Ryan Sigar
ECEN 4632: Introduction to Digital Filtering
Final Project Part 1

Solutions to Question 1:

Below is code that plays T seconds of an audio file from the beginning or middle of the file. The appendix has more information of how the slice_audio function - used to calculate the start and end frames of the current song - was implemented.

```
close all

%Put all song names into a cell array
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'
            };

%Question 1:
%Write a MATLAB function that extract T seconds of
%music from a given track. You will use the MATLAB function audioread to
%read a track and the function play to listen to the track.


%take_audio input parameters: audio file, amount of time to extract, middle
%or not middle switch

%Choose your song according to the index number given above. 1 corresponds to
track201-classical.wav
%and 13 corresponds to sample1.wav.
songChoice = 1;

pathToSong = char(songList(songChoice));

amountOfTime = 3; %seconds
takeFromMiddle = 1; %software switch to enable taking audio from middle

[song_object, audio_data, fs ,start, stop] = slice_audio(pathToSong,...
                                             amountOfTime,takeFromMiddle);

%sound(audio_data,fs) %don't use this, you won't be able to pause the music

play(song_object,[start,stop])
```

Ryan Sigar
ECEN 4632: Introduction to Digital Filtering
Final Project Part 1

Solutions to Question 2:

The following is code used to generate a N_B x 1 matrix of the mfcc coefficients and plot it. It computes the mfcc coefficients of one frame of song data given to the function. An easy way of computing values for this matrix is to pull the first 512 samples from the beginning of a song. However, in many cases, the first few values of a song can be empty either simply due to the delay in start time, or an intended buildup that an artist uses as an introduction to the song. Taking the fourier transform of an empty frame will result in no meaningful data. So one way to prevent this from occurring, we can scan the song for the first index containing a non-zero sample, and collect the first 512 frames from there.

```
%Question 2:
%Implement the computation of the mfcc coeffcients, as defined in
%(7). You simply need to add your code in the MATLAB code in the previous
pages.

%Note: these are the values extracted from part 1: fs, song_object,
%audio_data, fs ,start frame, and stop frame

%samples per frame
N = 512;

%want this to be the number of samples per frame (i think)
fftSize = N;

%Create a window
% w = kaiser(N,beta);
w = hann(fftSize);

%Compute mfcc coefficient for one frame of the song that pathToSong points
%to. This should produce a boring graph of the mfcc coefficients of one
frame.
y=mfcc(audio_data,fs,fftSize,w,pathToSong);
```

Solution to Question 3:

Below is code appended to the end of the original code give at the start of the project. It is responsible for computing the matrix of mfcc coefficients for 517 frames of a given song. It is asked that we collect 24 seconds of data for processing. Since the sample rate of all of the songs given in the first portion of this project is 11,025Hz, and the amount of samples in a frame to be processed is 512 samples, the number of frames to process in 24 seconds of a song with the given sampling frequency will be approximately 517.

