The University of Texas at Austin

Spring 2025 EE 445S Real-Time Digital Signal Processing Laboratory Prof. Evans

Homework #3 Solutions

3.1. Using Filtering to Improve Signal Quality. 27 points.

Johnson, Sethares & Klein, exercise 4.21, on page 78. ***9 points for each of the two parts.***

For the simulation of the narrowband interferers in parts (a) and (b) please use a chirp signal.

In addition, please complete the following part: (c) for the narrowband interference occurring inside the transmission band that you simulated in part (b), design a notch filter to use after the bandpass filter. For the notch filter, please use a second-order infinite impulse response (IIR) filter. Write the Matlab code to apply the notch filter to the output of the bandpass filter used to reduce out-of-band interference, and compute the change in SNR due to your notch filter. ***9 points.***

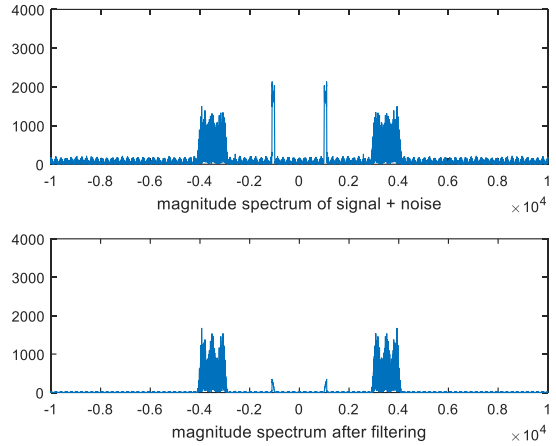
Prologue: This problem explores limitations in improving quality of a signal that has been corrupted by narrowband interference. The interference can be at the fundamental frequency or its harmonics emitted by an oscillator. Here are several examples:

- In a laptop, electromagnetic interference from the LCD pixel clock circuitry and its first 100 harmonics are significant sources of interference; a 65 MHz LCD pixel clock's harmonic clobbers one of the IEEE 802.11g wireless LAN channels.
- On power lines, power electronics for electronically-ballasted compact fluorescent lights generate narrowband interferers (harmonics) in the 20-80 kHz powerline communication band.
- WWVB timing signal is broadcast from a single transmitter in Ft. Collins, Colorado, at a carrier frequency of 60 kHz. From this signal, which is available in North America, the Caribbean, and West Africa, watches and clocks can automatically adjust and reset time. However, this narrowband signal can be an interferer, e.g. in the 20-80 kHz powerline communication band.
- "Impact of power line telecommunication systems on radiocommunication systems operating below 80 MHz", June 2013
- A 50/60 Hz power signal, and its odd harmonics, can couple into other lines in a system.
- In cellular communications, reflections of transmitted signal by reflectors in the environment (e.g. nearby metal fences) fall in the transmission band as well as in adjacent bands (due to passive intermodulation)

Solutions for (a) and (b): In part (a), the chirp signal occupies different frequencies from those of the signal of interest, whereas chirp signal falls within the range of frequencies of the signal of interest in part (b). The signal of interest occupies 3000-4000 Hz. We will model the chirp signal using a cosine that increases in frequency over time, starts with frequency of 1000 Hz and ends with 1100 Hz for part (a) and starts with 3500 Hz and ends with 3600 Hz for part (b).

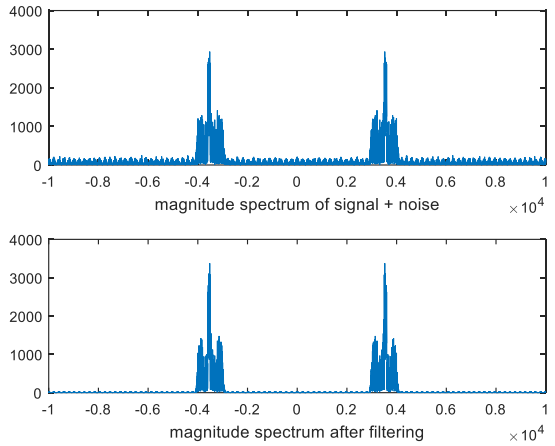
We design a bandpass finite impulse response (FIR) filter with passband 3000-4000 Hz using the Parks-McClellan algorithm. Hence, the FIR filter will have ripples in the passband and stopband of the magnitude response. In the stopband, the magnitude response will vary with frequency. When narrowband interference occurs in the stopband, it will be attenuated, and some of it will likely get through. ***For this problem, we use interference and noise interchangeably.***

This problem uses signal-to-noise ratio (SNR) to measure signal quality. SNR assumes that signal and noise are in theory statistically independent and in practice uncorrelated. We will discuss more about these assumptions after midterm #1.



(a) Spectrum of signal plus interference outside of transmission band (above) and bandpass filter response (below).

$\text{SNR}_{\text{inp}} = 0.9776 \quad \text{SNR}_{\text{out}} = 67.0338$



(b) Spectrum of signal plus interference inside of transmission band (above) and bandpass filter response (below).

$\text{SNR}_{\text{inp}} = 0.9675 \quad \text{SNR}_{\text{out}} = 0.7430$

For part (a), the signal and interference occupy different frequency bands. The bandpass filter removes most of the energy of the out-of-band interference and output SNR increases as a result.

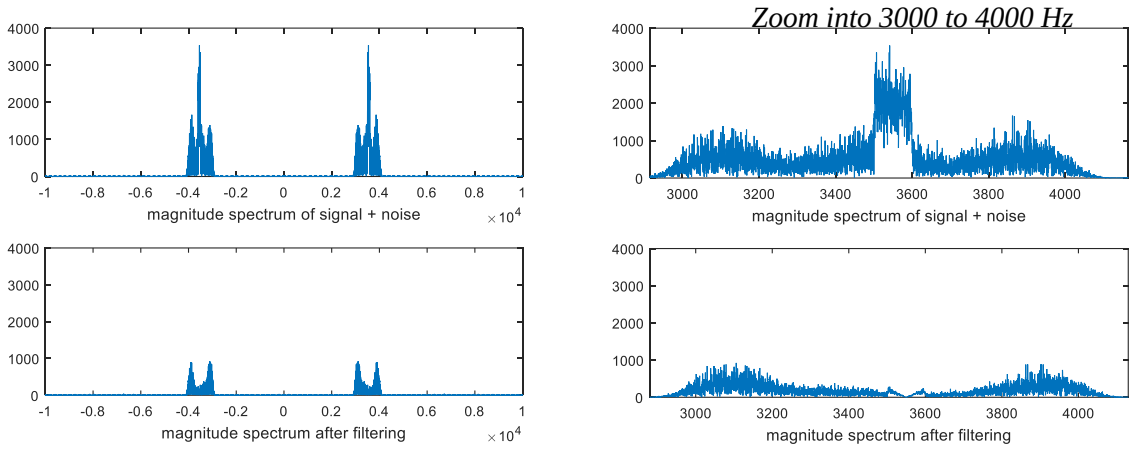
For part (b), the in-band interference passes through the bandpass filter. However, some of the energy in the signal and interference resides outside the 3000-4000 Hz band. The bandpass filter passes a higher percentage of interference energy than signal energy, and the SNR decreases.

```
%% #3.1(a), (b)
% improvesnr.m: using linear filters to improve SNR
clear all; close all; clc;
time=3;                                  % length of time for the simulation
Ts=1/20000;                              % sampling interval for fs = 20000 Hz
b=remez(100, [0 0.29 0.3 0.4 0.41 1], [0 0 1 1 0 0]); % Bandpass filter passing 3000-4000 Hz
t=0: Ts: time-Ts;
% % % chirp noise outside passband for part (a)
% f0 = 1000;
% chirpBW = 100;
% n = cos(2*pi*f0*t + pi*(chirpBW/time)*(t.^2));
% % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% % chirp noise inside passband for parts (b) and (c)
f0 = 3500;
chirpBW = 100;
n = cos(2*pi*f0*t + pi*(chirpBW/time)*(t.^2));
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x=filter(b, 1, 2*randn (1, time/Ts));    % bandlimited signal between 3K and 4K
y=filter(b, 1, x+n);                     % filter the received signal+noise
yx=filter(b, 1, x); yn=filter(b, 1, n);  % filter signal and noise separately
z=yx+yn;                                 % add them
diffzy=max(abs(z-y))                     % and make sure y and z are equal
snrinp=pow(x)/pow(n)                     % SNR at input
snrout=pow(yx)/pow(yn)                   % SNR at output

% check spectra
figure (1), plotspec(n, Ts)
```

```
figure (2), plotspec(x, Ts)
figure (3), plotspec(x+n, Ts)
figure (4), plotspec(y, Ts)

% Here's how the figure was actually drawn
N=length(x);                                % length of the signal x
t=Ts*(1:N);                                 % define time vector
ssf=(-N/2 : N/2-1) / (Ts*N);                % frequency vector
fx=fftshift(fft(x(1:N)+n(1:N)));
figure(5), subplot(2, 1, 1), plot(ssf, abs(fx));
xlabel('magnitude spectrum of signal + noise');
ylim( [0, 4000] );
fy=fftshift(fft(y(1: N)));
subplot(2, 1, 2), plot(ssf, abs(fy))
xlabel('magnitude spectrum after filtering');
ylim( [0, 4000] );
```

