1. Given that the signal in `Raw Signal.mat` contains 3 modulated sound samples for pop, country, and rock music, it must be demodulated by writing a function titled `demod.m` to demodulate the signal at any given carrier frequency and bandwidth.

   For this problem, the sampling frequency, F_s, equals 88,200 Hz for all the stations as well as `yt`, the carrier frequencies of each station is found using the table below

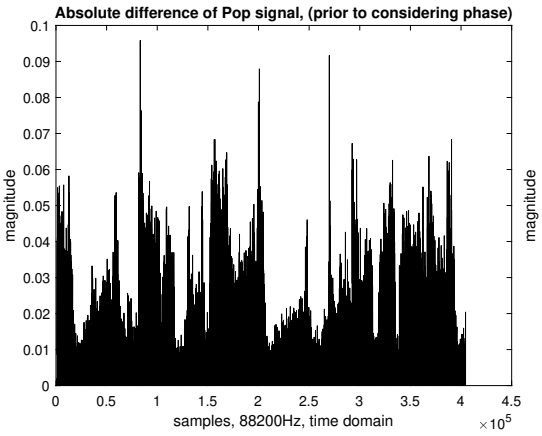
Station	rad/s	Hz
pop	0.7124	10,000
country	1.4248	20,000
rock	2.1371	30,000

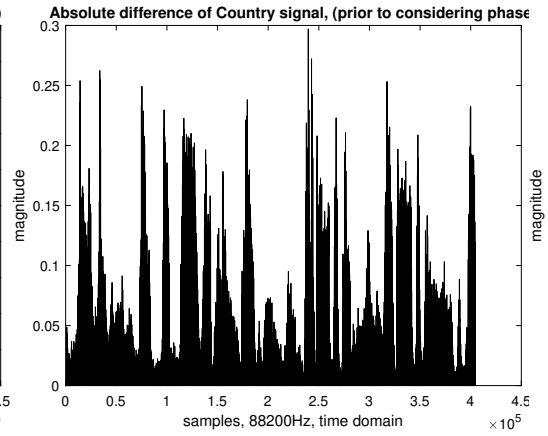
   In addition, each station utilized a bandwidth of around 10,000 Hz, meaning there could be some slight overlap in the radio. To demodulate the signal, the following steps must be taken:

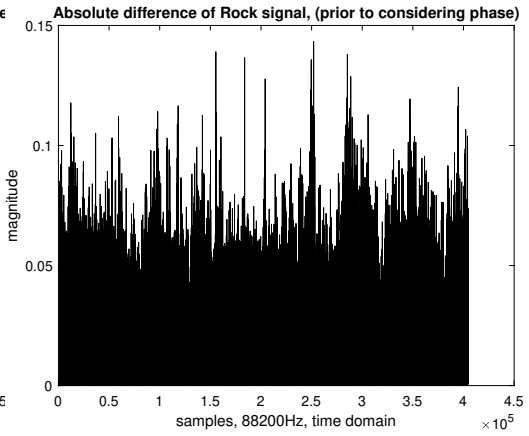
   1. Multiply the signal with a cosinusoidal function of it's carrier frequency. This will convolute replicas of the center frequency to $\omega = 0$, so that the low frequency envelope is reproduced
   2. Apply a lowpass filter to remove any unwanted frequencies
      a. Frequencies produced by the convolution
      b. Frequencies from unwanted bandwidths

   An FIR filter design using a Hanning window will be used. The window selection was arbitrary. The number of coefficients to the Hanning window are 49. This number was selected due to it's ability to reproduce the sound with little-to-no error, as well as keeping the order of the filter to a minimum. The cutoff frequency for the FIR filter will simply be the bandwidth w.r.t. the sampling frequency.

   Here are the absolute differences for the pop, country, and rock signals:



