Assignment 2 on Modal Synthesis

Teo Nordström - teon@kth.se

February 2025

1 Time Domain Waveforms

Code in **A1.py**

As the base of all further partial assignments, we have to first just read and take a simple look at the waveform of the glockenspiel samples. I decided to choose both with the same pitch but with different attack strengths (soft hit **F#5 f** and hard hit **F#5 p**). I felt as if these would be the best to compare, as it would be easier to compare same base pitch rather than different ones.

For 1. I just use the Scipy library built-in function wavfile to read the wav file and return the sample rate and sampled data. This data is then put into a Pyplot line plot with linearly spaced x-points generated with Numpy. I also made some more plots with only the first 40k samples to take a closer look, but this did not change much when it came to the fidelity of the plot.

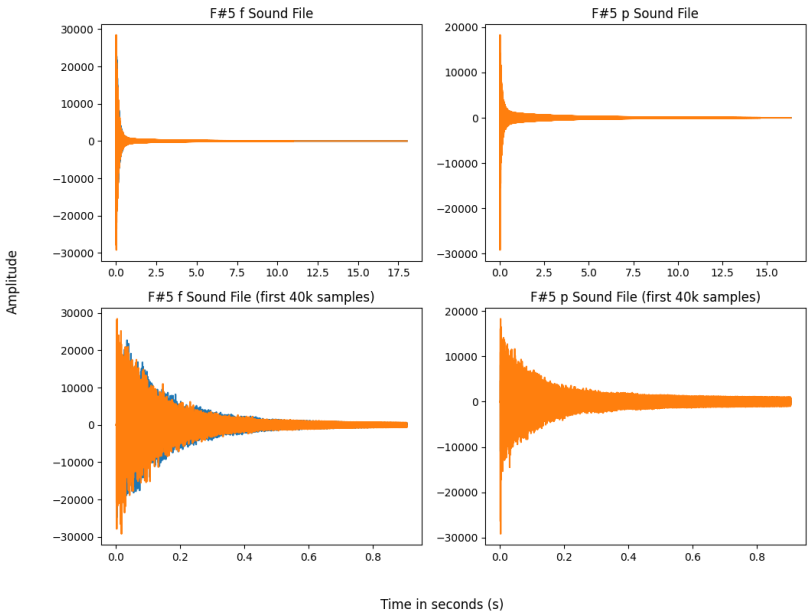

Figure 1: Plot received from code in A1.py. Note the blue colour, signifying that this is a stereo file.

2 Magnitude Spectra

Code in **A2.py**

This part of the exercise is to plot magnitude spectra and get the strongest peaks partials of each of the sound files. This part took by far the bulk of the time working on the exercise, as understanding all of the parts and finding tools that could help with the problem took some figuring out.

To begin with, I extract the first channel since stereo caused some issues further on in the program. Then, using the rate we get the amount of data points that would constitute a 100ms window. I could not figure out how to find the onset through code, so I zoomed in on the waveform in Audacity and took the offset and inserted it in the code. I can now get the two 100ms windows, one at the onset and one almost 700ms in the future. Then, using the `magnitude_spectrum` function from Pyplot I use this data to get the magnitude spectrum and plot it in one fell swoop.

