Lab 4

Michael Eller

March 28, 2016

Contents

1 Introduction

During this lab, we will be investigating the implementation of Layer III of MPEGG 1, also known as mp3. We will fist develop several subband filters to decompose and reconstruct the original audio signal. We will be using a polyphase pseudo Quadrature Mirror Filter to deconstruct and eventually reconstruct the original audio signal.

2 Cosine Modulated Pseudo Quadrature Mirror Filter: Analysis

In this section, we will manipulate the equations that mathematically describe the analysis filter.

Consider the filter h_K, then the output of the combined filtering by h_k and decimation is given by

$$s_k[n] = \sum_{m=0}^{511} h_k[m]x[32n - m] \tag{1}$$

where

$$h_k[n] = p_n[n] \cos\left(\frac{(2k+1)(r-16)\pi}{64}\right) \quad k = 0, \dots, 31,\ n = 0, \dots, 511 \tag{2}$$

and p_0 is a prototype lowpass filter. The role of h_k is clear: the modulation of p_0 by $\cos\left(\frac{(2k+1)(r-16)\pi}{64}\right)$ shifts the lowpass filter around frequency $(2k+1)\pi/64$. Equation 1 requires $32 \times 512 = 16,384$ combined multiplications and additions to compute the 32 outputs $s_1, \dots, s_{32}$ for each block of 32 samples of the incoming signal. This is simply too slow to be properly effective.

We define:

$$c[n] = \begin{cases} -p_0[n] & \text{if } [n/64] \text{ is odd} \\ +p_0[n] & \text{otherwise} \end{cases} \tag{3}$$

then

$$h_k[64q + r] = c[64q + r] \cos\left(\frac{(2k+1)(r-16)\pi}{64}\right) \tag{4}$$

Using the notations of the standard, we further define

$$\boxed{M_{k,r} = \cos\left(\frac{(2k+1)(r-16)\pi}{64}\right), \quad k = 0, \dots, 31,\ r = 0, \dots, 63} \tag{5}$$

then

$$h_k[64q + r] = c[64q + r]M_{k,r} \tag{6}$$

and

$$s_k[n] = \sum_{r=0}^{63} \sum_{q=0}^{7} c[64q + r]M_{k,r}x[32n - 64q - r] \tag{7}$$

$$= \sum_{r=0}^{63} M_{k,r} \sum_{q=0}^{7} c[64q + r]x[32n - 64q - r] \tag{8}$$

In summary, for every integer $m = 32n$, multiple of 32, the convolution from equation 1 can be quickly computed using the following three steps,

$$
\begin{aligned}
z[64q + r] &= c[64q + r]x[m - 64q - r], && r = 0, \ldots, 63,\ q = 0, \ldots, 7 && (9)\\
y[r] &= \sum_{q=0}^{7} z[64q + r], && r = 0, \ldots, 63 && (10)\\
s[k] &= \sum_{r=0}^{63} M_{k,r}\, y[r], && k = 0, \ldots, 31 && (11)
\end{aligned}
$$

Even further speedup can be obtained by using a fast DCT algorithm to compute the matrix-vector multiplication in equation 11.

Assignment

1. Write the MATLAB `pqmf` that implements the analysis filter bank described in equations 5-9. The function will have the following template:

   `[coefficients] = pqmf (input)`

   where `input` is a buffer that contains an integer number of frames of audio data. The output array `coefficients` has the same size as the buffer `input`, and contains the subband coefficients.

   The array `coefficients` should be organized in the following manner:

   $$\texttt{coefficients} = [S_0[0]\ \ldots\ S_0[N_S - 1]\ \ldots S_{31}[0]\ \ldots\ S_{31}[N_S - 1]] \qquad (12)$$

   where $S_i[k]$ is the coefficient from subband $i = 0, \ldots, 31$ computed for the packet k of 32 audio samples. Also N_S is the total number of packets of 32 samples:

   $$N_S = \frac{\texttt{Samples}}{32} = 18 * \texttt{nFrames} \qquad (13)$$

   The organization of `coefficients` is such that the low frequencies come first, and then the next higher frequencies, and so on and so forth.

2. Analyse the first 5 seconds of the following tracks, and display the array `coefficients`,

   - sample1.wav, sample2.wav
   - sine1.wav, sine2.wav
   - handel.wav
   - cast.wav
   - gilberto.wav

   Comment on the visual content of the arrays `coefficients`.

The MATLAB code used to implement the `pqmf` function can be seen below in Listing 1.

Listing 1: pqmf.m

```
function [ coefficients ] = pqmf( inputBuffer,~ )
%PQMF Implements the analysis filter bank
%   Takes an input "inputBuffers" that contains an integer number of frames
%   of audio data. The output array "coefficients" has the same size as the
```

```
%   buffer "inputBuffers", and contains the subband coefficients.
filenameFlag=0;
if(ischar(inputBuffer)) % I lied, it's actually the filename
    filenameFlag=1;
    filename=inputBuffer;
    info=audioinfo(filename);
    SampleTime=5;
    if(SampleTime>info.Duration)
        SampleTime=info.Duration;
    end
    inputBuffer=audioread(filename,[1,(info.SampleRate*SampleTime)]);
    % Remove opening silence
    inputBuffer=inputBuffer(find(inputBuffer~=0,1):end);
end

totalSamples=length(inputBuffer);
frameSize=576;
nFrame=floor(totalSamples/frameSize);
inputBuffer=inputBuffer(1:(nFrame*frameSize));
[C,~] = loadwindow(); %C is the analysis window
%D is the synthesis window, but is not needed

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
Ns=18*nFrame;
bufferSize=512;
y=zeros(1,64);
S=zeros(32,1);

coefficients=zeros(size(inputBuffer));
packet=1; % counter for inner loop
for frame = 1:nFrame          % chunk the audio into blocks of 576 samples
    offset = (frame -1)*frameSize+1;  % absolute address of the frame
    frameTemp=inputBuffer(offset:(offset+frameSize-1));
    Buffer=zeros(size(C));
    for index = 1:18                % 18 non overlapping blocks of size 32
        Buffer(1:bufferSize-32)=Buffer(33:end);
        newBlock=frameTemp(((index-1)*32+1):index*32); % 32 new samples
        Buffer((bufferSize-31):end)=newBlock;
        % process a block of 32 new input samples
        % see flow chart in Fig. 2
        Z=C.*Buffer; % Window by 512 Coefficients to produce vector

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

        for i=0:31
            S(i+1)=sum(M(i+1,:).*y); % Calculate 32 samples by matrixing
        end

        % Frequency inversion
        if(mod(index,2)==1)
            channel=1:2:32;
            S(channel)=-S(channel); % invert odd-numbered frequencies
        end
        % Spaced Ns apart
        coefficients(packet+(Ns*(0:31)))=S; % Assign coefficients

        packet=packet+1;
    end % end index=1:18
end % end frame=1:nFrame

coefficients=coefficients/max(coefficients); % Normalize
```

```
if(nargin==2 && filenameFlag)
    h=figure;
    [~,name,ext]=fileparts(filename);
    plot(coefficients);
    title([name, ext,' pqmf ', num2str(SampleTime) ' seconds']);
    xlabel('Coefficient');
    ylabel('Amplitude');
    saveas(gca,[name,'_',num2str(SampleTime),'sec.png']);
    close(h);
end
end % end function
```

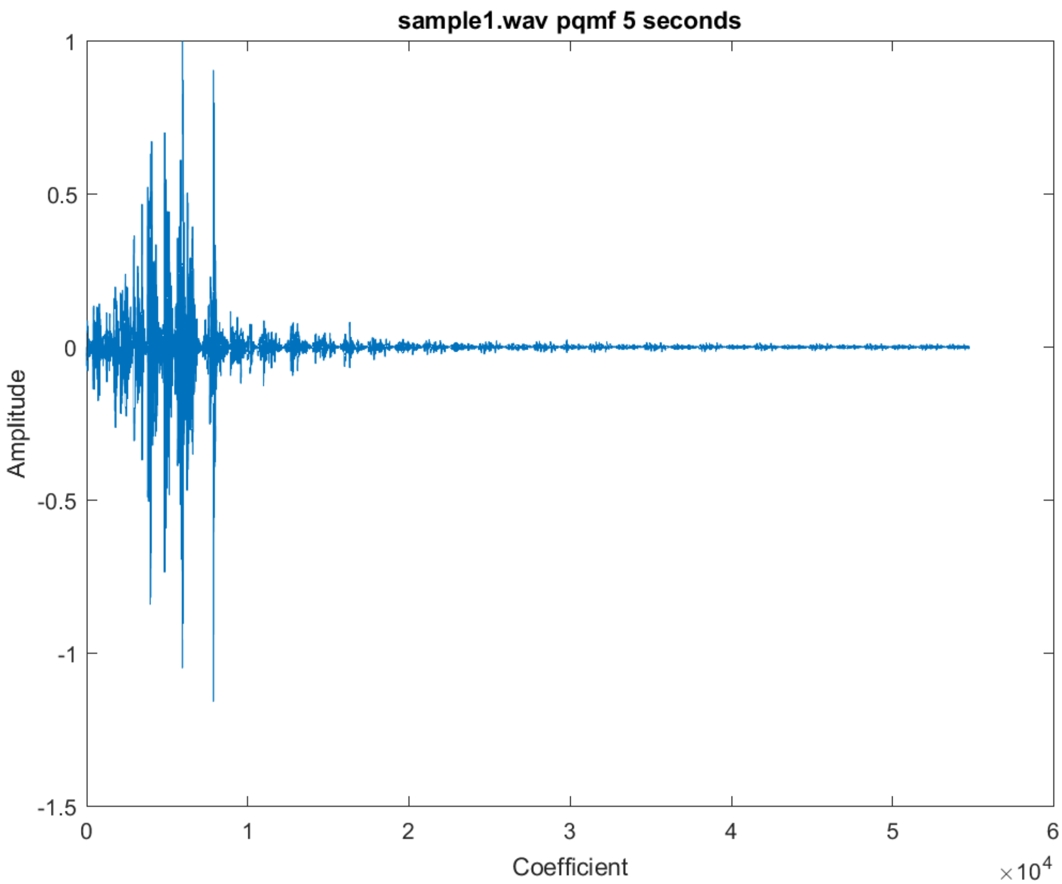


Figure 1: PQFM: Sample1 5 seconds

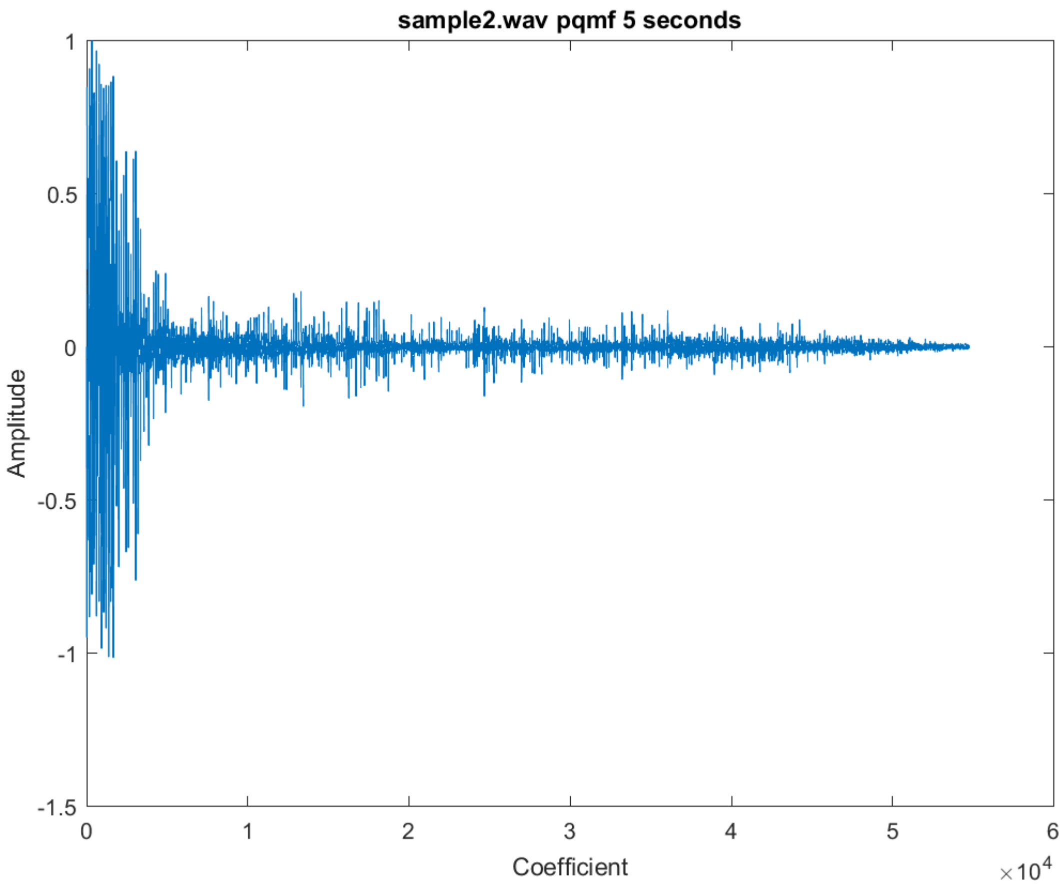
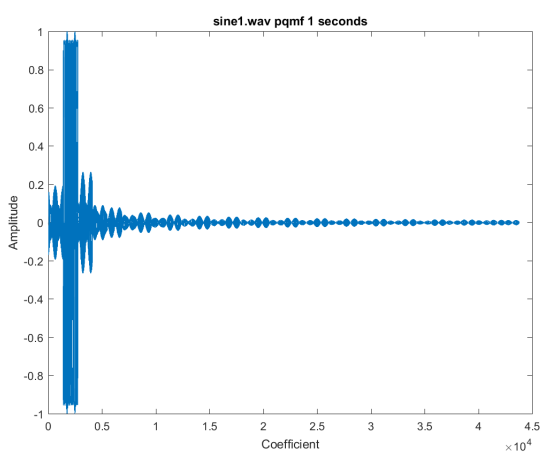
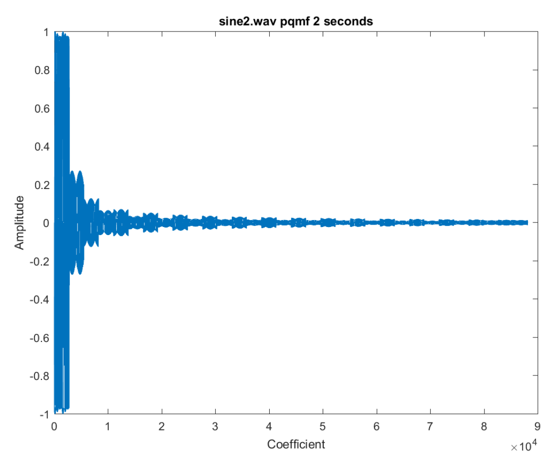