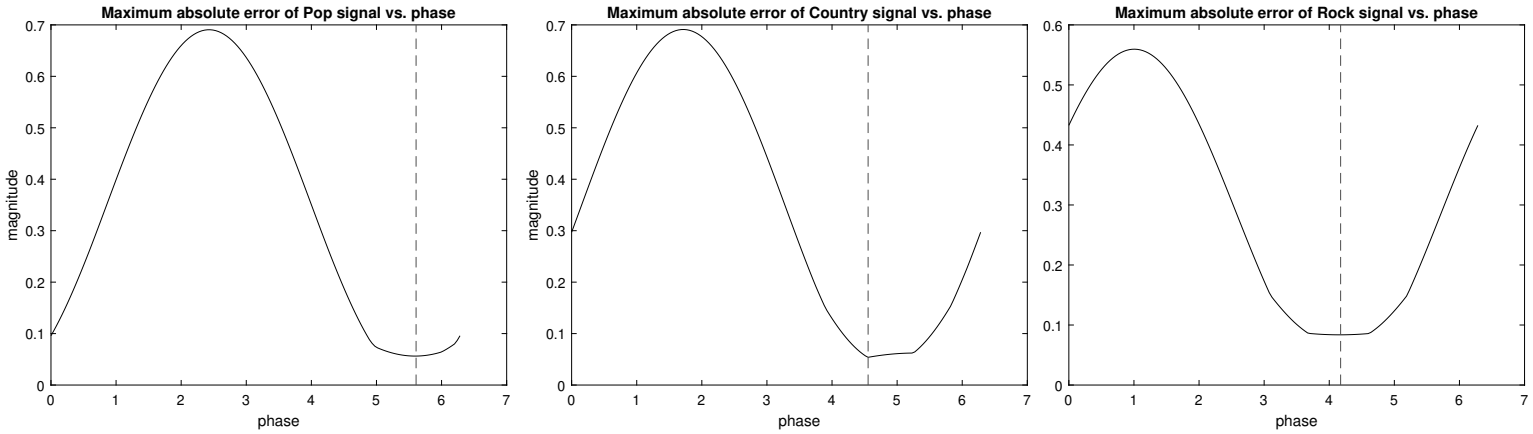




Listing for `demod.m`

```
function yout = demod(yt, Fs, Fc, B, varargin)
% AUTHOR: ARPAD ATTILA VOROS
% DEMOD amplitude demodulates a signal
%   INPUTS: yt - incoming raw signal
%           Fs - sampling freq (Hz)
%           Fc - carrier freq (Hz)
%           B - bandwidth of signal (Hz)
%           varargin - variable input:
%               1. phase - phase delay of multiplied cosinusoid
%                          (defaults to 0)

% PHASE CONSIDERATION
% ================================
if ~isempty(varargin)
    phase = varargin{1};
else
    phase = 0;
end
% ================================

% INITIALIZATION
% ================================
% sampling period
Ts = 1 / Fs;
% number of samples
numsamp = length(yt);
% get time array
t = (0:Ts:(numsamp - 1)* Ts)';
% make same size as input, just in case
t = t(1:numsamp);
% ================================

% REMOVE CARRIER FREQ
% ================================
yout = 2 * yt .* cos(2 * pi * Fc * t - phase);
% ================================

% LOWPASS FILTER
% ================================
% fir filter with 49 coefficients
order = 48;
% use hanning window FIR filter design, lowpass filter
w = window(@hanning, order + 1);
% cutoff frequency
wc = B / Fs;
% get lowpass filter coefficients
low_filt = fir1(order, wc, 'low', w);
% apply lowpass filter to incoming signal
yout = filtfilt(low_filt, 1, yout);
% ================================

% PLAY SOUND
% ================================
sound(yout, Fs);
% ================================
end
```

However, you can clearly see that pop and rock have a distinctly low absolute errors, whereas country's error is significantly high. This is due to the fact that when in the process of demodulating, if you multiply the signal by its carrier frequency and it's out-of-phase, then there will be deconstructive interference, thus reducing the amplitude of the music. Due to this phase dependence, I went ahead and found the maximum error of each of the three stations by implementing a phase offset ranging from $0 \rightarrow 2\pi$. Here are the results for the corresponding phase for the minimum produced error



You can see the error is plotted versus phase, Θ, (in radians), where the incoming signal is multiplied by the following cosinusoid

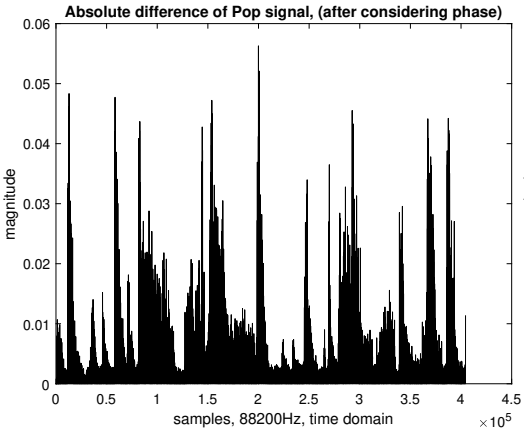
$$y_{demod} = 2y_t cos(2\pi F_c t - \Theta) \tag{1}$$

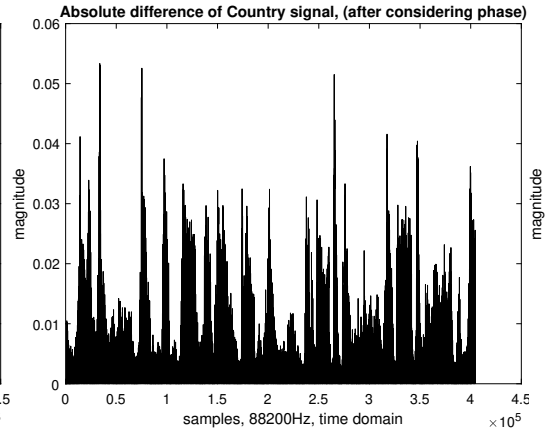
The errors should follow a sinusoidal path, but fail to do so exactly due to the error becoming negative near the lower portions. These are the 'kinks' are seen near the bottom. The minimum of each of the phases for each station is plotted along the dotted line, which are