This code generates the plots that allow us to visualize the mfcc coefficients computed for each of the 517 frames to be processed and the 40 filterbanks that the frequency components of a given frame will be filtered against. The generated plots will be shown and discussed in following data and discussion section.

```
%_____________________________________________________________
%
% PART 2 : processing of the audio vector In the Fourier domain.
%_____________________________________________________________
%
% YOU NEED TO ADD YOUR CODE HERE

%need to take a 512 sized chunk from wav (audio data from user)

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

%take 512 samples starting from the first non-zero point (add 512 samples to
this )
%xn = wav(first_nonzero_index:first_nonzero_index+511);


N = 512; %number of samples in frame



question2 = 0;


%Form audio object to easily collect data about song
song = audioplayer(wav,fs);

if question2
    numFrames = 1;

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

    %take 512 samples starting from the first non-zero point (add 512 samples
to this )
    xn = wav(first_nonzero_index:first_nonzero_index+511);
```

```
    %Just set K to the maximum size of freqResponse. The coefficients of the
    %filterbanks are positioned in the freqResponse matrix such that values
    %will only exist near the values you see in the filterbank plot. Look at
the value of
    %freqResponse if you need that to make sense. The filter bank at Nb = 1
is small, and there
    %are a limited number of nonzero values there, as we would expect.
Whereas the
    %filterbank size at the last filter bank is large.
    K = size(freqResponse,2);

    %Compute Fourier transform of audio signal with window of size N = 512
    Y = fft (window.*xn);
    K = N/2 + 1;
    Xn = Y(1:K);

    %Just want to generate the mfcc coefficient for one frame (resulting in a
40 x 1 matrix,
    %40 rows for each of the filter banks, 1 for the single frame being
processed)
    mfcc = zeros(nBanks,numFrames);

    for p=1:nBanks
        %Implement equation (7)
        %k runs from 1 to 257 because there are 257 columns in the
        %freqResponse matrix, which carries the coefficients of the filter.

        for k = 1:K
            mfcc(p) = (abs(freqResponse(p,k)*Xn(k))^2)+mfcc(p);
        end
    end
else
    %number of frames in 24 seconds at sampling rate fs and frame size N
    numFrames = ceil(24*(fs/N));

    %initialize matrix to hold all mfcc values
    mfcc = zeros(nBanks, numFrames);

    %Get all audio samples from .wav file

    %Precondition: Audio file has been passed through audioread() prior to
    %passing anything to this functon. So audio samples have been extracted

    %Form audio object to easily collect data about song
    song = audioplayer(wav,fs);

    %Want to start in middle of the song. Gather this data before entering
loop
    middleIndex = song.TotalSamples/2;

    %Just set K to the maximum size of freqResponse. The coefficients of the
    %filterbanks are positioned in the freqResponse matrix such that values
    %will only exist near the values you see in the filterbank plot. Look at
the value of
```

```
    %freqResponse if you need that to make sense. The filter bank at Nb = 1
is small, and there
    %are a limited number of nonzero values there, as we would expect.
Whereas the
    %filterbank size at the last filter bank is large.
    K = size(freqResponse,2);

    %pull in new frames and process the next 517 frames
    for frameNumber = 1:numFrames

        %Take 512 samples. Need to get the start and end of the individual
        %frames we're currently analyzing. Increment beginning of frame by
        %512 after processing previous frame.
        frameStart = ceil(middleIndex+((frameNumber-1)*N)); %needs to be an
integer
        frameEnd   = frameStart+511;

        %extract samples from current frame of interest
        try
            extracted_audio = wav(frameStart:frameEnd);
        catch
            %This will only trigger if we go over the total amount of
            %samples. i.e. sample1.wav, is less than 24 seconds, so we
            %cannot process 24 seconds worth of frames.
            warning('Total Samples: ' + song.TotalSamples + 'frame end: ' +
frameEnd + 'frame start: ' + frameStart)
        end

        %Just want to make this explicitly clear for my future self.
        xn = extracted_audio;

        Y = fft (window.*xn);
        K = N/2 + 1;
        Xn = Y(1:K);

        %Generate mfcc coefficient matrix for each frame and filter bank
        for p=1:nBanks
            %k is essentially the size of the filterbanks.
            for k = 1:K
                mfcc(p,frameNumber) =
(abs(freqResponse(p,k)*Xn(k))^2)+mfcc(p,frameNumber); %k from 1 to 257
            end
        end
    end
end %end of if statement

 %imagesc(mfcc);

 %As requested, flipping the matrix upside down
 mfcc_flipped = flipud(mfcc);
 %mfcc_flipped = mfcc; %for debugging purposes

 fig = imagesc(10*log10(mfcc_flipped));
 title(song_path);
 xlabel('MFCC Coefficients per Frame');
 ylabel('Filterbank Number');
```

```
colorbar;
```

Data and Discussion

Below are graphical representations of the mfcc coefficients calculated for each frame and segment of the filterbank generated by the code above. Inspecting the various mfcc representations, one could quickly form the hypothesis that in general music of the same genre will contain similar patterns in the frequency content.

