Tutorial 5: Audio Signals

1. More on Sampling and Quantising

Sampling is the process of selecting regularly spaced points in time in which to record the level of a signal. Quantisation is the process of turning a continuous or analogue signal into one which has discrete numerical values at each point in (for instance) time. Voice was sampled for telephone at 8,000 samples/sec each sample quantised with 8 bits. In audio applications, a voltage is typically quantised into a 16-bit number, that can represent 65536 different levels of voltage. The numbers are usually proportional to the input voltages, but logarithmic relationships have also been used. It is important to note that quantisation is a separate process from sampling; after quantisation the amplitude of the signal is still represented continuously in time.

The previous sampling rates were selected because it allows us to reconstruct the original signal with adequate precision. Nyquist's theorem says that if we sample a signal in which the highest frequency we wish to reproduce correctly is ***f*** then we must sample the signal at a minimum frequency of ***2f***. For this sampling rate, the frequency ***f*** is known as the *Nyquist* frequency, f_N.

A signal which is sampled below this minimum rate is *undersampled*, and the effect of doing this is to generate spurious contributions of lower frequencies in the reconstructed signal. This process is known as *aliasing*.

Analogue-to-digital conversion is the overall process of taking an analogue signal, such as the fluctuating voltage from a microphone and turning it into a digital signal, consisting of a stream of numbers.

In the previous Laboratory Experiment you analysed sampling and quantising images. Quantising and sampling follow the same rules for audio and images, both are considered as *signals*. Now we will deal with 1-D signals.

2. Hands - on sampling and quantising.

First, we will visually see the implications of sampling and quantising. Down load the programme *aToD.m* from the web page. Save it on your local directory and on the matlab promt run:

```
>> aToD
```

A figure will appear with an Analogue to Digital conversion programme. An image of appears in figure 1. Basically, you can select from 3 types of signals, sine wave, square wave and triangular wave, you can modify the frequency of the signal, you can select the sampling frequency and you can select the number of bits for quantising. With this parameters you can convert the signal from analogue to digital and back.

- Try different signals, frequencies and number of bits. Can you see aliasing?
- How do the number of bits introduce noise?

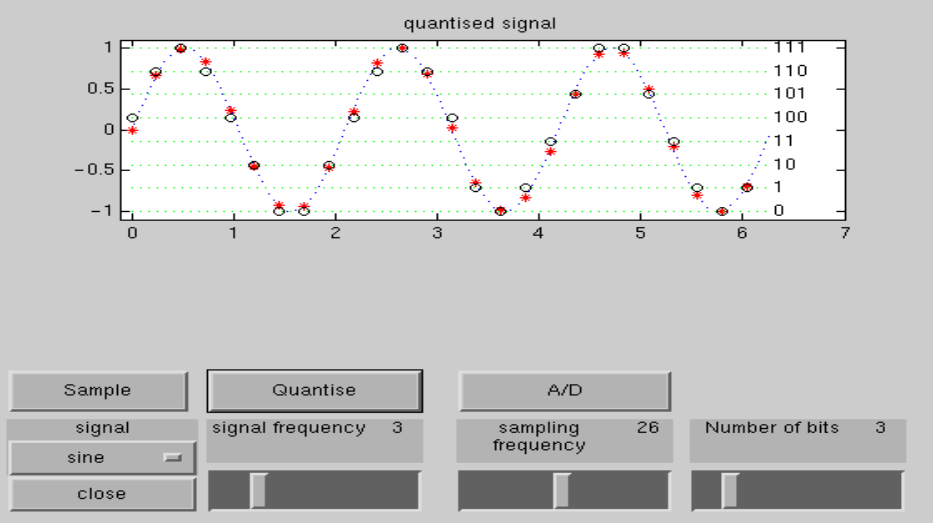

Figure 1

3. Sampling and quantising audio signals

Matlab is useful and powerful while handling audio files. You can visualise and hear an audio signal in the following way.

```
>> load handel
```

This loads a (tiny) bit of the *Messiah* by *Handel*, the well-known *Hallelujah*. (If you are interested in this masterpiece here is some information: http://en.wikipedia.org/wiki/Messiah_%28Handel%29). You will notice that two variables are loaded, *Fs* = 8192 samples/s, the sampling rate and *y* a 73113 x 1 array. This looks something like:

5.1 To play this, (provided that your computer has the proper output, some of the computers in the MSc room do not have loud speakers, try in the undergraduate machines or if you have your own computer with matlab do it at home), type:

```
>> sound(y*0.5)
```

5.2 The *0.5* value controls the volume, (try not to disturb other users!). You can manipulate *y* as any other matrix, try for instance:

```
>> sound(y(end:-1:1)*0.5);      % Will play the sound backwards
>> sound(y(1:2:end)*0.5);       % Plays half of the samples
>> sound((y>0)*0.5);            % Plays only positive part
```

5.3 Now let's quantise the signal. Use the *uencode* command for quantising. Besides you will need to keep the values between -1 and 1:

```
>> y2=double(uencode(y,2))/1.5-1;
```

That will quantise to 2 bits, i.e. 4 levels, what quality can you expect? Try now quantising with more bits of resolution, at which point does it *sounds* as the original? Would this be the same if you would have a high fidelity equipment? Figure 3 shows the difference between the original signal and the sampled signal.

