ECE 2713
Design Project

Spring 2025 Dr. Havlicek

DUE: 04/27/2025, 11:59 PM
CAN BE TURNED IN LATE WITHOUT PENALTY UNTIL: 05/2/2025, 11:59 PM

What to Turn In:

Submit your solution for this assignment electronically by uploading **two files** to to the course Canvas page:

1. a regular MS WORD or PDF report file just like you made for Homework 3 and Homework 6;

2. a filtered digital audio wave file called `filteredsig.wav` that you will create.

Remember:

Make sure to include your name in your turn-in report file and add a title at the top of the first page that says "ECE 2713" and "Design Project." Number the problems and paste in your Matlab code, the resulting command window output, and the figures and graphs. Number the problems and paste in your Matlab code and the resulting command window output. Paste in the figures and graphs and make sure to include your answer for the explanation/discussion question in Problem 3(c).

- If you are using the Virtual Labs, make sure to save all your files before you log out!

- If you use *m*-files, you can paste them into your turn-in file.

- As you work the problems, you can also use the mouse to cut your Matlab code and resulting output from the command window and paste them into your turn-in file.

- You can also use the Matlab `diary` command as explained in Homework 3 and Homework 6 to save a session log from the command window into a file which you can then insert into your turn-in file.

- For figures and graphs, you can use the File pulldown menu of the Matlab Figure Window to save them as JPEG or BMP files and then insert them into your turn-in file as pictures. To make the color work on the Virtual Labs, I had to open "Export Setup" from the File menu of the Matlab Figure window and **uncheck** the "custom color" box.

The Assignment:

Professional compact disc digital audio is sampled with a sampling frequency of $F_s = 44.1$ kHz. This means that there are 44,100 samples per second. The time interval between samples is called the *sampling period*. It is given by $T_s = 1/F_s \approx 22.676$ μsec. Although professional compact disc audio signals are stereo and have two channels of audio data, in this project we will only consider single-channel (mono) audio signals.

The audio samples are stored as 16-bit two's complement integers. In high performance professional applications, they are stored without compression. For computer processing, the digital audio samples are usually stored in a *wave file* (`.wav`). Matlab provides a built-in function `audioread` that can read the digital audio data in a wave file into a Matlab array. It also provides a built-in function `audiowrite` that can write the digital audio data in a Matlab array out to a wave file.

The Matlab statement

```
[x,Fs] = audioread('test.wav');
```

reads the digital audio signal contained in the file `test.wav` into the Matlab array `x`. The sampling rate, which is stored in the wave file, is placed in the Matlab variable `Fs`. For professional compact disc digital audio, it is always 44.1 kHz. When Matlab reads in the 16-bit two's complement integer audio samples, it converts them to double precision floating point numbers in the range $[-1, 1]$.

The Matlab statement

```
audiowrite('test.wav',x,Fs);
```

writes the digital audio data stored in the double precision floating point array `x` out to the wave file `test.wav` in 16-bit two's complement integer format. It is important for you to make sure that the digital audio data in the array `x` is normalized to the range $[-1, 1]$ before you call `audiowrite`, since operations like filtering and adding signals together will generally change the range. This can be done by placing the Matlab statement

```
x = x / max(abs(x));
```

just before the call to `audiowrite`.

The Matlab statement

```
sound(x,Fs,16);
```

will play the digital audio data in the array `x` through the sound card as 16-bit two's complement integers with a sampling frequency of `Fs`. As with `audiowrite`, it is important for you to ensure that the data stored in `x` are normalized to the range $[-1, 1]$ before you call `sound`.

The Nyquist frequency is given by $F_s/2 = 22.05$ kHz. A/D and D/A conversion, i.e., sampling, maps the analog Nyquist frequency to the digital frequency $\omega = \pi$ radians per sample. Thus, if the Matlab array `x` holds an N-point digital audio signal, then, in the N-point centered DFT array `X = fftshift(fft(x))`,

- the digital frequency ω goes from $-\pi$ to $\pi - \frac{2\pi}{N}$ in steps of $\frac{2\pi}{N}$, and
- the analog frequency in Hertz goes from $-\frac{Fs}{2}$ to $\frac{Fs}{2} - \frac{Fs}{N}$ in steps of $\frac{Fs}{N}$.