Classical

The song played in figure 2 was played by a piano. Not very many high frequency elements existed in this piece; the highest frequency components occurred at the end of a few of the scales played throughout the song. Figure 1, on the other hand, shows a classical piece with a higher content of high frequencies, added by the violin which was not used in the former track. Figure 2 also shows high energies near at low frequencies because of the bass that accompanied the violin.

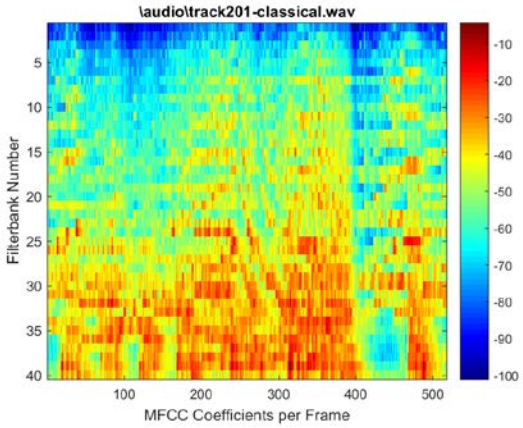

Figure 1

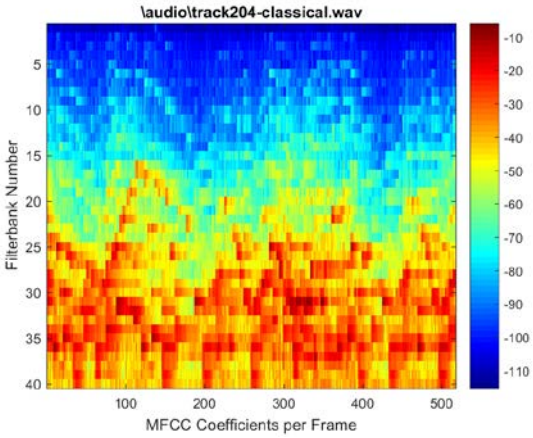

Figure 2

Ryan Sigar
ECEN 4632: Introduction to Digital Filtering
Final Project Part 1

Jazz

The mfcc representations for jazz music in figures 3 and 4, appear very similar in structure; with prevalent features of high energy, low frequency components. This is not surprising as the jazz songs that were analyzed contain music created by a variety of instruments that play in a relaxed, low tone. The drums may have been responsible of the contribution of the few high frequency component signals, while the low frequency components seem to be generated by the other lower pitch instruments (i.e. the keyboard and bass).

