

ECE 2610 Spring 2010 Final Project: Audio Signal Processing

1 Introduction

In this final team project you will be investigating the use of digital filters for audio signal processing. The audio signal processing of interest is for high quality audio, as opposed to speech processing for low bit rate modems, as found in cellular telephony and speech communications over the internet (VoIP and Skype). A package of m-code functions are documented in Appendix A and the source code itself is on the course web site as a ZIP file package. You will be writing many functions yourself, but also using ready-to-go functions.

Honor Code: The project teams will be limited to at most three members. Teams are to work independent of one another. Bring questions about the project to me. I encourage you to work in teams of at least two. Since each team member receives the same project grade, a group of two or three should attempt to give each team member equal responsibility. The due date for the completed project will be 4:00 pm, Wednesday May 12, 2010.

2 Background

High quality audio signal processing covers a range of applications from the musical recording studio, musical performance, home theatre, home stereo, car stereo system, or even a personal audio player. One class of applications is linear filtering, where there is a need or desire to reshape the audio spectrum of a source, which runs from about 20 Hz to 20 kHz. Here the linear filters of interest include *shelving* and *peaking* filters [1]. Another class of applications is that of special effects [2], such as *delay*, *echo*, *reverberation*, *comb filtering*, *flanging*, *chorusing*, *pitch shifting*, *distortion*, *compression*, *expansion*, and *noise gating*.

2.1 Filtering to Gain Equalize Selected Frequency Bands

In this project two types gain equalizing filters are considered: (1) shelving filters, which are like the bass and treble controls in a stereo system, and (2) peaking filters, which can be used to form a multi-band graphic equalizer. You will be implementing m-code modules for these filter types, so that you can design audio spectrum shaping filters to meet specific requirements.

2.1.1 Shelving Filters

The shelving filter can either be lowpass or highpass [3]. The lowpass form is used to raise or lower the spectrum level of an audio signal passing through the filter, below a cutoff frequency f_c Hz. This is basically what an ordinary bass control on a stereo accomplishes. The highpass form is used to raise or lower the audio spectrum level of an audio signal passing through the filter, above a cutoff frequency f_c . This is basically what an ordinary treble control accomplishes.

A digital lowpass shelving filter has system function [1]

$$H_{\text{lp}}(z) = C_{\text{lp}} \left(\frac{1 - b_1 z^{-1}}{1 - a_1 z^{-1}} \right), \quad (1)$$

where

$$\begin{aligned} C_{\text{lp}} &= \left(\frac{1 + k\mu}{1 + k} \right) \\ b_1 &= \left(\frac{1 - k\mu}{1 + k\mu} \right), \quad a_1 = \left(\frac{1 - k}{1 + k} \right) \\ k &= \left(\frac{4}{1 + \mu} \right) \tan \left(\frac{\hat{\omega}_c}{2} \right), \end{aligned}$$

and

$$\mu = 10^{G_{\text{dB}}/20}.$$

Note that this is a simple first-order IIR filter (note the signs in front of the coefficients). What makes this filter seem complicated, is the fact that as a shelving filter it has been parameterized in terms of the lowpass cutoff frequency

$$\hat{\omega}_c = 2\pi \frac{f_c}{f_s}, \quad (2)$$

where f_c is the cutoff frequency in Hz, f_s is the sampling frequency in Hz, and the lowpass gain in dB

$$G_{\text{dB}} = 20 \log_{10} |H(e^{j\hat{\omega}})| \text{ for } \hat{\omega} \ll \hat{\omega}_c. \quad (3)$$

Above the cutoff frequency the filter gain approaches 0 dB, that is $|H_{\text{lp}}(e^{\hat{\omega}})| = 1$ for $\hat{\omega} \gg \hat{\omega}_c$. A family of plots for a lowpass shelving filter is shown in Figure 1. Notice that in this plot we have used the analog frequency variable, f , in Hz, thus we have effectively plotted $|H_{\text{lp}}(e^{j2\pi f/f_s})|$ for $0 \leq f \leq f_s/2$. We have chosen $f_s = 44.1$ kHz as this is the sampling frequency standard for audio CDs. The studio standard is typically an integer multiple of 48 kHz, e.g., 48, 96, or 192 kHz. Also note in Figure 1 the frequency axis is logarithmic. In MATLAB we use the function `semilogx(x_axis_vec, y_axis_vec)` for plotting. To obtain frequency axis values uniformly spaced on log axis, the MATLAB function `logspace()` is useful. The filter magnitude response is also plotted in dB, which means that the y-axis is $20 \log_{10} |H_{\text{lp}}(e^{j2\pi f/f_s})|$. The MATLAB function `freqz()` can manage all of this for you, as you will see in the problems section of this project. Additionally, the helper function `s1_freqz_mag()` was written to encapsulate this type of plot, and make sure the frequency axis runs over just $0 < f \leq f_s/2$.

A digital highpass shelving filter has system function [1]

$$H_{\text{hp}}(z) = C_{\text{hp}} \left(\frac{1 - b_1 z^{-1}}{1 - a_1 z^{-1}} \right), \quad (4)$$

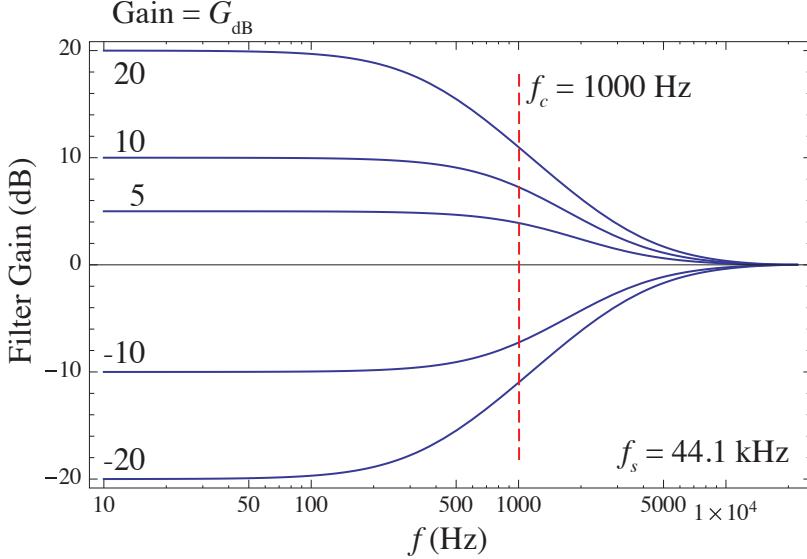


Figure 1: Lowpass shelving filter magnitude response in dB versus analog frequency f in Hz for $f_s = 44.1 \text{ kHz}$.

where

$$\begin{aligned} C_{\text{hp}} &= \left(\frac{\mu + p}{1 + p} \right), \\ b_1 &= \left(\frac{\mu - p}{\mu + p} \right), \quad a_1 = \left(\frac{1 - p}{1 + p} \right), \\ p &= \left(\frac{1 + \mu}{4} \right) \tan \left(\frac{\hat{\omega}_c}{2} \right), \end{aligned}$$

and as before

$$\mu = 10^{G_{\text{dB}}/20}.$$

Note that this is also a simple first-order IIR filter (note the signs in front of the coefficients). This filter has been parameterized in terms of the highpass cutoff frequency

$$\hat{\omega}_c = 2\pi \frac{f_c}{f_s}, \quad (5)$$

where f_c is the cutoff frequency in Hz, f_s is the sampling frequency in Hz, and the highpass gain in dB

$$G_{\text{dB}} = 20 \log_{10} |H(e^{j\hat{\omega}})| \text{ for } \hat{\omega} \gg \hat{\omega}_c. \quad (6)$$

Below the cutoff frequency the filter gain approaches 0 dB. A family of plots for a highpass shelving filter is shown in Figure 2.

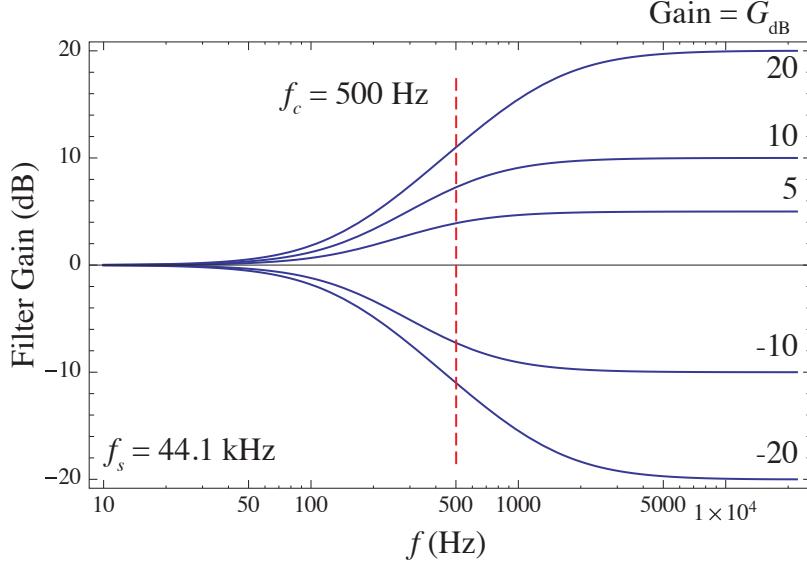


Figure 2: Highpass shelving filter magnitude response in dB for $f_s = 44.1 \text{ kHz}$.