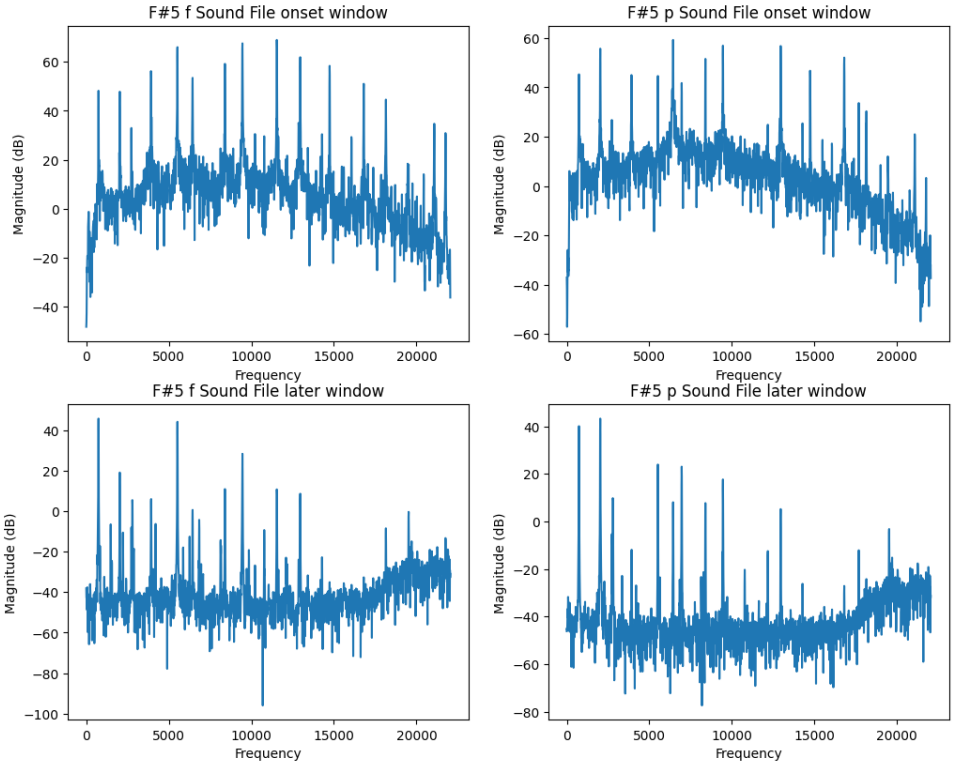

Figure 2: Magnitude spectra of the sound files. Clear peaks can be seen that are similar for all sounds

With the magnitude spectrum part completed, we need to find the partials. The `magnitude_spectrum` output the magnitudes of the frequencies, and using this data I use the Scipy function `find_peaks` to get the peaks and then remove the peaks over 15kHz. I then use some zip magic to sort both

lists in order of magnitude (so the frequencies and their corresponding magnitudes still are on the same indices). The 15 highest energy peaks are kept, then the lists are again sorted, this time in order of frequency.

With the frequencies and their corresponding amplitudes gotten, we plot these together as bar plots to show which partials have the highest energies.

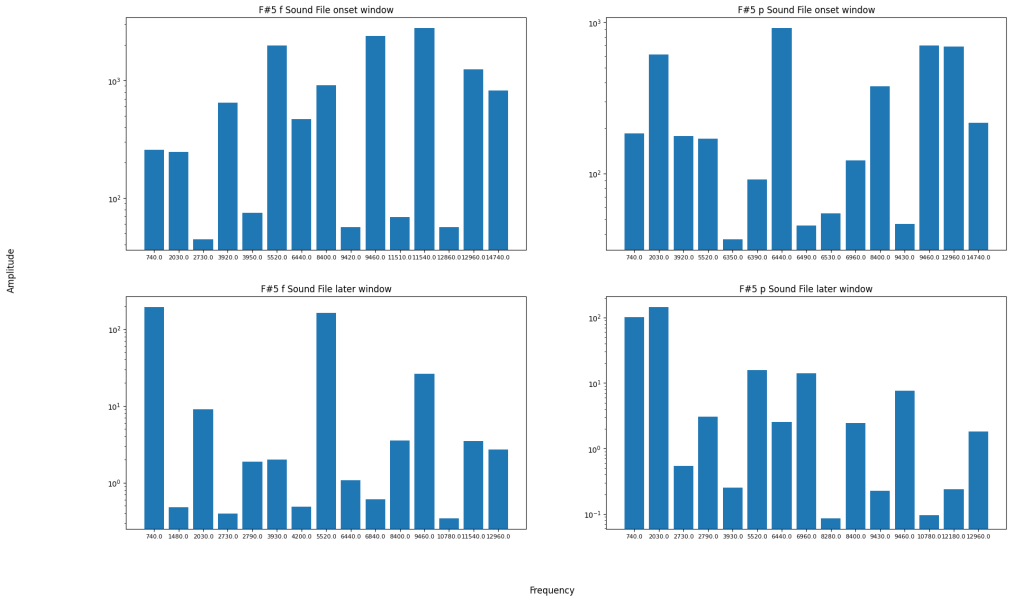

Figure 3: Frequency amplitude bar plots. It can be seen that most of the frequencies are the same.

