SF1: Data Analysis - Statistical Signal Processing with Application in Audio

Week 1 Report

Joseph Johnson

May 2021

1 Introduction

Contained within this first short report are text answers to the questions posed in the week 1 section of the project handout, plus supplementary figures and code. Topics covered include the discrete (and fast) fourier transform, windowing and spectrum analysis.

2 Answers to the tasks

2.1 Task 1

My implementation of the discrete fourier transform (dft) in python utilises the matrix form of the dft:

$$\underline{X}(\omega) = W\underline{x}(t)$$

where W is the fourier matrix containing appropriate powers of the complex exponential, $\omega = \exp(-j\frac{2\pi}{N})$, such that the p^{th} element of the column vector $\underline{X}$ is precisely X_p.

There are some important notes to consider with this implementation. First and foremost, the use of vectorisation along with library functions provides significant speed-up over a nested 'for' loop approach to this calculation. There is, however, one major issue - the number of elements in the fourier matrix scales with N^2. Considering the case where each element of W is a complex number stored in double form (i.e. 8 bytes for the each the real and imaginary component), and there is a modest 8GB of RAM available, then the maximum data length is around 23,000 samples. This may be inadequate for audio analysis where 23,000 samples represents around 25ms of stereo audio.

2.2 Task 2

To test the accuracy, 100 complex numbers were drawn from a standard gaussian and checked against the numpy fft function with a python assert statement. Further to this, timing was done using the inbuilt 'time' library in python. There was some difficulty in providing good timing data for plotting purposes as the fft scales much more gracefully with N than the DFT - $O(NlogN)$ vs $O(N^2)$, meaning the compute time for the dft quickly becomes too long.

2.3 Task 3

Demonstration of the important results in task 3 requires a significant number of plots, all of which are log spectral magnitude plotted against normalised frequency.

The important takeaways from figure (3) are:

- Cosine and sine waves of the same frequency have very similar spectra, with their main lobes in the same position. Furthermore, they are the sum of two complex exponentials at each of the positive and negative frequencies.
- The main lobe is centred on the normalised frequency.
- An insufficiently long window does not capture any frequency behaviour and appears very flat on the spectral plot. Increasing the window length causes the spectrum to tend to a spike at the normalised frequency (i.e. a fourier series).
- Window choice has a strong effect on the spectrum. This decision will depend on the application, but some options are choosing for the largest main-lobe to first side-lobe distance, or strongest energy concentration in the main lobe.

2.4 Task 4

Figure (4) contains several spectra for sinusoidal data corrupted by additive white gaussian noise.

Immediately obvious is that non-zero mean noise has created a spectral peak at zero frequency - unsurprising considering this is a constant 'dc' component. There is also no extra peak for the zero mean case. In addition, the erratic sample white noise spectrum has been superimposed, perturbing the spectra at all frequencies. In the presence of strong enough noise the main-lobes of the true signal have been hidden.

2.5 Task 5

Figures (5), (6), (7) and (8) contain some spectra for signals of the form:

$$y_n = a_n e^{-j\omega n}$$

Figure (6) demonstrates the increase in signal power for $A, B \geq 1$. Further to this if $A \geq B$ then the energy is focused in the main-lobe and the side-lobe ripples dramatically decrease as the signal puts most of its energy into $ne^{j\omega n}$ as opposed to $e^{j\omega n}$. Figure (7) shows how two extra peaks occur symmetrically around the main-lobe due to the oscillating amplitude modulation, their positions and magnitudes are controlled by ϕ and β respectively. Using a gaussian AR process for amplitude modulation, as in figure (8) leads to a noisy signal spectrum, though this time it does not seem additive. Again, if the noise has high enough power (i.e. variance) then the main-lobe becomes hidden.

2.6 Tasks 6&7

Samples for windowed spectral analysis were taken from real audio. They were isolated using Audacity from a song on my local drive in .wav format. I tried to cover a range of sounds, including a bass transient note; a steady chord; and a 'th' and a 'ma' sound, both in the presence of noise. The relevant graphs are shown in figures (9), (10), (11) and (12)

These spectra are quite complex but also very rich with information. The most prominent peaks represent the primary frequency components of the signal, but this analysis cannot be perfect due to the presence of noise and harmonics. As demonstrated earlier, noise can hide peaks, or even create peaks where we would note expect. Noise in our audio signal manifests itself as imperfections in recording equipment, as well as 'artistic' noise in the music. To read from these spectra, we can convert from normalised frequency to a true frequency using the formula:

$$f = \frac{N\theta}{2\pi}$$

where N is the number of samples before windowing, and θ is our normalised frequency.

