Ryan Sigar (Partners: SJ Park, Jeong Min Lee)
ECEN 4632: Introduction to Digital Filtering
Final Project Part 3

Solutions to Question 4:

"Implement the computation of the distance d given by (24), and Fig. 7. Your function should be able to use the 12 merged MFCC coefficients or the Normalized Pitch Class Profile."

The code which performs this operation is written under the function distanceBetweenSongs(), and is given in the appendix below. The function takes in the MFCC coefficients or NPCP values calculated using the algorithms implemented in the previous parts of this project and computes the distances using the symmetric Kullback-Leibler divergence approach described in the project handout.

Solution to Question 5:

"Compute the matrix of pairwise distance detailed in equation 25. Then display the distance matrix as an image, and discuss its structure."

The code that performs this operation is written under the part3.m MATLAB script and is given in the appendix. Part3.m uses the code developed for the previous question and iteratively computes the distance between each song in the given data set. To reduce computation time of the pairwise distance matrix, the code was segmented into two stages: (1) pre-computation of the MFCC/NPCP values, then (2) iteration over each song and using the pre-computed values to compute the final piecewise distance matrices. Avoiding the segmentation of these two stages and interleaving the steps to compute the final distance matrices results in an unacceptably long computation time. More details and comments of this process are included in the part3.m file description in the appendix.

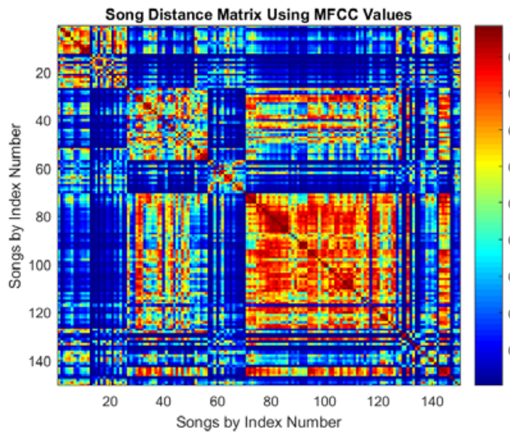

Figure 1

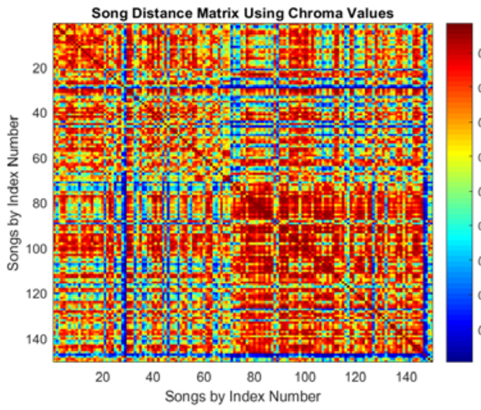

Figure 2

Shown above are the resulting pairwise distance matrices from running the part3.m script. The Index of each song was computed as such:

$$Index\ Number = (Genre\ Number - 1) * 25 + Song\ Number\ In\ Genre\ Subfolder,$$

Where Genre Number = 1,2,3,4,5, or 6 referring to the classical, electronic, jazz, punk, rock, and world genres respectively, and Song Number is position the song appears in the genre subfolder when sorted in descending alphabetic and numeric order (positions 1 to 25).

Due to the fact that Figure 2 is heavily saturated in red, and distinct regions of interest are difficult to find, the results shown in Figure 1 will be used for discussion.

At least nine regions of interest can be distinguished in Figure 1. The largest region of difference occurs approximately where songs 75 to song 125 are compared. Here the genres being compared are Jazz (songs 75 to 100) and Punk (songs 100 to 125). When listening to these genres of music, it is clear that this large separation is due to the inherent style, tempo, and the instruments that each piece of music uses within that genre. Some works of Jazz may contain tempos similar to those in Punk, but in general the Punk genre contains higher energy than most Jazz works due to its aggressive style.