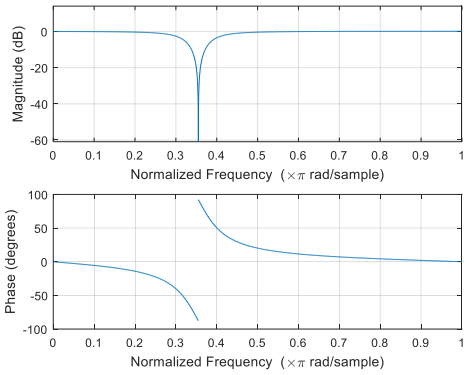
Solution for (c): We apply an IIR notch filter with two poles and two zeros, shown below. (See Lecture Slides 6-11 and 6-22).

```
fs = 1 / Ts;
f0 = 3500+chirpBW/2;                        % chirp is at 3500-3600 Hz, with center at 3550 Hz
% Design IIR biquad notch filter
w0 = 2*pi*f0/fs;                            % notch out chirp noise
r = 0.99;                                   % pole radius
snrnotch_min = snrout;                      % initialize snrnotch_min to conservatively low value
r_min = r;                                  % initialize r_min
N=length(x);                                % length of the signal x
while r>0.85                                % try different notch filters to pick best one
    p0 = r*exp(j*w0);                       % also to prevent removing signal frequency components
    p1 = r*exp(-j*w0);
    z0 = exp(j*w0);
    z1 = exp(-j*w0);
    num = [ 1  (-(z0+z1))  (z0*z1) ];
    den = [ 1  (-(p0+p1))  (p0*p1) ];
    zvec = [ 1  1  1 ];                     % vector of [1 z^(-1) z^(-2)] for z = 1
    gain = (den * zvec') / (num * zvec');   % set gain at w=0 (z=1) to be 1
    num = gain * num;                       % normalize gain, but does not affect SNR result

    % Apply IIR biquad
    wx = filter(num, den, yx);              % filter the output of bandpass filter from (b)
    wn = filter(num, den, yn);
    w = wx + wn;
    snrnotch = pow(wx)/pow(wn);
    if (snrnotch_min<snrnotch)
        snrnotch_min=snrnotch;
        r_min=r;
        t=Ts*(1:N);                                 % define time vector
        ssf=(-N/2 : N/2-1) / (Ts*N);                % frequency vector
        fx=fftshift(fft(yx(1:N)+yn(1:N)));
        figure(6), subplot(2, 1, 1), plot(ssf, abs(fx));
        xlabel('magnitude spectrum of signal + noise');
        ylim( [0, 4000] );
        fy=fftshift(fft(w(1: N)));
        subplot(2, 1, 2), plot(ssf, abs(fy))
        xlabel('magnitude spectrum after filtering');
        ylim( [0, 4000] );
    end
    r=r-0.001;                                  % step size of notch by 0.001
end

snrnotch_min
r_min
```



$SNR_{inp} = 0.9837$ $SNR_{notch\_min} = 54.4634$ $r_{min} = 0.8510$

The above code finds the best notch filter in terms of SNR_{notch} by changing the radius of the poles in the filter, r = [0.85 to 0.99] with Δr = 0.001. The chirp interference has been removed. The frequency response of the notch filter is shown on the right.



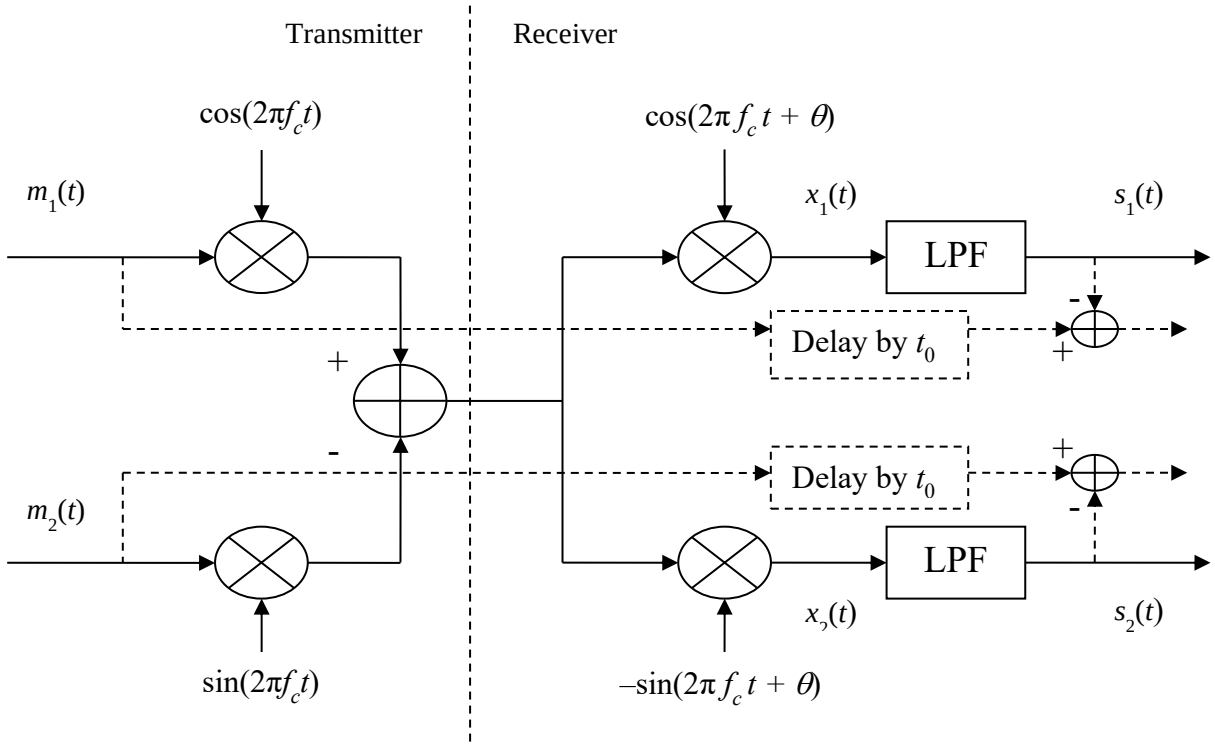
Here's an alternate solution to part (c) using the `iirnotch` command in Matlab:

```
fs = 1 / Ts;
w0 = (3500+chirpBW/2)*2*Ts;              % notch out noise at 3600 Hz
bw = w0*1;                               % bandwidth, preferable 100 Hz
[Num, Den] = iirnotch(w0, bw);
wx = filter(Num, Den, yx);               % filter the output of bandpass filter from (b)
wn = filter(Num, Den, yn);
w = wx + wn;
snrnotch = pow(wx)/pow(wn)
figure(8), freqz(Num, Den,fs);
```

3.2. Quadrature Amplitude Modulation. 27 points.

Johnson, Sethares & Klein, exercise 5.14 on page 92, parts (a) and (b).

