Lab 1

Michael Eller

February 1, 2016

Introduction

The goal of this lab is to explore different methods of categorizing music into specific genres. Several methods of automated music analysis exist already, including **score following**, **automatic music transcription**, **music recommendation**, and **machine listening**. Unfortunately, all of these tools are still rudimentary. Faster, more efficient, methods still need to be developed in order to allow automated music analysis to be implemented on portable music players. The goal of this lab is to develop tools to automatically extract some defining features of music that will help us to categorize them more easily.

1 Sampling Rates

Current high-quality audio standards have a 24-bit depth and is sampled at 96 kHz. This 24-bit depth means there are 16,777,216 possible values for the audio signal at any given instance. It also means we are able to replicate frequencies up to 48 kHz. While this is far above the 20 kHz limit of human hearing, DVD audio is simply not high enough quality enough for a dolphin to listen to.

Assignment

1. Dolphins can only hear sounds over the frequency range [7 - 120] kHz. At what sampling frequency f_s should we sample digital audio signals for dolphins?

Nyquist theorem states that in order to accurately recreate a signal with maximum frequency $f_s/2$, we must sample at a minimum frequency of f_s. Therefore, our sampling frequency for digital audio signals for dolphins should be at minimum 240 kHz. Currently, one of the highest portable audio standards of BD-ROM LPCM (lossless) allow for 24 bit/sample and a maximum sampling frequency of only 192 kHz. Less common standards do exist though: Digital eXtreme Definition at 352.8 kHz using for recording and editing Super Audio CDs, SACD at 2,822.4 kHz known as Direct Stream Digital, and Double-Rate DSD at 5,644.8 kHz used in some professional DSD recorders. Therefore, as it stands, we might want to stay away from producing audio for dolphins (at least until better recording standards are developed.

2 Audio signal

Assignment

2. Write a MATLAB function that extract T seconds of music from a given track. You will use the MATLAB function ~~waveread~~ audioread to read a track and the function play to listen to the track.

   In the lab you will use T =**24 seconds** from the middle of each track to compare the different algorithms. Download the files, and test your function.

Listing 1: extractSound.m

```
function [ soundExtract,p ] = extractSound( filename, time )
%extractSound Extracts time (in seconds) from the middle of the song
%   Write a MATLAB function that extract T seconds of music from a
%   given track. You will use the MATLAB function audioread to
%   read a track and the function play to listen to the track.
info = audioinfo(filename);
[song,~]=audioread(filename);
if time >= info.Duration
    soundExtract=song;
    p=audioplayer(soundExtract,info.SampleRate);
    return;
elseif time<= 1/info.SampleRate
    error('Too small of a time to sample');
end
samples=time*info.SampleRate;
soundExtract=song(floor(info.TotalSamples/2)-floor(samples/2):1: ...
    floor(info.TotalSamples/2)+floor(samples/2));
p=audioplayer(soundExtract,info.SampleRate);
end
```

This MATLAB function is fairly straightforward. MATLAB's built-in function *audioinfo* provided all the necessary attributes to allow my function to parse any .wav file and extract the number of samples needed.

3 Low Level Features: Time-Domain Analysis

The bulk of music analysis is done in the frequency spectrum, however, there are some low level features that can be found from the time-domain analysis of music as well.

Assignment

3. Implement the loudness and ZCR and evaluate these features on the different music tracks. Your MATLAB function should display each feature as a time series in a separate figure.

4. Comment on the specificity of the feature and its ability to separate different musical genre.

3.1 Loudness

There is not an easy way to describe *loudness*, but one way to mathematically quantize it is by finding it standard deviation, $\sigma(n)$.

$$\sigma(n) = \sqrt{\frac{1}{N-1}\sum_{m=-N/2}^{N/2-1}[x(n+m) - \mathbb{E}[x_n]]^2} \quad \text{with} \quad \mathbb{E}[x_n] = \frac{1}{N}\sum_{m=-N/2}^{N/2-1} x(n+m) \tag{1}$$

The following is an excerpt from my *loudness* function. My *extractSound* function extracts 24 seconds around the middle of the song. The song excerpt is then split into frames of size 255 overlapped (at least) halfway with the previous frame.

Listing 2: loudness.m

```
[y,~]=extractSound( filename, time );
frames_data = buffer(y,frameSize,ceil(frameSize/2));
loudness_data=std(frames_data,0,1);
```

Figure 1 shows an example of the function applied to "track201-classical.wav". The figure would imply that, for the selection of the song, the beginning is quite loud, then quiets down until two loud moments, and finishes at a moderately quiet section. If one listens to the song selection in question, the results of my loudness function can be confirmed. The other loudness plots can be found in Appendix A.1

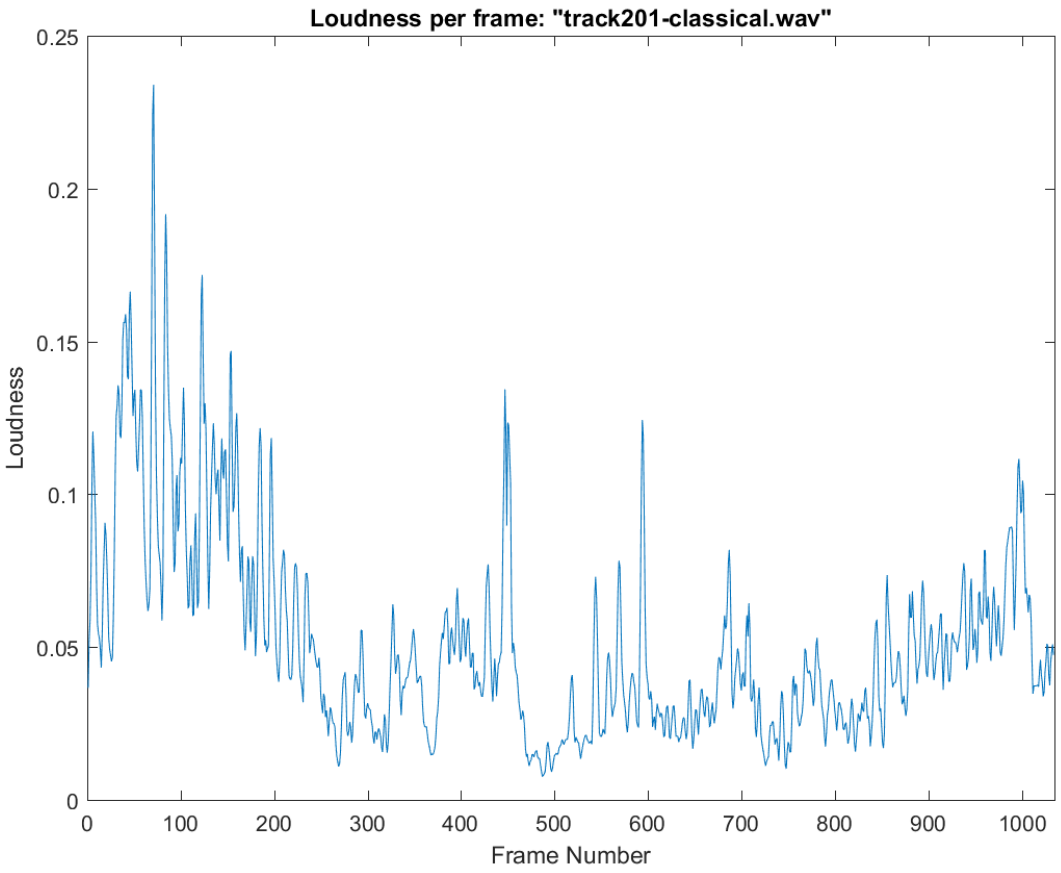

Figure 1: Loudness value per frame

The **loudness** of a track appears to be most helpful in identifying *classical* and *jazz* music. Classical is very fluid and will not transition quickly between a high vs a low loudness. Jazz is a very periodic music and that is evident by the loudness plots where there are periodic spikes with a lull in between.

3.2 Zero-Crossing Rate (ZCR)

Another low-level feature that can be gathered from a time-domain analysis is the Zero-Cross Rate (ZCR). This feature has been used heavily in both speech recognition and music information retrieval, being a key feature to classify percussive sounds, where a frequency analysis would not be able to identify such features.

$$ZCR(n) = \frac{1}{2N} \sum_{m=-N/2}^{N/2-1} |sgn(x(n+m)) - sgn(x(n+m-1))| \tag{2}$$

The following is an excerpt from my *zeroCross* function. Once again, I extract the middle 24 seconds of audio from my track and store it into frames of size 255. Once **ZCR_data** has been allocated, each adjacent sample is compared to each other to check if they have crossed the zero-axis.

