Ryan Sigar (Partners: SJ Park, Jeong Min Lee)
ECEN 4632: Introduction to Digital Filtering
Final Project Part 2

Solutions to Question 4:

"*Implement the computation of the Normalized Pitch Class Profile (NPCP), described by (22). You will compute a vector of 12 entries for each frame n."*

The code that computes the NPCP matrix is defined in the NormPitchProfile() function described in the Appendix. The NPCP matrix is a 12 x k matrix, where k is the number of frames contained in the song. The NormPitchProfile MATLAB function evaluates the PCP of each frame, by looking at its frequency content, determining the prevalent notes played at the given frame, and creating a 'heat map' (using a weight function) detailing how often the notes occur (taking into account how much energy a note contained when played).

Solutions to Question 5:

"Evaluate and plot the NPCP for the 12 audio tracks found in http://ecee.colorado.edu/~fmeyer/.private/audio.zip"

The results of evaluating the NPCP of the 12 given audio tracks are shown and discussed in the following section.

Data and Discussion

Below are graphical representations of the NPCP matrix calculated using 50 percent overlapping frames of size 2048 (samples), and hann windowing. After observing the figures, it is again apparent that songs of the same genre contain similar structure due to similar note variations and quantity of the notes played at each frame. Results from part 1 will be used to aid the discussion of the figures below.

Note: The figures below plot the NPCP coefficients vs the Frame Number. To plot NPCP vs. time (which makes more sense), we can take the frame number, multiply by the frame size, and divide by the sampling rate 11,025Hz to obtain the time (in seconds) that the beginning of the frame was played at. Then we can map the frame numbers to their respective times.

Ryan Sigar (Partners: SJ Park, Jeong Min Lee)
ECEN 4632: Introduction to Digital Filtering
Final Project Part 2

Classical

The song in figure 1 was played at a high tempo, with very quick variations in notes. Upon inspecting of figure 1, it is apparent that there exists a high rate of "scattering" of what notes are played in each frame of the song. The scatter rate is also higher than what is shown in figure 2, where more spaces in the NPCP are left vacant (as evident by the blue areas in the figure). This can be explained by making observations of the energy in the audio. The song in figure 2 was played by a piano at relatively lower energies than the violin in figure 1. This element of scattering can potentially be used to identify music in the classical genre.

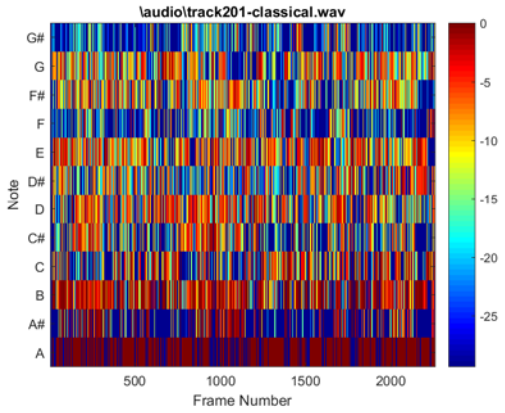

Figure 1

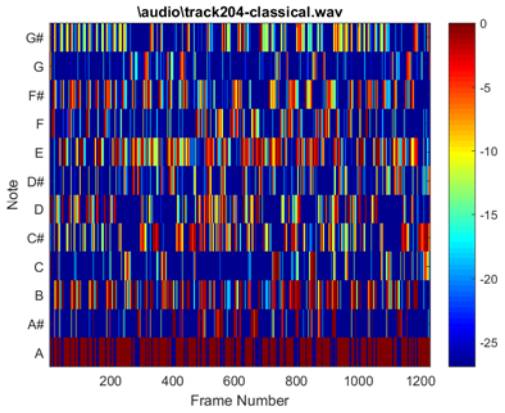

Figure 2

Jazz

The NPCP matrix evaluated for the song in figure 4 demonstrates that the song has a relatively high amount of energy in comparison to the song in figure 3 due to the amount of color depicted in the plot. Both songs played were reliant primarily upon the keyboard and drums. However, the song in figure 3 is more relaxed than that of figure 4. One interesting characteristic of these two figures and songs is the appearance of a band of vacant areas at different locations of the songs. Specifically, there exist relatively empty bands located near the middle of the song in figure 3 and at about 75% of the song in figure 4. This empty band is the result of solos added to the musical piece. This 'empty' band attribute may be used to identify Jazz music.

