Lab 4

Clint Olsen

ECEN 4532: DSP Lab

April 3, 2018

University of Colorado
Boulder

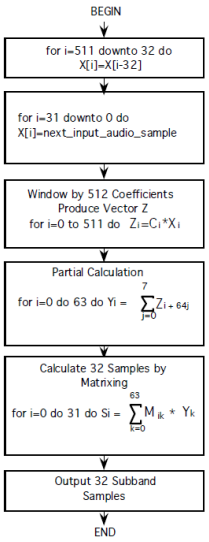
Contents

Introduction

This lab focuses on implementing two of the major stages in the MPEG 1 Layer III (mp3) audio codec. There are normally three stages: analysis, compression, and synthesis. In this lab, only the analysis and synthesis stages will be explored in detail, with a very small introduction to basic and crude compression (which is usually done using a quantization scheme when analyzing the energy associated with particular frequency sub-bands). These stages will be implemented using MATLAB using provided audio tracks for analysis. Through the sub-band analysis of each of these tracks, an intuition can be built for at which frequencies compression can occur (if compression were to be implemented), as well as how reconstruction affects the audio signals.

Assignment 1

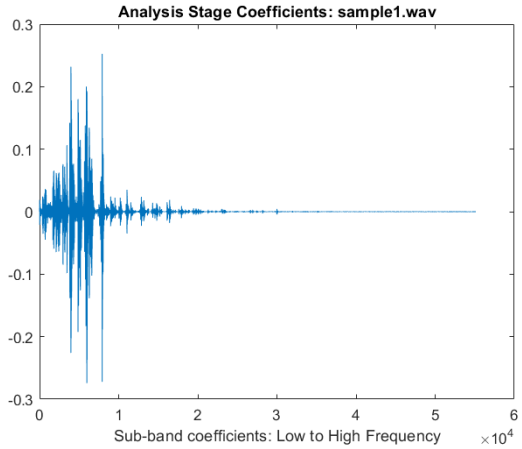
The goal of the first stage, analysis, is to take a given audio signal, break it up into 32 sub-bands, filter the sub-bands, and down sample the result of each of the individual filters in the 32 tap filter bank. The way that the 32 sub-bands are determined is through the use of a cosine modulated pseudo quadrature mirror filter. Put more simply, the input signal will be multiplied by a cosine function that is modulated to be centered around a particular frequency that is desired for that particular sub-band. This will occur 32 times, where the cosine is modulated to be centered around 32 different frequencies, resulting in the 32 sub-bands. From here, each sub-band will be comprised of the same number of samples as the original signal, resulting in 32 times the amount of desired samples. To solve this, each sub-band is downsampled by a factor of 32, which maintains the desired frequency components of that particular sub-band. The mathematics in the first part of the lab describe how this procedure can be re-written in an algorithmically optimal way, which ultimately leads to the programmatic implementation described by the MPEG standard in the figure below. The algorithmic optimization reorders several of the mathematical steps for speed, which explains discrepancies between the mathematics and the programming procedure.

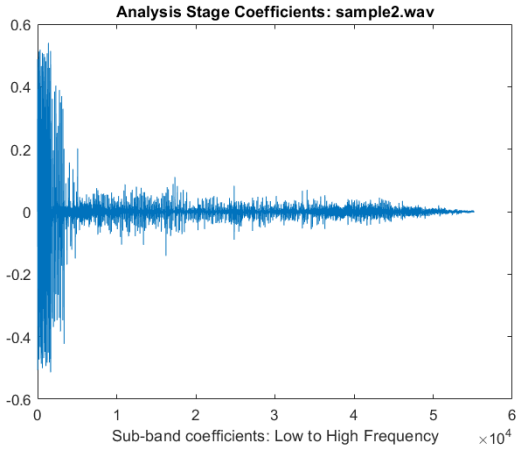


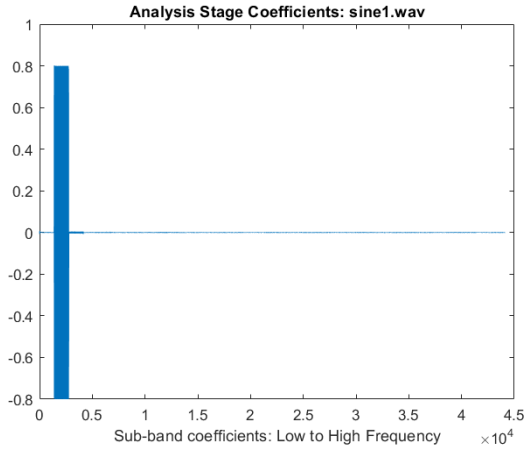
The MATLAB implementation consists of breaking up the audio track into a matrix with 32 rows and a number of columns corresponding with how many groups of 32 samples there are in the song as a whole. Each column can then be read as a 32 sample packet, which is then shifted into a buffer of size 512. This occurs because the audio analysis needs to happen with a vector that is the same size as the PQMF filter (size 512). By shifting in 32 samples at a time, the input is effectively downsampled prior to filtering. From here, the intermediate calculations extract the 32 sub-band samples for this particular iteration (where sub-band 1 is the lowest frequency and 32 is the highest). This process is then repeated for each group of 32 samples in the track. The final step that needs to be taken is a mathematical result of downsampling in the frequency domain, frequency inversion. Frequency inversion is caused by aliasing from the periodic signal in the Fourier domain on the interval $[-\pi, \pi]$. However, this only occurs on every sample in every other sub-band. Frequency inversion can be un-inverted by modulating the signal in the frequency domain, which in the time domain is a multiplication by a complex exponential. Therefore, the frequency inversion is correct by multiplying every other sample of every other sub-band by -1, which is every other value of the sequence $e^{i\pi n}$. The MATLAB code for the analysis stage can be found in the Assignment 1 Code Appendix.

