Homework #3 Solutions

3.1. Using Finite Impulse Response Filtering to Improve Signal Quality. 18 points.

Johnson, Sethares & Klein, exercise 4.21, on page 78.
Suppose that the noise in improvesnr.m is replaced with narrowband noise (as discussed in Section 4.1.3). Investigate the improvements in SNR
a. when the narrowband interference occurs outside the 3000 to 4000 Hz passband.
b. when the narrowband interference occurs inside the 3000 to 4000 Hz passband.

Solution introduction: This problem explores limitations in improving quality of a signal that has been corrupted by narrowband interference. Narrowband interference can come from an oscillator or from a harmonic of an oscillator frequency. Inside a laptop, electromagnetic interference from the LCD pixel clock circuitry and its first 100 harmonics are significant sources of interference; e.g., for a 65 MHz LCD pixel clock, one of the harmonics clobbers one of the channels in an IEEE 802.11g wireless LAN system. On indoor power lines, the power electronics for electronically-ballasted compact fluorescent lights generate narrowband interferers in the 20-80 kHz band.

In part (a), narrowband interference occupies different frequencies from those of the signal of interest. In part (b), narrowband interference falls within the range of frequencies of the signal of interest. The signal of interest occupies frequencies 3000-4000 Hz. We will model narrowband interference using a cosine, with frequency of 1000 Hz for part (a) and 3500 Hz for part (b).

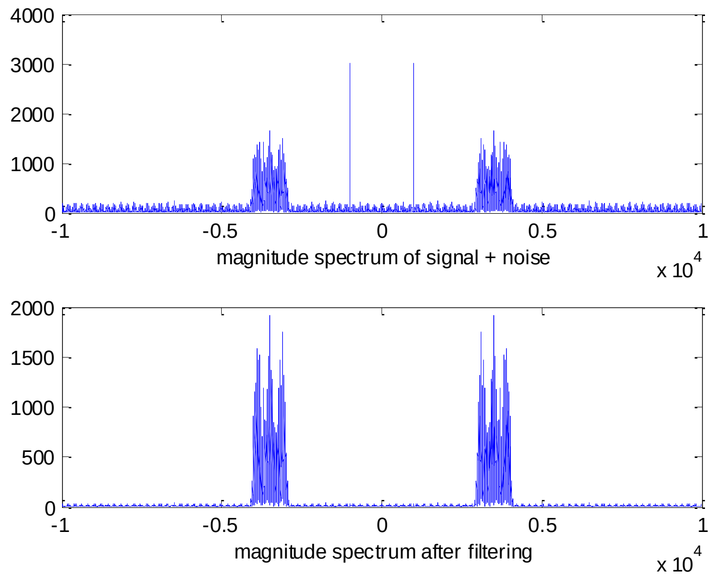
We design a bandpass finite impulse response (FIR) filter with passband 3000-4000 Hz using the Parks-McClellan algorithm. Hence, the FIR filter will have ripples in the passband and stopband of the magnitude response. In the stopband, the magnitude response will vary with frequency. When narrowband interference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

This problem uses a signal-to-noise ratio (SNR) for the measure of signal quality. SNR assumes that signal amplitudes and noise amplitudes are in theory independent random variables and in practice uncorrelated random variables. We will discuss more about these assumptions in class after midterm #1.

```
% improvesnr.m: using linear filters to improve SNR
clearall; close all;
time=3; Ts=1/20000;                        % length of time and sampling interval
b=remez (100, [0 0.29 0.3 0.4 0.41 1], [0 0 1 1 0 0]); % BP filter
t=0: Ts: time-Ts;
% n=0.25*cos (2*pi*1000*t);                % narrowband noise outside passband
n=0.1*cos (2*pi*3500*t);                   % narrowband noise inside passband
x=filter (b, 1, 2*randn (1, time/Ts));     % bandlimited signal between 3K and 4K
y=filter (b, 1, x+n);                      % (a) filter the received signal+noise
yx=filter (b, 1, x); yn=filter (b, 1, n); % (b) 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 improvesnr.eps was actually drawn
```