2.1.2 Peaking Filters

A peaking filter is used to provide gain or loss (attenuation) at a specific center frequency f_c . As with the shelving filters, the peaking filter has unity frequency response magnitude or 0 dB gain, at frequencies far removed from the center frequency. At the center frequency f_c , the frequency response magnitude in dB is G_{dB} . At the heart of the peaking filter is a second-order IIR filter

$$H_{\text{pk}}(z) = C_{\text{pk}} \left(\frac{1 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \right), \quad (7)$$

where

$$\begin{aligned} C_{\text{pk}} &= \left(\frac{1 + k_q \mu}{1 + k_q} \right) \\ b_1 &= \left(\frac{-2 \cos(\hat{\omega}_c)}{1 + k_q \mu} \right), \quad b_2 = \left(\frac{1 - k_q \mu}{1 + k_q \mu} \right), \\ a_1 &= \left(\frac{-2 \cos(\hat{\omega}_c)}{1 + k_q} \right), \quad a_2 = \left(\frac{1 - k_q}{1 + k_q} \right), \\ k_q &= \left(\frac{4}{1 + \mu} \right) \tan \left(\frac{\hat{\omega}_c}{2Q} \right), \end{aligned}$$

and as before

$$\mu = 10^{G_{\text{dB}}/20}.$$

The peaking filter is parameterized in terms of the peak gain, G_{dB} , the center frequency f_c , and a parameter Q , which is inversely proportional to the peak bandwidth. Examples of the peaking

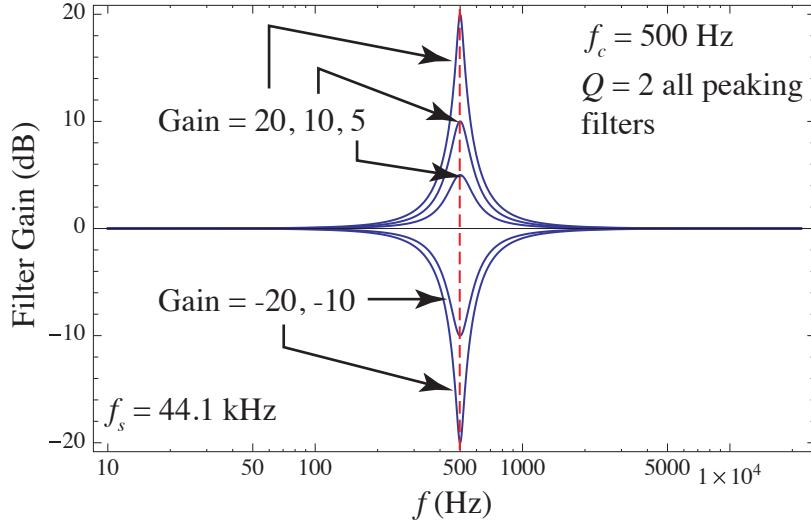


Figure 3: Individual peaking filter magnitude responses in dB for f_c fixed at 500 Hz, $f_s = 44.1$ kHz, Q fixed at 2, and $G_{\text{dB}} = -20, -10, 5, 10$, and 20.

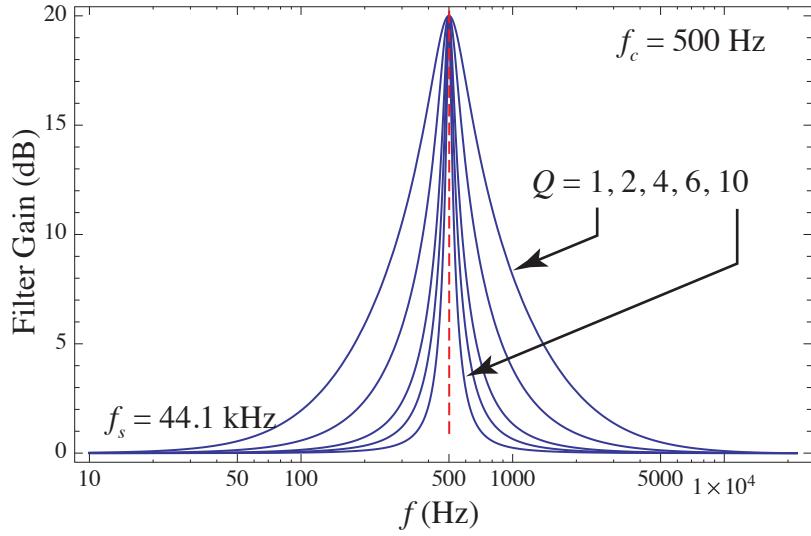


Figure 4: Individual peaking filter magnitude responses in dB for f_c fixed at 500 Hz, $f_s = 44.1$ kHz, and $Q = 1, 2, 4, 6$, and 10.

filter frequency response can be found in Figure 3 and Figure 4. The impact of changing the gain at the center frequency, f_c , can be seen in Figure 3. The impact of changing Q can be seen in Figure 4, in particular.

Peaking filters are generally placed in cascade to form a graphic equalizer [1, 4]. With the peaking filters in cascade, and the gain setting of each filter at 0 dB, the cascade frequency response is unity gain (0 dB) over all frequencies from 0 to $f_s/2$. In Figure 5 we see the individual response

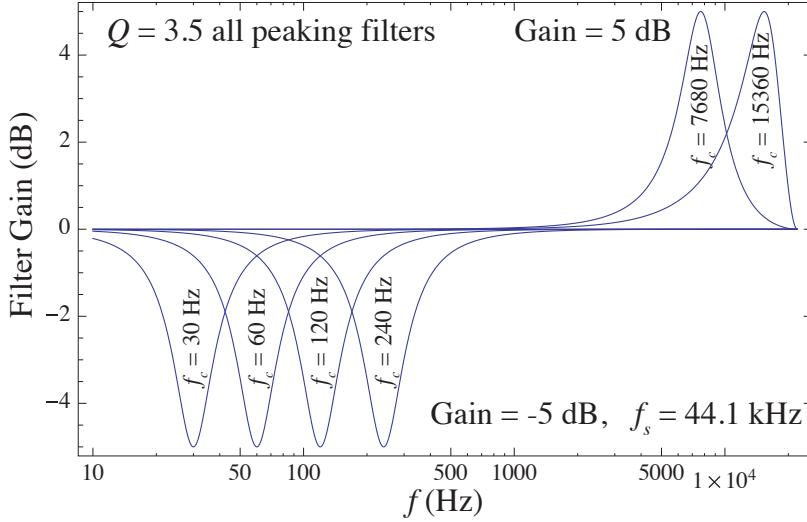


Figure 5: Individual peaking filter magnitude responses in dB with $Q = 3.5$, $G_{\text{dB}} = \pm 5$ dB, $f_c = 30 \times 2^i$, $i = 0, 1, 2, 3, 8, 9$, and $f_s = 44.1$ kHz.

of six filters set at octave center frequencies according to the formula

$$f_{ci} = 30 \times 2^i \text{ Hz}, \quad i = 0, 1, \dots, 9. \quad (8)$$

This corresponds to a ten octave graphic equalizer (*octave band equalizer*), where the octave band frequencies are spread from 30 Hz to 15,360 Hz. The idea here being a means to reasonably cover the 20 – 20 kHz audio spectrum. Note that human hearing begins to fall off severely above 15 kHz. With $f_s = 44.1$ kHz, we have a usable bandwidth up to $f_s/2 = 22.05$ kHz, but the design of antialiasing filters make it difficult to work all the way up to the folding frequency, and we do not hear signals at these frequencies anyway. From Figure 6, in particular, we notice that when the peaking filters are placed in cascade, the responses do not mesh together perfectly. In fact, the gain flatness in a particular band of frequencies depends on how much gain, G_{dB} , you want to achieve at a particular center frequency, relative to the adjacent frequency bands.

2.2 Special Effects

There is a long list of audio special effects available to the musician, recording engineer, and home audio enthusiast. In this project will consider just two of them. The first, and probably most well known is reverberation. Reverberation is something that happens naturally when a musical performance is heard in a large auditorium. The second effect considered here is flanging, which dates back to the 60's and 70's when rock bands were experimenting with special effects in recording electrified music (guitars, keyboards, and vocals).

2.2.1 Digital Reverberation

Reverberation occurs naturally when you are in a large listening space. Digital signal processing can be used to create the large space in a small space, using special digital filters. Consider the

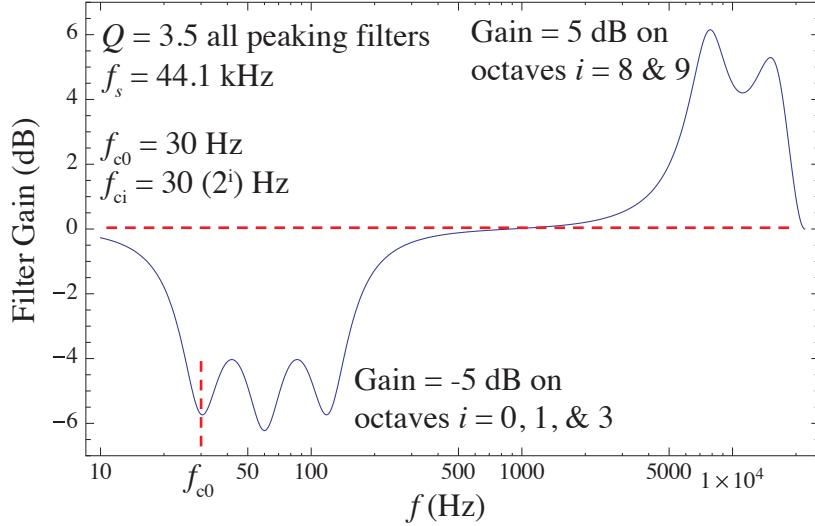


Figure 6: Peaking filter cascade magnitude response in dB with $Q = 3.5$, $G_{\text{dB}} = \pm 5$ dB, $f_c = 30 \times 2^i$, $i = 0, 1, 2, 3, 8, 9$, and $f_s = 44.1$ kHz.

large listening space depicted in Figure 7. A sound *launched* from the stage will propagate to the listener in roughly three distinct time periods [2]: direct sound, early reflections, and late reflections. Viewed as a linear filtering operation, we can consider the impulse response of the *channel* that exists between the sound source and the listener. The impulse response plotted versus time (as opposed to discrete-time), is depicted in Figure 8. Here we see the direct sound shows up as a delayed impulse function (first spike in the plot), followed by the early reflections (a few more individual spikes), followed by a very dense collection of spikes, which compose the many many late reflections (shaded in light blue).

