Lab 1

Clint Olsen

ECEN 4532: DSP Lab

February 6, 2018

University of Colorado
Boulder

Contents

Introduction

This lab introduces many of the important aspects of analyzing the characteristics of signals, and in particular, audio signals. Many mathematical tools can be used to describe the relationship between what we hear in music and what is physically happening in the time and frequency domain. These characteristics can be built up to begin using computer algorithms to analyze and categorize music based on its frequency response, loudness, flatness, and many other spectral characteristics. This lab focuses on using MATLAB to compute these various qualities in order to begin gaining an understanding the relationship between computation and digital signal processing.

Assignment 1

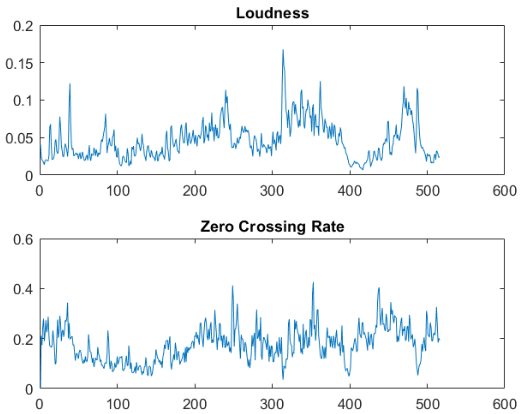
Using the example of the appropriate sampling frequency of CDs for humans at 44.1 kHz, this means a CD can perfectly reconstruct a signal that has a Nyquist frequency up to 22.05 kHz, which happens to be the top end of human frequency response. In order to cover the full range of frequencies for a dolphin, the signal should be sampled at a frequency greater than 240 kHz to perfectly reconstruct the range of frequencies perceivable by dolphins.

Assignment 2

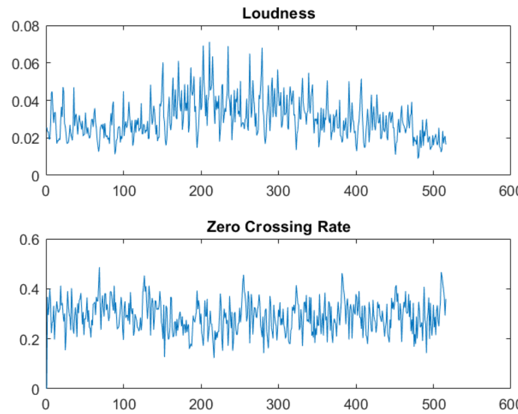
For this programming task, MATLAB will be used to extract T seconds from an audio $.wav$ file, starting at the midpoint of the song. The file can be read in using the $audioread()$ function and simple vector manipulation can be used as a parameter to this function to only extract the desired amount of time. In order to perform this calculation, the total number of samples must be known as well as the sampling frequency. These can be determined using the $audioinfo()$ function. See appendix for Assignment 2 for the MATLAB code.

Assignment 3

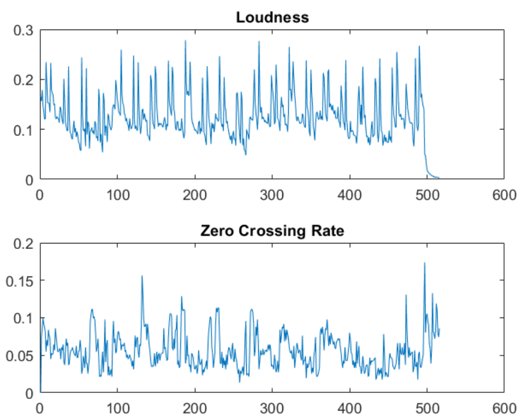
Following the given equations for both the Loudness and the ZCR equations, MATLAB was used to implement and plot their effect on given audio signals. Loudness, which is the standard deviation of the signal over a given frame, was implemented using a frame size of $N = 512$, although the provided function allows for input of any frame size. The sum is performed over the total number of frames in the signal, which is the number of samples in the entire signal divided by the number of frames. A separate function was also used to calculate the expectation, or mean, which is also provided. The provided function also calculates the Zero Crossing Rate, which is a measure of how many times the signal crosses the x-axis. This was implemented in a similar manner, summing over each frame and sample within each frame. Below are the plots of the Loudness and ZCR for each of the audio tracks (one from each genre) and the MATLAB code can be found in the Assignment 3 appendix:



