Synthesis

Alex Booth
a.booth9@edu.salford.ac.uk

October 2022

1 Abstract

Here goes the abstract

Contents

2 Introduction

Musical instruments have been part of human culture and craft since prehistory, playing a part in both written and verbal art, ceremony and celebration [1]. This report describes an attempted recreation using digital synthesis of two acoustic musical instruments from the western European musical tradition: a mandolin and a flute. Acoustic musical instruments use an excited string, SOMETHING OR SOMETHING - REF to generate waves which are manipulated and shaped by the body or of the musical instrument NEEDS A TIED-IN CITATION.

The generation of these waves, and their manipulation by the body of the musical instrument can be modelled using a series of oscillators, filters and modulators. Audio synthesis using analog circuitry was explored as soon as simple oscillators were readily available, leading to early electronic musical instruments such as the theremin. The synthesis of musical instruments can be achieved using many techniques. In order to emulate the sound of musical instruments using synthesis, first we must understand the mathematical nature of sound as a signal. Thus, this report will describe the programmatic mathematical analysis of the aforementioned two recordings, and an attempt to use the data gathered in the analysis to guide the chosen synthesis technique.

3 Theory

3.1 Signal processing, generation and analysis

Sound can be understood as a signal which is a function of time. As sound travels in air it is a longitudinal pressure wave, whilst when in the electronic domain, such as after transduction by a microphone, it is transverse wave of voltage as a function of time CITE?. When transformed from the analogue electronic domain to the digital domain, the signal remains a transverse wave and a function of time. However, it changes from a continuously changing voltage to a numerical sample value that increments in discreet steps. CITE

Any periodic continuous signal can be broken down into an infinite sum of weighted sine and cosine waves [2]. A further development of the theoretical foundation began with the Fourier series can be found in the Fourier transform, which in continuous time takes the form:

$$X(\omega) = \int_{-\infty}^{+\infty} x(t) e^{-j\omega t} dt \tag{1}$$

However, as digital audio works in the discrete time domain, the integral seen in Eq.1 is changed to a sum and the Fourier transform becomes:

$$X(\omega) = \sum_{-\infty}^{+\infty} x[n] e^{-j\omega t} dt \tag{2}$$

In the 1960s, J.W. Cooley and John Tukey developed an algorithm to efficiently compute the discrete Fourier transform [3]. This, and other discrete-time Fourier transform algorithms, became known as the fast Fourier transform(s) (FFT). MATLAB's in-built FFT function uses an FFT algorithm from the the FFTW library [4], which builds on Cooley and Tukey's original FFT. Using a Fourier transform, a complex signal of any length such as a musical note or speech excerpt, can be broken down into it's frequency components, showing their respective magnitudes averaged over the length in time of the signal. Phase information can be extracted from the Fourier transform, but this information was not particularly useful for the purposes of the synthesis described in this report.

The Fourier transform allows the fundamental frequency and harmonics of the played note to be identified, due to the fact that in a good quality recording they will be the most prominent components of the frequency spectrum. However, the Fourier transform does not only show the harmonic or musical frequency content of a signal, instead it shows all the frequency content of a

signal. As such, any noise or other anharmonic frequency component, musical or not, will be shown on the transform. Thus, the simple inclusion of all observed frequency components may or may not be constructive to emulating the original played note. For example, this may include unwanted recording artifacts.

Early musical synthesizers were constructed to facilitate additive synthesis techniques. This is where a series of oscillators generate basic waveforms such as sines, sawtooth and pulse waves; a sawtooth wave can be seen in Figure 1.

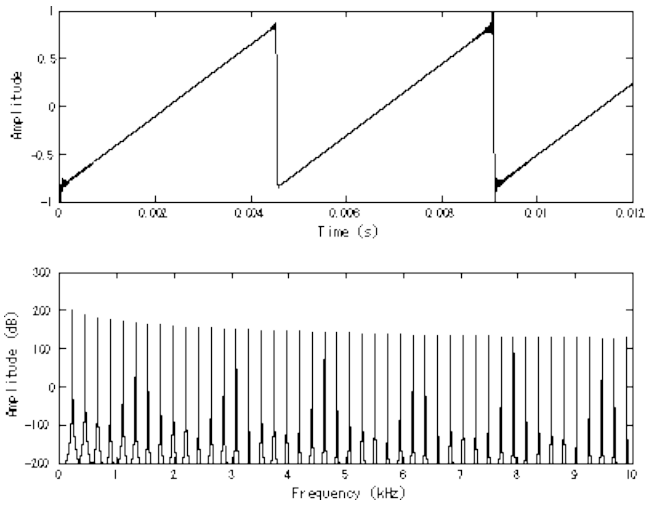

Figure 1: Waveform and frequency response of a sawtooth wave.
[5]

These oscillators may be used to generate sound, or used as modulators to other oscillators's outputs. Shown in Fig. 2 is an oscillator bank from a synthesizer, it illustrates the basic wave shapes that were available as building blocks for additive synthesis.