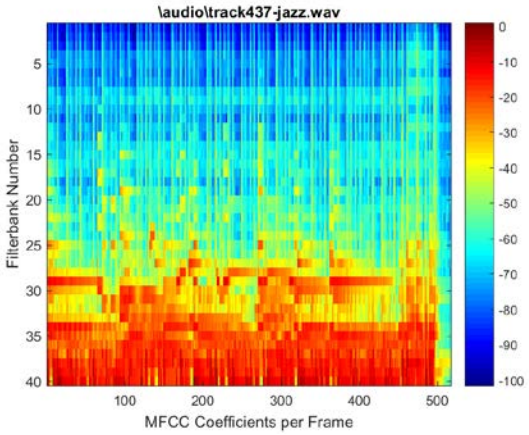

Figure 3

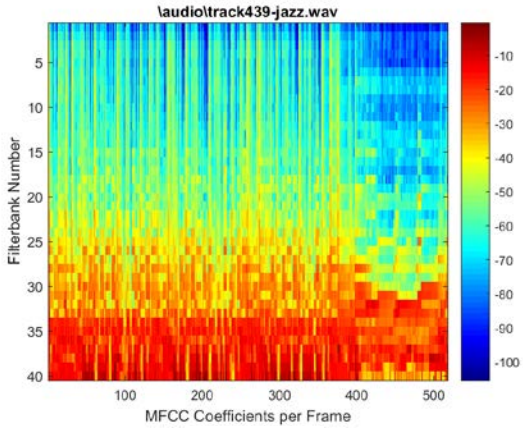

Figure 4

Electronic

The song played in figure 5 did not contain any bass, and this is apparent with the lack of energy shown at the bottom of the figure. It was a highly repetitive tune which contained sounds in the mid frequency ranges. The song in figure 3 had a very large element of bass and is clearly shown with the red in the figure. The “non-bass” elements of the song did not vary much. The fluctuations in energy seen toward the middle may have been due to the voice added midway through the song. When comparing these two songs, one noteworthy observation is that they both contain a similar lack in variation, and this can be used to identify this genre of music.

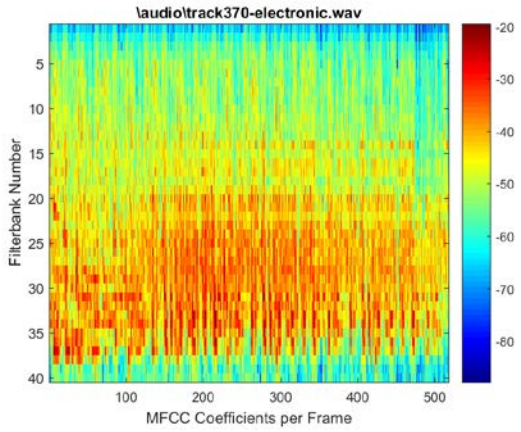

Figure 5

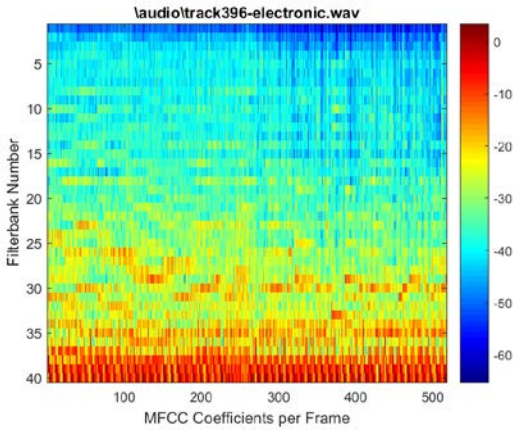

Figure 6

Rock

Looking at figures 7 and 8, we can see that both songs contained a high content of low frequencies. This is due to the bass, and other supporting instruments which played in a low relaxed manner. Figure 8 contains more middle frequency components because of the addition of acoustic guitar, and more higher frequency components because of the "chimes" added throughout the song, which did not exist in the piece analyzed in figure 7.

