

EE432 Fall 09 Project 05: Filter Design and Implementation (Due: 10/13/09)

In this lab, you will learn about designing frequency filters using MATLAB, and then how to filter music or other audio signals with the filters you design.

I. Plucked String Filter

1. Design a plucked string filter with $L=100$ and $R=0.999$. Determine the transfer function $H(z)$ in positive powers of z . Use *fvtool* to plot the magnitude response and the pole-zero plot.
2. Random numbers are used as an input to the filter to generate the sound of the plucked string. We will use a sample frequency of 8000 Hz, and your filter from step 1. Create an input signal consisting of $L=100$ random numbers (use *randn* to create them) followed by enough zeros to make the signal last for one second. Use *filter* to filter this input with your plucked string filter.
3. Use *wavplay* or *soundsc* to play your filter's output. Don't forget to use the correct sample frequency. Does it sound like a plucked string? What is the fundamental frequency of the note that is played? Plot the output note versus time (in sec) in a well annotated plot and turn it in with your report. Use *wavwrite* to write this out as a sound file and email it to the professor. Call the file "note1.wav".
4. Now try a different note. Create a filter as you did in step 1, but use $L=100$ and $R=0.9999999999999999$. What will be the fundamental frequency of the note? Create the correct input to the filter and the note, then use *wavwrite* to write out the note, and send it to the professor. Call the file "note2.wav". Plot the output note versus time (in sec) in a well annotated plot and turn it in with your report. What are the audible differences between note1 and note2?

II. Filter Design using *fdatool*:

1. In MATLAB, start up the Frequency Design and Analysis tool by typing *fdatool* at the command line.
2. Create two separate FIR frequency filters using the following specifications:

LPF: FIR, Equiripple in the passband and stop band, minimum order, filter order 80.
Frequency specifications: sample frequency 44.1 kHz, $F_{\text{pass}} = 2500$ Hz, $F_{\text{stop}} = 4000$ Hz.
Magnitude specifications: $W_{\text{pass}} = 1$ dB, $W_{\text{stop}} = 80$ dB

Export these filter coefficients using *File*→*Export* with the variable name *lpf*.

HPF: FIR, Equiripple in the passband and stop band, minimum order, filter order 80.
Frequency specifications: sample frequency 44.1 kHz, $F_{\text{stop}} = 2500$ Hz, $F_{\text{pass}} = 4000$ Hz.
Magnitude specifications: $W_{\text{pass}} = 1$ dB, $W_{\text{stop}} = 80$ dB

Export these filter coefficients using *File*→*Export* with the variable name *hpf*.

3. Choose one of the professor's music clips, or choose one of your own and bring it into MATLAB using *wavread*. Remember to also import the sample frequency. Use *wavplay* to listen to the music. From now on, when using the *wavplay* function, make 'async' the last input variable to *wavplay*...this will allow the music to play in the background while you do other MATLAB work.
4. Filter your music using your low pass filter coefficients as one input to the *filter* function, then listen to the filtered result. Does the sound you hear make sense (does it sound as if the higher frequencies are removed)?
5. Filter your music using your high pass filter coefficients as one input to the *filter* function, then listen to the filtered result. Does the sound you hear make sense (are low frequencies removed)?
6. Note that the high pass and low pass filters combined make something very close to an all-pass filter. If that is the case, then when you add the high-pass filtered music to the low-pass filtered music, and play the sum of the two, it should sound like the original music clip. Does it?
7. The impulse response of an all-pass filter is just an impulse (a delta function), as we'll see in class lectures soon. Determine if the sum of the impulse responses (coefficients) of the HPF and the LPF together gives the impulse response of an all-pass filter. Use the *stem* function to plot this sum. Turn in this plot and describe your result.
8. Now add the two filters' impulse responses together to form a new filter, and filter the music with the sum of these two impulse responses. If this is an all-pass filter, the music should sound like the original. Does it?

Graphic Equalizer:

You have probably seen a graphic equalizer somewhere...your car has something like a graphic equalizer...you can adjust the bass (low frequencies) and the treble (higher frequencies). In a true graphic equalizer, you can adjust the amplitude of a number of ranges of frequency information as the music is played, most using a slide control as in Figures 1 and 2 below. In this project, we keep it simple by modeling a 4-band equalizer only, but not in real-time.

9. Emphasize the bass: take the LPF filter coefficients and multiply them by 1.0, and deemphasize the treble by multiplying the HPF coefficients by 0.20. Add these scaled impulse responses together, filter the music, then listen to the result. Is the bass emphasized, with weak treble?
10. Emphasize the treble: take the LPF filter coefficients and multiply them by 0.20, and deemphasize the treble by multiplying the HPF coefficients by 1.0. Add these scaled impulse responses together, filter the music, then listen to the result. Is the treble emphasized, with weak bass?
11. Design a 4-band equalizer for your music. Use FIR filters that have passbands of 0-2 kHz, 2-4 kHz, 4-8 kHz, and 8 kHz and higher. Don't forget that your F_{pass} and F_{stop} settings should match up. Demonstrate this to your professor in class using the filtering techniques from this project. Print out the 4 magnitude responses and turn them in with your writeup, annotating which filter is which.

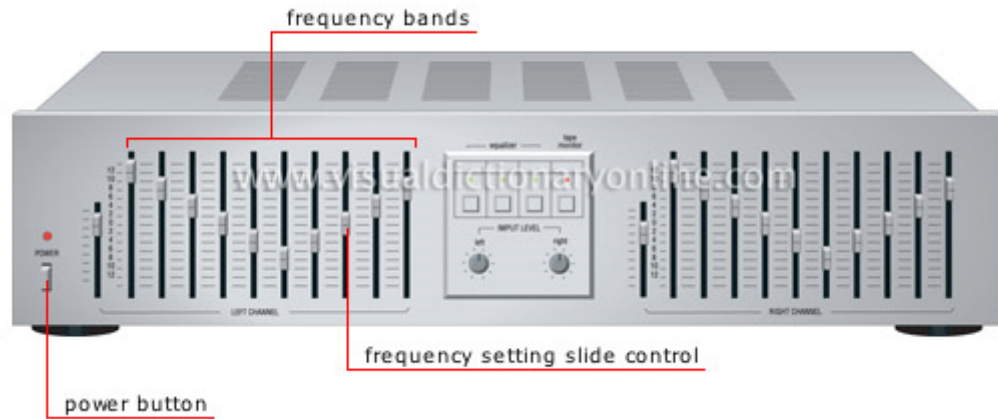


Figure 1: Graphic Equalizer

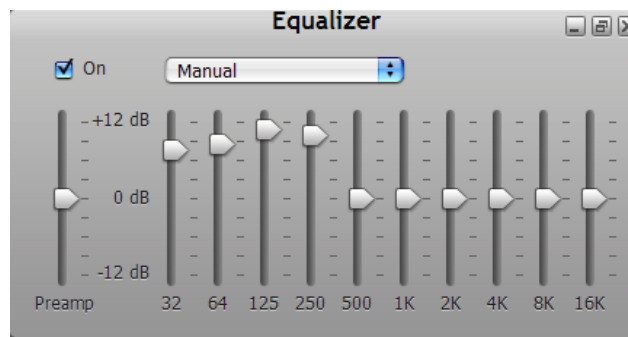


Figure 2: iTunes Graphic Equalizer

For this project, answer all questions, turn in all plots asked for, and email all way files asked for.