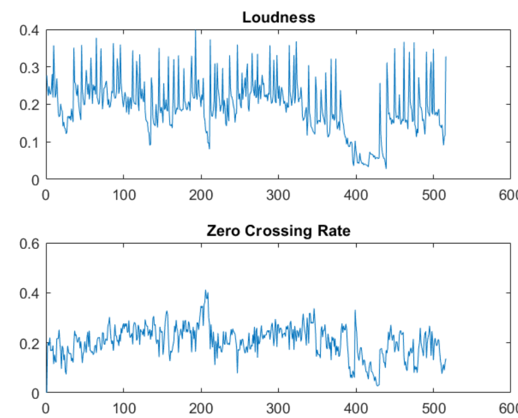
(a) Classical 201



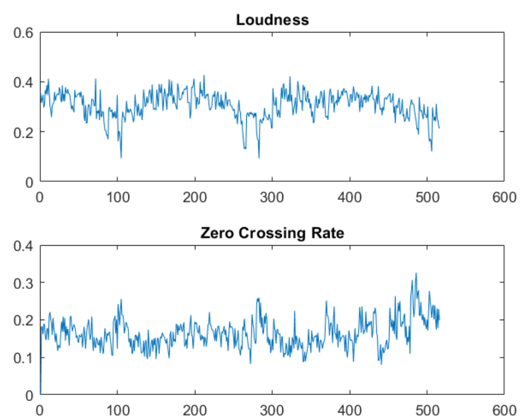
(b) Electronic 370



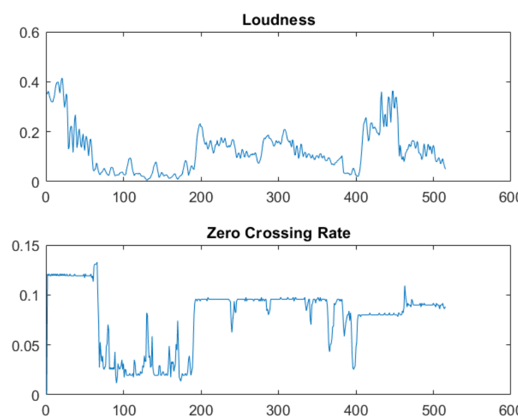
(c) Jazz 437



(d) Metal 463



(e) Rock 547



(f) World 707

Assignment 4

In looking at various plots of the loudness and ZCR of the tracks provided by the lab, there are a few characteristics that can be seen that would help classify the music into broad categories. Starting with the loudness, there are certain genres that are on average "louder" and less dynamic than others, such as comparing a piano concerto to heavy metal. The classical tracks show a loudness pattern that varies greatly in comparison to the metal tracks, which indicates there are more fluctuations in volume throughout the piece. The metal track has a constant and high amplitude loudness throughout the song, which matches the actual perception when compared to the classical pieces. This would be useful in narrowing down genres, if the simple case was "loud music" and "quieter music" or "dynamic music" and "static music".

The ZCR also allows for a different type of classification that is more driven by frequency. Using the classical pieces as an example again, it can be seen that there are high frequency patterns of zero crossing throughout the piece which indicates higher frequency components. Comparing this with the world track $track707 - world$ which is a lower pitched flute, there are spikes of zero crossing at the beginning and end likely due to noise, and in the middle, less frequent zero crossing due to the lower frequencies. This would be useful in classifying perhaps classical from modern genres due to the appeal of lower frequencies in modern music, which could be detected from ZCR.

Assignment 5

To compute the Discrete Time Fourier Transform of the function

$$x[n] = cos(\omega_0 n)$$

We start with the definition of the DTFT to solve:

$$\sum_{n=-\infty}^{\infty} x[n]e^{-j\omega n}$$
$$\sum_{n=-\infty}^{\infty} cos(\omega_0 n)e^{-j\omega n}$$
$$\sum_{n=-\infty}^{\infty} \frac{e^{j\omega_0 n}+e^{-j\omega_0 n}}{2}e^{-j\omega n}$$
$$\sum_{n=-\infty}^{\infty} \frac{e^{jn(\omega_0-\omega)}+e^{-jn(\omega_0+\omega)}}{2}$$
$$\sum_{n=-\infty}^{\infty} \frac{e^{j\omega_0 n}e^{-j\omega n}}{2} + \sum_{n=-\infty}^{\infty} \frac{e^{-j\omega_0 n}e^{-j\omega n}}{2}$$

From here, we can use the frequency shifting property of the DTFT to see that this is just the frequency shifted transform of $x[n] = 1$, shifted by ω_0 in the frequency domain:

$$\frac{1}{2}2\pi \sum_{k=-\infty}^{\infty} \delta(\omega - \omega_0 - 2\pi k) + \delta(\omega + \omega_0 - 2\pi k)$$

$$\boxed{\pi \sum_{k=-\infty}^{\infty} \delta(\omega - \omega_0 - 2\pi k) + \delta(\omega + \omega_0 - 2\pi k)}$$

The final result is an infinitely periodic sequence of delta functions at negative and positive values of the frequency ω_0, 2π periodic.

Assignment 6

When windowing a function, the original function is multiplied by a particular framing window:

$$y[n] = x[n]w[n + \tfrac{N}{2}]$$

To find the Fourier transform, it can be recognized that this is multiplication in the time domain with a time shift, which results in multiplication by an exponential and convolution in the frequency domain:

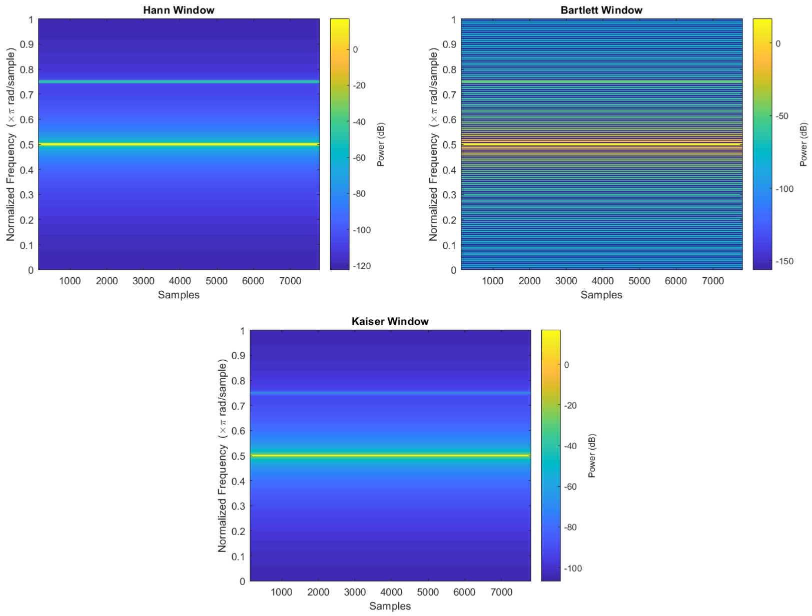
$$y[n] = x[n]w[n] \rightarrow Y(e^{j\omega}) = X(e^{j\omega}) * W(e^{j\omega})$$
$$y[n] = x[n]w[n + \tfrac{N}{2}] \rightarrow \boxed{Y(e^{j\omega}) = e^{j\omega(\frac{N}{2})}X(e^{j\omega}) * W(e^{j\omega})}$$

Assignment 7

One method of frequency analysis in the frequency domain is through the use of window functions. In order to test the different behavior of the Hann, Bartlett, and Kaiser windowing functions, the Fourier Transform was taken on the sum of two pure sinusoidal functions:

$$x[n] = cos(4\pi n) + .01cos(6\pi n)$$

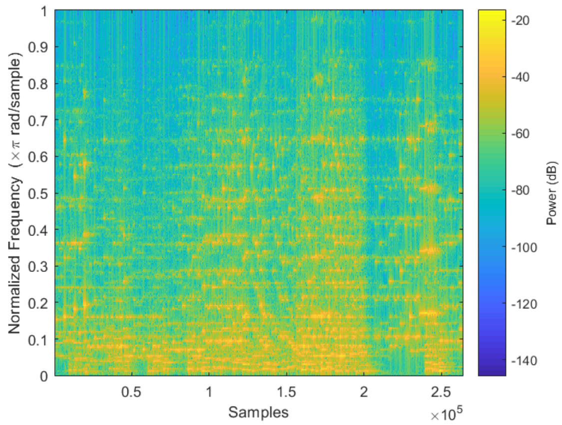
with a sampling frequency of 8 Hz. This means that we can reconstruct signals up to 4 Hz. The two frequencies of these sinusoids are 2 Hz and 3 Hz respectively:



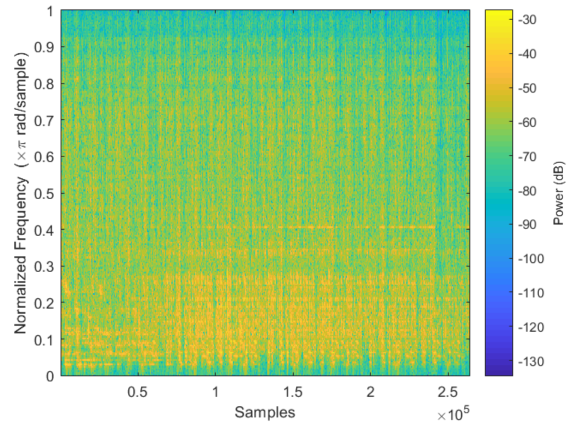
In analyzing these windows, it can be seen that the Bartlett window clearly has the widest stopband, due to the fact that the frequencies extending out from the frequencies of the sinusoid have slowly diminishing power, so more frequencies are being captured. The Kaiser window has the narrowest stop band, because it distinctly shows the 2 and 3 Hz frequencies and thin, high power lines, so this window is likely meant for capturing many specific frequencies. Finally, the Hann window has similar resemblance to the Kaiser, but both frequencies have thicker lines, indicating a wider stopband. However, the lower amplitude sinusoid shows up more clearly and with higher power. This indicates that this window is meant to detect lower amplitude frequencies.

Assignment 8

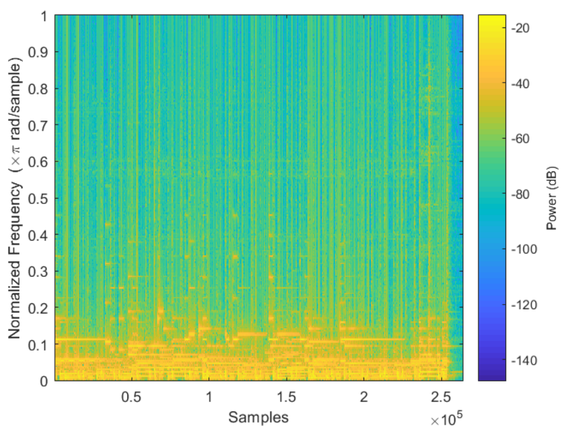
Another form of analysis that can be performed on the audio signals is to use the $spectrogram()$ function used above to analyze the frequency characteristics and magnitudes of various genres of music. The $spectrogram()$ function can be used to compute the FFT of a given signal, window it with a given window function, and provide overlapping of frames if necessary. The spectrograms for each genre are shown below using a Kaiser window:



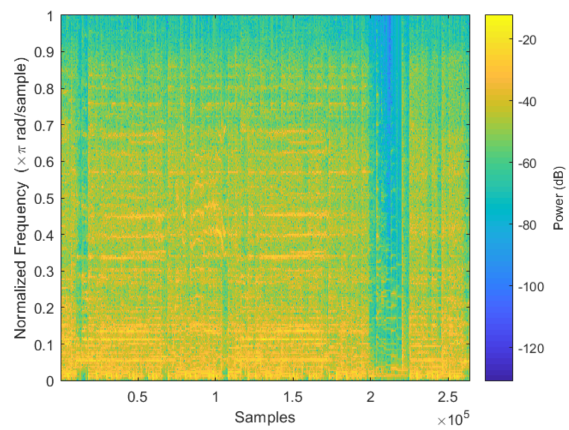
(a) Classical 201



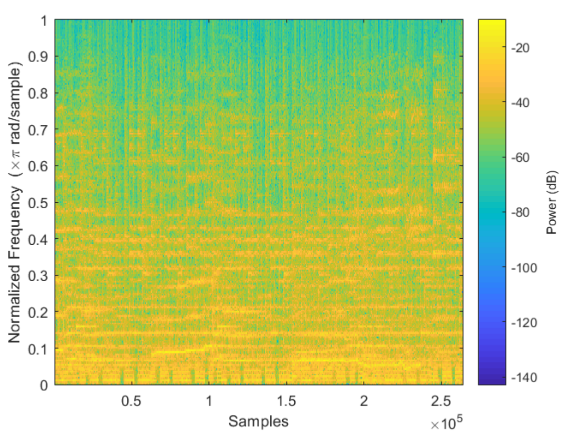
(b) Electronic 370



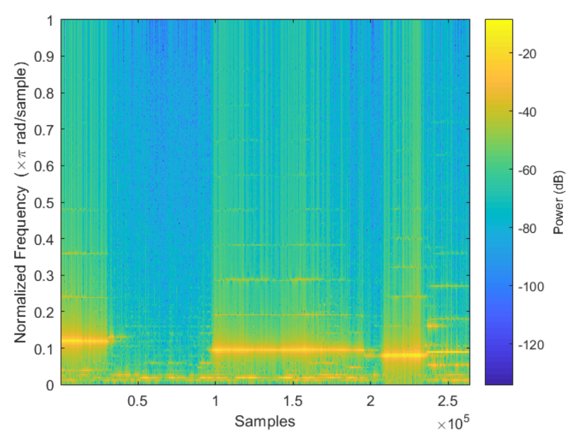
(c) Jazz 437



(d) Metal 463



(e) Rock 547



(f) World 707

Upon looking at the spectrograms, the aural qualities of the music can be seen visually within these plots. For examples, the rock and metal tracks have large power over almost all of the track and frequencies, indicating the magnitude of these signals overall are large.

