# 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

```matlab
1  function [ soundExtract,p ] = extractSound( filename, time )
2  %extractSound Extracts time (in seconds) from the middle of the song
3  %   Write a MATLAB function that extract T seconds of music from a
4  %   given track. You will use the MATLAB function audioread to
5  %   read a track and the function play to listen to the track.
6  info = audioinfo(filename);
7  [song,~]=audioread(filename);
8  if time >= info.Duration
9      soundExtract=song;
10     p=audioplayer(soundExtract,info.SampleRate);
11     return;
12 elseif time<= 1/info.SampleRate
13     error('Too small of a time to sample');
14 end
15 samples=time*info.SampleRate;
16 soundExtract=song(floor(info.TotalSamples/2)-floor(samples/2):1: ...
17     floor(info.TotalSamples/2)+floor(samples/2));
18 p=audioplayer(soundExtract,info.SampleRate);
19 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

```matlab
30  [y,~]=extractSound( filename, time );
31  frames_data = buffer(y,frameSize,ceil(frameSize/2));
32  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
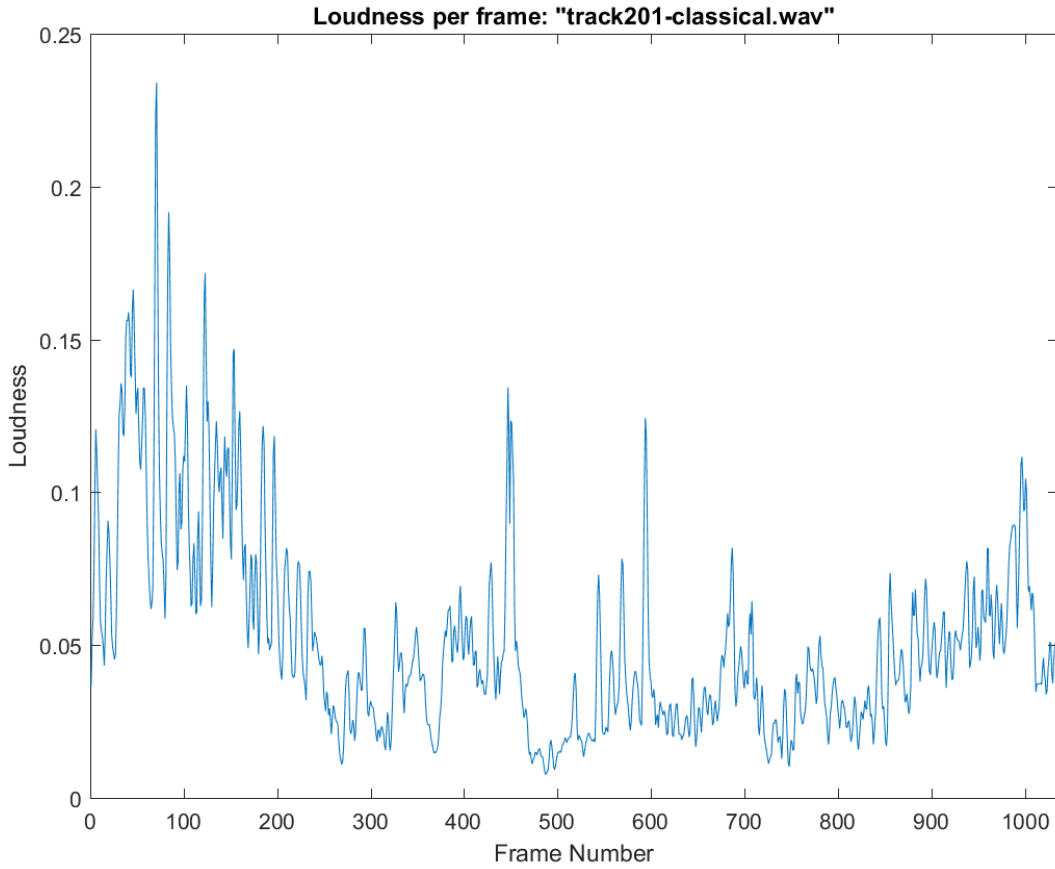
Figure 1: Loudness value per frame

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.

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

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);
```
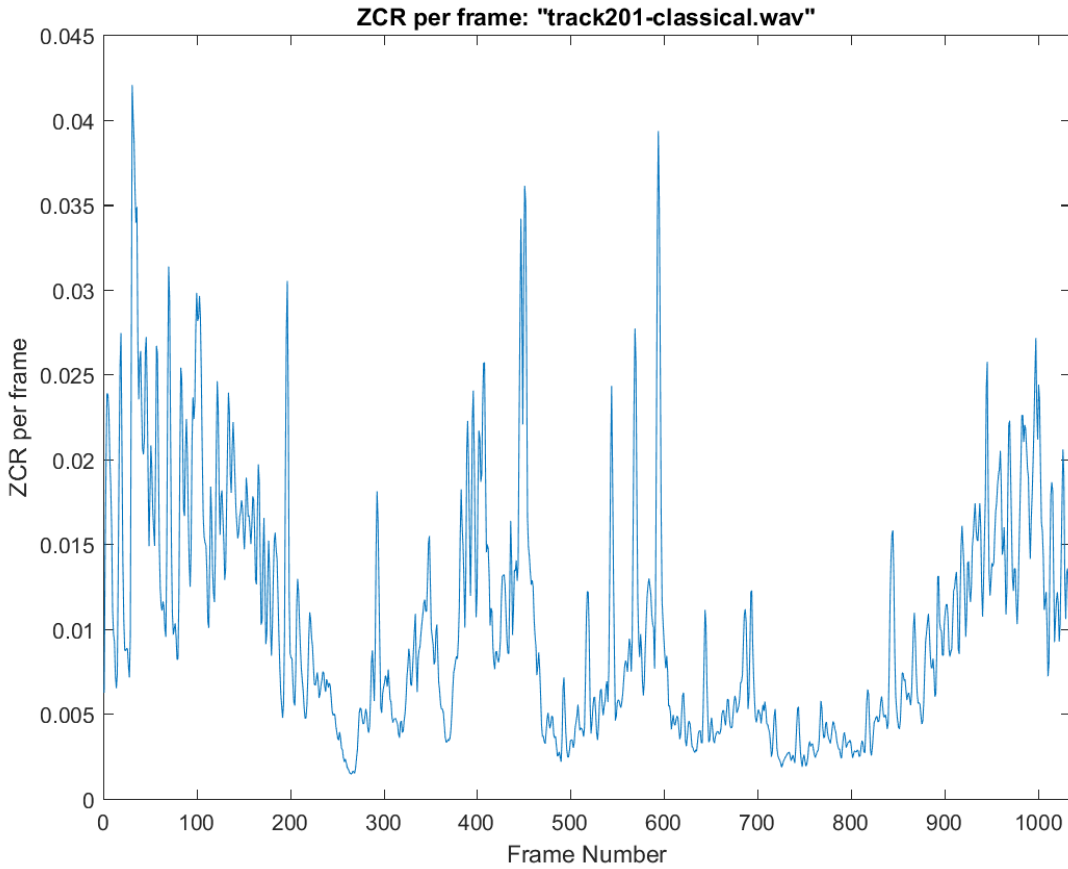


Figure 2: Average ZCR per frame

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:

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

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 $\omega$. 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 $y[n]$.

$$\text{if } y[n] = x[n]w[n - N],$$

$$\text{then } Y(e^{j\omega}) = \frac{1}{2\pi} \int_{-\pi}^{\pi} X(e^{j\theta}) \cdot (H(e^{j(\omega-\theta)}) \cdot e^{-j\omega N})d\theta$$

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

8. Compute the spectrogram of an audio track as follows:

   (a) Decompose a track into a sequence of $N_f$ overlapping *frames* of size $N$. The overlap between two frames should be $N/2$.

   (b) Compute the magnitude squared of the Fourier transform, $|X(k)|^2, k = 1, \ldots, K$ over each frame $n$.

   (c) Display the Fourier transform of all the frames in a matrix of size $K \times N_f$.

   You will experiment with different audio tracks, as well as pure sinusoidal tones. Do the spectrograms like what you hear?

In the rest of the lab, we will be using a Kaiser window to compute the Fourier transform, as explained in Equation 3.

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.
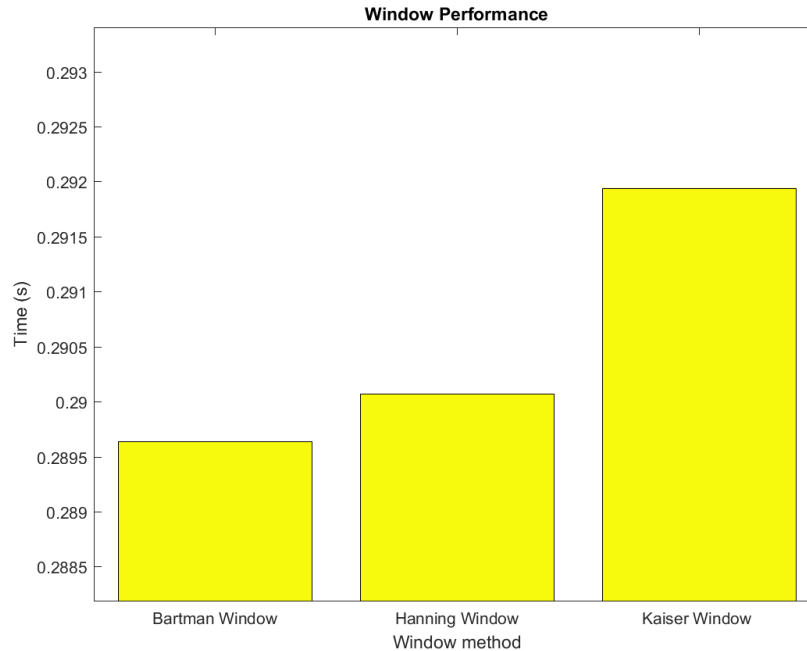


Figure 3: Timing comparison of different windows

Listing 4: windows.m