Analysing the bass transient first, there is a large peak centred at zero frequency and prominent peaks at roughly $\theta = \pm 0.16$, which corresponds to $f = 127Hz$. A crude estimate of the fundamental signal frequency can be made by reading the period of the signal from figure (9). This gives $T = 8.11ms$ or $f = 123Hz$ - in fitting with our spectrum estimates. There are also lower amplitude higher frequency components, but these are lost in with the noise.

Similar analysis can be carried out to find the contents of the other signals, for example we can estimate that the steady note has a $530Hz$ component. All signals seem to have a very low frequency component too, often centred at or around zero. It is hard to tell whether this is part of the signal or due to the noise as it usually below $20Hz$. It is also difficult to compare frequencies across spectra without explicit calculation of the true frequency as the number of samples between signals is different, therefore we can only compare relative locations within one spectrum.

In terms of differences between the spectra, the spoken sounds seemed to have more peaks, but these were buried in quite strong noise in the original signal (see audio samples). Therefore one may tentatively conclude that spoken sounds generally contained a broader variety of sounds and harmonics than the musical sections that were isolated.

2.7 Task 8: Challenge

Figure (13) shows the high resolution windowed spectrum of the organ chord. The window length was continuously adjusted to balance getting enough resolution in the low frequencies, whilst trying to keep only the most important peaks in the high frequencies. If the window is too short then the spectrum at low frequencies is just one very wide (and slightly irregular) peak, if it is too long then the high frequency spectrum becomes a mess of very narrow peaks and becomes unreadable. In order to try and guess the main frequencies present, I compared the two spectra and tried to read off the distinguishable peaks that were common to both. This gave me a set of normalised frequencies which I could convert to frequencies and hence musical notes, as in table (14). I expect that this is far from an exhaustive list of the components present in the organ recording, but when played back through a sine wave generator this combination vaguely resembles an organ sound.

3 Conclusion

Windowed spectral analysis is a powerful tool for estimating the spectra (fourier transforms) of long samples of periodic signals. There are several important parameters to consider when windowing including the length and type of the window; these determine the resolution, shape and characteristics of the spectrum.

```
def dft(xt):

    # extract data length, N
    N = xt.size
    axis = np.arange(N)

    # build a fourier matrix by multiplying appropriate matricies
    k,l = np.meshgrid(axis, axis)
    F = np.matmul(l, k) / N

    # raise complex exponential to powers in F
    w = np.exp(-2*np.pi*1j/N)
    W = np.power(w, F)

    # simple matrix multiplication to extract dft
    Xw = np.matmul(W, xt)

    return Xw
```