Problem 3.2



Dashed lines indicate the steps for additional calculations of signal quality in the epilogue.

Prologue: An analog QAM receiver must perfectly track the carrier frequency and carrier phase to recover the transmitted messages perfectly (same goes for a digital QAM receiver) *assuming an ideal channel*. This problem asks you to explore degradation in the received messages if the carrier frequency has been perfectly tracked but the carrier phase tracking has a slight error in it.

The block diagrams for the QAM transmitter and receiver are given at the top of this page. Note the negative sign on the sine carrier signal in the receiver to match the negated path for the sine carrier signal in the transmitter. Delay t_0 matches the delay through the lowpass filter (LPF).

Solution for (a): Because of the phase error θ, energy in transmitted message signal #1 will leak into received message signal #2, and vice-versa, which is known as cross-interference:

$s_1(t) = \text{LPF}\{\ m_1(t) \cos(2\pi f_c t) \cos(2\pi f_c t + \theta) - m_2(t) \sin(2\pi f_c t) \cos(2\pi f_c t + \theta)\ \}$
$s_1(t) = \text{LPF}\{\ 1/2\ m_1(t)\ (\ \cos(\theta) + \cos(4\pi f_c t + \theta)\ ) - 1/2\ m_2(t)\ (\ \sin(-\theta) + \sin(4\pi f_c t + \theta)\ )\ \}$
$s_1(t) = 1/2 \cos(\theta)\ m_1(t) + 1/2 \sin(\theta)\ m_2(t)$

When θ = 0, i.e. when there is no phase error,

$s_1(t) = 1/2\ m_1(t)$

which recovers the in-phase component up to a scalar gain.

When $\theta = \pi/2$, i.e. when there is severe phase error,

$s_1(t) = 1/2\ m_2(t)$

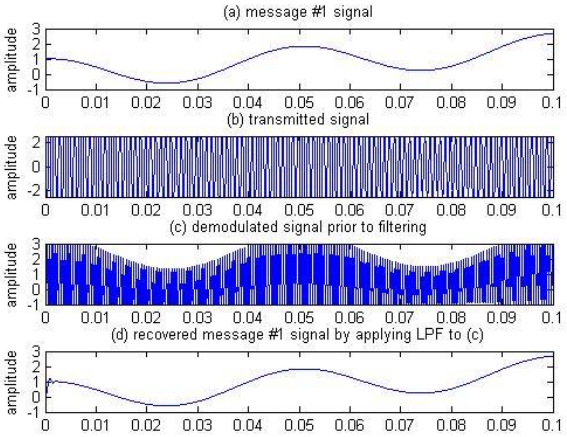
which recovers the wrong message signal.

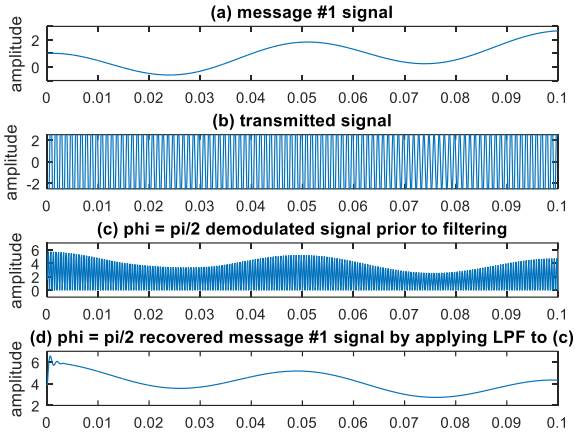
In the above equations, the lowpass filter is assumed to be ideal with no delay. In practice, the input signal experiences delay through the filter, which is called the group delay. The group delay is equal to the negative of the slope of the phase response vs. frequency. For a linear phase FIR filter with N coefficients, the group delay n_0 is a constant equal to $(N\text{-}1)/2$ samples. Please use an odd-length FIR filter so that the group delay through the FIR filter is an integer.

For the Matlab code, please use two different signals for the baseband message signals $m_1(t)$ and $m_2(t)$. Also, it is easier to use the same lowpass filter in each branch.

```
% QAM.m which is a modified version of
% AM.m suppressed carrier AM with freq and phase offset
% from Johnson, Sethares and Klein, Software Receiver Design

% Define sampling rate, sampling time, and time duration
fs=10000;                           % sampling rate
time=0.3; Ts=1/fs;                  % sampling interval & time
t=Ts:Ts:time; lent=length(t);       % define a time vector

% Transmitter
upramp=(5/lent)*(1:lent);
fm=20;
m1=upramp+cos(2*pi*fm*t);           % create message m1(t)

downramp=(5/lent)*(lent:-1:1);
fm=20;
m2=downramp+cos(2*pi*fm*t);         % create message m2(t)

fc=1000;
c=cos(2*pi*fc*t);                   % cosine carrier at freq fc
s=sin(2*pi*fc*t);                   % sine carrier at freq fc
v=m1.*c - m2.*s;                    % modulate

% Receiver
fbe=[0 0.1 0.2 1]; damps=[1 1 0 0]; % fpass=500 Hz and fstop=1000 Hz
fl=100; b=firpm(fl,fbe,damps);      % LPF with (f1+1) coefficients
n0=fl/2;                            % group delay through LPF

gam=0; phi=0;                       % freq & phase offset
cr=cos(2*pi*(fc+gam)*t+phi);        % create cosine for demod
x1=v.*cr;                           % demod received signal
s1=2*conv(b,x1);                    % LPF the demodulated signal
s1short=s1(n0+1:n0+lent);           % remove first and last n0 samples
sr=-sin(2*pi*(fc+gam)*t+phi);       % create negated sine for demod
x2=v.*sr;                           % demod received signal
s2=2*conv(b,x2);                    % LPF the demodulated signal
s2short=s2(n0+1:n0+lent);           % remove first and last n0 samples

% used to plot figure
subplot(4,1,1), plot(t,m1)
axis([0,0.1, -1,3])
```

```
ylabel('amplitude'); title('(a) message #1 signal');
subplot(4,1,2), plot(t,v)
axis([0,0.1, -2.5,2.5])
ylabel('amplitude'); title('(b) transmitted signal');
subplot(4,1,3), plot(t,x1)
axis([0,0.1, -1,3])
ylabel('amplitude');
title('(c) demodulated signal prior to filtering');
subplot(4,1,4), plot(t,s1short)
axis([0,0.1, -1,3])
ylabel('amplitude');
title('(d) recovered message #1 signal by applying LPF to (c)');
```





Connections to lecture. We had discussed cosine modulation and demodulation on lecture slides 1-6 through 1-7 for the two-sided case. Consider a two-sided baseband signal $x_1(t)$ with bandwidth ω_1 modulated by $\cos(\omega_c t)$ which gives $y(t) = x_1(t) \cos(\omega_c t)$ as a two-sided signal. The spectrum $Y(\omega)$ is bandpass over ω in $[\omega_c - \omega_1, \omega_c + \omega_1]$ and $[-\omega_c - \omega_1, -\omega_c + \omega_1]$ with bandwidth $2\omega_1$ as shown on both slides. For demodulation back to baseband, we'll multiply $y(t)$ by $\cos(\omega_c t)$ and apply a lowpass filter. In modulating $y(t)$ by $\cos(\omega_c t)$, we'll get a combination of a baseband component for ω in $[-\omega_1, \omega_1]$ and a bandpass component for ω in $[2\omega_c - \omega_1, 2\omega_c + \omega_1]$ and $[-2\omega_c - \omega_1, -2\omega_c + \omega_1]$ as shown on lecture slide 1-7. Applying a lowpass filter with passband frequency of ω_1 will extract the baseband component and attentuate/reduce the bandpass component. Here are several lowpass filtering approaches. Lecture slides 1-6 and 1-7 apply an ideal IIR filter whose frequency response is a rectangular pulse for ω in $[-\omega_1, \omega_1]$ and whose impulse response is a two-sided sinc. Not implementable, but useful as a starting point. A practical demodulating filter could be an FIR or IIR filter with a stopband frequency between $1.1\omega_1$ and $2\omega_c - \omega_1$.