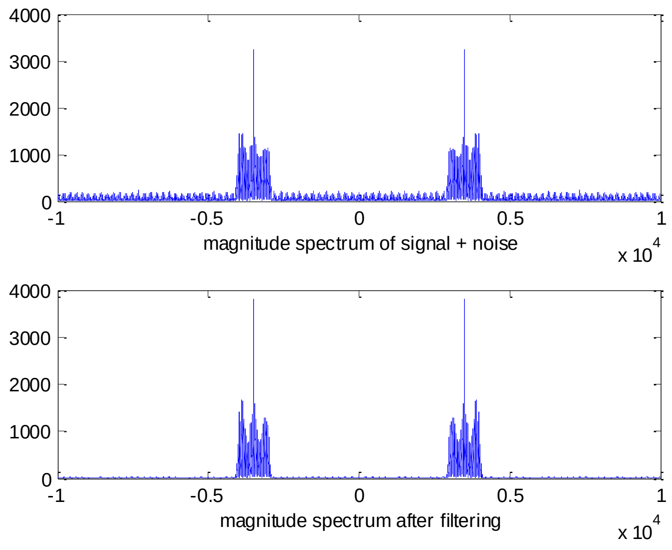
```
N=length(x);                              % length of the signal x
t=Ts*(1: N);                              % define time vector
ssf= (-N/2: N/2-1)/ (Ts*N);               % frequency vector
fx=fftshift (fft(x (1: N) +n (1: N)));
figure (5), subplot (2, 1, 1), plot (ssf, abs(fx))
xlabel ('magnitude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

Solution for (a): Filter response for narrowband noise outside passband (SNR increases):



snrinp = 97.7756 snrout = 6.5072e+005

Solution for (b): Filter response for narrowband noise inside passband (SNR decreases):



snrinp = 97.4446 snrout = 67.7584

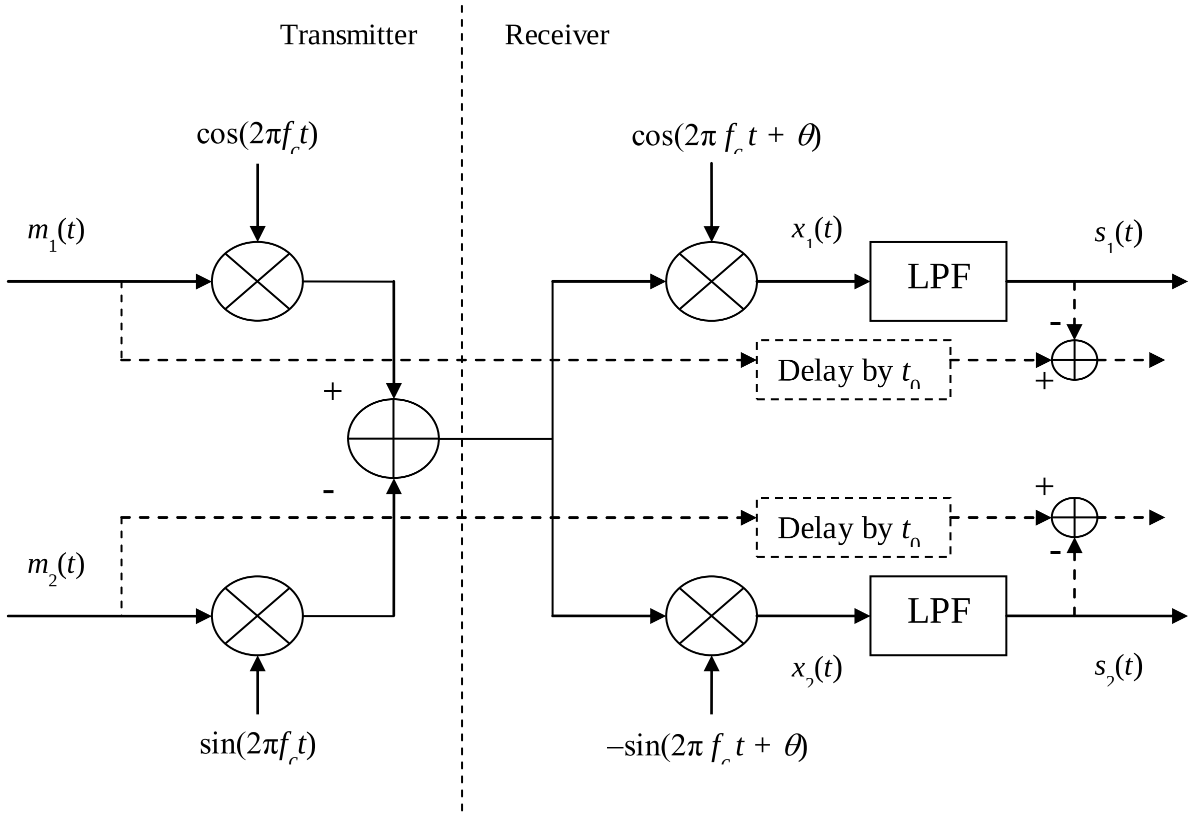
Some of the energy in the signal and interference resides outside the 3000-4000 Hz band. The bandpass filter passes a higher percentage of interference energy than signal energy.

3.2. Quadrature Amplitude Modulation and Demodulation. 32 points.

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

Use AM.m as a starting point to create a quadrature modulation system that implements the block diagram of Figure 5.10.