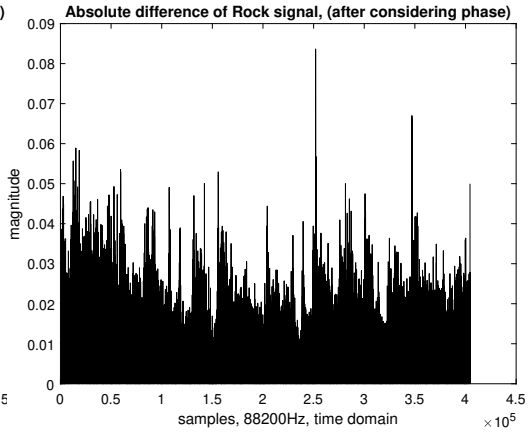
Station	Θ which minimizes error
pop	5.6085
country	4.5543
rock	4.1747

The function `demod.m` now takes into account a variable argument at the end which is this phase offset, Θ. With no argument, it defaults to 0. However with an argument, it takes the value of said argument. The three initial plots are recreated to consider their phases which result in the minimum absolute error in the following page.

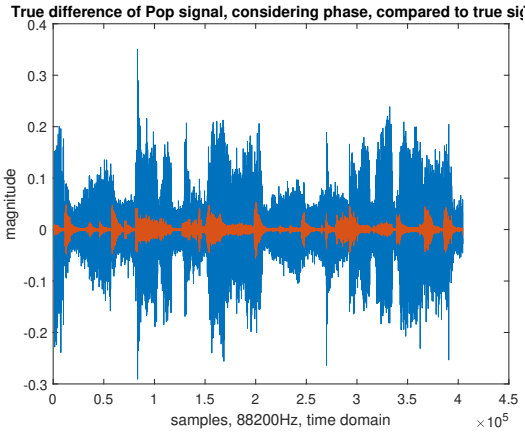
Now considering the phase component in demodulation, we see the following absolute errors:

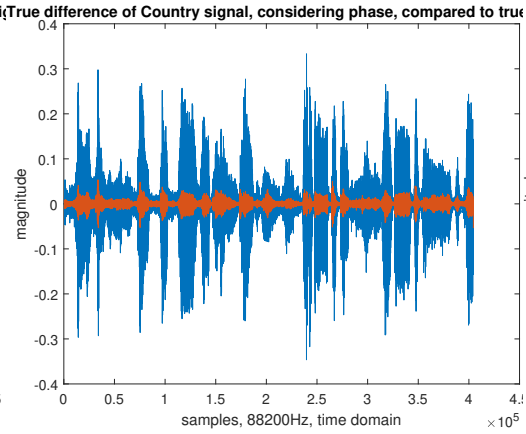


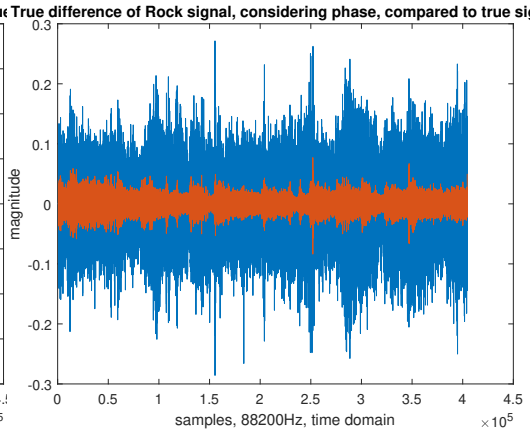




Here we can observe a significant decrease in error for the country station, as well as a little bit for the pop and rock stations as well. To get better grasp at the size of the error, the true pop, country, and rock signals are plotted alongside their respective errors:







This error is small enough to where a human cannot hear the difference between the two.

Some next steps to take for this problem would be to consider quadrature demodulation. Due to having an error in phase as well as a little in magnitude, quadrature demodulation can split up the incoming signal into it's in-phase and quadrature-phase components to observe and accommodate for this phase and magnitude dependent property. However, for now, this routine `demod.m` works just fine.