```matlab
15   [x,~]=extractSound(filename,24);
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.
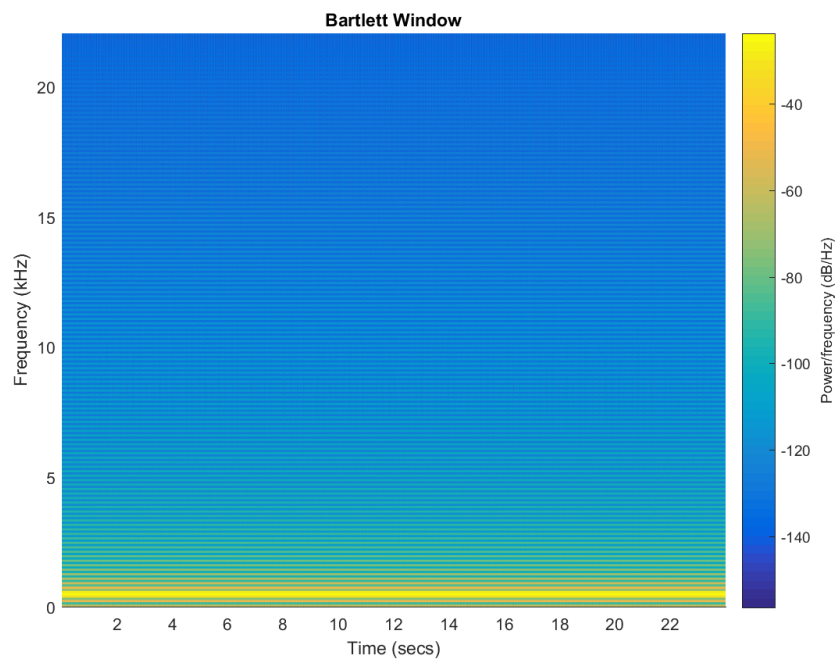


Figure 4: Spectrogram of Bartlett Window
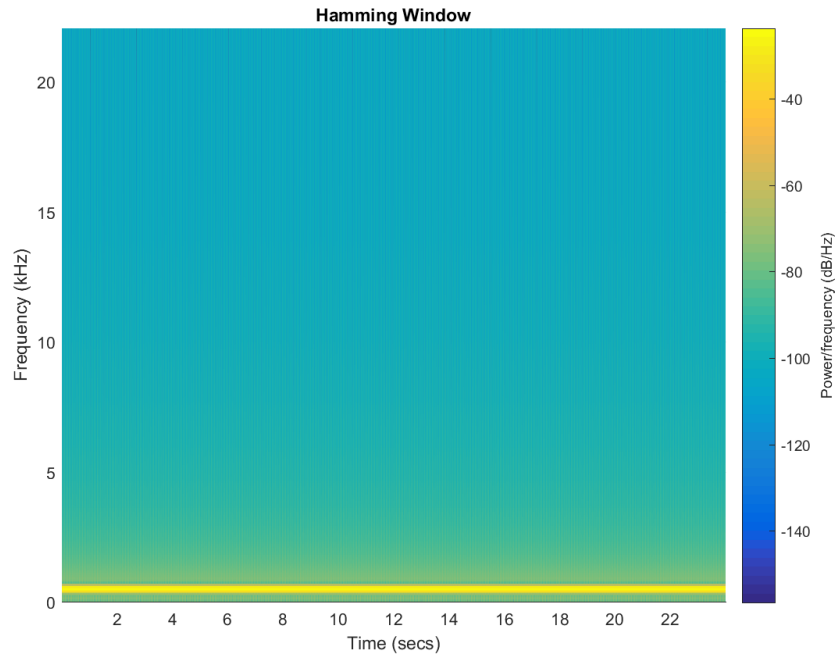
Figure 5: Spectrogram of Hamming Window
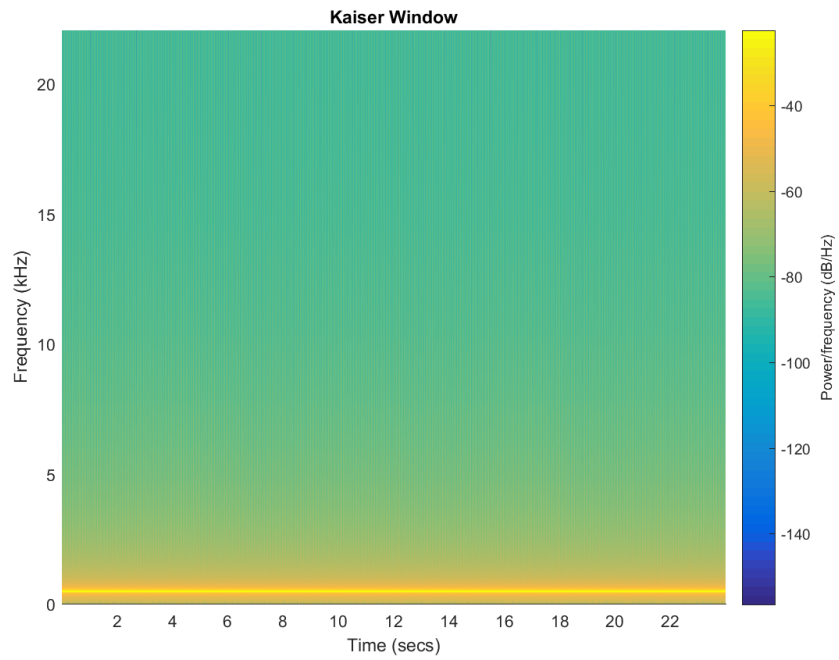


Figure 6: Spectrogram of Kaiser Window

In addition to being fairly fast, the Bartlett window seemed to be the most effective window in that its power to frequency ratio was very low away from the target frequency of 440 Hz.

In addition to a song having low-level time-domain characteristics, low-level frequency-domain characteristics exist as well. With modern DSP chips, computing an FFT is no longer that big of an issue for DSP engineers. More plot for frequency centroids and frequency spreads can be found in Appendices A.4 and A.5, respectively.
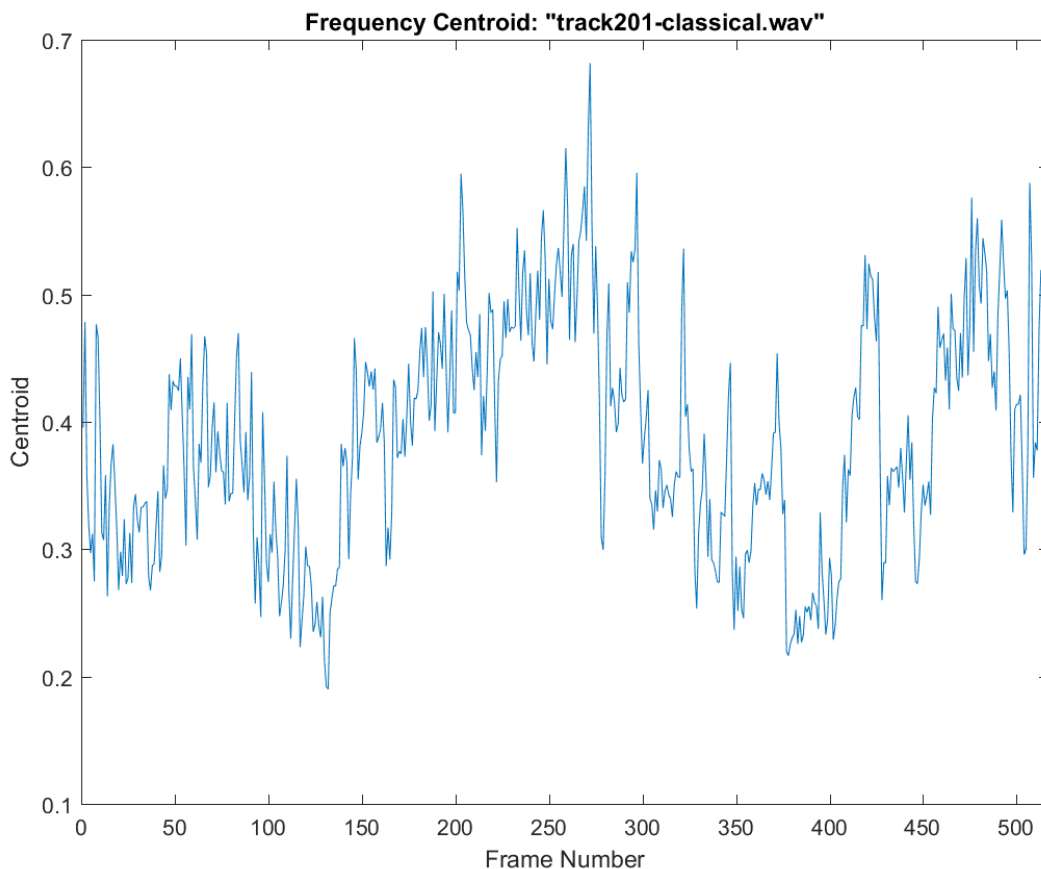