3 Envelope Parameter Estimation

Code in **A3.py**

Now, instead of plotting the partials, the peak frequencies are checked. If there are two peaks with the same frequency that are within the 15 highest peaks in both windows, the frequency and amplitude is put into a list. From these, we use the equation

$$\gamma_f = \frac{ln(A_{f_1}) - ln(A_{f_2})}{t_1 - t_2}$$

to get the envelope parameter γ_f for each frequency f. Now that the parameter is acquired, we just have to calculate the amplitude of the envelope over time and plot those values.

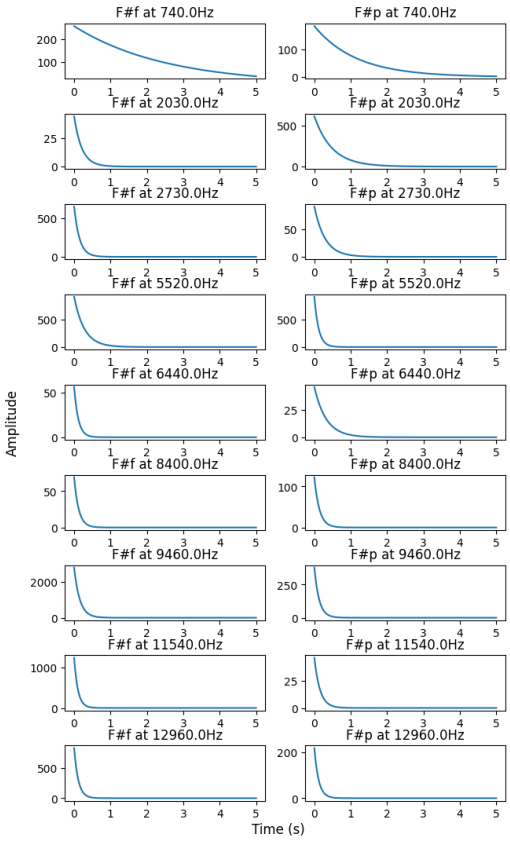

Figure 4: The Exponential Amplitude Envelopes of the different partials of the different sound files.

4 Additive Synthesis

Code in **A4.py**

By combining everything we have done done thus far, additive synthesis can be done to get a note that sounds approximately like the input sound. To do this, we first compute the envelope parameters as in 3. Then, we generate a list of linearly spaced values from 0 to 3 seconds, and a zeroed list for the note to collect into. Using the envelope parameter and the envelope equation

$$env_f(t) = A_f e^{-\gamma_f t}$$

to generate an envelope that will be multiplied together with the sine wave

$$sin(2\pi f t)$$

to get our sound. This is done for each partial, and the result is combined into a single list. The list is then normalized (as it peaked really bad otherwise) and saved to **synthesised_f.wav** (for

soft F#5 f) and **synthesised_p.wav** (for hard F#5 p). The sounds are then played using the Sounddevice library for Python, and clear tones resembling those of the input can be heard.

5 Sing us a song, you're the Glockenspiel man

Code in **A5.py**

If you have a soundfile and you want to change the pitch of it, generally you just speed up or slow it down to raise or lower the frequencies. However, this has the effect of also, well, speeding it up and slowing it down. Since we have the partials and are synthesizing the sound, we can instead just change the frequencies of the partials themselves, making the tone ring for just as long and have a more well-behaved pitch change.

I created some lists that include the pitch change from the original F#5 note, when in the song the tone should be played, and which of the two tones to use. For each of the times where a tone should be played, a new tone is generated, but with the frequency of each partial being multiplied with $2^{\frac{s}{12}}$ where s is the semitone difference (equal temperament) from the base note. To prevent aliasing, a check for the Nyquist limit for the frequency was also added. Then we just take a long line of zeroes and add the tone sounds onto this to make the song. This has to be normalized again, as it caused some horrid peaks. The generated simple tune can be found in `song_a5.wav`, and can be heard to sound pretty okay considering that the pitch changes keep quite close to the original pitch!