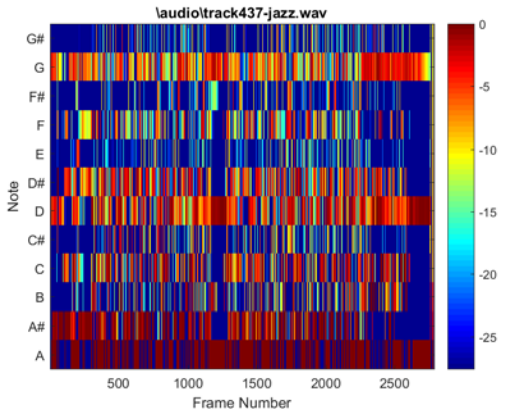

Figure 3

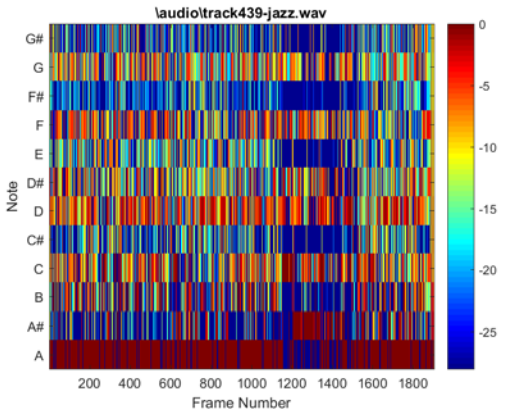

Figure 4

Ryan Sigar (Partners: SJ Park, Jeong Min Lee)
ECEN 4632: Introduction to Digital Filtering
Final Project Part 2

Electronic

The song played in figure 5 was a highly repetitive tune, carrying high tempo, lacking in bass, and energy (although very colorful in the figure). The high amount of color in the figure may be due to the song's consistent lack of energy, which may have caused the threshold value, used in finding the peaks of a given frame, to drop to values that will allow (mostly) all detected peaks in the frame to contribute to its PCP matrix and result in an 'oversaturation' when tallying the presence of notes in the frame. The "scattering" factor of the NPCP matrix changes slightly when the song begins to change in the last quarter of the song to a lower tempo. The song in figure 6 contained more bass, a more consistent tempo, as well as a voice added in the second quarter of the song. If studied closer, one could associate the consistent bass with one of the prominent notes: G# or D# (most likely D# due to its high energy and presence in the song). Potentially, the change in tempo as seen in figure 5, consistent bass as in figure 6, and low variation can be used to identify Electronic music.

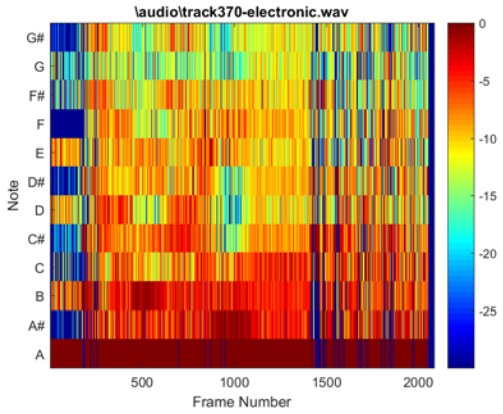

Figure 5

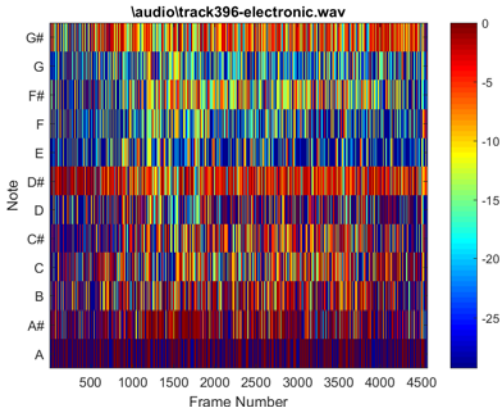

Figure 6

Rock

The most prominent features of the song used in figure 7 were the singer, consistent drum beat, and an electric guitar. In comparison to the song used in creating figure 8 (a sad song played in a minor tone), the song in figure 7 is more joyful, and it is apparent through the amount of color displayed in the image. It is difficult to identify any patterns through these images, but what is possible is the identification of the top notes played throughout the piece (i.e. E and C# in fig. 7 and D# and B in fig. 8).