(a) Examine the effect of a phase offset in the demodulating sinusoids of the receiver, so that $x_1(t) = v(t)\cos(2\pi f_c t + \varphi)$ and $x_2(t) = -v(t)\sin(2\pi f_c t + \varphi)$ for a variety of φ. Refer to Exercise 5.6.
(c) Confirm that a $\pm 1^\circ$ phase error in the receiver oscillator corresponds to more than 1% cross-interference.

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



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:

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

When $\theta = 0$, 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. The code below scales the result of the filter by 2. When $\theta = \pi/2$, i.e. when there is severe phase error,

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

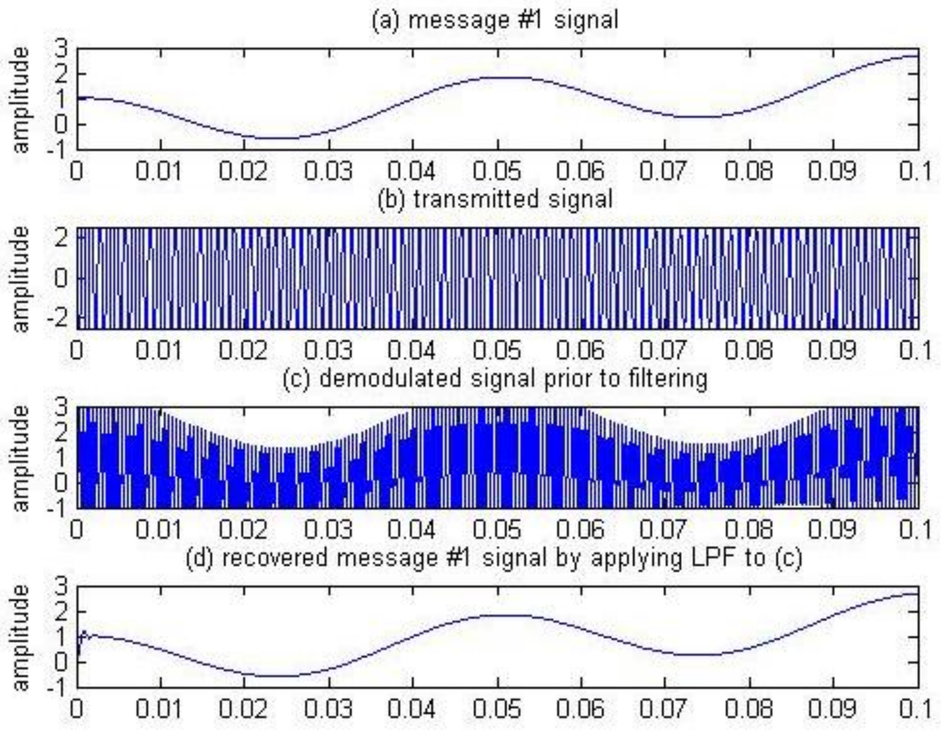
which recovers the wrong message signal.

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 the negative of the derivative of the phase response with respect to frequency. For a linear phase FIR filter with N coefficients, the group delay is a constant n_0 which is equal to $(N-1)/2$ 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)');
```