Figure 1: Python DFT code

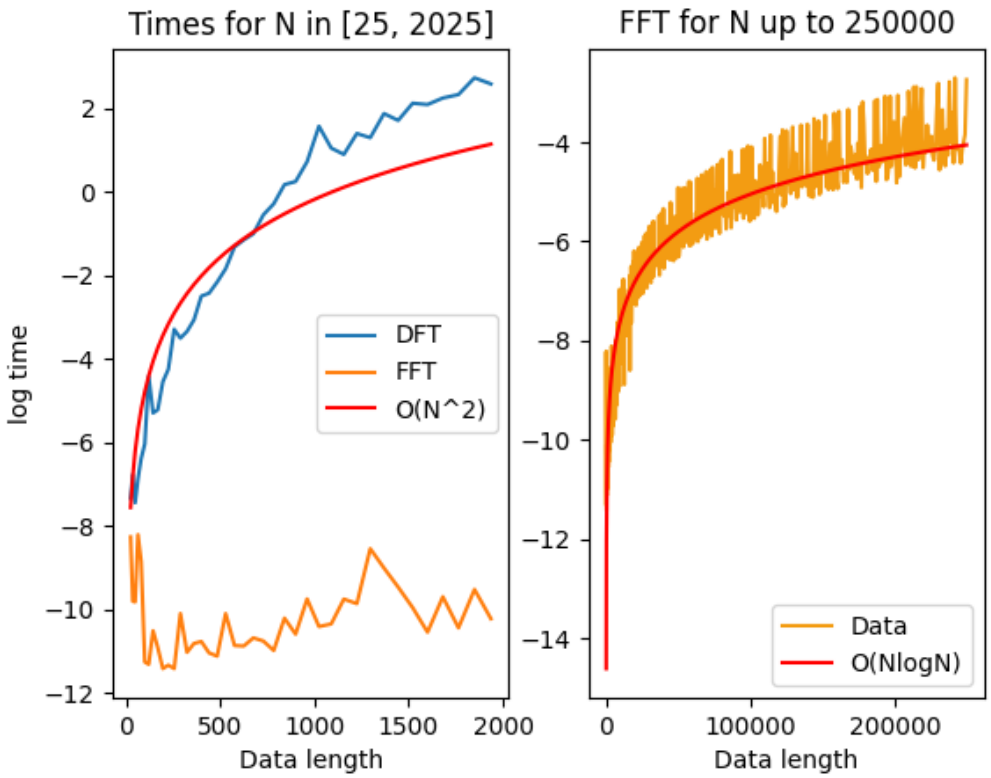
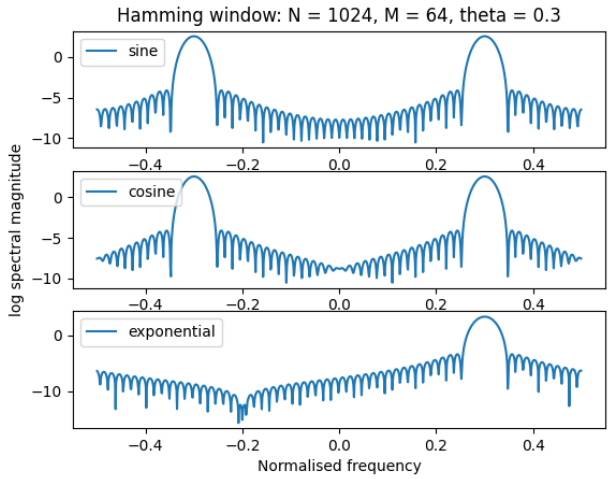
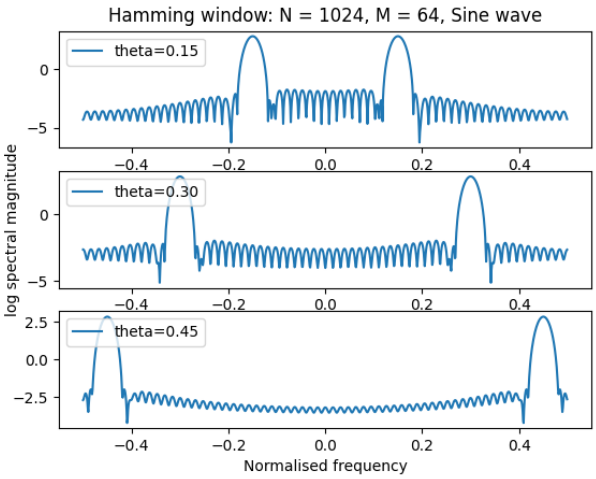


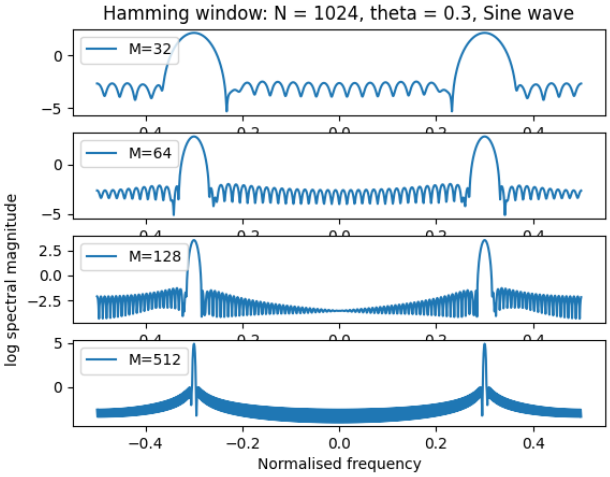
Figure 2: Left: fft vs dft timing data up to 2025 samples. Right: full fft timing data up to 250000 samples