The region to the left of this area are the resulting distances when comparing the music in the Electronic genre (songs 25 to 50) to Rock (songs in 50 to 75), Punk (75 to 100), and Jazz (100 to 125). In general, these discrepancies are lesser than those of the formerly discussed region. There are obviously many inherent (perhaps subjective) differences between these genres (i.e. instruments/equipment used to produce the sounds, overall repetitiveness and structure, the amount of bass in a piece, etc.), but a number of these songs contained similar characteristics. For instance, when comparing song 40 to the music in the other 3 genres, similarities in frequency content (bass and mid frequencies) can be observed.

Additionally, high levels of distance between songs can be seen internally within certain genres, such as classical and electronic. This simply demonstrates that the music belonging to those genres can be highly diverse with respect to their possible frequency contents.

Solution to Question 6:

"Compute the 6 x 6 average distance matrix between the genres, defined by equation 26. Experiment with different values of gamma, and find a value of gamma that maximizes the separation between the different genres, as defined by the 6 x 6 average distance matrix D."

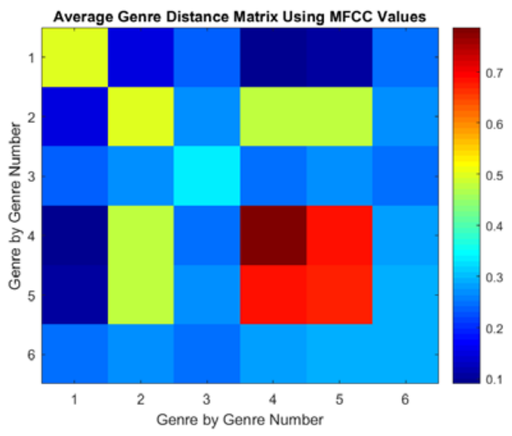

Figure 3

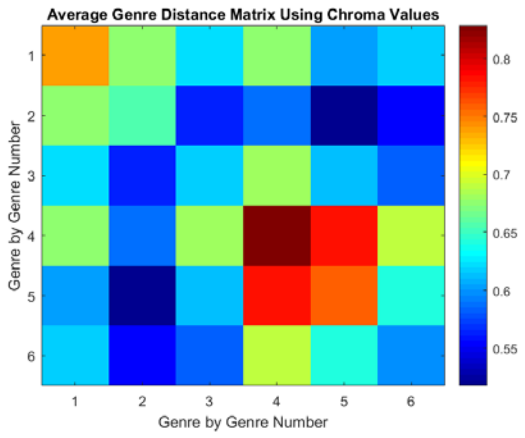

Figure 4

Above are the resulting matrices detailing the average distance between genres using MFCC Coefficients (frequency content) and NPCP (Chroma; semitone content) values. Information from the pairwise Chroma distance matrix has been compacted into the results shown in Figure 4. In contrast with the information presented in figure 3, this matrix shows the distance between genres based on the average semitone that each genre is characterized by in the given data set. The highest distances still exist in the red region where genres 4 and 5 are compared amongst each other, as well as internally within the classical and electronic genres. However – interestingly enough - the blue region comparing genre 2 (Electronic) to genres 4 and 5 (Punk and Rock) in this image indicate that these genres are characterized by similarities in semitone. This change in distance indicate that these two pieces of information (average distance based on frequency content and semitone content) are necessary to classify the genre of a given song.

