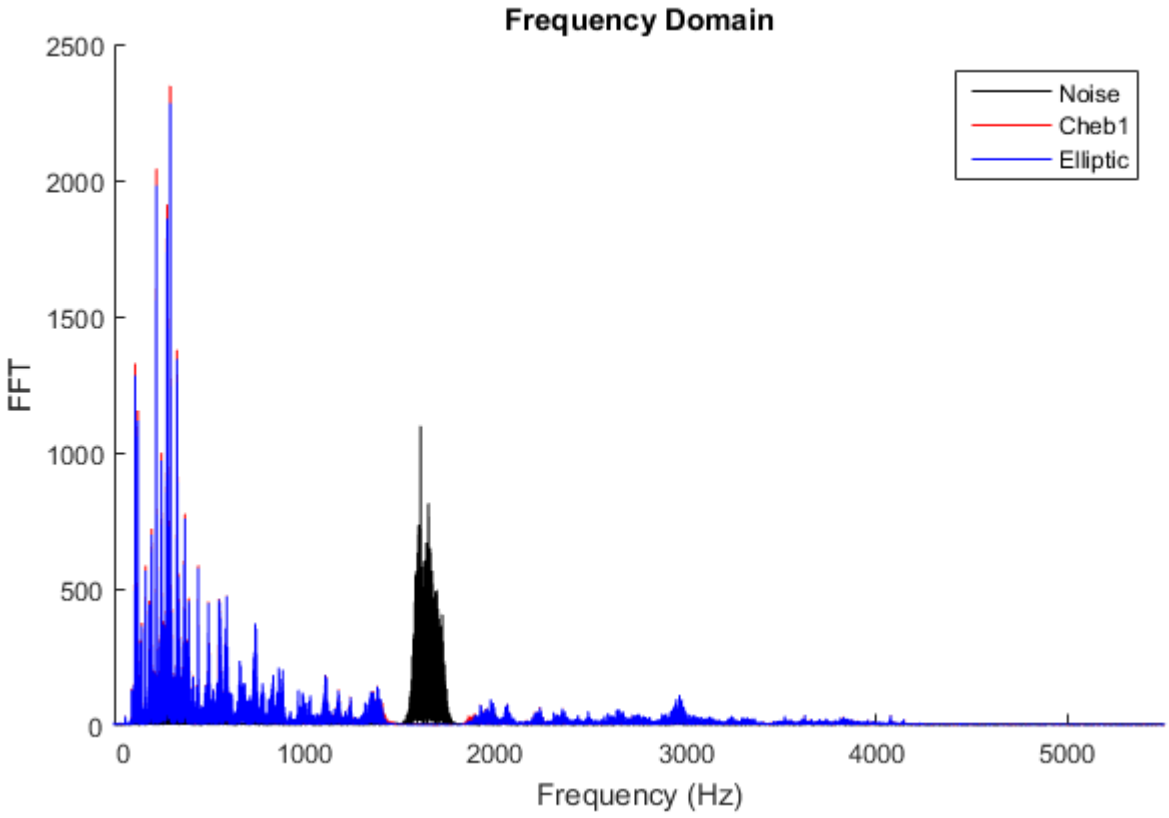

Figure 32: Frequency Domain

Appendix

BUTTERWORTH

```
function y = butter_test(x)
%BUTTER_TEST Filters input x and returns output y.

% MATLAB Code
% Generated by MATLAB(R) 8.6 and the DSP System Toolbox 9.1.
% Generated on: 13-Dec-2015 12:07:23

%#codegen

% To generate C/C++ code from this function use the codegen command.
% Type 'help codegen' for more information.

persistent Hd;

if isempty(Hd)

    % The following code was used to design the filter coefficients:
    %
    % Fpass1 = 1400;   % First Passband Frequency
    % Fstop1 = 1600;   % First Stopband Frequency
    % Fstop2 = 1700;   % Second Stopband Frequency
    % Fpass2 = 1900;   % Second Passband Frequency
    % Apass1 = 0.5;    % First Passband Ripple (dB)
    % Astop  = 100;    % Stopband Attenuation (dB)
    % Apass2 = 1;      % Second Passband Ripple (dB)
    % Fs     = 11025;  % Sampling Frequency
    %
    % h = fdesign.bandstop('fp1,fst1,fst2,fp2,ap1,ast,ap2', Fpass1, Fstop1, ...