Figure 2: PQFM: Sample2 5 seconds



(a) PQFM: Sine1



(b) PQFM: Sine2

Figure 3: Sine Waves

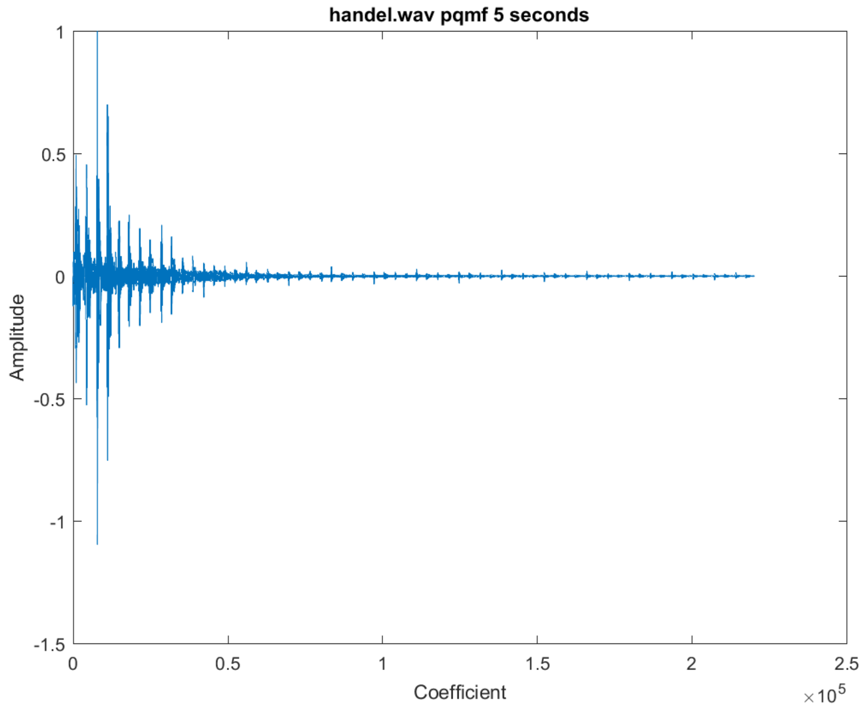


Figure 4: PQFM: Handel 5 seconds

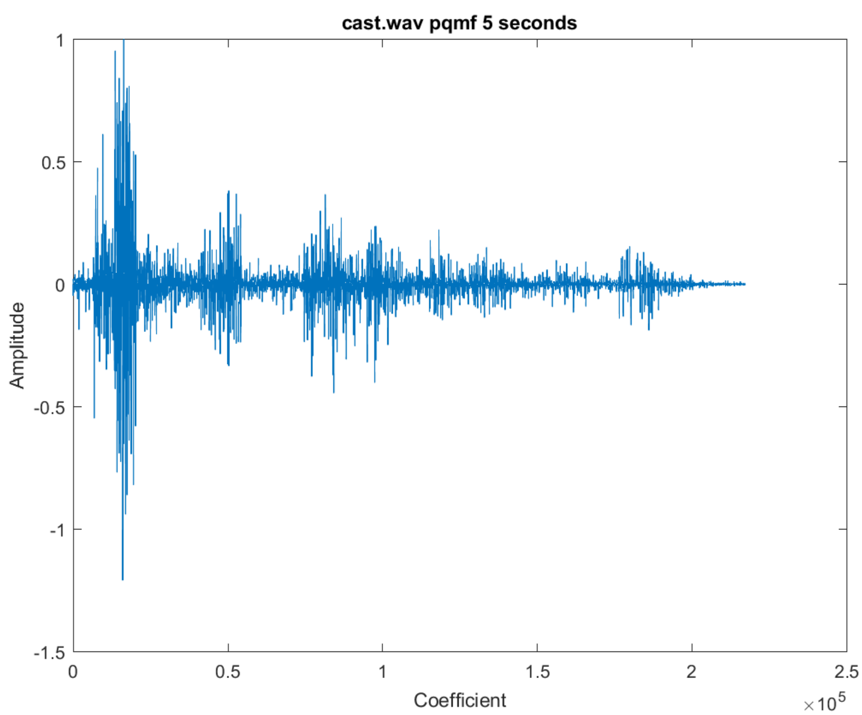


Figure 5: PQFM: Castanets 5 seconds

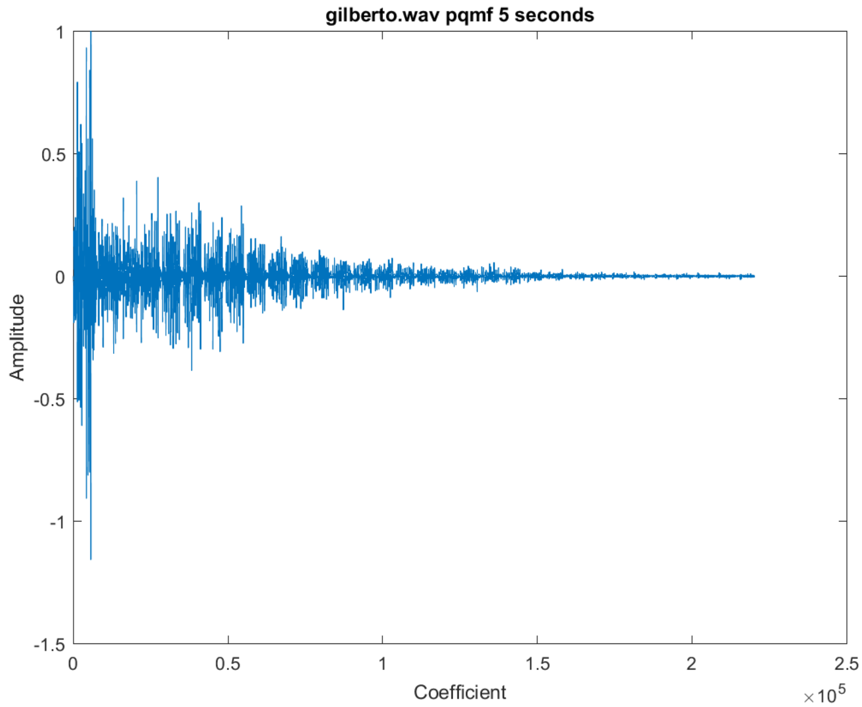


Figure 6: PQFM: Gilberto 5 seconds

The PQFM coefficients essentially show an outline of the frequency spectrum of the various songs.

As seen in Figure 1, the extreme lower end of the PQFM coefficients are low, while they get larger as you approach coefficient number 5000. If one listens to the song, it can be easily heard how the piano lacks a substantial bass, but the higher notes can be heard more easily.

Looking at Figure 2, the lower end of the spectrum is much more substantial, as can be heard by the thumping bass in the song. Since the music is mostly electronic, it lacks many of the stronger overtones that are commonly found on natural classical instruments. As you go even higher in the spectrum, the coefficients do not approach zero as quickly as was the case in Figure 1. Perhaps this is due to the inevitable higher-order harmonics that result from electronic music.

Figures 3a and 3b show the most discernible distinctions. The first sine wave is obviously lower than the second because of the larger concentration around the lower end of the spectrum.

Figure 4 is about the same as Figure 1. There is more activity on the lower end of the spectrum though. This is probably due to the fact that "handel.wav" includes vocals as well as piano and violin.

Figure 5 is quite interesting. While the previous figures had a fairly consistent decline as the coefficients increased, this trend seems fairly haphazard. Something of considerable note though is that this is the first plot to have neighbouring coefficients of unequal magnitude. While the other plots were extremely symmetrical about the 0 amplitude point, Figure 5 is not. Perhaps this is due to the highly percussive nature of the castanets and its atonal sounds.

In Figure 6, you see a combination of Figures 4 and 5. In place of the castanets, we have the Brazilian tamborim. While it has a tonal center, when the tamborim is struck in rapid succession it produces a much noisier sound that lacks a discernible tonal center. The applause at the beginning of the song also adds some noisiness that can be seen in Figure 6 by the large spikes at seemingly random intervals.

3 Cosine Modulated Pseudo Quadrature Mirror Filter: Synthesis

The synthesis, or reconstruction, from the coefficients is performed in a very similar manner. The following equations yield the reconstruction of 32 audio samples from 32 subband coefficients

$$\texttt{for } i = 1023 \texttt{ down to } 64 \texttt{ do} \quad v[i] = v[i-64] \tag{14}$$

$$\texttt{for } i = 63 \texttt{ down to } 0 \texttt{ do} \quad v[i] = \sum_{k=0}^{31} N_{i,k}\, s[k] \tag{15}$$

for i =0 to 7 do
  for $j = 0$ to 31 do

$$u[64i + j] = v[128i + j] \tag{16}$$

$$u[64i + j + 32] = v[128i + j + 96] \tag{17}$$

$$\texttt{for } i = 0 \texttt{ to } 511 \texttt{ do} \quad w[i] = d[i]u[i] \tag{18}$$

$$\texttt{for } j = 0 \texttt{ to } 31 \texttt{ do} \quad x[j] = \sum_{j=0}^{15} w[j + 32i] \tag{19}$$

where

$$\boxed{N_{i,k} = \cos\left(\frac{(2k+1)(16+i)\pi}{64}\right), \quad i = 0, \dots, 63,\ k = 0, \dots, 31} \tag{20}$$

Assignment

3. Write the MATLAB function `ipqmf` that implements the synthesis filter bank described in equations 15-20. The function will have the following template:

   ```
   [recons] = ipqmf (coefficients)
   ```

   where `coefficients` is a buffer that contains the coefficients computed by `pqmf`. The output array `recons` has the same size as the buffer `coefficients`, and contains the reconstructed audio data.

4. Reconstruct the first 5 seconds of the following tracks, and display the signal `input`. You should observe that the reconstructed signal is slightly delayed. This is due to the fact that the processing assumes that a buffer of 512 audio samples is immediately available.

   - sample1.wav, sample2.wav
   - sine1.wav, sine2.wav
   - handle.wav
   - gilberto.wav

5. Compute the maximum error between the reconstructed signal and the original, taking into account the delay. The error should be no more than 10^{-5}. Explain how you estimate the delay.