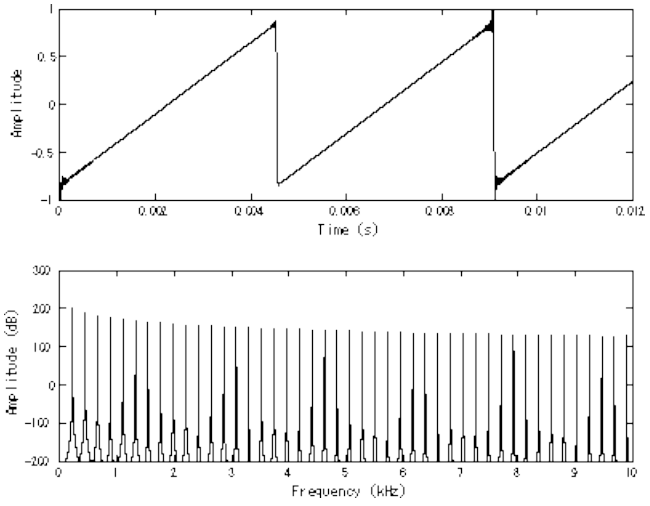

Figure 2: Waveform and frequency response of a sawtooth wave.
[5]

Different waveforms generated by these oscillators will have a different harmonic content. For

example: in Figure 1 a sawtooth wave is shown as having many harmonics in it's frequency domain plot. In fact, a sawtooth wave contains all harmonics both even and odd, of a fundamental tone [6].

In the 1970s, FM synthesis was beginning to take form as a method of musical instrument emulation, especially after Chowning's work outlined specific FM techniques for the emulation of various instruments [7]. Products such as the Yamaha DX7 helped to popularize FM synthesis into the 1980s.

3.2 Musical instruments

4 Methodology

4.1 Analysis

4.1.1 Mandolin

All code written for this report was executed in MATLAB. First, the sample values and sampling rates of the recordings were imported into MATLAB as a vector and constant, respectively. Using the sampling rate and total number of samples, a vector of time data was created. This allowed the sample values to be plotted against time, as seen in Figure 3.

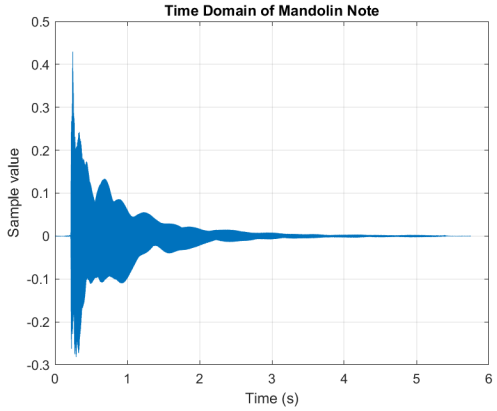

Figure 3: Mandolin sample value over time.

Next, analysis of the signal in the frequency domain was performed. Using MATLAB's in-built FFT function, a frequency domain plot of the signal, averaged over time, was plotted.

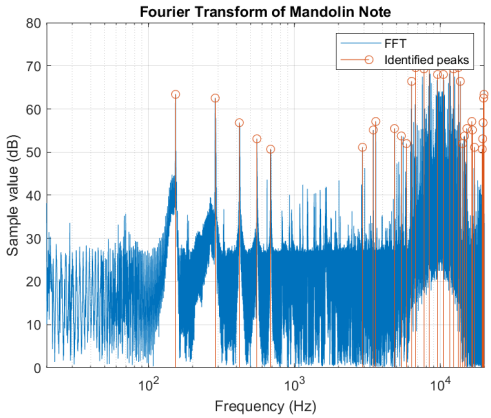

Figure 4: Mandolin sample value over time.