Solution for (c): To compute the cross-interference in the upper channel, we'll need to align transmitted message #1 and received message #1 in time. Here are two ways:

- Delay the transmitted signal by the group delay n_0 -OR-
- Remove the first n_0 samples and the last n_0 samples of the LPF output as we did in part (a):

```
s1=2*conv(b,x1);                 % LPF the demodulated signal
s1short=s1(n0+1:n0+lent);        % remove first and last n0 samples
```

Then, sum the squares of the samples in the difference signal to compute the amount of energy leakage. Finally, divide the energy leakage by the energy in transmitted message signal #1 to find out the fraction of energy leakage to get

$$Fractional\ Energy\ Leakage = \frac{\sum_n (m_1[n-n_0] - s_1[n])^2}{\sum_n m_1^2[n-n_0]}$$

The reciprocal of fractional energy leakage is a measure of signal-to-noise ratio at the LPF output:

$$SNR = \frac{Signal\ Power}{Noise\ Power} = \frac{Average\ Power\ of\ Message\ \#1}{Average\ Power\ of\ Cross\ Interference\ at\ LPF\ Output}$$

We'll search a range of phase offset values between -8^o and 8^o to find the smallest one that gives 1% energy leakage from the quadrature channel to the in-phase channel. Here's the code:

```
phi= (-8:8)*pi/180;                     % phase in radians
for i = 1:length(phi)
    gam=0;                              % freq & phase offset
    cr=cos(2*pi*(fc+gam)*t+phi(i));     % 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(i));    % 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
    fprintf('Phi is %f degrees ', phi(i)*180/pi) % print phi (in degrees)
    leakage1 = sum((s1short-m1).^2)/sum(m1.^2)   % print the energy leakage
    leakage2 = sum((s2short-m2).^2)/sum(m2.^2)
end
```

A phase offset of 6^o causes about 1% energy leakage. The result depends on the message signals.

3.3. Infinite Impulse Response Filter Design. 50 points.

This problem asks you to design a discrete-time digital IIR filter for an electrocardiogram (ECG) signal. This problem is a sequel to homework problem 2.3.

An ECG device "monitors the cardiac status of a patient by recording the heart's electrical potential versus time. Such devices play a very important role to save life of patients who survive heart attack or suffer from serious heart diseases. The time to respond to a heart attack is very critical for these patients. An early detection of conditions that lead to the onset of cardiac arrest allows doctors to provide proper treatment on time and prevents death or disability from cardiac arrest." [1]

"There exist three types of noise that contaminate the ECG signal: the baseline wander noise (BW), electromyographic interference (EMG), and the power line interference. The BW is induced by electrodes' changes due to perspiration, movement and respiration, and is typically below 0.5 Hz. The power line interference either 50 Hz or 60 Hz and its harmonics are a significant source of noise." [1].

Our goal in designing the filter is to attenuate baseline wander noise and powerline interference. It would take more sophisticated processing to track and cancel the EMG noise because EMG noise appears in the same frequencies as the ECG signal.

Here are the bandpass filter specifications for your design:

- For frequencies 0 Hz to 1 Hz, the stopband attenuation should be at least 40 dB.
- For frequencies 6 Hz to 40 Hz, the passband ripple should be no greater than 1 dB.
- For frequencies above 45 Hz, the stopband attenuation should be at least 40 dB.

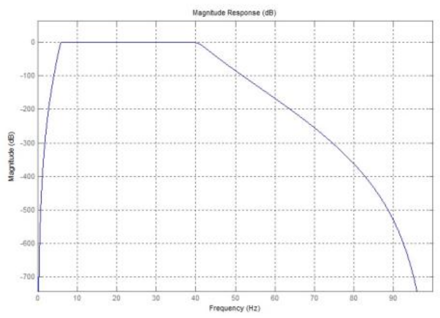
These specifications would be compatible with the monitor mode in modern ECG monitors.