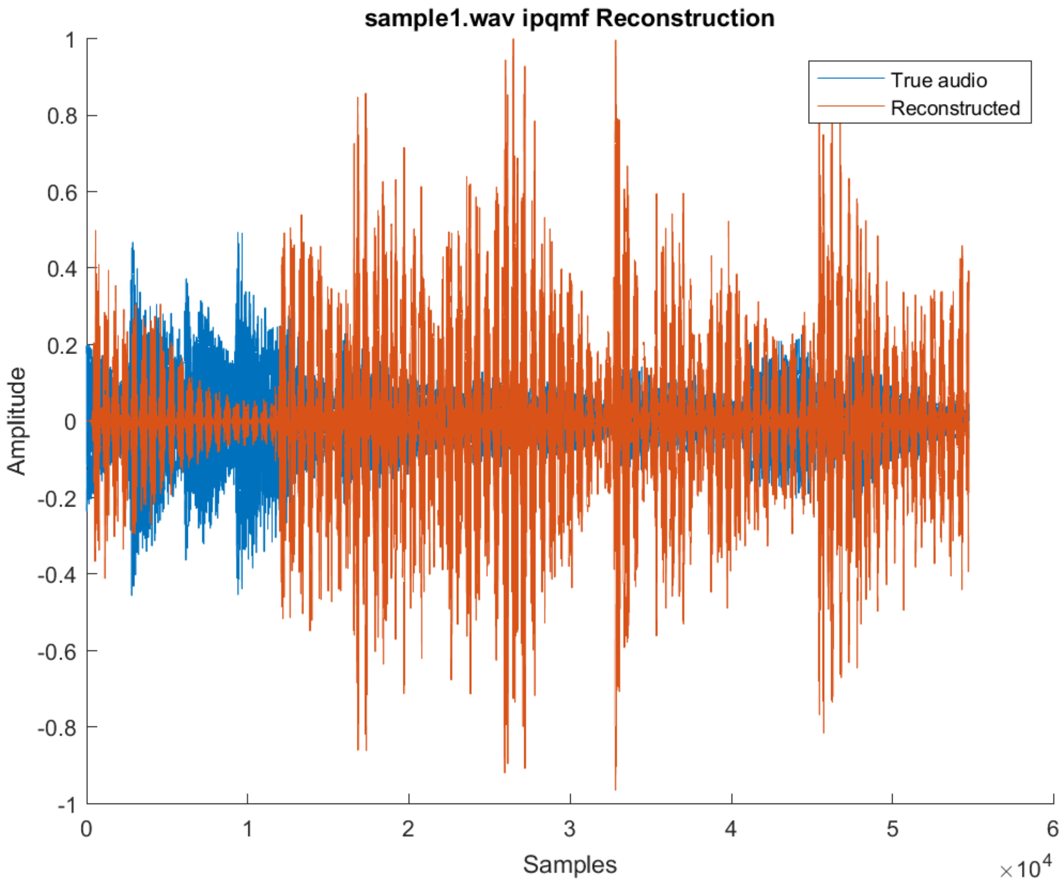


Figure 7: PQMF: Sample1 IPQMF

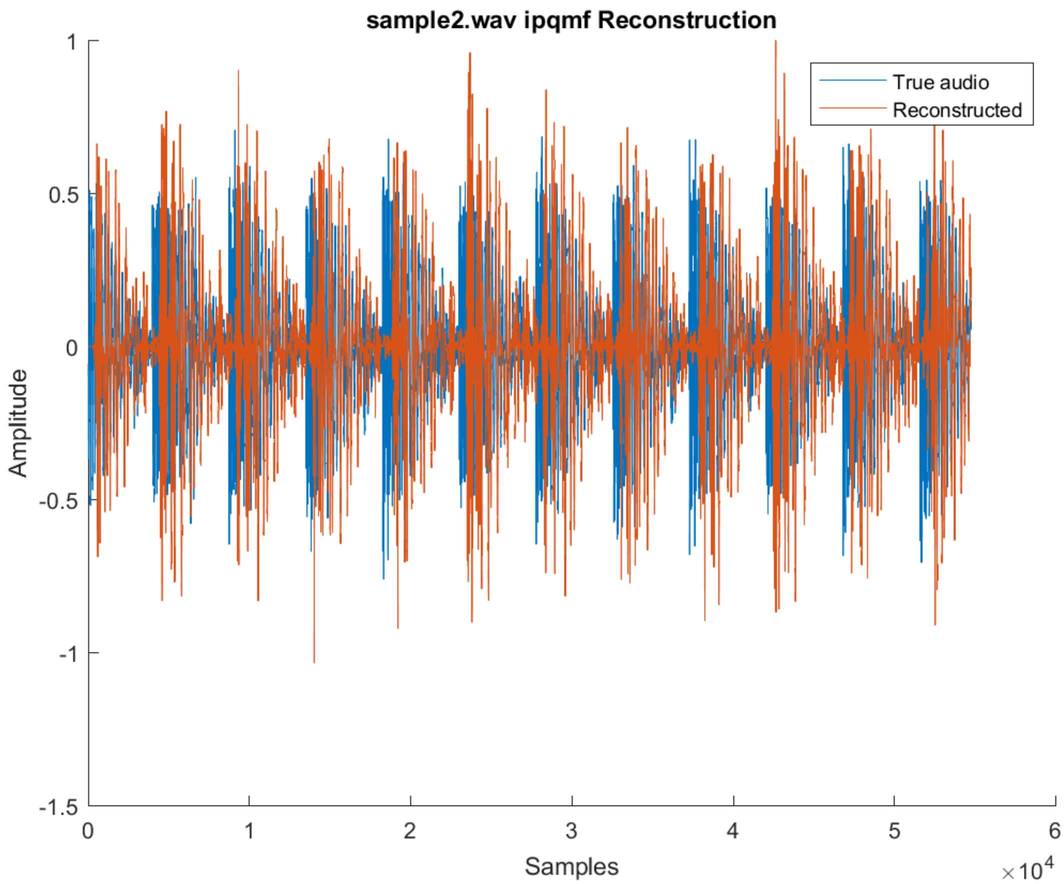
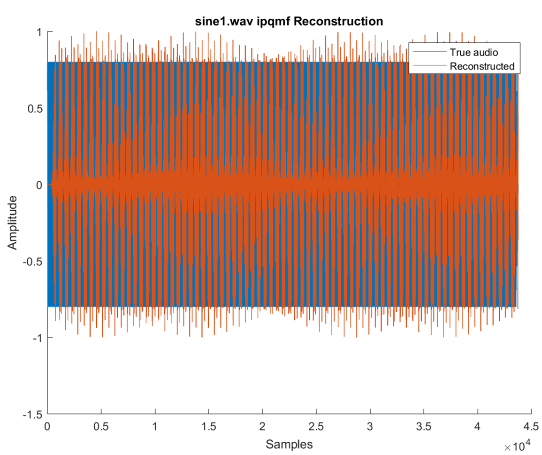
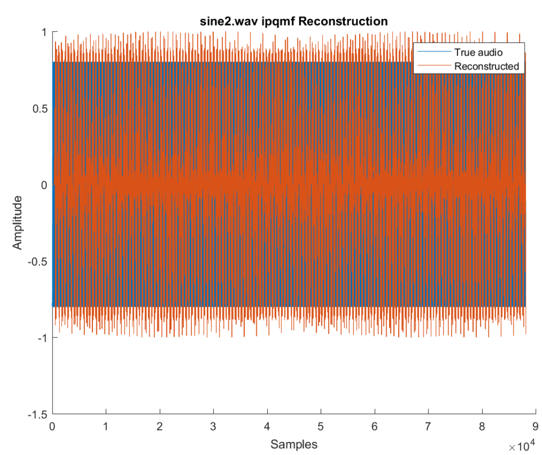


Figure 8: PQMF: Sample2 IPQMF



(a) IPQMF: Sine1



(b) IPQMF: Sine2

Figure 9: Sine Waves

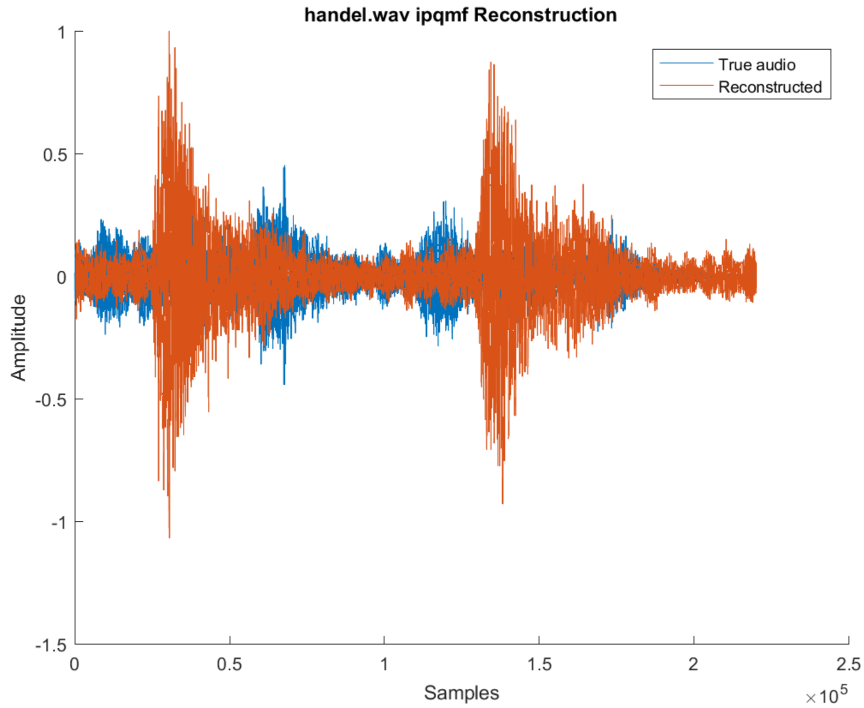


Figure 10: IPQMF: Handel 5 seconds

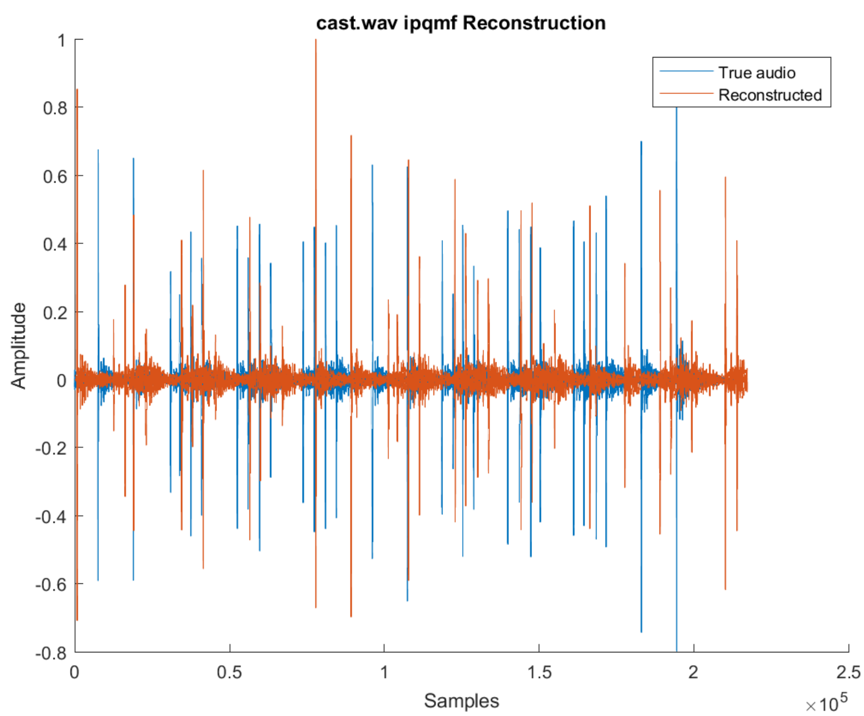


Figure 11: PQMF: Castanets 5 seconds

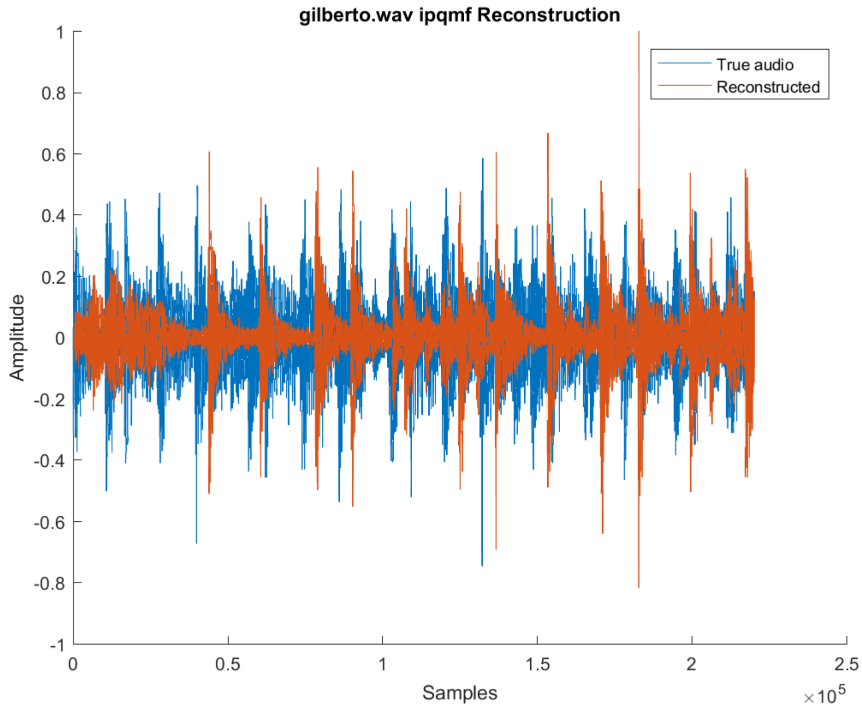


Figure 12: PQMF: Gilberto 5 seconds

As can be seen cleary in figures 9-12, there are some obvious problems with the reconstruction, but nonetheless the general form of the reconstruction follows the original audio. After a careful analysis of the issue, it could be seen that there are very slight differences in the PQMF plot(s). While I originally tried a very systematic manner for computing the delay between the original signal and the reconstruction, I eventually found it to be just as effective to view the delay visually from a plot showing the original and the reconstruction. Reading through “Pan95-mega.pdf” reveals that *The net offset is 320 points to time-align the psychoacoustic model data with the filter bank outputs.* This number doesn't produce a good result though, so I stick with my experimental value instead.

Listing 2: ipqmf.m

```
function [ recons,difference ] = ipqmf( coefficients,thebands,filename,~)
%ipqmf Reconstructs the audio data
%   "coefficients" is a buffer that contains the coefficients as computed
%   by pqmf. The output array "recons" has the same size as the buffer
%   "coefficients", and contains the reconstructed audio data.

%% Validate input
narginchk(1,4);
nargoutchk(0,2);
bandFlag=1;
if ischar(thebands) % thebands was probably "filename"
    filename=thebands;
    bandFlag=0;
    thebands=zeros(1,32);thebands(:)=1;
end
if(((nargout==2) && (~((nargin==4)||(nargin==3)))...
        ||  ((nargout~=2) && ((nargin==4)||((nargin==3)&&(~bandFlag))))))
    error('Two outputs requires three or four inputs');
end

if ischar(thebands) % thebands was probably "filename"
```

```matlab
    filename=thebands;
    bandFlag=0;
    thebands=zeros(1,32);thebands(:)=1;
end
frameSize=576;
if(mod(length(coefficients),frameSize)~=0)
    error('ipqmf:invalidInputSize',...
        ['Invalid size for ''coefficients''.'...
        ' Length must be multiple of %d.'],frameSize);
end
%% Setup
audioSampleSize=32;
processingSize=64;
bufferSize=1024;
Ns=length(coefficients)/audioSampleSize;
% Because multiple of 576, Ns will be an integer
N=zeros(processingSize,audioSampleSize);
for i=0:processingSize-1
    for k=0:audioSampleSize-1
        N(i+1,k+1)=cos(((2*k+1)*(16+i)*pi)/64);
    end
end

coefficients=buffer(coefficients,Ns); % Matrix is easier
[~,D] = loadwindow(); %C is the analysis window, but is not needed
%D is the synthesis window

%% Init vars
recons=zeros(audioSampleSize,Ns);
Buffer=zeros(bufferSize,1);

%% Do the magic
for packet = 1:Ns
    U=zeros(size(D));
    S=coefficients(packet,:).*thebands; % extract subband
    if (mod(packet,2) == 1) % Act on every other packet
        channel = 1:2:32; % Invert every other coefficent
        S(channel) = -S(channel); % Invert coefficent
    end
    %% Shift Buffer
    Buffer((processingSize+1):bufferSize)=...
        Buffer(1:(bufferSize-processingSize));
    for i=0:processingSize-1
        Buffer(i+1) = sum(N(i+1,:).*S);
    end
    %% DSP Magic
    j=0:(audioSampleSize-1);
    for i=0:7
        U(i*64+j+1) = Buffer(i*128+j+1); % DSP magic
        U(i*64+32+j+1) = Buffer(i*128+96+j+1); % DSP magic
    end
    W=U.*D; % Windows by 512 coefficients

    for j=0:audioSampleSize-1 % Calculate 32 samples
        recons(j+1,packet) =(sum(W((j+1) + audioSampleSize*(0:15))));
    end
end
% Output 32 reconstructed PCM samples
recons=recons(:); % Change back to vector
recons=recons/max(recons); % Normalize it

%% Plot the magic
if(nargin > 1)
    h=figure;
    hold on;
    audio=audioread(filename);
    audio=audio((length(audio)-length(recons)):end);
    plot(audio(1:length(recons)));
```

```
    xlabel('Samples');
    ylabel('Amplitude');
    [~,name,ext]=fileparts(filename);
    plot(recons);
    title([name, ext, ' ipqmf Reconstruction']);
    legend('True audio','Reconstructed');
    saveas(h,[name,'_ipqmf.png']);
    close(h);

    if((nargin == 4)|| ((nargin==3)&& (bandFlag==0)))
        offset1=489;
        h=figure;
        hold on;
        plot(audio);
        xlabel('Samples');
        ylabel('Amplitude');
        plot(recons(offset1:end));

        title({[name, ext, ' ipqmf Reconstruction'],'Delay Fixed'});
        legend('True audio','Reconstructed (Fixed for delay)');
        saveas(h,[name,'_D_ipqmf.png']);

        close(h);

        h=figure;
        hold on;
        plot(audio(1:1024));
        plot(recons((1:1024)+offset1));
        xlabel('Samples');
        ylabel('Amplitude');
        legend('True audio','Reconstructed (Fixed for delay)');
        if(bandFlag)
            title({[name, ext, ' ipqmf Reconstruction (1024 samples)'],...
                'Delay Fixed', 'Specialized Subbands'});
            saveas(h,[name,'_DS1024_ipqmf.png']);
        else
            title({[name, ext, ' ipqmf Reconstruction (1024 samples)'],...
                'Delay Fixed'});
            saveas(h,[name,'_D1024_ipqmf.png']);
        end
        close(h);

        difference=sum(abs(audio(1:1024)-recons((1:1024)+offset1)));
    end
end
end
```

To compute the error between the original audio and the reconstruction, I simply added a slight bit on the end of my **buildReport.m** file as can be seen in Listing 3.