2. A transfer function for a continuous-time system is given

   (a) Calculate a discrete-time transfer function by applying the impulse invariant approach

   Using the `residue` function in MATLAB, we get the system in the following expanded form

$$H(s) = \frac{2.39}{s+3.12} - \frac{3.42}{s+1.67} + 1.5 \tag{2}$$

   After taking the transfer function to the time domain (inverse Laplace transform), sampling it at $T = 1.15$ (substitute in $nT = t$), taking the transfer function to the Z domain (Z-transform), and multiplying by T, we simply the now-impulse invariant transfer function, which is given by

$$H_{IP}(z) = \frac{0.8855 - 0.6190z^{-1} + 0.005z^{-2}}{1 - 0.1661z^{-1} + 0.00287z^{-2}} \tag{3}$$

   (b) Using the bilinear transformation, which is defined as

$$s \Rightarrow \frac{2(z-1)}{T(z+1)} \tag{4}$$

   where $T = 1.15$ again. We take the transfer function given in (2) and substitute for all s. After simplifying, we get the post-bilinear transform transfer function, given by

$$H_{BT}(z) = \frac{1.0481 - 0.3012^{-1} - 0.3175z^{-2}}{1 + 0.3055z^{-1} - 0.0066z^{-2}} \tag{5}$$

   (c) Plotting the three magnitude responses w.r.t. ω ranging from $0 \to \pi/2$

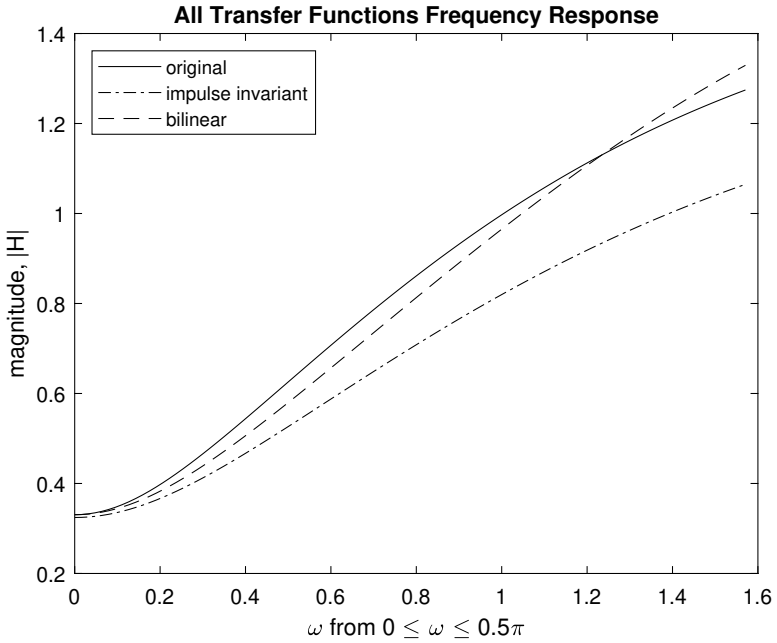


Figure 1: The continuous-time, impulse invariant, and bilinear transfer functions

(d) Since we are attempting to approximate the continuous-time transfer function, the magnitude response closest to the original is the better approximation. In this case, the bilinear transformation of the transfer function resulted in a more similar response, meaning it was the better result.

3. A second order continuous-time transfer function for the Butterworth lowpass filter will be transformed into a digital highpass filter using the bilinear transformation. The transfer function results in an attenuation at the cut-off frequency of 3dB.

   (a) Know the bilinear transformation from (4), and the prewarped frequency, given by

$$\Omega_c \Rightarrow \frac{2}{T}\tan\left(\frac{\omega_c}{2}\right) \tag{6}$$

   we can calculate Ω_c for the new cutoff frequency, $\omega_c = 0.7\pi$. Then using MATLAB, we are able to compute the coefficients of the filter

```
%% 3 A

% omega ranges from 0 to 2pi
w = linspace(0, 2 * pi, 500);

% cutoff ratio, cutoff frequency, and normalized prewarped cutoff ...
    frequency
wc_rat = 0.7;
wc = wc_rat * pi;
Oc_hat = tan(wc / 2);

% bilinear transform performed
z = exp(1i * w);
s = (z - 1) ./ (Oc_hat * (z + 1));

% calculate transfer function (s already substituted)
% this results in the same magnitude response, when plotted
% as the freqz function using the coefficients
Hs = 1 ./ (s.^2 + sqrt(2) * s + 1);

% precalculate some constants
c0 = Oc_hat;
c1 = 1 + sqrt(2) * c0 + c0^2;
% calculate numerator coefficients
b = ones(1, 3);
b(1) = c0^2 / c1;
b(2) = 2 * b(1);
b(3) = b(1);
% calculate denominator coefficients
a = ones(1, 3);
a(2) = 2 * (c0^2 - 1) / c1;
a(3) = (1 - sqrt(2) * c0 + c0^2) / c1;

% see magnitude and phase response of the newly transformed highpass ...
    filter
freqz(b, a);
title("Converted Filter Response using freqz");
```

   Where the coefficients give the new transfer function

$$H_{hp} = \frac{0.1311 - 0.2622z^{-1} + 0.1311z^{-2}}{1 + 0.7478z^{-1} + 0.2722z^{-2}} \tag{7}$$