Message bandwidth. The message bandwidth is not explicitly given. By using plotspec(m1, Ts), the magnitude of the frequency content of message #1 contains one large DC component plus a smaller peak at 20 Hz plus slowly decaying magnitude values that follow a $1/f$ shape. The carrier frequency is 1000 Hz. The demodulating filter is a linear phase FIR filter designed using the Parks-McCellan algorithm (`firpm` command) using a passband frequency of 500 Hz and stopband frequency of 1000 Hz. From the demodulating filter specifications, one can infer the message bandwidth to be 500 Hz, and the stopband frequency is twice the message bandwidth.

Solution for (b): A frequency error γ can also cause cross-interference:

$s_1(t)$ = LPF{ $m_1(t) \cos(2\pi f_c t) \cos(2\pi (f_c + \gamma) t) - m_2(t) \sin(2\pi f_c t) \cos(2\pi (f_c + \gamma)t)$ }
$s_1(t)$ = LPF{ $1/2\ m_1(t)$ ($\cos(-2\pi\gamma t) + \cos(2\pi (2f_c + \gamma)t) - 1/2\ m_2(t) (\sin(2\pi (2f_c + \gamma)t) - \sin(-2\pi \gamma t))$}
$s_1(t) = 1/2 \cos(2\pi \gamma t)\ m_1(t) - 1/2 \sin(2\pi \gamma t)\ m_2(t)$

When $\gamma = 0$,

$s_1(t) = 1/2\ m_1(t)$, which recovers the in-phase component up to a scalar gain.

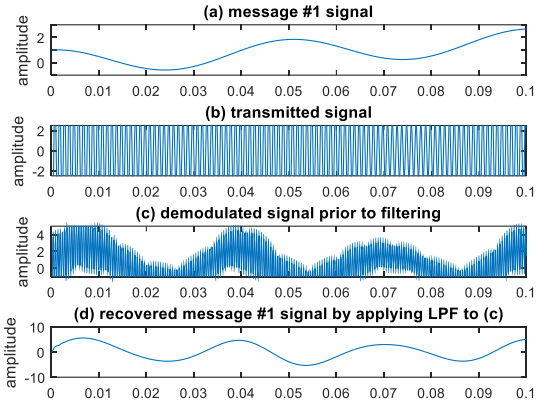
When $\gamma = 1/4$,

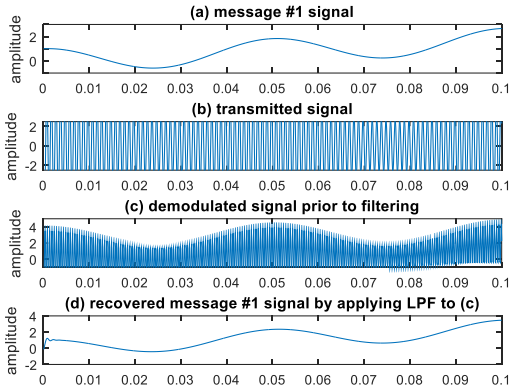
$s_1(t) = 1/2 \cos((\pi/2) t)\ m_1(t) - 1/2 \sin((\pi/2) t)\ m_2(t)$, which recovers the wrong message signal.

As in part (a), the lowpass filter is assumed to be ideal with no delay.

The same code used in part (a) can be used in part (b). To investigate the effect of a frequency offset, set `phi=0;` and change `gam=0;` to values other than 0 (e.g. `gam=pi/2`).

For example, setting gamma = 10π and (1/10)π, respectively, results in the figures below:





Epilogue: Compare the signal-to-noise ratio (SNR) for phase offset and frequency offset. The signal of interest is the message signal $m_1(t)$. The noise in $s_1(t)$ is computed as $s_1(t) - m_1(t)$.

Note that this part is not required to complete the homework.

$$SNR = SignalPower / NoisePower = Power\ of\ m_1(t) / Power\ of\ (s_1(t) - m_1(t))$$

A gain of 2 should be applied to $s_1(t)$ before calculating the SNR. The total power of a discrete-time signal $v[n]$ is the sum of $v^2[n]$ for all n being observed. To calculate SNR, the following line of MATLAB code is appended to the code used in parts (a) and (b):

```
snr = sum(m1.^2) / sum((s1short-m1).^2)
```

As can be seen from these calculations, SNR decreases when there is phase error and/or frequency error. Also, note that even without phase error or frequency error, $s_1(t)$ does not exactly equal $m_1(t)$.

γ	θ	SNR
0	0	2803
0	π/2	1.062
0.25	0	53.79
0.25	π/2	0.6580
10π	0	0.5360

3.3. Infinite Impulse Response (IIR) Filter Design for Treatment of Tinnitus Loudness.

This problem asks you to design a discrete-time digital IIR filter for the treatment of tinnitus loudness. *This problem is a sequel to homework problem 2.3.*

Tinnitus, a.k.a. "ringing of the ears", is a symptom due to an underlying condition in the auditory system. It could have resulted from injury, infection, or other causes. People with tinnitus hear a tone, clicking, hiss, roaring or buzzing when no external sound is present [1][2]. The tinnitus sound could be at low, medium or high audible frequencies, and may occur in one ear or both ears. The tinnitus sound might be temporary or chronic. Those suffering from chronic tinnitus would hear the same sound in the same frequency range each time. The tinnitus sound has a principal frequency that can be determined through auditory testing. Hearing sound that contains the principal frequency and frequencies close to the principal frequency is particularly painful.

This problem asks you to design a discrete-time filter to alleviate the loudness of tinnitus:

> "Maladaptive auditory cortex reorganization may contribute to the generation and maintenance of tinnitus. Because cortical organization can be modified by behavioral training, we attempted to reduce tinnitus loudness by exposing chronic tinnitus patients to self-chosen, enjoyable music, which was modified ("notched") to contain no energy in the frequency range surrounding the individual tinnitus frequency. After 12 months of regular listening, the target patient group (n = 8) showed significantly reduced subjective tinnitus loudness and concomitantly exhibited reduced evoked activity in auditory cortex areas corresponding to the tinnitus frequency compared to patients who had received an analogous placebo notched music treatment (n = 8). These findings indicate that tinnitus loudness can be significantly diminished by an enjoyable, low-cost, custom-tailored notched music treatment, potentially via reversing maladaptive auditory cortex reorganization." [3]

The proposed treatment for tinnitus [3] alters participants' favorite music to remove an octave of frequencies around the tinnitus frequency f_c. An octave means a range of frequencies from f_1 to 2 f_1. Since f_c would be in the middle of the octave, $f_1 = (2/3)\, f_c$. After 12 months of listening to the filtered music, patients reported lessening of tinnitus loudness.

A good rule of thumb in filter design is that the transition region is about 10% of the passband width. In this case, the passband width is (2/3) f_c.

Here are the bandstop filter specifications for your design:

- For frequencies 0 Hz to 0.6 f_c, the passband ripple should be no greater than 1 dB.
- For frequencies (2/3) f_c to (4/3) f_c, the stopband attenuation should be at least 80 dB.
- For frequencies above 1.4 f_c, the passband ripple should be no greater than 1 dB

Please use a tinnitus frequency f_c of 3000 Hz and a sampling rate f_s of 44100 Hz.