Listing 3: zeroCross.m

```
10 [y,~]=extractSound( filename, time ); % Operate on middle 24 seconds
11 frames_data = buffer(y,frameSize,ceil(frameSize/2));
12 ZCR_data=zeros(1,length(frames_data));
```

```
13  ZCR_data(1:end)=sum(abs(diff(frames_data(1:end,:))))/length(frames_data);
```

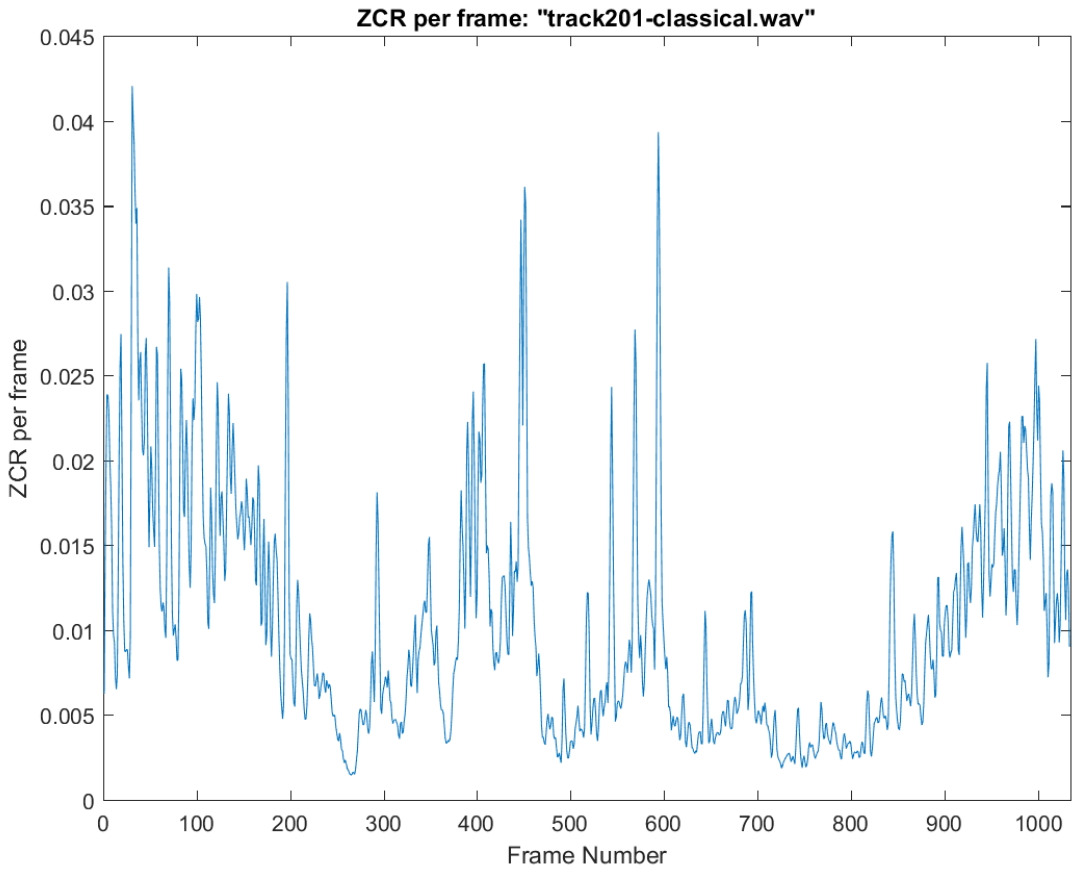

Figure 2: Average ZCR per frame

Figure 2 shows an example of the ZCR function applied to "track201-classical.wav". The plot produced by the ZCR function appears similar in shape to the plot from the previous loudness function. The ZCR function is more closely representative of how **noisy** a function is though. The other ZCR plots can be found in Appendix A.2.

Just as the **loudness** function, the **ZCR** function appears to be most helpful in identifying *classical* and *jazz* music. There doesn't seem to be any advantage in using **loudness** vs **ZCR** in identifying music genres. One method may prove to be faster on a large-scale integration, but such results cannot be determined on such a small unit test such as this one.

4 Low Level Features: Spectral Analysis

In this section, our goal is to reconstruct a musical score based on the spectral analysis of a piece. If we can apply the proper window function to each frame, we might be able to ascertain which note(s) are being played at that specific moment.

To properly evaluate the frequency domain of a frame, we need to perform the following operations:

$$\begin{aligned} Y &= \mathrm{FFT}(w. * x_n); \\ K &= N/2 + 1; \\ X_n &= Y(1:K); \end{aligned} \tag{3}$$

where w is our windowing function and N is the number of samples per frame.

Assignment

5. Let
$$x[n] = \cos(\omega_0 n), n \in \mathbb{Z} \tag{4}$$
Derive the theoretical expression of the discrete time Fourier transform of x, given by
$$X(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x[n] e^{-j\omega n} \tag{5}$$

The Fourier transform of a cosine function is trivial at this point. A cosine function can be regarded as a sum of two complex exponentials. And the Fourier transform of a complex exponential is simply the dirac delta function offset by frequency ω. Therefore, the Fourier transform of $x[n] = \cos(\omega_0 n), n \in \mathbb{Z}$ is:

$$\sqrt{\frac{\pi}{2}}\delta(\omega - \omega_0) + \sqrt{\frac{\pi}{2}}\delta(\omega + \omega_0)$$

Assignment

6. In practice, we work with a finite signal, and we multiply the signal $x[n]$ by a window $w[n]$. We assume that the window w is non zero at times $n = -N/2, \ldots, N/2$, and we define
$$y[n] = x[n]w[n - N/2] \tag{6}$$
Derive the expression of the Fourier transform $y[n]$ in terms of the Fourier transform of x and the Fourier transform of the window w.

From prior knowledge, we know that the Fourier transform multiplication of two functions in time domain is equal to the periodic convolution of two functions in the Fourier domain, as follows:

$$\text{if } y[n] = x[n] \cdot h[n]$$
$$\text{then, } Y(e^{j\omega}) = \frac{1}{2\pi}\int_{-\pi}^{\pi} X(e^{j\theta}) \cdot H(e^{j(\omega-\theta)}) d\theta$$

We also know that a shift in time equates to a scaling in the frequency domain:

$$\mathcal{F}(x[n-k]) = X(e^{j\omega})e^{-j\omega k}$$

From these two properties, we can equate the DTFT of $y[n]$.

$$\text{if } y[n] = x[n]w[n-N],$$
$$\text{then } Y(e^{j\omega}) = \frac{1}{2\pi}\int_{-\pi}^{\pi} X(e^{j\theta}) \cdot (H(e^{j(\omega-\theta)}) \cdot e^{-j\omega N}) d\theta$$

Assignment

7. Implement the computation of the windowed Fourier transform of y, given by Equation 3. Evaluate its performance with pure sinusoidal signals and different windows:
   - Bartlett
   - Hann
   - Kaiser

In order to nicely bridge the gap between our continuous time perception of sound and a computer's discrete time processing, a windowing function is often used. This way, we are able to recreate the desired signal as close as possible without generating any high frequency components that result from abruptly stopping a time sample. In Figure 3 below, we are able to see the performance differences between each window. Overall, the Bartmann and Hamming windows seemed to perform the fastest, albeit by only a few milliseconds.

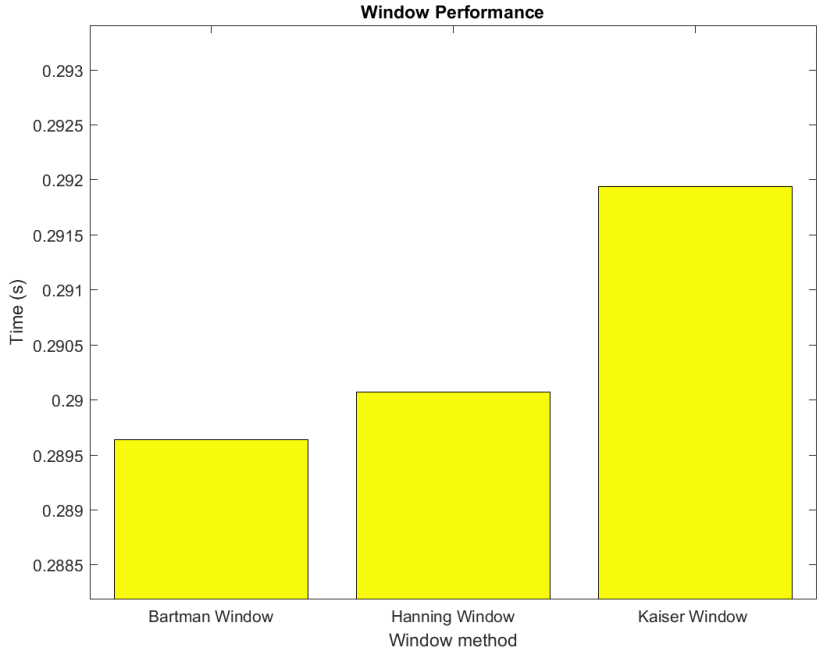

Figure 3: Timing comparison of different windows

Listing 4: windows.m

```
15 [x,~]=audioread(filename);
16 xn=buffer(x,frameSize);
17 Y=zeros(size(xn));
18 for i=1:length(xn)
19     Y(:,i)=fft(xn(:,i).*w);
20 end
21 K=frameSize/2+1;
22 Xn=size(Y);
23 for i=1:length(xn)
24    Xn(1:K,i)=Y(1:K,i);
25 end
```

```
26 end
```

In addition to how quickly each window is able to operate, we can also see how effectively each window extracts our signal. This can be seen in figures 4, 5, and 6.