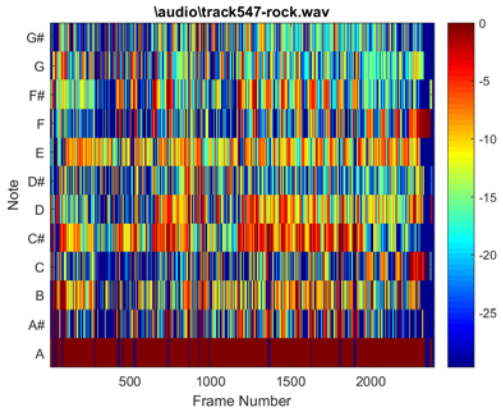

Figure 7

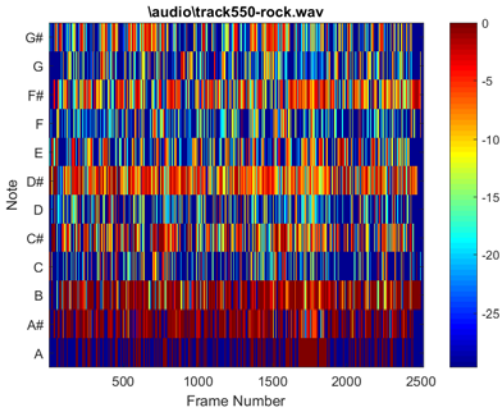

Figure 8

Ryan Sigar (Partners: SJ Park, Jeong Min Lee)
ECEN 4632: Introduction to Digital Filtering
Final Project Part 2

World

In comparison to the other figures shown, the NPCP plot for figure 9 is very empty (Perhaps a better threshold value for prominent note identification should have been used. The implementation of the NPCP identifies the highest note played at a given frame and sets the threshold to be -20dB below the maximum point). This can be explained by the slow pace, and small variation in notes when listening to the Japanese flute. However, in this figure, it is apparent that the E note was played consistently throughout the song. The song analyzed in figure 10 contained music played by an acoustic guitar, and what may be a sitar (and a few other string instruments). Throughout most of the piece, the notes the string instruments played deviated very infrequently. Interestingly, it was obvious that each instrument was designated to play around its 'own' note, and this can be seen by the consistent coloring of the notes G, D, and C in the figure. This reliance on only a few notes may be used to identify World music.

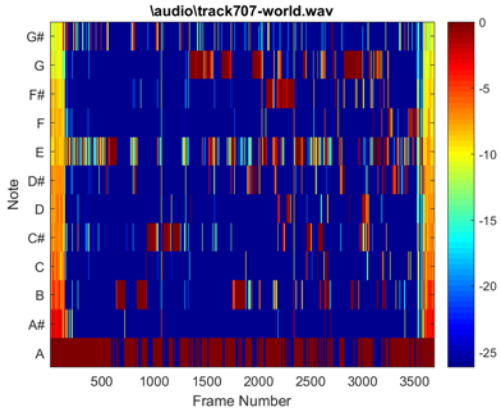

Figure 9

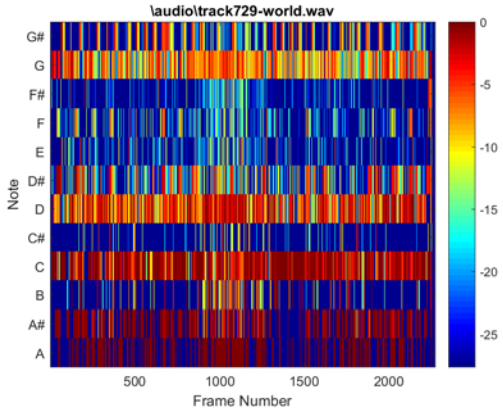

Figure 10

Metal

The songs used to generate figures 11 and 12 contained heavy usage of the electric guitar, and a consistent drum beat. Because of this fact, it is possible to narrow these instruments for being responsible for contributing to the high amount of A and A# notes played in these songs. The song used in figure 12 was primarily an electric guitar piece, while the song in figure 11, was primarily a more balanced combination of voice, drum beat, and the electric guitar.

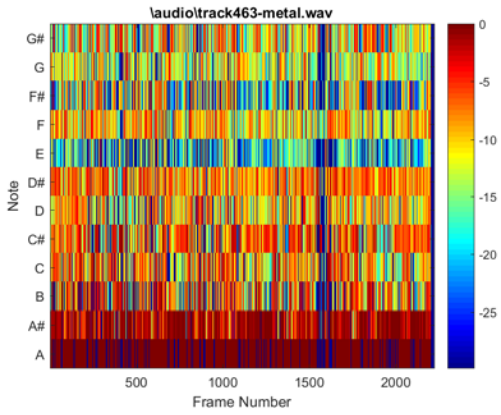

Figure 11

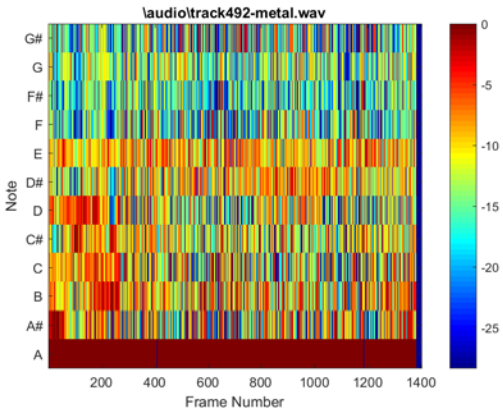

Figure 12

Ryan Sigar (Partners: SJ Park, Jeong Min Lee)
ECEN 4632: Introduction to Digital Filtering
Final Project Part 2