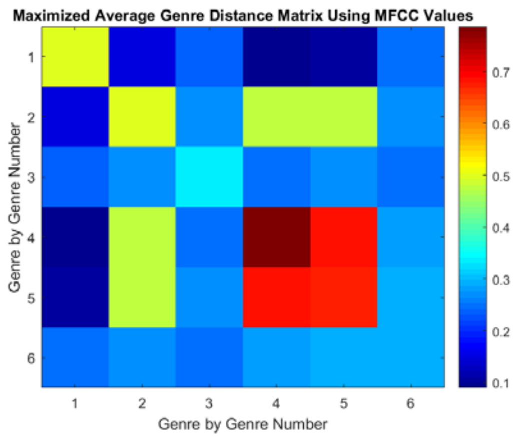

Figure 5

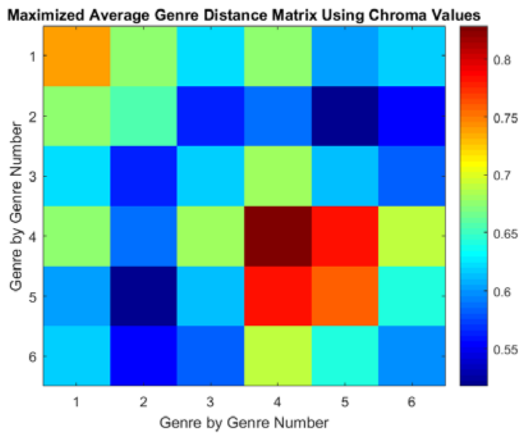

Figure 6

The function find_optimal_gamma_values() (described in the appendix) computes the KL values (given the Average Distance Matrix, floating point relative accuracy (eps in MATLAB), and the gamma value used to compute the average distances), performs a sweep across gamma values from 10 to 100, and determines the optimal gamma values in order to obtain the most distance between each genre. The results of this analysis are shown in figures 5 and 6. These results are identical to those of figures 3 and 4, which were computed using a gamma value of 100. Therefore, the results indicate that the optimal gamma value to increase the distance between each genre is 100.

Ryan Sigar (Partners: SJ Park, Jeong Min Lee)
ECEN 4632: Introduction to Digital Filtering
Final Project Part 3

Appendix

How Part 3 of this Project was Completed: part3.m, distanceBetweenSongs.m and a subfolder ‘data’ (containing the songs used for this project) are placed into a folder which is added to the workspace in MATLAB. The current folder in the MATLAB workspace must be pointed to the folder containing all of these files for Part3.m to run. Part3.m then pre-computes the MFCC and Chroma values and then iteratively computes pairwise distance matrices and average distance matrices using each song, and the two structs containing the precomputed values MFCC and Chroma values.

