**Project**: Morse code envelope extraction and decoding

**Goal**: write a code that processes audio (mp3) files to extract Morse code messages

**Background**: In many types of communications, it’s typical to embed a message on a higher-frequency ‘carrier’ frequency. One example is AM radio, where the amplitude modulation of the radio station’s broadcast frequency carries the information about the music or speech being transmitted. Another example is audio signals generated with Morse code, where the ‘dot’ or ‘dash’ information basically encoded in turning an audio tone turned on or off.

**Data source:**

The website <http://www.arrl.org/15-wpm-code-archive> has a bunch of mp3 audio files for Morse code, along with the text, for ham radio operators (and others) to use in training. I downloaded one of these, 150107\_075WPM.mp3.

**Basic steps:**

The basic steps would be:

1. Read the mp3 file into matlab, using the audioread command. This returns both the data (in a vector) and the sampling rate.
2. For debugging, you may want to pull out a smaller vector consisting of, say, the first 5 seconds of data. You can use the sampling rate to figure out how many samples that is.
3. Plot the data, and listen to it using soundsc. You should be able (and hear) to see periods of silence and then ‘dot’ and ‘dash’ periods. If you zoom in on one of the ‘dash’ or ‘dot’ periods, you’ll see the sine wave that is audio tone.
4. To convert this audio data back into dashes and dots, you need to extract the envelope of the signal. There are quite fancy ways to do this (Hilbert transform) but it’s better to keep things simple.
   1. For some background, see: <http://dsp.stackexchange.com/questions/1522/simplest-way-of-detecting-where-audio-envelopes-start-and-stop> and read the answer starting with 'This is the classic problem of speech detection. First thing to...."
5. A straightforward implementation of envelope extraction would be:
   1. Square the signal, so all values are positive
   2. To estimate the amplitude, smooth the squared signal using a moving average filter. One way of writing the equation for a moving average filter is:

In other words, to get the filtered output at a time sample k, average the (2N+1) values of the input x from samples k-N to k+N. For this case, a good first guess might be to average over something like 0.02 seconds. You can figure out how many samples this is by using the sampling rate.

1. After you have done the step above, plot your estimate of the amplitude. Can you see fairly clearly where the dashes and dots are?
2. Next, compare the data to some amplitude threshold (which you’ll have to choose). This should give a logical vector as output that is 1’s during dashes and dots, and zeros elsewhere.
3. Now, you need to write some code to identify dashes and dots. You can do this by counting how many samples in a row stay above the threshold. You'd need to define how many samples indicate ‘dot’ and how many indicate ‘dash’.

**Some extensions**

The above may well be enough, but some “stretch” ideas might include:

1. Can you reliably identify the longer gaps of silence between letters? If so could identify that a particular letter is a ‘dash-dot-dot’, for example.
2. If you can do #1, can you do a lookup table to convert the dots and dashes back to text?
3. For the basic idea, you’d pick all the timing thresholds (how long a dot is, how long a dash, etc) by hand. Can you automate this process?
4. some further ideas if you are looking to "stretch" this project
5. add noise using randn, as described in Lecture 22 starting around slide 31. Measure the SNR and see how your accuracy in interpreting the Morse code drops off as noise increases.
6. 2) After you add noise, try also removing it using the data smoothing techniques like the moving average filter we’ve looked at. Does this let you get back some of your performance?