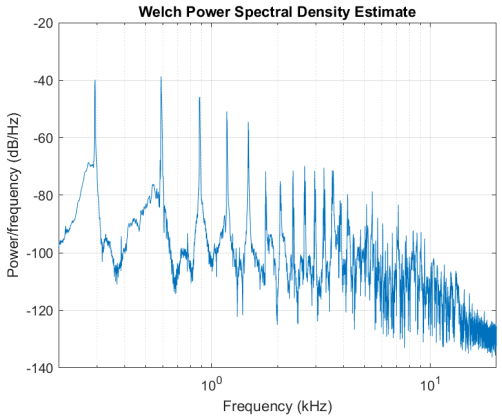

Figure 5: Mandolin sample value over time.

This data was supplemented with another frequency domain plot using `pwelch()`, a function that uses the Welch algorithm to obtain power spectral density as a function of frequency, instead of sample magnitude [8]. A plot of the frequency and phase responses generated by `pwelch()` are shown in Figure 5.

The frequency amplitude information gathered from the Fourier transform and Welch power spectral density estimate is averaged over the entire duration of the signal. While this information is very useful, it is limited: Much of the frequency content of a signal varies in time, and this time-varying nature of the frequency response contributes greatly to the To see the amplitudes of each frequency component as a function of time, a spectrogram was used.

MATLAB's `spectrogram()` function has a high degree of operability and can take many arguments. The function works using a Used as a standalone function, it gives the user a two dimensional graph of frequency versus time, with frequency amplitude as a third, coloured dimension. There is a trade-off between resolution in the frequency and time domains when using this spectrogram function. Higher resolution in the time domain allows for a more visually understandable spectrogram when using a waterfall plot. If a high frequency-domain resolution is used, a waterfall plot becomes cluttered and hard to read, as shown Using a hamming window with length $n = 2560$, a 2650-point spectrogram was computed and plotted on a waterfall graph, shown in Figure 7.

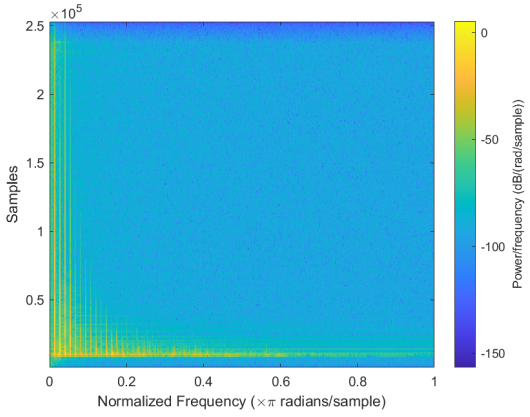

Figure 6: Mandolin sample value over time.

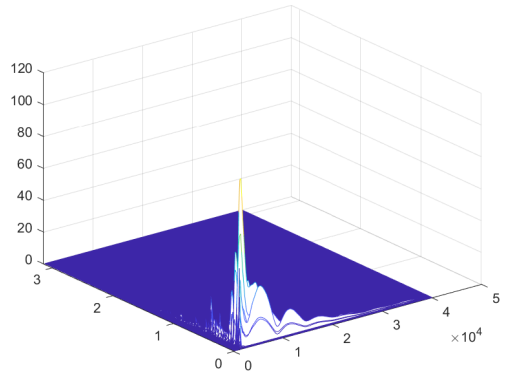

Figure 7: Waterfall plot of mandolin frequency-domain amplitudes with respect to time.

4.1.2 Flute

Much like with the mandolin, time and frequency domain plots were generated of the flute recording. These can be seen in Figs.8, 9, 11 and 10.

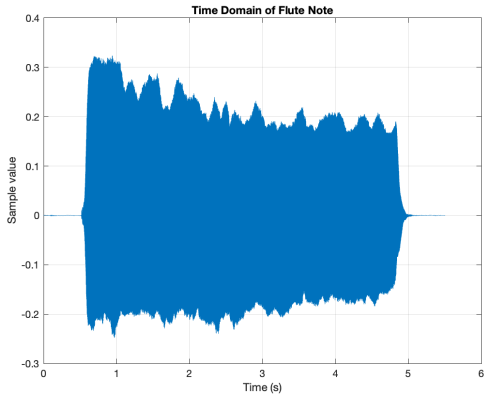

Figure 8: Time-domain of the flute recording.