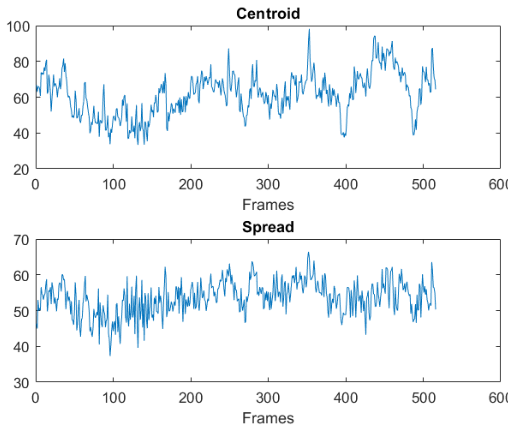
Comparing these tracks with the classical and jazz tracks, it is apparent that there are lot less high power samples and frequencies, which show the dynamic nature of these genres in comparison to more monotonous genres like rock and metal. The MATLAB code for generating these plots can be seen in the Assignment 8 Appendix.

Assignment 9: Centroid and Spread

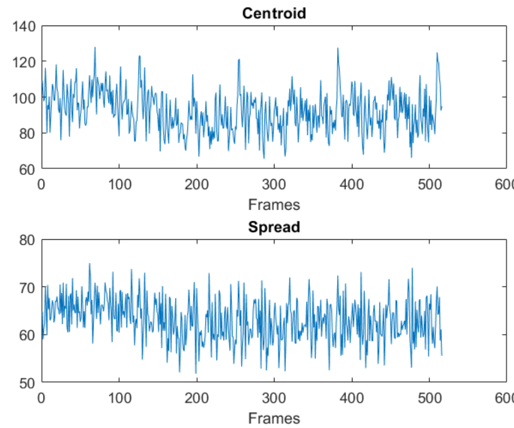
This portion of the lab focuses on the spectral characteristics of the centroid (expectation value of a frequency) and the spread of a signal. The following plots were generated for one track of each genre using non-overlapping frames of size $N = 512$ and the given equations 7,8, and 9. The plots can be seen on the next page. See the Assignment 9: Centroid and Spread Appendix for MATLAB code.

Assignment 10: Centroid and Spread Analysis

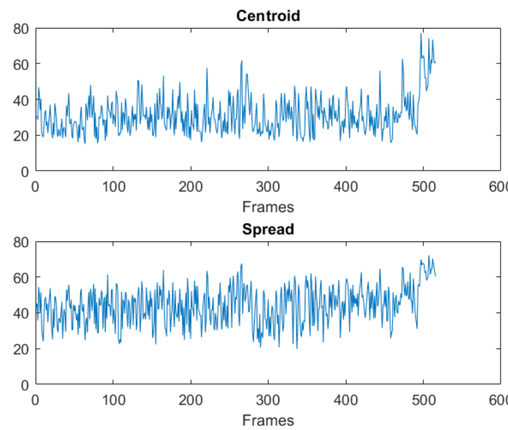
The centroid of the signal is meant to help indicate the brightness of the signal, which can generally be attributed to higher recording quality, higher frequencies, and major harmonic structure in the music. One example of where this quality is exemplified is in between the World track and the electronic track. The World track is composed of low monotonous sounds in the form of a flute, where the classical piece is a violin concerto that consists of any high and "bright" sounds, which is shown by the much larger Centroid amplitude in the plots. The spread can help distinguish between noise and tone like differences, where greater variations would indicate more tonal signals. An example of where this characteristic is exemplified is in the metal track, which is low recording quality and has lots of audible noise. The spread plot shows a consistent, high amplitude spread with little oscillations, which visualize the noise.