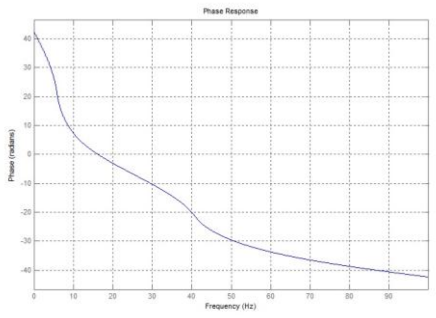
Please use a sampling rate of 200 Hz. From a quick survey of commercial ECG systems, I found sampling rates that vary from 100 Hz to 1000 Hz. The PTB Diagnostic ECG Database uses a sampling rate of 1000 Hz and the QT ECG Database uses a sampling rate of 250 Hz.

(a) Design IIR filters using the Butterworth, Chebyshev type I, Chebyshev type II, and Elliptic (Equiripple) 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.

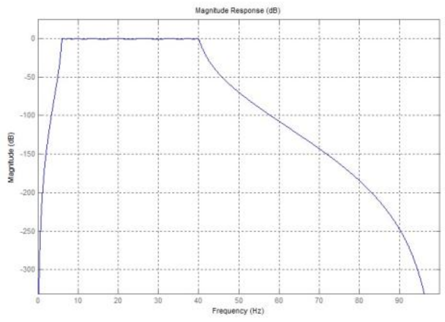
Solution for (a): f_{sample}= 200 Hz, f_{stop1}= 1 Hz, f_{pass1}= 6 Hz, f_{pass2}= 40 Hz, f_{stop2}= 45 Hz, A_{pass}= 1 dB, A_{stop1}= 40 dB, A_{stop2}= 40 dB. Magnitude responses (left) and phase responses (right) are below.

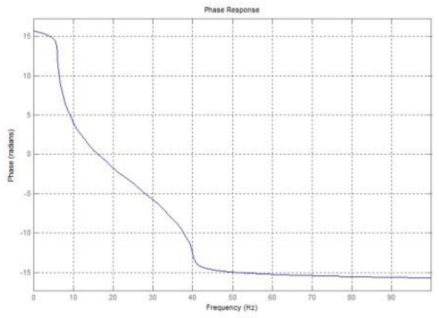
Butterworth: (Order 54) which has approximate linear phase over 10-40 Hz.



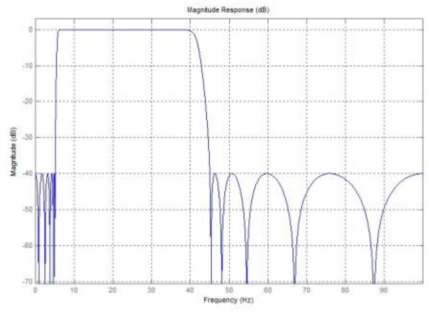


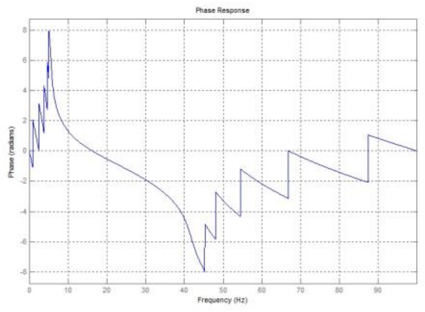
Chebyshev Type I: (Order 20) which has approximate linear phase over 15-30Hz.



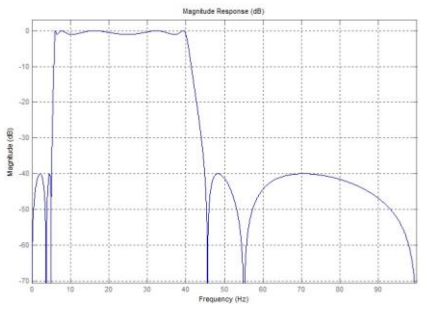


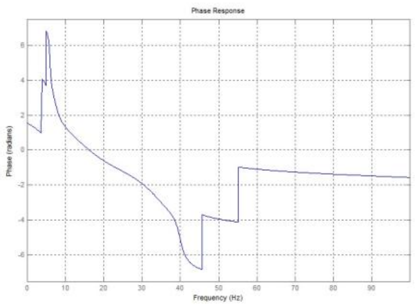
Chebyshev Type II (Order 20) which has approximate linear phase 15-30 Hz.