Listing 3: buildReport.m

```
songNames='';
for i=1:len
    [~,name,~]=fileparts(tracks{i});
    songNames=[songNames ' ' name];
end
songNames(1)=[]; % Remove extra space

h=figure;
bar(totalError);
title('Total Error');
xlabel('Track');
ylabel('Error Difference');
ax=gca;
axis([-inf inf 0 max(totalError)*1.025]);
ax.XTickLabel=strsplit(songNames);
```

```
saveas(h,'totalError.png');
close(h);

h=figure;
bar(totalError1);
title({'Total Error','Specialized Subbands'});
xlabel('Track');
ylabel('Error Difference');
ax=gca;
axis([-inf inf 0 max(totalError)*1.025]);
ax.XTickLabel=strsplit(songNames);
saveas(h,'totalError1.png');
close(h);
```

6. Modify your code to reconstruct an audio signal using only a subset of bands. This is the beginning of compression. Your function prototype should look like this:

   `[recons] = ipqmf (coefficients, thebands)`

   where **thebands** is an array of 32 integers, such that `thebands[i] =1` if band i is used in the reconstruction, and `thebands[i] =0` if band i is not used.

   Experiment with the files

   - sample1.wav, sample2.wav
   - sine1.wav, sine2.wav
   - handel.wav
   - cast.wav
   - gilberto.wav

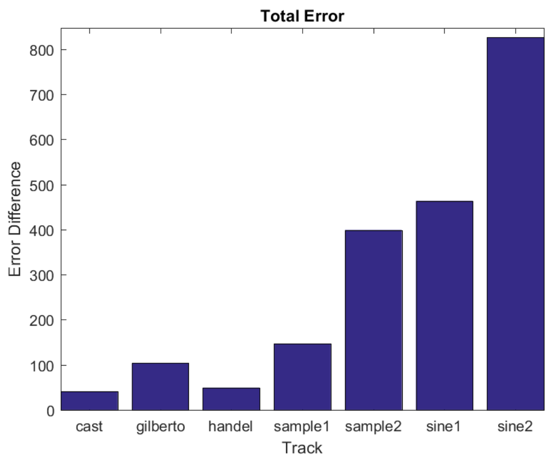
   and describe the outcome of the experiments, when the certain bands are not used to reconstruct. If you describe the compression ratio as

   $$\frac{\texttt{number of bands used to reconstruct}}{32} \tag{21}$$

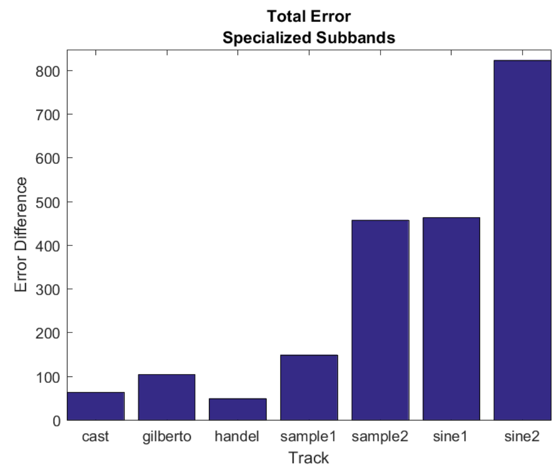
   explain what is a good compression ratio, and a good choice of bands for each audio sample. Note that the psychoacoustic model of MP3 performs this task automatically.

After modifying the `ipqmf` function, I used bands 1 through 7 and computed the reconstruction. Surprisingly, as can be seen in Figure 13, the error is actually decreased. So far, this only means that my reconstruction is so bad that no results are better than what I have. Further improvements will lead to a more expected result. An expected result would be that the error increases as less bands are used. Right now, I have a compression rate of 6/32 but that doesn't matter until my output is actually correct.

The psychoacoustic model is designed so that frequencies that have lower bark values (similar to the mel values computed in Lab 1) do not need to be included in the subband calculation.



(a) Total Error



(b) Total Error (specialized subbands)

Figure 13: Total Errors

A Figures

A.1 PQMF Models

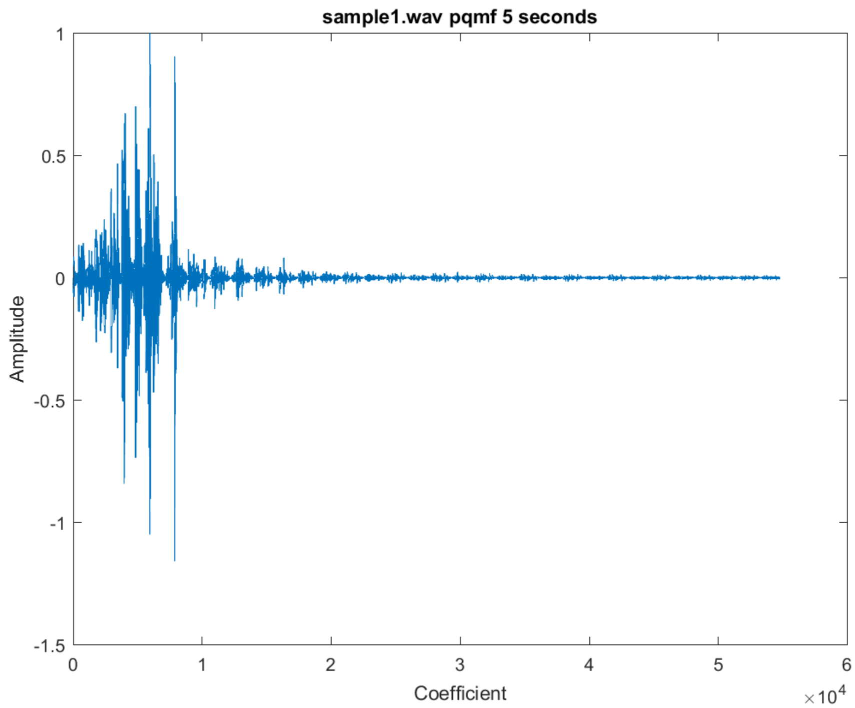


Figure 14: PQFM: Sample1 5 seconds

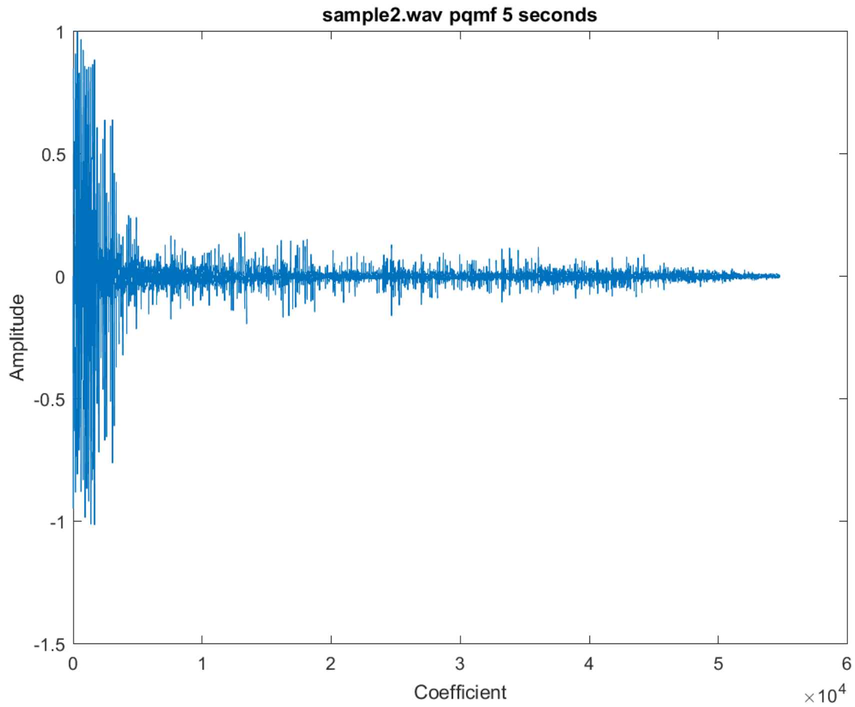
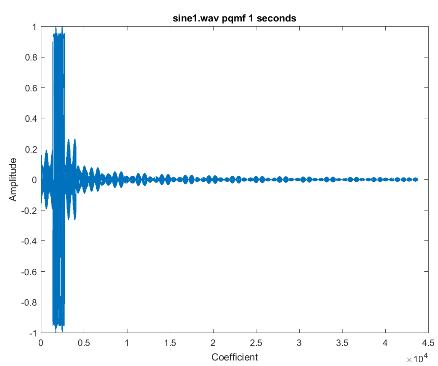
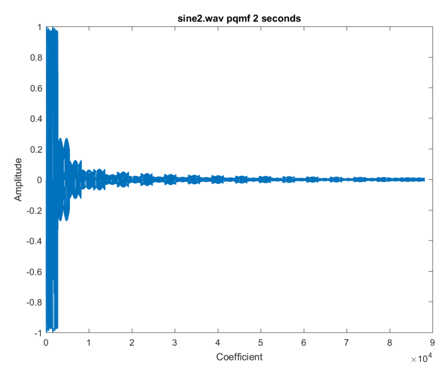