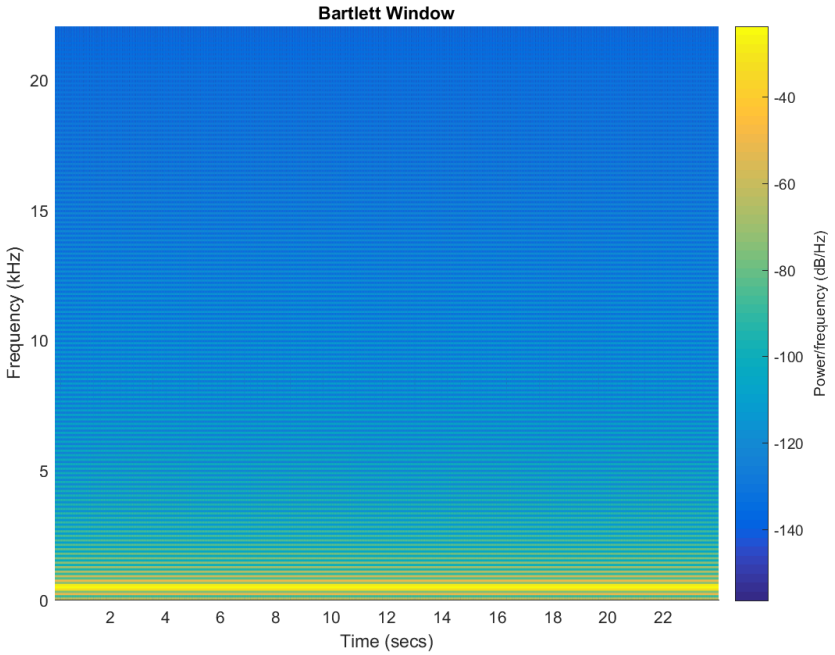

Figure 4: Spectrogram of Bartlett Window

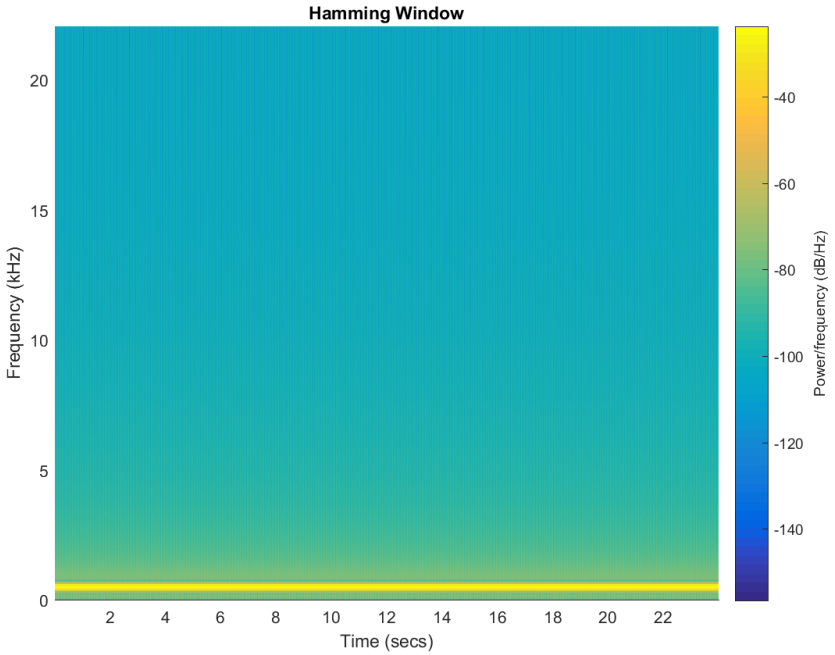

Figure 5: Spectrogram of Hamming Window

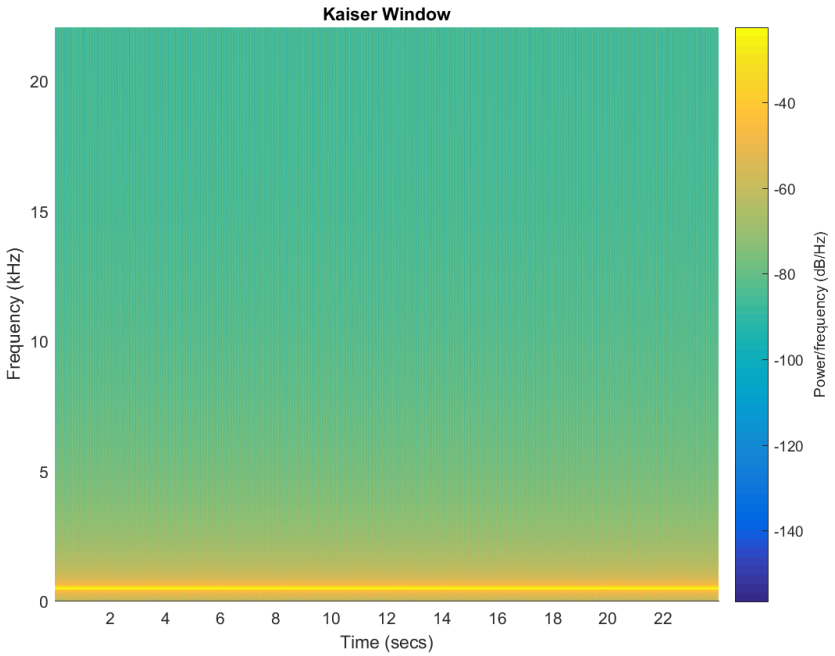

Figure 6: Spectrogram of Kaiser Window

A Figures

A.1 Loudness

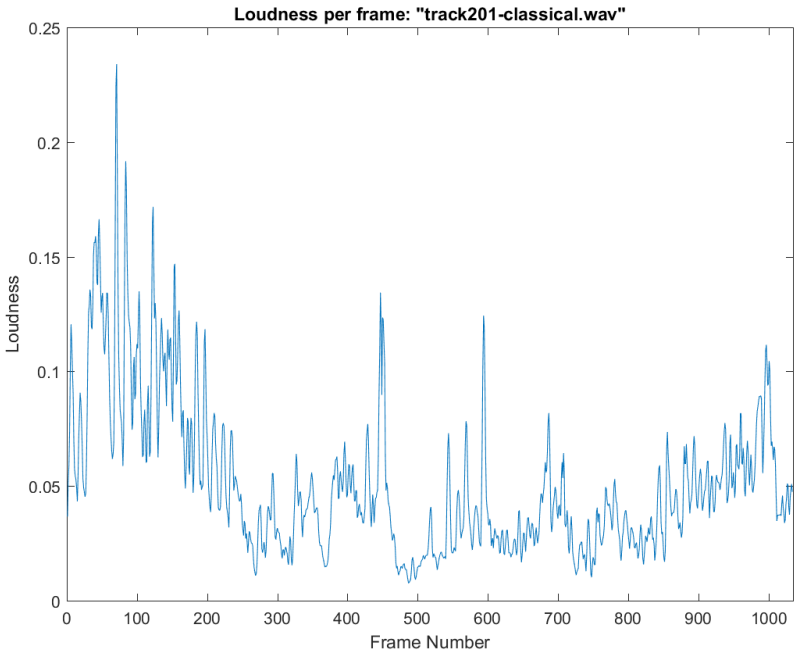

Figure 7: Loudness value per frame, classical201

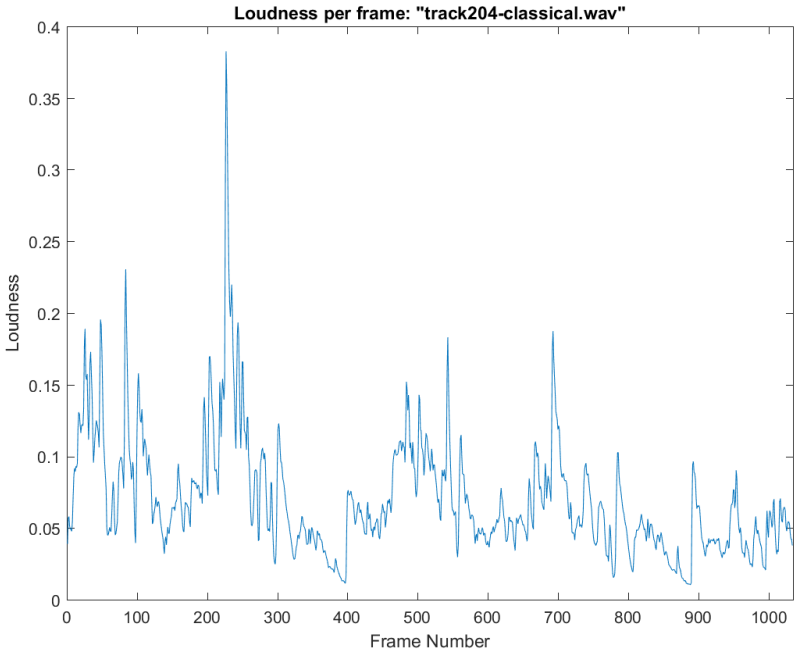

Figure 8: Loudness value per frame, classical204

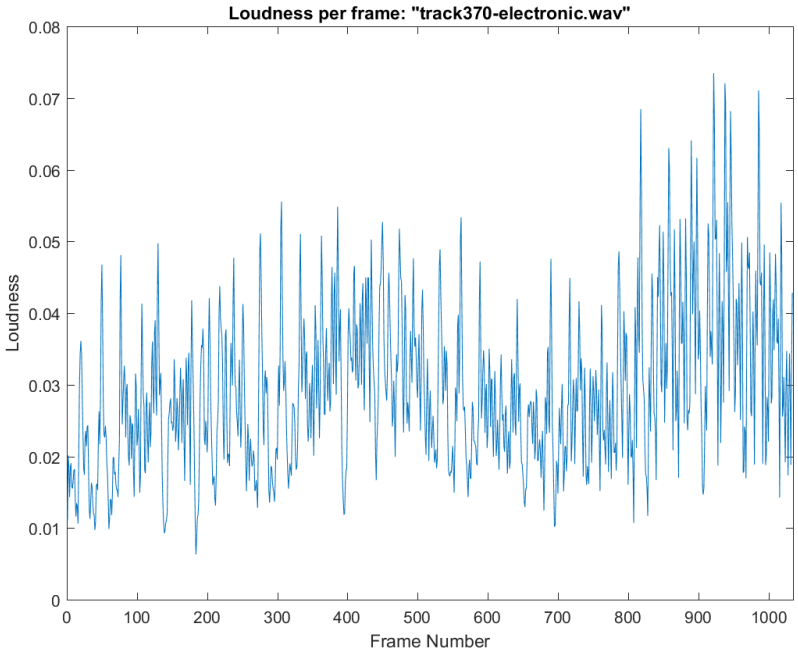

Figure 9: Loudness value per frame, electronic370

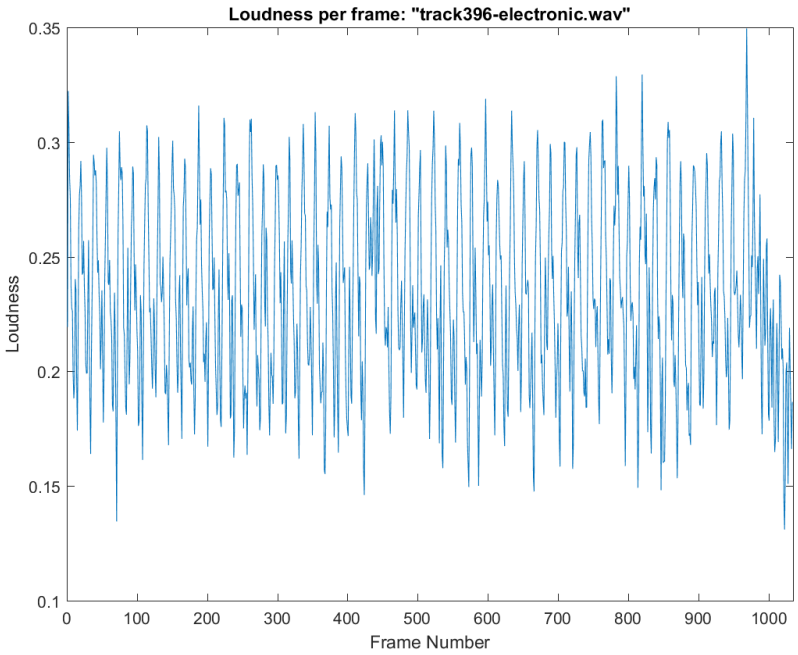

Figure 10: Loudness value per frame, electronic396

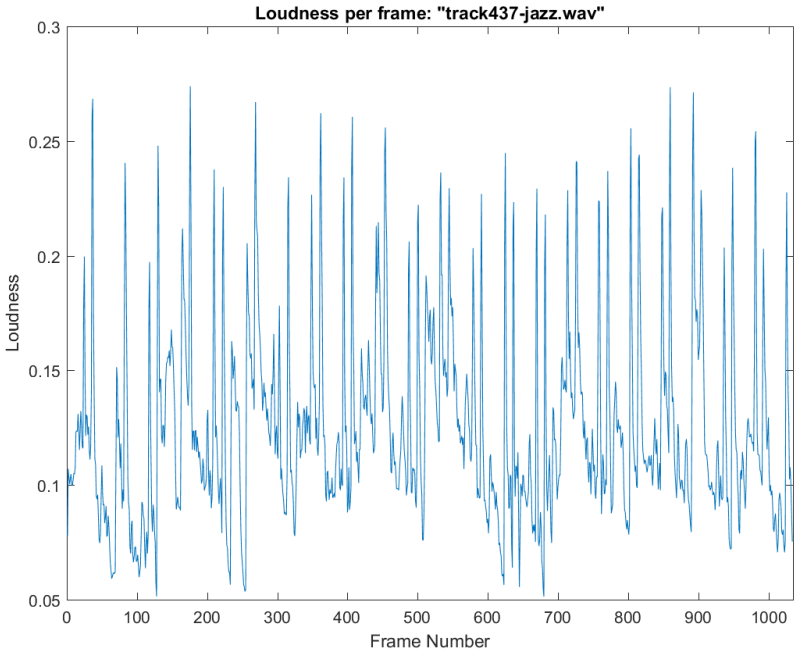

Figure 11: Loudness value per frame, jazz437

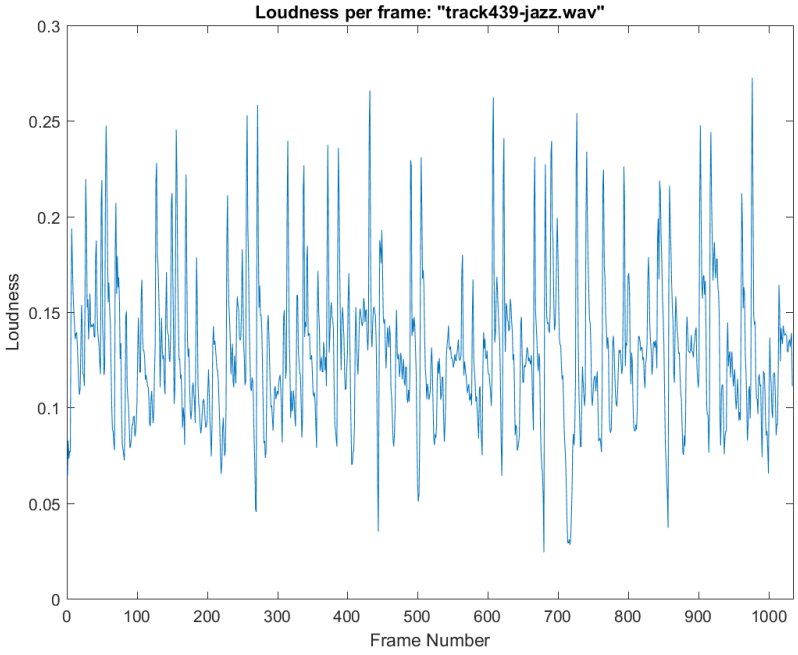

Figure 12: Loudness value per frame, jazz439

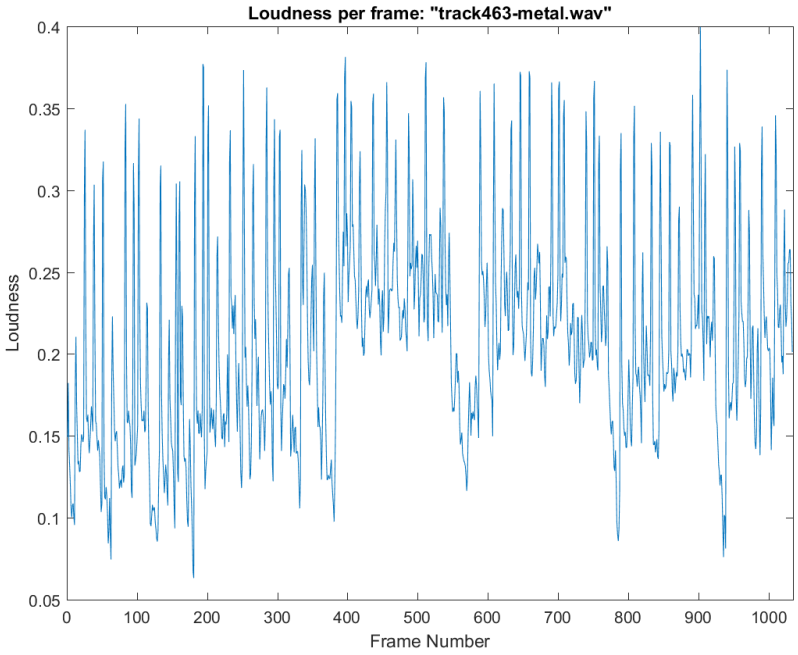

Figure 13: Loudness value per frame, metal463

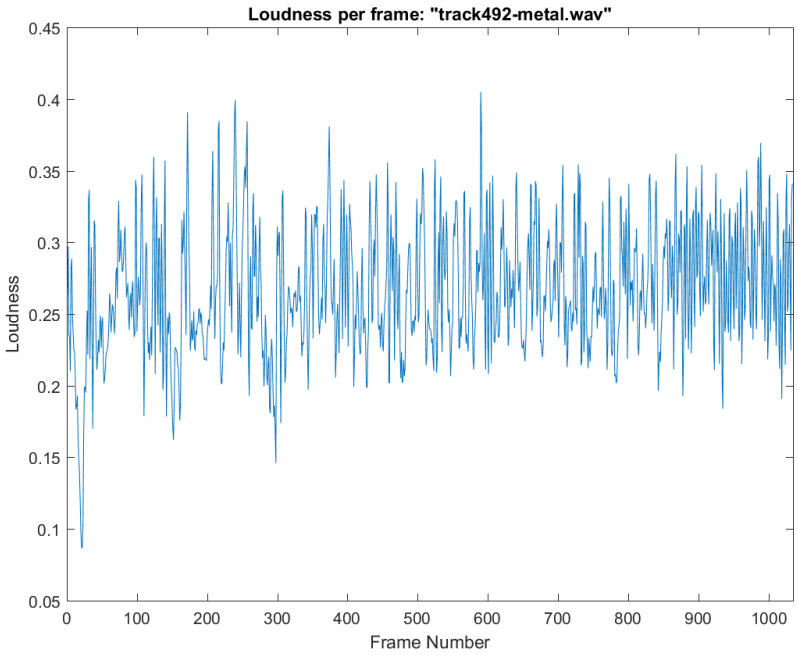

Figure 14: Loudness value per frame, metal492

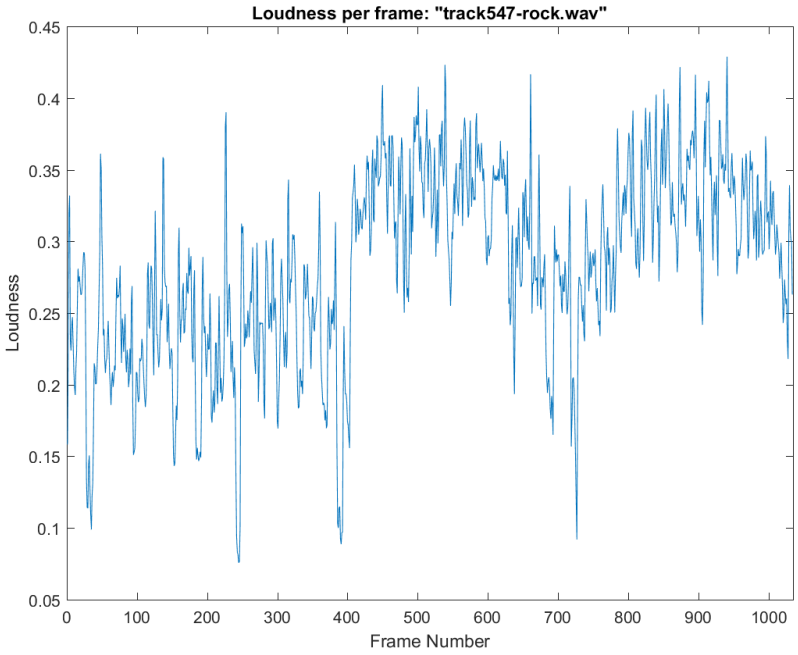

Figure 15: Loudness value per frame, rock547

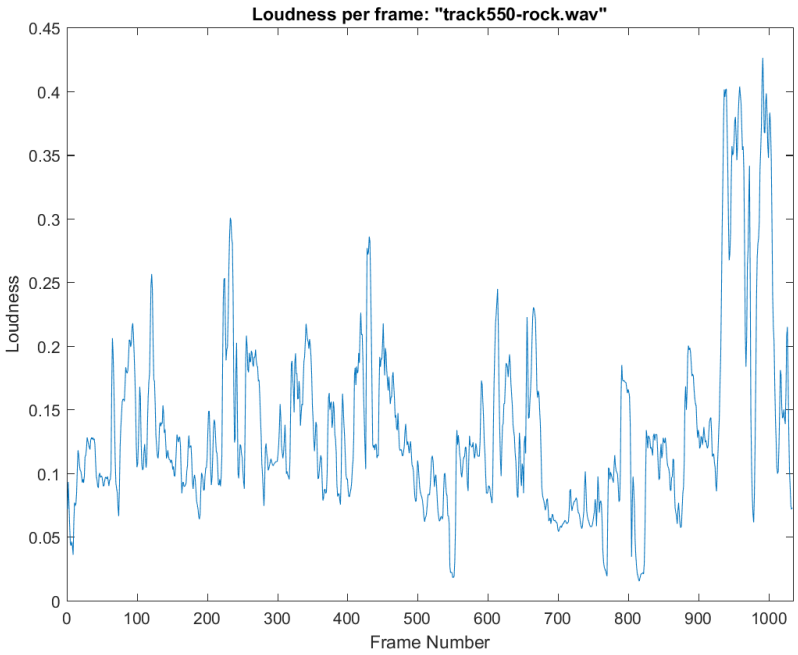

Figure 16: Loudness value per frame, rock550

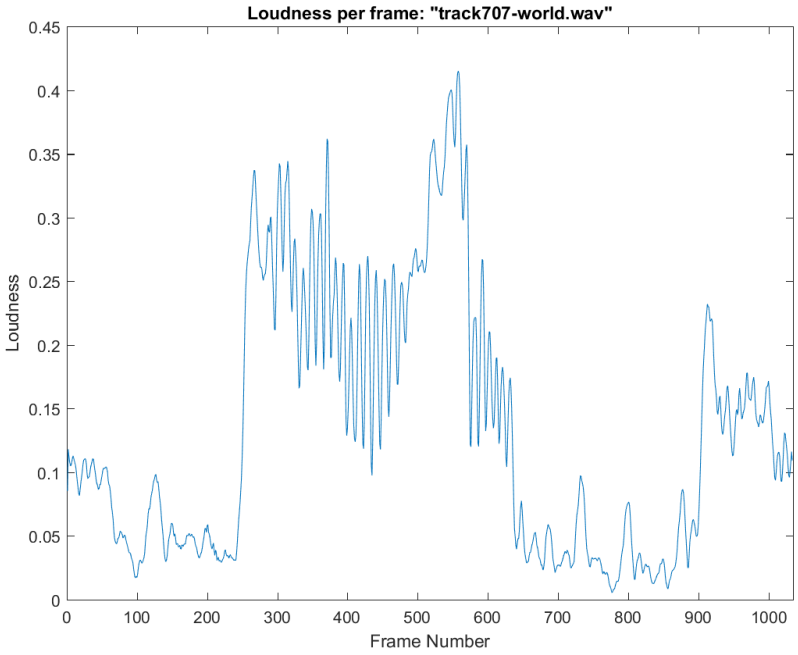

Figure 17: Loudness value per frame, world707

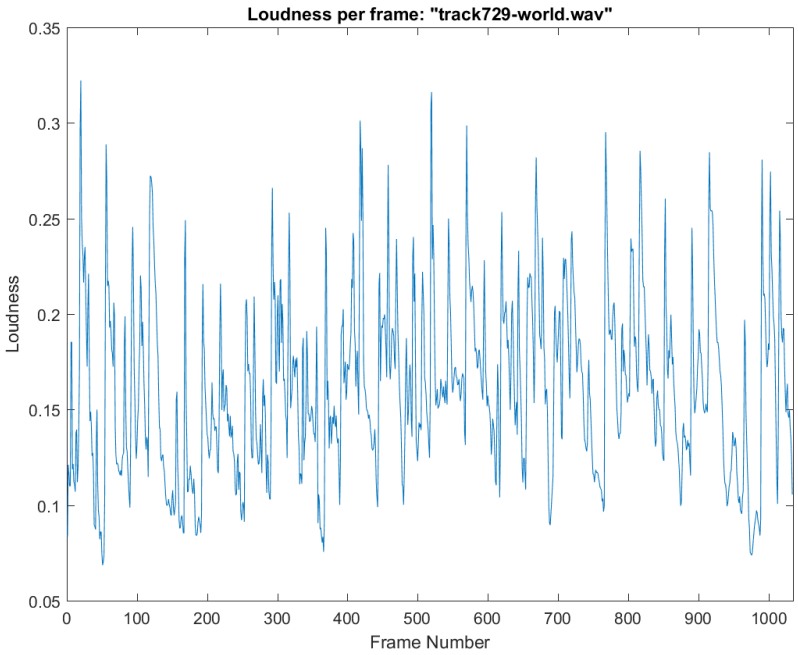

Figure 18: Loudness value per frame, world729

A.2 Zero-Cross Rate

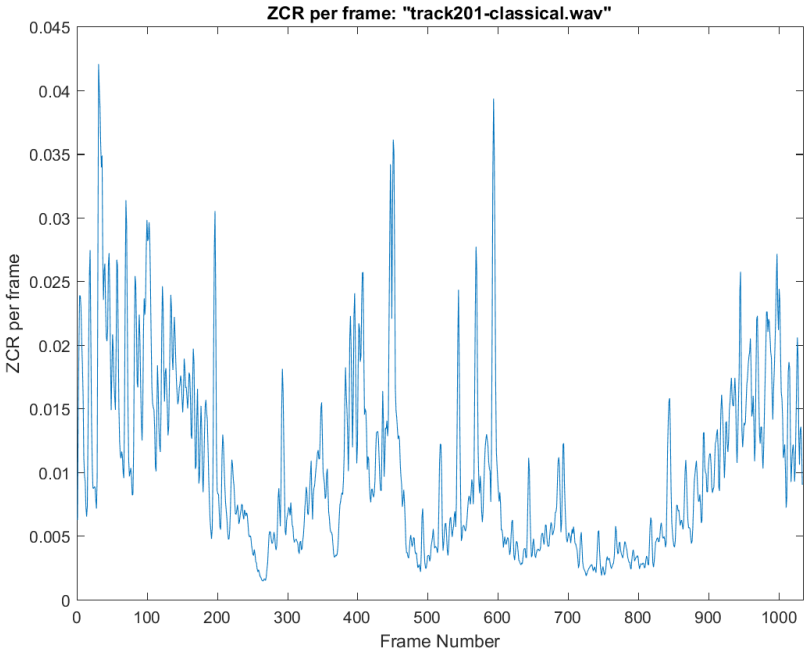