    %                      Fstop2, Fpass2, Apass1, Astop, Apass2, Fs);
    %
    % Hd = design(h, 'butter', ...
    %     'MatchExactly', 'stopband', ...
    %     'SystemObject', true);

    Hd = dsp.BiquadFilter( ...
        'Structure', 'Direct form II', ...
        'SOSMatrix', [1 -1.19109457724896 1 1 -0.980328424576093 ...
        0.963132878071047; 1 -1.19109457724896 1 1 -1.33318904630876 ...
        0.968603590486784; 1 -1.19109457724896 1 1 -0.966025775271948 ...
        0.897114330387966; 1 -1.19109457724896 1 1 -1.28015440920878 ...
        0.910624460100616; 1 -1.19109457724896 1 1 -1.2209273416032 ...
        0.861673973791691; 1 -1.19109457724896 1 1 -0.974023145117099 ...
        0.845694548694776; 1 -1.19109457724896 1 1 -1.15916088762967 ...
        0.824721179100316; 1 -1.19109457724896 1 1 -1.00158300525366 ...
        0.812161435480477; 1 -1.19109457724896 1 1 -1.09891304620724 ...
        0.802630775168496; 1 -1.19109457724896 1 1 -1.04476587509749 ...
        0.797900644579592], ...
        'ScaleValues', [0.97693211531984; 0.97693211531984; ...
        0.947174457545188; 0.947174457545188; 0.923893301844649; ...
        0.923893301844649; 0.908015947672573; 0.908015947672573; ...
        0.8999903406633; 0.8999903406633; 1]);
end

s = double(x);
y = step(Hd,s);
y=y./max(abs(y));
audiowrite('clean_Butter.wav',y,11025);
```

CHEBYSHEV TYPE I

```
function y = cheb1_test(x)
%CHEB1_TEST Filters input x and returns output y.

% MATLAB Code
% Generated by MATLAB(R) 8.6 and the DSP System Toolbox 9.1.
% Generated on: 13-Dec-2015 12:08:18

%#codegen

% To generate C/C++ code from this function use the codegen command.
% Type 'help codegen' for more information.

persistent Hd;

if isempty(Hd)

    % The following code was used to design the filter coefficients:
    %
    % Fpass1 = 1400;   % First Passband Frequency
    % Fstop1 = 1600;   % First Stopband Frequency
    % Fstop2 = 1700;   % Second Stopband Frequency
    % Fpass2 = 1900;   % Second Passband Frequency
    % Apass1 = 0.5;    % First Passband Ripple (dB)
    % Astop  = 100;    % Stopband Attenuation (dB)
    % Apass2 = 1;      % Second Passband Ripple (dB)
    % Fs     = 11025;  % Sampling Frequency
    %
    % h = fdesign.bandstop('fp1,fst1,fst2,fp2,ap1,ast,ap2', Fpass1, Fstop1, ...
    %                      Fstop2, Fpass2, Apass1, Astop, Apass2, Fs);
    %
    % Hd = design(h, 'cheby1', ...
    %     'MatchExactly', 'stopband', ...
    %     'SystemObject', true);

    Hd = dsp.BiquadFilter( ...
        'Structure', 'Direct form II', ...
        'SOSMatrix', [1 -1.19109457724896 1 1 -0.944149011850109 ...
        0.973798414849296; 1 -1.19109457724896 1 1 -1.36887580249377 ...
        0.978408201077349; 1 -1.19109457724896 1 1 -0.874167408701267 ...
        0.898973788280301; 1 -1.19109457724896 1 1 -1.3500293652318 ...
        0.918326777233549; 1 -1.19109457724896 1 1 -1.28067939458272 ...
        0.777152115990144; 1 -1.19109457724896 1 1 -0.744444676222408 ...
        0.719633757020691; 1 -1.19109457724896 1 1 -0.899161809643456 ...
        0.509807578370859], ...
        'ScaleValues', [0.979366707361909; 0.979366707361909; ...
        0.943432867998684; 0.943432867998684; 0.860189949036155; ...
        0.860189949036155; 0.754903789185429; 1]);
end

s = double(x);
y = step(Hd,s);
% Impulse response settles down after 15 samples
% Remove first 15 samples
y(1:15)=0;
% Normalize between -1 and 1
y=y./max(abs(y));
audiowrite('clean_Cheb1.wav',y,11025);
```

CHEBYSHEV TYPE II

```
function y = cheb2_test(x)
%CHEB2_TEST Filters input x and returns output y.