Elliptic (Order 10) which has approximate linear phase over 20-30 Hz.



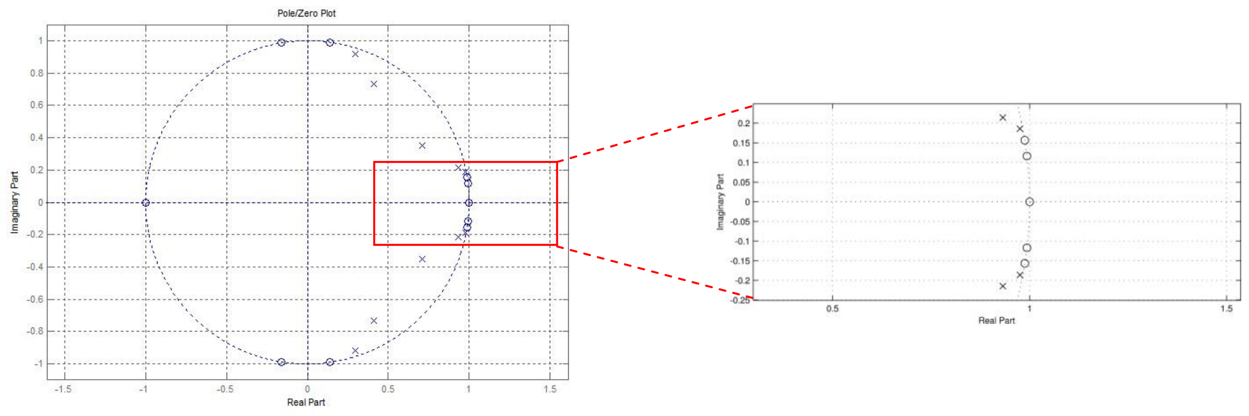


(b) List the filter orders required for filters to meet the specification. Which IIR filter family gives the lowest filter order? Plot the pole-zero diagram for the IIR filter with shortest filter order you designed in (a). Because the poles and zeros are close to the unit circle and separated in angle,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

Solution for (b): The filter orders required to meet the specs are given below:

Filter Type	Order
Butterworth	54
Chebyshev Type I	20
Chebyshev Type II	20
Elliptic	10

Elliptic design produces lowest-order IIR filter. Its pole-zero plot is below. Filter has five pairs of conjugate symmetric poles, four pairs of conjugate symmetric zeros and two real zeros. ***Poles near the unit circle indicate the passband(s) and zeros on/near unit circle represent the stopband(s).***



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.

	b0	b1	b2	a0	a1	a2	Gain
1st Biquad	1	-0.27392	1	1	-0.58766	0.929460	0.519074
2nd Biquad	1	-1.97530	1	1	-1.95045	0.985682	0.519074
3rd Biquad	1	0.31741	1	1	-0.81831	0.707349	0.522444
4th Biquad	1	-1.98635	1	1	-1.86395	0.914577	0.522444
5th Biquad	1	0	-1	1	-1.41946	0.628808	0.481637

(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): In the course reader, Appendix N entitled "Tapped Delay Line on C6700 DSP" is very useful here. On the C6700 DSP, the single-precision floating-point instruction MPYSP has a delay of three instruction cycles and a throughput of one instruction cycle. When coefficients and data are in single-precision floating-point format, the bottleneck is actually the speed at which we can load coefficients and data from on-chip memory to registers using the two data buses. Each data bus is 32 bits wide and a single precision floating point number is 32 bits wide. Hence, we can read only one filter coefficient and one data value per instruction cycle. This will leave one of the two multiplier units idle. ADDSP has three delay slots and a throughput of one cycle. For a tapped delay line with N coefficients, the code given in Appendix N takes N+28 instruction cycles.

There are more than 20 commonly used IIR filter structures. For the sake of this problem, we will assume that the IIR filter is implemented as a cascade of a biquads so as to match the design procedure used in part (a). For the single-precision case, assuming a filter order of M the number of stages is given by $\left\lceil \frac{M}{2} \right\rceil$ where $\lceil x \rceil$ returns the smallest integer that is greater than or equal to x. If the order of the filter were odd, then one of the stages would just be a first-order section.