Figure 19: ZCR value per frame, classical201

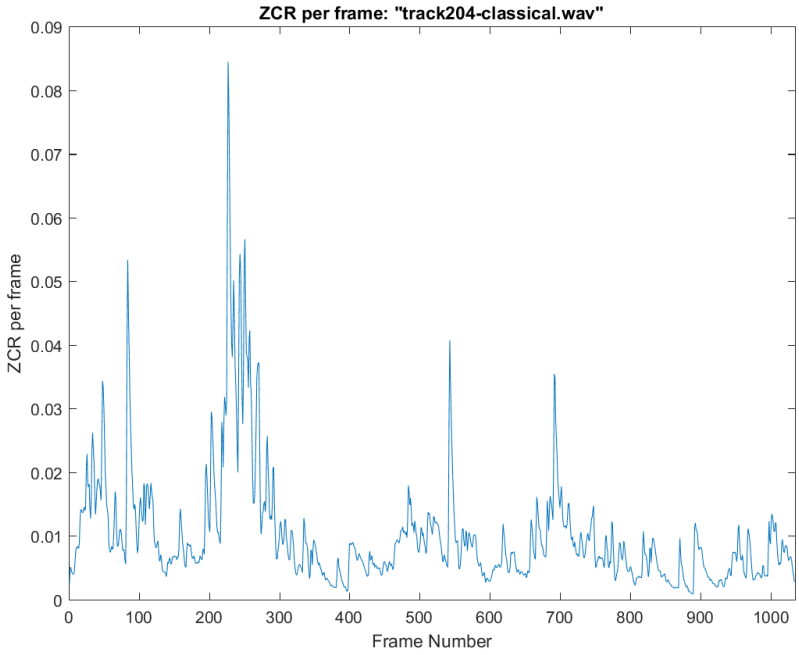

Figure 20: ZCR value per frame, classical204

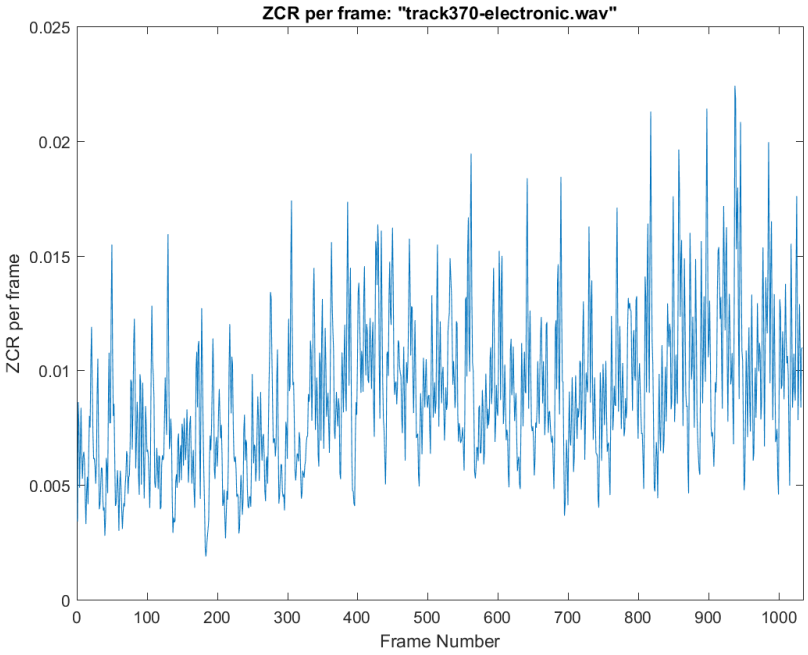

Figure 21: ZCR value per frame, electronic370

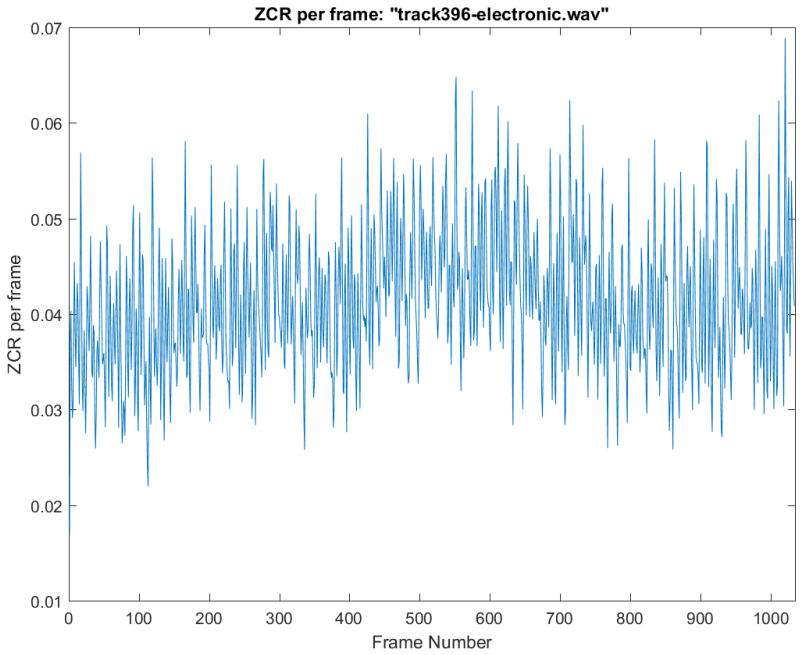

Figure 22: ZCR value per frame, electronic396

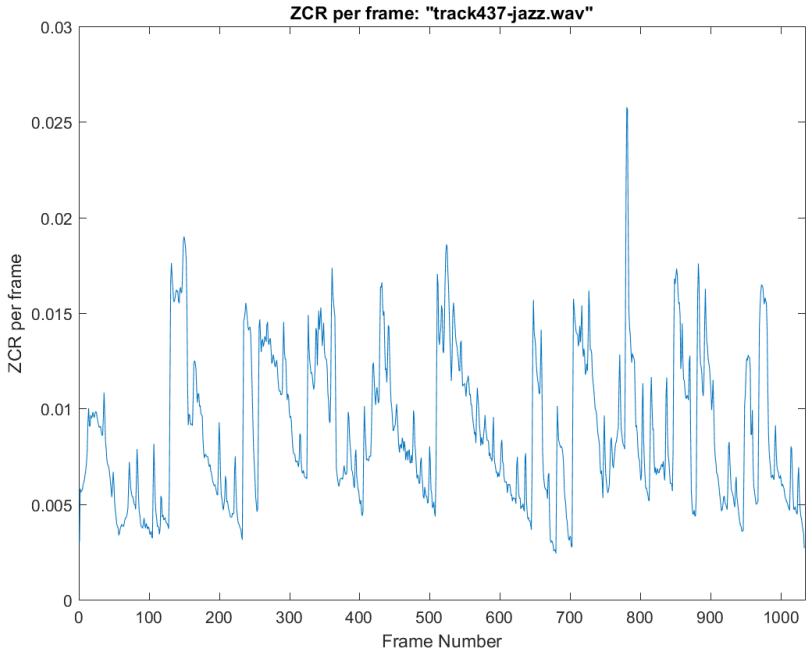

Figure 23: ZCR value per frame, jazz437

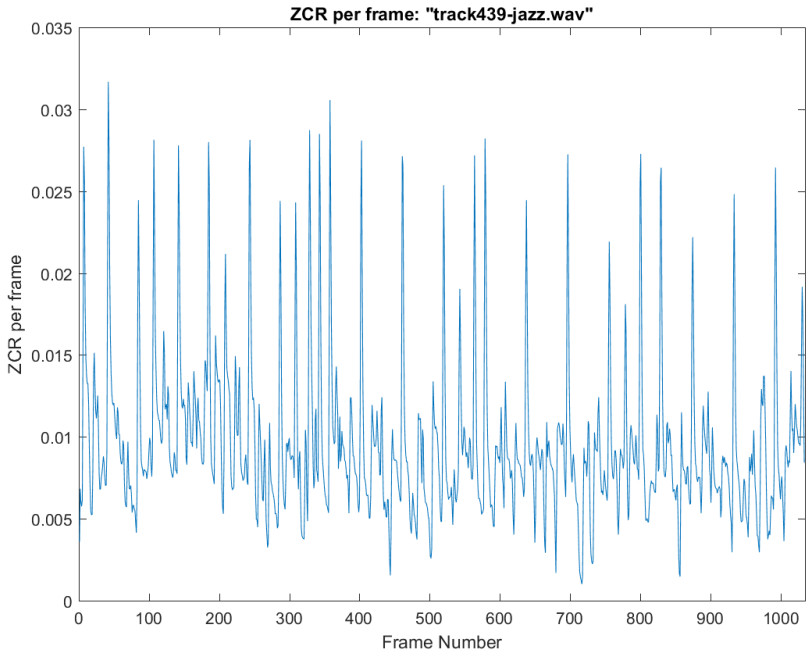

Figure 24: ZCR value per frame, jazz439

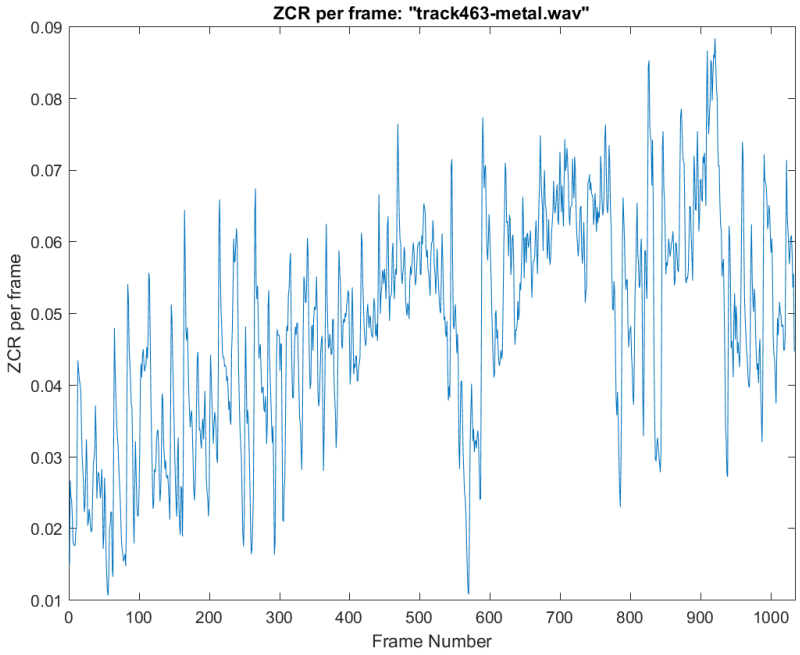

Figure 25: ZCR value per frame, metal463

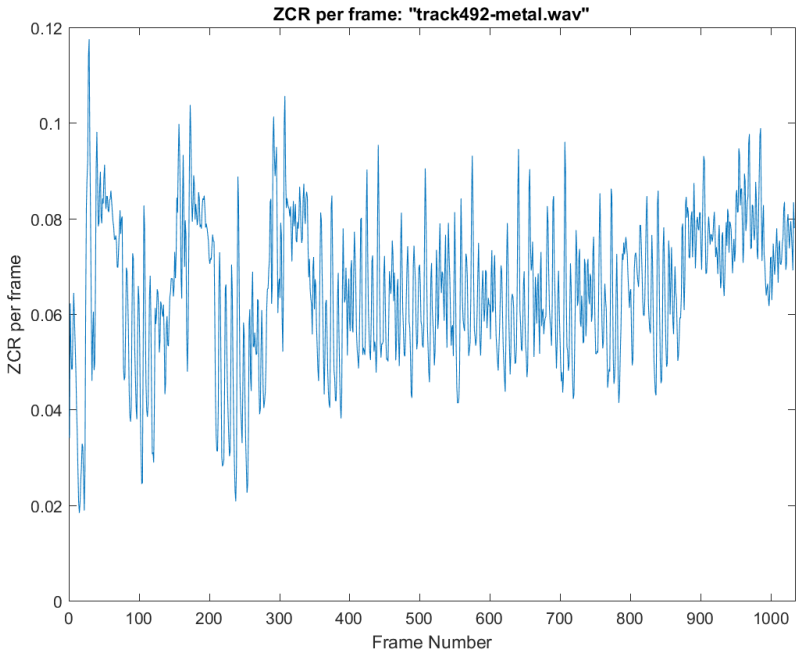

Figure 26: ZCR value per frame, metal492

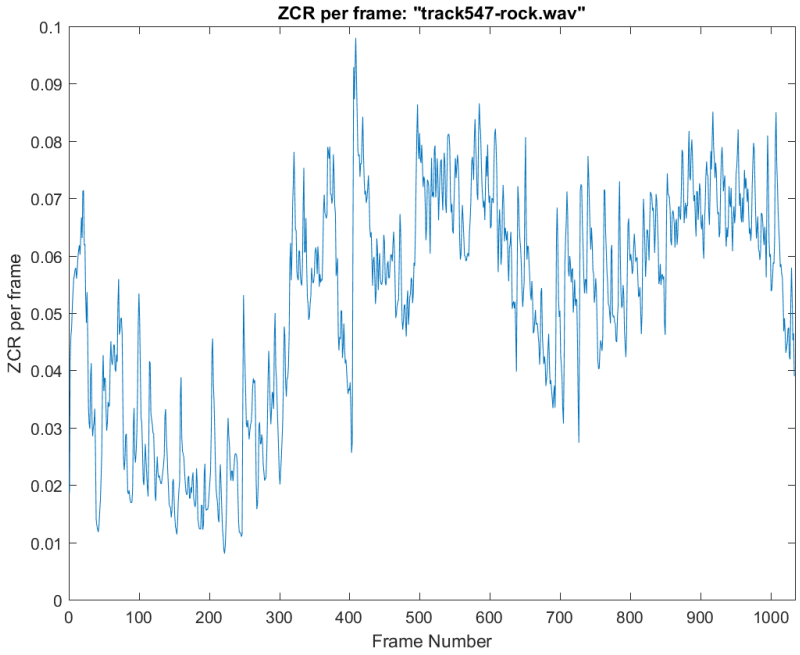

Figure 27: ZCR value per frame, rock547

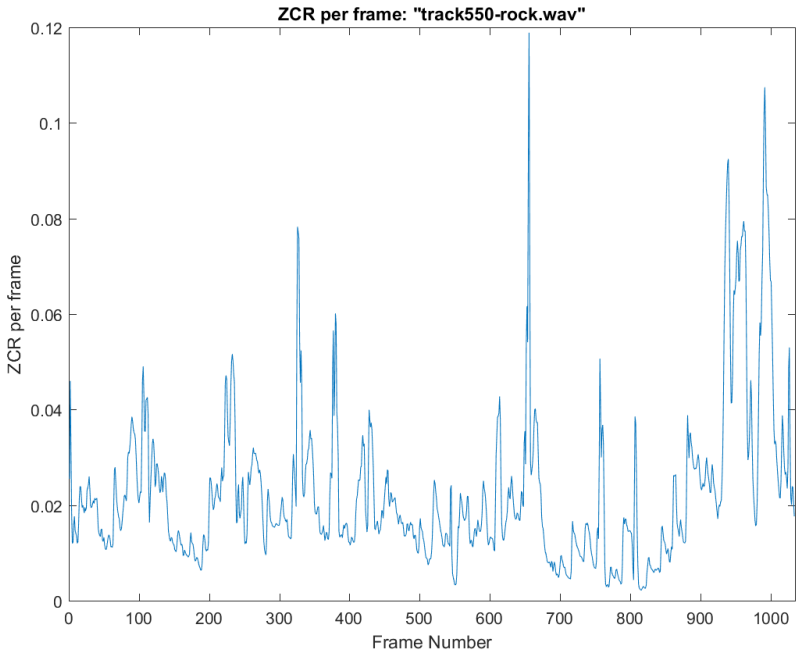

Figure 28: ZCR value per frame, rock550

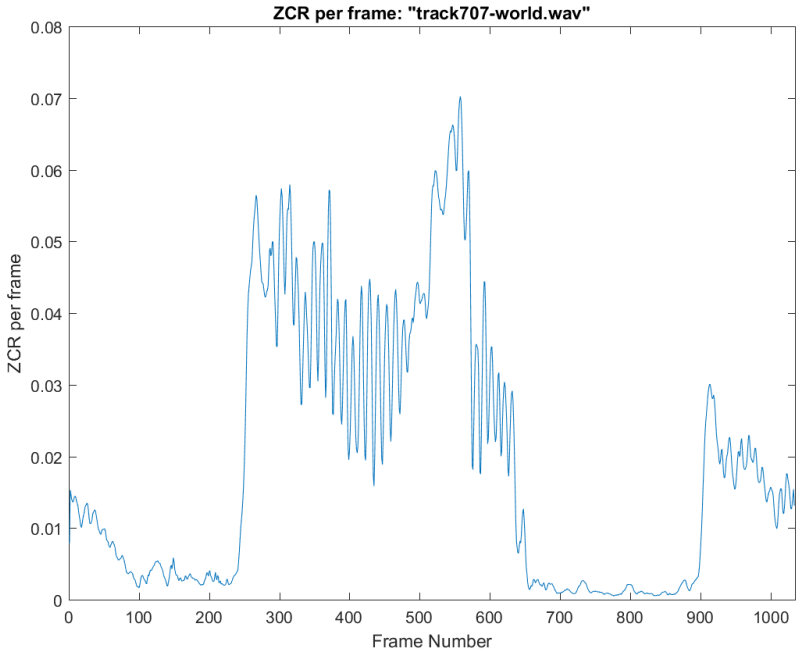

Figure 29: ZCR value per frame, world707

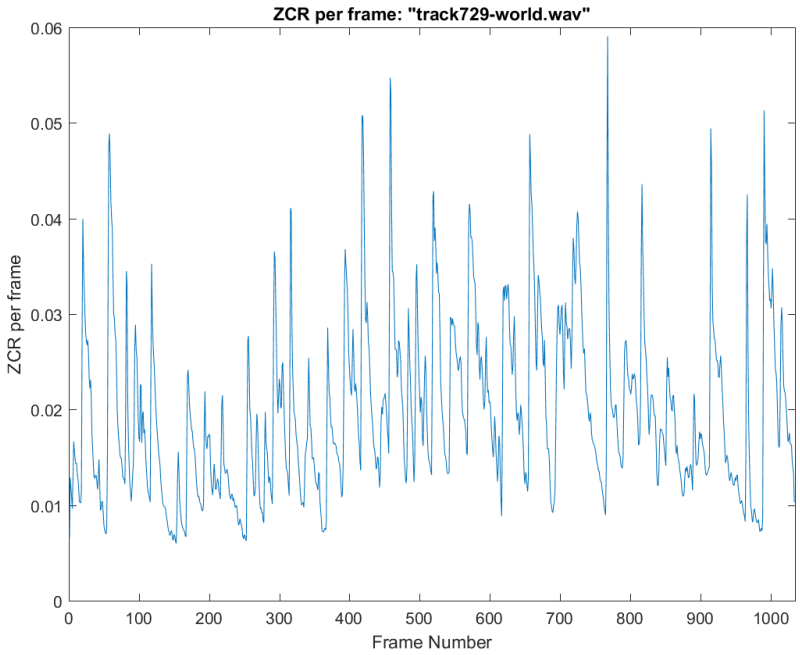

Figure 30: ZCR value per frame, world729

A.3 Timings

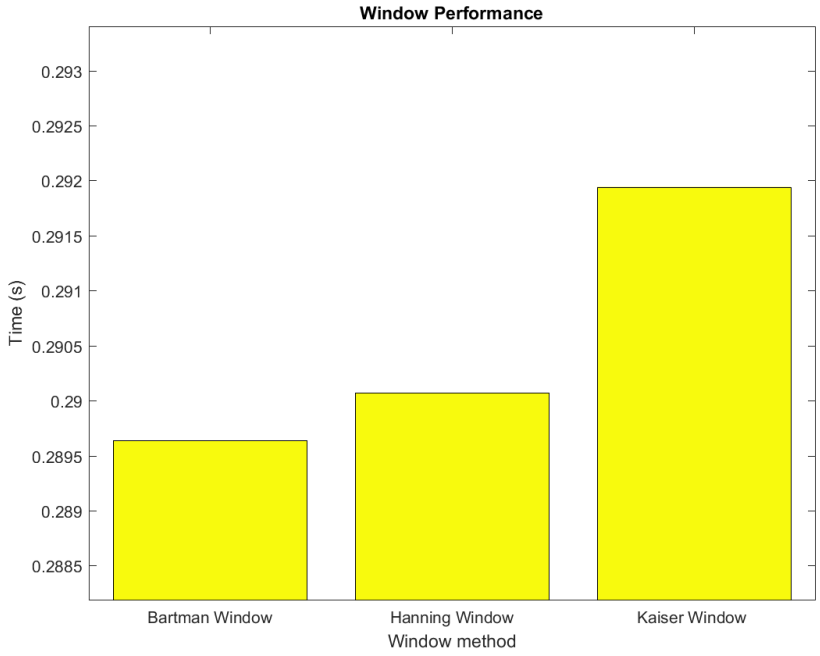

Figure 31: Timing comparison of different windows

B Code

Listing 5: extractSound.m

```
function [ soundExtract,p ] = extractSound( filename, time )
%extractSound Extracts time (in seconds) from the middle of the song
%   Write a MATLAB function that extract T seconds of music from a
%   given track. You will use the MATLAB function audioread to