For practical digital audio signals, the magnitude of the centered DFT is usually plotted in dB as $20 \log_{10} |X[k]|$. Note that this will make a numerical error if $|X[k]| = 0$. So, if there are places k where $|X[k]| = 0$, you have to change it to a small nonzero number instead when you compute the logarithm.

For a digital audio signal with a sampling rate of F_s Hz,

- to convert (Hertzian) analog frequency to radian digital frequency, multiply the analog frequency by $\frac{2\pi}{F_s}$.
- to convert (Hertzian) analog frequency to Hertzian digital frequency, multiply the analog frequency by $\frac{1}{F_s}$.
- to convert (Hertzian) analog frequency to normalized digital frequency, multiply the analog frequency by $\frac{2}{F_s}$.

1. Recall that for professional compact disc digital audio, the sampling rate is $F_s = 44.1$ kHz. On a piano, the first "A note" that is located above middle C on the keyboard has an analog frequency of 440 Hz. Consider the Matlab code below, which does the following:
   - Makes a two-second digital audio cosine signal with analog frequency 440 Hz (this will require $44{,}100 \times 2 = 88{,}200$ samples). Such a signal is called a "pure tone."
   - Plays the 440 Hz pure tone through the sound card.
   - Plots the centered DFT magnitude in dB as a function of Hertzian analog frequency, radian digital frequency, and normalized digital frequency.
   - Writes the signal to a wave file.
   - Reads the signal back in from the wave file.
   - Plays the read-in signal through the sound card.

```
%-------------------------------------------------------
% P1a
%
% Make a 2 second digital audio signal that contains a pure
% cosine tone with analog frequency 440 Hz.
% - play the signal through the sound card
% - plot the centered DFT magnitude in dB against
%     Hertzian analog freq, radian digital freq,
%     and normalized digital freq.
% - Write the signal to a wave file, read it back in, and
%     play it through the sound card again.
%

Fs = 44100;                     % sampling frequency in Hz
N = Fs * 2;                     % length of the 2 sec signal
n = 0:N-1;                      % discrete time variable
f_analog = 440;                 % analog frequency in Hz
w_dig = 2*pi*f_analog/Fs;       % radian digital frequency
x = cos(w_dig * n);             % the signal

% Normalize samples to the range [-1,1]
%  Not really needed here b/c cos is already in this range,
%  but done anyway to illustrate how you normalize.

x = x / max(abs(x));

sound(x,Fs,16);                 % play it through sound card

X = fftshift(fft(x));           % centered DFT
Xmag = abs(X);                  % centered DFT magnitude
XmagdB = 20*log10(Xmag);        % convert to dB

% Plot the centered magnitude against analog frequency
w = -pi:2*pi/N:pi-2*pi/N;       % dig rad freq vector
f = w * Fs /(2*pi);             % analog freq vector
figure(1);
plot(f,XmagdB);
xlim([-20000 20000]);
title('Centered DFT Magnitude for 440 Hz Pure Tone');
xlabel('analog frequency, Hz');
ylabel('dB');
```

```
% Plot the centered magnitude against radian digital freq
figure(2);
plot(w,XmagdB);
xlim([-pi pi]);
title('Centered DFT Magnitude for 440 Hz Pure Tone');
xlabel('radian digital frequency \omega');
ylabel('dB');

% Plot against normalized digital frequency
figure(3);
plot(w/pi,XmagdB);
xlim([-1 1]);
title('Centered DFT Magnitude for 440 Hz Pure Tone');
xlabel('normalized digital frequency \omega/\pi');
ylabel('dB');

% wait 3 seconds in case sound card is still busy
pause(3);

audiowrite('A-440.wav',x,Fs);   % write to wave file
[x2,Fs] = audioread('A-440.wav'); % read it back in
sound(x2,Fs,16);                % play it again Sam!
```

(a) Type in this code and run it. You can type it in line-by-line at the command prompt or you can create an *m*-file. You can also download the *m*-file from the course Canvas page under "Files for the Design Project."

(b) Modify the Matlab code to generate and play a cosine pure tone with an analog frequency of 5 kHz.