Figure 15: PQFM: Sample2 5 seconds



(a) PQFM: Sine1



(b) PQFM: Sine2

Figure 16: Sine Waves

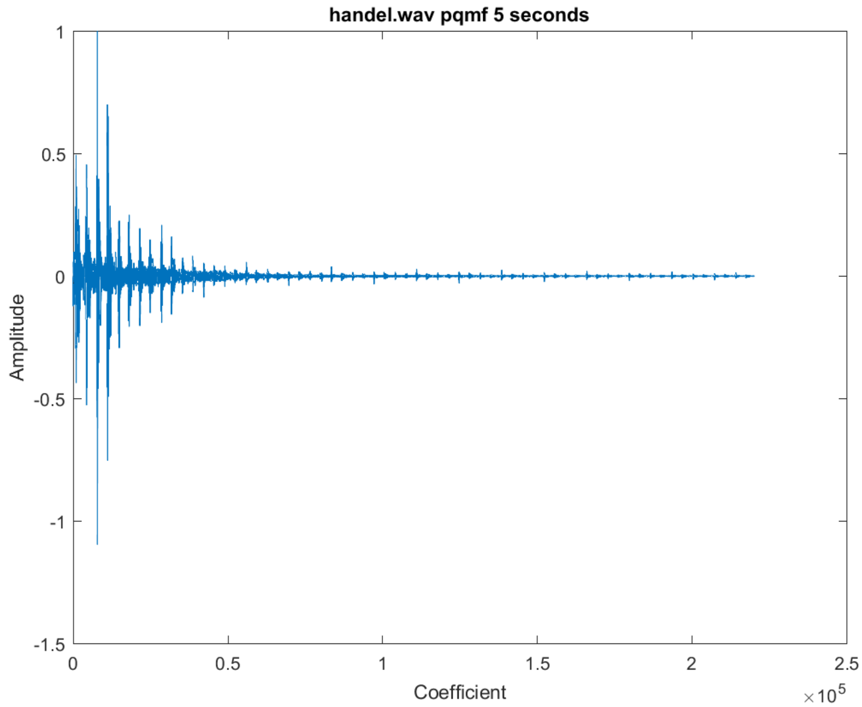


Figure 17: PQFM: Handel 5 seconds

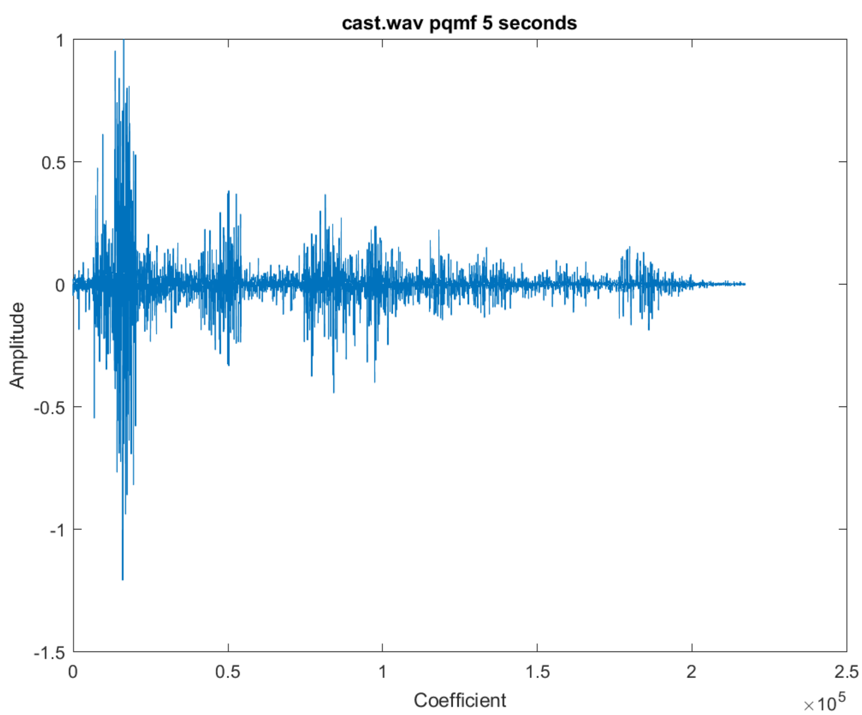


Figure 18: PQFM: Castanets 5 seconds

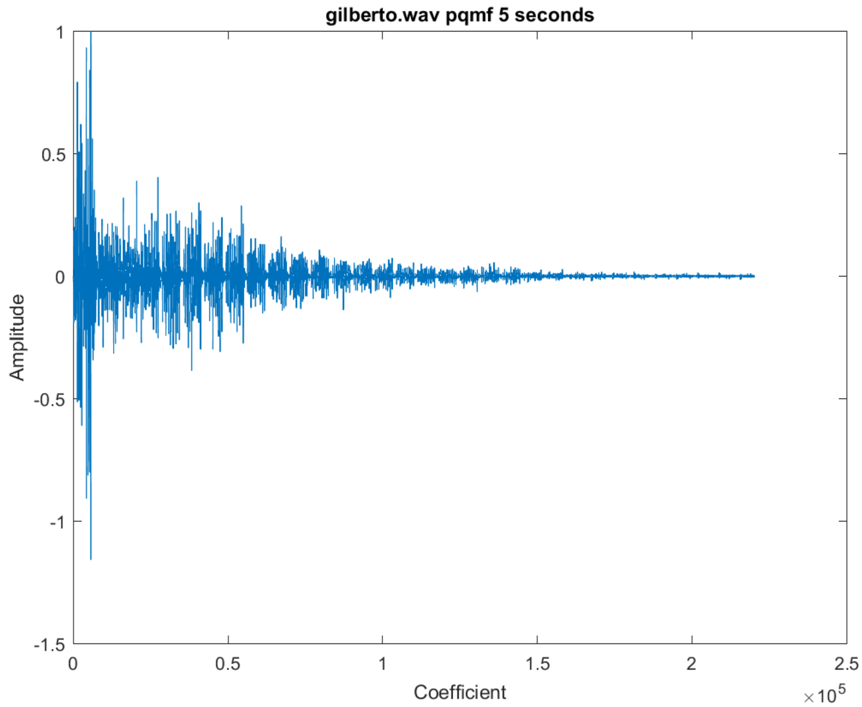


Figure 19: PQFM: Gilberto 5 seconds

A.2 IPQMF Models

A.2.1 Normal

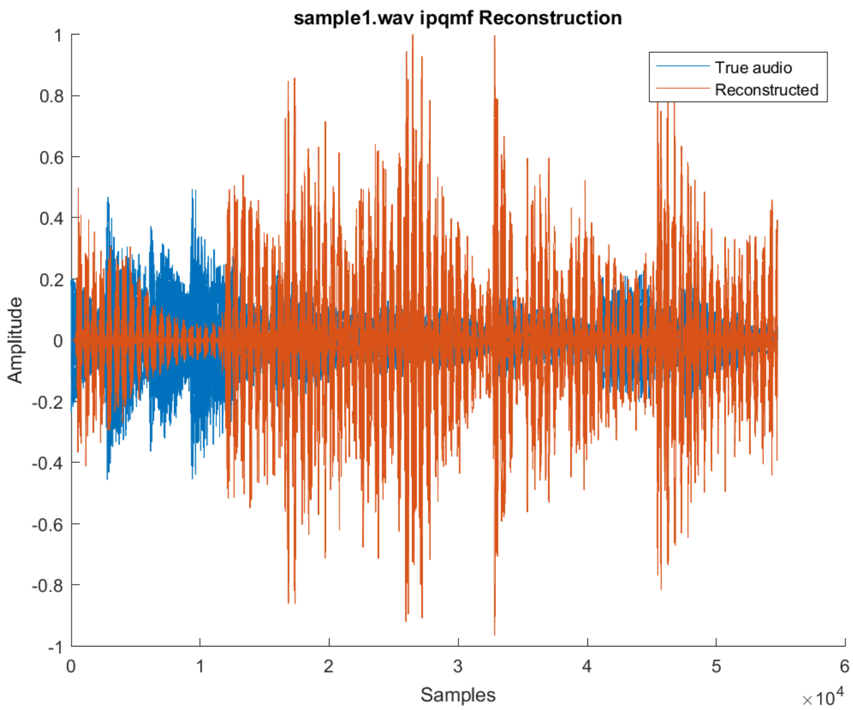


Figure 20: PQMF: Sample1 IPQMF

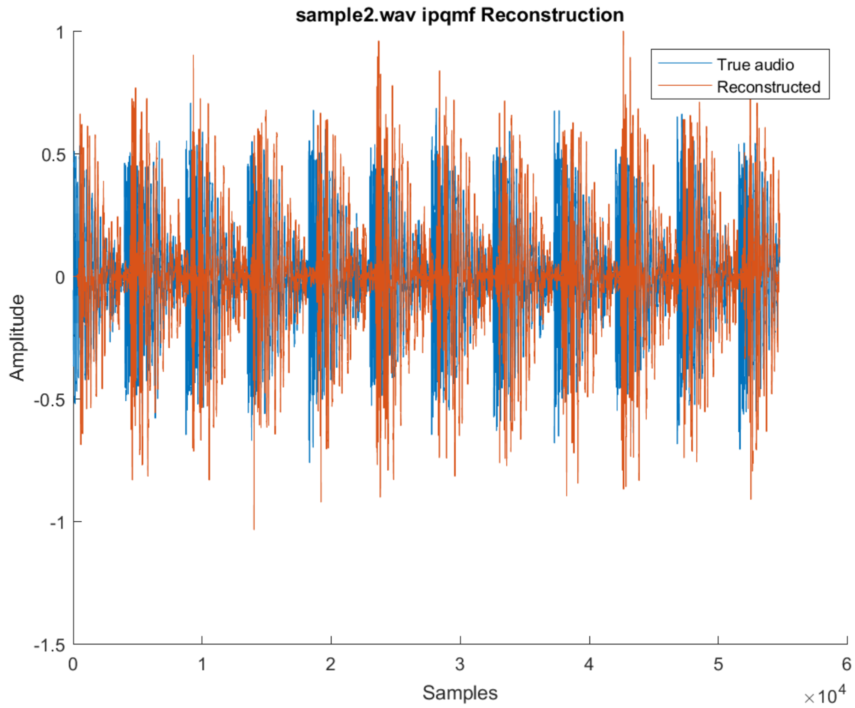
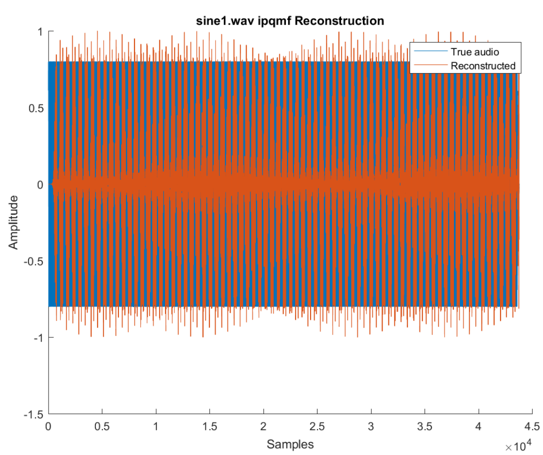
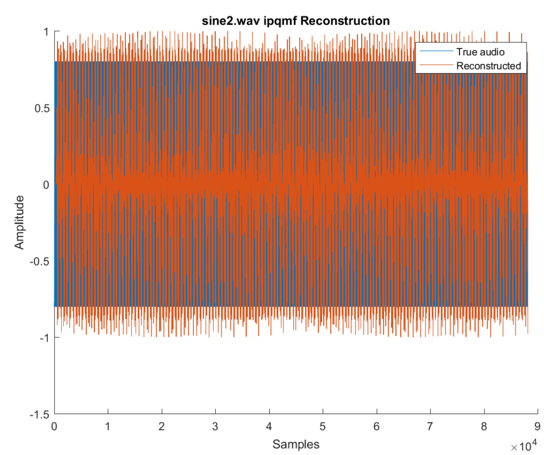