To create an approximation to the impulse response of Figure 8, an interconnection of two digital filter types can be employed [2]. The first filter is the *all-pass* reverberator, which has system function

$$H_{\text{ap}}(z) = \frac{-a + z^{-D}}{1 - az^{-D}}. \quad (9)$$

The parameter D is generally a large integer, corresponding to the arrival of the first of the early reflections. The parameter a controls the intensity of the early reflections. A second filter is the *plain* reverberator, which has system function

$$H_p(z) = \frac{1}{1 - az^{-D}} \quad (10)$$

The complete reverberator, attributed to Schroeder [2, 5], is shown in block diagram form in Figure 9. The input section consists of four plain reverberators connected in parallel, followed by a cascade connection of two all-pass reverberators.

In MATLAB the function `sc_reverb(x, a, D, b)` implements Schroeder's reverb. The parameters following x in the function call are vectors corresponding to the feedback coefficients a_1, \dots, a_6 , the delays D_1, \dots, D_6 , and the summing gains b_1, \dots, b_4 , respectively. Simulation of the impulse response is shown below as command line code:

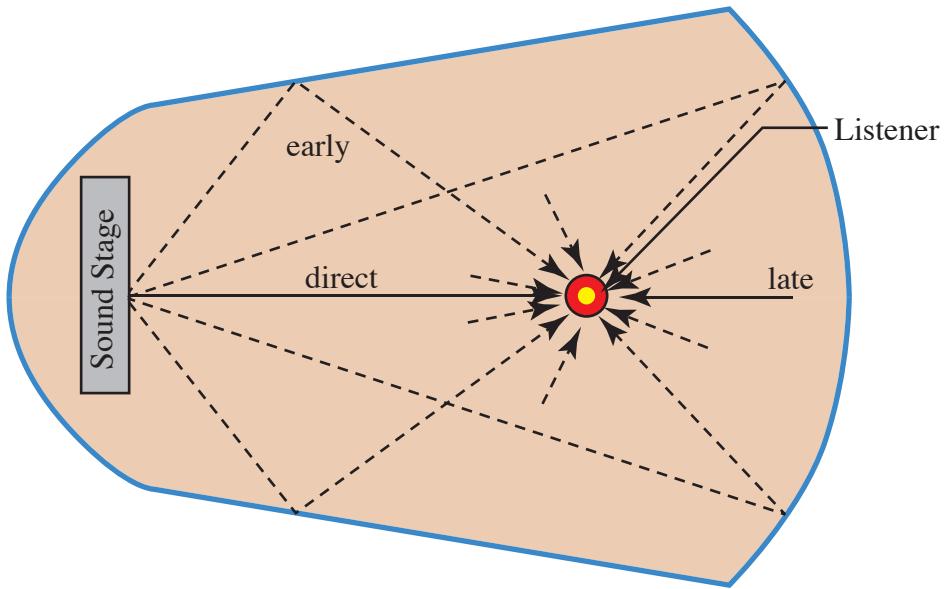


Figure 7: Reverberation in a listening space, e.g., a concert hall

```

>> t = 0:1/44100:2; % sampling rate fs = 44,100 samples/s
>> x = [1 zeros(1,length(t)-1)];
>> y = sc_reverb(x,0.75*ones(1,6),[29 37 44 50 27 31]*1,[1 .9 .8 .7]);
>> plot(t*1000,y) % plot as continuous since so many samples present
>> axis([0 15 -1.5 2.5])
>> grid
>> xlabel('Time (ms)')
>> ylabel('Impulse Response')

```

The impulse response is then plotted as if it is a continuous function of time, by letting $n/f_s \rightarrow t$ in the plot of Figure 10. Note that the early reflection are only about 1 ms delayed from the direct path. Late reflections are present out to 10 ms and beyond, however. In the problems section you will explore larger values for the D vector, i.e.,

$$[29 \ 37 \ 44 \ 50 \ 27 \ 31] \times 1 \longrightarrow [29 \ 37 \ 44 \ 50 \ 27 \ 31] \times M,$$

where $M \in [8, 30]$. You will also explore larger feedback values by increasing the a parameter vector, i.e.,

$$a=0.75*ones(1,6) \longrightarrow a=0.85*ones(1,6) \longrightarrow a=0.95*ones(1,6). \quad (11)$$

2.2.2 Flanging

Flanging started out as a mechanically induced effect created by playing back a musical recording through two tape players, and alternately slowing down the tape speed on each machine by pressing on the *flange* of the tape reels [2]. A third tape machine could be used to record the sum of the two signals and produce the final recorded version of the piece.

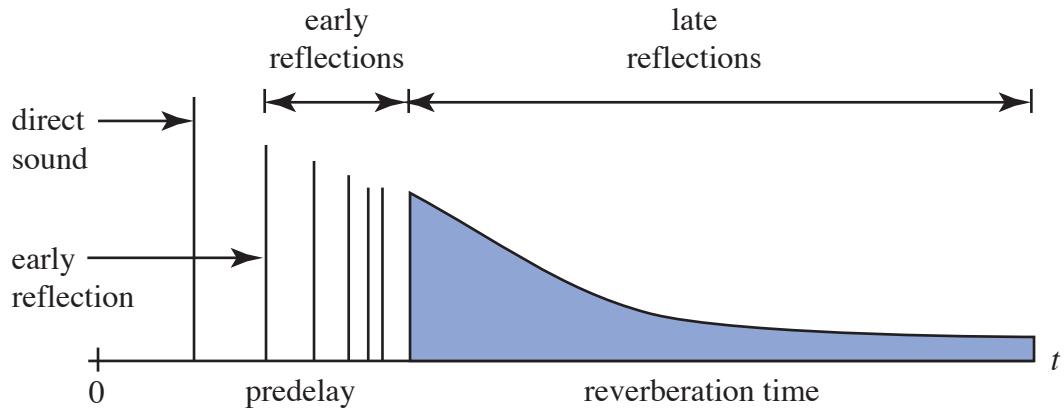


Figure 8: Reverberation impulse response

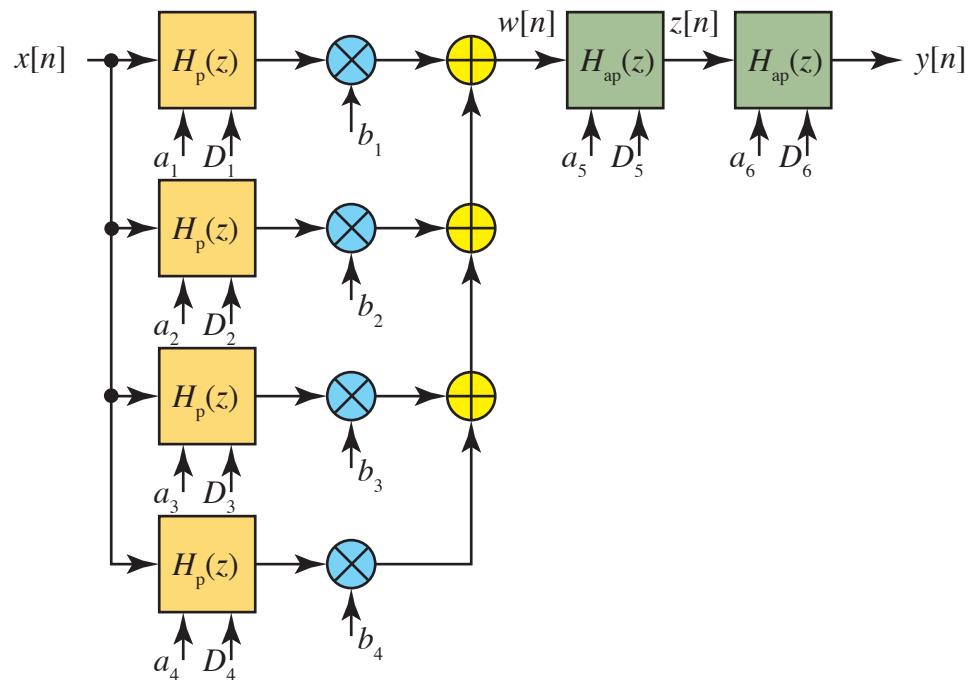


Figure 9: Schroeder's reverberator [2].