Now we are going to do some filtering. The following Matlab code will design a lowpass IIR digital Butterworth filter:

```
Wp = 0.4;
Ws = 0.6;
Rp = 1;
Rs = 60;
[Nf, Wn] = buttord(Wp,Ws,Rp,Rs);
[num,den] = butter(Nf,Wn);
```

The frequency response magnitude is shown in Fig. 1 on the next page. The x-axis is in normalized digital frequency and the y-axis is in dB. Since $|H(e^{j\omega})|$ is even symmetric, we normally plot it for the non-negative frequencies only.

The parameter `Wp` specifies the passband edge frequency. For this filter, we set `Wp` to a normalized digital frequency of 0.4. This makes the passband go from DC to 0.4π rad/sample. In the passband, $|H(e^{j\omega})| \approx 1 = 0$ dB. The parameter `Rp` specifies the allowable passband ripple, which is the amount that $|H(e^{j\omega})|$ is allowed to deviate from 0 dB in the passband. For this filter, we set `Rp` to 1. This means that $|H(e^{j\omega})|$ has to be between -1 dB and +1 dB everywhere in the passband.

The parameter `Ws` specifies the stopband edge frequency. For this filter, we set `Ws` to a normalized digital frequency of 0.6. So the stopband goes from 0.6π rad/sample up to π rad/sample. The parameter `Rs` specifies the minimum stopband attenuation. For this filter, we set `Rs` to 60 dB. This means that everywhere in the stopband $|H(e^{j\omega})|$ has to be below -60 dB.

The region between `Wp` and `Ws` is called the transition band. For this filter, the transition band goes from 0.4 to 0.6 in units of normalized digital frequency, which is 0.4π to 0.6π rad/sample.

The main features of the digital Butterworth filter are that it is maximally flat in the passband, the passband is monotonic (there is no rippling), and the phase is approximately linear in the passband.

The parameter `Nf` returned by `buttord` gives the filter order. This is the highest power of $e^{-j\omega}$ that appears in the numerator or denominator of the frequency response $H(e^{j\omega})$. It is also the highest power of z^{-1} that appears in $H(z)$. The parameter `Wn` gives the Butterworth natural frequency, which is the point in the transition band where the frequency response magnitude has dropped by 3 dB compared to the passband. For this filter, the order is 12.

The vectors `num` and `den` returned by `butter` contain the coefficients for the numerator and denominator polynomials of $H(e^{j\omega})$, which are the same as the numerator and denominator coefficients of the transfer function $H(z)$.

The Matlab statement to run the filter is `y = filter(num,den,x);` where `x` is the input signal and `y` is the output signal.

You can also design a highpass digital Butterworth filter like this:

```
Wp = 0.6;
Ws = 0.4;
Rp = 1;
Rs = 60;
[Nf, Wn] = buttord(Wp,Ws,Rp,Rs);
[num,den] = butter(Nf,Wn,'high');
```

Notice that this time `Wp` > `Ws`. That is because the stopband is now on the left starting at DC, followed by the transition band in the middle and the passband on the right. The frequency response magnitude for this filter is shown in Fig. 2 on the page 8.