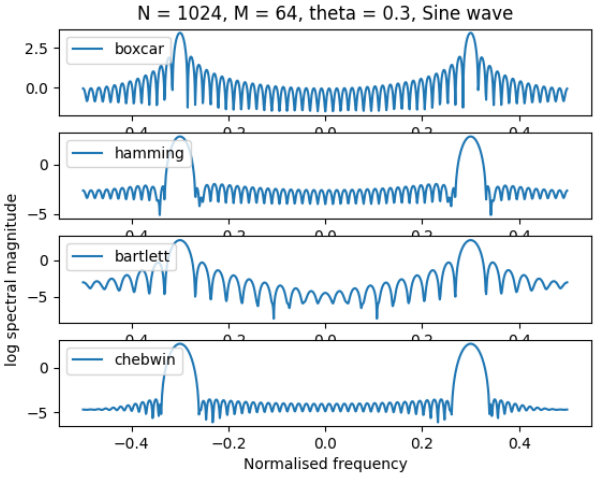
(a) Different oscillating functions



(b) Increasing signal frequency



(c) Increasing window length



(d) Different choices of window

Figure 3: Plots from task 3

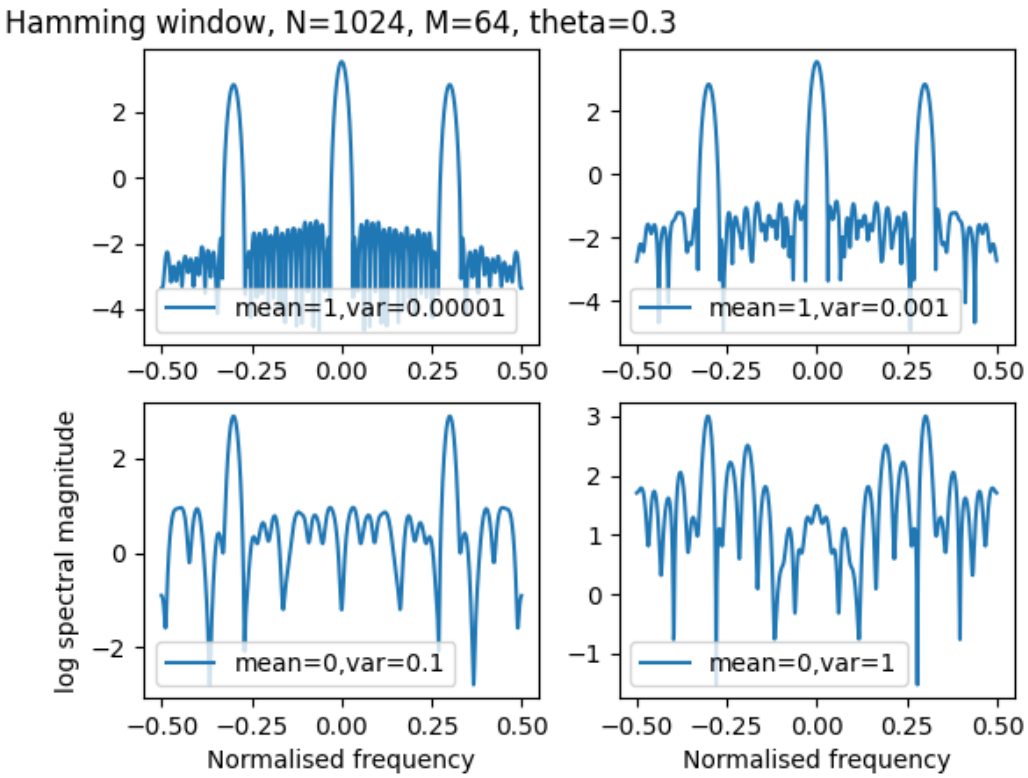


Figure 4: Noisy spectra

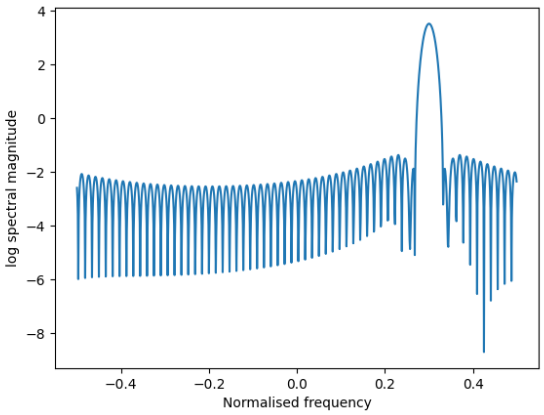
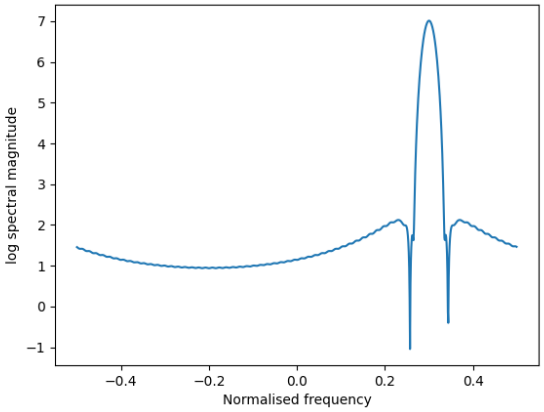
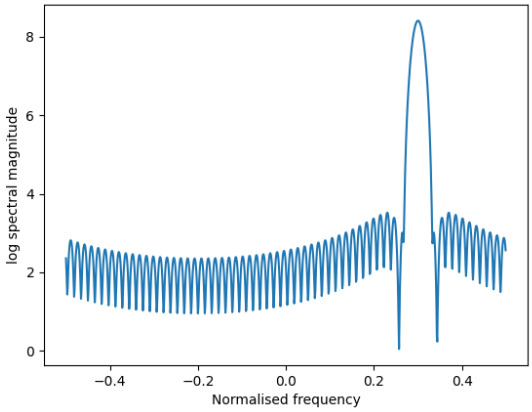