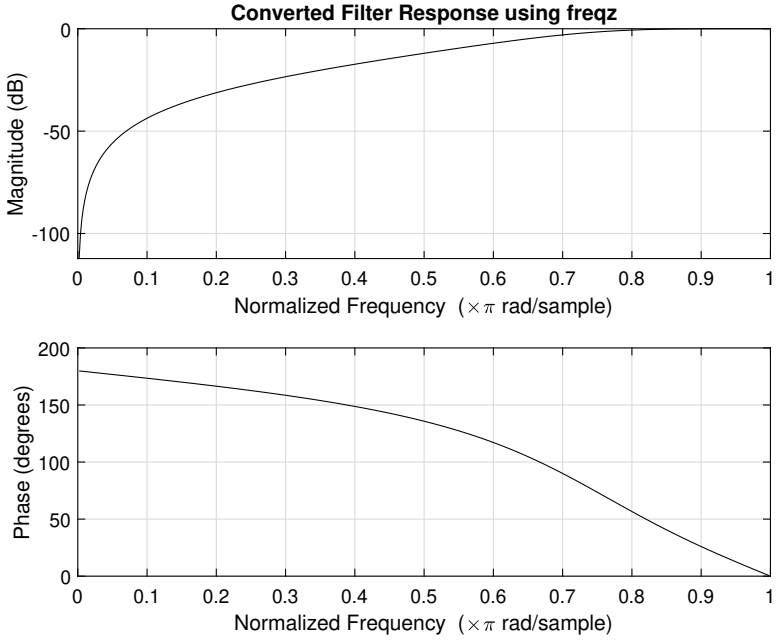


Figure 2: Magnitude and Phase response of the new highpass filter

and we can see that at 0.7π, we attenuate at 3dB

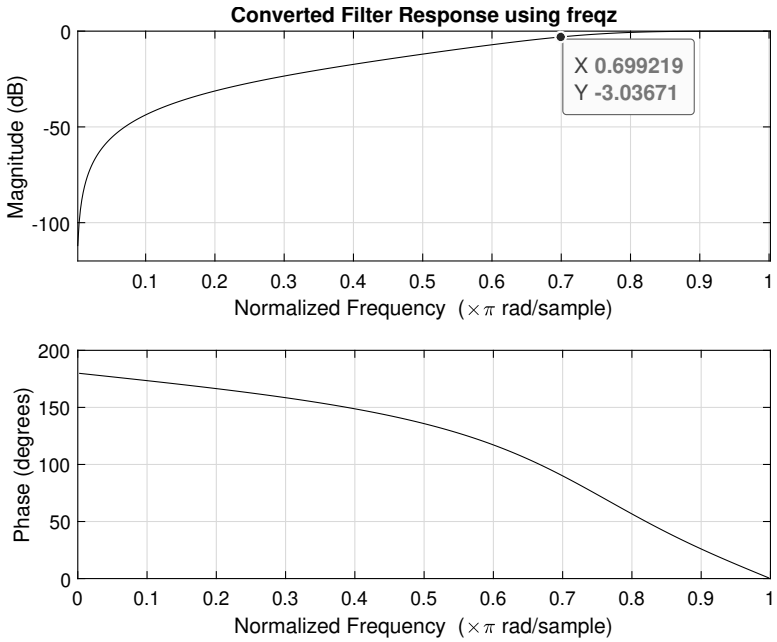


Figure 3: Showing the filter matches specifications

4. A fourth order lowpass filter with 1dB attenuation at the normalized pass band edge $\omega_1 = 0.4\pi$ and an attenuation of 34dB in the stop band transfer function given

   (a) Transforming the discrete-time lowpass filter to a highpass filter, the following transformation must be made

   $$Z^{-1} = -\frac{z^{-1} + \alpha}{1 + \alpha z^{-1}} \tag{8}$$

   where

   $$\alpha = -\frac{\cos(0.5(\omega_c + \hat{\omega_c}))}{\cos(0.5(\omega_c - \hat{\omega_c}))} \tag{9}$$

   In this case, $\hat{\omega_c}$ equals the new cutoff frequency of the transformed filter and ω_c equals the current cutoff frequency of the filter. Now, $H(z = Z(z))$ is calculated, which gets simplified down to the post-transformation transfer function

   $$H(z) = \frac{0.05 - 0.0081z^{-1} + 0.075z^{-2} - 0.0081z^{-3} + 0.05z^{-4}}{1 + 2.2992z^{-1} + 2.6650z^{-2} + 1.5697z^{-3} + 0.4186z^{-4}} \tag{10}$$

   (b) Using the MATLAB `ellip` routine, we can design a filter to follow the same specifications

```
%% 3 B
order = 4;
[b2, a2] = ellip(order, 1, 34, wc_rat, 'high');
```

```
>> b2
    0.0500   -0.0082    0.0750   -0.0082    0.0500

>> a2
    1.0000    2.2992    2.6650    1.5697    0.4186
```