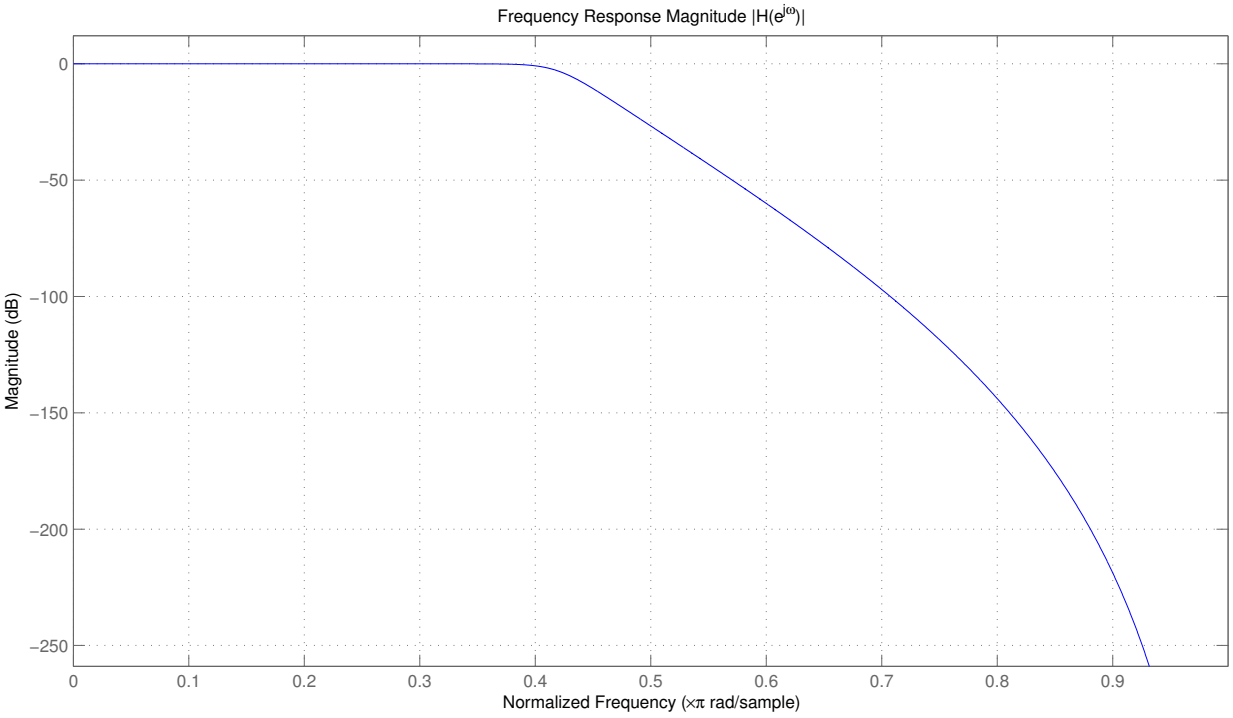

Figure 1: Lowpass digital Butterworth filter frequency response.

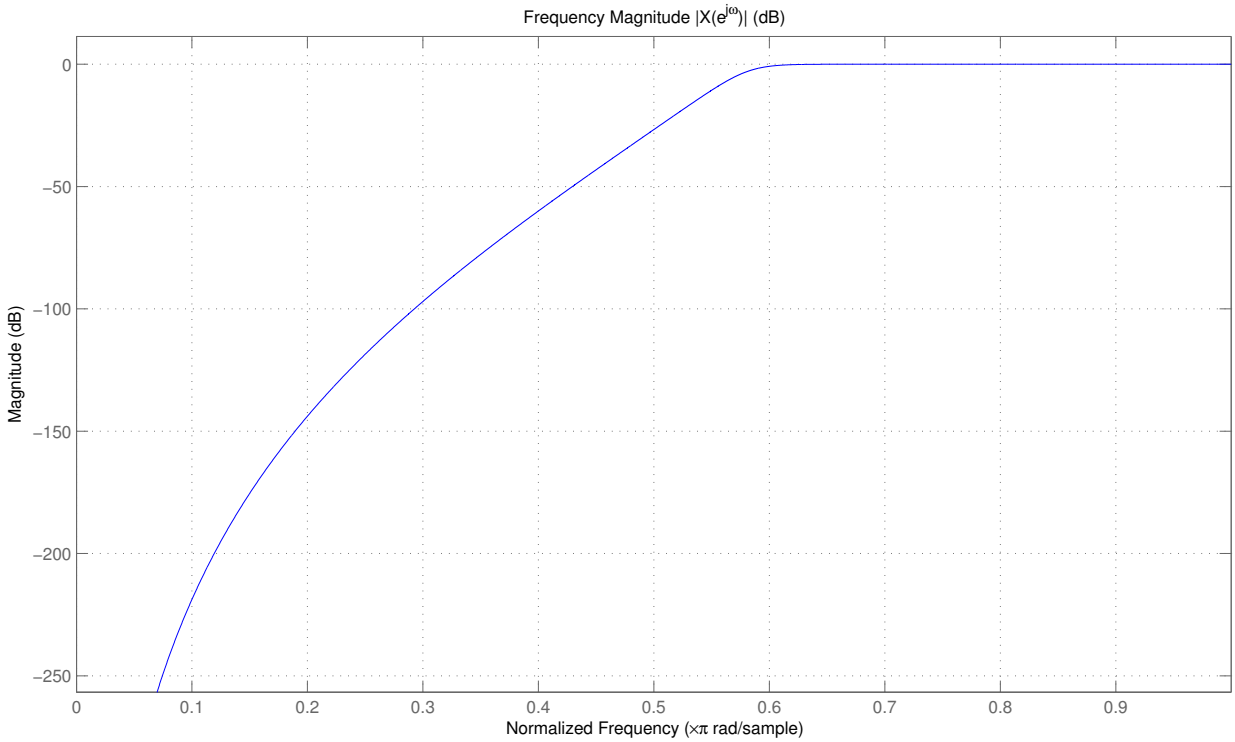

Figure 2: Highpass digital Butterworth filter frequency response.