Figure 5: Unmodulated signal

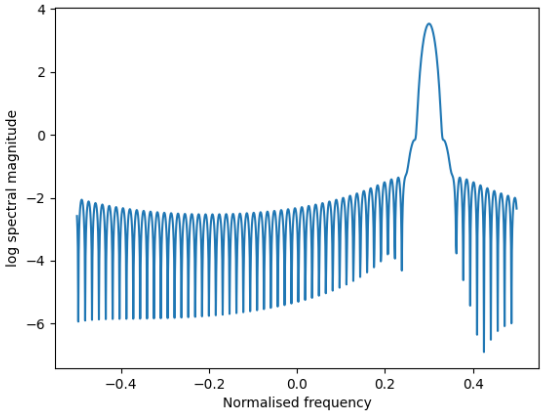


(a) $A = B$

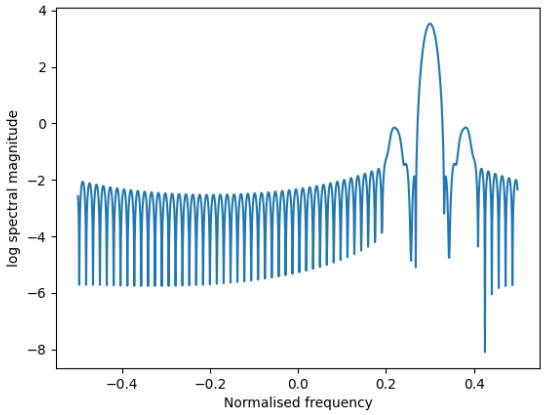


(b) $A << B$

Figure 6: $a_n = An + B$

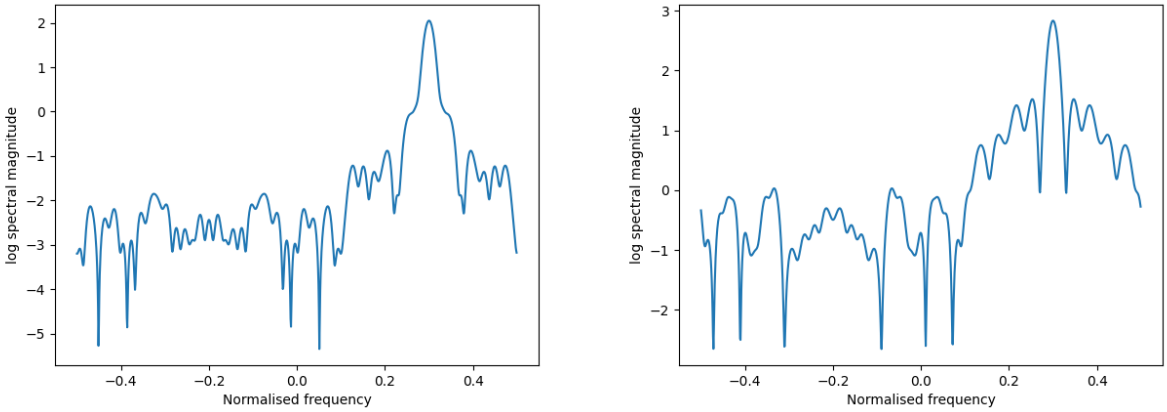


(a) $\beta = 0.05, \phi = 0.2$