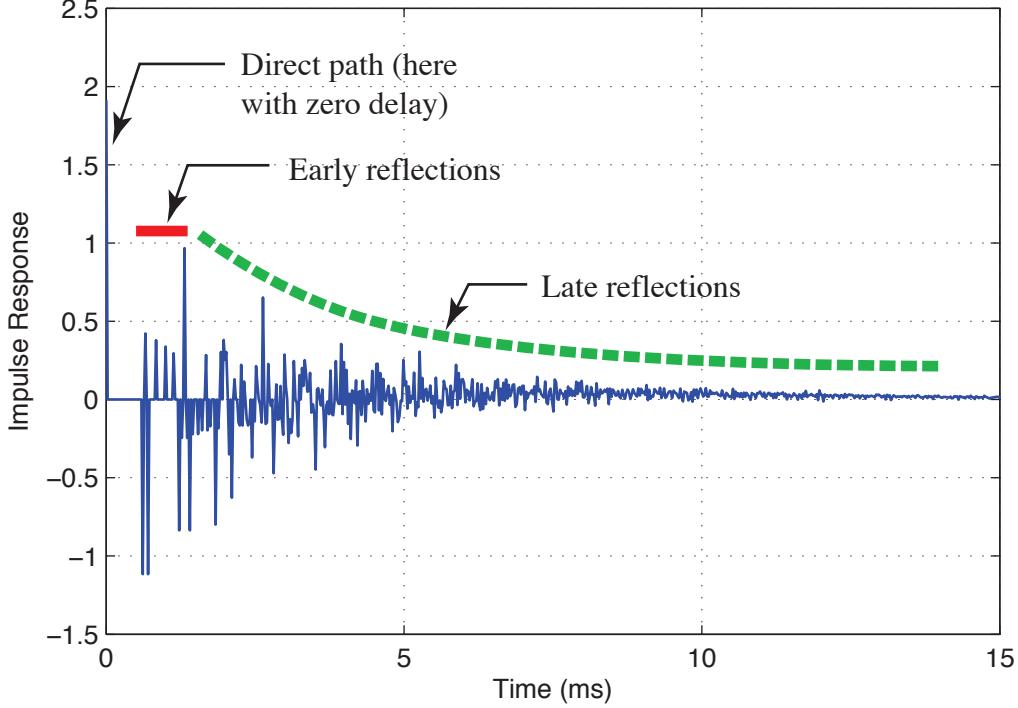


Figure 10: Impulse response obtained with Schroeder's reverberator.

In the discrete-time domain, the flanging effect is based on a time-varying delay processor having difference equation [2]

$$y[n] = (1 - a)x[n] + ax[n - D[n]]. \quad (12)$$

Notice that $D[n]$ is not in general a simple fixed delay, but now is a function of time index n . In practice $D[n]$ is periodic of the form

$$D[n] = \frac{D_{\max}}{2} \left\{ 1 - \cos \left(2\pi \frac{f_D}{f_s} n \right) \right\}, \quad (13)$$

where we choose the peak delay to correspond roughly to 10 ms and frequency f_D about 1 Hz. A complication with digital implementation is that $D[n]$ is not always integer valued. The fractional part of $D[n]$ implies that a fractional sample delay is needed. A delay processor can achieve fractional delay via interpolation, by using samples adjacent to the interval where the interpolation is required [6]. The details of this is beyond the scope of this project, but the interested reader may consult the reference provided.

When $D[n]$ changes continuously we are effectively warping the time axis, much like the Doppler effect we hear when a train passes by blowing its whistle. To make this more clear, consider a continuous-time sinusoid $x(t) = \cos(2\pi f_0 t)$. Let the delay $D(t)$ result from the listener moving away from the audio source at v_x m/s along the x axis. The resulting time delay is

$$D(t) = \frac{x}{v_p} = \frac{v_x t}{v_p}, \quad (14)$$

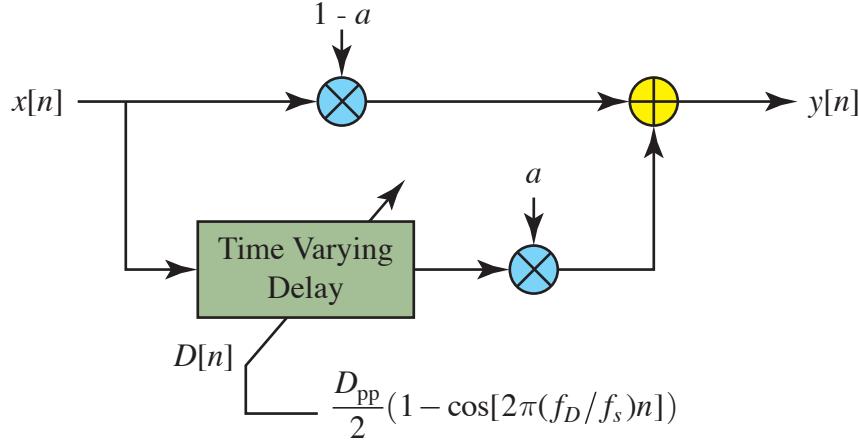


Figure 11: Block diagram of DSP based flanging effect generator.

where $v_p = 333.3 \text{ m/s}$ is the velocity of sound wave propagation. Forming $y(t)$ as the output of a time-varying delay processor, we have

$$y(t) = \cos [2\pi f_0(t - D(t))] \quad (15)$$

$$= \cos \left[2\pi \left(f_0 - \frac{f_0 v_x}{v_p} \right) t \right] \quad (16)$$

$$= \cos [2\pi(f_0 - \Delta f)t], \quad (17)$$

where Δf is the Doppler frequency

$$f_D = \frac{f_0}{v_p} \times v_x \quad (18)$$

Taking this a bit further, if we let $D(t)$ vary sinusoidally, we should obtain a sinusoidally varying Doppler frequency. You will verify this in the problems, and more.

The fact that the time varying delay can effectively *bend* the pitch of musical notes is the foundation of the flanging effect. With a DSP implementation it is however very easy to warp the signal in such a way that the processed audio is no longer recognizable.

3 Project Tasks

1. In this first task you will create a function which obtains the coefficients for the lowpass shelving filter of Section 2.1.1. The prototype for this function, `bass.m`, is in Appendix A.2. You will then test the function.
 - (a) Code the function `[b, a] = bass(GdB, fc, fs)`.
 - (b) Verify that the function works properly by creating overlay plots similar to Figure 1. Plot filter gain in dB versus log frequency. The helper function documented in Appendix A.1 should be of use here.

-
2. In this task you will create a function which obtains the coefficients for the highpass shelving filter of Section 2.1.1. The prototype for this function, `treble.m`, is in Appendix A.3. You will then test the function.
- Code the function $[b, a] = \text{treble}(GdB, fc, fs)$.
 - Verify that the function works properly by creating overlay plots similar to Figure 2. Plot filter gain in dB versus log frequency.
3. In this problem you will design an equalizer filter by cascading one or more of the shelving filters. The impulse response of two test channels, `h_chan1` and `h_chan2`, can be found by loading the file `channels.mat` into the MATLAB workspace. The design requirement is to gain equalize the audio channel from 20 Hz to 20 kHz back to approximately 0dB gain (flat response). The channel impulse response data was recorded with $f_s = 44.1$ kHz. To validate your equalizer designs, you will find the frequency response of the cascade formed with the given channel impulse response and your equalizer (composed of one or more shelving filters). The gain in dB of the cascade should be within ± 1 dB of 0 dB overall gain over the indicated frequency band. Design your filters to operate with $f_s = 44.1$ kHz.
- Design an equalizer cascade to gain equalize the cascade of `h_chan1` with one or more shelving filters, to the specifications given above.
 - Design an equalizer cascade to gain equalize the cascade of `h_chan2` with one or more shelving filters, to the specifications given above.
4. In this problem you will write code to design the peaking filter used in the graphic equalizer of Section 2.1.2. The prototype for this function is in Appendix A.5. You will then analyze the performance of an octave band equalizer using the supplied function `eq_sweep.m`. Note `eq_sweep` calls `peaking.m`, so your function must meet the function interface set forth in Appendix A.5. The equalizer will be employed to gain flatten one of the channels studied in Problem 2.
- Code the function $[b, a] = \text{peaking}(GdB, fc, Q, fs)$.
 - Verify that the function works properly by creating a cascade plot similar to Figure 6 using the supplied function `eq_sweep(GdB, Fc, Q)` to automatically plot the frequency response of a cascade of peaking filters. This will not test your function, but test that it integrates properly with the calling function `eq_sweep(GdB, Fc, Q)`. To match Figure 6 choose the peaking filter center frequency to be the 10 element vector
- $$fc = 30 * 2.^{[0:9]}.$$
- Note that you are actually working with a 10 band octave equalizer having center frequencies at 30 Hz, 60 Hz, 120 Hz, ..., 7680 Hz, 15,360 Hz.
- Using the same octave band spacing as in part (b) determine experimentally equalizer gain settings (10 total values) that will gain equalize `h_chan1` as used in Problem 3. You may use any value for Q , but all of the peaking filters should use the same Q value. Try to obtain a cascade gain, channel plus 10 band equalizer response, of 0 ± 2 dB. Do the best you can do with a reasonable amount of effort. Note that here you can

not simply use `eq_sweep()` as you need to cascade the impulse response `h_chan1` into your analysis. The support function `y = eq_filter(x,GdB,Fc,Q)`, Appendix A.7, will be helpful, however. The equalizer inputs for `(GdB,Fc,Q)` should be of the form