%   read a track and the function play to listen to the track.
info = audioinfo(filename);
[song,~]=audioread(filename);
if time >= info.Duration
    soundExtract=song;
    p=audioplayer(soundExtract,info.SampleRate);
    return;
elseif time<= 1/info.SampleRate
    error('Too small of a time to sample');
end
samples=time*info.SampleRate;
soundExtract=song(floor(info.TotalSamples/2)-floor(samples/2):1: ...
    floor(info.TotalSamples/2)+floor(samples/2));
p=audioplayer(soundExtract,info.SampleRate);
end
```

Listing 6: loudness.m

```
function [ loudness_data ] = loudness( filename,varargin )
%loudness Computes the standard deviation of the original audio divided
%into frames of size 255 loudness(filename,[franeSize,time,plotBool])
%   The standard deviation of the original audio signal x[n] computed over
%   a frame of size N provides a sense of the loudness
%
%   frameSize = size of frame to compute loudness over; defaults to 255
%   time = duration of song to analyze; defaults to 24 seconds
%   plotBool = whether or not to plot results; default is true
switch nargin
    case 1
        frameSize=512;
        time=24;
        plotBool=1;
    case 2
        frameSize=varargin{1};
        time=24;
        plotBool=1;
    case 3
        frameSize=varargin{1};
        time=varargin{2};
        plotBool=1;
    case 4
        frameSize=varargin{1};
        time=varargin{2};
        plotBool=varargin{3};
    otherwise
