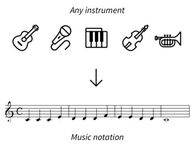
Maximum Likelihood Chord Detection

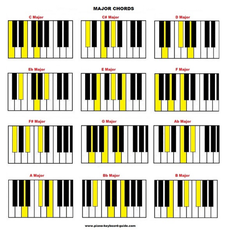
Yujia Qiu, Professor James W. Beauchamp
Department of Electrical and Computer Engineering, College of Engineering, University of Illinois at Urbana-Champaign

INTRODUCTION

As a fan of music and a pianist, I always wonder if there is a software to automatically generate the score of a song perfectly, it would be highly convenient.



The very first thing of music transcription is to do the chord detection.



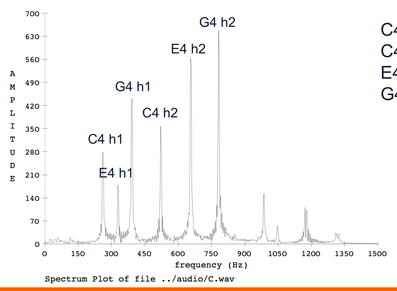
This project is to process input audio files and recognize one of 12 major chords, including C, C#, D, Eb, E, F, F#, G, Ab, A, Bb, B.

The whole project is mainly based on the FFT and the Maximum Likelihood methods, to find the three most likely notes in the input audio. Based on these notes, a decision is made about whether the file contains a valid chord and if so which chord it is. Piano triad chords were used to test the algorithm. The algorithm was implemented as the C program chordrecog which ran under Unix.

METHOD

1. FFT

Fast Fourier transform algorithm (FFT), was used to compute the discrete Fourier transform of an audio sample sequence. This method converts a signal from its original time domain to a representation in the frequency domain. This provides a means for measuring the magnitude of each frequency in the audio file. A typical chord spectrum is shown below, 8192(2^13) samples were used as the input to the FFT, which resulted in 8192 frequency bins.



C4 major:
C4 - 261.626 Hz
E4 - 329.628 Hz
G4 - 391.995 Hz

METHOD

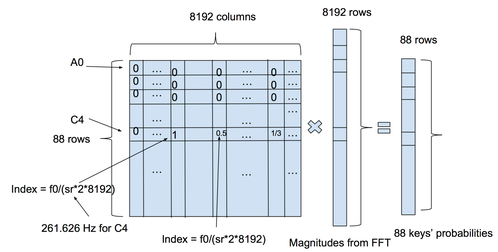
2. Maximum Likelihood Method

Maximum Likelihood is a method for estimating the parameters of a statistical model given multiple observations, by finding the parameter values that maximize the likelihood of making the observations given certain parameters. In this project, I used a matrix whose size was 88x8192, containing the frequency information for the 88 keys of a piano. Each row represents the corresponding key in a piano. For example, row 1 means A0 in a piano. 8192 represents the number of bins in the signal frequency spectrum. For each row, each column corresponds to the estimated magnitude of the frequencies contained in one note. An equation to compute the column index corresponding to each spectral frequency is given by

$$index = \left(\frac{f}{sr}\right) \times 2 \times 8192 \text{ (sr is the sample rate).}$$

(Nyquist rate)

Every note is the combination of several harmonic frequencies, including f0 (fundamental frequency), 2*f0, 3*f0… The weight for each nth harmonic of f0 was chosen assumed to be 1/n. These weights are assigned into their corresponding cells of the matrix, according to their frequency indices. The remaining cells are set to zero.



The inner product of the 88x8192 matrix and the 8192x1 FFT column vector yields an 88x1 vector, which gives the estimated probability of each note to become a note contained in the input audio.

In this project, the matrix is very sparse. Therefore, I did not in fact create a matrix but instead for each harmonic of each piano note directly computed the column index and retrieved the corresponding spectral magnitude which was then multiplied by its corresponding weight.

RESULTS

After obtaining the 88x1 vector z, the first thing is to find the three highest values in the vector, because a triad chord consists of 3 notes. After getting the corresponding indices, three most likely notes in the audio are easily found.

After determining the most likely three notes, the next goal is to check whether the three notes match one of the chords in the collection of 12 major chords. In this project, I only consider the chords made of three consecutive notes.

Four Possible Results

```
Three most likely notes in the audio:
note:E4 probability:1.000000
note:C5 probability:0.839892
note:G4 probability:0.403658
This is C chord
```

Three Notes Match: The three notes exactly match those in the input (within an octave). For the second, so it is declared to be a C chord.

Two of Three Notes Match, No Foreign Note: Here the C is missing and the G is duplicated, but there are no non-C chord notes.

```
Three most likely notes in the audio:
note:G5 probability:1.000000
note:E5 probability:0.870592
note:G4 probability:0.651155
This is C chord
```

```
Three most likely notes in the audio:
note:E4 probability:1.000000
note:C4 probability:0.984503
note:F#4 probability:0.918432
This might be C chord or chord can't be recognized
```

Two of Three Notes Match, One Foreign

Note: Here is G is missing and is replaced by an F#4. This may or may not be a C chord.