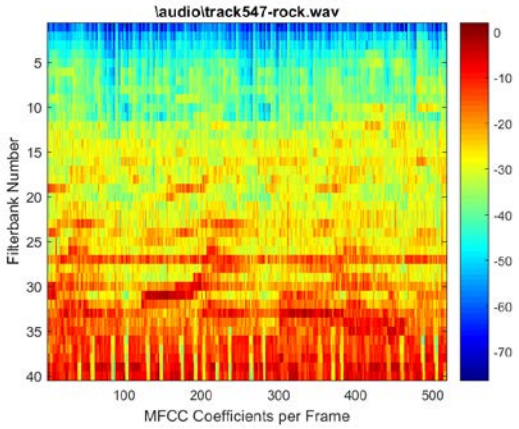

Figure 7

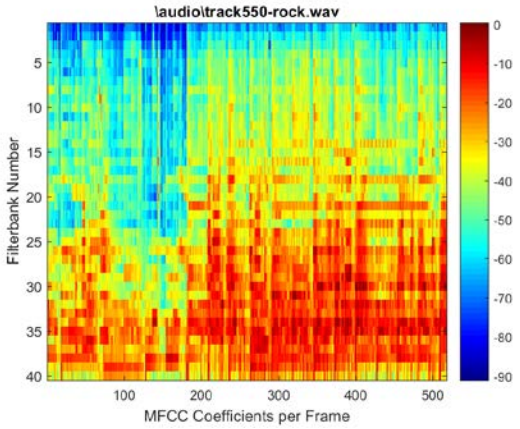

Figure 8

World

Figures 9 and 10 are very different, and this amount of variation could lead one to believe that classification of world music is difficult. The music analyzed in figure 9 was created using instruments that played in a very low tone. It was not meant to as energetic as rock or electronic music, and was more relaxed than the previous tracks. The song in figure 10 contained more variety, and did not carry the very low, ominous tone of figure 9. Unlike the song in figure 9, string instruments were included in this song which contributed to the content of high frequencies.

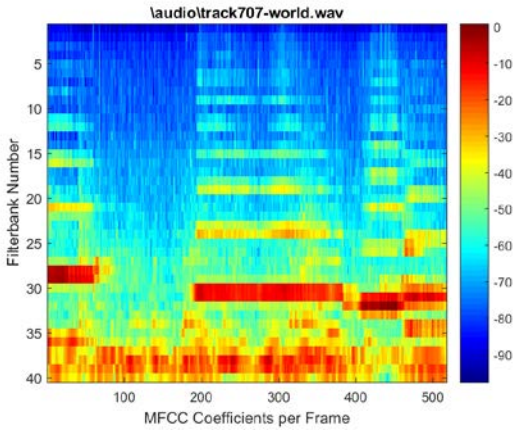

Figure 9

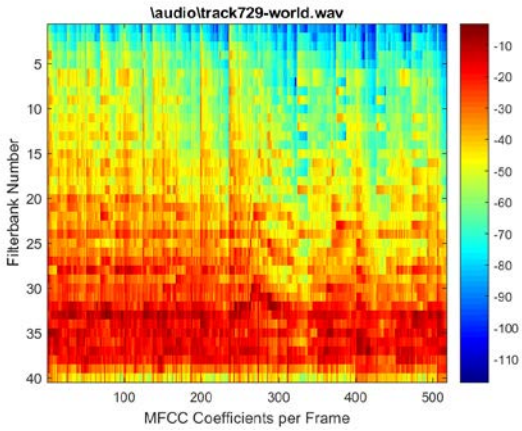

Figure 10

Metal

The song analyzed in figure 11 was very energetic, as the electric guitar was used heavily. Figure 12 contained relied more on drums and the low tones played by the electric guitar. Although both songs contained an electric guitar, the style in which they were played carries a large effect in what their frequency contents contain. As with many of the other songs.