   We can observe that the coefficients are identical (minus the slight error where we got 0.0081 in the numerator but `ellip` got 0.0082)

   (c) Using the MATLAB routine `freqz`, we can see that our filter has the appropriate magnitude response

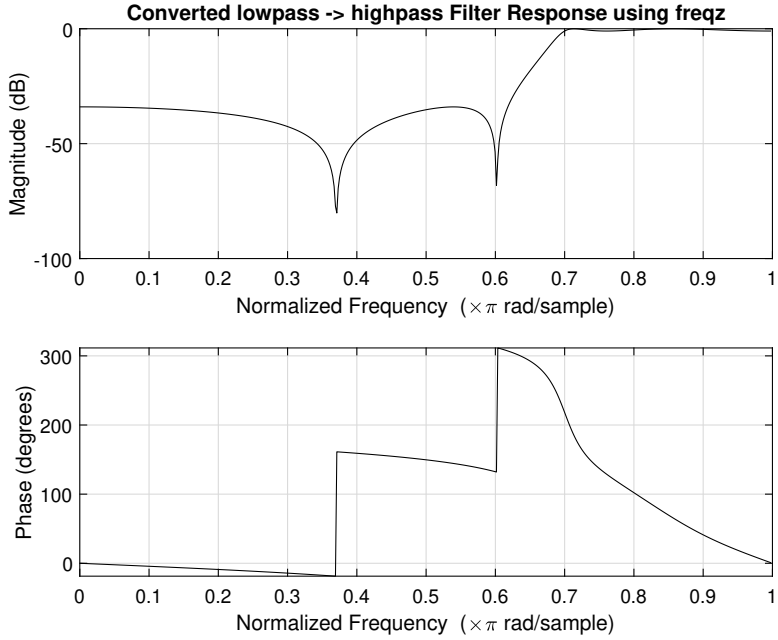


Figure 4: Magnitude and Phase response of the new highpass filter

and we can see that at 0.7π, we attenuate at 1dB, as well as approaching 34dB

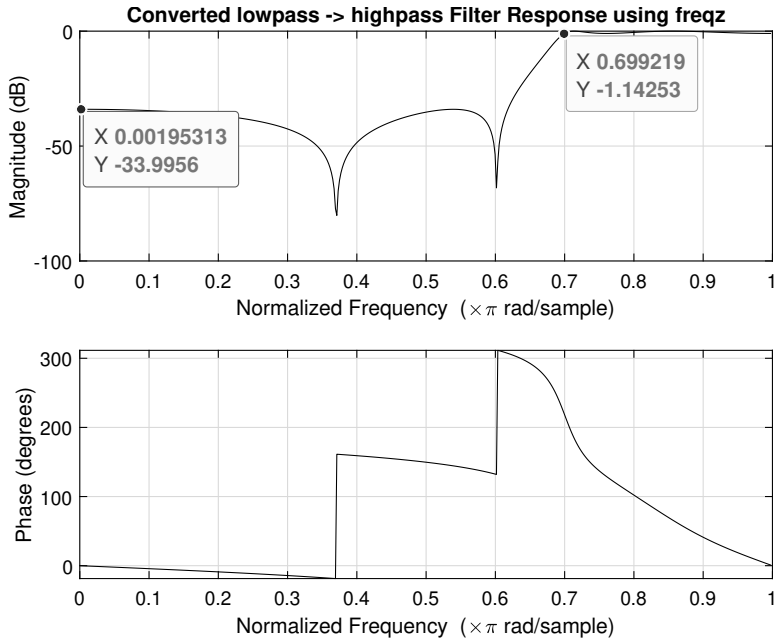


Figure 5: Showing the filter matches specifications

5. Designing an FIR filter using Parks-McClellan optimal equiripple design to meet the following specifications

   - Sampling frequency - 44056 Hz
   - Stop band 1: 0 to 2000 Hz
   - Pass band: 4000 to 18000 Hz
   - Stop band 2: 20000 to 22028 Hz
   - Pass band ripple - $\delta_p = 0.15$
   - Stop band ripple - $\delta_s = 0.01$

   (a) Using the MATLAB routine `firpmord`, we are able to find the optimal order (and other inputs) to the other routein `firpm` given our band edges and ripple values. The proper syntax is found in the following listing

```
%% 5 A
% sampling frequency
Fs = 44056;
% stop band edge 1
ws1 = 2000;
% pass band edges
wp1 = [4000, 18000];
% stop band edge 2
ws2 = 20000;

% ripple values
Δ_s = 0.01;
Δ_p = 0.15;

% array of band edges, used for inputs into firpmord
f = [ws1, wp1, ws2];
% array of the amplitudes for the intermediate sections of
% each band, so first is band stop (0), then band pass (1), then
% band stop again (0)
amp = [0, 1, 0];
% ripple values for the stop/pass bands respectively
ripple = [Δ_s, Δ_p, Δ_s];

% pass them into firpmord along with sampling frequency
% to receive inputs to firpm
[order, f0, a0, w] = firpmord(f, amp, ripple, Fs);
```

   The best order system to meet our specifications from `firpmord` yields `order` to be 25.