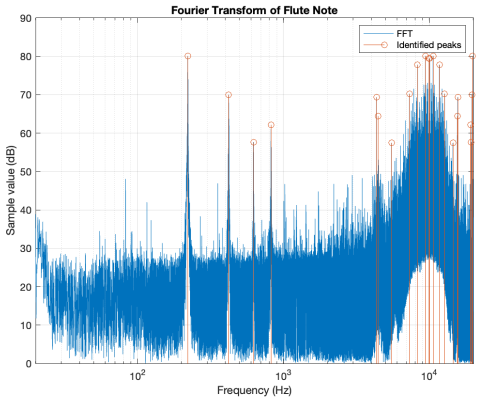

Figure 9: Fourier transform of the flute recording.

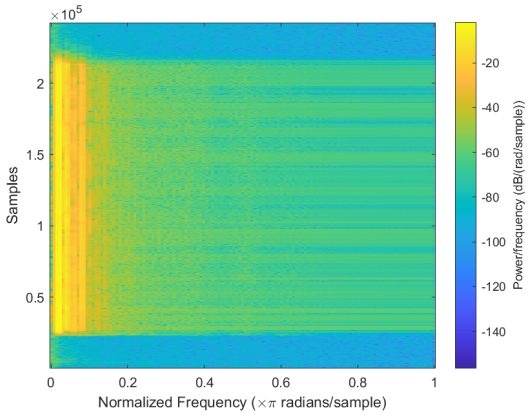

Figure 10: Spectrogram of the flute recording.

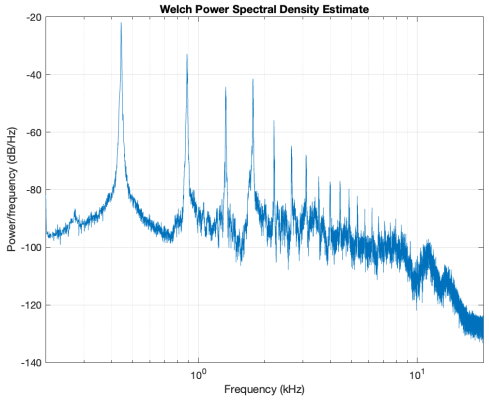

Figure 11: Welch-algorithm frequency analysis of the flute recording.

From Fig. 8 it can be observed that the flute has a similar attack, smaller decay and a more constant sustain compared to the mandolin. Figures. 11 and 9 show a harmonically rich frequency content. Upon further inspection, the first four harmonics are read as $f_n = 220, 421, 623, 823$Hz. These values roughly correspond to both odd and even harmonics of the fundamental $f = 220$Hz. Figure. 10 shows that harmonic content, especially in the lower end of the spectrum, remains fairly constant throughout the duration of the note. This suggests that FM is a suitable method for synthesis, as the fine control over harmonics given by TVPAS is not particularly important.

4.2 Synthesis

4.2.1 Additive Synthesis

Additive synthesis follows the intuition found in Fourier analysis: Any continuous signal is a sum of an infinite number of sine waves. Thus, it should be possible to recreate any given signal by breaking it down into it's frequency components, and reproducing these frequency components as sinusoids weighted according to the frequency domain of the analysed signal. A programmatic approach was taken in MATLAB: A fourier transform of the recorded signal would return a set of data that may be parsed to find the frequencies of the dominant harmonics of the Mandolin. This was implemented by feeding the result of the fourier transform into MATLAB's `findpeaks()` function, which allows the user to define arguments as to the minimum height and prominence of maxima within a set of data, these maxima would be the harmonics. Using `findpeaks()` required some trial and error adjusting function arguments to avoid less useful peaks within the Fourier transform. The found frequencies and their amplitudes would then be substituted into a looping summation of generic sine wave formulae, to produce the output: a complex signal with a similar frequency response to the input; importantly with the same harmonics. This formula is shown in Eq.3.

$$y[t] = \sum_{n=1}^{N} A_n \sin\left(2\pi f_n t\right), \text{ where: } n = \text{harmonic number} \tag{3}$$

This produced a signal which contained similar frequency-domain characteristics as the Mandolin recording, but had no amplitude envelope. Thus when played using MATLAB's `sound()` function, the synthesised sound was missing the characteristic plucked attack of a Mandolin. To compensate for this, an amplitude envelope was extrapolated from the recorded signal using MATLAB's `env()` function. When passing only the sample value vector to `env()` a vector of the absolute value of the sample values is returned. This is problematic; the sharp changes in amplitude, as shown in Fig. 12, will result in additional high-frequency components being added to the signal when the envelope is applied to the synthesized sound.