The parameter `Wp` again specifies the passband edge frequency, which we set to a normalized digital frequency of 0.6. So the passband goes from 0.6π rad/sample to π rad/sample. As before, we set `Rp` to 1, so $|H(e^{j\omega})|$ has to be between -1 dB and +1 dB everywhere in the passband.

We set the stopband edge frequency `Ws` to a normalized digital frequency of 0.4. So the stopband goes from DC to 0.4π rad/sample. We set the minimum stopband attenuation parameter `Rs` to 60, so $|H(e^{j\omega})|$ is below -60 dB everywhere in the stopband.

The transition band lies between the stopband and the passband. For this filter, the transition band goes from 0.4 to 0.6 in normalized digital frequency, which is 0.4π rad/sample to 0.6π rad/sample in radian digital frequency.

For any given filtering problem, we usually want to use the smallest filter order `Nf` that we can. A higher filter order means more delay, more complexity, and increased implementation cost. It also means that the frequency response phase $\arg H(e^{j\omega})$ will be more nonlinear, which is undesirable for digital audio. Here are the factors that increase the filter order:

- for a given passband ripple `Rp` and minimum stopband attenuation `Rs`, making the width of the transition band smaller will increase the order.
- for a given transition bandwidth, decreasing `Rp` or increasing `Rs` will increase the order.

Now, having `Rp` bigger than 1 dB could distort the signal in the filter passband, which is bad. So, for a given filtering problem, we generally want to use the widest transition bandwidth $|\texttt{Ws} - \texttt{Wp}|$ and the smallest stopband attenuation `Rs` that will do the job.

2. Consider the Matlab code below, which does the following:
   - Makes the signal `x1` a 250 Hz pure tone that lasts for 4 sec.
   - Plays `x1` through the sound card.
   - Makes the signal `x2` a swept frequency chirp that goes from 1 kHz to 3 kHz. The details of how I made this chirp signal are not really important to you for this project.
     - But in case you are interested, here they are: human hearing perceives the signal $x(t) = \cos[\varphi(t)]$ as a tone with a time-varying frequency $\varphi'(t)$. The quantity $\varphi'(t)$ is called the *instantaneous frequency*. For a chirp, $\varphi'(t)$ changes linearly with time, which means that the instantaneous phase $\varphi(t)$ has to be quadratic in time. A digital audio chirp signal is given by $x[n] = \cos(\varphi[n])$, where the instantaneous phase $\varphi[n]$ is quadratic in n. So I set $\varphi[n] = an^2 + bn + c$. I set the initial phase offset c to zero to get $\varphi[n] = an^2 + bn$ and $\varphi'[n] = 2an + b$. At $n = 0$, I wanted the analog starting frequency of the chirp to be 1 kHz, which is a radian digital frequency of $\omega_1 = 2\pi \times 1000/F_s$. At $n = 0$, this gave me $\varphi'[0] = b = \omega_1$. At $n = N - 1$, I wanted the analog ending frequency of the chirp to be 3 kHz, which is a radian digital frequency of $\omega_2 = 2\pi \times 3000/F_s$. At $n = N - 1$, this gave me $\varphi'[N - 1] =$

$2a(N-1)+\omega_1=\omega_2$, or $a=\frac{\omega_2-\omega_1}{2(N-1)}$. So the desired digital chirp signal is given by $x[n]=\cos(\varphi[n])$ where $\varphi[n]=\frac{\omega_2-\omega_1}{2(N-1)}n^2+\omega_1 n$.

- Plays `x2` through the sound card.
- Makes the signal `x3` = `x1` + `x2`.
- Normalizes `x3` to the range [-1, 1] and plays it through the sound card.
- Designs a lowpass digital Butterworth filter to process the signal `x3` by keeping the 250 Hz pure tone but filtering out the chirp. I set the passband edge frequency at 250 Hz, which is a radian digital frequency of $2\pi \times 250/F_s$. To get the value of `Wp` for `buttord`, I divided this by π to convert it to normalized digital frequency. I set the stopband edge frequency to the starting frequency of the chirp, which is 1 kHz in analog frequency and $2\pi \times 1000/F_s$ in radian digital frequency. To get the value of `Ws` for `buttord`, I divided this by π to convert it to normalized digital frequency. These values of `Wp` and `Ws` place the 250 Hz pure tone `x1` in the filter passband and the chirp signal `x2` entirely in the filter stopband. I set the maximum passband ripple `Rp` to 1 dB and the minimum stopband attenuation `Rs` to 60 dB. The lowpass filtered signal is called `y1`.
- Calls the Matlab `fvtool` and `freqz` functions to display the filter frequency response.
- Plays the lowpass filtered signal `y1` through the sound card.
- Designs a highpass digital Butterworth filter to process the signal `x3` by keeping the chirp signal but filtering out the 250 Hz pure tone. I set the stopband edge frequency at 250 Hz, which is a radian digital frequency of $2\pi \times 250/F_s$. For the `buttord` parameter `Ws`, I divided this by π to convert it to normalized digital frequency. This places the pure tone in the filter stopband. I set the passband edge frequency at the starting frequency of the chirp, or 1 kHz, which is a radian digital frequency of $2\pi \times 1000/F_s$. For the `buttord` parameter `Wp`, I divided this by π to convert it to normalized digital frequency. This places the chirp signal entirely in the filter passband. I used 1 dB for the maximum passband ripple `Rp` and 60 dB for the minimum stopband attenuation `Rs`. The highpass filtered signal is called `y2`.
- Adds the highpass filter to `fvtool` and calls `freqz` to plot the frequency response.
- Plays the highpass filtered signal `y2` through the sound card.

```
%-------------------------------------------------------
% P2a
%
% Make some digital audio signals and demonstrate filtering.
% All signals are 4 seconds in duration.
% - Make x1 a 250 Hz pure tone.
% - Play x1 through the sound card.
% - Make x2 a swept frequency chirp from 1 kHz to 3 kHz.
% - Play x2 through the sound card.
% - Make x3 = x1 + x2.
% - Play x3 through the sound card.
% - Apply a lowpass digital Butterworth filter to x3 to
%     keep the pure tone and reject the chirp.
% - Play the filtered signal through the sound card.
% - Apply a highpass digital Butterworth filter to x3 to
%     keep the chirp and reject the pure tone.
% - Play the filtered signal through the sound card.
%

Fs = 44100;                     % sampling frequency in Hz
N = Fs * 4;                     % length of the 4 sec signal
n = 0:N-1;                      % discrete time variable

% Make x1 a 250 Hz pure tone
f_analog = 250;                 % pure tone analog frequency
w_dig = 2*pi*f_analog/Fs;       % radian digital frequency
x1 = cos(w_dig * n);            % the pure tone
sound(x1,Fs,16);                % play it through sound card
pause(5);                       % wait for sound card to clear

% Make x2 a chirp.  Sweep analog freq from 1 kHz to 3 kHz
f_start_analog = 1000;
w_start_dig = 2*pi*f_start_analog/Fs;
f_stop_analog = 3000;
w_stop_dig = 2*pi*f_stop_analog/Fs;
phi = (w_stop_dig-w_start_dig)/(2*(N-1))*(n.*n) + w_start_dig*n;
x2 = cos(phi);
sound(x2,Fs,16);                % play it through sound card
pause(5);                       % wait for sound card to clear

% Add the two signals
x3 = x1 + x2;
```

```
x3 = x3 / max(abs(x3));          % normalize the range to [-1,1]
sound(x3,Fs,16);                 % play it through sound card
pause(5);                        % wait for sound card to clear

% Use a lowpass digital Butterworth filter to keep the 250 Hz
%   pure tone and reject the chirp.
Wp = w_dig/pi;                   % normalized passband edge freq
Ws = w_start_dig/pi;             % normalized stopband edge freq
Rp = 1;                          % max passband ripple
Rs = 60;                         % min stopband attenuation
[Nf, Wn] = buttord(Wp,Ws,Rp,Rs);  % design filter order
[num,den] = butter(Nf,Wn);       % design the filter
h=fvtool(num,den);               % show frequency response
figure(2);
freqz(num,den,1024);             % plot frequency response
title('Lowpass Frequency Response');
y1 = filter(num,den,x3);         % apply the filter
y1 = y1 / max(abs(y1));          % normalize filtered signal
sound(y1,Fs,16);                 % play it through sound card
pause(5);                        % wait for sound card to clear

% Use a highpass digital Butterworth filter to keep the chirp
%   and reject the 250 Hz pure tone.
Ws = w_dig/pi;                   % normalized stopband edge freq
Wp = w_start_dig/pi;             % normalized passband edge freq
Rp = 1;                          % max passband ripple
Rs = 60;                         % min stopband attenuation
[Nf, Wn] = buttord(Wp,Ws,Rp,Rs); % design filter order
[num2,den2] = butter(Nf,Wn,'high'); % design the filter
Hd = dfilt.df1(num2,den2);       % make filter object
addfilter(h,Hd);                 % add filter 2 to fvtool
figure(3);
freqz(num2,den2,1024);           % plot frequency response
title(' Highpass Frequency Response');
y2 = filter(num2,den2,x3);       % apply the filter
y2 = y2 / max(abs(y2));          % normalize filtered signal
sound(y2,Fs,16);                 % play it through sound card
```