Figure 21: PQMF: Sample2 IPQMF



(a) IPQMF: Sine1



(b) IPQMF: Sine2

Figure 22: Sine Waves

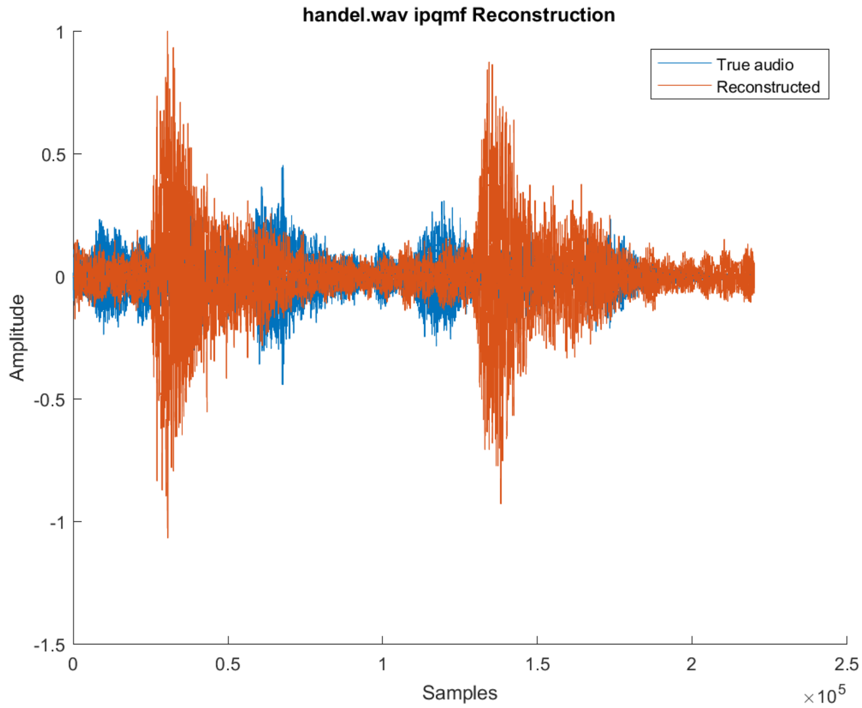


Figure 23: IPQMF: Handel 5 seconds

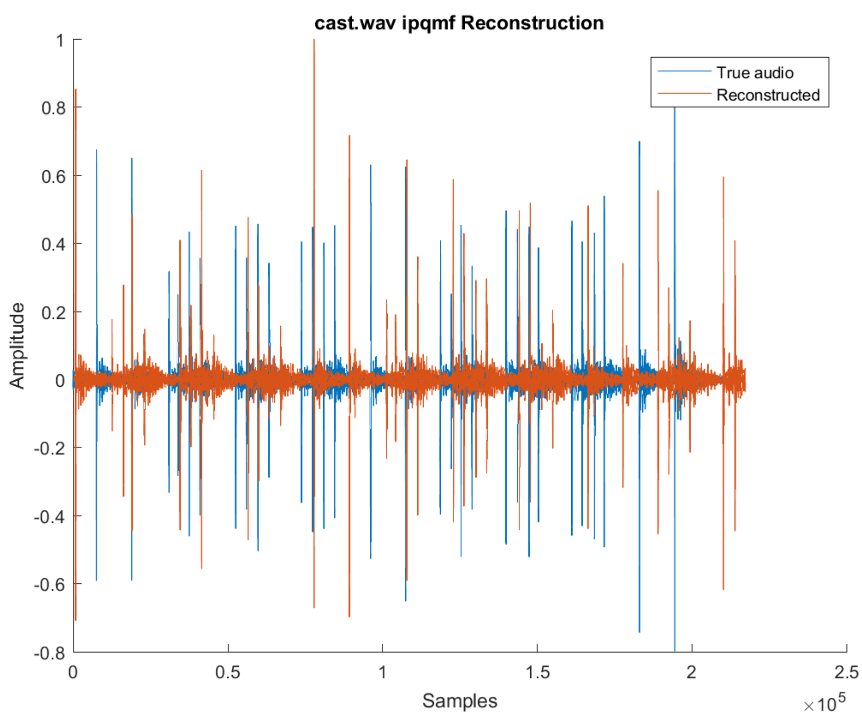


Figure 24: PQMF: Castanets 5 seconds

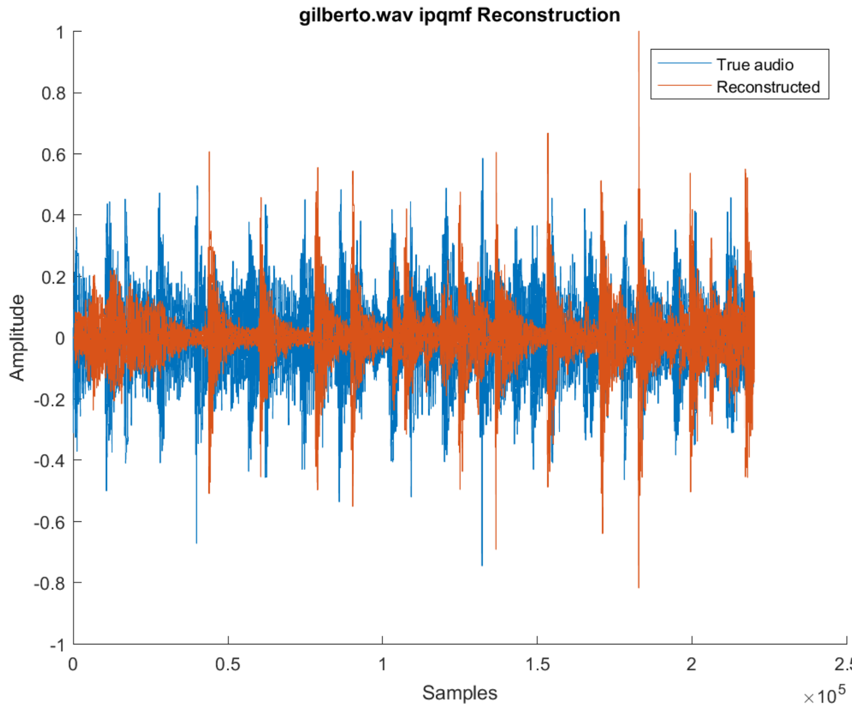


Figure 25: PQMF: Gilberto 5 seconds

A.2.2 Delay Fixed

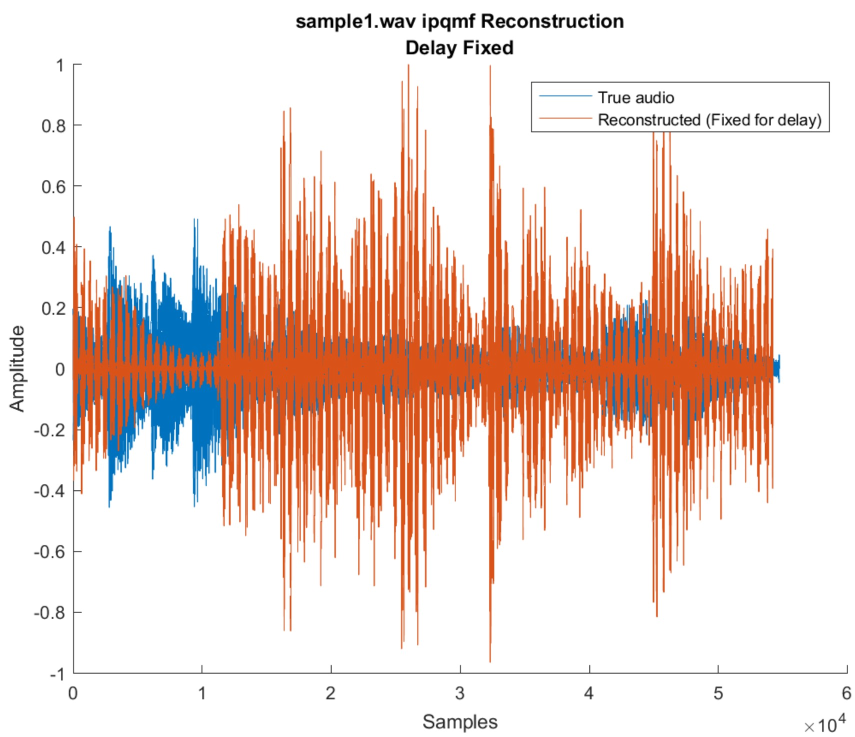


Figure 26: PQMF: Sample1 IPQMF

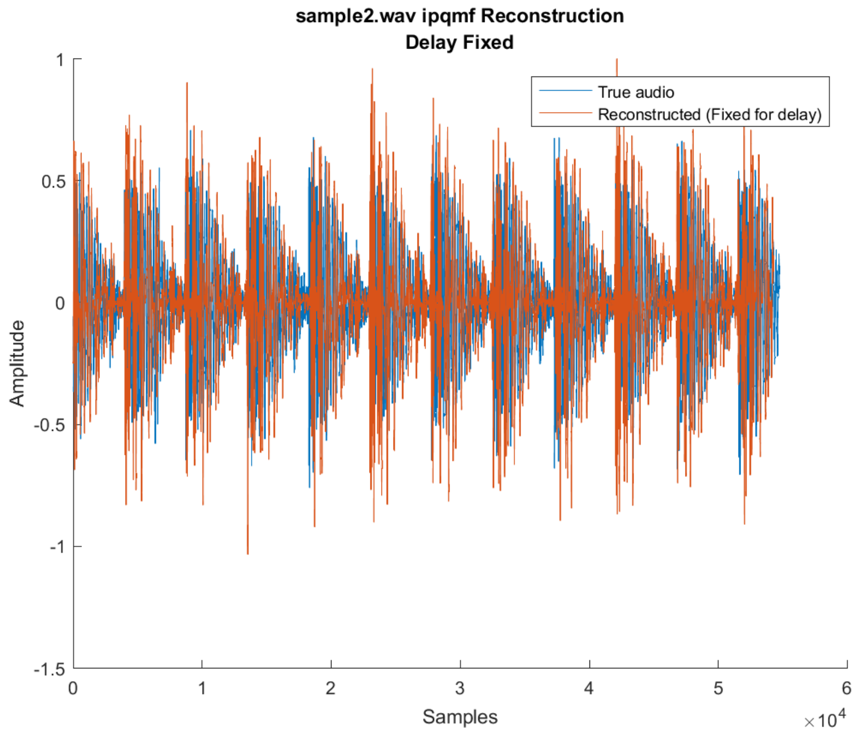
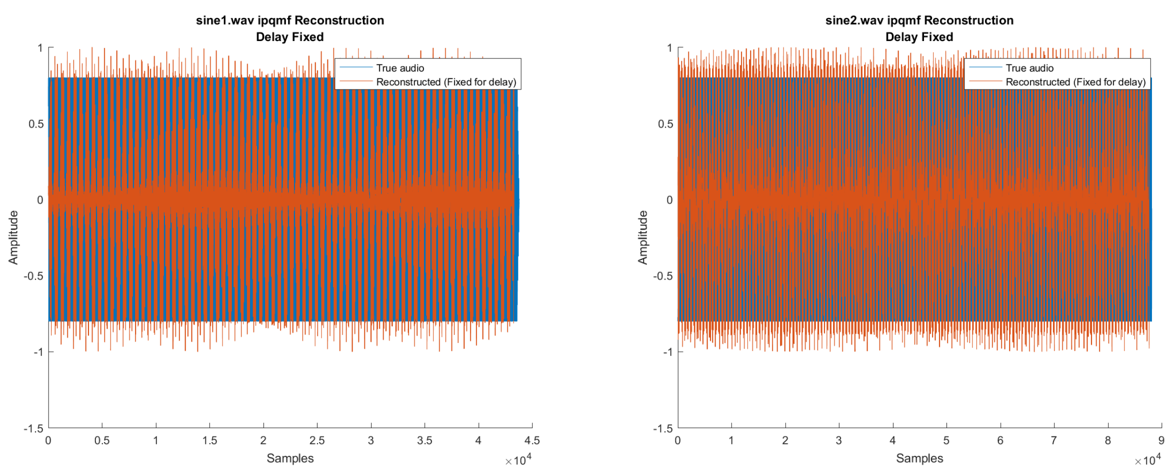