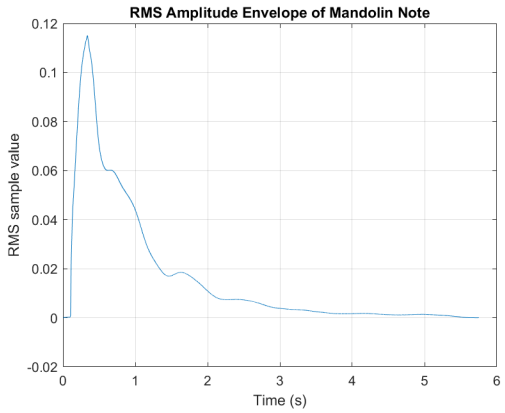

Figure 12: Envelope reflecting the absolute value of each sample.

Passing an `'rms'` argument to `env()` gave a smoother envelope than simply taking the absolute value of the time-domain waveform, this addressed the aforementioned problems with using an absolute envelope. This new envelope is seen superimposed in Fig. 12. To apply the envelope, simply multiplying it with the synthesized signal was sufficient. Figure 13 shows the envelope applied to the synthesized signal. Comparing to Fig. 3, we can see a similarity in the time-domain.

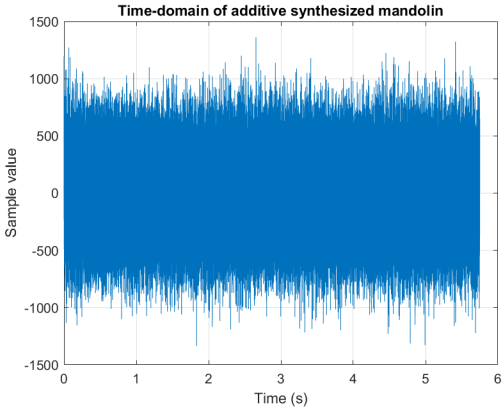

Figure 13: Time-domain of the synthesized signal with envelope applied.

Whilst the spectrograms shown in Figures 6 and 7 were useful in illustrating and conveying the amplitude envelopes of a large amount of frequencies across the recording's spectrum, the additive synthesis method chosen chooses to reproduce only a specific few of the harmonic components of the recorded signal. Thus, a new spectrogram function will have to be computed targeting the harmonics identified with `findpeaks()`. MATLAB's spectrogram function thankfully allows for an argument to define target frequencies, so after passing the frequency vector extrapolated from `findpeaks()`, a two-dimensional array of amplitude envelopes for each identified frequency was generated. This spectrogram can be seen in Figs.14 & 15.

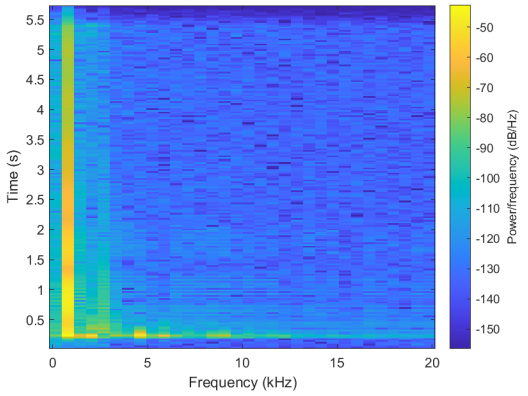

Figure 14: Mandolin sample value over time.

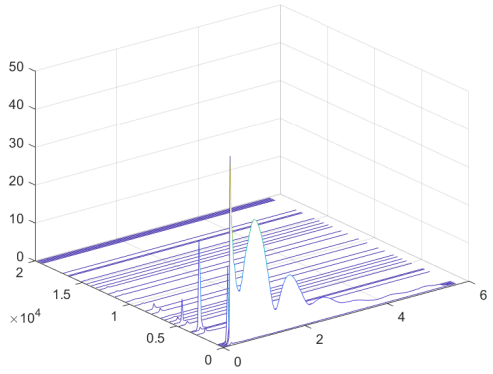

Figure 15: Mandolin sample value over time.