(b) $\beta = 0.05, \phi = 0.5$

Figure 7: $a_n = 1 + \beta sin(\phi n)$



(a) $a_0 = 0, \sigma^2 = 0.001$ (b) $a_0 = 0.05, \sigma^2 = 0.1$

Figure 8: $a_n = a_{n-1} + \sigma z_n$

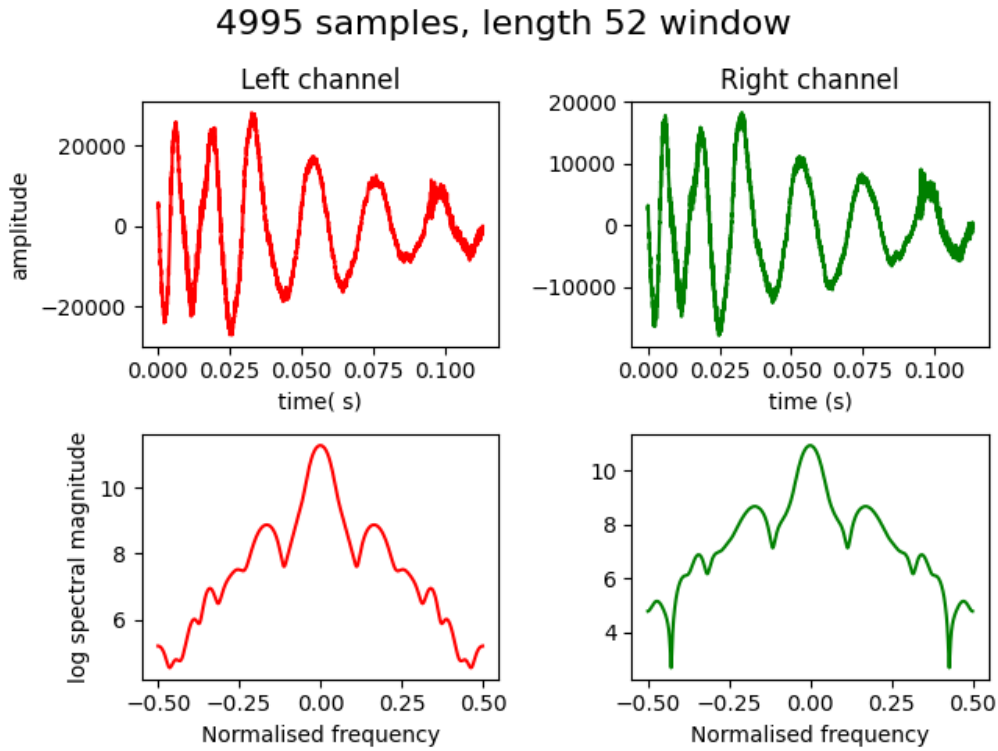