Figure 27: PQMF: Sample2 IPQMF



(a) IPQMF: Sine1

(b) IPQMF: Sine2

Figure 28: Sine Waves

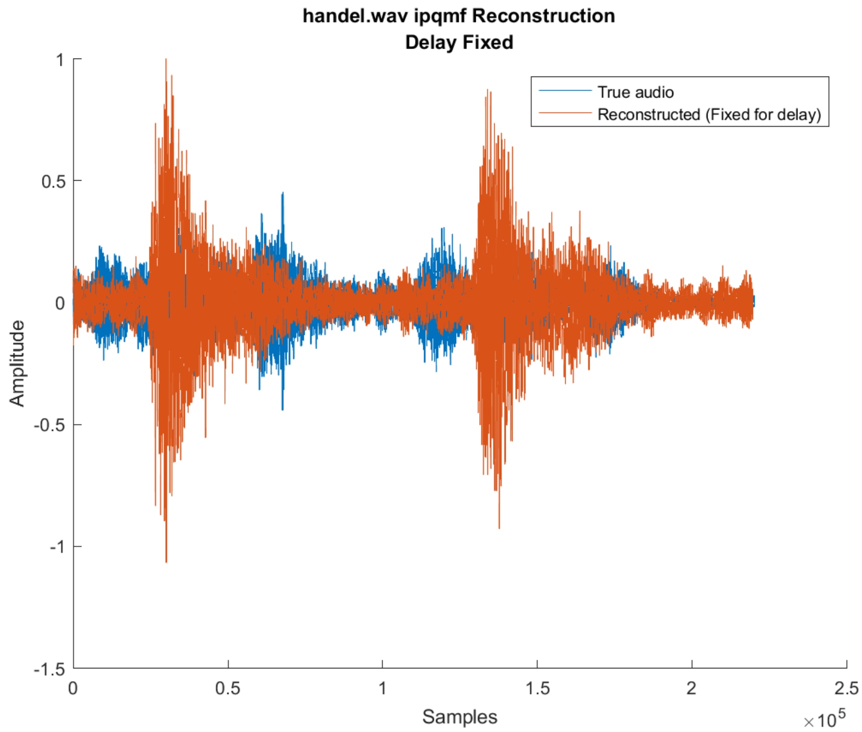


Figure 29: IPQMF: Handel 5 seconds

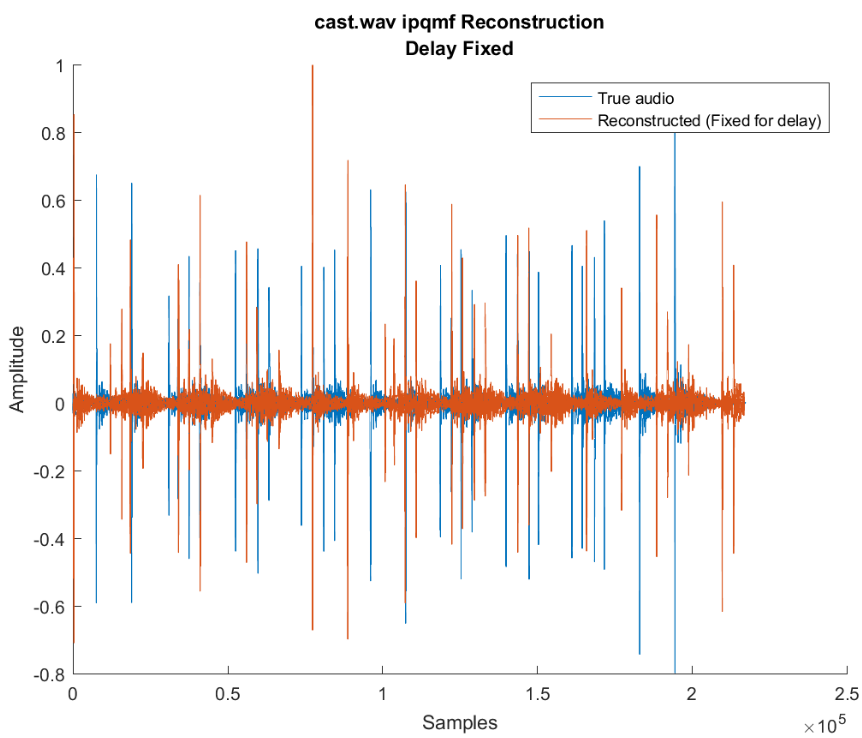


Figure 30: PQMF: Castanets 5 seconds

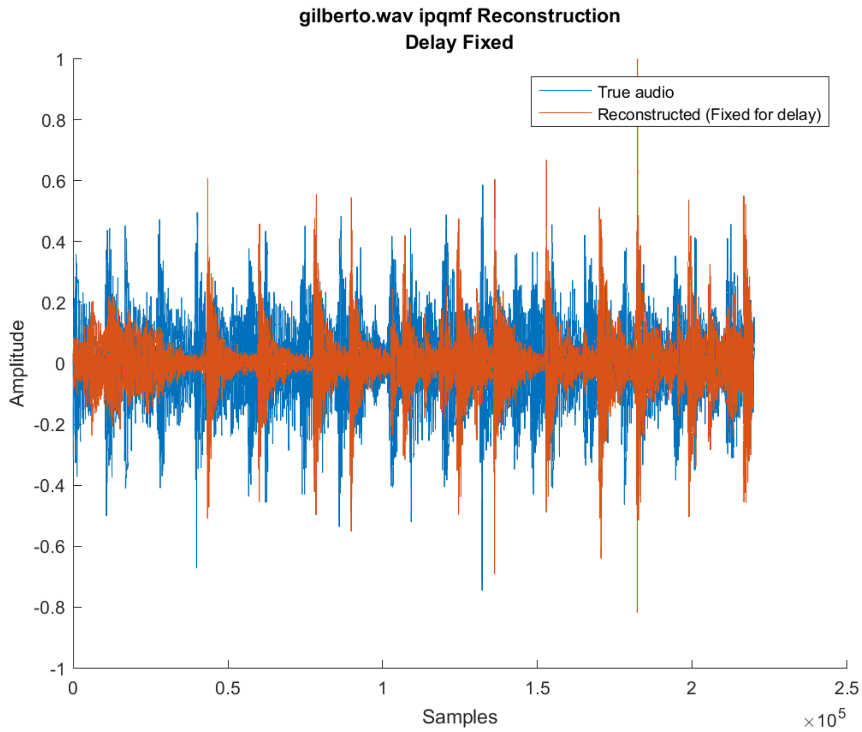


Figure 31: PQMF: Gilberto 5 seconds

A.2.3 Delay Fixed Zoomed

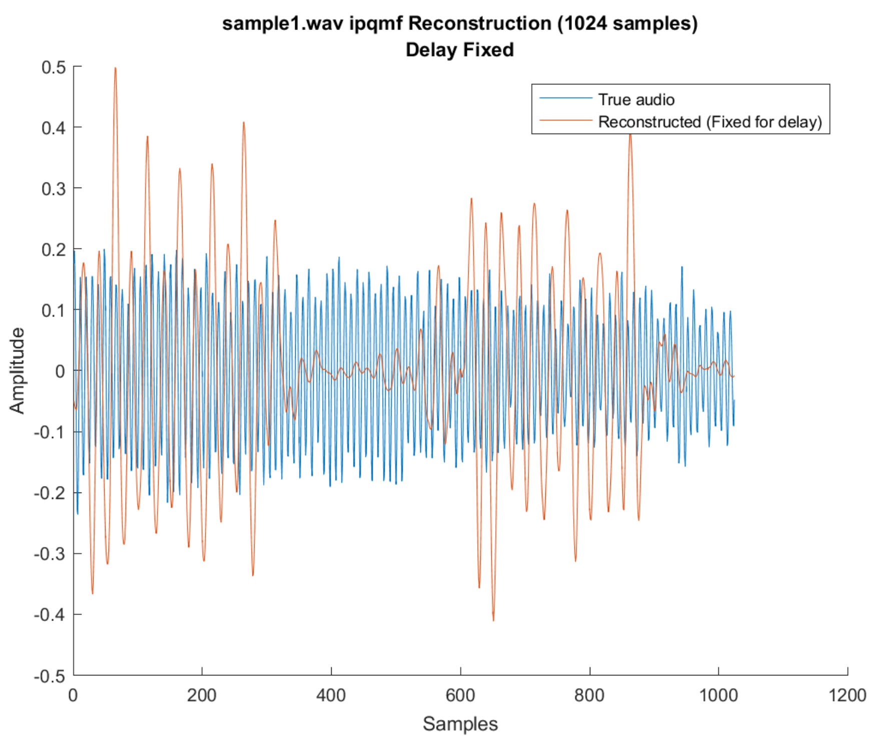


Figure 32: PQMF: Sample1 IPQMF

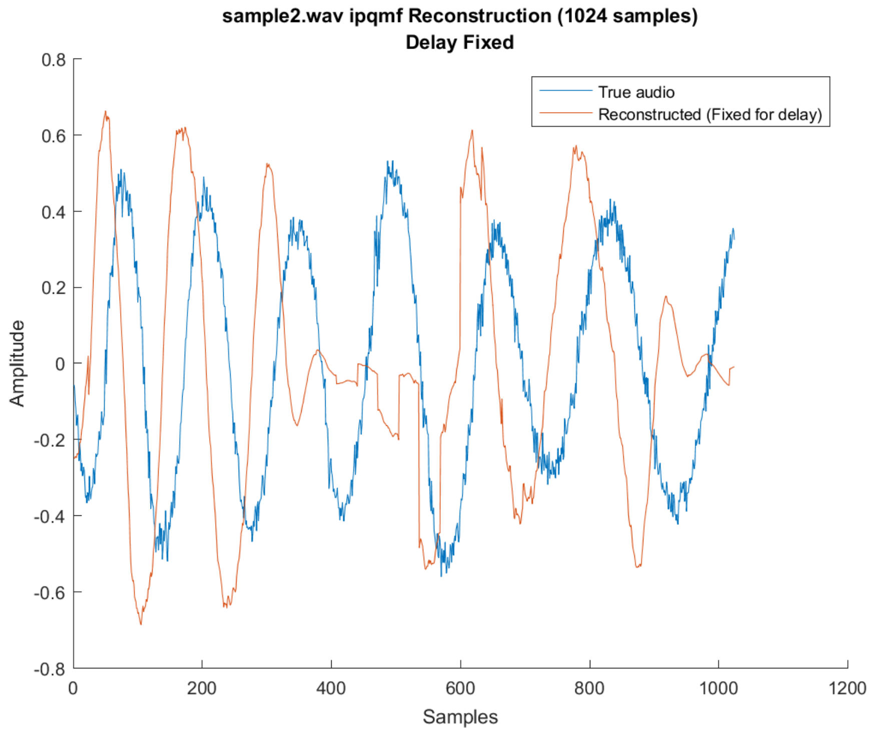
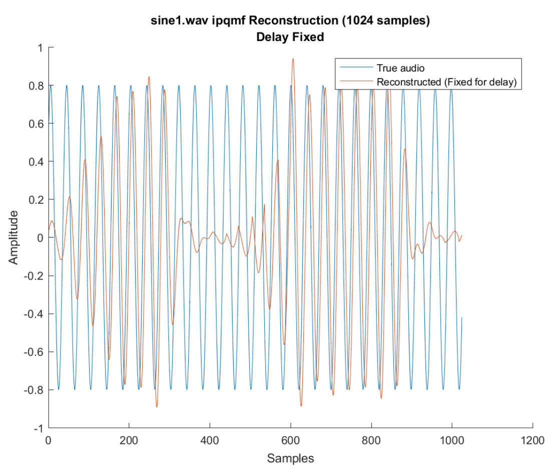
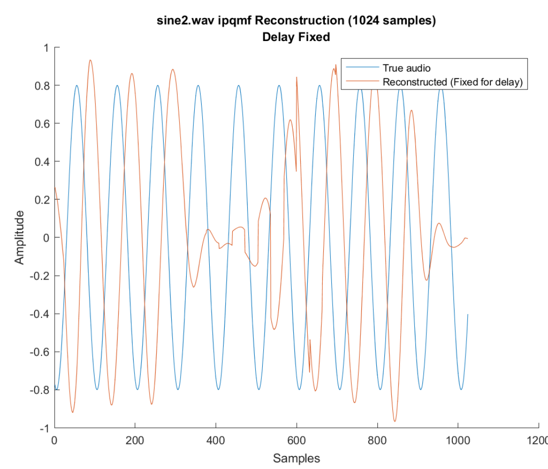


Figure 33: PQMF: Sample2 IPQMF



(a) IPQMF: Sine1



(b) IPQMF: Sine2

Figure 34: Sine Waves

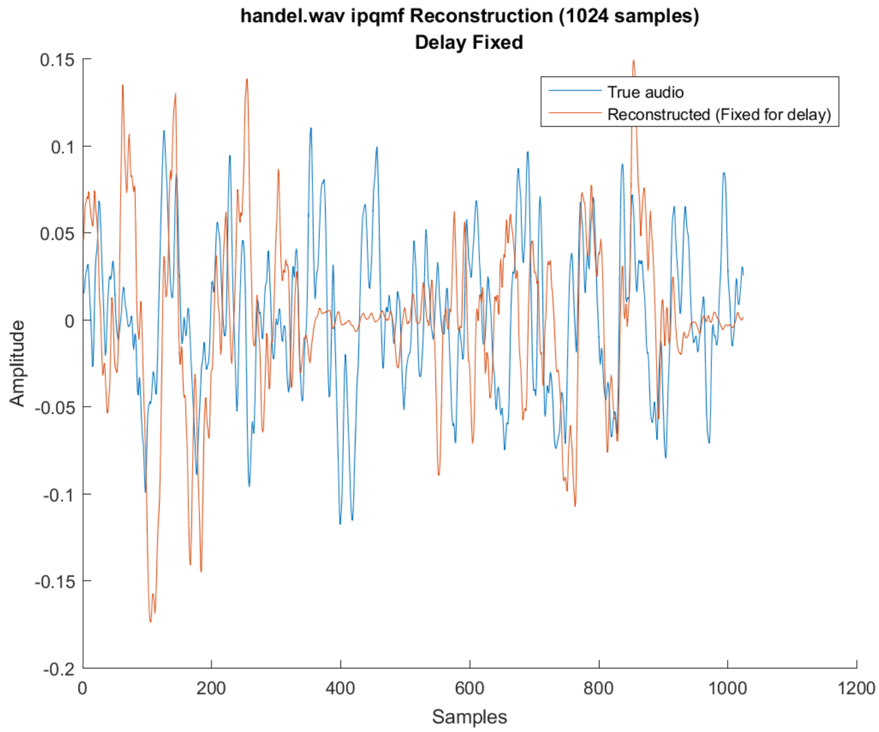


Figure 35: IPQMF: Handel 5 seconds

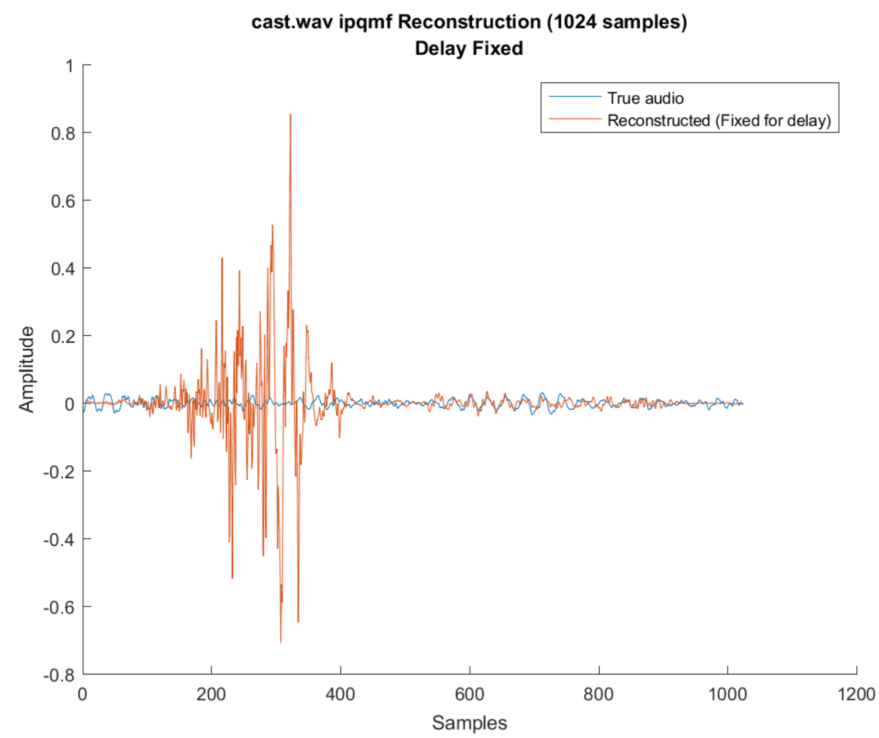


Figure 36: PQMF: Castanets 5 seconds

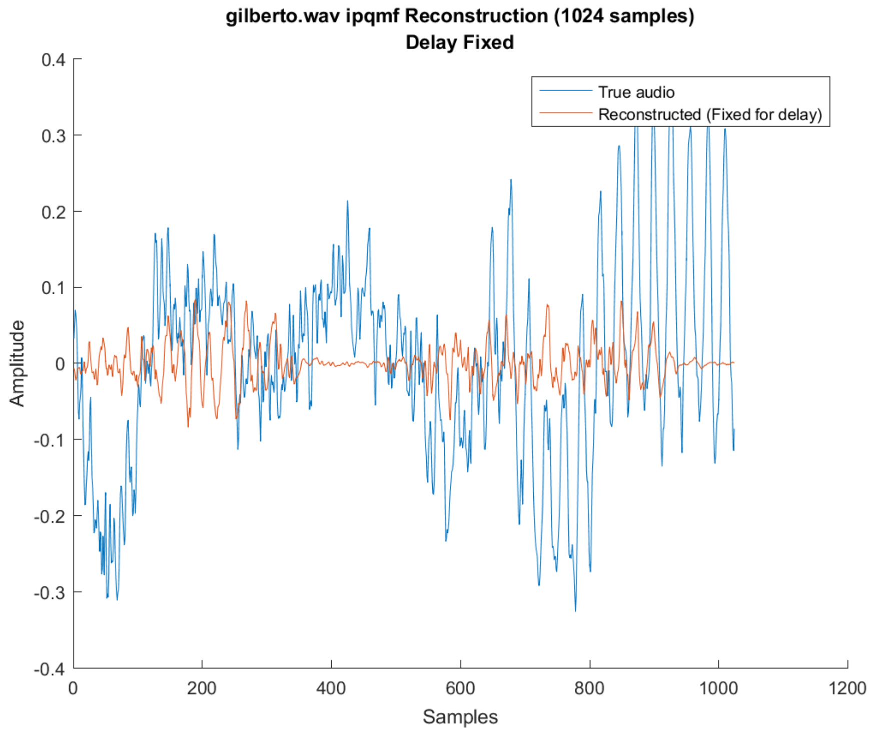


Figure 37: PQMF: Gilberto 5 seconds

A.2.4 Delay Fixed: Specialized Subband (Zoomed)

Figure 38: PQMF: Sample1 IPQMF

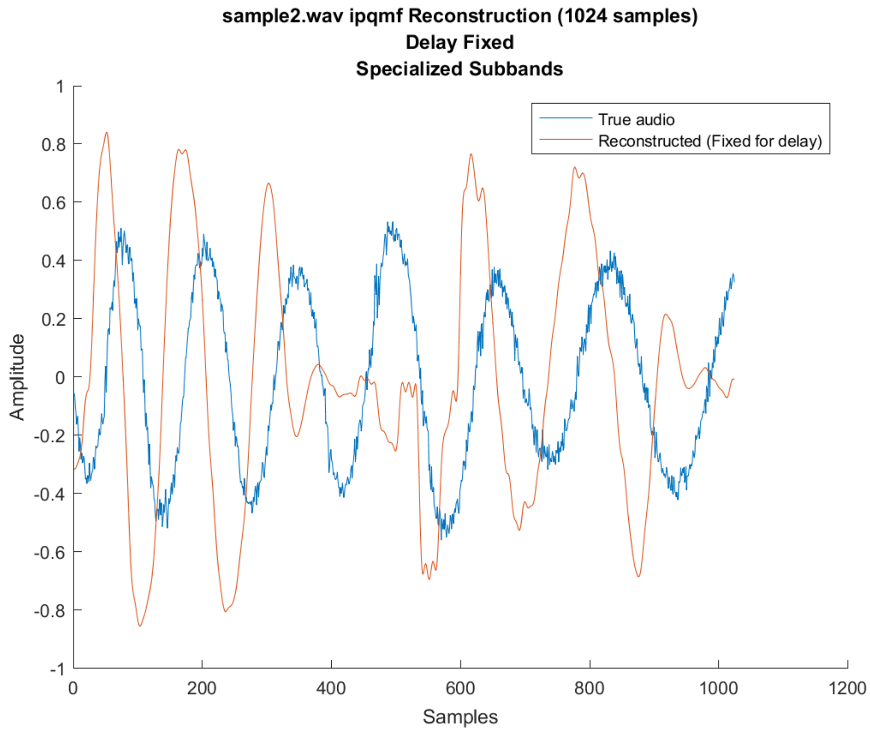
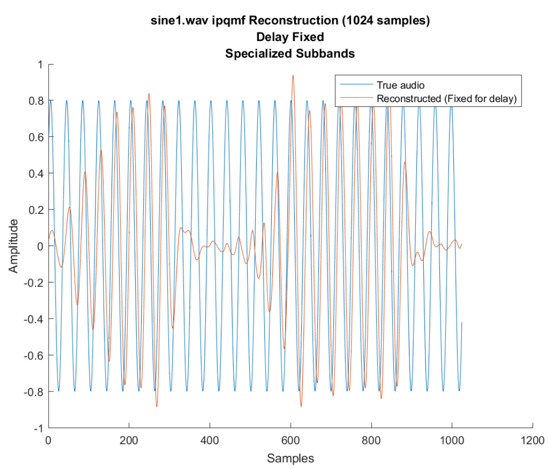
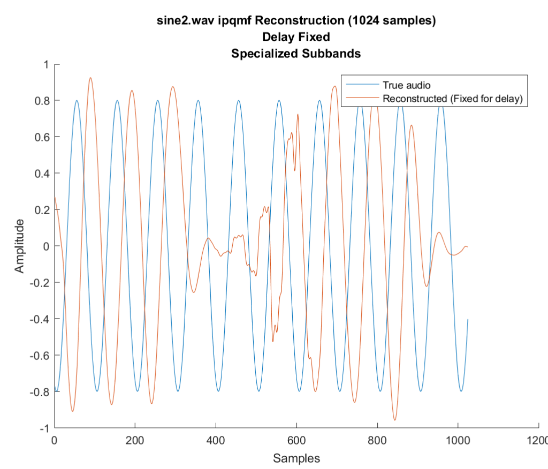