4.2.2 Frequency Modulation Synthesis

To emulate the recorded flute, frequency modulation (FM) synthesis was used. FM synthesis generates signals with rich harmonic content using only two waveforms, as opposed to the theoretically limitless number of oscillators typically used in additive synthesis. Instead of summing the oscillators as in additive synthesis, the two oscillators are multiplied. The two waves are referred to as the "carrier" and the "modulator" Chowning elaborates on the relationship between the two waves: "In FM the instantaneous frequency of a carrier wave is varied according to a modulating wave, or modulating frequency" [7]. This is reflected in the general equation for FM synthesis:

$$\begin{aligned} y(t) &= A\sin\left(\omega_c t + I\sin\omega_m\right) \\ &\text{Where:} \\ A &= \text{Amplitude coefficient} \\ \omega_c &= \text{angular frequency of carrier} \\ \omega_m &= \text{angular frequency of modulator} \\ I &= \text{modulation index} \end{aligned} \tag{4}$$

Returning to Chowning's paper, parameter sets and methods for use of the above equation targeting the sound of certain instruments are given. Chowning states that a woodwind sound may be achieved: "By setting the carrier frequency to be an integral multiple of the modulating frequency, or by making the index function inversely proportional to the amplitude function". To implement this, a frequency equivalent to the identified fundamental tone of the flute, $f_c = 220$Hz was used for the carrier. The modulator was then set to a value of $f_m = f_c * 4 = 880$Hz. By making the index function $I = \frac{\delta_f}{f_m}$ both of Chowning's listed woodwind criteria are met. A fourier transform of the output of the frequency modulation synthesis is shown in Figs.16 & 17.

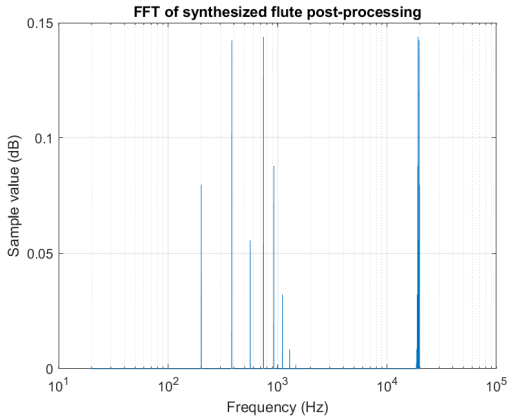

Figure 16: FFT of output of FM synthesis.

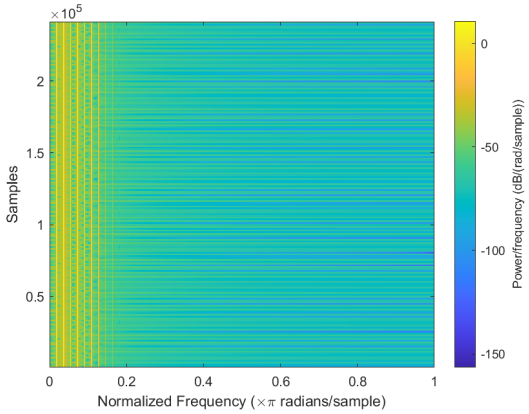

Figure 17: Spectrogram of output of FM synthesis.

The spectrogram in Fig. 17 shows a good similarity to the frequency response over time shown in Fig. 10. Thus, the next step was to match the amplitude response of the synthesized note to the recorded note. A programmatic amplitude envelope using the exact same `env()` function as the mandolin synthesis was generated. Also, a manual attack-decay-sustain-release envelope was drawn in MATLAB from observation of Fig. 8 and used to weight the amplitude of the signal over time. The manual and programmatic amplitude envelopes are shown in Figs.19 & 18.

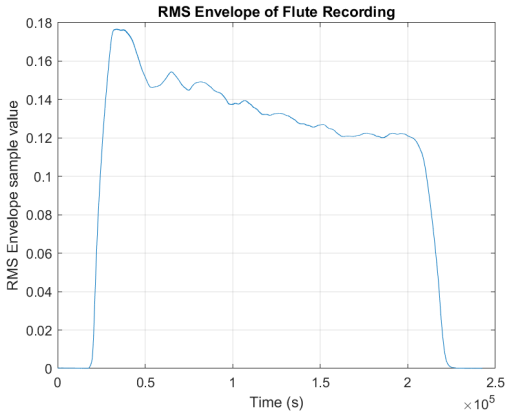

Figure 18: RMS amplitude envelope of recorded flute.

