Benjamin Ledoux
Prof. Ozgur Izmirli
COM312
December 16, 2022

Estimating Multiple Fundamental Frequencies

My final project was implementing my own method of estimating multiple concurrent fundamental frequencies, based on my research done for the paper. The `main()` function goes through all of the provided sound files of different chords, plugging each one into the primary function. The project is split up into two primary functions. In order to run the project, make sure the sounds folder is included in the file path in MATLAB so the `main()` function can access the sound files. If you want to test other audio files, just change the name of the file in `estFundFreq(signal)` to whatever you want. I tested a file that goes through the different notes in a chord with varying levels of concurrency (playing notes D, F, A, D and F, F and A, D and A, and finally all three at once), as well as a set of many different chords being played. The results were promising. There are some small errors in the estimation, and I attempted to fix them with a post-processing filter that would eliminate notes that only existed for a single frame. This seemed to help a little bit but left a few errors, most likely a result of the `find_fund_mult()` function. Overall I am happy with the implementation. It draws on information gathered in my research, has a strong output, and is based on what I've learned this semester.

`find_fund_mult(spectrum, sFreq, fft_size)`

This function takes in a frame of a spectrogram performed on a sound file, the sampling rate of the sound file, and the size of the Fast Fourier Transform performed. Starting from the lower frequencies, it goes through the frame and identifies peaks in amplitude, marking the frequencies associated with those peaks as significant frequencies (audible sound). The resulting list of significant frequencies is then analyzed and any frequency determined not to be double that of another element (a harmonic partial) if concatenated into a new list. Due to the nature of performing a FFT, frequencies within a musical quarter tone of a partial of another frequency are considered partials. This method does not take amplitude into account, and thus cannot detect fundamental frequencies that overlap with partials of other fundamental frequencies (a sound signal with F0's at 100 Hz, 150 Hz, and 200 Hz will return a list of 100 Hz and 150 Hz).

`estFundFreq(signal)`

This function takes in a sound file, finds the spectrogram of the file, then uses `find_fund_mult()` on each frame of the spectrogram. The results are concatenated to a matrix of fundamental frequencies, FUND, where each F0 is assigned to its own associated column (new F0's create new columns). The result is a matrix of fundamental frequencies ordered as they appear, with the rows denoting each frame of the spectrogram (a header vector is made to easily determine the associated columns of each F0). Wherever a fundamental frequency is not present in the file, a 0 serves as a placeholder.

The FUND matrix is then converted into a matrix of MIDI notes, which is then filtered to omit frequencies deemed out-of-range (in this case, A5 was determined to be the upper limit, non-inclusive). The MIDI matrix is converted into a set of vectors denoting the MIDI number of the note, the start time in seconds, and the duration in seconds. It is filtered so consecutive notes will be combined into longer notes. Notes are added by going through each column and processing the integers present, meaning the final vectors will be stacked and ordered by MIDI note in order of appearance, not ordered by time. The vectors are then incorporated into a matrix, which is then filtered to omit non-valid notes (-Inf) and notes that only last for a single frame of the spectrogram to reduce short-term errors. The resulting matrix is readable by `matrix2midi()`, converted into a MIDI object, and written to a MIDI file using the name of the original sound file.

It should be noted that the large blocks of commented-out code are the implementation of a direct approach of turning the FUND matrix into an audio file using `generate_note()`, hence the file's presence in the project folder. This produced some splicing issues where between each generated note, the length of which were determined by the width of the spectrogram, a spike in white noise was present. A fix was attempted to combine consecutive notes before generating them, however the MIDI approach proved much better in the end.