        error('loudness:argChk', 'Wrong number of input arguments');
end
[y,~]=extractSound( filename, time );
frames_data = buffer(y,frameSize,ceil(frameSize/2));
loudness_data=std(frames_data,0,1);
if plotBool==1
    p=plot(1:length(loudness_data),loudness_data(1,:));
    title(['Loudness per frame: "' filename '"']);
    xlim([0 length(loudness_data)+1]);
    xlabel('Frame Number');
    ylabel('Loudness');
    saveas(p,['Loudness_' filename(1:end-4) '.png']);
```

```
end
end
```

Listing 7: loudnessPlot.m

```
% plot loudness features
close all;
figure(1)
loudness('track201-classical.wav');
figure(2)
loudness('track204-classical.wav');
figure(3)
loudness('track370-electronic.wav');
figure(4)
loudness('track396-electronic.wav');
figure(5)
loudness('track437-jazz.wav');
figure(6)
loudness('track439-jazz.wav');
figure(7)
loudness('track463-metal.wav');
figure(8)
loudness('track492-metal.wav');
figure(9)
loudness('track547-rock.wav');
figure(10)
loudness('track550-rock.wav');
figure(11)
loudness('track707-world.wav');
figure(12)
loudness('track729-world.wav');
```

Listing 8: zeroCross.m

```
function [ ZCR_data ] = zeroCross( filename )
%zeroCross Computes the zero cross rate of the audio file
%   The Zero Crossing Rate is the average number of times the audio signal
%   crosses the zero amplitude line per time unit. The ZCR is related to...
%   the pitch height, and is also correlated to the noisiness of the...