part3.m

```
%-------------------------------------------------------------------------
%                       VARIABLE/MATRIX INITIALIZATION
%-------------------------------------------------------------------------


numberOfSongs = 150;
numberOfSongsInFolder = 25;
numberOfGenres = 6;


%IMPORTANT: Matlab must be operating in the folder containing for the
%following two lines to work. Will need to eliminate this dependency sometime.
datasetSubfolder = './data/';
genreSubfolders = dir(datasetSubfolder);

%Ad-hoc solution to get rid of source controlled remnants detected in
%the folder. These are added as the first contents in the struct
genreSubfolders(1:2) = [];

%Amount of seconds in two minutes
twoMinutes = 2*60;

%frame size shall be 512 as described in part 3
frameSize_N = 512;
window = hann(frameSize_N);

%Matrix to contain info of average distance from genreX to genreY
avgGenreDistanceUsingMFCC = zeros(numberOfGenres,numberOfGenres);
avgGenreDistanceUsingChroma = zeros(numberOfGenres,numberOfGenres);

%Matrix to contain info of distance of songX from songY
songDistancesUsingMFCC = zeros(numberOfSongs,numberOfSongs);
songDistancesUsingChroma = zeros(numberOfSongs,numberOfSongs);

%Data structure to hold all precomputed mfcc values
precomputed_mfcc_values = struct('result',[]);

%Data structure to hold all precomputed chroma values
precomputed_chroma_values = struct('result',[]);

%Data structure holding the filepaths to all songs to be analyzed
song = struct('filepath',[]);
```

```
%-----------------------------------------------------------------------------
%                                COMPUTATIONS
%-----------------------------------------------------------------------------
%Catalog the filepaths to each song

for i = 1:numberOfGenres
    %Get path to genre subfolder
    path_to_genre_subfolders = [datasetSubfolder genreSubfolders(i).name '/'];
    songs_in_subfolder = dir(path_to_genre_subfolders);

    %Get rid of source controlled remnants (usuallyy first two files listed)
    songs_in_subfolder(1:2) = [];

    for j = 1:25
        %Create an index for each song.
        songIndex = ((i-1)*25) + j;
        %Catalog song path in a struct
        song(songIndex).filepath = [path_to_genre_subfolders
songs_in_subfolder(j).name];
    end
end

%Pre-load MFCC coefficients
try
    load('precomputed_mfcc_values.mat')
    need_to_compute_mfcc = 0;
catch
    warning('File: precomputed_mfcc_values.mat not present.')
    need_to_compute_mfcc = 1;
end


%Pre-Compute MFCC coefficients if no values can be preloaded
%(1.5 minutes using a i7-6700U dual core laptop)
if need_to_compute_mfcc
    tic
    for i = 1:numberOfGenres
        for j = 1:numberOfSongsInFolder
           [song_signal, song_sampleRate] =
slice_audio(song(songIndex).filepath,twoMinutes,1);


           songIndex = ((i-1)*25) + j;

           mfcc_result = mfcc(song_signal,song_sampleRate,frameSize_N,window);
           precomputed_mfcc_values(songIndex).result = mfcc_result;
        end
    end
    toc
end

%Pre-load chroma coefficients
try
    load('precomputed_chroma_values.mat')
    need_to_compute_chroma = 0;
catch
    warning('File: precomputed_chroma_values.mat not present.')
    need_to_compute_chroma = 1;
end


%Pre-Compute Chroma values if no values can be preloaded
```

```
if need_to_compute_chroma
    tic
    for i = 1:numberOfGenres
        for j = 1:numberOfSongsInFolder
           [song_signal, song_sampleRate] = 
slice_audio(song(songIndex).filepath,twoMinutes,1);

           songIndex = ((i-1)*25) + j;

           %chroma_result = NormPitchClassProfile(song_signal);
           chroma_result = mychroma(song_signal,song_sampleRate,512);
           precomputed_chroma_values(songIndex).result = chroma_result;
        end
    end
    toc
end


%-------------------------------------------------------------------------
%Computing the Individial and Average Distance Matrices
%-------------------------------------------------------------------------
tic
for genreX = 1:6
    for songX = 1:25
        songXIndexOffset = ((genreX-1)*25);
        songXIndex = songXIndexOffset + songX;
        for genreY = genreX:6
            for songY = 1:25
                %Computing index (song number) for songY
                songYIndexOffset = ((genreY-1)*25);
                songYIndex = songYIndexOffset + songY;

                %Code used for debugging
                %fprintf('genreX = %d, songX = %d, genreY = %d, songY = 
%d\n',genreX,songX, genreY, songY)
                %fprintf('distance(%d,%d)\n\n',songXIndex,songYIndex)

                %---------------------------------------------------------
                %Compute Distances using MFCC coefficients
                %---------------------------------------------------------
                try
                 songDistancesUsingMFCC(songXIndex,songYIndex) = ...
                 distanceBetweenSongs(precomputed_mfcc_values(songXIndex).result, ...
                                     precomputed_mfcc_values(songYIndex).result);

                 %Map values to other side of diagonol to complete the matrix
                 songDistancesUsingMFCC(songYIndex,songXIndex) = 
songDistancesUsingMFCC(songXIndex,songYIndex);
                catch
                    warning('Something is wrong.')
                    fprintf('Trouble computing songX = %d, songY = 
%d',songXIndex,songYIndex);
                end
                avgGenreDistanceUsingMFCC(genreX,genreY) = 
songDistancesUsingMFCC(songXIndex,songYIndex)...
                                                  + 
avgGenreDistanceUsingMFCC(genreX,genreY);
                %Map values to it's reflection across the matrix's diagonol
                avgGenreDistanceUsingMFCC(genreY,genreX) = 
avgGenreDistanceUsingMFCC(genreX,genreY);

                %---------------------------------------------------------
                %Compute Distances using chroma
                %----------------------------------------------------------
                 try
                 songDistancesUsingChroma(songXIndex,songYIndex) = ...
                 distanceBetweenSongs(precomputed_chroma_values(songXIndex).result,
...
                                      precomputed_chroma_values(songYIndex).result,0);

                 %Map values to other side of diagonol to complete the matrix
                 songDistancesUsingChroma(songYIndex,songXIndex) =
songDistancesUsingChroma(songXIndex,songYIndex);
                catch
                    warning('Something is wrong.')
                    fprintf('Trouble computing songX = %d, songY =
%d',songXIndex,songYIndex);
                end
                avgGenreDistanceUsingChroma(genreX,genreY) =
songDistancesUsingChroma(songXIndex,songYIndex)...
                                                 +
avgGenreDistanceUsingChroma(genreX,genreY);
                %Map values to it's reflection across the matrix's diagonol
                avgGenreDistanceUsingChroma(genreY,genreX) =
avgGenreDistanceUsingChroma(genreX,genreY);
            end
        end
     end
end
toc

%Compute the average distance: divide by number of data points used to
%compute the total distances (625)
avgGenreDistanceUsingMFCC = (1/(25^2))*avgGenreDistanceUsingMFCC;
avgGenreDistanceUsingChroma = (1/(25^2))*avgGenreDistanceUsingChroma;


figure
fig1 = imagesc(max_values_chroma)
colormap('jet')
title('Maximized Average Genre Distance Matrix Using Chroma Values');
ylabel('Genre by Genre Number');
xlabel('Genre by Genre Number')
colorbar

figure
fig2 = imagesc(songDistancesUsingChroma);
title('Song Distance Matrix Using Chroma Values');
ylabel('Songs by Index Number');
xlabel('Songs by Index Number')
colormap('jet')
colorbar

figure
fig3 = imagesc(avgGenreDistanceUsingChroma);
title('Average Genre Distance Matrix Using Chroma Values');
ylabel('Genre by Genre Number');
xlabel('Genre by Genre Number');
colormap('jet')
colorbar

figure
fig4 = imagesc(songDistancesUsingMFCC);
title('Song Distance Matrix Using MFCC Values');
ylabel('Songs by Index Number');
xlabel('Songs by Index Number')
```

```
colormap('jet')
colorbar

figure
fig5 = imagesc(avgGenreDistanceUsingMFCC);
title('Average Genre Distance Matrix Using MFCC Values');
ylabel('Genre by Genre Number');
xlabel('Genre by Genre Number');
colormap('jet')
colorbar
```