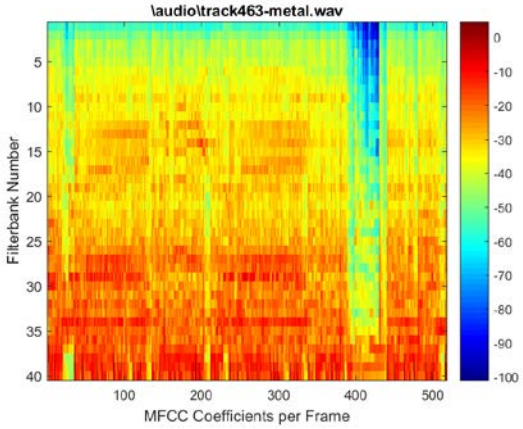

Figure 11

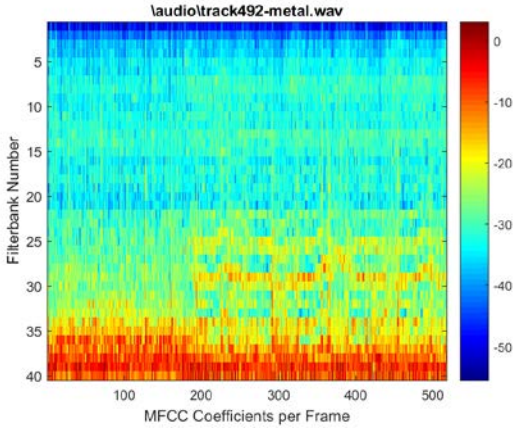

Figure 12

Appendix

How Part 1 of this Project was Completed: mfcc.m, script.m, slice_audio.m, and a subfolder 'audio' are placed into folder. Script.m calls mfcc.m and slice_audio.m and is responsible for executing code that generates all of the plots and values given above.