% MATLAB Code
% Generated by MATLAB(R) 8.6 and the DSP System Toolbox 9.1.
% Generated on: 13-Dec-2015 12:08:50

%#codegen

% To generate C/C++ code from this function use the codegen command.
% Type 'help codegen' for more information.

persistent Hd;

if isempty(Hd)

    % The following code was used to design the filter coefficients:
    %
    % Fpass1 = 1400;   % First Passband Frequency
    % Fstop1 = 1600;   % First Stopband Frequency
    % Fstop2 = 1700;   % Second Stopband Frequency
    % Fpass2 = 1900;   % Second Passband Frequency
    % Apass1 = 0.5;    % First Passband Ripple (dB)
    % Astop  = 100;    % Stopband Attenuation (dB)
    % Apass2 = 1;      % Second Passband Ripple (dB)
    % Fs     = 11025;  % Sampling Frequency
    %
    % h = fdesign.bandstop('fp1,fst1,fst2,fp2,ap1,ast,ap2', Fpass1, Fstop1, ...
    %                      Fstop2, Fpass2, Apass1, Astop, Apass2, Fs);
    %
    % Hd = design(h, 'cheby2', ...
    %     'MatchExactly', 'stopband', ...
    %     'SystemObject', true);

    Hd = dsp.BiquadFilter( ...
        'Structure', 'Direct form II', ...
        'SOSMatrix', [1 -1.24531029499365 1 1 -0.991983826052835 ...
        0.954422875092953; 1 -1.13402336637907 1 1 -1.31608268163385 ...
        0.960665393118484; 1 -1.23480289639158 1 1 -0.987525704849297 ...
        0.87847600070225; 1 -1.14554909272881 1 1 -1.24356407639515 ...
        0.891600569352674; 1 -1.21558056483997 1 1 -1.16254906230578 ...
        0.842367831947616; 1 -1.16604248003981 1 1 -1.02167023604817 ...
        0.83225398716707; 1 -1.19109457724896 1 1 -1.08429705223745 ...
        0.820673308314146], ...
        'ScaleValues', [0.974290896092787; 0.974290896092787; ...
        0.939715203602687; 0.939715203602687; 0.917802814471459; ...
        0.917802814471459; 0.910336654157073; 1]);
end

s = double(x);
y = step(Hd,s);
% Impulse response settles down after ~ 28 samples
% Ignore first 28 samples
y(1:28)=0;
% Bound between -1 and 1
y=y./max(abs(y));
audiowrite('clean_Cheb2.wav',y,11025);
```

ELLIPTIC

```
function y = elliptic_test(x)
%ELLIPTIC_TEST Filters input x and returns output y.

% MATLAB Code
% Generated by MATLAB(R) 8.6 and the DSP System Toolbox 9.1.
% Generated on: 13-Dec-2015 12:09:18

%#codegen

% To generate C/C++ code from this function use the codegen command.
% Type 'help codegen' for more information.

persistent Hd;

if isempty(Hd)

    % The following code was used to design the filter coefficients:
    %
    % Fpass1 = 1400;   % First Passband Frequency
    % Fstop1 = 1600;   % First Stopband Frequency
    % Fstop2 = 1700;   % Second Stopband Frequency
    % Fpass2 = 1900;   % Second Passband Frequency
    % Apass1 = 0.5;    % First Passband Ripple (dB)
    % Astop  = 100;    % Stopband Attenuation (dB)
    % Apass2 = 1;      % Second Passband Ripple (dB)
    % Fs     = 11025;  % Sampling Frequency
    %
    % h = fdesign.bandstop('fp1,fst1,fst2,fp2,ap1,ast,ap2', Fpass1, Fstop1, ...
    %                      Fstop2, Fpass2, Apass1, Astop, Apass2, Fs);
    %
    % Hd = design(h, 'ellip', ...
    %     'MatchExactly', 'both', ...
    %     'SystemObject', true);

    Hd = dsp.BiquadFilter( ...
        'Structure', 'Direct form II', ...
        'SOSMatrix', [1 -1.12677877928282 1 1 -0.828458819100098 ...