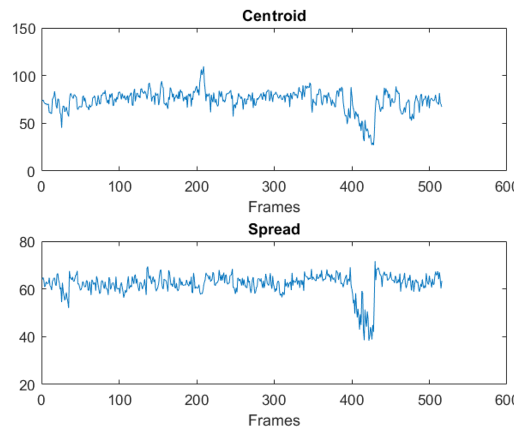
(a) Classical 201



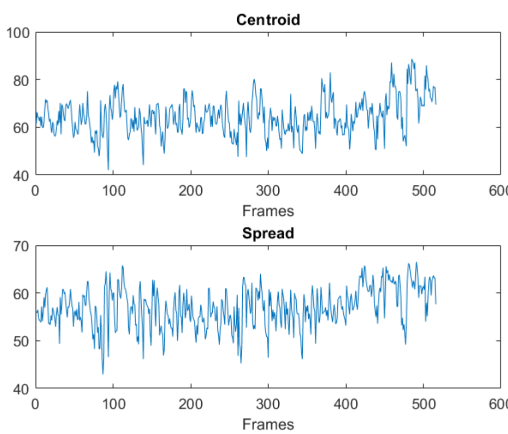
(b) Electronic 370



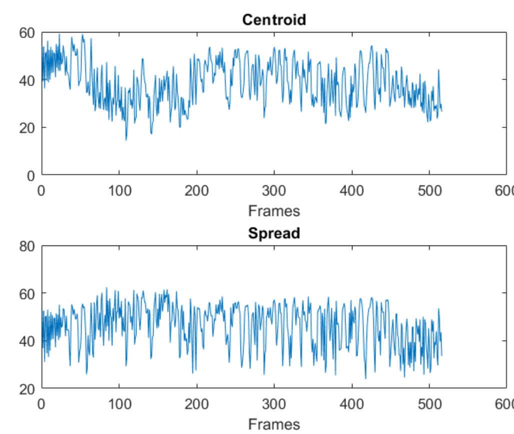
(c) Jazz 437



(d) Metal 463



(e) Rock 547



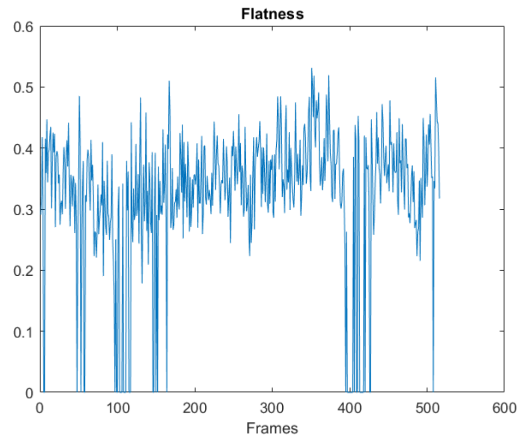
(f) World 707

Assignment 9: Spectral Flatness

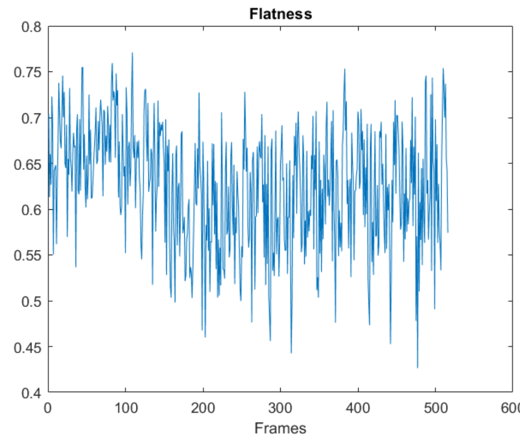
This spectral characteristic is useful in determining the overall noisiness of a audio signal. It is calculated in a similar manner in terms of the Fourier Transform, window, and framing as the Centroid, but instead implementing equation 10. See the Assignment 9: Spectral Flatness Appendix for MATLAB code.

Assignment 10: Flatness Analysis

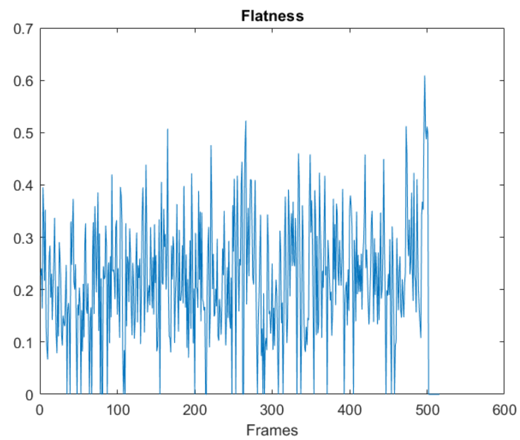
One area where Flatness is useful is determining the quality of the recordings that the analysis is being performed on. The Electronic and Metal tracks are both tracks that have older sounding recordings and have audible crunchiness to them. On the other hand, the jazz and classical pieces are clearer sounding and in the plots, are resembled with a lower amplitude flatness. This could perhaps be useful in categorizing older recordings from newer ones, which may be the first step in detecting the genre of a song based on the spectral components.



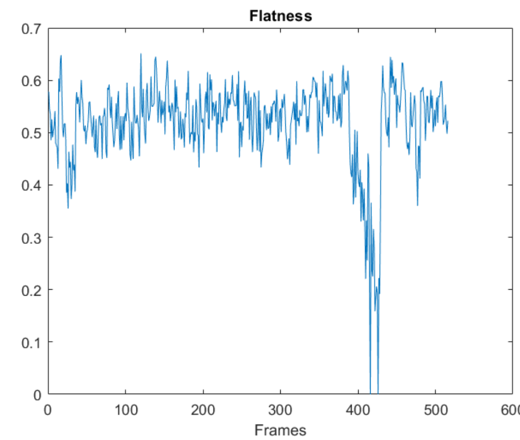
(a) Classical 201



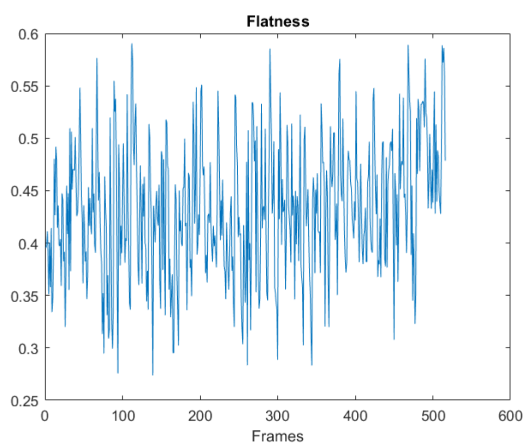
(b) Electronic 370



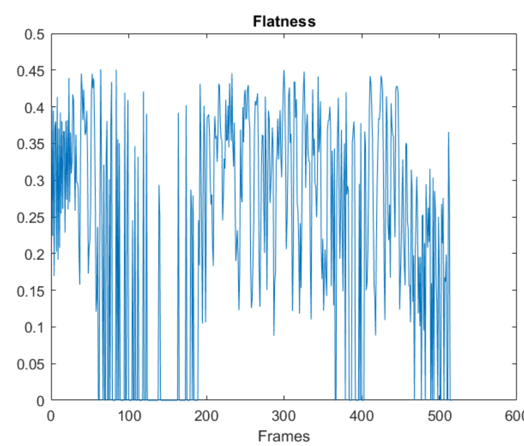
(c) Jazz 437



(d) Metal 463



(e) Rock 547



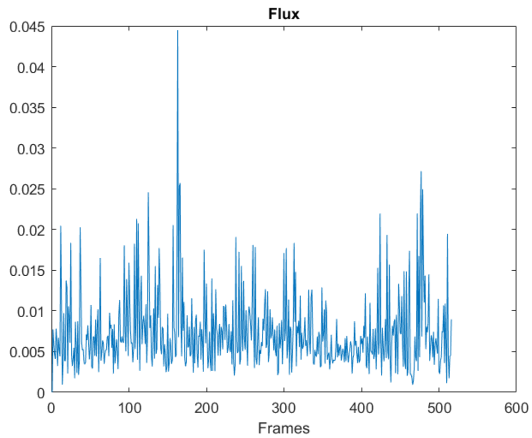
(f) World 707

Assignment 9: Spectral Flux

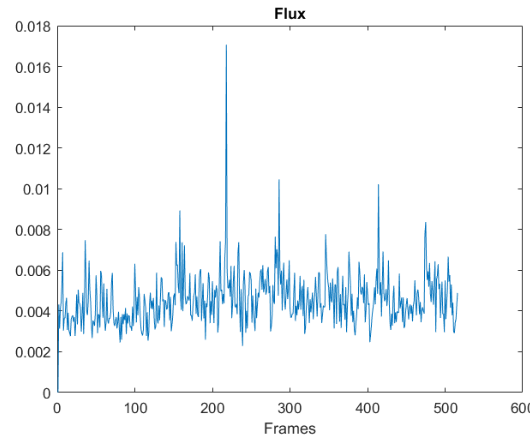
The final spectral descriptor is spectral flux which is a indication of how much a signal is changing from frame to frame. This phenomenon is described by equation 11 and is shown below for the tracks. See the Assignment 9: Spectral Flux Appendix for MATLAB code.