%   signal.
frameSize=512;
time=24;
% info = audioinfo(filename);
[y,~]=extractSound( filename, time ); % Operate on middle 24 seconds
frames_data = buffer(y,frameSize,ceil(frameSize/2));
ZCR_data=zeros(1,length(frames_data));
ZCR_data(1:end)=sum(abs(diff(frames_data(1:end,:))))/length(frames_data);

p=plot(1:length(ZCR_data),ZCR_data(1,:));
title(['ZCR per frame: "' filename '"']);
xlim([0 length(ZCR_data)+1]);
xlabel('Frame Number');
ylabel('ZCR per frame');
saveas(p,['ZCR_' filename(1:end-4) '.png']);
end
```

Listing 9: zeroPlot.m

```
% plot zeroCross features
close all;
figure(1)
zeroCross('track201-classical.wav');
figure(2)
zeroCross('track204-classical.wav');
```

```
figure(3)
zeroCross('track370-electronic.wav');
figure(4)
zeroCross('track396-electronic.wav');
figure(5)
zeroCross('track437-jazz.wav');
figure(6)
zeroCross('track439-jazz.wav');
figure(7)
zeroCross('track463-metal.wav');
figure(8)
zeroCross('track492-metal.wav');
figure(9)
zeroCross('track547-rock.wav');
figure(10)
zeroCross('track550-rock.wav');
figure(11)
zeroCross('track707-world.wav');
figure(12)
zeroCross('track729-world.wav');
```

Listing 10: windows.m

```
function [Xn] = windows( filename, window )
%windows Implement the computation of the windowed Fourier transform of y
% start=cputime;
frameSize=512;
switch window
    case 1
        w=bartlett(frameSize); % Bartlett window
    case 2
