Music Transcription through Fourier Analysis

Thomas Slap

May 17, 2020

Abstract

Music provides a rich playground for signal processing as all music is nothing more than a the combination of sound waves. Through Fourier analysis we are able to measure the frequency of both the notes and rhythms in order to describe what is happening in the music. For this project, I created a program that takes a clip of music as an input and returns the notes that are played as well as the tempo of the clip. I will not outline step by step how my program is able to do this.

Windowing

The first step is to cut the inputted clip into many subclips of length with N samples and then to take the Fourier transform of each clip. The value chosen for N is very important because any given value will have a trade-off between how accurately a note can be identified versus how accurately the rhythm can be identified. This is because of the relationship between the length of a signal and how precise one can be at determining the frequencies from the signal. The smallest useable clip length depends on the lowest two notes that we want to be able to differentiate. Under the equal temperament tuning systems each note is given by $f_{n+1} = f_n * 2^{1/12}$. In order to identify a note , we can define a frequency as a given note if it falls between $f * 2^{-1/24}$ and $f * 2^{1/24}$ as this includes half of the interval between the notes one half step higher or lower from the note we're looking for. From this we can conclude that the number of samples per each clip, N, must obey $N > \frac{sr}{\Delta f}$ where $\Delta f = f_{min} * (2^{\frac{1}{24}} - 2^{\frac{-1}{24}})$ in order for the frequencies to be sufficiently precise to correctly identify the note corresponding to f_{min}. Using $N = 14080$ allows us to identify pitches as low as 29hz which is nearing the lower limit

of human hearing. At $sr = 44100$, a nearly universal standard for recorded music, this corresponds to a clip with a length of 0.32s. This corresponds to cutting the clip at a rate of 3.125hz which was too close for my liking to the frequency of quarter notes at 120bpm, a very common tempo for popular music. To increase rhythmic accuracy, I cut the original clip into overlapping segments to preserve pitch accuracy while boosting rhythmic accuracy. These overlapping windows are able to identify pitch like a 0.32s long clip, while also being able to identify rhythms like a 0.16s long clip.

After cutting the inputted clip into many sub-clips, I apply a Hann windowing function, $H(n) = \frac{1}{2} - \frac{1}{2}Cos(\frac{2\pi n}{N})$, in order to smooth out the sharp corners thereby reducing the high frequency noise that is produced from a square windowing function. I then take the fourier transform of each clip and discard the negative frequencies since for a real signal the negative frequencies are the same as their corresponding positive frequencies.

```
function FFTs = ClipWindower(clip,t,dt,sr)
  %This will split the clip into smaller clips of length dt
  windowLength = dt*sr;


  %slices inputed clip into subclips
  Windows = [];
  Clips = [clip(1:windowLength,:)];
  for n = 1: floor(2*t/dt)-2
   Windows = cat(3, Windows, clip( (n*windowLength)/2 : ((n+2)*windowLength)/2-1 , : ));
  endfor

  %Creates Hanning function
  X = [0:windowLength-1];
  Hann = (1/2)*(1-cos(2*pi*X/windowLength));
  Hann = transpose(Hann);
  Hann = cat(2, Hann, Hann);

  %Implements hanninng funnction
  for n = 1:size(Windows)(3)
    Windows(:,:,n) = Windows(:,:,n).*Hann;
  endfor

  %Takes the FFT of each window
  FFTs = [fft(Windows(:,:,1))];
  for n = 2:size(Windows)(3)
    FFTs = cat(3, FFTs, fft(Windows(:,:,n)));
  endfor


  %Takes only the positive frequencies
  FFTs = FFTs(1:size(FFTs)(1)/2,:,:);

  %averages two channels into one
  FFTs = (FFTs(:,1,:) + FFTs(:,2,:)) / 2;
  FFTs = squeeze(abs(FFTs));
endfunction
```