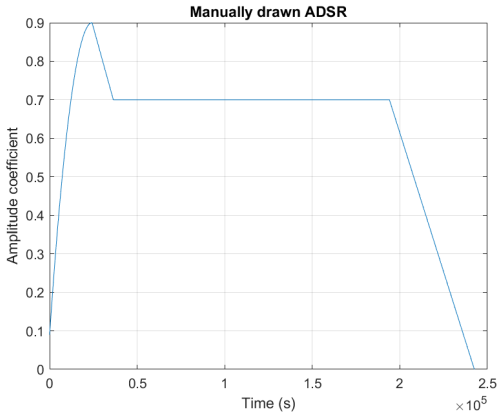

Figure 19: Manually drawn ADSR.

Listening back to the synthesized note with the manual and programmatic amplitude envelopes applied revealed the manual envelope to sound "unhuman" with none of the minor fluctuations in amplitude caused by an air-reed and slightly inconsistent airflow from the mouth. Whilst this may be fixed by adding low-frequency amplitude modulation to achieve a subtle "tremelo" effect, the programmatic amplitude envelope was chosen instead to keep things simple.

Looking back to Fig. 16 a large amount of clustered high frequency content can be seen and is also audible on playback of the synthesized sound; this is unlike the flute recording. To combat this, a 16^{th} order low-pass Butterworth filter was generated using `butter` and applied to the synthesized signal using `filter()`. In addition, a low-amplitude noise signal was summed with the synthesized tone to attempt to recreate the "airy" or "breathy" nature of the flute's sound. A plot of the FM synthesis after going through this processing is seen in Figs.20 & 21

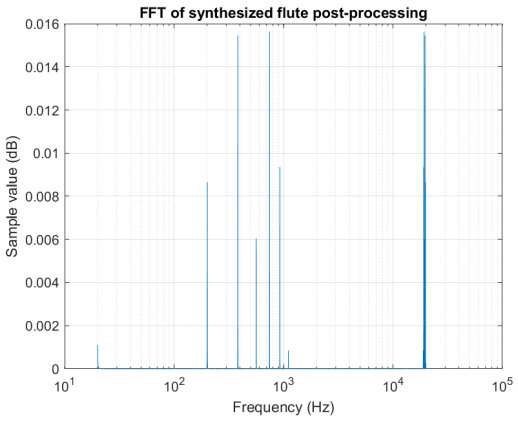

Figure 20: FFT of post-processing FM synthesis.

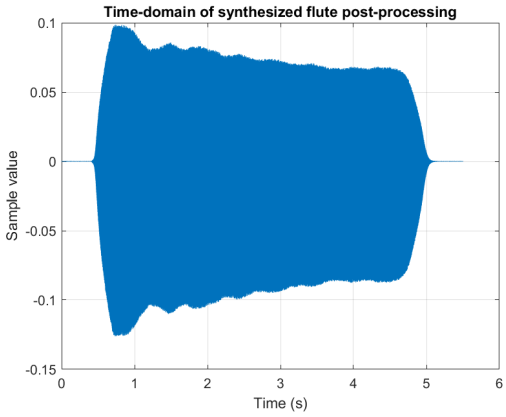

Figure 21: Time-domain of post-processing FM synthesis.

5 Discussion and Conclusions

6 Appendix

6.1 Code

References

[1] L. Rault, *Musical Instruments, Craftmanship and traditions from Prehistory to the Present.* Harry N. Abrams, 2000. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02072647

[2] E. W. Weisstein, "Fourier series," *https://mathworld. wolfram. com/*, 2004.

[3] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex fourier series," *Mathematics of computation*, vol. 19, no. 90, pp. 297–301, 1965.

[4] M. Frigo and S. G. Johnson, "Fftw: An adaptive software architecture for the fft," in *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP'98 (Cat. No. 98CH36181)*, vol. 3. IEEE, 1998, pp. 1381–1384.

[5] S. Kraft and U. Zölzer, "Lp-blit: Bandlimited impulse train synthesis of lowpass-filtered waveforms," 2017.

[6] J. G. Roederer, *The physics and psychophysics of music: an introduction.* Springer, 1995.

[7] J. M. Chowning, "The synthesis of complex audio spectra by means of frequency modulation," *Journal of the audio engineering society*, vol. 21, no. 7, pp. 526–534, 1973.

[8] O. M. Solomon Jr, "Psd computations using welch's method.[power spectral density (psd)]," Sandia National Labs., Albuquerque, NM (United States), Tech. Rep., 1991.