        0.899177919339725; 1 -1.25180680764017 1 1 -1.38069539555517 ...
        0.921186660004898; 1 -1.16757466579574 1 1 -1.26893005210355 ...
        0.672115818827893; 1 -1.2141148174614 1 1 -0.564888943974982 ...
        0.561628515567345; 1 -1.10384883940518 1 1 -0.931191370102105 ...
        0.978450438565336; 1 -1.27183907814981 1 1 -1.38183833986224 ...
        0.982451669581542], ...
        'ScaleValues', [1.58772242934511; 1.58772242934511; ...
        4.00379512721522; 4.00379512721522; 0.110695037322561; ...
        0.110695037322561; 1]);
end

s = double(x);
y = step(Hd,s);
% Bound between -1 and 1
% Impulse response settles down after ~ 17 samples
% Ignore first 17 samples
y(1:17)=0;
y=y./max(abs(y));
audiowrite('clean_Elliptic.wav',y,11025);
```

KAISER

```
function y = kaiser_test(x)
%KAISER_TEST Filters input x and returns output y.

% MATLAB Code
% Generated by MATLAB(R) 8.6 and the DSP System Toolbox 9.1.
% Generated on: 13-Dec-2015 12:10:23

%#codegen

% To generate C/C++ code from this function use the codegen command. Type
% 'help codegen' for more information.

persistent Hd;

if isempty(Hd)

    % The following code was used to design the filter coefficients:
    % % FIR Window Bandstop filter designed using the FIR1 function.
    %
    % % All frequency values are in Hz.
    % Fs = 11025;  % Sampling Frequency
    %
    % Fpass1 = 1400;            % First Passband Frequency
    % Fstop1 = 1600;            % First Stopband Frequency
    % Fstop2 = 1700;            % Second Stopband Frequency
    % Fpass2 = 1900;            % Second Passband Frequency
    % Dpass1 = 0.028774368332;  % First Passband Ripple
    % Dstop  = 1e-05;           % Stopband Attenuation
    % Dpass2 = 0.028774368332;  % Second Passband Ripple
    % flag   = 'scale';         % Sampling Flag
    %
    % % Calculate the order from the parameters using KAISERORD.
    % [N,Wn,BETA,TYPE] = kaiserord([Fpass1 Fstop1 Fstop2 Fpass2]/(Fs/2), [1 ...
    %                              0 1], [Dpass1 Dstop Dpass2]);
    %
    % % Calculate the coefficients using the FIR1 function.
    % b  = fir1(N, Wn, TYPE, kaiser(N+1, BETA), flag);

    Hd = dsp.FIRFilter( ...
        'Numerator', [6.56153688222712e-07 5.2249100959207e-07 ...
        -5.01282059316627e-07 -1.7993100623494e-06 -1.89491379751209e-06 ...
        1.67199451283142e-07 3.287261056848e-06 4.50557872436725e-06 ...
        1.53718772167542e-06 -4.31886286165996e-06 -8.11613920815473e-06 ...
        -5.21276361309272e-06 3.71096963322e-06 1.16686171016685e-05 ...
        1.06780070265017e-05 -4.78930124682955e-07 -1.3465267410279e-05 ...
        -1.65494505683495e-05 -5.32212332607869e-06 1.19280340569405e-05 ...
        2.03783841292684e-05 1.19039859483253e-05 -6.78763107727459e-06 ...
        -1.95944091715384e-05 -1.57418693371942e-05 1.94582787535658e-07 ...
        1.31340531736552e-05 1.26654149557076e-05 2.97948395141583e-06 ...
        -3.14974023489601e-06 -2.13545837611823e-18 3.72280412422975e-06 ...
        -4.1628217737219e-06 -2.09232767042945e-05 -2.56650414499744e-05 ...
        -4.49995918249966e-07 4.31131698206945e-05 6.36043530675154e-05 ...
        2.61391514486973e-05 -5.44467125662112e-05 -0.000110845188315403 ...
        -7.72726164441393e-05 4.11319905140002e-05 0.00015287602477497 ...
        0.000148990606635157 6.36278044280809e-06 -0.000170792304256256 ...
        -0.00022538238298432 -8.68555065897738e-05 0.000148411679019094 ...
        0.000282323431620743 0.000184475997548786 -8.10901889742343e-05 ...