Figure 7: Frequency Centroid
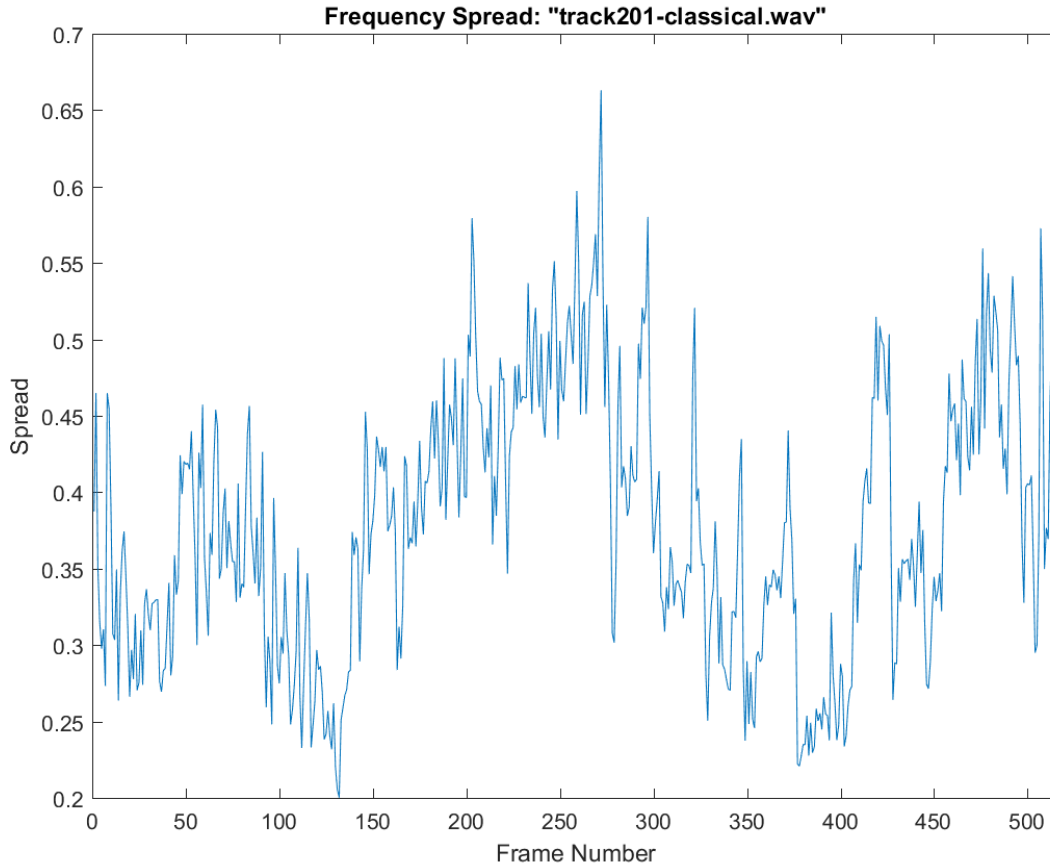
Figure 8: Frequency Spread

Just as the in the time-domain analyses, the frequency-domain functions appear to be most helpful in identifying *classical* and *jazz* music. There doesn't seem to be any advantage in using **centroid** vs **spread** in identifying music genres. One method may eventually be better, but it is unclear at the moment.

## 5    Basic Psychoacoustic Quantities

Humans are very good at identifying music genres. Our problem is that we are too slow at it though. Humans generally identify a specific genre of music with its subjective features such as timbre, melody, harmony, rhythm, tempo, mood, lyrics, etc. In this section, we will focus solely on the aspects that are mathematically quantifiable.

---

**Assignment**

11. Implement the computation of the triangular filterbanks $H_p, p = 1, \ldots, N_B$. Your function will return an array **fbank** of size $N_B \times K$ such that **fbank**$(\mathbf{p}, :)$ contains the filter bank $H_p$.

12. Implement the computation of the mfcc coeffecients, as defined by:

$$\text{mfcc}[p] = \sum_{k=1}^{K} |H_p(k)X_n(k)|^2 \tag{7}$$

---

10

# A Figures

## A.1 Loudness
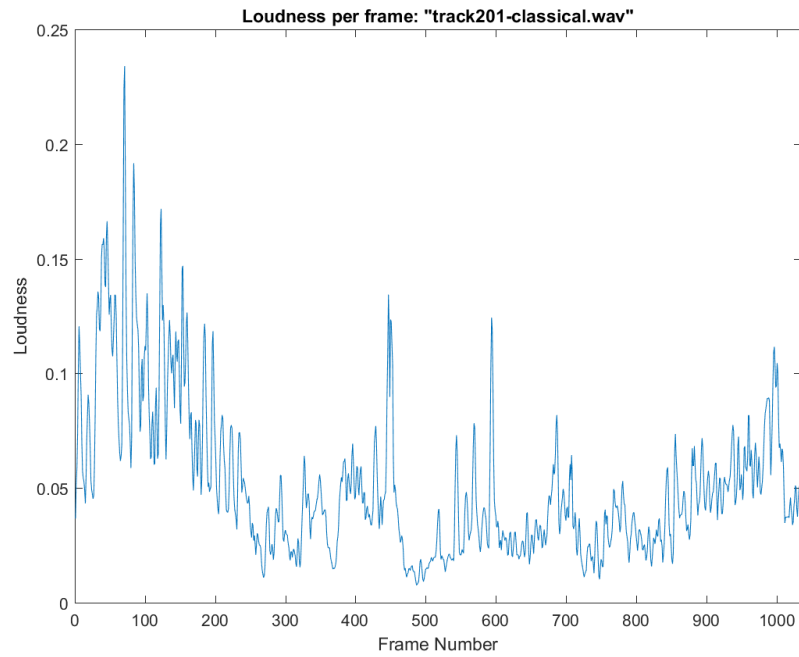


Figure 9: Loudness value per frame, classical201
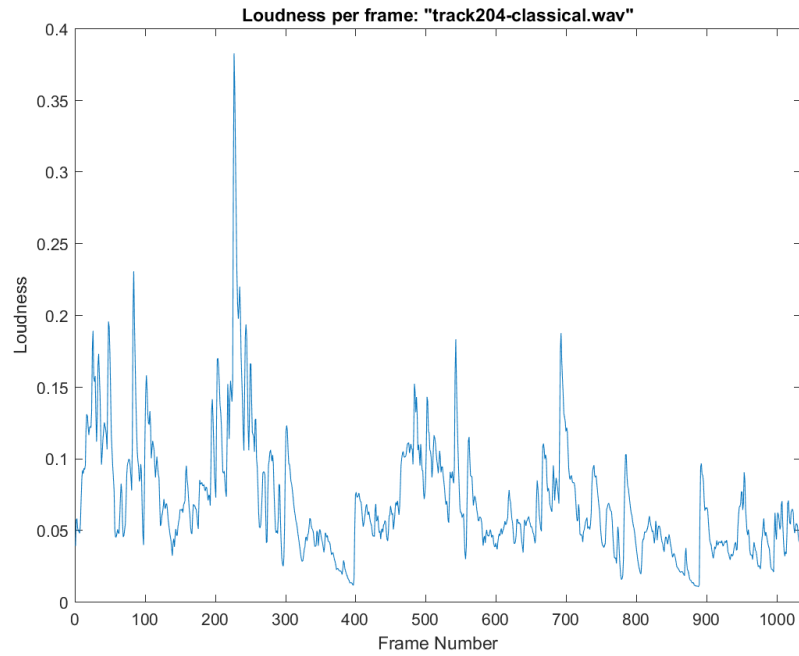


Figure 10: Loudness value per frame, classical204
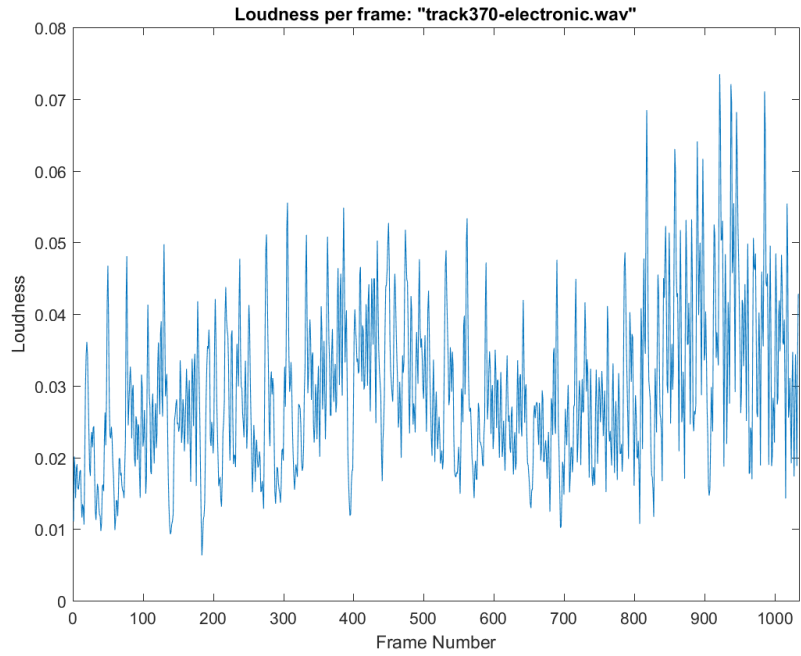
Figure 11: Loudness value per frame, electronic370
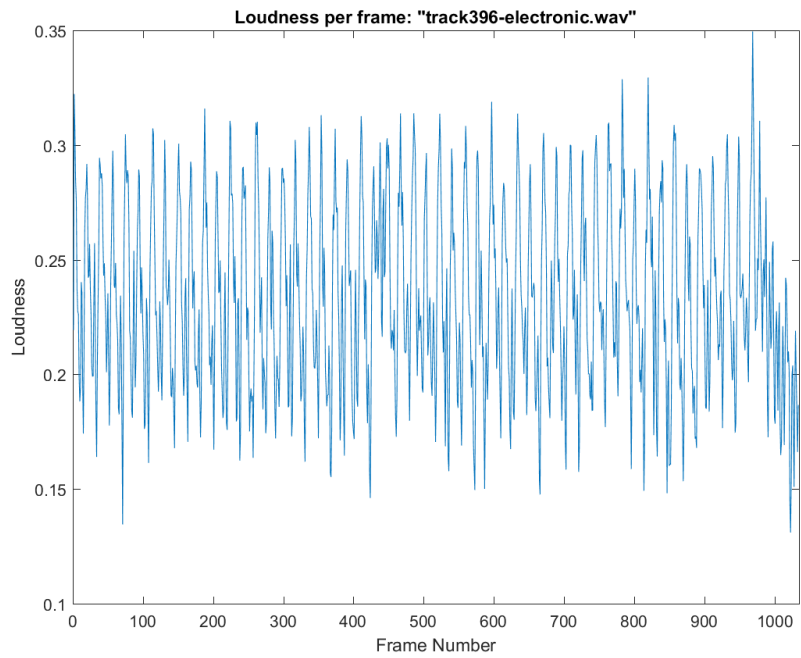


