CS591 S1

Yida Xu, Wenjun Shen

xyds1522@bu.edu,[wenjun@bu.edu](mailto:wenjun@bu.edu)

Summary of Chord Recognition Project

In this project, we are doing chord recognition, which takes in a .wav file and outputs a graph showing what piano notes are of each individual sound signal in the file. Within Spyder, simply put a .wav file in the same directory as of chordRecognition.py and change the global variable filename at the top of the program. Use Python Console instead of iPython Console to run the program so that the output graph is large enough with details. Specific usage of functions and variables is written in comments in the program.

Now let me introduce the logic of this program. After reading in a wave file, mu-Law is used to eliminate all trivial frequencies within the array so that these noises will not affect the accuracy of latter note detection. Then, set window size to 216 and window slide to 214 (these two numbers work the best after trying different set of values). Within each window, FFT is performed to extract all frequencies and their corresponding amplitudes. Filtering by eliminating all frequencies with amplitudes lower than 350 (this value is chosen after trying different possibilities). Until now, the only thing needs to be done is finding each frequencies’ octave and note. To achieve this goal, I create a variable named deviation. For each frequency o within the third octave, the smaller the absolute value of round(ln(f/o))-ln(f/o) where f is the frequency waiting to be assigned a note, the closer f is to that specific o’ supposing they were all in the third octave. Therefore, f’s note is found, but we still don’t know what octave f is in. To know this, simply compute round(ln(f/o’) and the result is the span. However, we need to make sure f is within the range between A0 and C8. 2span \* o’ gives the exact note on a piano the closest to f. After repeating the same process for each frequencies of the filtered output of FFT, accurate notes thus are detected. All it has to be done is plotting the final results. Figure 1 and Figure 2 below shows the output graph on file PianoChordsElectric.wav and FminorPart2.wav. Download Addresses are listed on the last page.

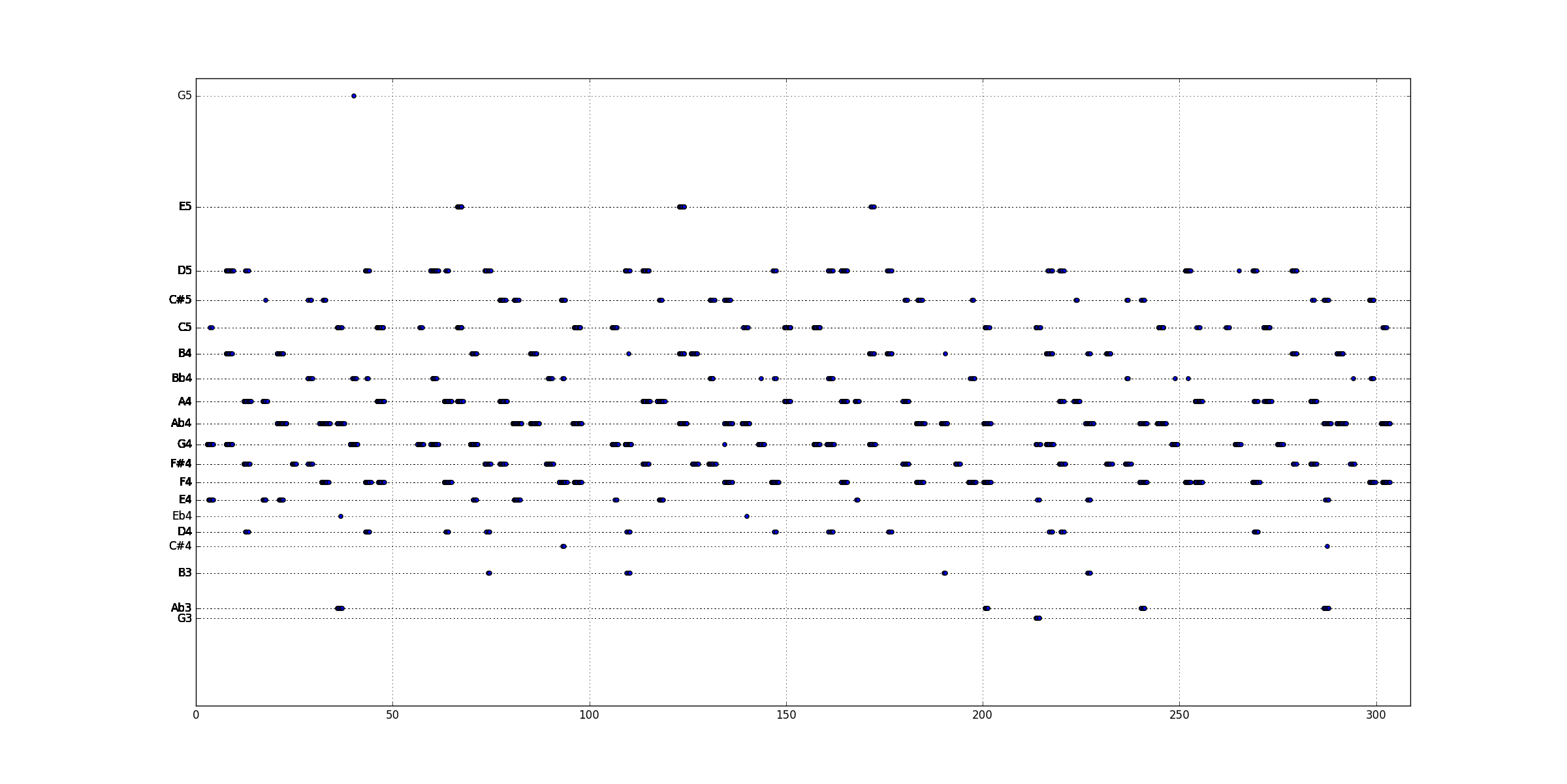


Figure 1.