mfcc.m

```
function [mfcc] = mfcc(wav, fs, fftSize, window,song_path)
%
% USAGE
% [mfcc] = mfcc(wav, fs, fftSize,window)
%
% INPUT
% vector of wav samples
% fs : sampling frequency
% fftSize: size of fft
% window: a window of size fftSize
%
% OUTPUT
% mfcc (matrix) size coefficients x nFrames
% hardwired parameters

hopSize = fftSize/2;
nBanks = 40;

% minimum and maximum frequencies for the analysis
fMin = 20;
fMax = fs/2;
%_____________________________________________________________________
%
% PART 1 : construction of the filters in the frequency domain
%_____________________________________________________________________
% generate the linear frequency scale of equally spaced frequencies from 0 to
fs/2.
linearFreq = linspace(0,fs/2,hopSize+1);
fRange = fMin:fMax;
% map the linear frequency scale of equally spaced frequencies from 0 to fs/2
% to an unequally spaced mel scale.
melRange = log(1+fRange/700)*1127.01048;

% The goal of the next coming lines is to resample the mel scale to create
uniformly
% spaced mel frequency bins, and then map this equally spaced mel scale to
the linear
%& scale.
% divide the mel frequency range in equal bins
```

```
melEqui = linspace (1,max(melRange),nBanks+2);
fIndex = zeros(nBanks+2,1);
% for each mel frequency on the equally spaces grid, find the closest
frequency on the
% unequally spaced mel scale
for i=1:nBanks+2,
    [dummy fIndex(i)] = min(abs(melRange - melEqui(i)));
end
% Now, we have the indices of the equally-spaced mel scale that match the
unequally-spaced
% mel grid. These indices match the linear frequency, so we can assign a
linear frequency
% for each equally-spaced mel frequency
fEquiMel = fRange(fIndex);
% Finally, we design of the hat filters. We build two arrays that correspond
to the center,
% left and right ends of each triangle.
fLeft = fEquiMel(1:nBanks);
fCentre = fEquiMel(2:nBanks+1);
fRight = fEquiMel(3:nBanks+2);
% clip filters that leak beyond the Nyquist frequency
[dummy, tmp.idx] = max(find(fCentre <= fs/2));
nBanks = min(tmp.idx,nBanks);
% this array contains the frequency response of the nBanks hat filters.
freqResponse = zeros(nBanks,fftSize/2+1);
hatHeight = 2./(fRight-fLeft);

% for each filter, we build the left and right edge of the hat.
for i=1:nBanks,
    freqResponse(i,:) = ...
        (linearFreq > fLeft(i) & linearFreq <= fCentre(i)).* ...
        hatHeight(i).*(linearFreq-fLeft(i))/(fCentre(i)-fLeft(i)) + ...
        (linearFreq > fCentre(i) & linearFreq < fRight(i)).* ...
        hatHeight(i).*(fRight(i)-linearFreq)/(fRight(i)-fCentre(i));
end
%
% plot a pretty figure of the frequency response of the filters.
figure;set(gca,'fontsize',14);semilogx(linearFreq,freqResponse');
axis([0 fRight(nBanks) 0 max(freqResponse(:))]);title('FilterbankS');



%_____________________________________________________________
%
% PART 2 : processing of the audio vector In the Fourier domain.
%_____________________________________________________________
%
% YOU NEED TO ADD YOUR CODE HERE

%need to take a 512 sized chunk from wav (audio data from user)

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

%take 512 samples starting from the first non-zero point (add 512 samples to
this )
```

```
%xn = wav(first_nonzero_index:first_nonzero_index+511);


N = 512; %number of samples in frame



question2 = 0;


%Form audio object to easily collect data about song
song = audioplayer(wav,fs);

if question2
    numFrames = 1;

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

    %take 512 samples starting from the first non-zero point (add 512 samples
to this )
    xn = wav(first_nonzero_index:first_nonzero_index+511);

    %Just set K to the maximum size of freqResponse. The coefficients of the
    %filterbanks are positioned in the freqResponse matrix such that values
    %will only exist near the values you see in the filterbank plot. Look at
the value of
    %freqResponse if you need that to make sense. The filter bank at Nb = 1
is small, and there
    %are a limited number of nonzero values there, as we would expect.
Whereas the
    %filterbank size at the last filter bank is large.
    K = size(freqResponse,2);

    %Compute Fourier transform of audio signal with window of size N = 512
    Y = fft (window.*xn);
    K = N/2 + 1;
    Xn = Y(1:K);

    %Just want to generate the mfcc coefficient for one frame (resulting in a
40 x 1 matrix,
    %40 rows for each of the filter banks, 1 for the single frame being
processed)
    mfcc = zeros(nBanks,numFrames);

    for p=1:nBanks
        %Implement equation (7)
        %k runs from 1 to 257 because there are 257 columns in the
        %freqResponse matrix, which carries the coefficients of the filter.

        for k = 1:K
            mfcc(p) = (abs(freqResponse(p,k)*Xn(k))^2)+mfcc(p);
        end
    end
else
    %number of frames in 24 seconds at sampling rate fs and frame size N
```

```
    numFrames = ceil(24*(fs/N));

    %initialize matrix to hold all mfcc values
    mfcc = zeros(nBanks, numFrames);

    %Get all audio samples from .wav file

    %Precondition: Audio file has been passed through audioread() prior to
    %passing anything to this functon. So audio samples have been extracted

    %Form audio object to easily collect data about song
    song = audioplayer(wav,fs);

    %Want to start in middle of the song. Gather this data before entering
loop
    middleIndex = song.TotalSamples/2;

    %Just set K to the maximum size of freqResponse. The coefficients of the
    %filterbanks are positioned in the freqResponse matrix such that values
    %will only exist near the values you see in the filterbank plot. Look at
the value of
    %freqResponse if you need that to make sense. The filter bank at Nb = 1
is small, and there
    %are a limited number of nonzero values there, as we would expect.
Whereas the
    %filterbank size at the last filter bank is large.
    K = size(freqResponse,2);

    %pull in new frames and process the next 517 frames
    for frameNumber = 1:numFrames

        %Take 512 samples. Need to get the start and end of the individual
        %frames we're currently analyzing. Increment beginning of frame by
        %512 after processing previous frame.
        frameStart = ceil(middleIndex+((frameNumber-1)*N)); %needs to be an
integer
        frameEnd   = frameStart+511;

        %extract samples from current frame of interest
        try
            extracted_audio = wav(frameStart:frameEnd);
        catch
            %This will only trigger if we go over the total amount of
            %samples. i.e. sample1.wav, is less than 24 seconds, so we
            %cannot process 24 seconds worth of frames.
            warning('Total Samples: ' + song.TotalSamples + 'frame end: ' +
frameEnd + 'frame start: ' + frameStart)
        end

        %Just want to make this explicitly clear for my future self.
        xn = extracted_audio;

        Y = fft (window.*xn);
        K = N/2 + 1;
        Xn = Y(1:K);
```

```
        %Generate mfcc coefficient matrix for each frame and filter bank
        for p=1:nBanks
            %k is essentially the size of the filterbanks.
            for k = 1:K
                mfcc(p,frameNumber) =
(abs(freqResponse(p,k)*Xn(k))^2)+mfcc(p,frameNumber); %k from 1 to 257
            end
        end
    end
end %end of if statement

 %imagesc(mfcc);

 %As requested, flipping the matrix upside down
 mfcc_flipped = flipud(mfcc);
 %mfcc_flipped = mfcc; %for debugging purposes

 fig = imagesc(10*log10(mfcc_flipped));
 title(song_path);
 xlabel('MFCC Coefficients per Frame');
 ylabel('Filterbank Number');
 colorbar;
```