Figure 12: Loudness value per frame, electronic396
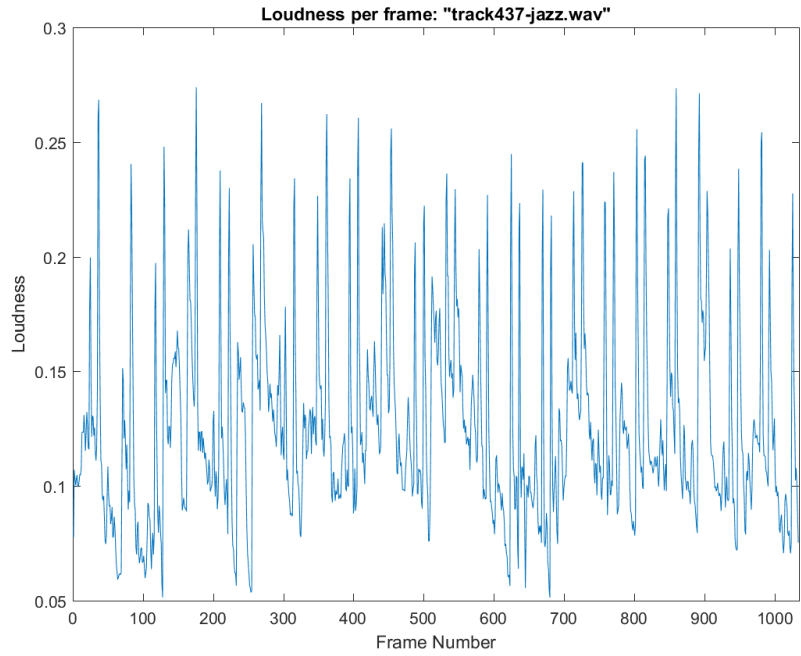
Figure 13: Loudness value per frame, jazz437
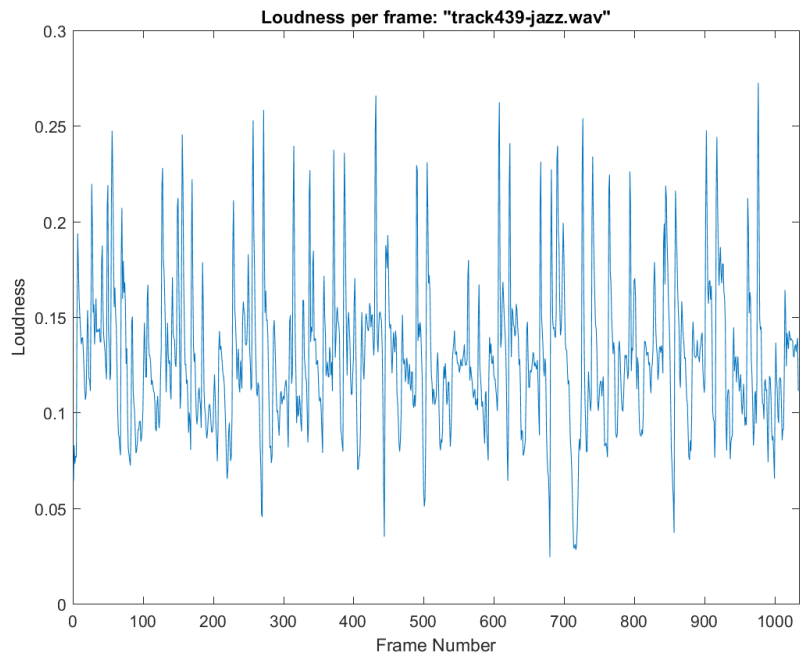


Figure 14: Loudness value per frame, jazz439
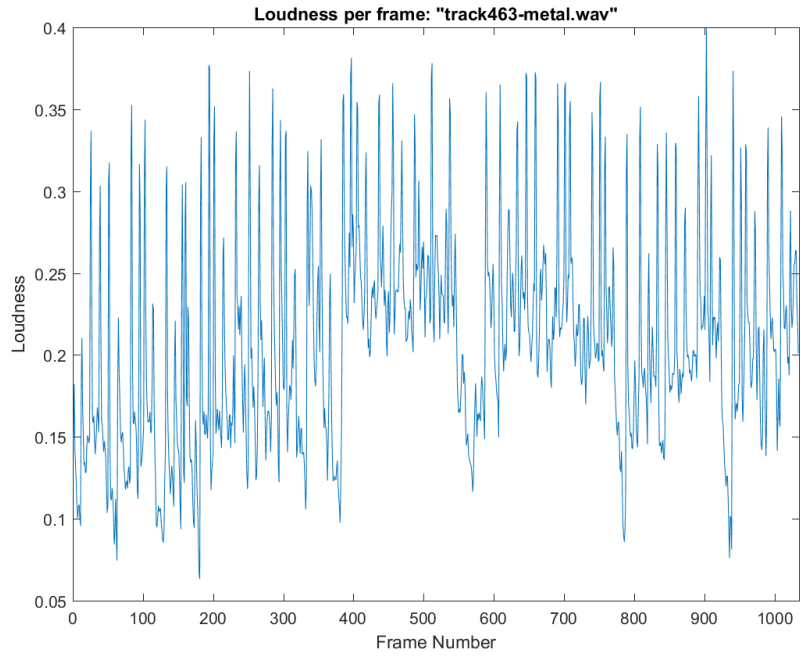
Figure 15: Loudness value per frame, metal463
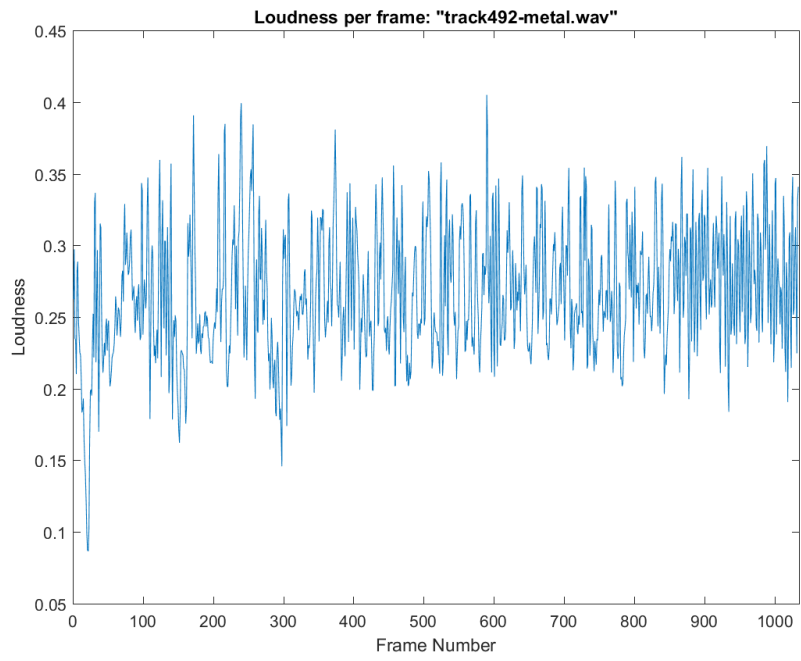


Figure 16: Loudness value per frame, metal492
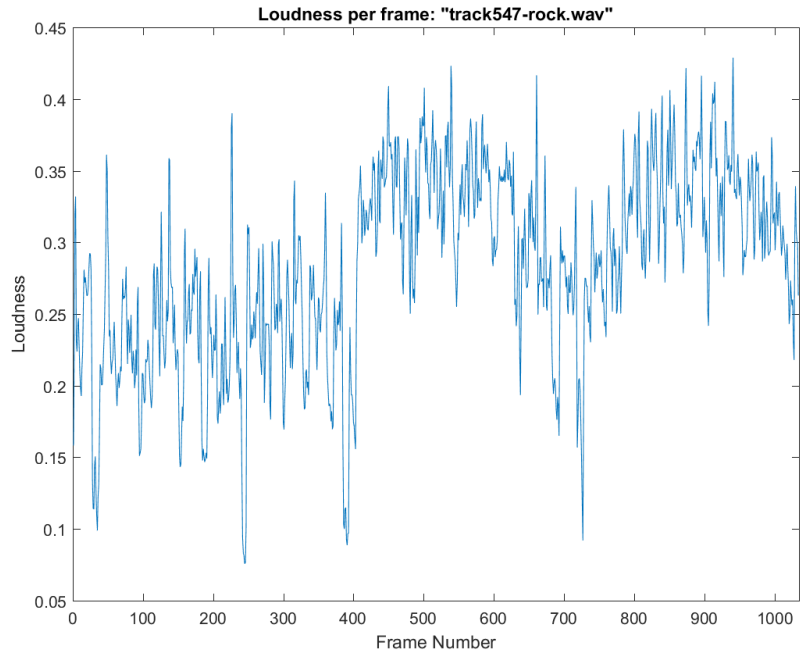
Figure 17: Loudness value per frame, rock547
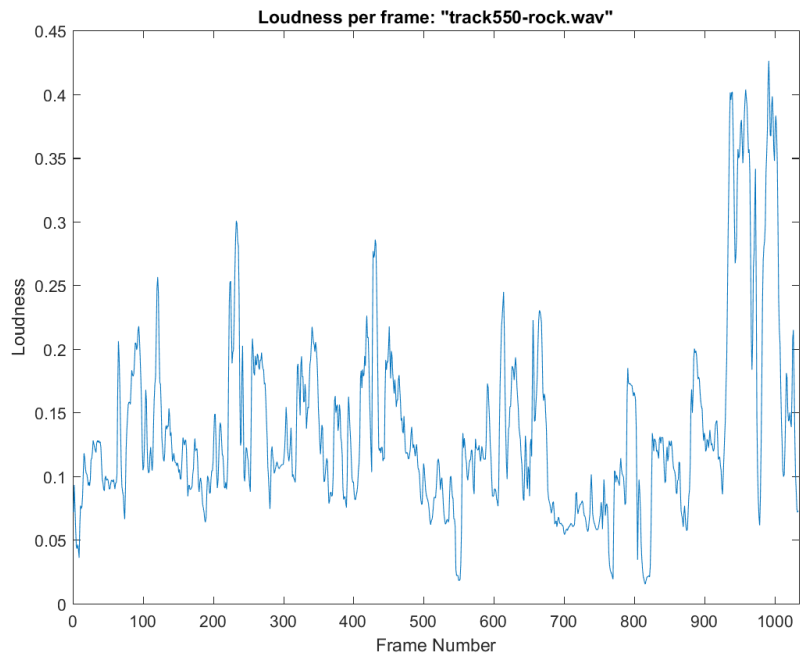