$$GdB = [GdB1 \ GdB2 \ GdB3 \ GdB4 \ GdB5 \ GdB6 \ GdB7 \ GdB8 \ GdB9 \ GdB10]$$

$$Fc = [30*2.^{[0:9]}] \text{ (10 octave bands starting at 30 Hz)}$$

$$Q = [Q*ones(1,10)] \text{ (constant } Q \text{ over all bands).}$$

- (d) Listening test: Process one or more of the sample sound vectors contained in `audio_test_vecs` (e.g., `combo`, `news_intro`, `piano_solo`, or `piano_scales`). Comment on what you hear when you listen to a signal that has been distorted by the impulse response `h_chan1` compared with the equalized versions of parts (3a) and (4c).
5. In this problem you will experiment with Schroeder's reverberator. The function file `sc_reverb.m` plus two support functions, `plain_reverb.m` and `ap_reverb.m` are supplied in complete form for this project. The function listings can be found in Appendix A.9. Your task is to perform a few experiments.
- (a) We would like the first early delay *spike* in the impulse response to occur at about 10 ms. Use the same parameters as used to generate Figure 10, except the value of M will be determined to make the first early delay at 10 ms. Plot the resulting impulse response.
 - (b) Using `subplot()` to create a stack of three plots, plot first the impulse response of part (a), then on plots two and three of the stack, plot the impulse response with the feedback parameter changed to 0.85 and 0.95 respectively (see equation (11)). Comment on the shape of the impulse response and explain how you think this will impact the sound you hear.
 - (c) Listening test: Process one or more of the sample sound vectors contained in `audio_test_vecs` (e.g., `combo`, `news_intro`, `piano_solo`, or `piano_scales`). Comment on what you hear. Try varying the delay M and the feedback parameter explored in part (b).
6. In this final experiment series you will work with the flanging special effect using the supplied function `Flanging(x,a,fD,fs,Dpp)`. The function listing can be found in Appendix A.10.
- (a) In the description of the flanging effect, it was conjectured that for $x[n]$ a single sinusoid and $D[n]$ a sinusoidally varying time delay, the output of the flanging processor (with $a = 1$) would be a signal whose instantaneous frequency is sinusoidally varying. To validate this, let $x[n]$ be a 1 kHz sampled sinusoid with $f_s = 44.1$ kHz for a duration of 2 s. Send $x[n]$ through `Flange(x,1,2,44100,1000)`. Analyze the output using MATLAB's spectrogram (`specgram(x,L,fs)`, start with $L=512$) function. Note the instantaneous frequency variation you observe. You will likely need to zoom into a region of the spectrogram plot to see what you're looking for. Also listen to $y[n]$ using `sound(y/2,44100)`. The 2 is to scale the amplitude so it does not overload the computer audio player (`soundsc()` may be used to, but sometimes overload still occurs).

- (b) From part (a) you know that you can create a pitch that wavers rapidly and over a wide frequency range. Let the sound reference vector piano_scales serve as $x[n]$. Pass $x[n]$ through the flanging processor using the configuration of part (a). Comment on the sound you hear before and after flanging. Is this too extreme?
- (c) Repeat part (b), but now let $a = 1/2$, so you get an even mixture of direct and time delay varied signals. Also let $f_D = 1$ Hz and $D_{pp} = 200$. Comment on the sound you hear now.
- (d) Listening test: Comment on the flanging effect when applied to the solo piano piece of sound vector piano_solo, using the settings of part (c). Try the other test vectors as you see fit and make additional comments. Feel free to experiment with the flanging parameters. I think you will find that it is very easy to make an audio recording sound terrible.
- (e) For bonus points, that can overflow over 100% for this project, analytically determine the instantaneous frequency at the output of the flanging processor for a single sinusoid input. As a starting point work from equation (15) with

$$D(t) = \frac{D_{pp}/f_s}{2} \cos(2\pi f_D t). \quad (19)$$

Note: Here we have taken $D[n]$ and modeled it in the continuous-time domain using the fact that $t \rightarrow n/f_s$, when we convert from continuous to discrete in a digital-to-analog converter. A time delay of D_{pp} in samples thus becomes a delay in seconds of D_{pp}/f_s .

- 7. Comment on your overall experience with this team project.

References

- [1] A. Spanias, T. Painter, and V. Atti, *Audio Signal Processing and Coding*, Wiley, New York, 2007.
- [2] S. Orfanidis, *Introduction to Signal Processing*, Prentice Hall, New Jersey, 1996.
- [3] http://en.wikipedia.org/wiki/Shelving_filter.
- [4] http://en.wikipedia.org/wiki/Graphic_Equalizer.
- [5] M.R. Schroeder, "Natural Sound Artificial Reverberation," *J. Audio Eng., Soc.*, 10, 219, 1962.
- [6] T. Laakso, V. Valimaki, M. Karjalainen, and U. Laine, "Splitting the Unit Delay," *IEEE Signal Processing Magazine*, January 1996, pp. 30–60.

A MATLAB Function Listings

In this appendix the complete MATLAB m-code listings are provided for many of the custom functions used in this project. For functions that the student has to write, only prototypes are given. These prototypes serve as a template for how to write the function to have proper input/output variable lists.

A.1 Frequency Response Magnitude Plot in dB versus Log Frequency

```
function [H,f] = sl_freqz_mag(b,a,fs,style)
% [H,f] = sl_freqz_mag(b,a,fs,style)
% Digital filter magnitude response plot in dB using a logarithmic
% frequency axis.
%
%/////////// Inputs ///////////
%
%      b = Denominator coefficient vector
%      a = Numerator coefficient vector
%      fs = Sampling frequency in Hz
% style = String variable list of plot options used by plot() etc.
%
%/////////// Output ///////////
%
% Output plot to plot window only
%
% Mark Wickert, April 2009

%f = logspace(0,log10(fs/2)); % start at 1 Hz
f = logspace(1,log10(fs/2)); % start at 10 Hz
w = 2*pi*f/fs;
H = freqz(b,a,w);

if nargin == 3
    style = 'b';
end

semilogx(f,20*log10(abs(H)),style);
ymm = ylim;
axis([10,fs/2,ymm(1),ymm(2)]);
grid on
%set(gca,'XminorGrid','off') % turn off minor xaxis grid
xlabel('Frequency (Hz)')
ylabel('Filter Gain (dB)')
```

A.2 Lowpass Shelving Filter Coefficients (function prototype only)

```
function [b,a] = bass(GdB, fc, fs)
% [b,a] = bass(GdB, fc, fs)
% Lowpass shelving (bass) filter having GdB gain in the passband
% and 0 dB otherwise.
%
%////// Inputs ///////////
```

```
%  
% GdB = Lowpass gain in dB  
% fc = Cutoff frequency in Hz  
% fs = Sampling frquency in Hz  
%  
%/////// Outputs ////////////  
%  
% b = Numerator filter coefficients in MATLAB form  
% a = Denominator filter coefficients in MATLAB form  
%  
% Mark Wickert, April 2009
```

A.3 Highpass Shelving Filter Coefficients (function prototype only)

```
function [b,a] = treble(GdB, fc, fs)  
% [b,a] = treble(GdB, fc, fs)  
% Highpass shelving (treble) filter having GdB gain in the passband  
% and 0 dB otherwise.  
%  
%/////// Inputs ////////////  
%  
% GdB = Highpass gain in dB  
% fc = Cutoff frequency in Hz  
% fs = Sampling frquency in Hz  
%  
%/////// Outputs ////////////  
%  
% b = Numerator filter coefficients in MATLAB form  
% a = Denominator filter coefficients in MATLAB form  
%  
% Mark Wickert, April 2009
```

A.4 Shelving Filter Cascade GUI

To assist with Problem 3a and b a MATLAB GUI was put together to study the cascade of two shelving filters. A MATLAB GUI is composed of two files, GUI_app.fig and GUI_app.m, where here the application name is shelving_cascade. To run the application type the function name at the command prompt. To edit the GUI layout you need to run the GUIDE editor.

To make use of this function in Problem 3a and b you will need to cascade the frequency response of the channel impulse responses, so that you can make the cascade response have the desired characteristics.

```
function varargout = shelving.Cascade(varargin)  
% SHELVING_CASCADE M-file for shelving.Cascade.fig  
% SHELVING_CASCADE, by itself, creates a new SHELVING_CASCADE or raises the existing  
% singleton*.  
%  
% H = SHELVING_CASCADE returns the handle to a new SHELVING_CASCADE or the handle to  
% the existing singleton*.  
%  
% SHELVING_CASCADE('CALLBACK', hObject, eventData, handles,...) calls the local
```