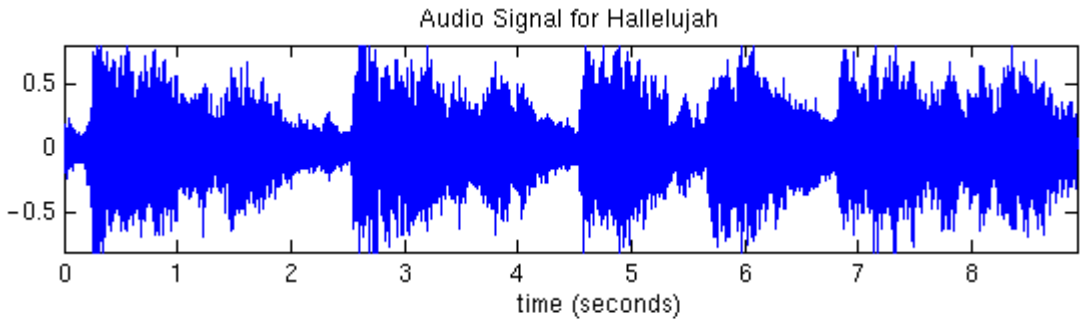

Figure 2

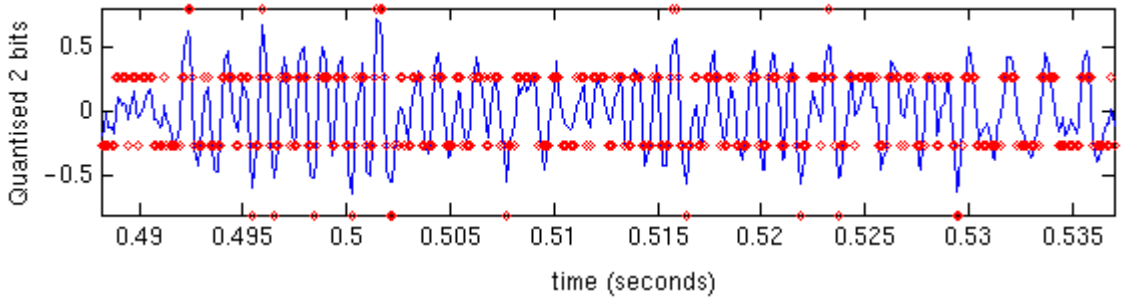

Figure 3

5.4 There are several ways to reduce the number of samples, an easy one would be to decimate (sub-sampling) the signal and then interpolate it (to keep it in the same pitch):

```
>> x2=interp(decimate(y,2),2);  % Down sample by 2
>> x4=interp(decimate(y,4),4);  % Down sample by 4
```

Play the signals and listen to them carefully, can you notice the degradation? Increase the down sampling and listen. Figure 4 shows a fragment of the interpolated signal.

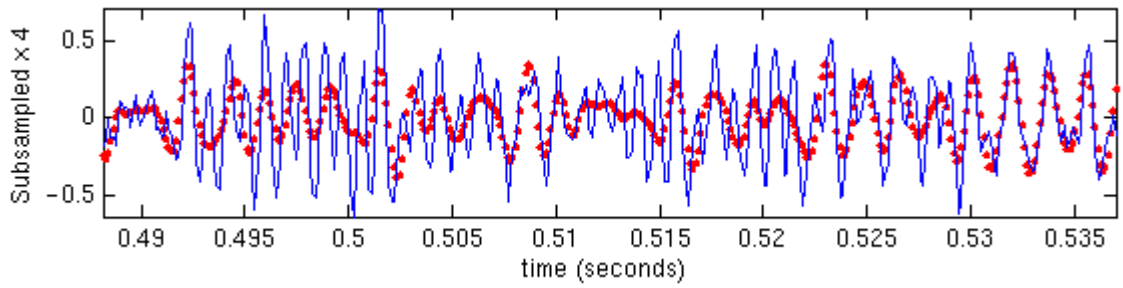

Figure 4

5.5 The previous instructions perform a linear interpolation, how could you interpolate using *nearest neighbours*? Try it and listen the difference, which one sounds better.