Assignment 10: Flux Analysis

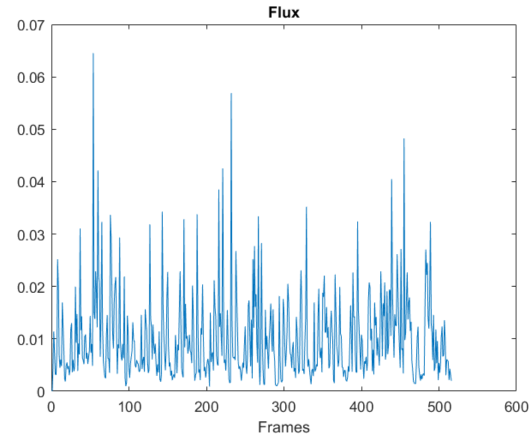
The mathematical definition of flux, or the change from one frame to the other, lends itself well to characterizing the dynamic contrast of particular genres of music. This means how much the volume and intensity of the sound changes throughout the piece. Many genres such as rock and metal will likely have little change from frame to frame due to the consistency that is commonly present in these genres, but a classical or world piece may have lots of contrast between frames. In looking at the plots, it can be seen that the World track has some of the highest fluctuations between frames, indicating it is changing volume and frequency often. Comparing this track to the Rock track, the Rock song has a much smaller amplitude in its flux, showing it is more consistent throughout the song which can be verified by listening to the track.



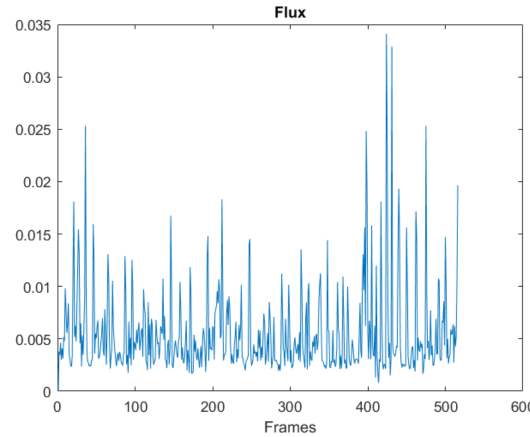
(a) Classical 201



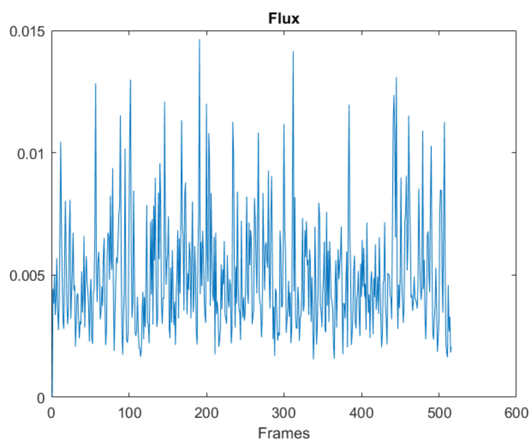
(b) Electronic 370



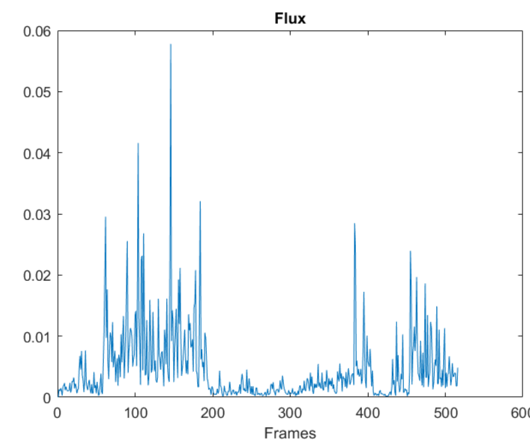
(c) Jazz 437



(d) Metal 463



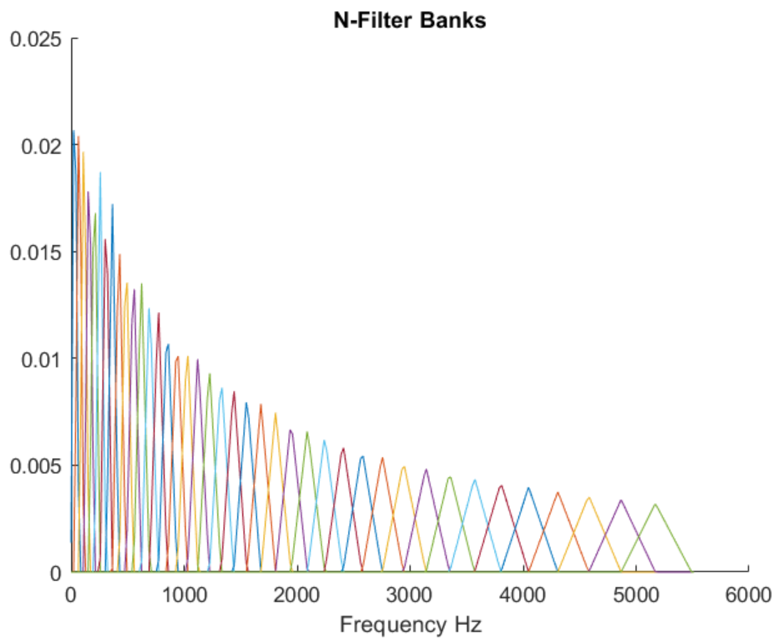
(e) Rock 547



(f) World 707

Assignment 11

This section of the lab focuses on the implementation of a cochlear filter bank of size 40. This is a very interesting topic because this can be directly applied to the development of frequency responses for cochlear implants, which generally have filter banks of around 22 filters of a very similar overlap and shape to the filter used in this section. The filters are based off of equally distanced peaks at frequencies over the range of the frequencies limited by the low end of the human hearing range at 20 Hz, up to half of the sampling frequency of a given track. The implementation of the filter bank is shown graphically below and the MATLAB code can be found the in Assignment 11 Appendix.