Figure 9: Bass transient note

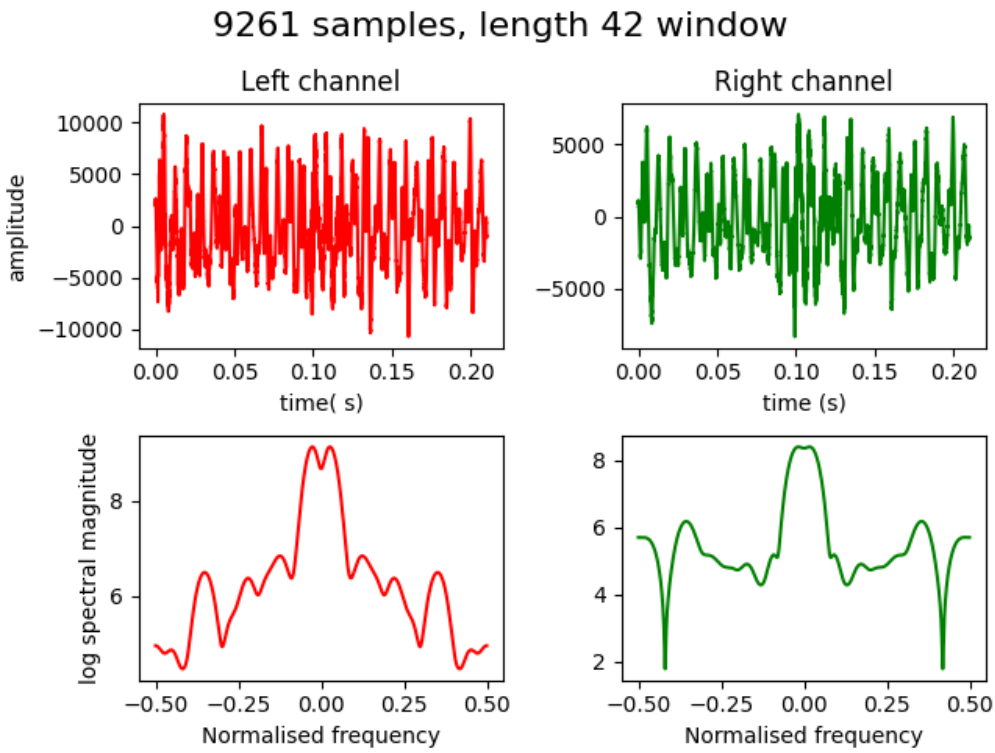


Figure 10: Steady note

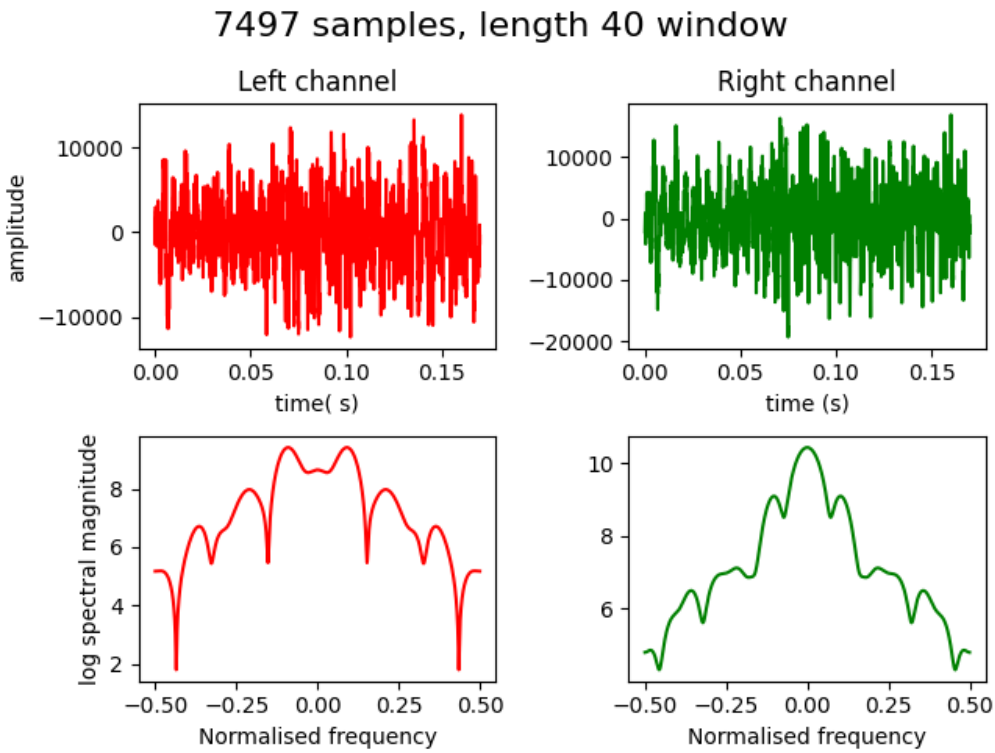


Figure 11: 'ma' sound

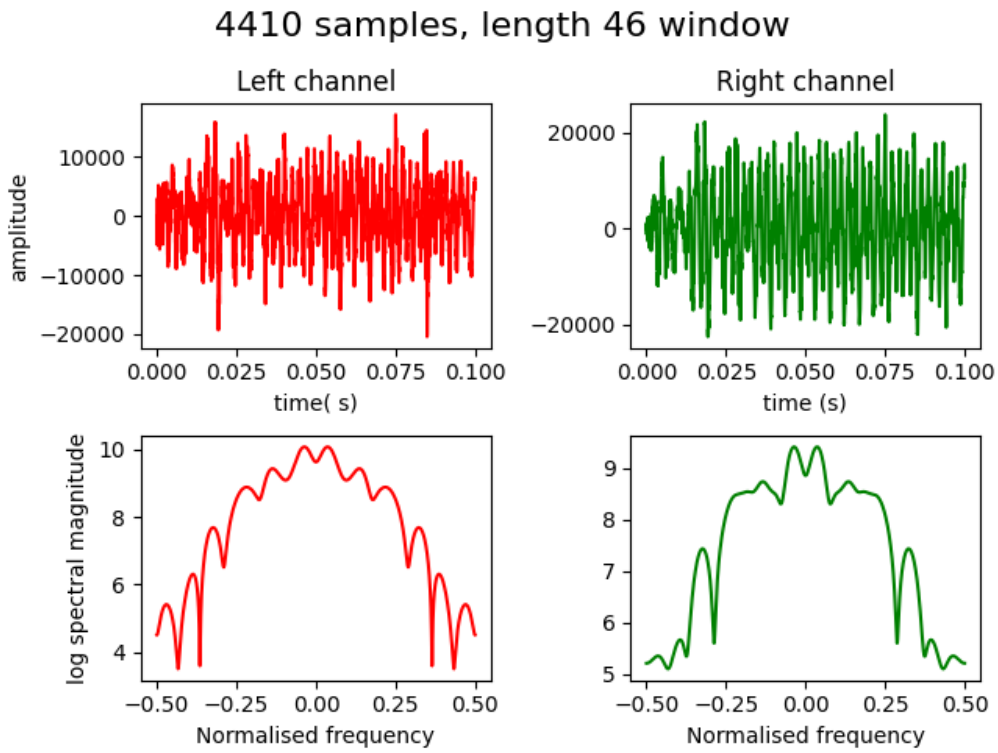


Figure 12: 'th' sound