(a) Design IIR filters using the Butterworth, Chebyshev type I, Chebyshev type II, and Elliptic design methods. For each design method, find the filter of smallest order to meet the specifications. The filter order is the number of poles. Turn in plots of the magnitude and phase responses for each IIR filter you have designed to meet the specifications. Describe the passband and stopband response for each filter design as either monotonic or rippling. ***24 points.***

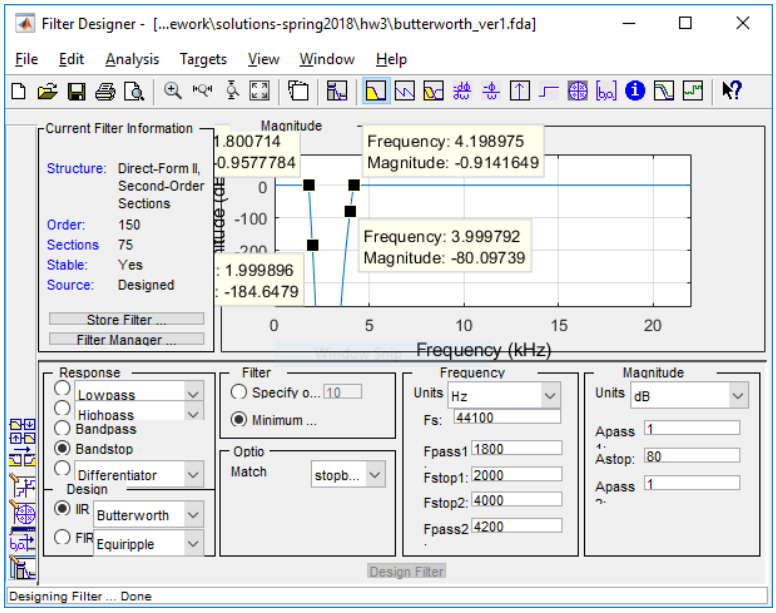
Hint: Please check the filter design and implementation to ensure that it meets specifications:

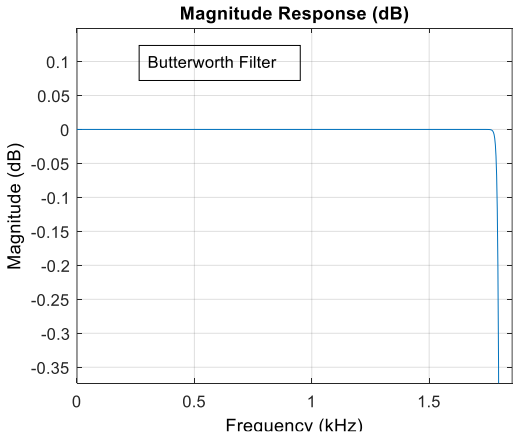
- Increase the plotting resolution for magnitude plots vs. frequency for the filter by going to the Analysis menu ... Analysis Parameters ... and setting the number of points to 65536.
- Display the magnitude response in filterDesigner by choosing "Magnitude Response" under the "Analysis" menu or the appropriate icon shortcut.
- Zoom into the passbands and stopband by using the zoom tool, which is available as the icon of the magnifying glass with a plus sign in the middle of it.
- Hit CONTROL-Z to undo the most recent zoom.

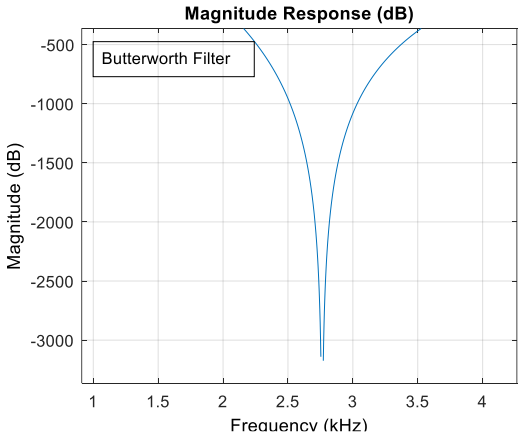
Solution for part (a) :

Butterworth filter of the following specifications meets the filter requirement:

Filter order = 150
Fs = 44100 Hz
Fpass1 = 1800 Hz
Fstop1 = 2000 Hz
Fstop2 = 4000 Hz
Fpass2 = 4200 Hz
Apass1 = 1 dB
Astop = 80 dB
Apass2 = 1 dB
Match Option: Stopband

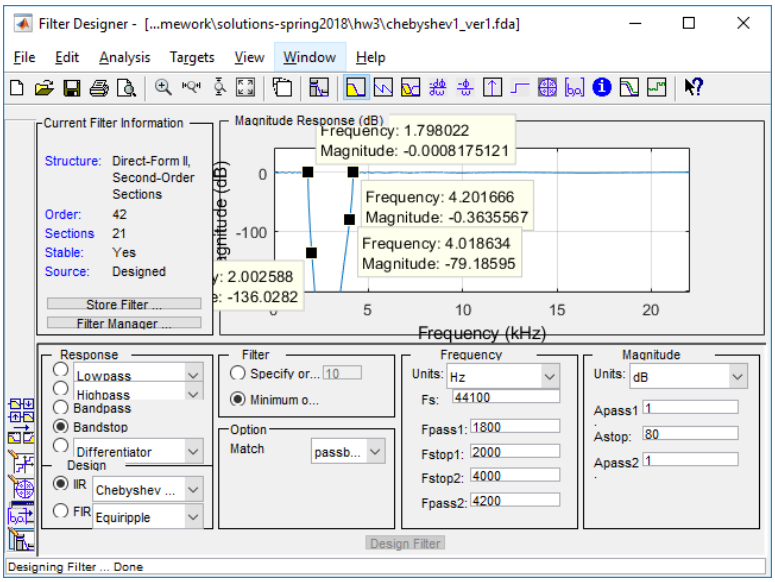


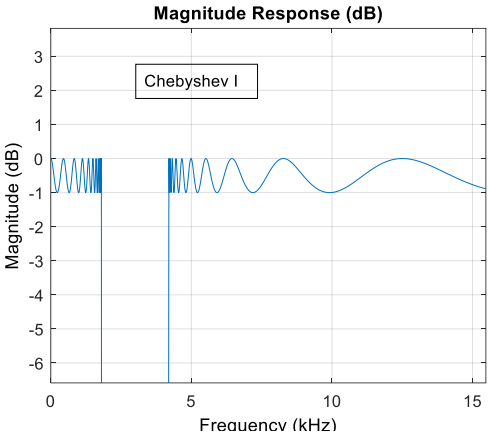


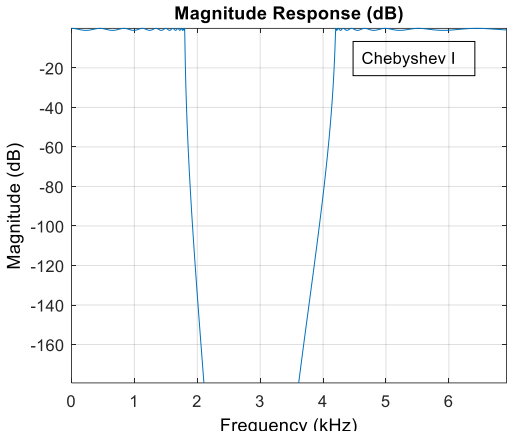


Chebyshev Type 1 filter of the following specifications meets the filter requirement:

Filter order = 42
Fs = 44100 Hz
Fpass1 = 1800 Hz
Fstop1 = 2000 Hz
Fstop2 = 4000 Hz
Fpass2 = 4200 Hz
Apass1 = 1 dB
Astop = 80 dB
Apass2 = 1 dB
Match Option: Passband