One interesting thing to note about the amplitudes of the response is that it is not uniformly decaying in all cases. Towards the lower end of the frequency spectrum, there are several filters that have greater response that the previous filters. This is due to the fact that these filters got placed more directly over the calculated mel frequencies that correspond to the their peak response. Others may be offset from these frequencies, resulting in the lower response.

Assignment 12

In order to complete the filter bank analysis, the MFCC coefficients need to be calculated for any given Fourier Transform frame. The implementation of this in MATLAB is shown in the Assignment 12 Appendix.

Code Appendix

Assignment 2

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Clint Olsen
% ECEN 4532: DSP Lab
% Lab 1: Assignment 2
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function signal = extractTSeconds(T, filename)
    %Store the audio file information
    info = audioinfo(filename);

    %Read in the audio file starting at the halfway point of track
    %(info.TotalSamples/2) and extract T seconds past this point of
    %samples
    [signal,fs] = audioread(filename,[ceil((info.TotalSamples)/2) ...
        ceil((info.TotalSamples)/2+(T*info.SampleRate))]);
    %Create and audioplayer handle to play back the extracted track
    track = audioplayer(signal,fs);

    %Play the track while blocking execution
    playblocking(track)
end
```

Assignment 3

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Clint Olsen
% ECEN 4532: DSP Lab
% Lab 1: Assignment 3 Loudness and ZCR
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function loudnessZCR(N,filename)
    %Read in the audio file
    track = extractTSeconds(24,filename);

    %%%%%%%%%%%%%% Loudness %%%%%%%%%%%%%%%%%
    loudness = zeros(1,floor(length(track)/N));

    %Sum over each frame
```

```
    for n=1:floor(length(track)/N)
        for m=1+((N)*(n-1)):1+((N)*(n-1)) + N-1 %Inner sum
            loudness(n) = loudness(n) + (track(m) - mean(track,N,n))^2;
        end
        loudness(n) = sqrt((1/(N-1))*loudness(n));
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    %%%%%%%%%% Zero Crossing Rate %%%%%%%%%%
    ZCR = zeros(1,floor(length(track)/N));

    %Sum over each frame
    for n=2:floor(length(track)/N)
        for m=1+((N)*(n-1)):1+((N)*(n-1))+ N-1 %inner sum
            ZCR(n) = ZCR(n) + abs(sign(track(m))-sign(track(m-1)));
        end
        ZCR(n) = (1/(2*N))*ZCR(n);
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    %%%%%%%%%%%%%%% Plots %%%%%%%%%%%%%%%%%
    subplot(2,1,1)
    plot(loudness);
    title('Loudness');
    subplot(2,1,2);
    plot(ZCR)
    title('Zero Crossing Rate')

end
```

Assignment 3: Mean

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Clint Olsen
% ECEN 4532: DSP Lab
% Lab 1: Assignment 3 Mean
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function E = mean(x,N,n)

    %Variable to hold sum
    sum = 0;
    %Calculate the summation over N for a given frame
```

```
        for m=n+((N-1)*(n-1)):N-1
            sum = sum + x(m);
        end
        %Return the mean value
        E = (1/N)*sum;
end
```

Assignment 7

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Clint Olsen
% ECEN 4532: DSP Lab
% Lab 1: Assignment 7 Window Types
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function testWindows()
    %Create a test signal; a pure sinusoid
    N = 512;
    Fs = 8;
    f = 2;
    T = 1/Fs;
    t = 0:T:1000;

    %Frequencies: 2 Hz and 3 Hz
    x = 10*cos(2*pi*f*t) + 0.01*cos(2*pi*1.5*f*t);

    %Plot the spectrograms of each window
    figure(1)
    spectrogram(x, hann(N),N/2, 'power', 'yaxis')
    title('Hann Window')
    figure(2)
    spectrogram(x, bartlett(N),N/2, 'power', 'yaxis')
    title('Bartlett Window')
    figure(3)
    spectrogram(x, kaiser(N,0.5),N/2, 'power', 'yaxis')
    title('Kaiser Window')
end
```

Assignment 8

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Clint Olsen
% ECEN 4532: DSP Lab
% Lab 1: Assignment 8 Audio Spectrograms
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```
function audioSpectrogram(N, filename)

    %Extract 24 seconds from the original audio file
    track = extractTSeconds(24,filename);

    %Fram minus overlap of N/2
    K = N/2 + 1;

    %Plot the specrogram with Kaiser window of size N
    spectrogram(track, kaiser(N,0.5),N/2, 'power', 'yaxis')

end
```

Assignment 9: Centroid and Spread

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Clint Olsen
% ECEN 4532: DSP Lab
% Lab 1: Assignment 9: Spectral Centroid and Spread
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function centroidSpread(N, filename)
    %Extract 24 seconds from the audio file
    track = extractTSeconds(24,filename);

    %Half of the Frame Size
    K = N/2 + 1;

    %Calculation Variables
    centroid = zeros(1,floor(length(track)/N));
    spread = zeros(1,floor(length(track)/N));
    Pn = zeros(1,K);
    kSum = zeros(1,floor(length(track)/N));

    % sum over frames in signal (x2 with overlap)
    for n=1:floor(length(track)/N)

        %Extract N samples based on frame number
        %xn = track(1+((N-1)*(n-1)):1+((N-1)*(n-1)) + N-1);
        xn = track(1+((N)*(n-1)):1+((N)*(n-1)) + N-1);

        %Perform the fft and extract the desired components up to K
        Y = fft((kaiser(N,0.5)).*xn);
        Xn = Y(1:K);
```

```
        %Center of Mass (Centroid) Sum
        for l=1:K %Probability sum
            kSum(n) = kSum(n) + abs(Xn(l));
        end
        for i=1:K
            Pn(i) = abs(Xn(i))/kSum(n);
            centroid(n) = centroid(n) + (i*Pn(i));
        end

        %Spectral Spread
        for j=1:K
            spread(n) = spread(n) + (((j-centroid(n))^2)*Pn(j));
        end
        spread(n) = sqrt(spread(n));
    end

    %Plots
    subplot(2,1,1);
    plot(centroid);
    title('Centroid')
    xlabel('Frames')
    subplot(2,1,2);
    plot(spread);
    title('Spread')
    xlabel('Frames')
end
```