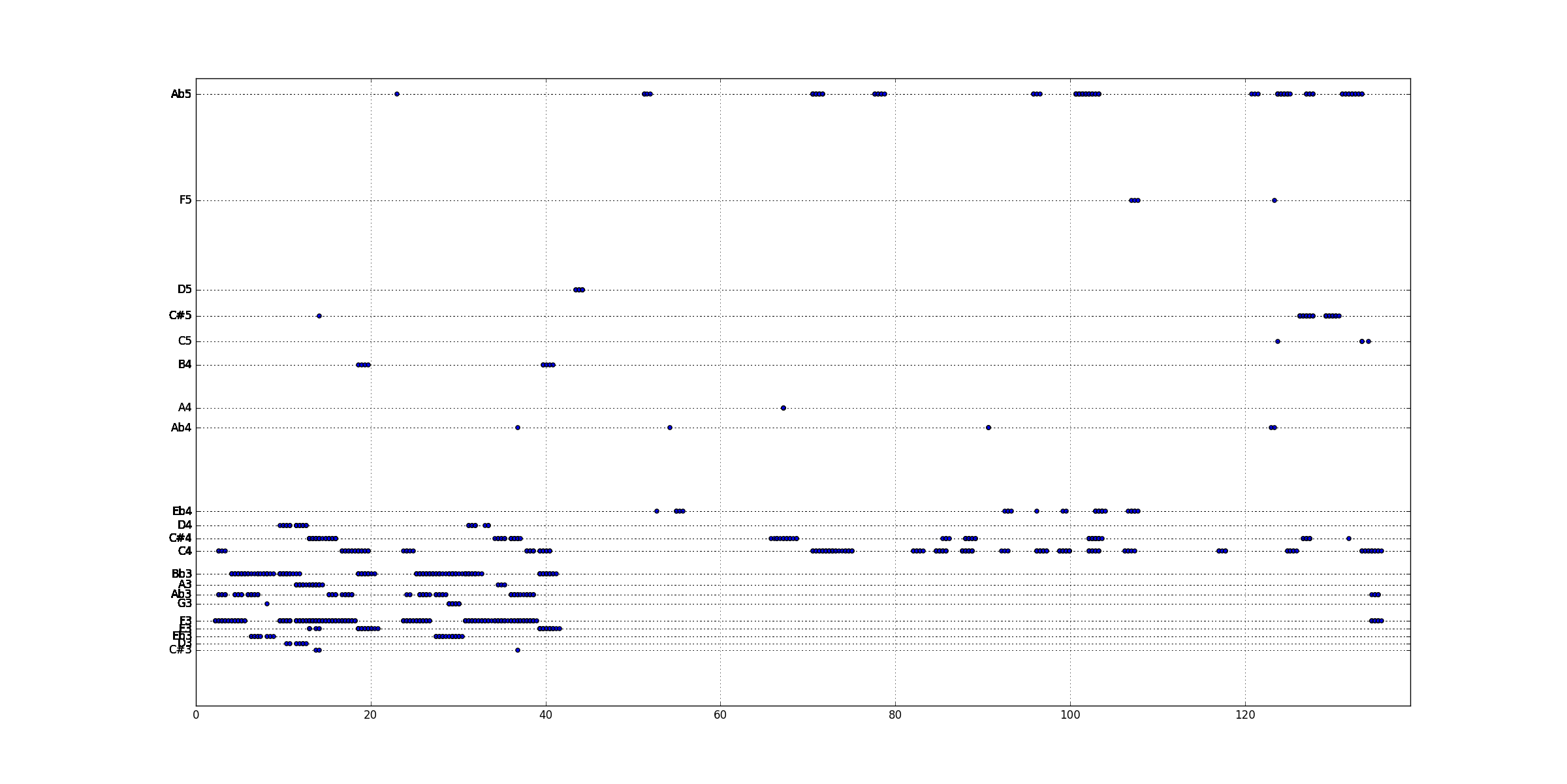


Figure 2.

Indeed, there are troubles encountered when doing this project. For example, chord G3D4E4 is detected by our program as G3D4E4G4. The biggest problem is that FFT sometimes assigns considerable amount of amplitude to the frequency that is one octave higher than the actual frequency. The best solution we have found is using mu-Law at the beginning of the program which smooths the signal and eliminates noises. It is obvious that only using mu-Law cannot completely solve this issue, but the output graph is pretty accurate for majority of the signals. Also, different values of window size and window slide will affect the output. Small values favor high speed music, vice versa.

Download Addresses:

FminorPart2.wav (Needs to be converted into mono file)

<https://archive.org/download/pianopieces_201312/Fminor%20Part%202%20wav.>wav

PianoChordsElectric.wav

<http://www.cs.bu.edu/~snyder/cs591/AudioSamples/PianoChordsElectric.wav>

Online Converter: <http://audio.online-convert.com/convert-to-wav>

Sources used:

mu-Law <http://www.ti.com/lit/an/spra163a/spra163a.pdf>

Piano key frequencies <https://en.wikipedia.org/wiki/Piano_key_frequencies>

Code distributions <http://www.cs.bu.edu/~snyder/cs591/CodeDistributions/>