distanceBetweenSongs.m

```
function d = distanceBetweenSongs(mfcc_or_npcp1, mfcc_or_npcp2)
    %mfcc coefficiencts of song 1 and 2 respectively

    %mfcc(wav, fs, fftSize, window)
    %[song1_signal, ] = audioread(song1_path)

    %Redefinition of filter bank
    t = zeros(1,36);
    t(1) =1;t(7:8)=5;t(15:18)= 9;
    t(2) = 2; t( 9:10) = 6; t(19:23) = 10;
    t(3:4) = 3; t(11:12) = 7; t(24:29) = 11;
    t(5:6) = 4; t(13:14) = 8; t(30:36) = 12;


    %{
    -----------------------Pass mfcc1 through merged mel bank filters-----------------
------
    %}
    mel2 = zeros(12,size(mfcc_or_npcp1,2));
    for i=1:12,
        mel2(i,:) = sum(mfcc_or_npcp1(t==i,:),1);
    end
    mfcc_or_npcp1 = mel2;


    %{
    -----------------------Pass mfcc2 through merged mel bank filters-----------------
------
    %}
    mel2 = zeros(12,size(mfcc_or_npcp2,2));
    for i=1:12,
        mel2(i,:) = sum(mfcc_or_npcp2(t==i,:),1);
    end
    mfcc_or_npcp2 = mel2;


    %{
    -----------------------4.7 COMPUTING DISTANCE BETWEEN FEATURES--------------------
---
                                Answers questions 6 and 7
    %}

    %{
        In general, we are collapsing the information gathered about the
        songs, in previous section, into an average note, and a covariance
        matrix (telling us how much each element of the song is correlated to itself.
- not a
        very good explanation of it...)

        Mean note of song used to categorize it
        mu = mean(mfcc,2);

        Covariance of song used to categorize it
        Cov = cov(mfcc);

        The Kullback-Leibler divergence is given in equation (23) on page
        20

        Computing this value will allow us to compute the distance between
        two songs.

    %}

    %Computing the distance d between the two tracks s1 and s2

    %Going to include implementation of Professor Meyer's code here:

    mu1 = mean(mfcc_or_npcp1,2);
    mu2 = mean(mfcc_or_npcp2,2);

    co1 = cov(mfcc_or_npcp1');
    co2 = cov(mfcc_or_npcp2');

    try
       %tic
            iCo1 = pinv(co1);
            iCo2 = pinv(co2);
        %toc
    catch
        warning('Pseudo-Inverse not working.')
    end

    %{
        try
            tic
                iCo1 = inv(co1);
                iCo2 = inv(co2);
            toc
        catch
            warning('Inverse not working.')
        end
    %}

    %{
        Moore-penrose pseudoinverse for a singluar matrix

        ico1 = pinv(co1);
        ico2 = pinv(co2);
    %}

    %tic
        KL = trace(co1*iCo2) + trace(co2*iCo1) + (mu1-mu2)'*(iCo1+iCo2)*(mu1-mu2);
    %toc
    gam = 1e2;

    %eps is the floating point relative accuracy
    d = 1 - exp(-gam/(KL + eps));
end
```