Only One Note Matches: No two notes are in the same chord, so chord cannot be recognized.

```
Three most likely notes in the audio:
note:D4 probability:1.000000
note:C#4 probability:0.953634
note:C4 probability:0.910691
Chord can't be recognized!
```

Code Base

```
for(i=0;i<row;i++){
    f0=fre[i];
    amp[i]=0;
    n=1;
    while(f0<.5*sr){
        amp[i]+=(1./n)*z[(int)(f0/sr*2*col)];
        f0+=fre[i];
        n++;
    }
}
```

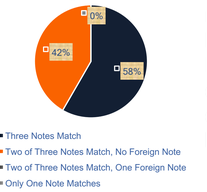
fre is an array storing the 88 fundamental frequencies of the 88 keys, row is 88, z is the 8192x1 FFT column vector, and amp is the array of output vector array, which gives the estimated probability of each note occurring in the audio input.

```
const float fre[88] = {27.5, 29.135, 30.868, 32.703, 34.648, 36.708, 38.891, 41.203, 43.654, 46.249, 48.999, 51.913,
    55.0, 58.27, 61.735, 65.406, 69.296, 73.416, 77.782, 82.407, 87.307, 92.499, 97.999, 103.826, 110.0, 116.541,
    123.471, 130.813, 138.591, 146.832, 155.563, 164.814, 174.614, 184.997, 195.998, 207.652, 220.0, 233.082,
    246.942, 261.626, 277.183, 293.665, 311.127, 329.628, 349.228, 369.994, 391.995, 415.305, 440.0, 466.164,
    493.883, 523.251, 554.365, 587.33, 622.254, 659.255, 698.456, 739.989, 783.991, 830.609, 880.0, 932.328,
    987.767, 1046.502, 1108.731, 1174.659, 1244.508, 1318.51, 1396.913, 1479.978, 1567.982, 1661.219, 1760.0,
    1864.655, 1975.533, 2093.005, 2217.461, 2349.318, 2489.016, 2637.02, 2793.826, 2959.955, 3135.963, 3322.438,
    3520.0, 3729.31, 3951.066, 4186.009};

const char *note[88]={"A0", "A#0", "B0", "C1", "C#1", "D1", "D#1", "E1", "F1", "F#1", "G1", "G#", "A1", "A#1", "B1", "C2",
 "C#2", "D2", "D#2", "E2", "F2", "F#2", "G2", "G#2", "A2", "A#2", "B2", "C3", "C#3", "D3", "D#3", "E3", "F3", "F#3",
 "G3", "G#3", "A3", "A#3", "B3", "C4", "C#4", "D4", "D#4", "E4", "F4", "F#4", "G4", "G#4", "A4", "A#4", "B4", "C5",
 "C#5", "D5", "D#5", "E5", "F5", "F#5", "G5", "G#5", "A5", "A#5", "B5", "C6", "C#6", "D6", "D#6", "E6", "F6", "F#6",
 "G6", "G#6", "A6", "A#6", "B6", "C7", "C#7", "D7", "D#7", "E7", "F7", "F#7", "G7", "G#7", "A7", "A#7", "B7", "C8"};
char *nonchord = {"!"};
char *chordname[12]={"C","C#","D","Eb","E","F","F#","G","Ab","A","Bb","B"};
const char *chordnote[12][3]={{"C","E","G"},{"C#","F","G#"},{"A","D","F#"},{"A#","D#","G"},{"B","E", "G#"},
                              {"F", "A", "C"},{"F#", "A#", "C#"},{"G","B","D"},{"G#", "C", "D#"},
                              {"A", "C#", "E"},{"A#", "D", "F"},{"B", "D#", "F#"}};
const int majorchord[12][3]={{1,5,8},{2,6,9},{3,7,10},{4,8,11},{5,9,12},{1,6,10},{2,7,11},{3,8,12},{1,4,9},{2,5,10},{3,6,11},{4,7,12}};
```

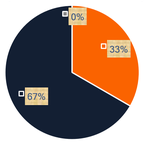
Analysis

For 12 major chords generated by GarageBand



For majors like Ab major, F# major, it's more possible to have "Two of Three Notes Match, No Foreign Note" detected, where there are duplicated notes, which is less accurate.

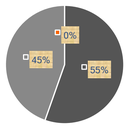
For 12 major chords recorded by a piano

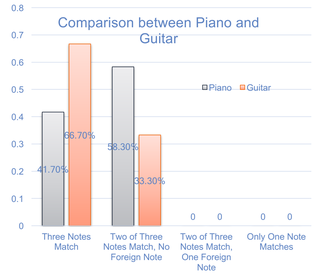


For audio recorded by pianos, the possibility of detecting "Three Notes Match" decreases, due to surrounding noise and difference between piano sounds and generated audio.

There is a high inaccuracy of detecting non chords, for the reason that there are often two pitches matching a chord.

For non chords





Compared to Piano, it is surprising that the accuracy of Guitar is higher. It also shows that the programs works for different instruments.

CONCLUSIONS

In a word, I am satisfied with the results obtained. But there are many improvements that could be made. First, I could increase my database so that my program could recognize more than 12 chords. Second, there are many chords which are not made of 3 notes. I did not consider this situation. To make the program more practical, there are many more things I would need to accomplish.

ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN