Lab 2: Filter Design and Usage

For this lab you will learn how to design some simple filters and how to apply them to solve some common audio problems. Python's `scipy.signal` package has an extensive set of commands to help you design filters (`firwin`, `firwin2`, `butter`, `cheby1`, `cheby2`, `ellip`, ...), so there is no shortage of options.

Part 1. When to use what

There will be four problems to this part, each requiring a different type of filter to address various issues in an audio signal. To start with, you should use FIR filters and the `firwin()` and/or `firwin2()` functions. Once you do that you should write code that designs the necessary filters without using the filter design functions (e.g. use a windowed sinc function for lowpass, etc). Finally you should use IIR filters (use the `scipy.signal` filter design functions for these). Whenever you design a filter you can the `freqz` command to obtain its response which you can plot and see if it is what you intended. To apply the filter you can use `scipy.signal.convolve` or `scipy,signal.lfilter`

You will find the sounds to process in this lab's zip file.

case1.wav, case2.wav: In these cases we have a corrupted speech signal. Listen to each soundfile and try to identify what might be wrong with it. Plot their spectrograms and see if you can get a more specific idea of what the problems are. Once you identify the problems, design the necessary filters that will improve the intelligibility of these recordings.

case3.wav: This signal contains some bird songs during a thunderstorm. Alas, as a world renowned ornithologist you need to have a cleaner recording of the the bird songs for further analysis. Find out how to clean up the sound and remove the thunder sounds with a filter.

case4.wav: The signal that we require to extract here is a Morse code which is buried in environmental noise. Design a filter to bring out the beeps.

Make some observations on how the results differ between an FIR and IIR filter and try to find the best possible filter size/type/parameters to produce the best result. Show results under various parameters (e.g. filter length) and show us some plots that demonstrate the effects of these parameters. Most importantly, try to get a sense of how the design choices you make sound. Being able to listen at a sound an identify what's wrong and how to fix it is a big part of audio processing.

Part 2. Designing a simple equalizer

For this part we will design a simple graphic equalizer. We will do so using a more straightforward approach as opposed to a bank of filters as discussed in class.

We want to make an equalizer which contains six bands with center frequencies at 100Hz, 200Hz, 400Hz, 800Hz, 1600Hz and 3200Hz. Your equalizer function will take two inputs, one for the input sound and a 6-element gain vector that will indicate how much to boost or suppress each frequency band. Use the `firwin2` function to design a filter that has the desired characteristics. For various settings of the gain vector use the `freqz` command to plot the response of the filter and verify that it behaves as indicated. Experiment with various filter lengths and see which works best.

Optional extra credit: Design a graphic equalizer with as many bands as you like (and arbitrary center frequencies as well). Apply it on the problems of part 1.