Appendix

How Part 2 of this Project was Completed: NormPitchProfile.m, script.m and a subfolder 'audio' are placed into a folder which is added to the workspace in MATLAB. Script.m then computes the PCP and NPCP by calling on the function NormPitchProfile() defined in NormPitchProfile.m.

NormPitchProfile.m

```
function [PCP, NPCP] = NormPitchClassProfile(song_path, evaluateOneFrame,
evaluateWholeSong)

    [wav]= audioread(song_path);

    %Number of notes in an octave
    notes_in_an_octave = 12;

    %Sampling rate of songs in Hz
    fs = 11025;

    %Frame size. Want a frame of size 2048
    frameSize_N = 2048;

    songSize = size(wav,1);

    %Create a window of size frameSize_N
    w = hann(frameSize_N);

    %Reference note in Hz
    f0=27.5;


    %jump = songSize/256;
    step_size = frameSize_N/2;


    %Determining the number of frames in a song
    framesInSong =  2*floor(songSize/frameSize_N)-1;

    %Finding first nonzero sample of song (used for part 1)
    first_nonzero_index = min(find(wav ~= 0));

    %take frameSize_N samples starting from the first non-zero point (add 512 samples
to this)
    %xn = wav(first_nonzero_index:first_nonzero_index+(frameSize_N-1));

    %Begin counting frames
    %frameNumber = 0;

    %Index of the last full frame of the song (to prevent overflow)
    startingIndexOfLastFullFrame = framesInSong*step_size;

    %PCP information gets stored inside NPCP matrix after evaluating each frame
    NPCP = zeros(notes_in_an_octave,framesInSong);

    %Process the whole song
    for frameStart=1:step_size:startingIndexOfLastFullFrame

        %Compute frame number from start frame
        %divide start frame by step size, add 1
        frameNumber = ((frameStart -1)/step_size) + 1;
```

```
        %determine the end sample index
        frameEnd = frameStart + (frameSize_N-1);

        %Extract one frame
        xn = wav(frameStart:frameEnd);

        %Take Fourier transform of signal using hann window, w
        Y = fft(w.*xn);
        K = frameSize_N/2 + 1;
        Xn = Y(1:K);


        %Finding peaks in fourier transform
        [peakMagnitudes,peakLocations] = findpeaks(abs(Xn));

        %Use a threshold, to extract only the most prevalent peaks
        %Wait, what's a decibel?? -> https://en.wikipedia.org/wiki/Decibel
        attenuation = 20; %decibels
        threshold = max(peakMagnitudes)*10^(-attenuation/20);

        %Re-evaluate where peaks occur (if we have peaks)
        if peakLocations
            [peakMagnitudes,peakLocations] =
findpeaks(abs(Xn),fs,'MinPeakHeight',threshold);
        end


        %Number of peaks = number of peak points detected
        numberOfPeaks = length(peakLocations);

        %Vector containing the value of the frequencies that the peaks occur at
        peakFrequencies_f = zeros(numberOfPeaks,1);

        %Matrix containing semitone values computed using the frequencies that the
peaks
        %occur at
        semitone_sm = zeros(numberOfPeaks,1);

        %PCP is reset at beginning of every frame
        PCP  = zeros(notes_in_an_octave,1);


        %Computing the frequency at the peaks
        %indexed according to peak number
        for k=1:1:numberOfPeaks
            %frequency = (index that peak occurs at * sampling rate)/frameSize_N

            %{

                Example Fourier Transform of Audio Signal (not to scale!)

                        Magnitude
                          |                         x
                          |              x          |
                          |              |          |
                          |     x        |     x    |
                          |     |    x   |     |    |
                        _|_____|____|____|_____|____|___ Index i
                          |                          |
                        i=0                         i_max=frameSize_N=2048
            %}

            peakFrequencies_f(k)=((peakLocations(k)*fs)/frameSize_N);
        end

        %Matrix containing the note (aka chroma) evaluated at each peak detected
```

```
        if numberOfPeaks
            note_c = zeros(numberOfPeaks,1);
        else
            %Sometimes there will be no peaks due to there being no sound in
            %segments of the song
            note_c = zeros(1,1);
        end


        %computing the octave for each peak we have
        octave = zeros(numberOfPeaks,1);

        %Matrix holding weighted sum of semitones
        weight = zeros(numberOfPeaks,notes_in_an_octave);

        %Computing the semitone, note, octave, and weight as required and described