Figure 18: Loudness value per frame, rock550
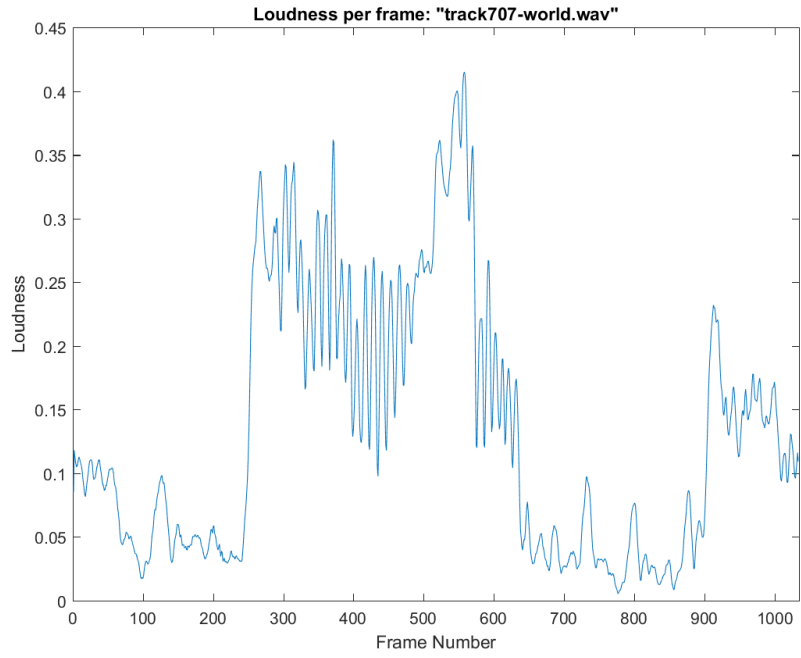
Figure 19: Loudness value per frame, world707
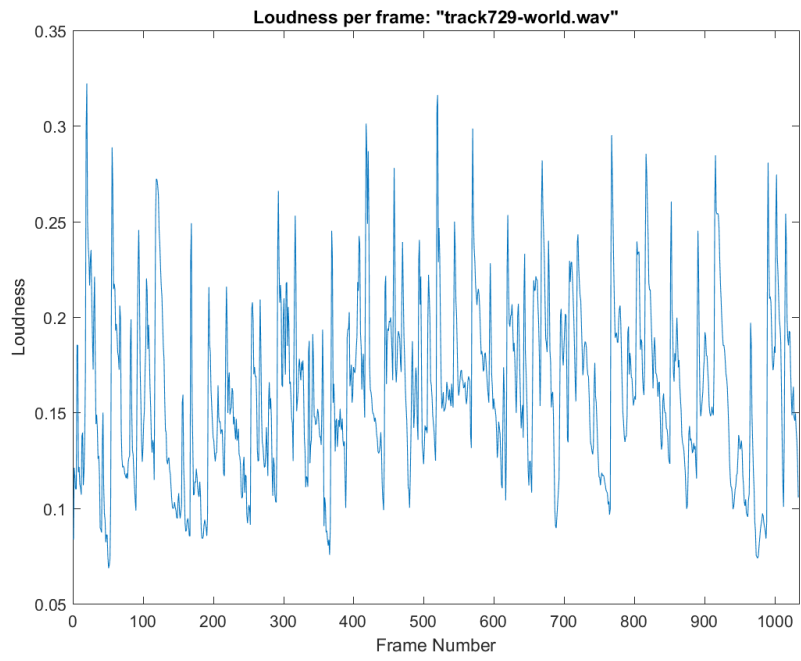


Figure 20: Loudness value per frame, world729

## A.2 Zero-Cross Rate
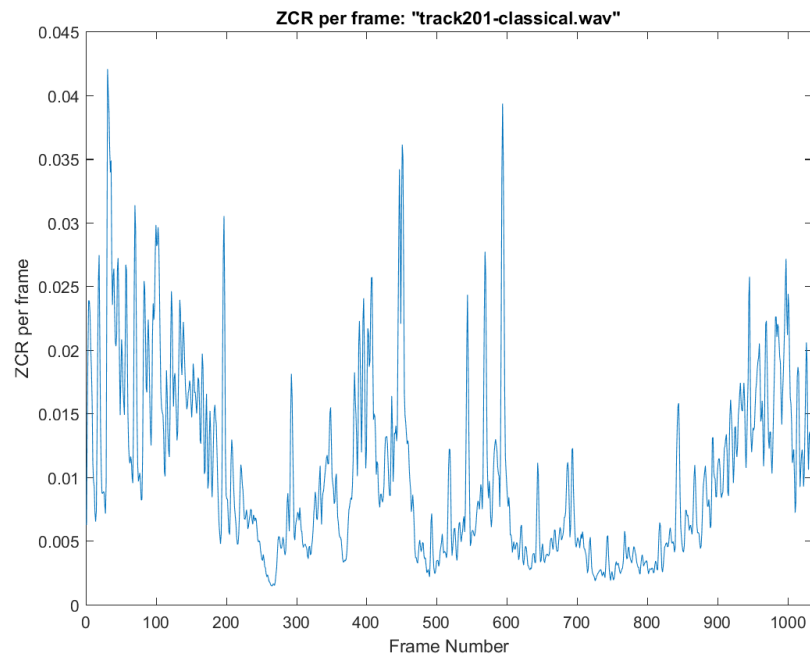


Figure 21: ZCR value per frame, classical201
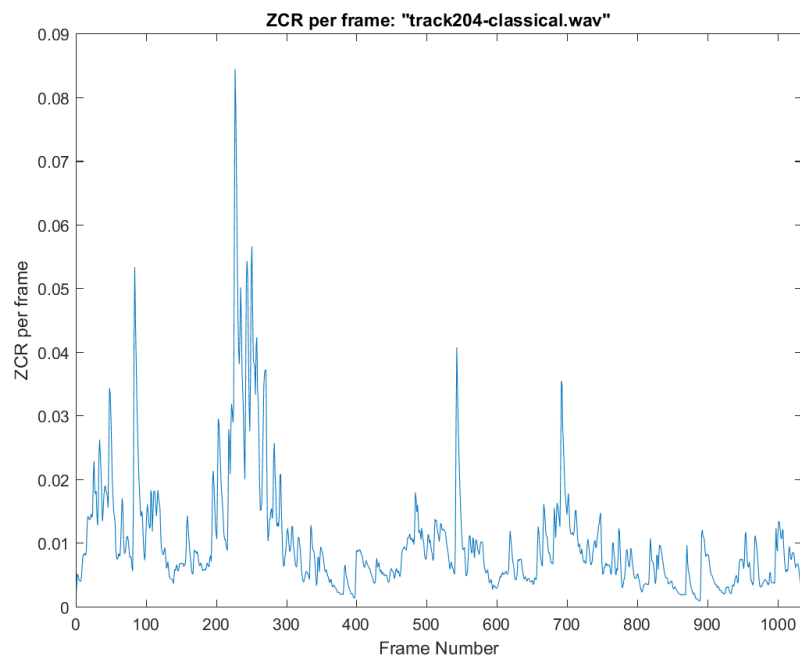


Figure 22: ZCR value per frame, classical204
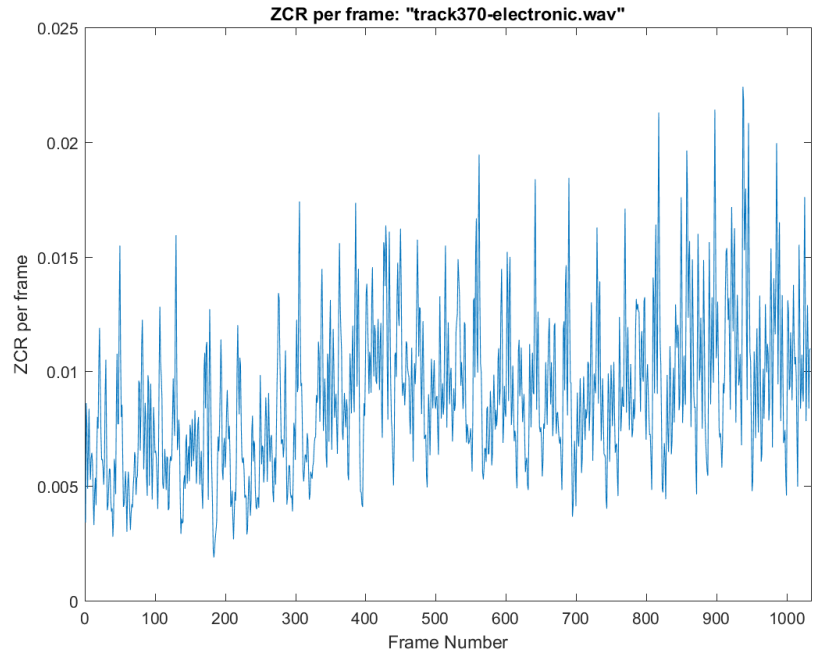
Figure 23: ZCR value per frame, electronic370
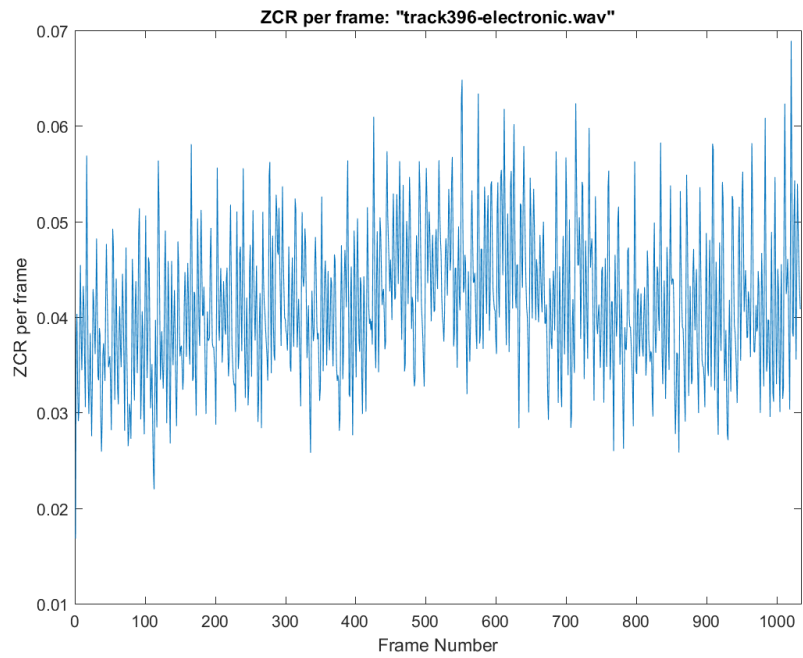