(b) Plotting the magnitude (not in dB) and phase response of the filter vs. our ideal magnitude specifications

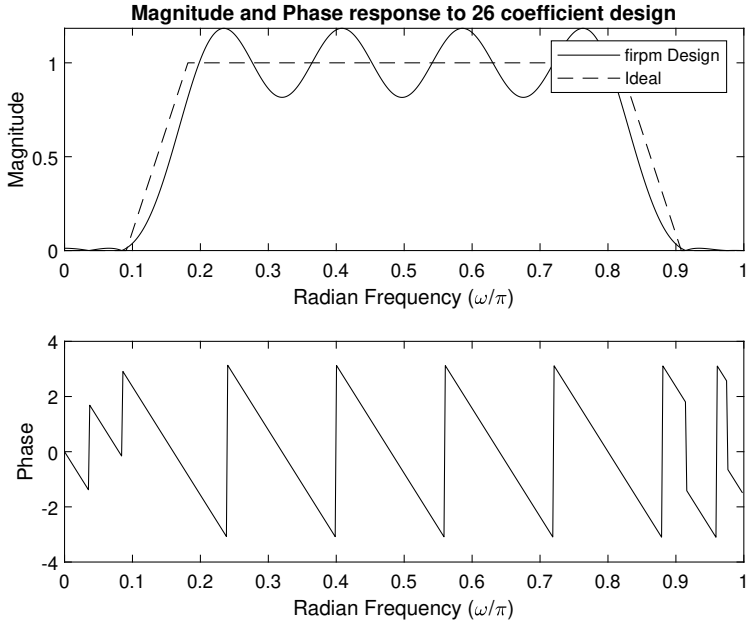


Figure 6: Filter designed using `firpmord` and `firpm`

(c) Showing the designed ripple boundaries versus what the ripple is actually at

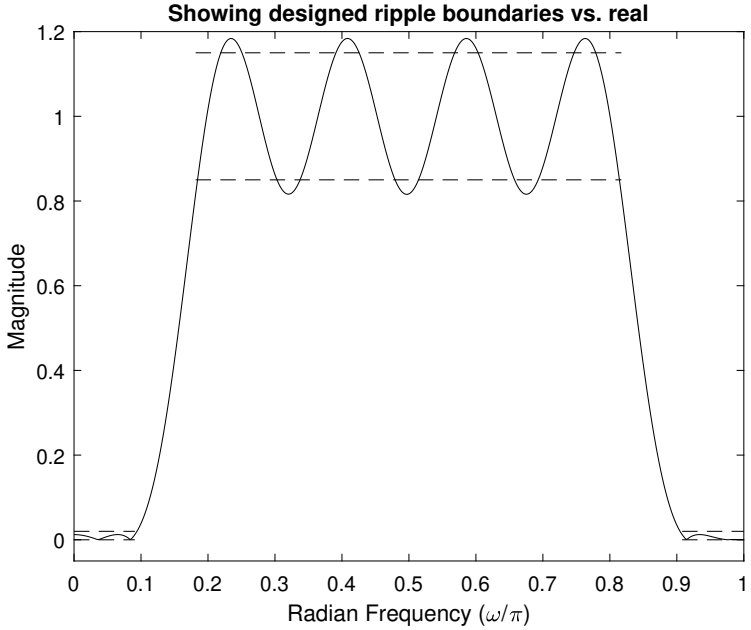


Figure 7: Resulting ripple boundaries of `firpmord`

(d) Modifying the `del` argument within `firpmord` to match the ideal ripple boundaries, the parameters passed into `firpmmord` to match are