Number of instruction cycles. Each biquad has 3 feedforward coefficients and 2 feedback coefficients, and computing the difference equation requires 5 multiplications and 4 additions per output sample. 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 single-order section needs 3 multiplications and 2 additions per output sample. The implementation complexity is the same as a tapped delay line with three coefficients, i.e. 31 instructions. If the computation for biquad #2 waits until the computation of biquad #1 is 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 five biquads, which would require 5 x 33 = 165 instructions.

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 + 17*4 = 101 instructions.

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) = 49 cycles.

Filter Coefficient Storage: Each biquad would require 2 numerator coefficients and 2 denominator coefficients. (In actuality, a biquad has six coefficients, but two of them are "1" always. Each biquad has a constant gain, and the constant gains for all of the biquads can be collapsed into one gain for the entire filter). So that's, (4*5) + 1 coefficients which takes up 21*4 = 84 bytes. If it is done in a single stage, it would have 11 numerator coefficients and 10 denominator coefficients (one of the denominator coefficients is always 1). So the required storage is (11+10)*4=84 bytes.

Previous Values Storage: We need to store 10 previous input samples (ignoring the current input) and 10 previous outputs. 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 16 words. So, for the two buffers, we'll need $16\times2\times4=128$ bytes.

(d) Based on the results obtained from this problem and problem 2.3 on the previous homework, 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.

Solution for (d): In **terms of computation**, the 60th-order FIR filter designed by the Parks-McClellan algorithm would take 61 multiplication-accumulation operations per output sample, whereas the elliptic IIR filter would take 11+10=21 multiplication-accumulation operations per output sample in direct form. Hence, the elliptic IIR filter in direct form would be about 3 times more efficient in terms of multiplication-accumulation operations. If we compared C6700 instructions, the FIR filter would take 61+28=89 instructions. Straightforward implementations of a direct form IIR structure would take (order+1+28)+(order+1+28) or 78 instruction cycles, and of a cascade-of-biquad IIR structure would take 165 instruction cycles, as mentioned in part (c). Efficient implementations of the direct form IIR structure would take (order+1+order+28) or 49 instruction cycles, and an efficient implementation of the cascade-of-biquad IIR structure would take 101 instruction cycles. An efficient pipelined implementation of the cascade-of-biquad IIR structure would take 49 instruction cycles, as described in part (c).

In **terms of stability**, the FIR filter is always stable, regardless of implementation. The IIR filter will need to be implemented with care, e.g. as a cascade of biquads in ascending order of quality factors from input to output.

In **terms of phase response**, the FIR filter designed by the Parks-McClellan algorithm has linear phase over all frequencies. The IIR filter has non-linear phase, although its phase response is approximately linear over part of the passband. IIR filters remain a very efficient solution since we can design filters with very low orders, and hence IIR notch filters can be used in subsequent processing to remove power line interference around 60 Hz. FIR filters have two major disadvantages because of their high orders: (1) they require lot of computation which is not suited for a portable ECG monitoring device which runs on a small battery, and (2) the long impulse response of the filter introduces delay that may not be desirable for real-time monitoring of the ECG signal. A group delay of 30 samples at a sampling rate of 200 Hz means a delay of 0.15 s.

MATLAB Scripts from in Johnson, Sethares and Klein's *Software Receiver Design* textbook

The Matlab scripts should run "as is" in MATLAB or LabVIEW Mathscript facility.

1. Copy the .m files on your computer from the "SRD - MatlabFiles" folder on the CD ROM: http://users.ece.utexas.edu/~bevans/courses/rtdsp/homework/SRD-MatlabFiles.zip
2. Add the folder containing the .m files from the book to the search path.:
   - In MATLAB, use the `addpath` command
   - In LabVIEW, open the Mathscript window in LabVIEW by going to the Tools menu and select "Mathscript Window" (third entry), go the File menu, select "LabVIEW MathScript Properties" an d add the path.

Johnson, Sethares and Klein intentionally chose not to copyright their programs so as to enable their widespread dissemination.

Discussion of this solution set will be available online soon.