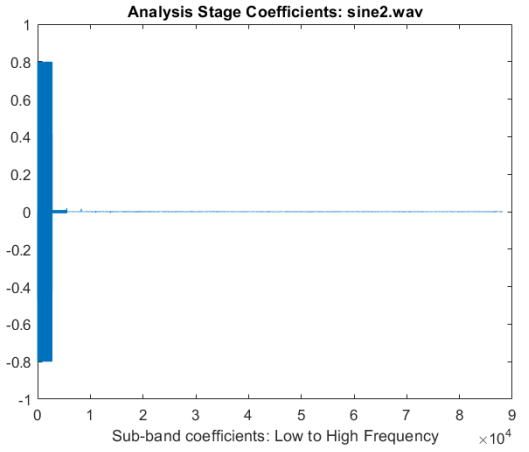
Assignment 2

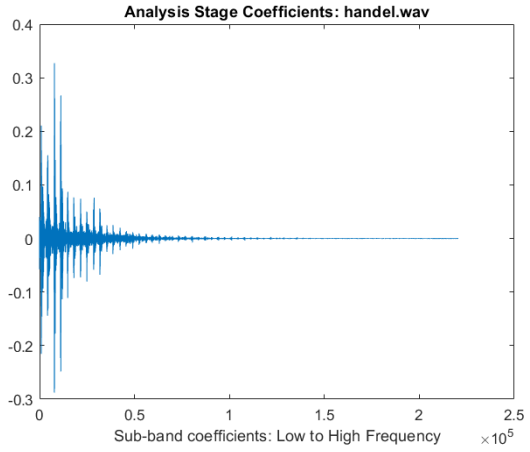
The output of the previous stage results in a MATLAB vector of sub-band coefficients where the values in the array are ordered from low to high frequencies. This array can then be plotted to show at what frequencies the energy exists within the track overall, based on this sub-band analysis. This is the first step to a compression algorithm, but for now it can be used to quantitatively and aurally confirm the results of the analysis stage. The following results are described using the first five seconds of the seven provided tracks:

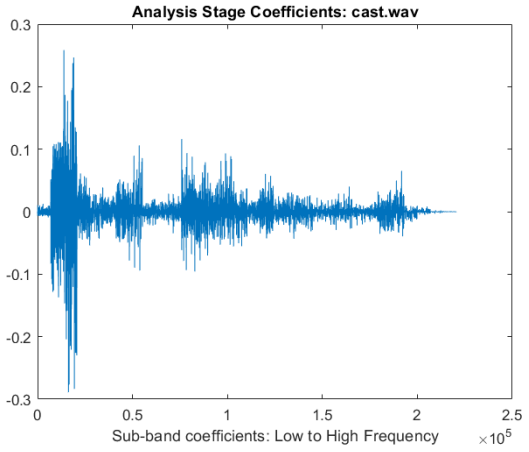


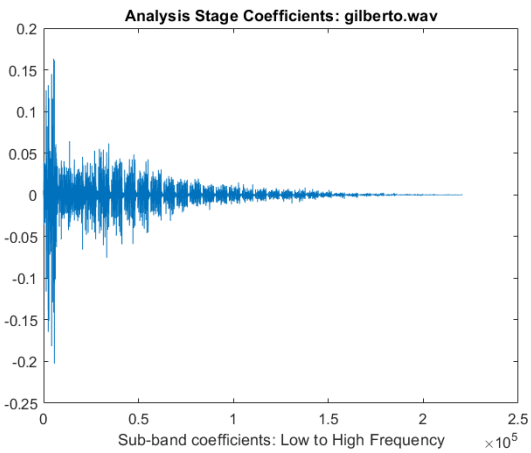



Analysis Stage Coefficients: sine1.wav
Sub-band coefficients: Low to High Frequency
×10^4


Analysis Stage Coefficients: sine2.wav
Sub-band coefficients: Low to High Frequency
×10^4


Analysis Stage Coefficients: handel.wav
Sub-band coefficients: Low to High Frequency
×10^5


Analysis Stage Coefficients: cast.wav
Sub-band coefficients: Low to High Frequency
×10^5


Analysis Stage Coefficients: gilberto.wav
Sub-band coefficients: Low to High Frequency
×10^5