$$\delta_p = 0.115$$

$$\delta_s = 0.015$$

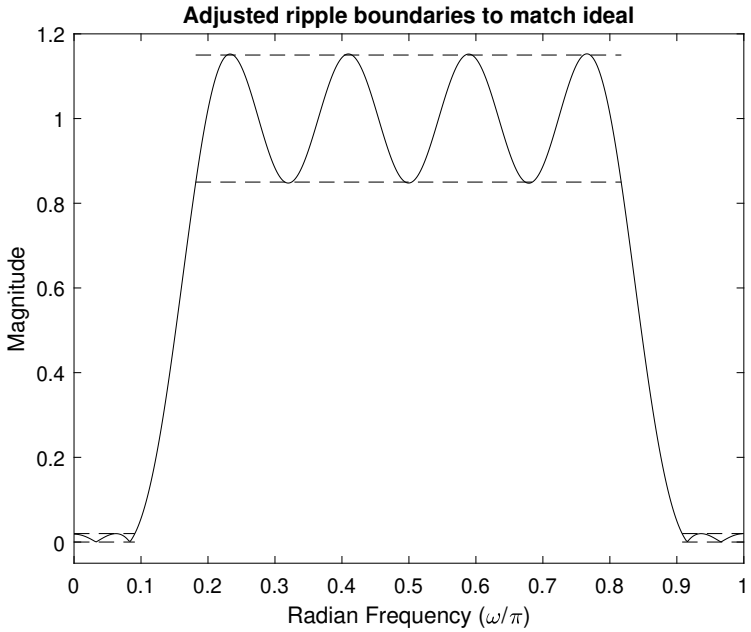


Figure 8: Matching ripple boundaries using `firpmord`

Using guessing and checking, I was able to get a near perfect match with the ripple values shown above. Note that the dotted lines represent the constant, ideal ripple boundaries, and only the filter response changed. With these inputs to `firpmord`, we were able to match our ideal ripple boundaries of $\delta_p = 0.15$ and $\delta_s = 0.01$.

Plotting the magnitude (not in dB) and phase response of the adjusted filter vs. our ideal magnitude specifications

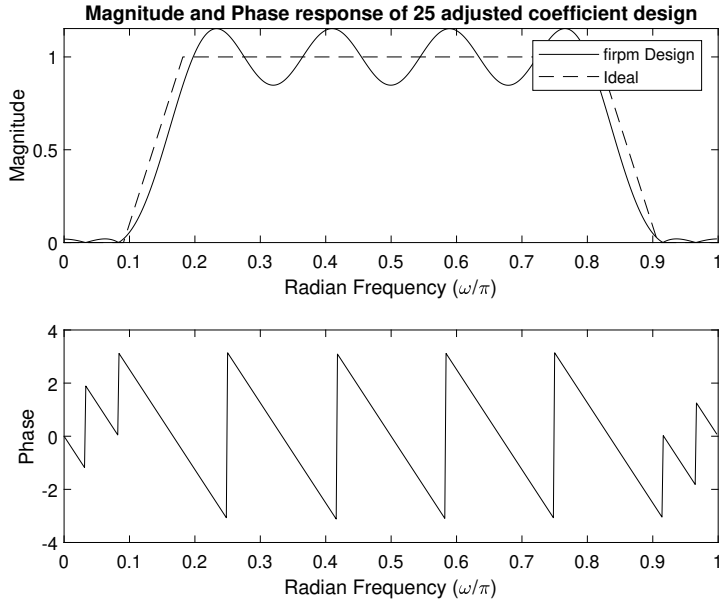


Figure 9: Adjusted magnitude and phase response

(e) Approximations of the pass band and stop band edges can be crudely shown in the following image. Note that during calculations, values will be slightly adjusted

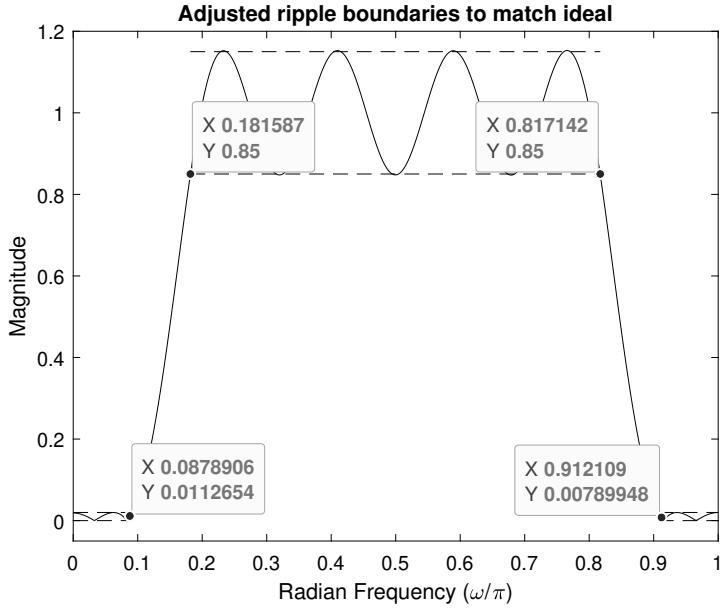


Figure 10: Approximations for pass band and stop band edges

where

$$\omega^1_{c,stop} \approx \left[0, 0.0885\frac{Fs}{2}\right] = [0, 1949]$$

$$\omega^1_{c,pass} \approx \left[0.1816\frac{Fs}{2}, 0.8171\frac{Fs}{2}\right] = [4000, 18000]$$

$$\omega^2_{c,stop} \approx \left[0.91\frac{Fs}{2}, \frac{Fs}{2}\right] = [20045, 22028]$$

The pass band shows the ideal values, since it just so happens to show exactly at $1 - \delta_p = 0.85$, whereas the other two are approximated. However, all of our band edges seem to match what our original filter was designed to for!