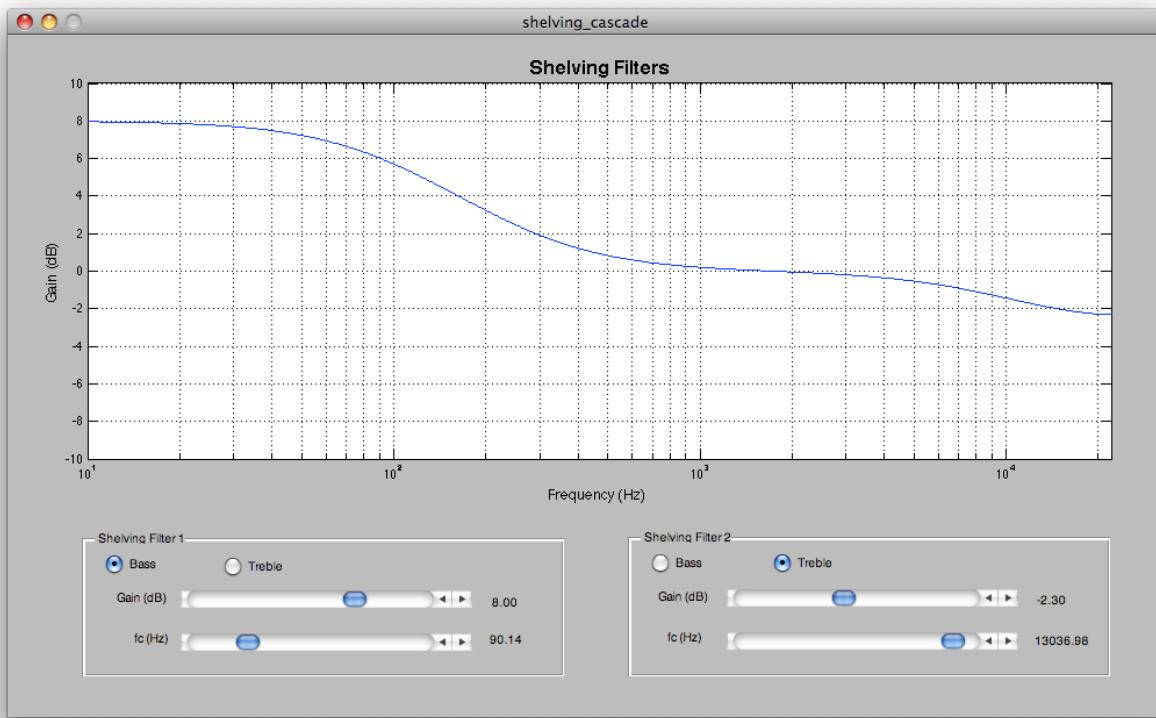


Figure 12: Screen shot of the shelving filter cascade GUI application displaying just the frequency response of two shelving filters in cascade.

```
%      function named CALLBACK in SHELVING CASCADE.M with the given input arguments.
%
%      SHELVING CASCADE('Property','Value',...) creates a new SHELVING CASCADE or raises the
%      existing singleton*. Starting from the left, property value pairs are
%      applied to the GUI before shelving_cascade_OpeningFcn gets called. An
%      unrecognized property name or invalid value makes property application
%      stop. All inputs are passed to shelving_cascade_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help shelving_cascade

% Last Modified by GUIDE v2.5 27-Apr-2009 17:14:58

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',        mfilename, ...
                   'gui_Singleton',   gui_Singleton, ...
                   'gui_OpeningFcn', @shelving_cascade_OpeningFcn, ...

```

```

'gui_OutputFcn', @shelving_cascade_OutputFcn, ...
'gui_LayoutFcn', [] , ...
'gui_Callback', []);

if nargin && ischar(varargin1)
    gui_State.gui_Callback = str2func(varargin1);
end

if nargout
    [varargout1:nargout] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before shelving_cascade is made visible.
function shelving_cascade_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata   reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to shelving_cascade (see VARARGIN)

% Choose default command line output for shelving_cascade
handles.output = hObject;
% Update handles structure
guidata(hObject, handles);

% UIWAIT makes shelving_cascade wait for user response (see UIRESUME)
% uiwait(handles.figure1);
global g1 g2 fc1 fc2 shelving1_mode shelving2_mode;
g1 = 0.0; g2 = 0.0; fc1 = 1000; fc2 = 1000;
shelving1_mode = 1;
shelving2_mode = 0;

set(handles.slider1,'Value',g1);
set(handles.text1,'String',sprintf('%6.2f',g1));
set(handles.slider2,'Value',log10(fc1));
set(handles.text2,'String',sprintf('%6.2f',fc1));
set(handles.slider3,'Value',g2);
set(handles.text3,'String',sprintf('%6.2f',g2));
set(handles.slider4,'Value',log10(fc2));
set(handles.text4,'String',sprintf('%6.2f',fc2));
set(handles.radioButton1,'Value',1.0);
set(handles.radioButton4,'Value', 1.0);
plot_response()

% --- Outputs from this function are returned to the command line.
function varargout = shelving_cascade_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata   reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

```

```
% Get default command line output from handles structure
varargout1 = handles.output;

% --- Executes during object creation, after setting all properties.
function axes1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes1 (see GCBO)
% eventdata   reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
plot_response()
%eq_sweep([0 0 2 -2],[30 60 120 240],[2 2 2 2])
% Hint: place code in OpeningFcn to populate axes1

% --- Executes on slider movement.
function slider1_Callback(hObject, eventdata, handles)
% hObject    handle to slider1 (see GCBO)
% eventdata   reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%         get(hObject,'Min') and get(hObject,'Max') to determine range of
%         slider
global g1
g1 = get(handles.slider1,'Value');
set(handles.text1,'String',sprintf('%6.2f',g1));
plot_response()

% --- Executes during object creation, after setting all properties.
function slider1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider1 (see GCBO)
% eventdata   reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes on slider movement.
function slider2_Callback(hObject, eventdata, handles)
% hObject    handle to slider2 (see GCBO)
% eventdata   reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%         get(hObject,'Min') and get(hObject,'Max') to determine range of slider
global fc1
fc1 = 10^(get(handles.slider2,'Value'));
set(handles.text2,'String',sprintf('%6.2f',fc1))
plot_response()

% --- Executes during object creation, after setting all properties.
```

```

function slider2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider2 (see GCBO)
% eventdata   reserved - to be defined in a future version of MATLAB
% handles     empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes on slider movement.
function slider3_Callback(hObject, eventdata, handles)
% hObject    handle to slider3 (see GCBO)
% eventdata   reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
global g2
g2 = get(handles.slider3,'Value');
set(handles.text3,'String',sprintf('%6.2f',g2))
plot_response()

% --- Executes during object creation, after setting all properties.
function slider3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider3 (see GCBO)
% eventdata   reserved - to be defined in a future version of MATLAB
% handles     empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes on slider movement.
function slider4_Callback(hObject, eventdata, handles)
% hObject    handle to slider4 (see GCBO)
% eventdata   reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
global fc2
fc2 = 10^(get(handles.slider4,'Value'));
set(handles.text4,'String',sprintf('%6.2f',fc2))
plot_response()

% --- Executes during object creation, after setting all properties.
function slider4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider4 (see GCBO)
% eventdata   reserved - to be defined in a future version of MATLAB
% handles     empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.

```

```

if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes on button press in radiobutton1.
function radiobutton1_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton1 (see GCBO)
% eventdata   reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton1
global shelving1_mode
shelving1_mode = get(handles.radiobutton1,'Value');
plot_response()

% --- Executes on button press in radiobutton2.
function radiobutton2_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton2 (see GCBO)
% eventdata   reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton2
global shelving1_mode
shelving1_mode = get(handles.radiobutton1,'Value');
plot_response()

% --- Executes on button press in radiobutton3.
function radiobutton3_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton3 (see GCBO)
% eventdata   reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton3
global shelving2_mode
shelving2_mode = get(handles.radiobutton3,'Value');
plot_response()

% --- Executes on button press in radiobutton4.
function radiobutton4_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton4 (see GCBO)
% eventdata   reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton4
global shelving2_mode
shelving2_mode = get(handles.radiobutton3,'Value');
plot_response()

////////////////// Custom Function for Updating Plot/////////////////
function plot_response()
///////////////////////////////
global g1 g2 fc1 fc2;
global shelving1_mode shelving2_mode;
load channels;

```

```

if shelving1_mode == 1.0
    [b1,a1] = bass(g1,fc1,44100);
else
    [b1,a1] = treble(g1,fc1,44100);
end
if shelving2_mode == 1.0
    [b2,a2] = bass(g2,fc2,44100);
else
    [b2,a2] = treble(g2,fc2,44100);
end
% Cascade two shelving filters by convolving filter coefficient sets
a = conv(a1,a2);
b = conv(b1,b2);
[H,F] = sl_freqz_mag(b,a,44100,'b');

%%%%%%%%%%%%%
% Add code here to allow you to graphically solve problem 3a&b
%%%%%%%%%%%%%

semilogx(F,20*log10(abs(H)));
%axis([10 fs/2 -10 10]);
grid
%set(gca,'XminorGrid','off')
axis([10 44100/2 -10 10])
xlabel('Frequency (Hz)', 'FontSize',12)
ylabel('Gain (dB)', 'FontSize',12)
title('Shelving Filters', 'FontSize',16, 'FontWeight','bold')

```

A.5 Peaking Filter Coefficients (function prototype only)

```

function [b,a] = peaking(GdB, fc, Q, fs)
% [b,a] = peaking(GdB, fc, Q, fs)
% Second-order peaking filter having GdB gain at fc and approximately
% and 0 dB otherwise.
%
%%%%%%%% Inputs /////////////////
%
% GdB = Lowpass gain in dB
% fc = Center frequency in Hz
% Q = Filter Q which is inversely proportional to bandwidth
% fs = Sampling frequency in Hz
%
%%%%%%%% Outputs ///////////////
%
% b = Numerator filter coefficients in MATLAB form
% a = Denominator filter coefficients in MATLAB form
%
% Mark Wickert, April 2009

```

A.6 Graphic Equalizer Frequency Response Plot

```

function [H,F] = eq_sweep(GdB,Fc,Q)
% eq_sweep(GdB,Fc,Q)

```

```
% Create a frequency response magnitude plot in dB of an NB band equalizer
% using a semilogplot (semilogx() type plot
%
%%%%// Inputs /////////////////////////////////
%
% GdB = Gain vector for NB peaking filters [G1,...,GNB]
% Fc = Center frequency vector assuming fs = 44100 Hz and NB bands
% [fc1,...,fcNB]
% Q = Quality factor vector for each of the NB peaking filters
%
% Mark Wickert, April 2009

fs = 44100; % Hz
NB = length(GdB);
B = zeros(NB,3);
A = zeros(NB,3);

% Create matrix of cascade coefficients
for k=1:NB
    [b,a] = peaking(GdB(k),Fc(k),Q(k),fs);
    B(k,:) = b;
    A(k,:) = a;
end

% Create the cascade frequency response
F = logspace(1,5,1000);
H = ones(size(F));
for k=1:NB
    H = H.*freqz(B(k,:),A(k,:),2*pi*F/fs);
end

semilogx(F,20*log10(abs(H)));
axis([10 fs/2 -10 10]);
grid
set(gca,'XminorGrid','off')
```

A.7 Graphic Equalizer Filter

```
function y = eq_filter(x,GdB,Fc,Q)
% y = eq_filter(x,GdB,Fc,Q)
% Filter the input signal x with an NB band equalizer having input
% parameters GdB, fc, andQ
%
%%%%// Inputs /////////////////////////////////
%
% x = Input signal vector
% GdB = Gain vector for NB peaking filters [G1,...,GNB]
% Fc = Center frequency vector assuming fs = 44100 Hz and NB bands
% [fc1,...,fcNB]
% Q = Quality factor vector for each of the NB peaking filters
%
%%%%// Outputs /////////////////////////////////
%
```

```
% y = Output signal vector
%
% Mark Wickert, April 2009

fs = 44100; % Hz
NB = length(GdB);
B = zeros(NB,3);
A = zeros(NB,3);

% Create matrix of cascade coefficients
for k=1:NB
    [b,a] = peaking(GdB(k),Fc(k),Q(k),fs);
    B(k,:) = b;
    A(k,:) = a;
end

% Pass signal x through the cascade
y = zeros(size(x));
for k=1:NB
    if k == 1
        y = filter(B(k,:),A(k,:),x);
    else
        y = filter(B(k,:),A(k,:),y);
    end
end

test = 0;
```

A.8 Graphic Equalizer GUI

To assist with Problem 4c a MATLAB GUI was put together to study the frequency response of a ten band graphic equalizer. The application name is `equalizer`. To run the application type the function name at the command prompt. To edit the GUI layout you need to run the GUIDE editor.