        %in eqns 15 and 16 for each peak frequency at
        for peakNumber=1:1:numberOfPeaks
            try
                %Computing semitone (sm) for each peak

                %peakFrequencies_f(peakNumber) gives frequency at peak
                %number, k
                semitone_sm(peakNumber)=
round(12*log2(peakFrequencies_f(peakNumber)./f0));
            catch
                warning('Could not compute semitone. Self-destruct sequence
initiated.')
            end


            try
                note_c(peakNumber) = mod(semitone_sm(peakNumber),12);
            catch
                warning('Could not compute note. Self-destruct sequence initiated.')
            end

            %Need to perform integer division to obtain the octave
            octave(peakNumber) = floor(semitone_sm(peakNumber)/12);



            try
                %Computing the weight as defined by equation 19

                r = 12*log2(peakFrequencies_f(peakNumber)./f0) -
semitone_sm(peakNumber);

                if (r>-1) && (r<1)
                    weight(peakNumber,note_c(peakNumber)+1) = cos(pi*r/2)^2;
                else
                    weight(peakNumber,note_c(peakNumber)+1) = 0;
                end
            catch
                warning('Could not compute weight. Self-destruct sequence initiated.')
            end

        end




        for note=1:1:12
            %Implementing the Pitch Class Profile (Equation 21)
            for peakNumber=1:numberOfPeaks
               try
```

```
                    PCP(note) = PCP(note) +
weight(peakNumber,note)*(peakMagnitudes(peakNumber,1)^2);
                catch
                    warning('Could not compute the Pitch Class Profile. Self-destruct
sequence initiated.')
                end
            end

            try
            %For efficiency's sake, we can just explicitly carry out the summation...

                if numberOfPeaks
                    NPCP(note,frameNumber) =  PCP(note)/...
                                                        (PCP(1)+...
                                                         PCP(2)+...
                                                         PCP(3)+...
                                                         PCP(4)+...
                                                         PCP(5)+...
                                                         PCP(6)+...
                                                         PCP(7)+...
                                                         PCP(8)+...
                                                         PCP(9)+...
                                                         PCP(10)+...
                                                         PCP(11)+...
                                                         PCP(12));
                else
                    %If no peaks detected, nothing is in this frame
                    NPCP(note,frameNumber) = 0;
                end
            catch
                        warning('Could not compute the Normalized Pitch Class Profile.
Self-destruct sequence initiated.')
            end
        end

    %   syms l;
    %
PCP(c(j)+1)=symsum(m(l,c(j)+1)*abs(Y(peakFrequencies_f(locs(l))))^2,l,1,numberOfPeaks)
;


    end


   %Plot result
    NPCP_flipped = flipud(NPCP);
    fig = imagesc(10*log10(NPCP_flipped))
    %fig = imagesc(10*log10(NPCP))
    %plot(weight)

    %Labels
    title(song_path);
    ylabel('Note');
    set(gca,'YTick',1:1:12)
    set(gca,'YTickLabel',{'G#','G','F#','F','E','D#','D','C#','C','B','A#','A'})
    xlabel('Frame Number');
    colorbar;
    colormap('jet');

    %Conditioning file name to properly save figure
    song_path(1:7)='';
    song_path(length(song_path)-3:length(song_path))='';

    fileLocation = [song_path '.png'];
    saveas(fig,fileLocation)
```

```
end
```

script.m

```
close all
songList = {
            '\audio\track201-classical.wav', ...
            '\audio\track204-classical.wav', ...
            '\audio\track370-electronic.wav', ...
            '\audio\track396-electronic.wav', ...
            '\audio\track437-jazz.wav', ...
            '\audio\track439-jazz.wav', ...
            '\audio\track463-metal.wav', ...
            '\audio\track492-metal.wav', ...
            '\audio\track547-rock.wav', ...
            '\audio\track550-rock.wav', ...
            '\audio\track707-world.wav', ...
            '\audio\track729-world.wav', ...
            '\audio\sample1.wav'
            };
for songChoice = 1:12
    pathToSong = char(songList(songChoice));
    [pcp, npcp] =NormPitchClassProfile(pathToSong,0,1);
end
```