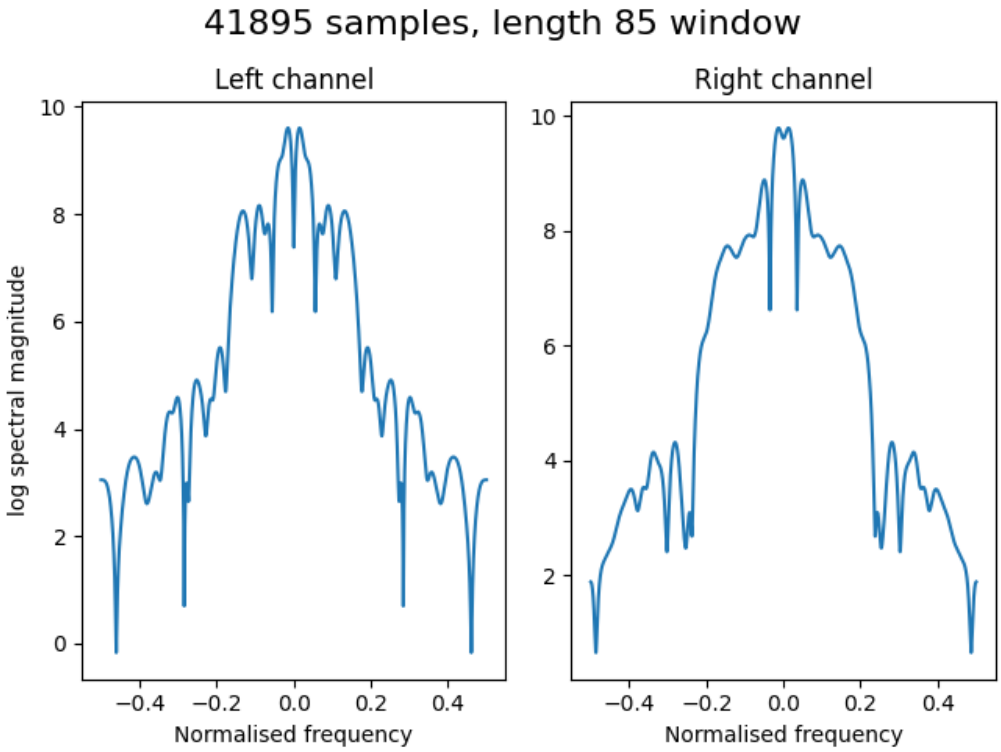


Figure 13: Windowed spectrum of organ recording

θ	f (Hz)	(Rough) Note
0.0126	84	E2
0.0500	333	E4
0.0663	442	A4
0.1316	877	A5
0.1911	1274	D#6/E♭6
0.2519	1679	G#6/A♭6

Figure 14: Table of estimated musical notes