        -0.000295497803771656 -0.000270134711665566 -1.737498422900005e-05 ...
        0.000251785430668264 0.000309876455261776 0.000113814678336449 ...
        -0.000159883303210391 -0.000278788139884979 -0.000163426374058597 ...
        5.48886518029912e-05 0.000176021781128319 0.000125760695710597 ...
```

```
7.46050687948777e-06 -3.44229909015708e-05 1.543823047587 35e-05 ...
3.21104836452406e-05 -8.14706698288458e-05 -0.000236169497347085 ...
-0.00021179715633673 9.20716782949775e-05 0.000468297167670055 ...
0.000525034061585348 6.85609051633612e-05 -0.000613485976669449 ...
-0.000908418549492577 -0.000420690154785993 0.000574455607389534 ...
0.00125005859797912 0.000916337179976936 -0.000295364317616398 ...
-0.00142090005214382 -0.00143863561309194 -0.00020317531854819 ...
0.00132245309524486 0.00182971791240046 0.000810743645354519 ...
-0.000935528885376239 -0.00194349161428207 -0.0013470452929531 ...
0.000349986348956637 0.00170824299405548 0.00161593797142194 ...
0.000241897002413327 -0.00117596784400239 -0.00148046719906345 ...
-0.000595292763380522 0.000534876309456941 0.000933877982975317 ...
0.000496840549127838 -7.06486978519532e-05 -0.000137013471650329 ...
0.000142333228952928 7.92021238451818e-05 -0.000601104607533154 ...
-0.00121938657732287 -0.000753785842867809 0.000905525926109994 ...
0.00243099673505528 0.00208469883342459 -0.000463015030944214 ...
-0.00334016552278785 -0.00381365383818674 -0.000844050505011853 ...
0.00351002758092154 0.00547287284111394 0.00284771045204421 ...
-0.00266832132116963 -0.00651288316595962 -0.00509250520939282 ...
0.00084668290972349 0.00649000821120678 0.00694157767575204 ...
0.00156321429281128 -0.0052560269617532 -0.00777429673149634 ...
-0.00387537289551018 0.00308003084826487 0.00722164108577007 ...
0.00529841356278088 -0.000643659807840189 -0.00536126332571264 ...
-0.00520447259162365 -0.00111441725183897 0.00279393084985357 ...
0.00339795290758817 0.00127948501713796 -0.000550912563333877 ...
-0.000289612456245122 0.000706733984409018 -0.00016779941448177 ...
-0.00310194314982617 -0.0047665158560214 -0.00163369440323365 ...
0.00535307411468069 0.0100628610038924 0.00636834485885507 ...
-0.00501012600044961 -0.015099439097922 -0.0136040569393269 ...
0.00104285895212582 0.0180468950478321 0.0220202295731117 ...
0.00675586583027301 -0.0172259066421255 -0.0296277968753111 ...
-0.0175558380291691 0.0116190248856165 0.0342197192287838 ...
0.0295493057955793 -0.00126006184858607 -0.033943689535161 ...
-0.0402979277919713 -0.0126276464621086 0.0278410607762433 ...
0.0472937620266787 0.0278080115461301 -0.0161931293975641 ...
-0.0485911703668989 -0.0414986801751998 0.000561832618768208 ...
0.0433359292750777 0.0510149805221481 0.0165050733260612 ...
-0.0320383933341053 0.945578295743065 -0.0320383933341053 ...
0.0165050733260612 0.0510149805221481 0.0433359292750777 ...
0.000561832618768208 -0.0414986801751998 -0.0485911703668989 ...
-0.0161931293975641 0.0278080115461301 0.0472937620266787 ...
0.0278410607762433 -0.0126276464621086 -0.0402979277919713 ...
-0.033943689535161 -0.00126006184858607 0.0295493057955793 ...
0.0342197192287838 0.0116190248856165 -0.0175558380291691 ...
-0.0296277968753111 -0.0172259066421255 0.00675586583027301 ...
0.0220202295731117 0.0180468950478321 0.00104285895212582 ...
-0.0136040569393269 -0.015099439097922 -0.00501012600044961 ...
0.00636834485885507 0.0100628610038924 0.00535307411468069 ...
-0.00163369440323365 -0.0047665158560214 -0.00310194314982617 ...
-0.00016779941448177 0.000706733984409018 -0.000289612456245122 ...