The first two tracks that can be interpreted from these plots, which can be used as a type of control experiment, are the *sine*1.*wav* and *sine*2.*wav* tracks. In listening to the tracks, they are both constant tones as one frequency. *sine*1.*wav* is the higher pitch of the two, which is reflected in the plot which shows more energy in higher frequency sub-band coefficients, where conversely, *sine*2.*wav* shows more energy at low frequency coefficients. Due to the lack of harmonic complexity, that means these signals could be greatly compressed at all other frequencies, greatly reducing the overall number of samples needed to represent this audio. Next, *sample*1.*wav* is a solo piano piece that contains both a dominant melody in the right hand of the player, as well as harmonic structure and chords played by the left hand of the player at the lower to mid range of the piano. This explains some of the higher frequency coefficients (which are from the melody) and the low and mid frequencies played which are fairly evenly distributed across the spectrum in this range. This would be a challenging piece to compress due to the fact a wide range of frequencies are important in the aural characteristic of the piece as a whole. *sample*2.*wav* is an electronic dance piece with a dominating constant and loud bass drum, which explains the high energy in the low frequency. The recording itself is not very high quality upon listening to it, so a lot of the higher frequency components spread across the spectrum are likely a result of noise, so when compression occurs, these regions could be compressed and quantized to less bits. *handel.wav* consists of an orchestra as well as a choir with several dominating vocal ranges (soprano, alto, bass etc.) that can be all heard with relatively even volume and energy. This is shown in the similar amplitude frequency spikes that are evenly spaced across the spectrum, where the higher amplitude spikes are likely the violin melody played at mid range frequencies. High frequencies could likely be quantized down in this with acceptable results. *cast.wav* is a solo castanet piece that has an interesting spectrum. Due to the interesting timbre of the instrument itself, it is likely is has a complex harmonic structure, which explains the response at many different frequencies for a single instrument, where the fundamental frequency lies near the low to mid frequency range. Compression may be difficult here as well because quantizing certain frequency components may lead to loss in the characteristic sound of the instrument itself by distorting its harmonic structure. Finally, *gilberto.wav* is a Brazilian piece that begins with a lower male voice that is the spike at low frequency coefficients. This is followed by a loosely tuned drum that is likely the mid to range frequencies shown due to the higher harmonics created by the drum. Compression here would be difficult if more than the first 5 seconds are listened to because it would seem that high frequencies could be compressed in the first 5 seconds, but later on a flute with a rhythmic melody begins, which could be lost with compression. The MATLAB code for the plots can be found in the Lab 4 Driver Script Code Appendix.

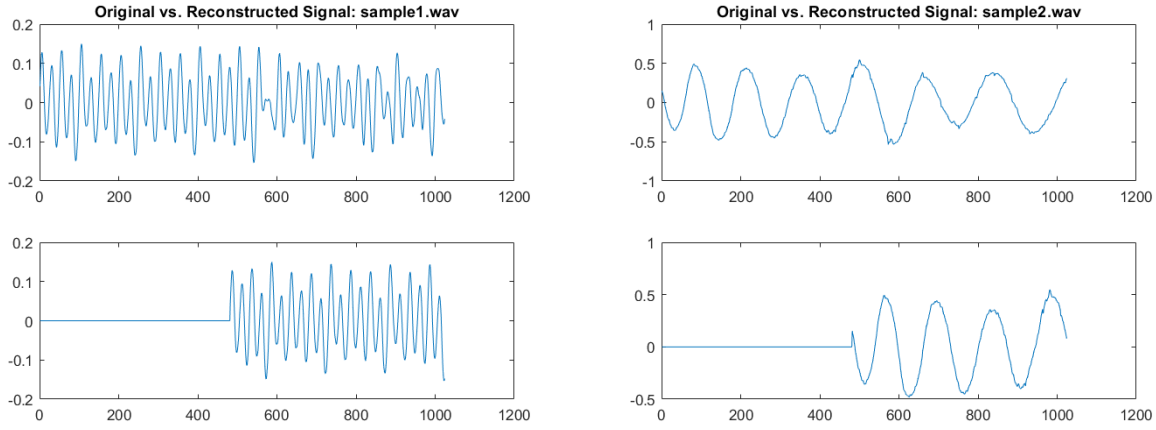
Assignment 3

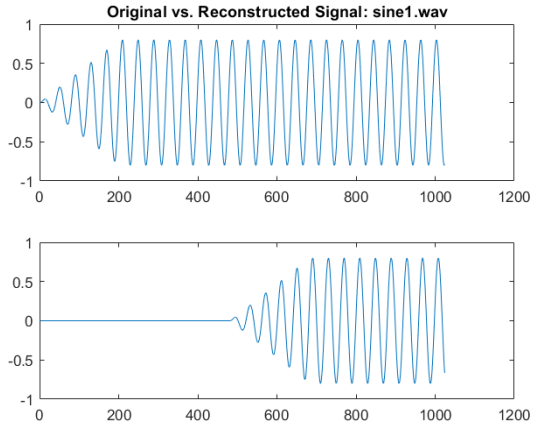
The synthesis portion of the codec involves taking the resulting signal after the compression and quantization and attempting to reconstruct the original signal as best as possible. The process is similar to the analysis stage, acting upon 32 sub-band samples per iteration. However, the nature of the reconstruction yields the use of a different cosine function for the filtering described in Assignment 1. The matrix holding the cosine values in this case is

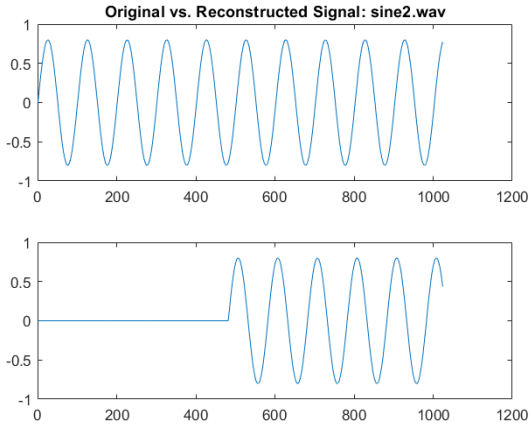
$64x32$ instead of $32x64$ as in the analysis stage. This results in the buffering of samples in this stage to occur with 64 sample chunks instead of 32, where each 64 samples correlates to a particular sub-bands. The samples are then passed through a reconstruction filter that is similar, but inverse to the windowing filter used in Assignment 1. The original signal is also re-inverted in terms of frequency, because the upsampling later at this stage re-inverts the frequency in the reconstruction due to the fact that the frequency domain will contract, removing the aliasing. This process is iterated over all of the groups of 32 samples, identical to Assignment 1. The result is a matrix(which is then converted to a vector for graphing) that holds the hopefully very similar signal. The MATLAB code for this can be found in the Code Appendix for Assignment 3/6.