Figure 24: ZCR value per frame, electronic396
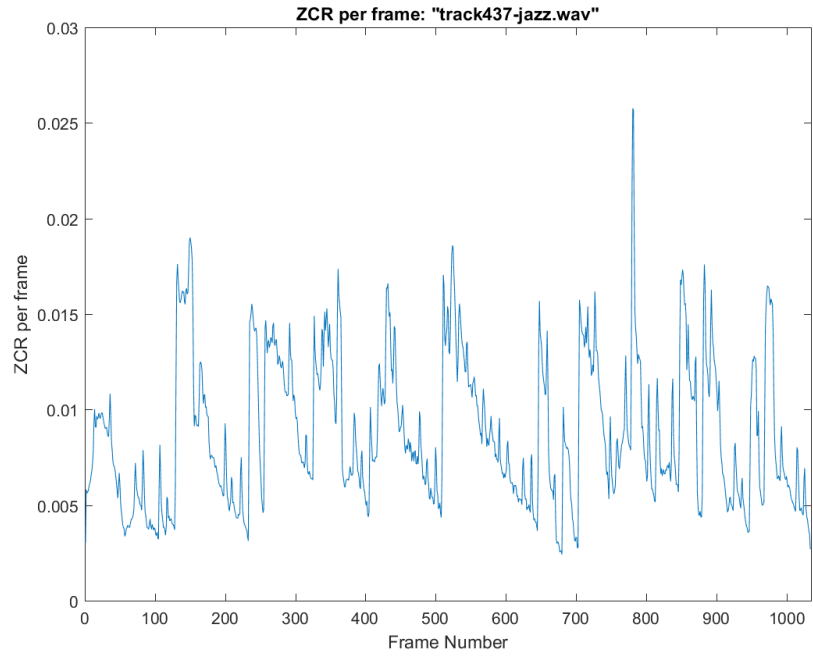
Figure 25: ZCR value per frame, jazz437
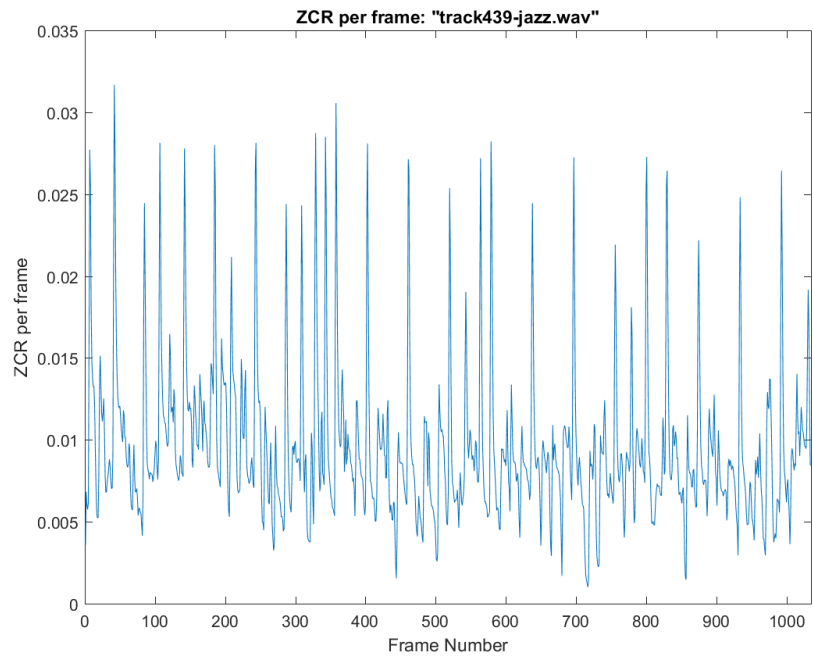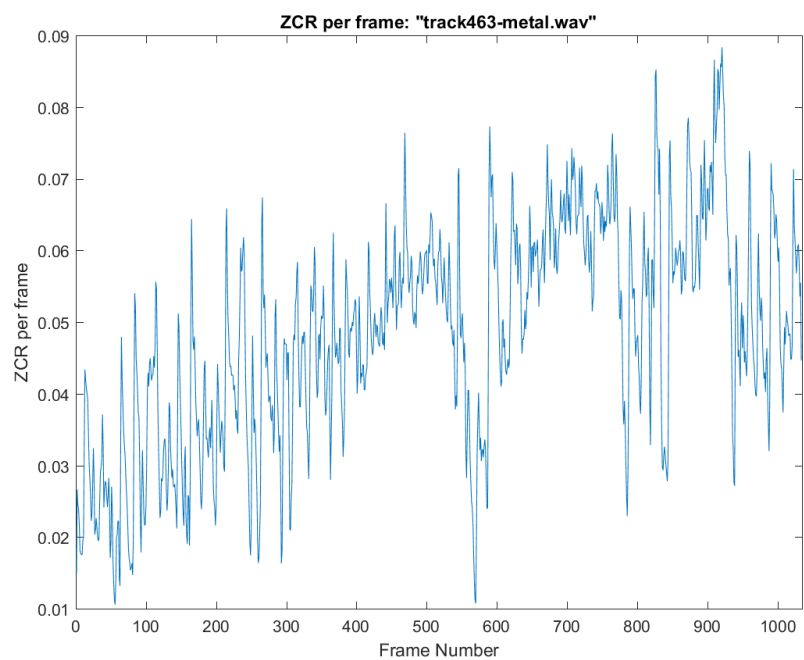


Figure 26: ZCR value per frame, jazz439
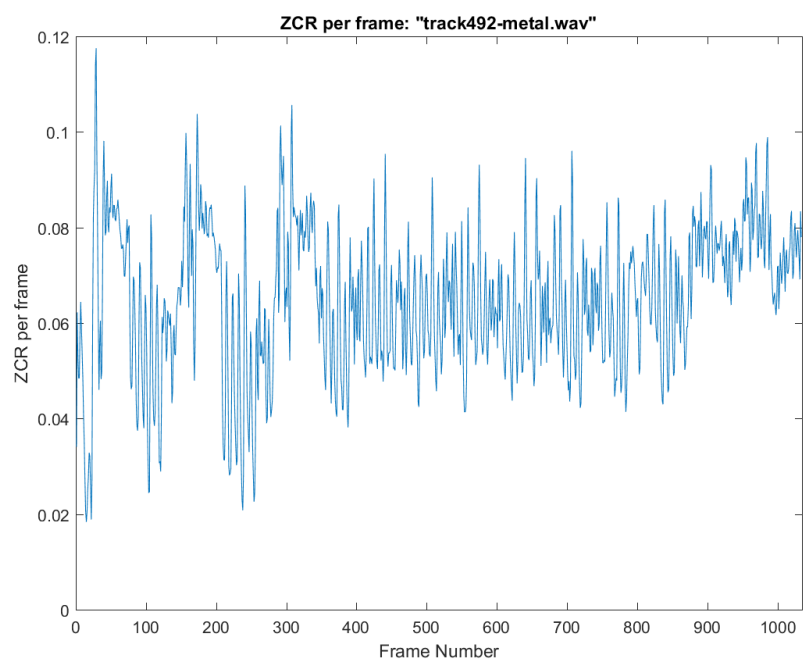
Figure 27: ZCR value per frame, metal463



Figure 28: ZCR value per frame, metal492
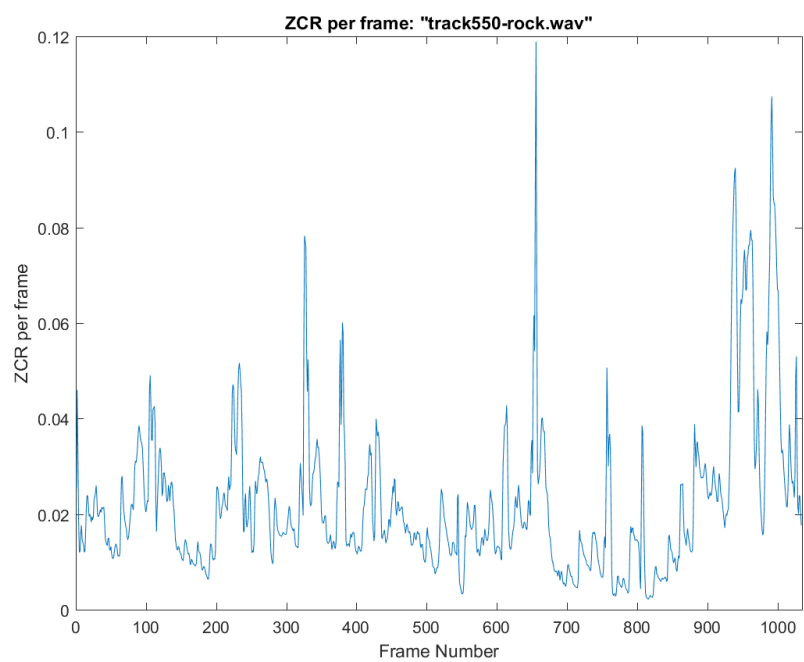
Figure 29: ZCR value per frame, rock547
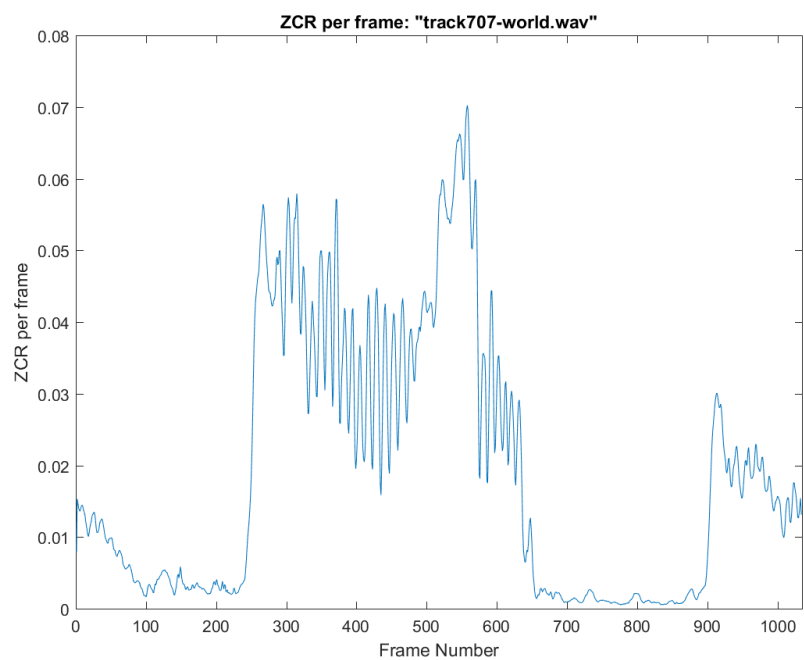