script.m

```
close all

%Put all song names into a cell array
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'
            };

%Question 1:
%Write a MATLAB function that extract T seconds of
%music from a given track. You will use the MATLAB function audioread to
%read a track and the function play to listen to the track.


%take_audio input parameters: audio file, amount of time to extract, middle
%or not middle switch

%Choose your song according to the index number given above. 1 corresponds to
track201-classical.wav
%and 13 corresponds to sample1.wav.
```

```
songChoice = 1;

pathToSong = char(songList(songChoice));

amountOfTime = 3; %seconds
takeFromMiddle = 1; %software switch to enable taking audio from middle

[song_object, audio_data, fs ,start, stop] = slice_audio(pathToSong,...
                                                amountOfTime,takeFromMiddle);

%sound(audio_data,fs) %don't use this, you won't be able to pause the music

play(song_object,[start,stop])



%Question 2:
%Implement the computation of the mfcc coeffcients, as defined in
%(7). You simply need to add your code in the MATLAB code in the previous
pages.

%Note: these are the values extracted from part 1: fs, song_object,
%audio_data, fs ,start frame, and stop frame

%samples per frame
N = 512;

%want this to be the number of samples per frame (i think)
fftSize = N;

%Create a window
% w = kaiser(N,beta);
w = hann(fftSize);

%Compute mfcc coefficient for one frame of the song that pathToSong points
%to. This should produce a boring graph of the mfcc coefficients of one
frame.
y=mfcc(audio_data,fs,fftSize,w,pathToSong);


%Question 3:
%Evaluate your MATLAB function mfcc on the 12 audio tracks, and display the
output as
%an image using imagesc. You will use T =24 seconds from the middle of each
track and
%compute a matrix of mfcc coefficients of size NB = 40 rows and 24 ×
11,025/512 = 517
%columns.

%Generate plots and compute the mfcc coefficients for each of the 12 songs
%given.

for songChoice = 1:12
    pathToSong = char(songList(songChoice));
    [song_object, audio_data, fs ,start, stop] = slice_audio(pathToSong,3,1);
    y=mfcc(audio_data,fs,fftSize,w,pathToSong);
```

```
end
```

slice_audio.m

```
function [song, audio_data, Fs, start, stop] =
slice_audio(audio_filepath,time_slice,from_middle)
    %Extracts T seconds from a given song (as explicitly required in question
1 and 3)

    %Usage:
    %Inputs: provide path to file, amount of time you want to extract and
    %a boolean value for if you want to take out information from the
    %middle or from the beginning (no other location supported yet)

    %Outputs:
    %Song Object, raw audio data, sampling rate, start time, stop time.

    %Read audio file
    [audio_data, Fs] = audioread(audio_filepath);

    %Form audio object to easily collect data about song
    song = audioplayer(audio_data,Fs);

    if not(from_middle)
        %Extract information from beginning to time "time_sample"
        start = 1;                          %Indicates beginning
        stop = song.SampleRate*time_slice;  %Ending time defined by user
    else
        %Take from the middle
        start = song.TotalSamples/2;
        %End at specified time after middle of song
        stop = start + song.SampleRate*time_slice;
    end
end
```