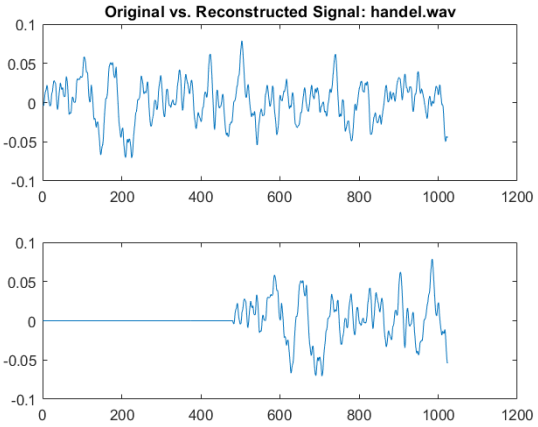
Assignment 4

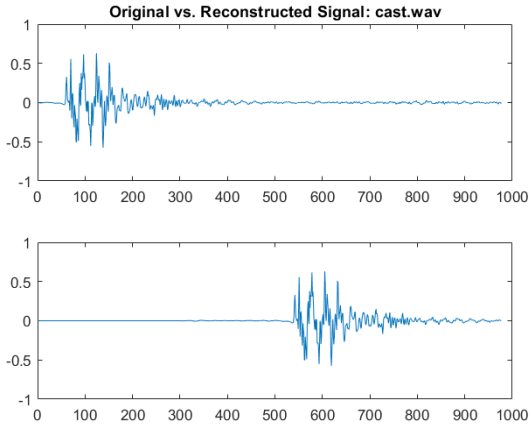
Upon reconstruction, the signal is almost resemblant of the original signal that was input into the system, however, there is one noticeable characteristic. All of the reconstructed signals are delayed by the same amount in time. This is due to the fact that the 512 length sample buffer takes a few iterations to fully fill up, so the beginning of the analysis is delayed. Initially, I thought this phenomenon was more complicated and related to the filter C being a linear phase filter that gave a constant group delay across all frequencies, but fortunately it turned out to be much simpler. Below are the plots for the reconstructed signals in comparison with the original signals that show the delay. The code for the plotting can be found in the Code Appendix for the Lab 4 Driver script:

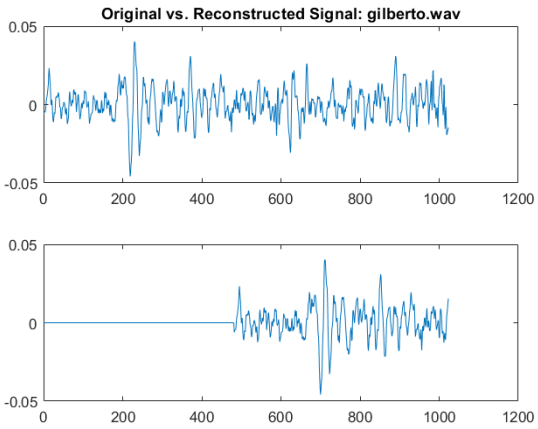



Original vs. Reconstructed Signal: sine1.wav


Original vs. Reconstructed Signal: sine2.wav


Original vs. Reconstructed Signal: handel.wav


Original vs. Reconstructed Signal: cast.wav


Original vs. Reconstructed Signal: gilberto.wav

Assignment 5

In order to calculate the error, the signals first had to be aligned correctly with each other to correct for the delay in the reconstruction and the few extra samples generated during the reconstruction. Then, in order to get a accurate error related to the concept at hand, the image processing toolbox from has functionality to calculate the mean-squared error between two signals with the function $immse(X, Y)$. This measures the deviations between the two signals, which is a good metric to have in calculating the error. Below is a table that shows the errors between the reconstructed signals and original signals: This shows that the

Tracks	Error
sample1.wav	2.27E-11
sample2.wav	1.85E-10
sine1.wav	1.94E-09
sine2.wav	3.58E-10
handel.wav	9.30E-12
cast.wav	3.02E-12
gilberto.wav	2.12E-12

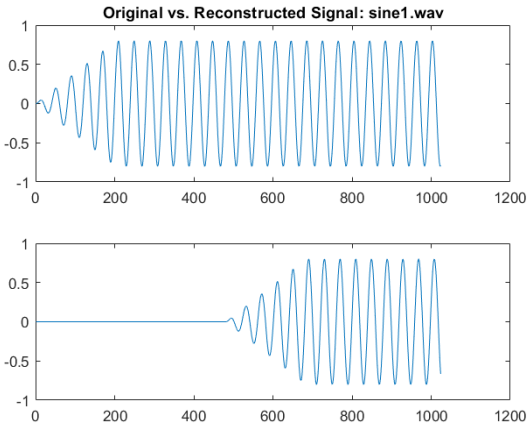
reconstructed signal is well within the tolerance across the entire song, due to the nature of mean-squared error, and that the reconstruction process was successful. The code for the error calculation can be found in the Code Appendix for Assignment 5.