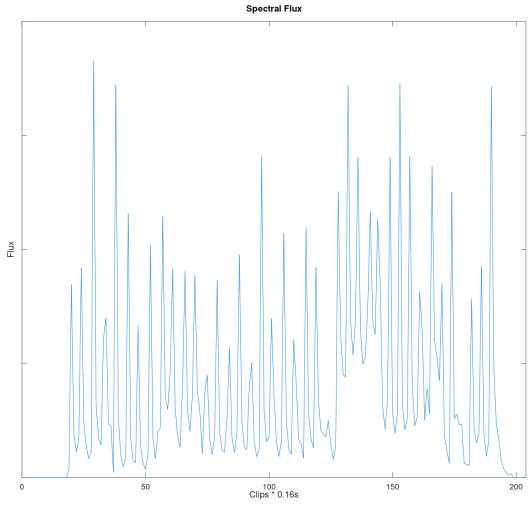
Figure 1: The described windowing function

Spectral Flux

Now that we have spliced the input and taken the Fourier Transform we can begin analyzing the music. In order to detect the onset of each note, we can take the spectral flux of the transforms. Spectral flux for clip i of length N with Fourier Transform f,

$$\Phi_i = \sum_{n=1}^{n=N} (f_{i+1}(n) - f_i(n))^2$$

is a measure of how the amplitude of each individual frequency changes over time. This is effective for identifying the onsets of music notes as the attack of a given instrument has many high frequency components that quickly decay so the start of each note corresponds to a large jump in the spectral flux.




```
function Flux = Flux(FFTs)
  sz = size(FFTs);
  Flux = [];

  %remove values that are only 0s

  for n = 1:sz(2)-1

    %Stops division by 0 for silennt clips
    if sum(FFTs(:,n)) == 0
      FFTs(1,n) = 1;
    endif

    %normalization
    FFTs(:,n) = FFTs(:,n) / sum(FFTs(:,n));

    %computes Spectral Flux
    Sum =0;
    Sum = sum(FFTs(:,n) - FFTs(:,n+1))^2;
    Flux = [Flux; Sum];

  endfor

endfunction
```

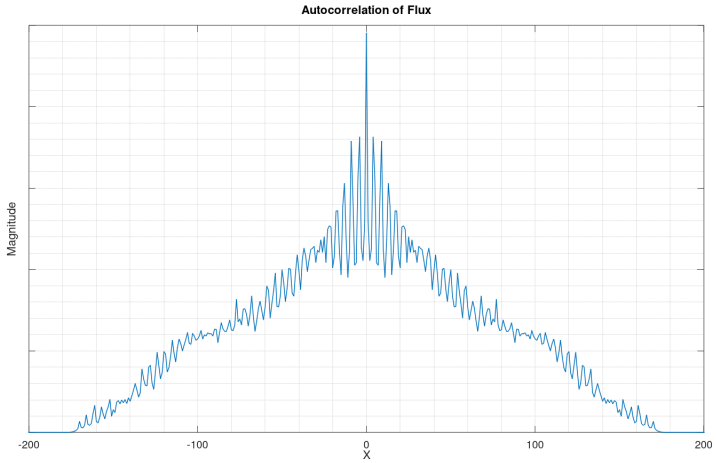
Figure 2: A plot of Flux and the code used to find Flux

Finding tempo

We can find the tempo of this song by convolving the spectral flux with itself. Convolution is defined as

$$(f \circledast g)(x) = \int_{-\infty}^{\infty} g(x - x')f(x')dx'$$

Convolving a function with itself, also known as autocorrelation, is a good way of determining frequency since when x is close to the period, $f(x - x')$ and $f(x)$ will constructively interfere producing a large value for $(f \circledast g)(x)$. There will also be larger values for integer multiples of the fundamental period. After taking the autocorrelation of the spectral flux, we can find the average space in-between peaks in order to find the tempo of the audio clip. Autocorrelation always has a maximum value at $x = 0$ because that effectively results in taking the integral of $f(x)^2$. Besides the peak at 0 the peaks of the autocorrelation generally respond to integer multiples of the tempo. Given a perfectly in time audio sample that contained only one value of note length, one would be able to merely find the value of first peak after 0 to determine the tempo. However, since this audio clip is not perfectly in time and because there are more complicated rhythms, it is best to take the average of the distance between each peak and to exclude some of the less prominent peaks.