(a) Type in this code and run it. You can type it in line-by-line at the command prompt or you can create an *m*-file. You can also download it from the course Canvas page under "Files for the Design Project."

(b) Modify the Matlab code to do the following:

- Make `x1` a four second cosine pure tone with analog frequency 1 kHz and play it through the sound card.
- Make `x2` a four second cosine pure tone with analog frequency 3 kHz and play it through the sound card.
- Make `x3` = `x1` + `x2` and play `x3` through the sound card.
- Apply a lowpass digital Butterworth filter to `x3` to keep the 1 kHz pure tone but filter out the 3 kHz pure tone. You can use 1 dB for the maximum passband ripple `Rp` and 60 dB for the minimum stopband attenuation `Rs`. You will need to set the passband edge frequency $\geq$ 1 kHz. Note that this is $2\pi \times 1000/F_s$ in radian digital frequency. Divide that by π to get the minimum value for the normalized digital passband edge frequency `Wp`. You will need to set the stopband edge frequency $\leq$ 3 kHz. This is $2\pi \times 3000/F_s$ in radian digital frequency. Divide that by π to get the maximum value for the normalized digital passband edge frequency `Ws`.
- Play the lowpass filtered signal through the sound card.

3. Get the wave files `noisysig.wav` and `noisesamp.wav` from the course Canvas page. You will find them under "Files for the Design Project." The file `noisysig.wav` contains a digital audio signal that has been corrupted by additive noise. The digital audio signal in `noisesamp.wav` is a sample of just the noise (without any signal). In this problem, you will design a digital Butterworth filter to remove the noise from the signal in `noisysig.wav`.

(a) Use the Matlab `audioread` function to read each signal. Use Matlab to play the noisy signal through the sound card. You can also play the file using any wave-capable media player like *Windows Media Player* or VLC.

(b) Use the Matlab `length` function to find the length of each signal and plot the centered DFT magnitude in dB as a function of normalized digital frequency.

(c) Design a lowpass digital Butterworth filter to remove the noise. You can use 1 dB for the maximum passband ripple `Rp` and 60 dB for the minimum stopband attenuation `Rs`. Determine appropriate normalized digital frequency values for the passband edge frequency `Wp` and stopband edge frequency `Ws` by analyzing the centered DFT magnitude plots. Note that the centered DFT magnitude plot for the file `noisesamp.wav` will show you the frequency spectrum for the noise *only*, whereas the plot for the file `noisysig.wav` will show you the frequency spectrum for the *combined* signal plus noise. You must design `Wp` and `Ws` so that the filter order `Nf` is **twelve or less**. Briefly explain how you chose the values for `Wp` and `Ws`.

(d) Call the Matlab `fvtool` and `freqz` functions to plot the filter frequency response like we did in Problem 2.

(e) Apply your filter to remove the noise. Play the filtered signal through the sound card and use the Matlab `audiowrite` function to save it to a wave file. Name this wave file `filteredsig.wav`. Upload your `filteredsig.wav` file to the course Canvas page along with your WORD or PDF solution file.