To make use of this function in Problem 4c you will need to cascade the frequency response of the channel impulse response, so that you can make the cascade response have the desired characteristics.

```
function varargout = equalizer(varargin)
% EQUALIZER M-file for equalizer.fig
%     EQUALIZER, by itself, creates a new EQUALIZER or raises the existing
%     singleton*.
%
%     H = EQUALIZER returns the handle to a new EQUALIZER or the handle to
%     the existing singleton*.
%
%     EQUALIZER('CALLBACK', hObject, eventData, handles,...) calls the local
%     function named CALLBACK in EQUALIZER.M with the given input arguments.
%
%     EQUALIZER('Property','Value',...) creates a new EQUALIZER or raises the
%     existing singleton*. Starting from the left, property value pairs are
%     applied to the GUI before equalizer_OpeningFcn gets called. An
%     unrecognized property name or invalid value makes property application
```

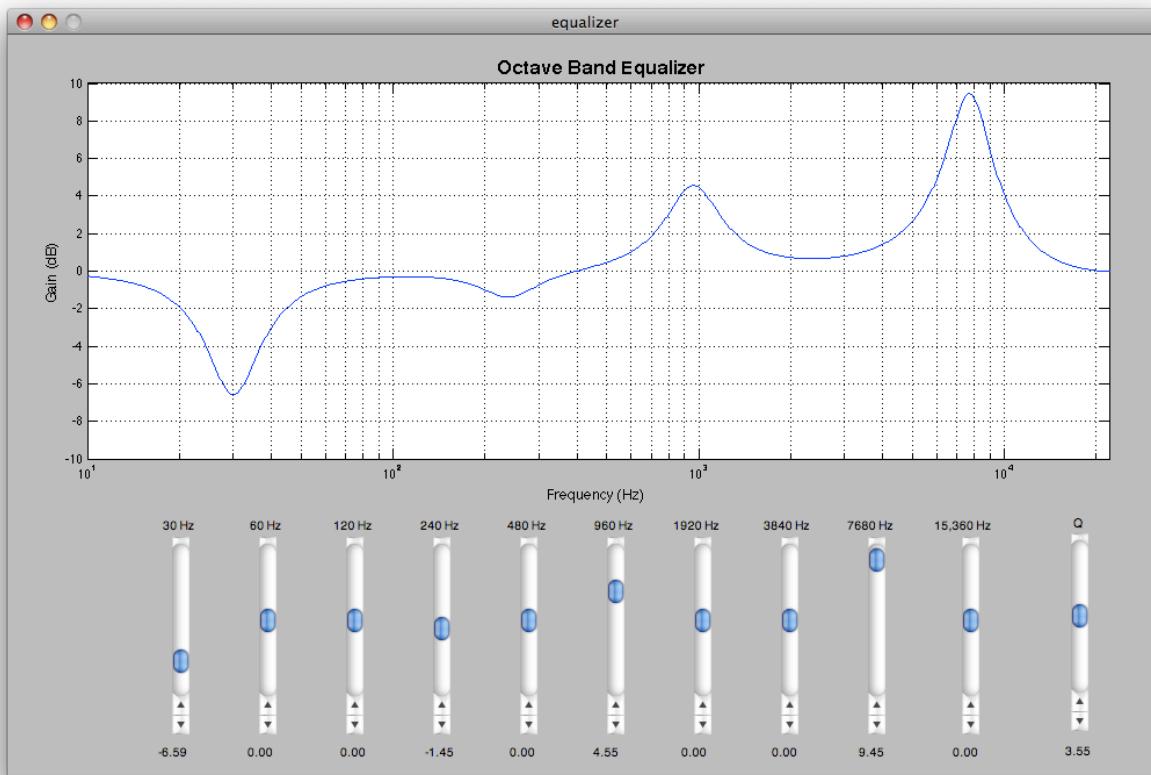


Figure 13: Screen shot of the equalizer GUI application displaying just the frequency response of the ten band octave equalizer alone.

```
% stop. All inputs are passed to equalizer_OpeningFcn via varargin.
%
% *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one
% instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help equalizer

% Last Modified by GUIDE v2.5 24-Apr-2009 16:55:20

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',         mfilename, ...
                   'gui_Singleton',   gui_Singleton, ...
                   'gui_OpeningFcn', @equalizer_OpeningFcn, ...
                   'gui_OutputFcn',  @equalizer_OutputFcn, ...
                   'gui_LayoutFcn', [], ...
                   'gui_Callback', []);
if nargin && ischar(varargin1)
    gui_State.gui_Callback = str2func(varargin1);
```

```

end

if nargout
    [varargout1:nargout] = gui_mainfcn(gui_State, varargin:);
else
    gui_mainfcn(gui_State, varargin:);
end
% End initialization code - DO NOT EDIT

% --- Executes just before equalizer is made visible.
function equalizer_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata   reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to equalizer (see VARARGIN)

% Choose default command line output for equalizer
handles.output = hObject;
% Update handles structure
guidata(hObject, handles);

% UIWAIT makes equalizer wait for user response (see UIRESUME)
% uiwait(handles.figure1);
global g1 g2 g3 g4 g5 g6 g7 g8 g9 g10 Q;
g1 = 0.0; g2 = 0; g3 = 0; g4 = 0; g5 = 0;
g6 = 0; g7 = 0; g8 = 0; g9 = 0; g10 = 0;
Q = 2;
set(handles.slider1,'Value',g1);
set(handles.text1,'String',sprintf('%6.2f',g1));
set(handles.slider2,'Value',g2);
set(handles.text2,'String',sprintf('%6.2f',g2));
set(handles.slider3,'Value',g3);
set(handles.text3,'String',sprintf('%6.2f',g3));
set(handles.slider4,'Value',g4);
set(handles.text4,'String',sprintf('%6.2f',g4));
set(handles.slider5,'Value',g5);
set(handles.text5,'String',sprintf('%6.2f',g5));
set(handles.slider6,'Value',g6);
set(handles.text6,'String',sprintf('%6.2f',g6));
set(handles.slider7,'Value',g7);
set(handles.text7,'String',sprintf('%6.2f',g7));
set(handles.slider8,'Value',g8);
set(handles.text8,'String',sprintf('%6.2f',g8));
set(handles.slider9,'Value',g9);
set(handles.text9,'String',sprintf('%6.2f',g9));
set(handles.slider10,'Value',g10);
set(handles.text10,'String',sprintf('%6.2f',g10));
set(handles.slider11,'Value',Q);
set(handles.text11,'String',sprintf('%6.2f',Q));
plot_response()

```

```
% --- Outputs from this function are returned to the command line.
function varargout = equalizer_OutputFcn(hObject, eventdata, handles)
% varargout cell array for returning output args (see VARARGOUT);
% hObject handle to figure
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout1 = handles.output;

% --- Executes during object creation, after setting all properties.
function axes1_CreateFcn(hObject, eventdata, handles)
% hObject handle to axes1 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called
plot_response()
%eq_sweep([0 0 2 -2],[30 60 120 240],[2 2 2 2])
% Hint: place code in OpeningFcn to populate axes1

% --- Executes on slider movement.
function slider1_Callback(hObject, eventdata, handles)
% hObject handle to slider1 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%         get(hObject,'Min') and get(hObject,'Max') to determine range of slider
global g1;
g1 = get(handles.slider1,'Value');
set(handles.text1,'String',sprintf('%6.2f',g1))
plot_response()

% --- Executes during object creation, after setting all properties.
function slider1_CreateFcn(hObject, eventdata, handles)
% hObject handle to slider1 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes on slider movement.
function slider2_Callback(hObject, eventdata, handles)
% hObject handle to slider2 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%         get(hObject,'Min') and get(hObject,'Max') to determine range of slider
global g2
g2 = get(handles.slider2,'Value');
set(handles.text2,'String',sprintf('%6.2f',g2))
```

```

plot_response()

% --- Executes during object creation, after setting all properties.
function slider2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider2 (see GCBO)
% eventdata   reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes on slider movement.
function slider3_Callback(hObject, eventdata, handles)
% hObject    handle to slider3 (see GCBO)
% eventdata   reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%         get(hObject,'Min') and get(hObject,'Max') to determine range of slider
global g3;
g3 = get(handles.slider3,'Value');
set(handles.text3,'String',sprintf('%6.2f',g3))
plot_response()

% --- Executes during object creation, after setting all properties.
function slider3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider3 (see GCBO)
% eventdata   reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes on slider movement.
function slider4_Callback(hObject, eventdata, handles)
% hObject    handle to slider4 (see GCBO)
% eventdata   reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%         get(hObject,'Min') and get(hObject,'Max') to determine range of slider
global g4;
g4 = get(handles.slider4,'Value');
set(handles.text4,'String',sprintf('%6.2f',g4))
plot_response()

% --- Executes during object creation, after setting all properties.
function slider4_CreateFcn(hObject, eventdata, handles)