-0.000550912563333877 0.00127948501713796 0.00339795290758817 ...
0.00279393084985357 -0.00111441725183897 -0.00520447259162365 ...
-0.00536126332571264 -0.000643659807840189 0.00529841356278088 ...
0.00722164108577007 0.00308003084826487 -0.00387537289551018 ...
-0.00777429673149634 -0.0052560269617532 0.00156321429281128 ...
0.00694157767575204 0.00649000821120678 0.00084668290972349 ...
-0.00509250520939282 -0.00651288316595962 -0.00266832132116963 ...
0.00284771045204421 0.00547287284111394 0.00351002758092154 ...
-0.000844050505011853 -0.00381365383818674 -0.00334016552278785 ...
-0.000463015030944214 0.00208469883342459 0.00243099673505528 ...
0.000905525926109994 -0.000753785842867809 -0.00121938657732287 ...
-0.000601104607533154 7.92021238451818e-05 0.000142333228952928 ...
```

```
        -0.000137013471650329 -7.06486978519532e-05 0.000496840549127838 ...
        0.000933877982975317 0.000534876309456941 -0.000595292763380522 ...
        -0.00148046719906345 -0.00117596784400239 0.000241897002413327 ...
        0.00161593797142194 0.00170824299405548 0.000349986348956637 ...
        -0.0013470452929531 -0.00194349161428207 -0.000935528885376239 ...
        0.000810743645354519 0.00182971791240046 0.00132245309524486 ...
        -0.00020317531854819 -0.00143863561309194 -0.00142090005214382 ...
        -0.000295364317616398 0.000916337179976936 0.00125005859797912 ...
        0.000574455607389534 -0.000420690154785993 -0.000908418549492577 ...
        -0.000613485976669449 6.85609051633612e-05 0.000525034061585348 ...
        0.000468297167670055 9.20716782949775e-05 -0.00021179715633673 ...
        -0.000236169497347085 -8.14706698288458e-05 3.21104836452406e-05 ...
        1.54382304758735e-05 -3.44229909015708e-05 7.46050687948777e-06 ...
        0.000125760695710597 0.000176021781128319 5.48886518029912e-05 ...
        -0.000163426374058597 -0.000278788139884979 -0.000159883303210391 ...
        0.000113814678336449 0.000309876455261776 0.000251785430668264 ...
        -1.73749842290005e-05 -0.000270134711665566 -0.000295497803771656 ...
        -8.10901889742343e-05 0.000184475997548786 0.000282323431620743 ...
        0.000148411679019094 -8.68555065897738e-05 -0.00022538238298432 ...
        -0.000170792304256256 6.36278044280809e-06 0.000148990606635157 ...
        0.00015287602477497 4.11319905140002e-05 -7.72726164441393e-05 ...
        -0.000110845188315403 -5.44467125662112e-05 2.61391514486973e-05 ...
        6.36043530675154e-05 4.31131698206945e-05 -4.49995918249966e-07 ...
        -2.56650414499744e-05 -2.09232767042945e-05 -4.1628217737219e-06 ...
        3.72280412422975e-06 -2.13545837611823e-18 -3.14974023489601e-06 ...
        2.97948395141583e-06 1.26654149557076e-05 1.31340531736552e-05 ...
        1.94582787535658e-07 -1.57418693371942e-05 -1.95944091715384e-05 ...
        -6.78763107727459e-06 1.19039859483253e-05 2.03783841292684e-05 ...
        1.19280340569405e-05 -5.32212332607869e-06 -1.65494505683495e-05 ...
        -1.3465267410279e-05 -4.78930124682955e-07 1.06780070265017e-05 ...
        1.16686171016685e-05 3.71096963322e-06 -5.21276361309272e-06 ...
        -8.11613920815473e-06 -4.31886286165996e-06 1.53718772167542e-06 ...
        4.50557872436725e-06 3.287261056848e-06 1.67199451283142e-07 ...
        -1.89491379751209e-06 -1.7993100623494e-06 -5.01282059316627e-07 ...
        5.2249100959207e-07 6.56153688222712e-07]);
end

y = step(Hd,x);
% Disregard sample 177, as it has an unusually high impulse response
y(177)=0;
% Bound between -1 and 1
y=y./max(abs(y));
% linearly interpolate actual value of y(177)
y(177)=-(y(176)+y(178))/2;
audiowrite('clean_Kaiser.wav',y,11025);
```

PARKS-MCCLELLAN

```
function y = pamc_test(x)

%PAMC_TEST Filters input x and returns output y.