Assignment 9: Spectral Flatness

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Clint Olsen
% ECEN 4532: DSP Lab
% Lab 1: Assignment 9: Spectral Flatness
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function flatness(N, filename)
    %Extract 24 seconds from the audio file
    track = extractTSeconds(24,filename);

    K = N/2 + 1;

    %Calculation Variables
    flatness = zeros(1,floor(length(track)/N));
    kProd = ones(1,floor(length(track)/N));
```

```
    kSum = zeros(1,floor(length(track)/N));

     % sum over frames in signal
    for n=1:floor(length(track)/N)

        %Extract N samples based on frame number
        xn = track(1+((N)*(n-1)):1+((N)*(n-1)) + (N-1));

        %Perform the fft and extract the desired components up to
            K
        Y = fft(kaiser(N,0.5).*xn);
        Xn = Y(1:K);

        %Calculate the Product and Sum series
        for k=1:K
            kProd(n) = kProd(n)*abs(Xn(k));
            kSum(n) = kSum(n) + abs(Xn(k));
        end

        %Flatness component of 1 frame
        kProd(n) = (kProd(n))^(1/K);
        kSum(n) = (1/K)*kSum(n);
        flatness(n) = kProd(n)/kSum(n);
    end

    %Plot the flatness function
    plot(flatness);
    title('Flatness');
    xlabel('Frames');
end
```

Assignment 9: Spectral Flux

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Clint Olsen
% ECEN 4532: DSP Lab
% Lab 1: Assignment 9: Spectral Flux
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function flux(N, filename)
    %Extract 24 seconds from the audio file
    track = extractTSeconds(24,filename);

    K = N/2 + 1;

```

```
    %Calculation Variables
    flux = zeros(1,floor(length(track)/N));
    kSum = zeros(1,floor(length(track)/N));
    kSumPrev = zeros(1,floor(length(track)/N));

    % sum over frames in signal
    for n=2:floor(length(track)/N)

        %Extract N samples based on frame number
        xn = track(1+((N)*(n-1)):1+((N)*(n-1)) + N-1);
        xnPrev = track(1+((N)*((n-1)-1)):1+((N)*((n-1)-1)) + N-1);

        %Perform the fft and extract the desired components up to K
        Y = fft(kaiser(N).*xn);
        Xn = Y(1:K);
        YPrev = fft(kaiser(N).*xnPrev);
        XnPrev = YPrev(1:K);

        for l=1:K %Probability sum
            kSum(n) = kSum(n) + abs(Xn(l));
            kSumPrev(n) = kSumPrev(n) + abs(XnPrev(l));
        end
        %Calculate the Flux
        for k=1:K
            flux(n) = flux(n) + ((abs(Xn(k))/kSum(n))-(abs(XnPrev(k))/kSumPrev(n)))^2;
        end
    end

    %Plot the flux function
    plot(flux);
    title('Flux')
    xlabel('Frames')
end
```

Assignment 11

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Clint Olsen
% ECEN 4532: DSP Lab
% Lab 1: Assignment 11: Cochlear Filterbank
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function fbank = cochlearFilterbank(N, filename)
```

```
info = audioinfo(filename);
fs = info.SampleRate;
K = N/2 + 1;

nbanks = 40; %% Number of filters

% linear frequencies from mel_min to mel_max, including the endpoints
linFrq  = 20:fs/2;

% mel values corresponding to the above frequencies
melFrq = log ( 1 + linFrq/700) *1127.01048;

% equispaced mel values including the min and max endpoints. Using
% nbanks+2 ensures that we have nbanks points between 20 and fs/2
melIdx = linspace(20,max(melFrq),nbanks+2);

% This array will end up holding the frequencies corresponding to the
% uniformly spaced mel values
melIdx2Frq = zeros(nbanks+2, 1);

% To populate the array, we find the nearest value using a brute force approach
% (we could solve analytically if we wanted to, but this teaches a useful technique)
for i=1:nbanks+2
    [val indx] = min(abs(melFrq - melIdx(i)));
    melIdx2Frq(i) = linFrq(indx);
end

%To create Nb x K Matrix, we need to step by K in the frequency array
kFreq = linspace(20,fs/2,K);

%Frequency Bank matrix to hold the values of each filter (nbanks x K)
fbank = zeros(nbanks,K);

%Compute the loop for each filter Hp(f)(Nb x K matrix)
for n=2:nbanks+1
    for k=1:K
```

```
            %Case 1: Frequency between current filter and previous filter
            if((k*(fs/N)) >= melIdx2Frq(n-1) && (k*(fs/N)) < melIdx2Frq(n))
                fbank((n-1),k) = ((2/(melIdx2Frq(n+1)-melIdx2Frq(n-1)))* ...
                    (((k*(fs/N))-melIdx2Frq(n-1))/(melIdx2Frq(n)-melIdx2Frq(n-1))));
            %Case 2: Frequency between current filter and next filter
            elseif((k*(fs/N)) >= melIdx2Frq(n) && (k*(fs/N)) < melIdx2Frq(n+1))
                fbank((n-1),k) = ((2/(melIdx2Frq(n+1)-melIdx2Frq(n-1)))* ...
                    ((melIdx2Frq(n+1)-(k*(fs/N)))/(melIdx2Frq(n+1)-melIdx2Frq(n))));
            %Case 3: Out of range
            else
                fbank((n-1),k) = 0;
            end
        end
    end

    % Plot the filter bank over the range of frequencies up to fs/2
    hold on;
    for p=1:nbanks
        plot(linspace(0,fs/2,K),fbank(p,:));
    end
    title('N-Filter Banks')
    xlabel('Frequency Hz')
    hold off;

end
```

Assignment 12

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Clint Olsen
% ECEN 4532: DSP Lab
% Lab 1: Assignment 12: MFCC Coefficients
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function mfcc = mfccComp(N, Xn, Hp, Nb)
    %Define K as half of the frame size N
    K = N/2 + 1;
```

```
    %Define the vector to hold the coefficients for each filter
    mfcc = zeros(1,Nb);

    %Compute the coefficents for each filter in the bank.
    for p=1:Nb
        for k=1:K
            mfcc(p) = mfcc(p)+(abs(Hp(p,k)*Xn(k)))^2;
        end
    end

end
```