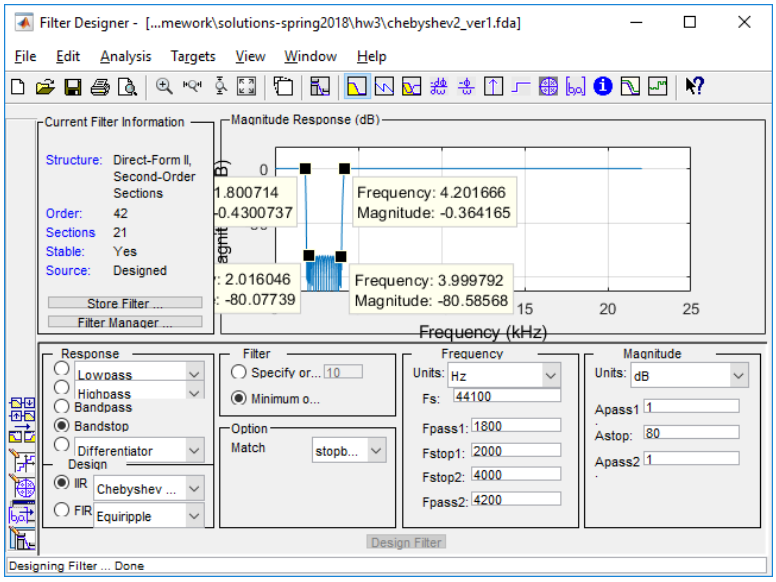


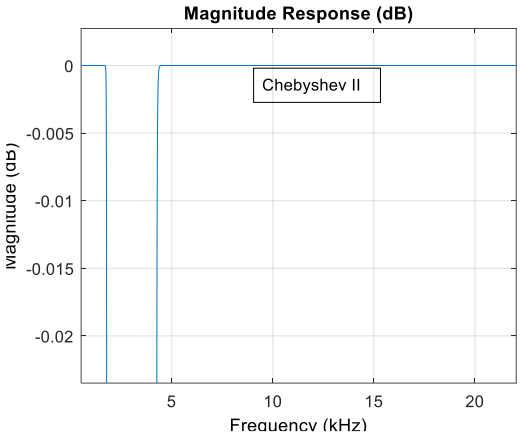


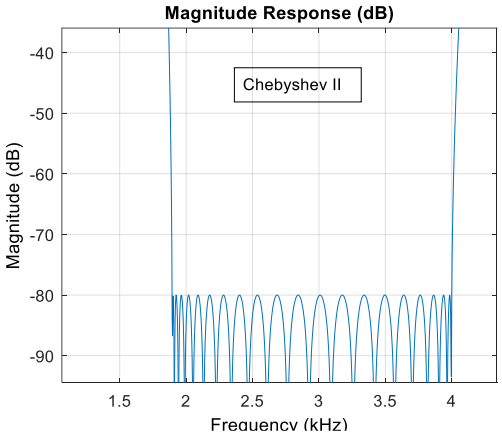
Chebyshev Type 2 filter of the following specifications meets the filter requirement:

Filter order = 42
Fs = 44100 Hz
Fpass1 = 1800 Hz
Fstop1 = 2000 Hz
Fstop2 = 4000 Hz
Fpass2 = 4200 Hz
Apass1 = 1 dB
Astop = 80 dB
Apass2 = 1 dB

Match Option: Stopband



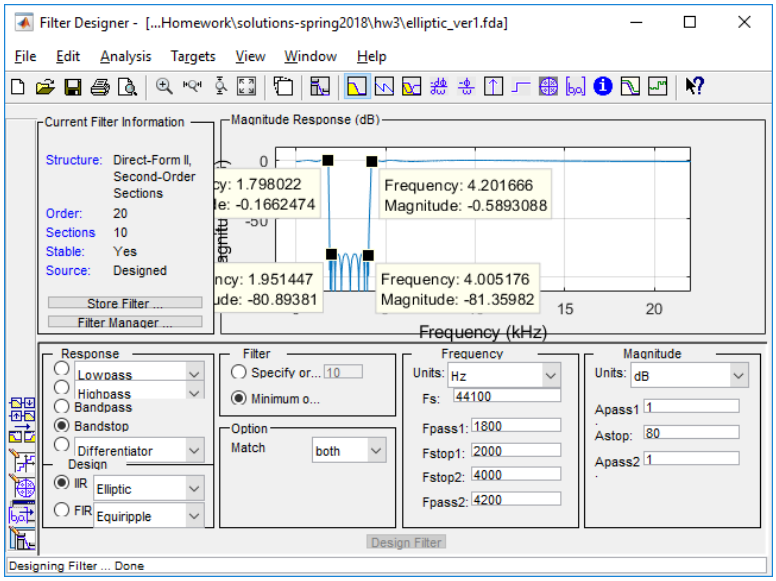


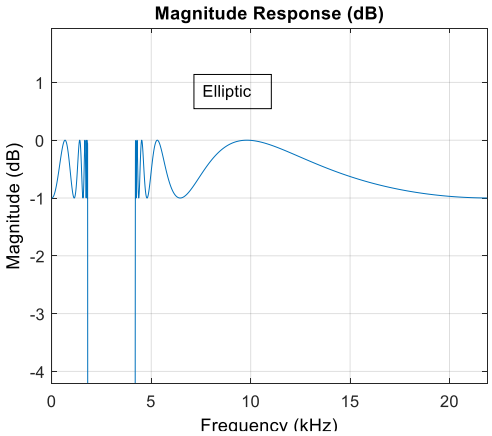


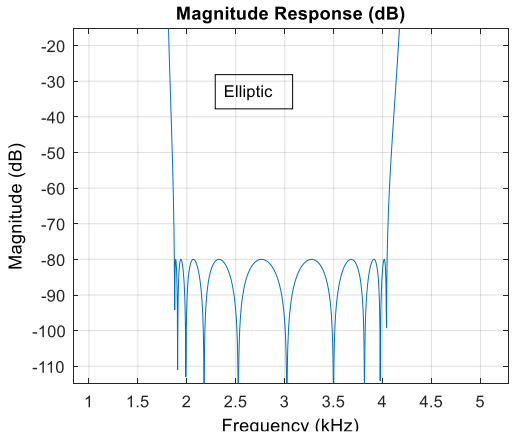
Elliptic filter of the following specifications meets the filter requirement:

Filter order = 20
Fs = 44100 Hz
Fpass1 = 1800 Hz
Fstop1 = 2000 Hz
Fstop2 = 4000 Hz
Fpass2 = 4200 Hz
Apass1 = 1 dB
Astop = 80 dB
Apass2 = 1 dB

Match Option: Both







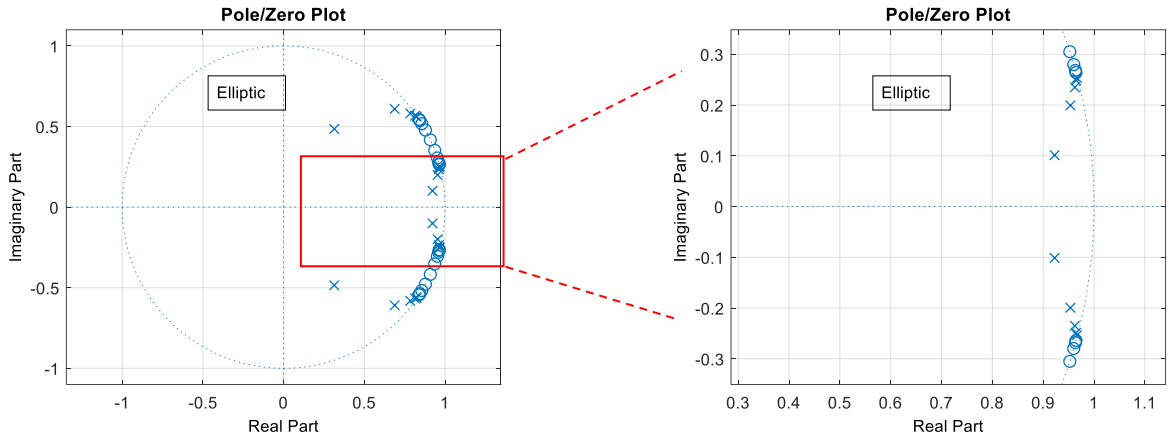
Design Method	Passband	Stopband
Butterworth	Monotonic	Monotonic
Chebyshev Type I	Rippling	Monotonic
Chebyshev Type II	Monotonic	Rippling
Elliptic	Rippling	Rippling

(b) List the filter orders required for filters to meet the specification. Which IIR filter family gives the lowest filter order? Plot the pole-zero diagram for the IIR filter with shortest filter order you designed in (a). Because the poles and zeros are close to the unit circle and separated in angle, poles indicate the passband and zeros indicate the stopband. Submit the feedforward and feedback coefficients for the filter with minimum order.