% MATLAB Code
% Generated by MATLAB(R) 8.6 and the DSP System Toolbox 9.1.
% Generated on: 13-Dec-2015 14:59:22

%#codegen

% To generate C/C++ code from this function use the codegen command. Type
% 'help codegen' for more information.

persistent Hd;
```

```
if isempty(Hd)

    % The following code was used to design the filter coefficients:
    % % Equiripple Bandstop filter designed using the FIRPM function.
    %
    % % All frequency values are in Hz.
    % Fs = 11025;  % Sampling Frequency
    %
    % Fpass1 = 1400;            % First Passband Frequency
    % Fstop1 = 1600;            % First Stopband Frequency
    % Fstop2 = 1700;            % Second Stopband Frequency
    % Fpass2 = 1900;            % Second Passband Frequency
    % Dpass1 = 0.028774368332;  % First Passband Ripple
    % Dstop  = 1e-05;           % Stopband Attenuation
    % Dpass2 = 0.028774368332;  % Second Passband Ripple
    % dens   = 16;              % Density Factor
    %
    % % Calculate the order from the parameters using FIRPMORD.
    % [N, Fo, Ao, W] = firpmord([Fpass1 Fstop1 Fstop2 Fpass2]/(Fs/2), [1 0 ...
    %                           1], [Dpass1 Dstop Dpass2]);
    %
    % % Calculate the coefficients using the FIRPM function.
    % b  = firpm(N, Fo, Ao, W, {dens});

    Hd = dsp.FIRFilter( ...
        'Numerator', [0.000160039296235752 -0.00327340841804536 ...
        -0.000418939383951311 7.62383463167554e-05 0.00034481604936328 ...
        0.000289614943313885 3.08502314606061e-05 -0.000136363890798895 ...
        -8.27111237553569e-05 1.48238319635741e-05 -8.22222836667422e-05 ...
        -0.000319974678743598 -0.000335852091838317 0.000121390111571906 ...
        0.000772704745451151 0.000928629717009743 0.000172160585434293 ...
        -0.00104910222775769 -0.00163664876285667 -0.000821743041045206 ...
        0.000947901936010082 0.0021940592435902 0.00166793510114932 ...
        -0.000403436556624593 -0.00233523032602963 -0.00240257145250944 ...
        -0.000434532578079916 0.00193562101095158 0.00268448846989101 ...
        0.00121747194984811 -0.00111729563257338 -0.0023064443384796 ...
        -0.00151864465237471 0.000256517990117414 0.001335064714034 ...
        0.00102248283081354 0.000127790833184467 -0.000156418762417842 ...
        0.000292503605352975 0.000412732919244857 -0.000622836430686504 ...
        -0.00206103907539546 -0.00202226276401295 0.000406452624585443 ...
        0.00360419467398028 0.00439411551312249 0.00110944324203413 ...
        -0.00416658326482286 -0.00680156703230628 -0.00372171702732933 ...
        0.00324917451024706 0.00832777966880916 0.00668978628521422 ...
        -0.000899680300288119 -0.00824204533250936 -0.00894025357560687 ...
        -0.00217243267736483 0.00637370842026181 0.00946434480608708 ...
        0.00474219424105215 -0.00333880346249821 -0.00778285069564525 ...
        -0.00547801558023553 0.000467573968422727 0.00432497424255977 ...
        0.00350850073093174 0.00056304049183348 -0.000435632453540443 ...
        0.00111246327165822 0.00164397875772264 -0.00189017714350953 ...
        -0.00716927379385937 -0.00758292185658081 0.00066004397733961 ...
        0.0124768627619328 0.0164322124952787 0.0053695710981417 ...
        -0.0144700682726484 -0.0260611545538405 -0.0160732839479668 ...
        0.0110306483902122 0.0334992132824031 0.0297329652336371 ...
        -0.00130617627483517 -0.0357892839180725 -0.0433211651403085 ...
        -0.0138078286840283 0.0309376486496485 0.0532714462599265 ...
        0.0316991385855097 -0.0186626069377135 -0.0565028764199159 ...
        -0.0486298586989787 0.000665620483072021 0.0513866085268384 ...