5.1 Going Overboard

Code in **midi_synthesizer.py** and **wintergatan.py**

This was technically **not part of the assignment**, but I wanted to see if I could make a MIDI synthesizer. I spent a bit too much time on it, but in the end it turns out that I could! I first divided the code into different classes to make it modular. Then, using the Mido library I read the Midi files to extract important information about timing. Then, I make instruments using base sound files (like in all the other parts of this assignment). Luckily for me, the values of Midi pitches are already split into semitones, so I could keep using the same calculations as earlier. The instruments can be assigned to tracks in the midi, so you can have different sounds for different parts. Tracks without instruments will be ignored. All of the notes are at last combined together, normalized, and played.

The program worked surprisingly well. My first thought was to try Nyan Cat, because I am stuck in 2011 for some reason, and it works well. Pitches that are far away from the base F#5 pitch sound a bit off, especially when going down in pitch. This made the bass track sound a bit off. This song can be found in `song_fun.wav`.

For something that might work a bit better I decided to find a midi for Wintergatans Marble Machine. This is a song that is originally played with a vibraphone and a bass (plus percussion which

will be ignored), so by assigning the instruments I synthesized we get quite a good facsimile of the original. Again, the bass sounds a bit odd and the percussion is missing, but this definitely sounds more passable than the nyan cat! This song can be found in `song_wintergatan.wav`.

To get these songs to sound better, it could be a good idea to process another sound that is more bass-y. While it would be quite easy to do I am straying far too far away from the assignment now so I'll leave it at that.

6 Discussion

By comparing the synthesized sounds and the original sounds we can hear that the originals have way more timbre - there's just much more going on in the sounds. This makes them sound a lot more interesting, and makes them have fuller spectrums. However, that considered, the facsimile synthesized sounds are in my opinion scarily good for something that is just the highest partials ripped out. It can be heard that the synthesized sounds have less of a resonance and depth of the sound, very likely because of this lack of a lot of the less powerful frequencies that the real glockenspiel exudes.

A problem with the synthesized tones, though, is that not all source tones are created equal. When comparing the soft hit F# f to its synthesized version it sounds a lot more similar than the hard hit F# p does to its synthesized version. This, again, is due to all of the lower power frequencies that are ignored. You can definitely tell from the sound which one is supposed to be the hard hit and which is the soft hit, but the hard hit is too ”clean” to really be interpreted by the brain as a hard strike. That property is much more audible in the original recording. To be able to rectify this, we could begin by just working with more of the partials, even the lower energy ones. There's also gonna be some noise in the signal that must be modeled to really get that authentic glockenspiel sound. To get this, we could try to use deconvolution to get a signal filter that approximates the differences between the real and the generated tone and adds it onto the output tone to get a more life-like sound.

Another problem is the pitch shifting. While it does sound better than just speeding up/slowing down a sound file, there are certain problems that arise when you try to shift the frequencies too far. From testing, I found that the sound generated by this program sounds best when you keep the pitch changes within an octave, especially when pitching down. Pitching down more than an octave led to a sound that wasn't too pleasant, however when pitching up you have a bit more leeway. The reason for these oddities is that all relationships between partials don't scale linearly, and this is more noticeable at larger pitch differences. These formants can make the sound odd. To improve this, the best thing to do is probably to use more input samples at different octaves, since then you would never have to pitch a sound up or down more than an octave. Outside of this, it can help to be more precise with the partials that are saved and which ones are discarded.

Finally, we can consider the envelope. While we do calculate a simple envelope with the envelope parameters, there is no attack to it. This means that the tone immediately starts without

any buildup - something that also adds to the reason why the synthesized hard hit doesn't sound like the real one. A way to fix this is to implement a more complex envelope, such as the ADSR envelopes that were talked about during the lectures.