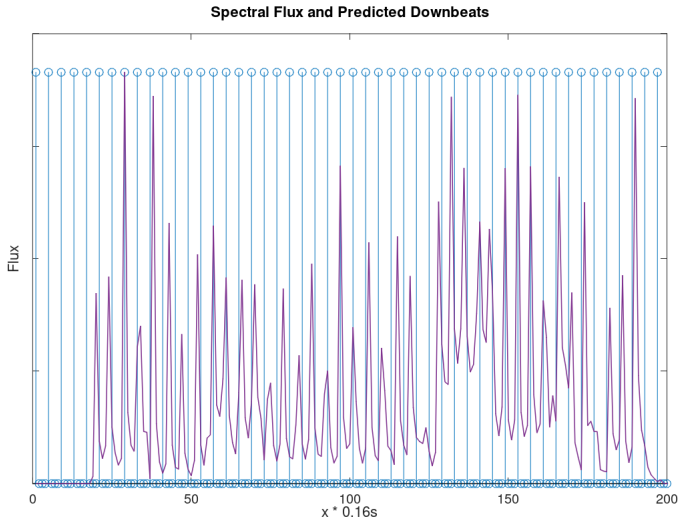


Figure 3: A plot of spectral flux with an overlaid stem plot of predicted downbeats

```
function Tempo = Rhythm(Flux, t, dt)


  szF = size(Flux);

  %Autocorrelation of Flux to find frequency
  [R, lag] = xcorr(Flux);
  szR = size(R);
  figure(2)

  %Finds large peaks which correspond to tempo and overtones
  [peak,spaces] = findpeaks(R,"MinPeakDistance", 2/(300/60*dt));
  szS = size(spaces)(1);

  %Finds the average distance between peaks to find fundemental frequency
  Kappa = 0;
  for n=1:szS-1
    Kappa = Kappa + spaces(n+1) - spaces(n);
  endfor
  avgSpace = Kappa/(szS-1);


  %Unit conversion from clips -> seconds
  Tempo= 1/(avgSpace*(dt/2))*60;

  %Create Comb at tempo to produce plot
  comb = zeros(1,round(avgSpace));
  comb(1) = max(Flux);
  comb = repmat(comb, 1,round(szF(1) / avgSpace));
```

Figure 4: This is the function used to identify tempo

Onset Detection

Continuing to use the spectral Flux, we can find the beginning of each note by finding the peaks in the spectral flux function. This is helpful because now that we know where each note begins, we can average all of the clips that correpsond to a single note in order to increase the accuracy of our pitch recognition. This also serves to clean up the output of the program as the value of each note will be displayed not the value of each clip.

```
function ts = SpecFlux(Flux, t, dt)

  sz = size(Flux);
  avg = sum(Flux)/size(Flux)(1);

  %Finds large peaks of flux for onset detection
  [mag, ts,w] = findpeaks(Flux, "MinPeakDistance",2/(200/60*dt), "MinPeakHeight" , avg/100);

  ts = [ts] ;
  ts = [1;ts;size(Flux)(1)]


endfunction
```

Pitch Recognition

The algorithm for detecting pitch is very similar to for detecting rhythm as once again we will take the autocorrelation in order to find the fundamental frequency. This time we are detecting the spacing between overtones of a given note. When a note is played on any instrument, the fundamental frequency is produced as well as some combination of integer multiples of the fundamental frequency. The timbre (read sound) of an instrument is dependent on the relative magnitudes of higher frequency overtones. To find pitch I first take the autocorrelation of the averaged Fourier transform for a given note and then find the large peaks in the autocorrelation. The spacing of these peaks corresponds to the frequency of the note being played.