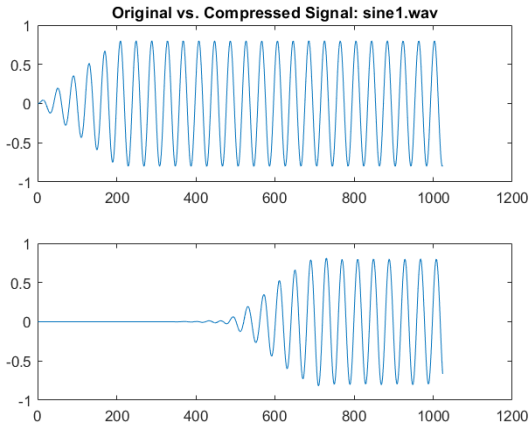
Assignment 6

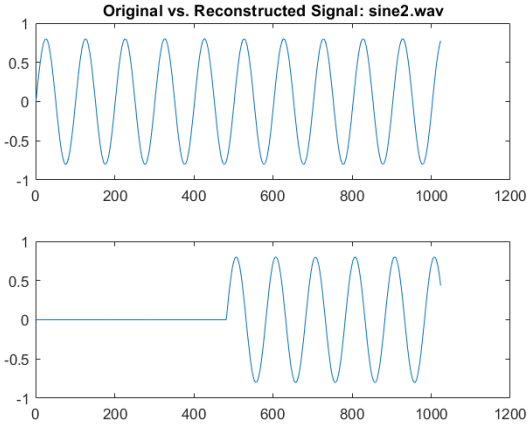
The final component of this lab was to add a very basic form of compression to the tracks immediately before the synthesis stage to show the effects of certain types of quantization. In this case, the compression scheme was simply including or excluding particular sub-bands frequencies from the reconstruction entirely, with the idea that the analysis stage would give information of which sub-bands are the most important, carrying the most energy. The compression was applied to each of the 7 tracks, but many sub-band exclusions were experimented with to see which frequencies were important in the track. The effect of the compression selected for each track is described by the compression ratio $\frac{usedbands}{32}$. The MATLAB code for the compression can be found in the Code Appendix for Assignment 3/6.

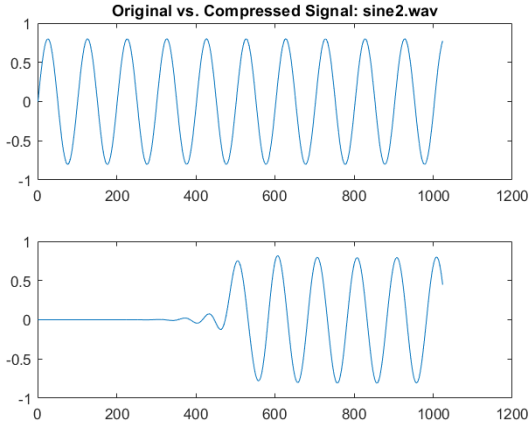
The first tracks that were tested in this experiment were the $sine1.wav$ and $sine2.wav$ tracks. The compression ratio in this case was very good, due to the fact that these tracks are only a single frequency, so only 1 sub-band is needed to get a near perfect reconstruction, with only a very slight, but acceptable audio difference with a little less sharpness. This makes the compression ratio $\frac{1}{32}$ which means only $\frac{1}{32}$ of the original file size is needed after the compression occurs for both sinusoid tracks. This is very good compression due to the reduction in size and the retention of audio integrity. For track $sine1.wav$ the sub-band used

was band 2 and for $sine2.wav$ the only band used was band 1. Below are the non-compressed reconstruction signals alongside the compressed reconstructions:



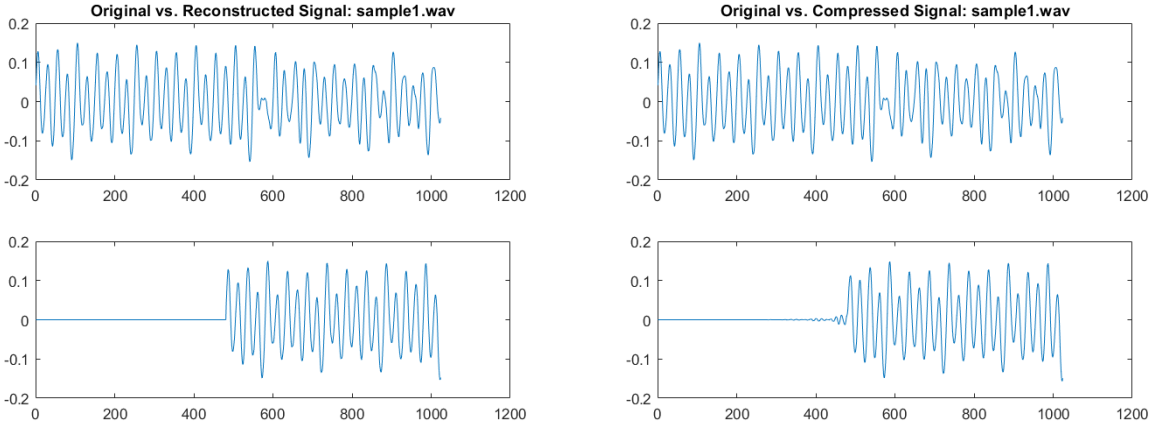






Graphically, it can be seen that the compressed signal only greatly differs from the non-compressed version in the slight ripple at the beginning of the signal, but aurally is very similar.