        0.0607355636607642 0.0197056878852779 -0.0383192984889689 ...
        0.934872809320677 -0.0383192984889689 0.0197056878852779 ...
        0.0607355636607642 0.0513866085268384 0.000665620483072021 ...
        -0.0486298586989787 -0.0565028764199159 -0.0186626069377135 ...
```

```
        0.0316991385855097 0.0532714462599265 0.0309376486496485 ...
        -0.0138078286840283 -0.0433211651403085 -0.0357892839180725 ...
        -0.00130617627483517 0.0297329652336371 0.0334992132824031 ...
        0.0110306483902122 -0.0160732839479668 -0.0260611545538405 ...
        -0.0144700682726484 0.0053695710981417 0.0164322124952787 ...
        0.0124768627619328 0.000660043977339 61 -0.00758292185658081 ...
        -0.00716927379385937 -0.00189017714350953 0.00164397875772264 ...
        0.00111246327165822 -0.000435632453540443 0.00056304049183348 ...
        0.00350850073093174 0.00432497424255977 0.000467573968422727 ...
        -0.00547801558023553 -0.00778285069564525 -0.00333880346249821 ...
        0.00474219424105215 0.00946434480608708 0.00637370842026181 ...
        -0.00217243267736483 -0.00894025357560687 -0.00824204533250936 ...
        -0.000899680300288119 0.00668978628521422 0.00832777966880916 ...
        0.00324917451024706 -0.00372171702732933 -0.00680156703230628 ...
        -0.00416658326482286 0.00110944324203413 0.00439411551312249 ...
        0.00360419467398028 0.000406452624585443 -0.00202226276401295 ...
        -0.00206103907539546 -0.000622836430686504 0.000412732919244857 ...
        0.000292503605352975 -0.000156418762417842 0.000127790833184467 ...
        0.00102248283081354 0.001335064714034 0.000256517990117414 ...
        -0.00151864465237471 -0.0023064443384796 -0.00111729563257338 ...
        0.00121747194984811 0.00268448846989101 0.00193562101095158 ...
        -0.000434532578079916 -0.00240257145250944 -0.00233523032602963 ...
        -0.000403436556624593 0.00166793510114932 0.0021940592435902 ...
        0.000947901936010082 -0.000821743041045206 -0.00163664876285667 ...
        -0.00104910222775769 0.000172160585434293 0.000928629717009743 ...
        0.000772704745451151 0.000121390111571906 -0.000335852091838317 ...
        -0.000319974678743598 -8.22222836667422e-05 1.48238319635741e-05 ...
        -8.27111237553569e-05 -0.000136363890798895 3.08502314606061e-05 ...
        0.000289614943313885 0.00034481604936328 7.62383463167554e-05 ...
        -0.000418939383951311 -0.00327340841804536 0.000160039296235752]);
end

y = step(Hd,x);
y=y./max(abs(y));
audiowrite('clean_PaMc.wav',y,11025);
```

TESTING THE FILTERS

```
%% Process all filters and output audio files
[noise, Fs] = audioread('noisy.wav'); % Read the noise file
% FFTs
fft_noise = fft(noise);
fft_cheb1 = fft(cheb1_test(noise));
fft_cheb2 = fft(cheb2_test(noise));
fft_elliptic = fft(elliptic_test(noise));
fft_kaiser = fft(kaiser_test(noise));
fft_butter = fft(butter_test(noise));
fft_pamc = fft(pamc_test(noise));
len = length(fft_noise) / 2;
% Plots
figure;
hold on;
plot((1:len), abs(fft_noise(1:len)), 'k'); % Noise
plot((1:len), abs(fft_cheb1(1:len)), 'r'); % Cheb1
plot((1:len), abs(fft_elliptic(1:len)), 'b'); % Elliptic
xtick = (0:5) * 1000 * 2 * len / 11025; % Define freqency
set(gca, 'XTick', xtick); % Set xaxis to evenly space frequency
set(gca, 'XTickLabel', num2str(xtick' * 11025 / 2 / len)); % Set xaxis to hz
title('Frequency Domain', 'FontSize', 16);
legend('Noise','Cheb1','Elliptic');
xlabel('Frequency (Hz)', 'FontSize', 14); % x label
```

```
ylabel('FFT', 'FontSize', 14); % y label
xlim([0 len]);
```