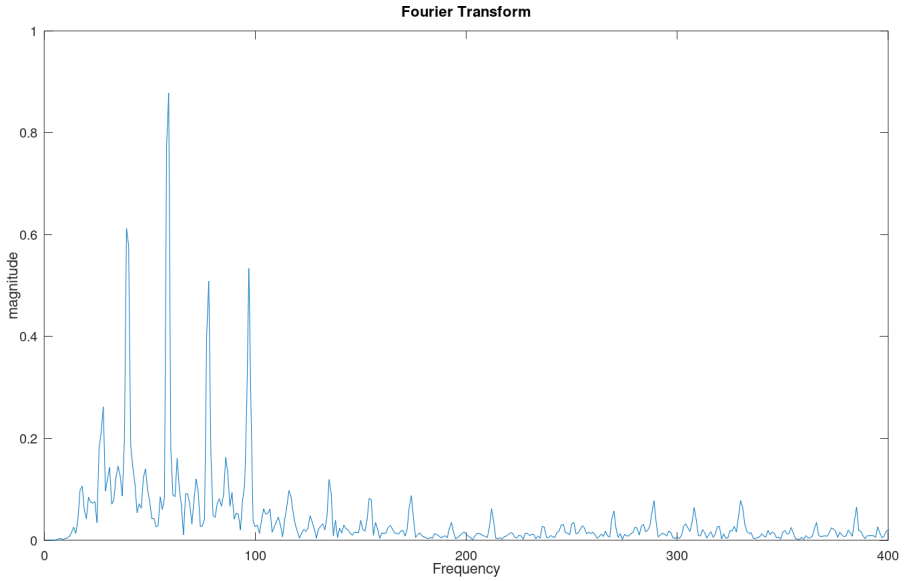


Figure 5: Here is the transform for a sample subclip

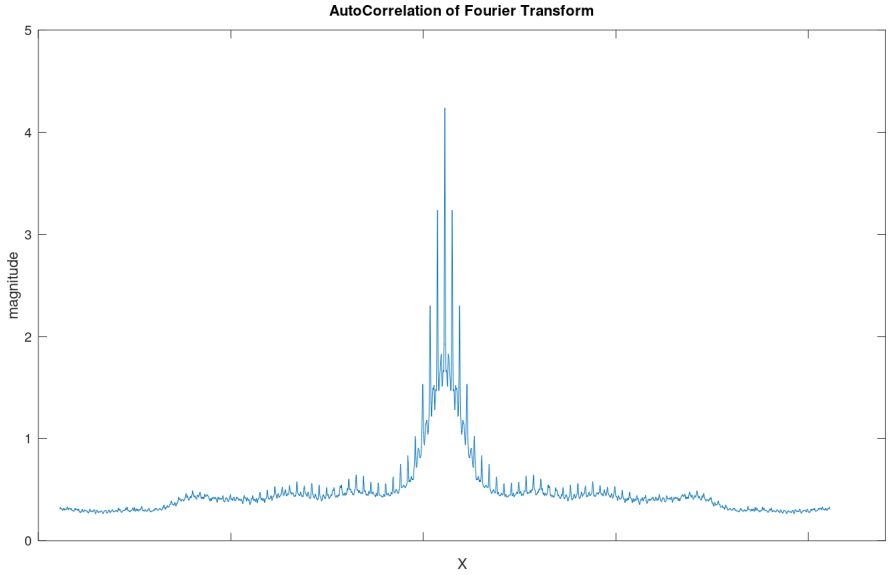


Figure 6: This is the autocorrelation of the above tranform

This is done through my pitch function.

```
function note = pitch(FFT,N,sr)

  %Autocorrelation of the fourier transform to pick out overtones
  [R , lag] = xcorr(FFT);

  %limits range to common frequencies to limit unnecessary computation
  szR = size(R);
  R = R(round(szR(1)/2 -1000):round(szR(1)/2 +1000));
  szLag = size(lag);
  lag = (szLag(2)/2 - 1000:szLag(2)/2 + 1000);


  %Picks out large peaks from the Autocorrelation
  [val , pitch] = findpeaks(R, "MinPeakDistance", 20, "MinPeakHeight",max(R)/10 );


  %Finds average spacing inbetween peaks
  Kappa = 0;
  for n=1:size(pitch)-1
    Kappa = Kappa + pitch(n+1) - pitch(n);
  endfor
  avgPitch = Kappa/ (size(pitch)(1)-1) ;

  %Unit conversion to hz before sending the note to be named
  freq = avgPitch * (sr/2)/size(FFT)(1);
  note = NoteIdentifyer(freq);
```