Figure 30: ZCR value per frame, rock550
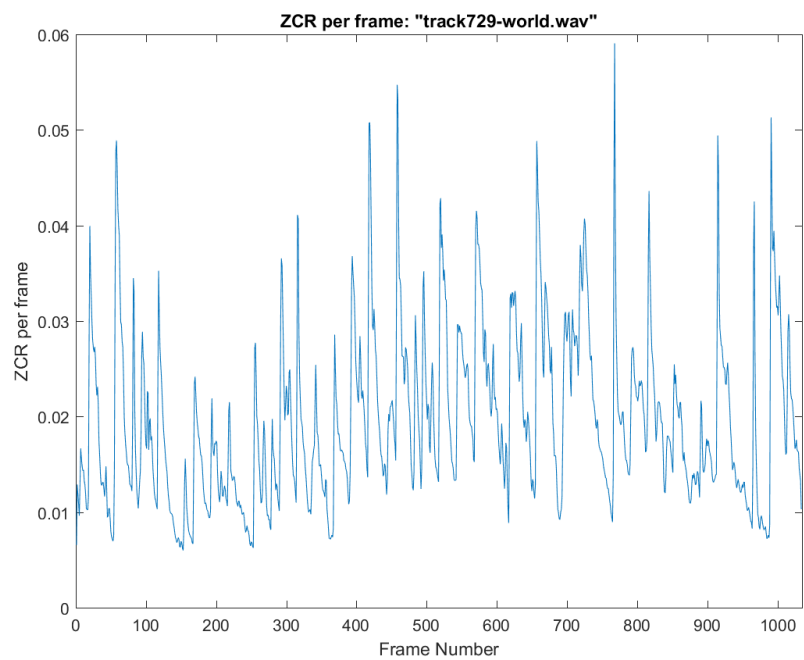
21

Figure 31: ZCR value per frame, world707



Figure 32: ZCR value per frame, world729
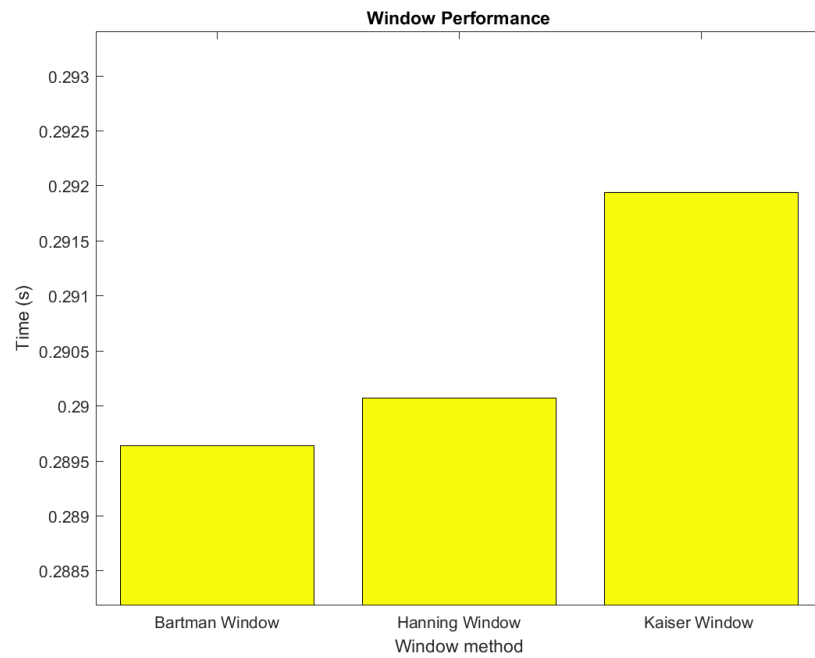
22

## A.3  Windows



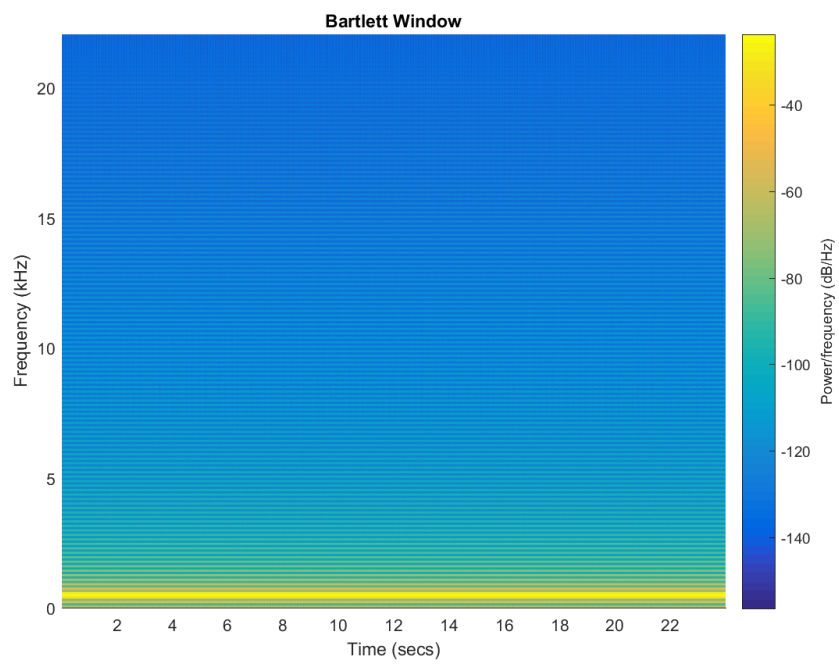Figure 33: Timing comparison of different windows



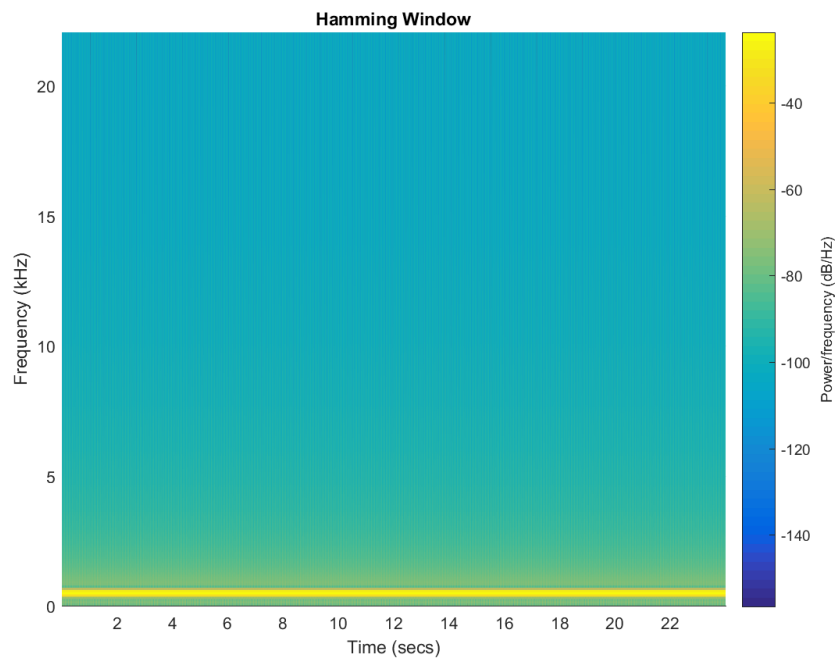Figure 34: Spectrogram of Bartlett Window

Figure 35: Spectrogram of Hamming Window
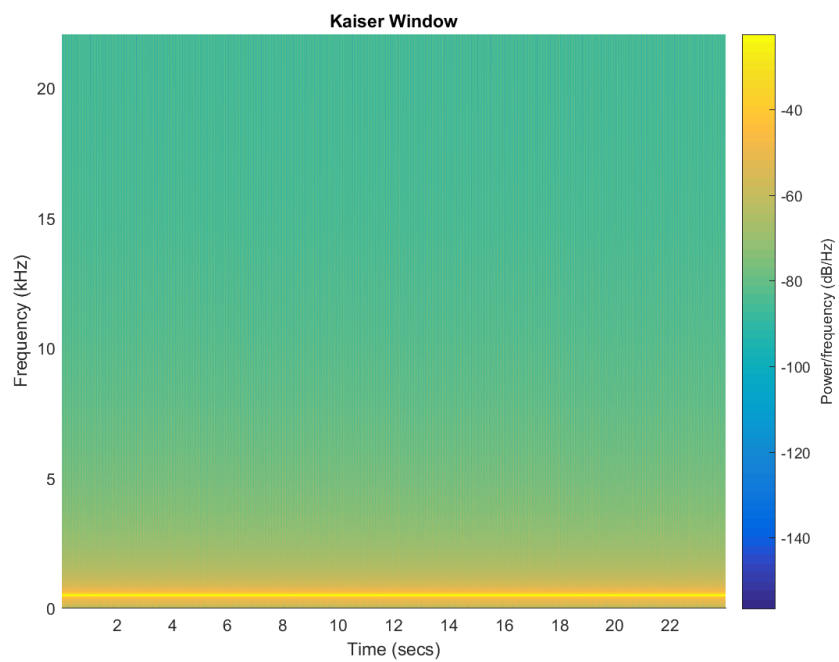