Solution for part (b)

Elliptic design produces the lowest-order IIR filter. Its pole-zero plot is below. Filter has 10 pairs of conjugate symmetric poles and 10 pairs of conjugate symmetric zeros.

Filter Type	Order
Butterworth	150
Chebyshev Type I	42
Chebyshev Type II	42
Elliptic	20



Here are the filter coefficients (expressed as biquads). All numerator coefficients **(b0, b1, b2)** are feedforward coefficients and all denominator coefficients **(a0, a1, a2)** are feedback coefficients. There are **k+1** Gain values for **k** biquads.

	b0	b1	b2	a0	a1	a2
1st Biquad	1	-1.687551	1	1	-1.627987	0.983799
2nd Biquad	1	-1.926695	1	1	-1.929910	0.992896
3rd Biquad	1	-1.711770	1	1	-1.568735	0.952980
4th Biquad	1	-1.920147	1	1	-1.924376	0.980993
5th Biquad	1	-1.756662	1	1	-1.375467	0.844027
6th Biquad	1	-1.904650	1	1	-1.907066	0.948963
7th Biquad	1	-1.817489	1	1	-1.844643	0.860928
8th Biquad	1	-1.871898	1	1	-0.627776	0.333779
9th Biquad	1	-1.677853	1	1	-1.648977	0.995936
10th Biquad	1	-1.929047	1	1	-1.932847	0.998169

	Gain
1st	1.018323
2nd	1.018323
3th	1.241994
4th	1.241994
5th	1.704523
6th	1.704523
7th	3.852219
8th	3.852219
9th	0.070229
10th	0.070229
11th	1.000000

Each biquad consists of a conjugate zero pair and a conjugate pole pair. Because all zeros are on the unit circle, each b2 coefficient is one. The transfer function for each biquad has the form

$$H(z) = C\,\frac{1 + b_1 z^{-1} + z^{-2}}{1 - a_1 z^{-1} - a_2 z^{-2}}$$

This would correspond to the difference equation that takes four multiplications and four additions:

$$y[n] = a_1\, y[n-1] + a_2\, y[n-2] + C\,(x[n] + b_1\, x[n-1] + x[n-2])$$

After the cascade of the 10 biquads, the multiplication by $C_{11} = 1$ can be removed.

(c) Use the Elliptic filter you designed in (a) for this part. Assuming that the input data samples and the IIR filter coefficients are stored in single precision IEEE floating point format,

1. How many instruction cycles on the TMS320C6700 DSP family would it take to compute one output value for each input value if the IIR filter routine were handcoded in assembly for optimal performance?
2. How much storage in bytes would it take to store the IIR coefficients and the circular buffer for the current and past inputs and the circular buffer for past outputs?

Solution for (c):

IIR filters of orders 12^{th} – 24^{th} would generally need to be implemented as a cascade of biquads and even then, the implementation might be very difficult to keep BIBO stable. IIR filters above 24^{th} order are extraordinarily difficult to maintain BIBO stability after implementation.

We'll assume the IIR filter is implemented as a cascade of biquads to match the design procedure used in part (a). Using a cascade of biquads limits the loss of precision that results from converting from factored form in poles and zeros to unfactored form because the polynomial expansion only occurs for each biquad. A filter of order *M* would have *M*/2 biquads for even *M*, and (*M*-1)/2 biquads and a first-order section for odd *M*. In this case, we have 20^{th} order filter arranged as a cascade of 10 biquads. Each biquad, as mentioned in part (b), requires four multiplication and four addition operations to compute each outputs sample.

Run-time computational complexity in multiplication-addition operations. There are 10 biquads and each biquad requires four multiplication-addition operations, as explained next, for a total of 40 multiplication-addition operations. ***This is twice the filter order.***

Run-time complexity in terms of TMS320C6700 DSP instructional cycles. In the course reader, Appendix N entitled "Tapped Delay Line on C6700 DSP" outlines an assembly language implementation for an FIR filter with 32-bit IEEE floating-point coefficients, input data and output data. ***For a tapped delay line with N coefficients, the code takes N + 28 instruction cycles to compute one output sample, not including updating the circular buffer of current and previous input values.*** For the DSP core, we need to load data from local on-chip memory into registers tto perform computations. The two data buses are each 32 bits wide. The C6700 DSP core has two data units, one per data bus, and each load or store operation has a throughput of one instruction cycle for each 32-value. Hence, we can read only one filter coefficient and one data value per instruction cycle. The 32-bit multiplication instruction (MPYSP) has a throughput of one cycle.

An FIR filter computes the difference equation

$$b_0\, x[n] + b_1\, x[n-1] + b_2\, x[n-2] + \cdots$$

and the biquad computes a difference equation in a similar form

$$y[n] = a_1\, y[n-1] + a_2\, y[n-2] + b_0\, x[n] + \; b_1\, x[n-1] + b_2\, x[n-2]$$

which takes five multiplications and four additions per output sample. The term b_0 is the biquad gain applied at the input. We'll replace b_0 with C and factor it out of the feedforward terms:

$$y[n] = a_1\, y[n-1] + a_2\, y[n-2] + C\left(x[n] + \overline{b}_1\, x[n-1] + \overline{b}_2\, x[n-2]\right)$$

which still requires five multiplications and four additions per output sample.

In the listing of feedforward coefficients above, two of the feedforward coefficients, b_0 and b_2, were equal to 1. This occurs when the zero locations are complex-valued and on the unit circle. In that case, which is common, only four multiplications per output sample is needed:

$$y[n] = a_1\, y[n-1] + a_2\, y[n-2] + C\,\left(x[n] + \overline{b}_1\, x[n-1] + x[n-2]\right)$$

When the zero locations are real-valued and distinct, i.e. +1 and -1, only three multiplications four additions per output sample is needed:

$$y[n] = a_1\, y[n-1] + a_2\, y[n-2] + C\,\left(x[n] - x[n-2]\right)$$

In the most general case, computing the difference equation for a biquad requires 5 multiplications and 4 additions. The implementation complexity on the C6700 DSP is the same as a single tapped delay line with five coefficients, i.e. 33 instructions according to the above discussion. (A first-order section would need 3 multiplications and 2 additions per output sample, which would have the same complexity as a tapped delay line with three coefficients or 31 instructions.) If the computation for biquad #2 waits until the computation of biquad #1 is finished, then the total number of cycles is obtained by adding up the instruction cycles for all the stages. This is because unless we obtain the value of the current stage, we cannot evaluate the next stage. In our case, we have ten biquads, which would require 10 * 33 = **330 instruction cycles.**

Optional: A fast implementation can be found by noticing that the difference equation in biquad #2 relies on the current input value (to be calculated by biquad #1), two previous inputs (already stored in the biquad) and two previous output values (already stored in the biquad). Hence, we could start the calculation of the four of the five terms in the difference equation for biquad #2 before biquad #1 is finished. That should save us 16 instruction cycles in each biquad. This implementation would require 33 instruction cycles for the first cycle and 17 cycles for each biquad thereafter due to the overlapping calculations, for a total of 33 + 17 * 9 = **186 instruction cycles.**

Optional: A pipelined implementation of a cascade of biquads results from inserting a delay of one sample between each pair of biquads. This insertion of a delay breaks the dependence of biquad *n* on the output of biquad *n*-1, thereby allowing biquad *n* and biquad *n*-1 to execute in an overlapped fashion. That is, the computation of biquad *n*-1 can occur in parallel with the reads in biquad *n*. Inserting delays changes the phase (by increasing the group delay) but does not change the magnitude response. The benefit is to reduce the number of C6000 cycles to be the same as direct form with two tapped delay lines, i.e. (order + 1 + order + 28) = **69 instruction cycles**.