Included in this function is the NoteIdentifyer function which gives a name to the frequency that is determined to be the fundamental of the note.

```
function Note = NoteIdentifyer(f)


  %finds interval from C4
  n = 12*log2(f/261.63);

  %Finds the difference in octave and difference from C within said octave
  DelOctave = floor(n/12);
  DelNote = round(n - 12 * DelOctave);

  Octave = 4 + DelOctave;

  if DelNote == 0 || DelNote == 12
    Note = 'C';
    elseif DelNote == 1
      Note = 'C#/Db';
    elseif DelNote == 2
      Note = 'D';
    elseif DelNote == 3
      Note = 'D#/Eb';
    elseif DelNote == 4
      Note = 'E';
    elseif DelNote == 5
      Note = 'F';
    elseif DelNote == 6
      Note = 'F#/Gb';
    elseif DelNote == 7
      Note = 'G';
    elseif DelNote == 8
      Note = 'G#/Ab';
    elseif DelNote == 9
      Note = 'A';
    elseif DelNote == 10
      Note = 'A#/Bb';
    elseif DelNote == 11
      Note = 'B';
   end

  Note = strcat('(',Note,num2str(Octave),')');

endfunction
```

Final Product

All of this comes together in my Transcriber function to produce a list of the notes that are played as well as a tempo. For a recording of the author playing a G scale exercise on the guitar, this is the function and the final output. You can listen and follow along to the audio file that I sent in an email earlier this evening.

```
function Transcriber(clip,t,dt,N,sr)
  %Makes everything two track
  if size(clip)(2) ~= 2
    clip = [clip,clip];
  endif


  Clips = ClipWindower(clip,t,dt,sr);
  Flux = Flux(Clips);
  ts = SpecFlux(Flux,t,dt);
  Tempo = Rhythm(Flux,t,dt)
  FluxFinder(Clips,ts,N,sr)


endfunction
```

```
Tempo =  95.270
warning: implicit conversion from numeric to char
warning: called from
    FluxFinder at line 186 column 11
    Transcriber at line 19 column 3
Notes =

(G#/Ab2)
(G2)
(G2)
(G2)
(G2)
(A2)
(A2)
(A2)
(A2)
(B2)
(B2)
(B2)
(B2)
(C3)
(C3)
(C3)
(C3)
(D3)
(D3)
(D3)
(D3)
(E3)
(E3)
(E3)
(E3)
(F#/Gb3)
(F#/Gb3)
(F#/Gb3)
(F#/Gb3)
(G3)
(G3)
(G3)
(G3)
(G3)
(G3)
(F#/Gb3)
(E3)
(D3)
(A#/Bb2)
(B2)
(A2)
(G2)
(G2)
```

Here there are only two incorrectly identified notes, the G-sharp at the beginning and the A-sharp near the end, as well as few repeated notes that are actually the same. While this program is not perfect, it comes close to

being able to transcribe the pitches of a monophonic recording and give the tempo.

Going forward I would like to increase the capabilities of this program in order to be able to recognize multiple notes at once as well as fitting the notes to the actual rhythm rather than just finding the times they are played and the tempo.

In addition, the tempo identification was able to correctly identify the tempo of Lovesong by Sarah Bareilles and Purple Haze by Jimi Hendrix to within 1-2 bpm of the actual tempo as counted out by the author.

Sources

$https://www.ee.columbia.edu/\ dpwe/pubs/MuEKR11-spmus.pdf$: This source gave me the idea of using autocorrelation in order to identify tempo as well as using spectral flux for onset detection.

$https://ccrma.stanford.edu/\ pdelac/154/m154paper.htm_ftnref4$: This source gave me the idea to use autocorrelation for pitch.

$https://pages.mtu.edu/\ suits/notefreqs.html$: This source contains a list of every note and its corresponding frequency which proved invaluable while troubleshooting.