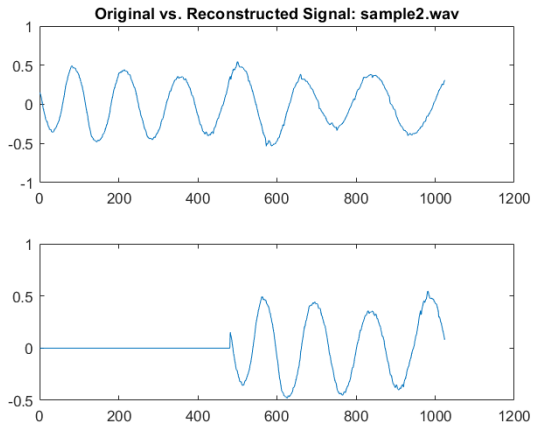
The next track that was used to experiment with compression is the piano piece in *sample*1.*wav*. This track is difficult to compress due to the more dynamic frequency range that is expressed in the piano. The first observation was that, as stated in Assignment 2, this track consists of evenly distributed low and mid frequencies, so compressing mid-high frequencies should give a reasonable result. In excluding bands 5,7,8,13, and 15:32, the result audibly is reasonable, with a slight dampening effect, but graphically, the signal is only slightly altered from the uncompressed version:

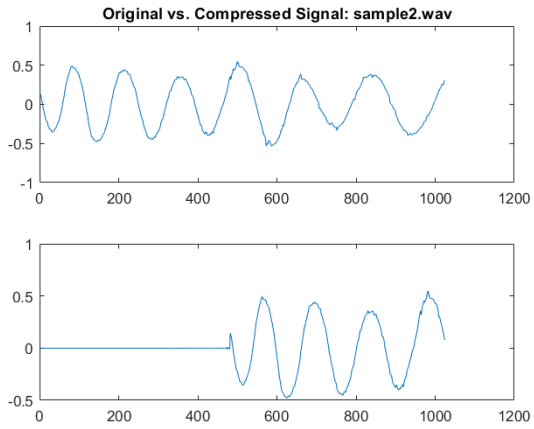


This results in a compression ratio of $\frac{9}{32}$ which for a more complex piece than the single frequency sinusoid, is good compression due to only $\frac{9}{32}$ the size needed to reconstruct the signal relatively well for the cost.

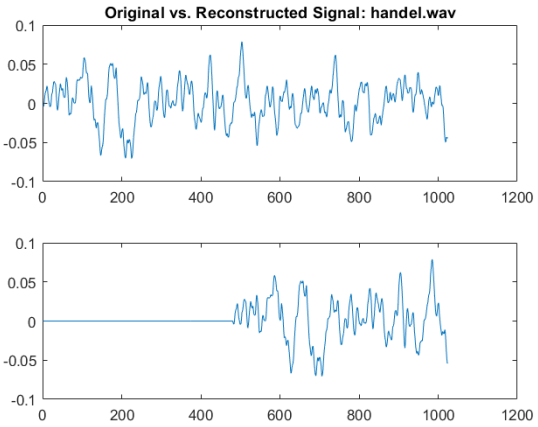
Next is the *sample*2.*wav* track. This is an electronic piece, and when looking at the coefficients from assignment 2, is can be seen that the signal primarily consists of low frequencies so compressing the high frequencies is a reasonable starting point. However, after experimenting with this track, I found it only acceptable to take away the some of the higher frequencies components because almost all of the low and mid frequencies were important to the track. When listening, taking out any of the mids or lows took away a characteristic of the track, whether it be the sound of the electronic bass drum or parts of the synthesizer lead, it damaged it too much for me to justify, so the compression ratio for this was much worse than the others, using all sub-bands but 26:29, at $\frac{29}{32}$. This shows that this is not the best compression technique for this particular style. Below is the comparison of the reconstruction and compressed signals:

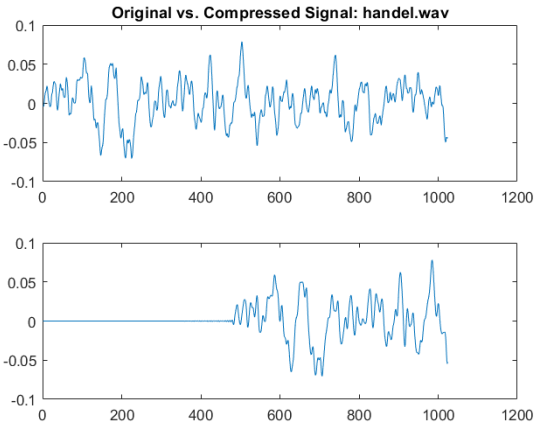
The next track is *handel.wav* which at first glance looks difficult to compress due to the coefficient distribution. After listening carefully to the effect of the compression, removing the majority of the high frequency bands had little to no aural effect on the track itself, due to the fact that the violins and choir are all within the low to mid frequency range, which





can also be verified by the coefficient plot from Assignment 2. The resulting compression is excluding bands 12:32, leading to a good compression ration of $\frac{11}{32}$ which reduced the size by almost $\frac{2}{3}$ which is very good given the small impact on sound quality shown graphically below:

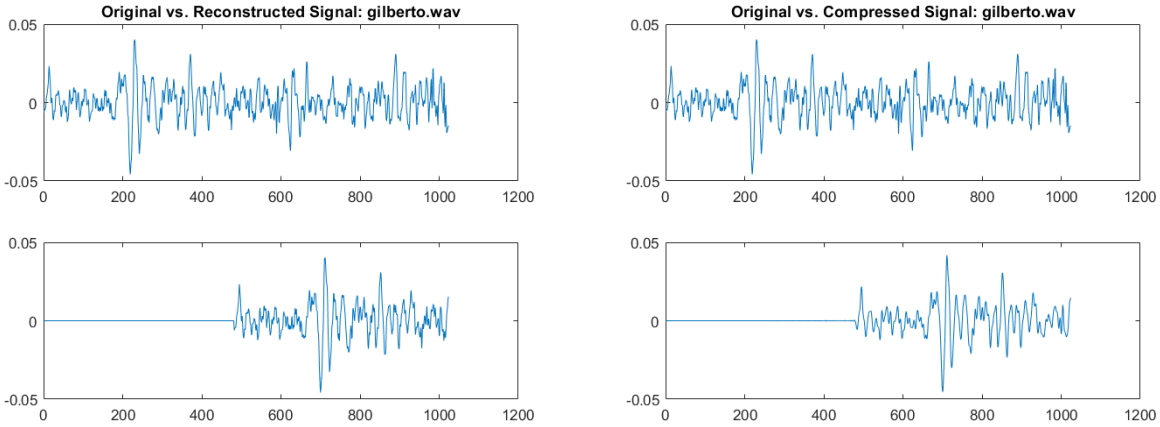




The track *cast.wav* in Assignment 2, gave a very interesting coefficient plot due to the unique timbre of the castanets. The dominant frequencies in the harmonic makeup of the instrument, however, lies in the low to mid frequencies, so the compression was able to exclude many of the high frequency bands with only a slight dampening effect to the signal, which sounds reasonably good. Bands 15:32 were excluded, resulting in a compression ratio of $\frac{14}{32}$ which considering the fact this is only one instrument, could be considered good or bad because a ensemble with this instrument would require less compression due to more frequencies, but the overall reduction in size is still more than half.

The final track used in the compression experimentation was *gilberto.wav*. From the coefficients plot, the frequencies primarily reside in the low to mid range, like in the castanet track, but perhaps with more room for compression. In this case, sub-bands 12:32 were removed with a reasonably good outcome, with only the higher harmonics of the drum getting removed if the audio is compared. This does not affect the overall feel of the track though

or distort the audio too badly. This results in a compression ratio of $\frac{11}{32}$ which is almost a $\frac{2}{3}$ reduction in size. This is good given the quality of the audio signal result. Below is the comparison between the reconstructed and compressed signals which is hardly different from a macroscopic perspective, besides the lack of some high frequencies:



The final conclusion that can be made from this experiment is that in general, low to mid frequencies are kept in the compression, due to the fact that the majority of the energy is carried within these frequencies. This is primarily due to the fact that instruments that are pleasant sounding in general do not carry extremely high frequencies (especially in more modern music which is what mp3 is primarily catered towards) and high frequencies represent noise as well. Also, in the previous steps it was mentioned that some of the sounds were dampened which is a result of removing higher frequencies and the resulting sharpness of transitions. In general, although this method is simple, a reasonable compression can still be obtained for the majority of the tracks.

Code Appendix

Lab 4 Driver Script

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Clint Olsen
% ECEN 4532: DSP Lab
% Lab 4 Driver Script
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Setup and Variable Declaration
%Load the Window Function from Prototype Filter
[C,D] = loadwindow();

%Generate Firse 5 Seconds on the Audio Track
filename = 'sample1.wav';
[track fs] = audioread(filename);
y = extract5Seconds(track,fs);

%Generate M Matrix once upon startup for analysis (32x64)
M = zeros(32,64);
for k=0:31
    for r=0:63
        M(k+1,r+1) = cos(((((2*k) + 1)*(r-16)*pi)/64));
    end
end

%Generate N Matrix once upon startup for synthesis (32x64)
N = zeros(64,32);
for k=0:63
    for r=0:31
        N(k+1,r+1) = cos(((((2*r) + 1)*(k+16)*pi)/64));
    end
end

%Create Buffer of size 32x(number of 32 sample frames in track)
x = buffer(y,32);

%% Assignment 1 and 2: PQMF Processing (Analysis Stage)
% Output Matrix for analysis stage
S = zeros(32,length(x));
[sArray S] = pqmf(x,C,M);

%Undo Frequency Inversion
uninvertMat = zeros(32,length(x));
[uninvertMat uninvertArray] = undoFreqInv(S);

%Plot the Result
plot(uninvertArray)
title(sprintf("Analysis Stage Coefficients: %s",filename));
xlabel('Sub-band coefficients: Low to High Frequency');

%% Assignment 3,4, and 6(with sub-band exclusion): IPQMF Processing (Synthesis Stage)
R = zeros(32,length(x));

% Bands to exclude in the output
thebands = []; % add which bands to exclude in this array ([2]) excludes 2
thebands = excludeBands(thebands);