Filter Coefficient Storage: Each biquad would require storage of 2 numerator and 2 denominator coefficients. (Even though a biquad has six coefficients, a_0 and b_0 are always of value "1".) Each biquad has a constant gain, and that constant gain is chosen to help keep the implementation remain BIBO stable. There are 10 biquads. Hence, the storage is 5 * 10 = 50 coefficients, which would take 50 * 4 bytes = 200 bytes. If the IIR filter were implemented as a single section, it would have 21 numerator coefficients and 20 denominator coefficients (one of the denominator coefficients is always 1). The required storage is (21 + 20) * 4 = **164 bytes**.

Previous Values Storage: We need to store 20 previous input samples (ignoring the current input) and 20 previous outputs. For the two buffers, we would need 2 × 20 × 4 = **160 bytes.** To utilize the modulo addressing mode in the C6000 instruction set architecture, the circular buffer size should be a power of 2, which means that its size should be at least 32 words. So, for the two buffers, we'll need 2 × 32 × 4 = **256 bytes**.

(d) How would you advocate using either an FIR filter or IIR filter for this application? In your answer, please include the design method for which you are advocating

Solution for (d): Based on the results of this problem and homework problem 2.3, we notice that the best design algorithms in minimizing filter length are the Parks-McClellan (Remez) algorithm for *linear phase* FIR filters and the elliptic design algorithm for IIR filters.

In **terms of computation**, the 564th-order FIR filter designed by the Parks-McClellan algorithm would take 565 multiplication-accumulation operations per output sample, whereas the elliptic IIR filter as a cascade of biquads would take 40 multiplication-operations as explained above, which is nearly 14x more efficient. For a C6700 implementation, the **FIR filter** would take 565 + 28 = **593 instructions**. Implementing a **direct form IIR structure** as a single tapped delay line on the input values and a separate single tapped delay line on the output values would take (order + 1 + 28) + (order + 28) or **97 instruction cycles**. As described in part (c), a **cascade of biquads** would take **330 instruction cycles**; an efficient implementation of the cascade would take 186 instruction cycles; and a pipelined implementation of the cascade would take 69 instruction cycles.

In **terms of stability**, an **FIR filter** is always **stable**, even under implementation. The **choice of IIR filter structure will affect stability**. See the epilog for more information.

In **terms of phase response**, an **FIR filter** designed by the Parks-McClellan algorithm has **linear phase over all frequencies**. An **IIR filter** has **non-linear phase**, although its phase response is approximately linear over parts of the passband. IIR filters have lower run-time complexity than FIR filters we can design IIR filters with much lower orders vs. FIR filters, e.g. 20th-order IIR vs. 564-order FIR filters in this case. FIR filters have two major disadvantages because of their high orders: (1) they require more computation and hence use more power, and (2) the long impulse response of the filter introduces delay that may not be desirable for real-time tinnitus treatment. A group delay of 282 samples at a sampling rate of 44100 Hz means a delay of 6.39 ms.

Epilog: Filtering Signals.

There are three steps to design and implement a discrete-time filter:

1. design the filter parameters according to desired goal
2. convert the design to the selected filter structure
3. realize the difference equation(s) for the filter structure to process the input signal to produce the output signal.

FIR filter. We can design the filter parameters by (a) manually placing zeros, or (b) using an automated design algorithm such as Parks-McClellan or Kaiser windowing, or (c) using a pre-designed filter such as an averaging filter or differencing filter and choose the order (length). The most common filter structure is the tapped delay line structure that directly implements convolution. For a causal implementation of the filter with input signal x[n] and output signal y[n], the difference equation is in terms of the current and previous input values, i.e.

$$y[n] = h[0]\,x[n] + h[1]\,x[n-1] + \ldots + h[N-1]\,x[n-(L-1)]$$

for $n \geq 0$ where L is the number of FIR filter coefficients and h[n] is the impulse response. As a necessary condition for linear and time-invariance (LTI) to hold, the initial conditions { x[-1], x[-2], ..., x[-(L-1)] } must be zero. The order of the filter would be N which is L - 1.

IIR filter. We can design the filter parameters by (a) manually placing poles and zeros, or (b) using an automated design algorithm such as elliptic design, or (c) using a pre-designed filter such as a second-order bandpass filter used in homework 1.1(d). An Nth order filter means there are N poles; there are N zeros although some might be zero-valued. The most common filter

structure is to use a cascade of biquads for even-order filters or a cascade of biquads plus a first-order section. One heuristic is order the biquads in ascending order of quality factors from input to output. See the "best mapping" below that uses an exhaustive search. For a causal filter implementation with input signal x[n] and output signal y[n] as a single section, the difference equation is in terms of previous output values as well as current and previous input values, i.e.

$$y[n] = a[1]\, y[n-1] + ... + a[N]\, y[n - N] + b[0]\, x[n] + b[1]\, x[n-1] + ... + b[N]\, x[n - N]$$

for n ≥ where { a[0], a[1], ..., a[N] } are the feedback coefficients with a[0] = 1 and { b[0], b[1], ..., b[N] } are the feedforward coefficients. As a necessary condition for LTI to hold, the initial conditions { y[-1], y[-2], ..., y[-N] } and { x[-1], x[-2], ..., x[-N] } must be zero.

Quantization error. For FIR and IIR filters, implementation of the difference equation in step 3 will be in particular numeric format, e.g. 16-bit integer or 32-bit IEEE floating point. The result of every multiplication and addition in the difference equation will be truncated or rounded to the numeric format, which will lead to quantization error in the output signal amplitude values. This quantization error in calculating the output values is in addition to any quantization errors that would have occured in step 2 in determining the filter coefficients. For an IIR filter, the quantization error for output value y[n] is propagated to future output values in an IIR filter, whereas this is not the case for an FIR filter. When we design an IIR in terms of poles and zeros, we expanding the factored form of the transfer function into unfactored form, i.e. a ratio of two polynomials, to obtain the feedback and feedforward coefficients. This can lead to perturbations of the pole and zero locations, which can sometimes lead to the filter not meeting the desired goal and even bounded-input bounded-output (BIBO) instability.

Best mapping of poles and zeros to cascaded form. We have a cascade of B biquads for an Nth-order IIR filter, where B = N/2 if N is even and B = (N-1)/2 if N is odd. To find the best mapping of poles and zeros to cascaded form, we'd have to simulate all possible mappings to capture the quantization effects mentioned above. We have B pairs of conjugate pole pairs, and B pairs of zeros. All the complex-valued zeros are in conjugate pairs.

For an exhaustive search of all possible mappings, we'll take one of the B conjugate pairs of poles and place it in the first biquad in the cascade. We'll then select one of the remaining B-1 conjugate pairs of poles for the second biquad, and so forth. This has B! possibilities. In addition, for the first biquad, there are B possible choices of zero pairs, B-1 choices for the second biquad, etc. So, in total, we would have B! B! possibilities. This would mean 1 possibility when B = 1, and 4 for B = 2, and 36 for B = 3, and 567 for B = 4; 14440 for B = 5, etc. We can simulate each mapping in parallel for speed. Each simulation would input test signals that have all discrete-time frequencies in them (e.g. a chirp signal, Gaussian noise, and an impulse) and compare the output spectrum to the ideal output spectrum. The filter with the closest match would be chosen.

We can reduce the number of mappings to be simulated. We'll proceed with the mapping of the conjugate pole pairs as before, which has B! possibilities. For each biquad, we'll chose the zero pair closest in distance to the pole pair. This reduces the number simulations from B! B! to B!.

References

[1] R. A. Levine and Y. Oron, "Tinnitus", *Handbook of Clinical Neurology*, vol. 129, pp. 409–431, 2015. *doi*:*10.1016/B978-0-444-62630-1.00023-8*.
[2] *"Tinnitus"*. December 16, 2016. Retrieved February 17, 2017.
[3] H. Okamoto, H. Stracke, W. Stoll and C. Pantev, "Listening to tailor-made notched music reduces tinnitus loudness and tinnitus-related auditory cortex activity", *Proceedings US National Academy of Sciences*, vol. 17, no. 3, pp. 1207-1210, 2010.