Creative Computing II
Signals and Systems: Applying Audio Effects
2nd February 2011

This lab sheet covers the application of digital audio effects. The instructions given are phrased in terms of *Octave*'s functionality; if you prefer, you may use *Processing* to accomplish the same goals – footnotes are included to indicate Minim analogues to the *Octave* functionality – but the *Octave* support for this kind of operation is more comprehensive.

1. This part helps to explain the return value of *Octave*'s `fft` operator, using as source material a real musical signal.

   (a) Obtain the provided `scale.wav` file, and use the `wavread` and `sound`[1] *Octave* functions to play it. What is the sampling rate for that sound file?

   The sampling frequency is 44.1kHz. You can get this, along with the bits per sample, by specifically asking `wavread` *for this information; you get just the audio data with a call like*

   ```
   y = wavread('scale.wav');
   ```

   but you get sampling frequency and bits per sample with

   ```
   [y, fs, bps] = wavread('scale.wav');
   ```

   This information is given in the "usage" section of the online help for Octave operators (e.g. from `help wavread`*).*

   (b) Plot the first thousand samples of the first channel. Can you determine any apparent regularity in your plot? If so, estimate the period of the regularity in samples and hence the frequency of that regularity.

   If the variable `x` *contains the audio data, then* `plot(x(1:1000,1))` *will plot the first thousand samples. You should be able to see, after the first 100 samples or so, that there is a repeating pattern about every 170 samples (look at the peaks at samples 186, 358, 532, 700, 864, for example). 170 samples at a sample rate of 44.1kHz is a period of 3.85 milliseconds, corresponding to a frequency of 259Hz.*

   (c) Take the Fourier Transform[2] of the first thousand samples of the first channel, and plot the magnitude of it. Note the symmetry around the mid-point.

   ```
   f = fft(x(1:1000,1)); plot(abs(f))
   ```

   (d) Plot the magnitude of the Fourier Transform you obtained over the first 100 frequency bins, and identify the bin number of the highest peak.

   You should be able to see clearly from the plot that the 7th vector element has the highest-magnitude peak.

   (e) The bins are in ascending order of frequency, each corresponding to an integer multiple of $\frac{1}{T}$, where T is the total time of the signal whose transform was taken. Compute the frequency corresponding to the bin identified in part 1d

[1] If `sound` is not working, find some alternative way of listening to the audio file.

[2] Minim provides an `FFT` class which performs Fast Fourier Transforms.

The 7th vector element corresponds to 6 times the fundamental frequency (remember that the 1st vector element contains the d.c. component). The total time of the signal whose transform was taken was the time for 1000 samples, or 0.02268 seconds, so the high magnitude peak corresponds to a frequency of $6 \times \frac{44100}{1000}$ *or 265Hz. That this is very close to the answer in part 1b is no accident; the Fourier Transform decomposes a signal into its regular components, and so any obvious regularities in the signal will correspond directly to a high peak in the frequency spectrum.*

(f) Repeat parts 1b to 1e with the thousand audio samples starting at 39001, 78001, 117001, 156001, 195001, 234001 and 273001. You may wish to write functions to automate some of this work. Note all the frequencies that you obtain, and attempt to relate them to the musical scales discussed last term.

You should find that the frequencies of those little samples of audio correspond to the notes of a musical scale, having frequency ratios approximating powers of $\sqrt[12]{2}$.

2. This part illustrates the effects of basic filtering techniques on simple musical signals.

   (a) Using the same `scale.wav` example file, take the Fourier Transform of an entire channel's signal. Verify that taking the inverse Fourier Transform (with `ifft`[3]) produces the original signal.

   (b) Removing a particular frequency component from a signal is equivalent to setting its Fourier component to zero. Do this for all frequency bins between 3001 and 311012 (inclusive).

   `f(3001:311012) = 0.`

   (c) Listen to the sound produced by taking the inverse Fourier Transform of your modified spectrum, and comment on the relationships between the new sound, the old one, and the frequency components removed.

   The signal's duration is about 7.1 seconds, so the 3001th frequency bin in the Fourier Transform corresponds to a frequency of about 421Hz. This filtering will therefore remove all frequency components above 421 Hz, cutting out some of the later notes of the scale completely, and altering the overtones of the notes that are passed.

3. This part takes you through applying the acoustics of a highly reverberant space (York Minster) to a simply synthesized piece of musical audio.

   (a) First, we need an impulse-response for York Minster. Visit `http://space-net.org.uk/node/54` and download the 'Stereo - ORTF' file; listen to it, and think about how that sound relates to an impulse-response.

   (b) Read it into *Octave* using `wavread`, using the way of calling `wavread` which tells you the sample rate of the wave data; note that sample rate.

   The sample rate for this file is 48kHz.

   (c) Obtain `toccata.wav` from the teaching website, and read that file into *Octave* using the `wavread` function, again noting its sample rate.

[3] In Minim, use the `FFT.inverse()` method.

The sample rate for this one is 44.1kHz.

(d) You should find that the two audio files were sampled at different rates. In order to treat one as the input to the other treated as a system, they must have the same sample rate. Use the `resample` `Octave` operator to convert one of the signals to the other's sample rate.

`resample(toccata, 48000, 44100);` will resample (upsample) the 44.1kHz audio to a sample rate of 48kHz. To downsample the minster impulse-response, you would need to swap the order of the second and third arguments to `resample`.

(e) Try convolving[4] the two signals together to produce the York Minster system's output from the input musical signal. How long are you prepared to wait?

In this instance, both sequences are long: of the order of a million samples. Doing a direct convolution of the two thus involves a million million operations.

(f) Use the Fourier Transform method for implementing convolution to apply the York Minster acoustic to the musical signal. How long did the computation take?

With the Fourier Transform method (from lecture slides), you should find that the computation is fast: no more than a few seconds. Because the Fourier Transform method is $O(N \log N)$ – rather than $O(N^2)$ for direct convolution – it can be used on large input data.

(g) Use `wavwrite` to write out the resulting data as an audio file, and listen to it using a media player.

You should have a much more realistic-sounding rendition of the opening of Bach's Toccata in D minor – as if performed under the tower of York Minster.

Other Resources:

- Spatial Audio Creative Engineering Network. `http://space-net.org.uk/`
- Stereo Room Impulse Responses, from Rebecca Stewart, Queen Mary, University of London; from University of London External System.
`http://www.londonexternal.ac.uk/current_students/programme_resources/cis/news/docs/Stereo%20IRs.zip`

[4] Minim provides a `Convolver` class which performs direct convolution.