find_optimal_gamma_values.m

```
function [max_avg_genre_distances, optimal_gamma_values] =
find_optimal_gamma_values(avg_genre_distance_matrix)
%Gamma value used in computing the avg_genre_distance_matrix values
```

```
gamma_value_used_to_compute_avg_distance_matrix = 100;

%Need to solve for the average KL value (refer to equation 24)
average_KL = (-gamma_value_used_to_compute_avg_distance_matrix./log(1-
avg_genre_distance_matrix))-eps;

optimal_gamma_values = zeros(6,6);
%genre_distances = zeros(6,6);
max_avg_genre_distances = zeros(6,6);

for gamma = 10:100

   genre_distances = abs(1 - exp(-gamma./(average_KL + eps)));

   max_occurances = genre_distances > max_avg_genre_distances;

   %Find the maximum values
   max_avg_genre_distances = (genre_distances.*max_occurances) ...
                            +(~max_occurances).*max_avg_genre_distances;

   %Find the optimal gamma values
   optimal_gamma_values = (gamma*max_occurances) ...
                            +(~max_occurances).*optimal_gamma_values;
end

end
```

slice_audio.m

```
function [audio_data, Fs] = 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:
    %audio_data = audio_data

    %Software switch to indicate we're working with part 3 of final project
    part3 = 1;

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

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

    %Song duration in seconds
    songDuration = song.TotalSamples/song.SampleRate;

    %Assume song will always be long enough (for now)
    song_is_long_enough = 1;

    %{
    -----------------------SLICE AUDIO MODIFIED TO EXTRACT 2 MIN FROM SONG------------
-----------
    %}
    %Criteria for a good song: we can extract two minutes starting at
    %middle of song
    if from_middle
        song_is_long_enough = ((songDuration/2) >= time_slice);
    end


    if song_is_long_enough && from_middle && part3
        %We can extract two minutes starting at middle of song.

        %Take two min of audio data from song_path starting from the middle
        %{
            Definitions of variables below:

            songObject - audio player object generated by slice_audio()
            fs = sampling rate of song
            start = frame at middle of song
            stop = end of two minute sample
        %}

        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


    elseif part3 && ~song_is_long_enough
        %If working on part 3, and song is not long enough...
        %Cannot extract two minutes starting from middle. Use the whole song.
        start = 1;
        stop = song.TotalSamples;
    elseif ~part3 && ~from_miffle
        %Normal audio slice. Take amount of time specified by user starting
        %at beginning
        start = 1;
        stop = song.SampleRate*time_slice;
    end

    %Slice the audio
    try
        audio_data = audio_data(start:stop);
    catch
        warning('Could not slice audio. Is song long enough?')
    end
end
```