Figure 39: PQMF: Sample2 IPQMF



(a) IPQMF: Sine1



(b) IPQMF: Sine2

Figure 40: Sine Waves

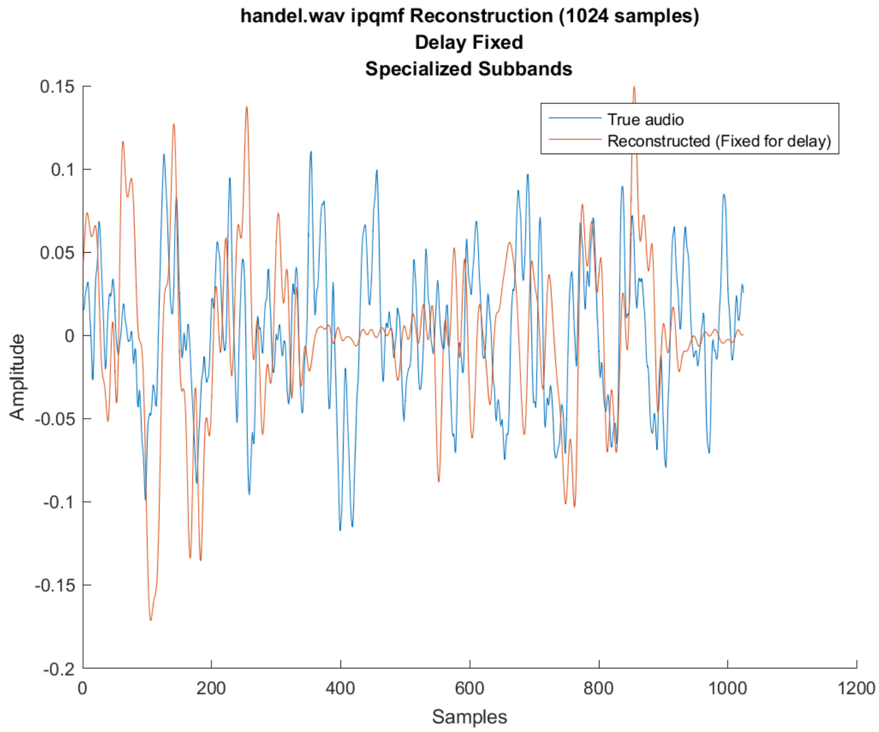


Figure 41: IPQMF: Handel 5 seconds

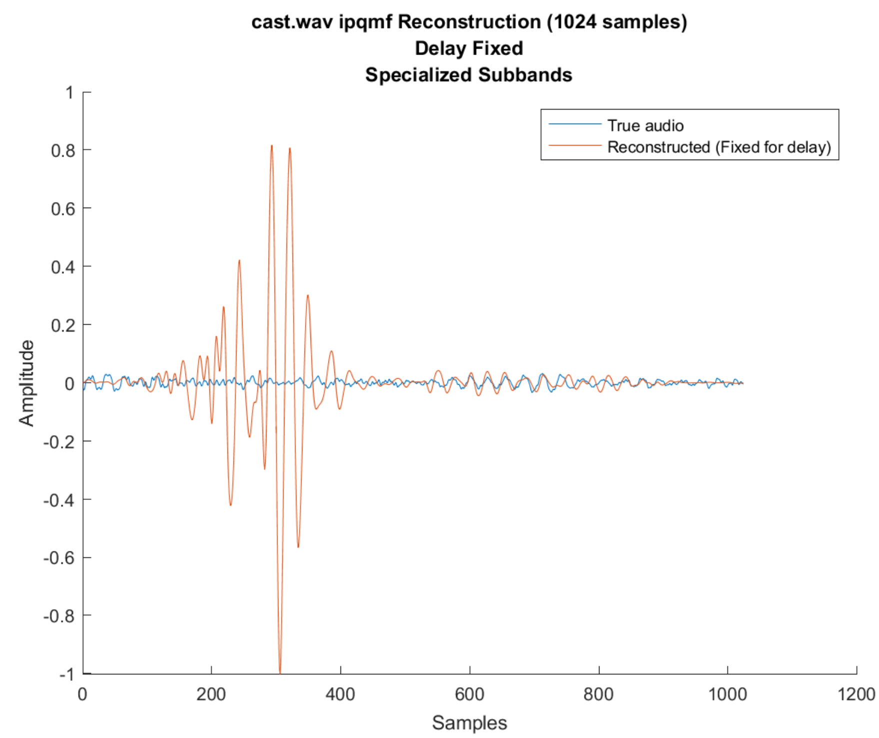


Figure 42: PQMF: Castanets 5 seconds

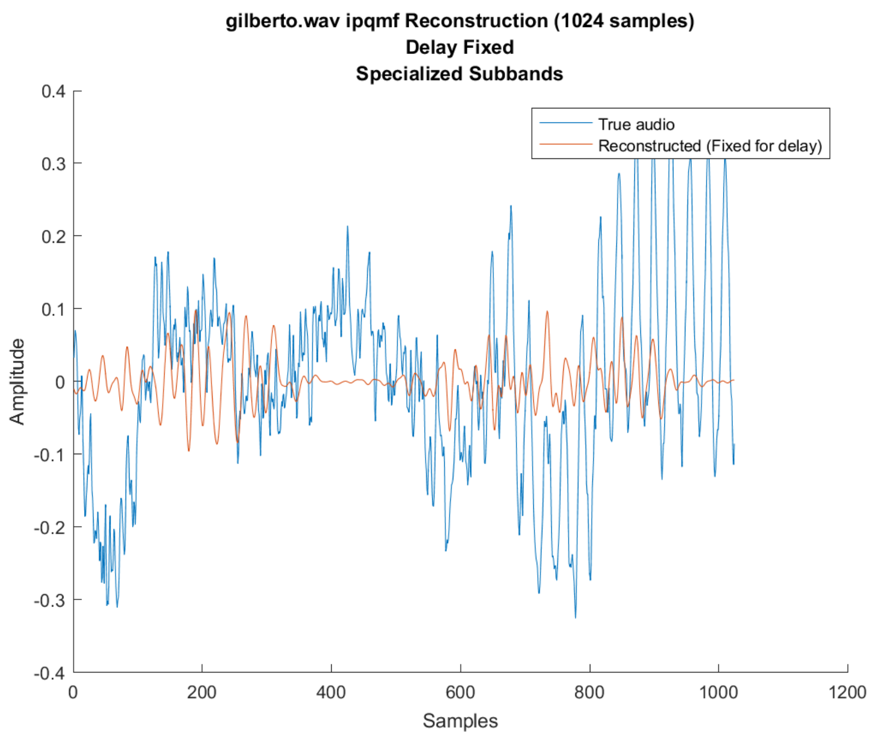


Figure 43: PQMF: Gilberto 5 seconds

B Listings

Listing 4: pqmf.m

```
function [ coefficients ] = pqmf( inputBuffer,~ )
%PQMF Implements the analysis filter bank
%   Takes an input "inputBuffers" that contains an integer number of frames
%   of audio data. The output array "coefficients" has the same size as the
%   buffer "inputBuffers", and contains the subband coefficients.
filenameFlag=0;
if(ischar(inputBuffer)) % I lied, it's actually the filename
    filenameFlag=1;
    filename=inputBuffer;
    info=audioinfo(filename);
    SampleTime=5;
    if(SampleTime>info.Duration)
        SampleTime=info.Duration;
    end
    inputBuffer=audioread(filename,[1,(info.SampleRate*SampleTime)]);
    % Remove opening silence
    inputBuffer=inputBuffer(find(inputBuffer~=0,1):end);
end

totalSamples=length(inputBuffer);
frameSize=576;
nFrame=floor(totalSamples/frameSize);
inputBuffer=inputBuffer(1:(nFrame*frameSize));
[C,~] = loadwindow(); %C is the analysis window
%D is the synthesis window, but is not needed

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
Ns=18*nFrame;
bufferSize=512;
y=zeros(1,64);
S=zeros(32,1);

coefficients=zeros(size(inputBuffer));
packet=1; % counter for inner loop
for frame = 1:nFrame          % chunk the audio into blocks of 576 samples
    offset = (frame -1)*frameSize+1;  % absolute address of the frame
    frameTemp=inputBuffer(offset:(offset+frameSize-1));
    Buffer=zeros(size(C));
    for index = 1:18                % 18 non overlapping blocks of size 32
        Buffer(1:bufferSize-32)=Buffer(33:end);
        newBlock=frameTemp(((index-1)*32+1):index*32); % 32 new samples
        Buffer((bufferSize-31):end)=newBlock;
        % process a block of 32 new input samples
        % see flow chart in Fig. 2
        Z=C.*Buffer; % Window by 512 Coefficients to produce vector

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

        for i=0:31
            S(i+1)=sum(M(i+1,:).*y); % Calculate 32 samples by matrixing
        end

        % Frequency inversion
        if(mod(index,2)==1)
            channel=1:2:32;
            S(channel)=-S(channel); % invert odd-numbered frequencies
        end
        % Spaced Ns apart
        coefficients(packet+(Ns*(0:31)))=S; % Assign coefficients

        packet=packet+1;
    end % end index=1:18
end % end frame=1:nFrame

coefficients=coefficients/max(coefficients); % Normalize



if(nargin==2 && filenameFlag)
    h=figure;
    [~,name,ext]=fileparts(filename);
    plot(coefficients);
    title([name, ext,' pqmf ', num2str(SampleTime) ' seconds']);
    xlabel('Coefficient');
    ylabel('Amplitude');
    saveas(gca,[name,'_',num2str(SampleTime),'sec.png']);
    close(h);
end
end % end function
```

Listing 5: ipqmf.m

```
function [ recons,difference ] = ipqmf( coefficients,thebands,filename,~)
%ipqmf Reconstructs the audio data
%   "coefficients" is a buffer that contains the coefficients as computed
%   by pqmf. The output array "recons" has the same size as the buffer
%   "coefficients", and contains the reconstructed audio data.

```

```
%% Validate input
narginchk(1,4);
nargoutchk(0,2);
bandFlag=1;
if ischar(thebands) % thebands was probably "filename"
    filename=thebands;
    bandFlag=0;
    thebands=zeros(1,32);thebands(:)=1;
end
if(((nargout==2) && (~((nargin==4)||(nargin==3)))...
        || ((nargout~=2) && ((nargin==4)||((nargin==3)&&(~bandFlag)))))
    error('Two outputs requires three or four inputs');
end

if ischar(thebands) % thebands was probably "filename"
    filename=thebands;
    bandFlag=0;
    thebands=zeros(1,32);thebands(:)=1;
end
frameSize=576;
if(mod(length(coefficients),frameSize)~=0)
    error('ipqmf:invalidInputSize',...
        ['Invalid size for ''coefficients''.'...
        ' Length must be multiple of %d.'],frameSize);
end
%% Setup
audioSampleSize=32;
processingSize=64;
bufferSize=1024;
Ns=length(coefficients)/audioSampleSize;
% Because multiple of 576, Ns will be an integer
N=zeros(processingSize,audioSampleSize);
for i=0:processingSize-1
    for k=0:audioSampleSize-1
        N(i+1,k+1)=cos(((2*k+1)*(16+i)*pi)/64);
    end
end

coefficients=buffer(coefficients,Ns); % Matrix is easier
[~,D] = loadwindow(); %C is the analysis window, but is not needed
%D is the synthesis window

%% Init vars
recons=zeros(audioSampleSize,Ns);
Buffer=zeros(bufferSize,1);

%% Do the magic
for packet = 1:Ns
    U=zeros(size(D));
    S=coefficients(packet,:).*thebands; % extract subband
    if (mod(packet,2) == 1) % Act on every other packet
        channel = 1:2:32; % Invert every other coefficent
        S(channel) = -S(channel); % Invert coefficent
    end
    %% Shift Buffer
    Buffer((processingSize+1):bufferSize)=...
        Buffer(1:(bufferSize-processingSize));
    for i=0:processingSize-1
        Buffer(i+1) = sum(N(i+1,:).*S);
    end
    %% DSP Magic
    j=0:(audioSampleSize-1);
    for i=0:7
        U(i*64+j+1) = Buffer(i*128+j+1); % DSP magic
        U(i*64+32+j+1) = Buffer(i*128+96+j+1); % DSP magic
    end
    W=U.*D; % Windows by 512 coefficients

```

```
    for j=0:audioSampleSize-1 % Calculate 32 samples
        recons(j+1,packet) =(sum(W((j+1) + audioSampleSize*(0:15))));
    end
end
% Output 32 reconstructed PCM samples
recons=recons(:); % Change back to vector
recons=recons/max(recons); % Normalize it

%% Plot the magic
if(nargin > 1)
    h=figure;
    hold on;
    audio=audioread(filename);
    audio=audio((length(audio)-length(recons)):end);
    plot(audio(1:length(recons)));
    xlabel('Samples');
    ylabel('Amplitude');
    [~,name,ext]=fileparts(filename);
    plot(recons);
    title([name, ext, ' ipqmf Reconstruction']);
    legend('True audio','Reconstructed');
    saveas(h,[name,'_ipqmf.png']);
    close(h);

    if((nargin == 4)|| ((nargin==3)&& (bandFlag==0)))
        offset1=489;
        h=figure;
        hold on;
        plot(audio);
        xlabel('Samples');
        ylabel('Amplitude');
        plot(recons(offset1:end));

        title({[name, ext, ' ipqmf Reconstruction'],'Delay Fixed'});
        legend('True audio','Reconstructed (Fixed for delay)');
        saveas(h,[name,'_D_ipqmf.png']);

        close(h);

        h=figure;
        hold on;
        plot(audio(1:1024));
        plot(recons((1:1024)+offset1));
        xlabel('Samples');
        ylabel('Amplitude');
        legend('True audio','Reconstructed (Fixed for delay)');
        if(bandFlag)
            title({[name, ext, ' ipqmf Reconstruction (1024 samples)'],...
                'Delay Fixed', 'Specialized Subbands'});
            saveas(h,[name,'_DS1024_ipqmf.png']);
        else
            title({[name, ext, ' ipqmf Reconstruction (1024 samples)'],...
                'Delay Fixed'});
            saveas(h,[name,'_D1024_ipqmf.png']);
        end
        close(h);

        difference=sum(abs(audio(1:1024)-recons((1:1024)+offset1)));
    end
end
end
```

Listing 6: buildReport.m

```
%% Build everything necessary for report
clear filename tracks;
close all;
```

```
tracks=setupFiles();

%% Plot windows
[C,D] = loadwindow();
h=figure;
plot(C);
title('Analysis Window');
xlabel('Coefficient');
ylabel('Magnitude');
axis auto;
saveas(h,'analysisWindow.png');
close(h);

h=figure;
plot(D);
title('Synthesis Window');
xlabel('Coefficient');
ylabel('Magnitude');
axis auto;
saveas(h,'synthesisWindow.png');
close(h);


%% Runs through all the files
len=length(tracks);
totalError=zeros(len,1);
totalError1=zeros(len,1);
subbands=zeros(1,32);
subbands(1:6)=1; % As can be seen in the Pan95-mpega.pdf
parfor i=1:len
    filename=tracks{i};
    coefficients=pqmf(filename,1);
    [~,totalError1(i)]=ipqmf(coefficients,subbands,filename,1);
    [~,totalError(i)]=ipqmf(coefficients,filename,1);

end

songNames='';
for i=1:len
    [~,name,~]=fileparts(tracks{i});
    songNames=[songNames ' ' name];
end
songNames(1)=[]; % Remove extra space

h=figure;
bar(totalError);
title('Total Error');
xlabel('Track');
ylabel('Error Difference');
ax=gca;
axis([-inf inf 0 max(totalError)*1.025]);
ax.XTickLabel=strsplit(songNames);
saveas(h,'totalError.png');
close(h);

h=figure;
bar(totalError1);
title({'Total Error','Specialized Subbands'});
xlabel('Track');
ylabel('Error Difference');
ax=gca;
axis([-inf inf 0 max(totalError)*1.025]);
ax.XTickLabel=strsplit(songNames);
saveas(h,'totalError1.png');
close(h);
```