%%%%%%%%%%Compression Happens Here Normally%%%%%%%%%%%%

%Undo Frequency Inversion again
[uninvertMat uninvertArray] = undoFreqInv(uninvertMat);

%Perform the reconstruction
[reconst R] = ipqmf(S,D,N,thebands);

%Plot the reconstructed signal vs. the original signal
hold on;
subplot(2,1,1);
plot(y(1:1024));
title(sprintf("Original vs. Reconstructed Signal: %s",filename));
subplot(2,1,2);
plot(reconst(1:1024));
hold off;


%% Assignment 5: Maximum Error Between Reconst. and Original
%maxerror = immse(reconst(482:end)',y(1:end-(481 - (length(reconst)-length(y)))));
maxerror = errorMax(reconst,y);
%Plot realigned signals
hold on;
plot(reconst(482:1505));
plot(y(1:1024));
hold off;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Clint Olsen
```

```
% ECEN 4532: DSP Lab
% Lab 4: Extract First Five Seconds
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function y = extract5Seconds(track,fs)
if length(track) >= fs*5
    y = track(1:fs*5);
else
    y = track;
end

end
```

Assignment 1

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Clint Olsen
% ECEN 4532: DSP Lab
% Lab 4 Assignment 1 and 2: PQMF
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [coefficients S] = pqmf(input,C,M)
    %Shift input to the filter bank
    X = zeros(1,512);
    % Z vector for intermediate computation
    Z = zeros(1,512);

    % Y vector for intermediate computation
    Y = zeros(1,64);

    % Output Matrix for analysis stage
    %coefficients = zeros(32,length(input));
    S = zeros(32,length(input));

    %Loop over each chunk of 32 audio samples in buffer
    for i=1:length(input)
        %Shift out last 32 samples
        X(33:512) = X(1:480);
        X(1:32) = zeros(1,32);

        %Add 32 new audio samples to X vector
        X(1:32) = fliplr(input(:,i)');

        %Prototype Filter multiplied by the input, x
        Z = C.*X;

        %Partial Calculation
        for r=0:63
            Y(r+1) = sum(Z(((r+1)+(64*(0:7)))));
        end

        %Calculate the 32 sub band samples
        for k=0:31
            S(k+1,i) = sum(M(k+1,(1:64)).*Y(1:64));
        end
    end
    coefficients = zeros(1,size(S,1)*size(S,2));
    coefficients = reshape(S',[1,size(S,1)*size(S,2)]);
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Clint Olsen
% ECEN 4532: DSP Lab
% Lab 4: Undo Frequency Inversion
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [uninvertMat, uninvertArray] = undoFreqInv(input)
    %Undo Frequency Inversion on even indices on even subands
    %This is a multiply by -1 (based on e^(i*pi*n) sequence)
    uninvert = ones(1,length(input));
    uninvertMat = zeros(32,length(input));

    %Every other value is -1
    uninvert(2:2:end) = -uninvert(2:2:end);
    for i=1:32
        if mod(i,2)==0
            uninvertMat(i,:) = input(i,:).*uninvert;
        else
            uninvertMat(i,:) = input(i,:);
        end

    end
    %Return the uninverted sub-band matrix
    uninvertArray = zeros(1,size(uninvertMat,1)*size(uninvertMat,2));
    uninvertArray = reshape(uninvertMat',[1,size(uninvertMat,1)*size(uninvertMat,2)]);
end
```

Assignment 3/6

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Clint Olsen
% ECEN 4532: DSP Lab
% Lab 4 Assignment 3: IPQMF
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [reconstruction S] = ipqmf(input,D,N,thebands)
    %Shift input to the reconstruction
    V = zeros(1,1024);
    % Z vector for intermediate computation
    U = zeros(1,512);

    % W vector windowing operation
    W = zeros(1,512);

    % Output Matrix for synthesis stage
    S = zeros(32,length(input));

    %Loop over each chunk of 32 audio samples in buffer
    for i=1:length(input)

        %Remove Subands from synthesis that are not desired
        input(:,i) = input(:,i).*thebands';

        %Shift out last 64 samples
        V(65:1024) = V(1:960);
        V(1:64) = zeros(1,64);

        %Add 64 samples to V vector
        for j=0:63
            V(j+1) = sum(N(j+1,(1:32)).*input(:,i)');
        end

        %Build the U Vector
        for j=0:7
            for k=1:32
                U((j*64)+k) = V((j*128)+k);
                U((j*64)+32+k) = V((j*128)+96+k);
            end
        end

        %Prototype Filter multiplied by the vector U
        W = D.*U;

        %Calculate the 32 reconstructed outputs
        for j=1:32
                S(j,i) = sum(W(j+(32*(0:15))));
        end
    end
    reconstruction = reshape(S,[1,size(S,1)*size(S,2)]);
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Clint Olsen
% ECEN 4532: DSP Lab
% Lab 4 Assignment 6: Zero out bands that should be excluded fro synthesis
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function bandArray = excludeBands(bands)
    bandArray = ones(1,32);
    for i = 1:length(bands)
        bandArray(bands(i)) = 0;
    end
end
```

Assignment 5

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Clint Olsen
% ECEN 4532: DSP Lab
% Lab 4 Assignment 5: Error Calcuation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function max = errorMax(reconst,y)
    %Calculate the error
    max = immse(reconst(482:end)',y(1:end-(481 - (length(reconst)-length(y)))));
end
```