        w=hann(frameSize); % Hann (Hanning) window
    case 3
        w=kaiser(frameSize,0.5); % Kaiser window
    otherwise
        w=kaiser(frameSize,0.5); % defaults to Kaiser window
end
[x,~]=audioread(filename);
xn=buffer(x,frameSize);
Y=zeros(size(xn));
for i=1:length(xn)
    Y(:,i)=fft(xn(:,i).*w);
end
K=frameSize/2+1;
Xn=size(Y);
for i=1:length(xn)
   Xn(1:K,i)=Y(1:K,i);
end
end
```

Listing 11: windowsTiming.m

```
% Windows timing
fileArray=cellstr(['440Amp1.wav  '; '440Amp5.wav  ';'11025Amp1.wav';...
    '11025Amp5.wav'; '14080Amp1.wav'; '14080Amp5.wav']);
timings=zeros(1,length(fileArray)*3);
for i=1:length(fileArray)
    for j=1:3
        g=@()audioread(['' fileArray{i} '']);
        h=@()windows(['' fileArray{i} ''],j);
        timings((i-1)*3+j)=timeit(h)-timeit(g);
    end
end

bartTime=timings(1:3:end);bartTotal=sum(bartTime);
```

```
14  hannTime=timings(2:3:end);hannTotal=sum(hannTime);
15  kaiserTime=timings(3:3:end);kaiserTotal=sum(kaiserTime);
16  x=1:length(fileArray);
17  a=bar([bartTime;hannTime;kaiserTime],'stacked');
18  ax=gca;
19  ylim([min([bartTotal,hannTotal,kaiserTotal])*0.995,...
20      max([bartTotal,hannTotal,kaiserTotal])*1.005]);
21  ylabel('Time (s)');
22  xlabel('Window method');
23  set(gca,'XTickLabel',['Bartman Window'; 'Hanning Window';...
24      'Kaiser Window ' ]);
25  title('Window Performance');
26  saveas(gca,'windowsTiming.png');
27  close all;
```