```

```
% hObject      handle to slider4 (see GCBO)
% eventdata   reserved - to be defined in a future version of MATLAB
% handles     empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes on slider movement.
function slider5_Callback(hObject, eventdata, handles)
% hObject      handle to slider5 (see GCBO)
% eventdata   reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%         get(hObject,'Min') and get(hObject,'Max') to determine range of slider
global g5
g5 = get(handles.slider5,'Value');
set(handles.text5,'String',sprintf('%6.2f',g5))
plot_response()

% --- Executes during object creation, after setting all properties.
function slider5_CreateFcn(hObject, eventdata, handles)
% hObject      handle to slider5 (see GCBO)
% eventdata   reserved - to be defined in a future version of MATLAB
% handles     empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes on slider movement.
function slider6_Callback(hObject, eventdata, handles)
% hObject      handle to slider6 (see GCBO)
% eventdata   reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%         get(hObject,'Min') and get(hObject,'Max') to determine range of slider
global g6
g6 = get(handles.slider6,'Value');
set(handles.text6,'String',sprintf('%6.2f',g6))
plot_response()

% --- Executes during object creation, after setting all properties.
function slider6_CreateFcn(hObject, eventdata, handles)
% hObject      handle to slider6 (see GCBO)
% eventdata   reserved - to be defined in a future version of MATLAB
% handles     empty - handles not created until after all CreateFcns called
```

```
% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes on slider movement.
function slider7_Callback(hObject, eventdata, handles)
% hObject    handle to slider7 (see GCBO)
% eventdata   reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%         get(hObject,'Min') and get(hObject,'Max') to determine range of slider
global g7
g7 = get(handles.slider7,'Value');
set(handles.text7,'String',sprintf('%6.2f',g7))
plot_response()

% --- Executes during object creation, after setting all properties.
function slider7_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider7 (see GCBO)
% eventdata   reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes on slider movement.
function slider8_Callback(hObject, eventdata, handles)
% hObject    handle to slider8 (see GCBO)
% eventdata   reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%         get(hObject,'Min') and get(hObject,'Max') to determine range of slider
global g8
g8 = get(handles.slider8,'Value');
set(handles.text8,'String',sprintf('%6.2f',g8))
plot_response()

% --- Executes during object creation, after setting all properties.
function slider8_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider8 (see GCBO)
% eventdata   reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end
```

```
% --- Executes on slider movement.
function slider9_Callback(hObject, eventdata, handles)
% hObject    handle to slider9 (see GCBO)
% eventdata   reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%         get(hObject,'Min') and get(hObject,'Max') to determine range of slider
global g9
g9 = get(handles.slider9,'Value');
set(handles.text9,'String',sprintf('%6.2f',g9))
plot_response()

% --- Executes during object creation, after setting all properties.
function slider9_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider9 (see GCBO)
% eventdata   reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes on slider movement.
function slider10_Callback(hObject, eventdata, handles)
% hObject    handle to slider10 (see GCBO)
% eventdata   reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%         get(hObject,'Min') and get(hObject,'Max') to determine range of slider
global g10
g10 = get(handles.slider10,'Value');
set(handles.text10,'String',sprintf('%6.2f',g10))
plot_response()

% --- Executes during object creation, after setting all properties.
function slider10_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider10 (see GCBO)
% eventdata   reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes on slider movement.
function slider11_Callback(hObject, eventdata, handles)
```

```
% hObject      handle to slider11 (see GCBO)
% eventdata   reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%         get(hObject,'Min') and get(hObject,'Max') to determine range of slider
global Q
Q = get(handles.slider11,'Value');
set(handles.text11,'String',sprintf('%6.2f',Q))
plot_response()

% --- Executes during object creation, after setting all properties.
function slider11_CreateFcn(hObject, eventdata, handles)
% hObject      handle to slider11 (see GCBO)
% eventdata   reserved - to be defined in a future version of MATLAB
% handles     empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

////////////////// Custom Function for Updating Plot/////////////////
function plot_response()
///////////////////////////////
global g1 g2 g3 g4 g5 g6 g7 g8 g9 g10 Q;
%Q = 2;
[H,F] = eq_sweep([g1 g2 g3 g4 g5 g6 g7 g8 g9 g10],...
    30*2.^[0:9],Q*ones(1,10));

///////////////////
% Add code here to allow you to graphically solve problem 4c
///////////////////

semilogx(F,20*log10(abs(H)));
grid
%set(gca,'XminorGrid','off') % turns off minor y grid
axis([10 44100/2 -10 10])
xlabel('Frequency (Hz)', 'FontSize',12)
ylabel('Gain (dB)', 'FontSize',12)
title('Octave Band Equalizer', 'FontSize',16, 'FontWeight','bold')
///////////////////
```

A.9 Schroeder's Reverb

To implement Schroeder's reverb two supporting functions were written: plain_reverb(x,a,D) and ap_reverb(x,a,D). The support functions will be listed first, followed by the top level reverb function, sc_reverb.m.

```
function y = plain_reverb(x,a,D)
% y = plain_reverb(x,a,D)
% Plain reverb or comb filter section.
%
```

```
% x = Input signal vector
% a = Feedback gain
% D = Feedback delay in samples
%
% y = Output signal vector
%
% Mark Wickert, April 2009
```

```
y = filter(1,[1 zeros(1,D-1) -a],x);
```

```
function y = ap_reverb(x,a,D)
% y = ap_reverb(x,a,D)
% All-pass reverb section.
%
% x = Input signal vector
% a = Feedback gain
% D = Feedback delay in samples
%
% y = Output signal vector
%
% Mark Wickert, April 2009
```

```
y = filter([-a zeros(1,D-1) 1],[1 zeros(1,D-1) -a],x);
```

```
function y = sc_reverb(x,a,D,b)
% y = sc_reverb(x,a,D,b)
% Schroeder's reverberator filter controlled by input parameters a, D, and
% b, with the sampling rate fixed at 44,1000 Hz.
%
%//////// Inputs /////////////////////////////////
%
% x = Input signal samples
% a = Feedback parameter vector [a1,a2,a3,a4,a5,a6]
% D = Delay vector [D1,D2,D3,D4,D5,D6]
% b = Gain combining coefficients from plain reverberator parallel paths
%     [b1,b2,b3,b4]
%
%//////// Outputs /////////////////////////////////
%
% y = Output signal vector
%
% Mark Wickert, April 2009
```

```
% Parallel bank of plain reverb sections
z = b(1) * plain_reverb(x,a(1),D(1));
z = z + b(2) * plain_reverb(x,a(2),D(2));
z = z + b(3) * plain_reverb(x,a(3),D(3));
z = z + b(4) * plain_reverb(x,a(4),D(4));

% Cascade of all-pass reverb sections
```

```
%z = x;
y = ap_reverb(z,a(5),D(5));
y = ap_reverb(y,a(6),D(6));

test = 0;
```

A.10 Flanging Processor

Note that this is a time-varying filter. Due to the fact that a continuously variable interpolator is involved (needed for fractional delays), the function executes rather slowly. A MEX (MATLAB executable function) version may be available to speed up execution for specific platforms (PC, MAC, & LINUX).

```
function y = Flanging(x,a,fD,fs,Dpp)
% y = Flanging(x,a,fD,fs,Dpp)
% Apply the audio special effect flanging to the signal vector x.
%
%/////////// Inputs ///////////
%
%   x = Input vector (may be two columns if a stereo signal)
%   a = Gain coefficient for combining variable time delayed signal
%   fD = Time delay sinusoidal frequency in Hz (1 Hz is good)
%   fs = Sampling frequency in Hz
%   Dpp = Maximum peak-to-peak delay variation
%
%////////// Output ///////////
%
%   y = Output signal vector = x + x[n - D(n)]
%
% Mark Wickert, April 2009

Dmax = Dpp;
n = 0:max(size(x))-1;
D = Dmax/2*(1 - cos(2*pi*fD/fs*n));
D = D + 2; % two sample delay need for the interpolator

% Make sure tapped delay line is long enough
N = Dmax + 4;

y = zeros(size(x));
X = zeros(1,N+1);
% Farrow filter tap weights
W3 = [1/6 -1/2 1/2 -1/6];
W2 = [0 1/2 -1 1/2];
W1 = [-1/6 1 -1/2 -1/3];
W0 = [0 0 1 0];

for k=1:length(x)
    Nd = fix(D(k))+1;
    mu = 1 - (D(k)-fix(D(k)));
    X = [x(k) X(1:end-1)];
    % Filter 4-tap input with four Farrow FIR filters
    v3 = W3*X(Nd-1:Nd+2).';
    v2 = W2*X(Nd-1:Nd+2).';
```

```
v1 = W1*X(Nd-1:Nd+2).';  
v0 = W0*X(Nd-1:Nd+2).';  
%Combine sub-filter outputs using mu = 1 - d  
y(k) = ((v3*mu + v2)*mu + v1)*mu + v0;  
end  
  
y = (1 - a)*x + a*y;
```