Figure 36: Spectrogram of Kaiser Window

## A.4 Centroid

;



Figure 37: Frequency Centroid



Figure 38: Frequency Centroid

Figure 39: Frequency Centroid



Figure 40: Frequency Centroid

Figure 41: Frequency Centroid



Figure 42: Frequency Centroid

Figure 43: Frequency Centroid



Figure 44: Frequency Centroid

Figure 45: Frequency Centroid



Figure 46: Frequency Centroid

## A.5   Spread



Figure 47: Frequency Spread



Figure 48: Frequency Spread

Figure 49: Frequency Spread



Figure 50: Frequency Spread

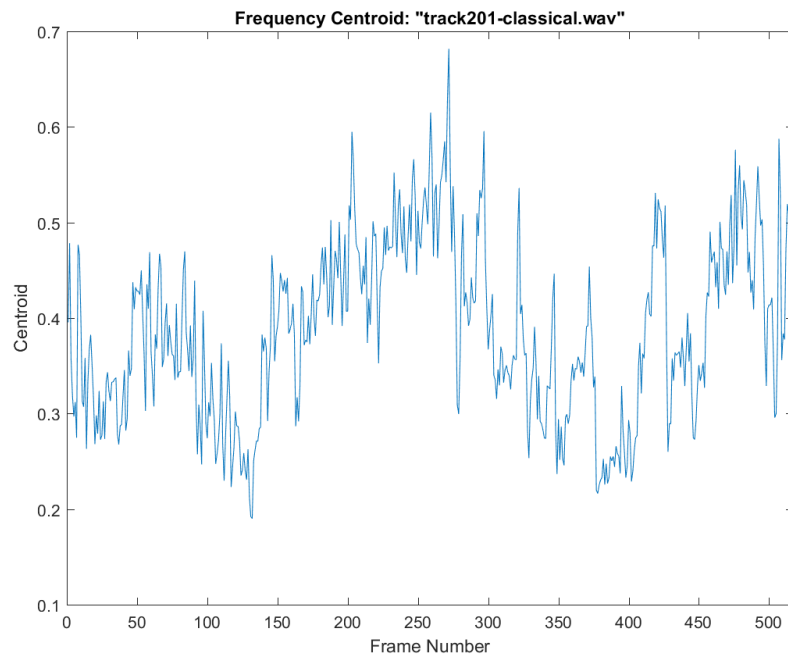Figure 51: Frequency Spread



Figure 52: Frequency Spread
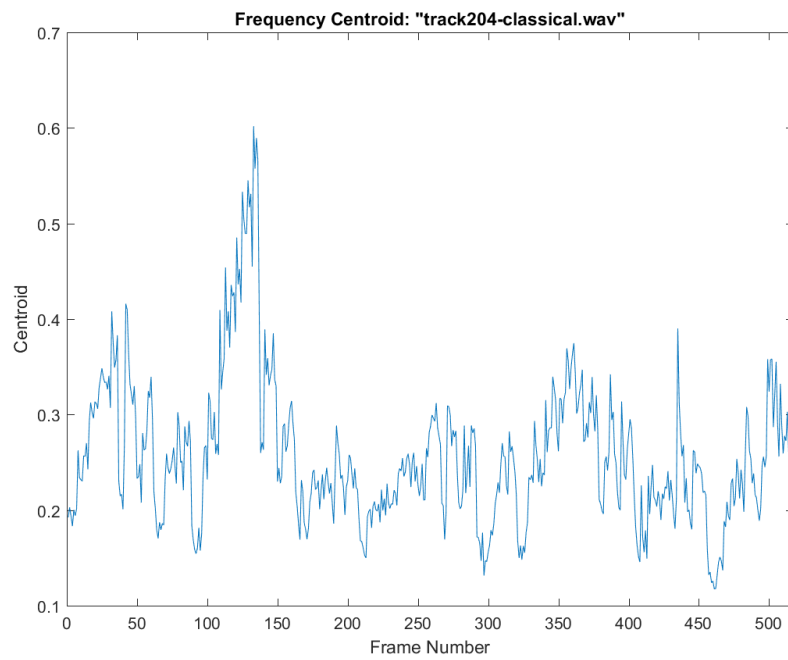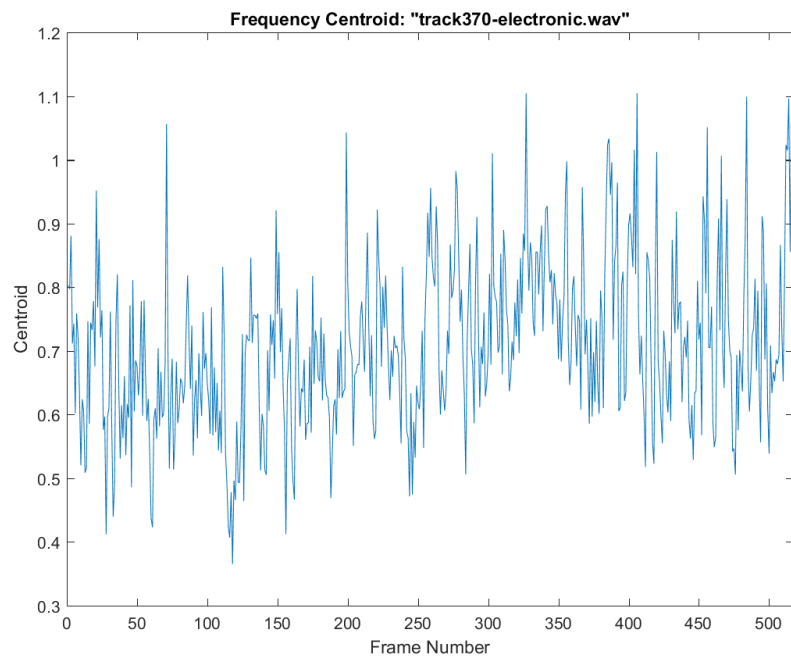
Figure 53: Frequency Centroid



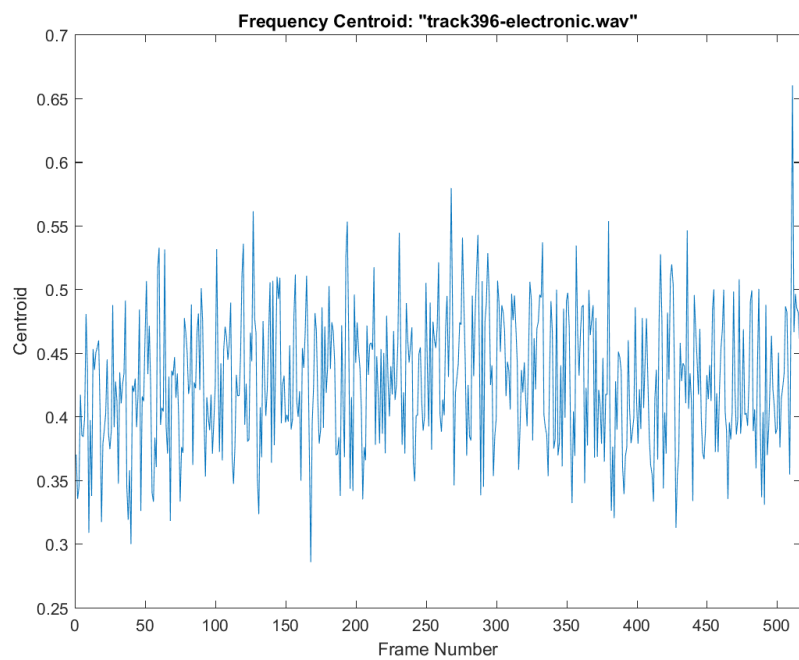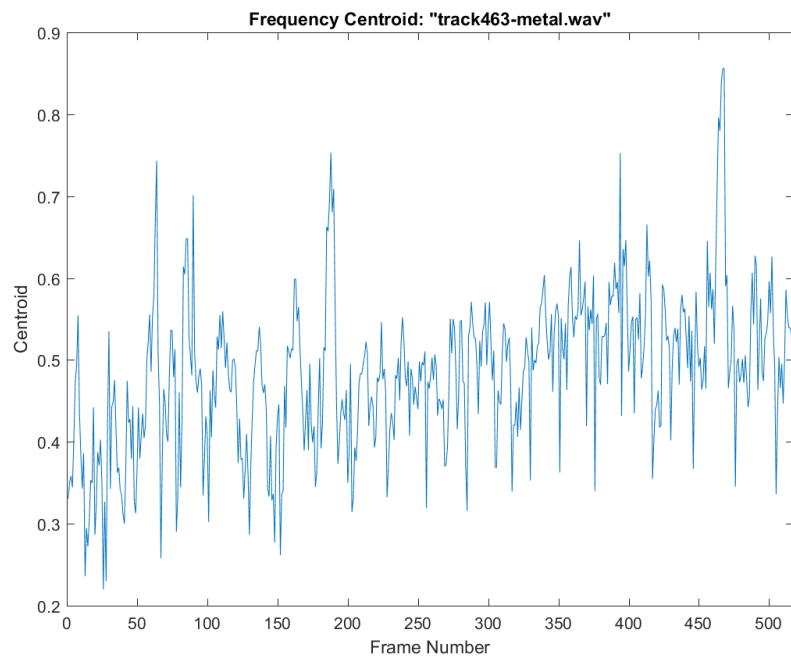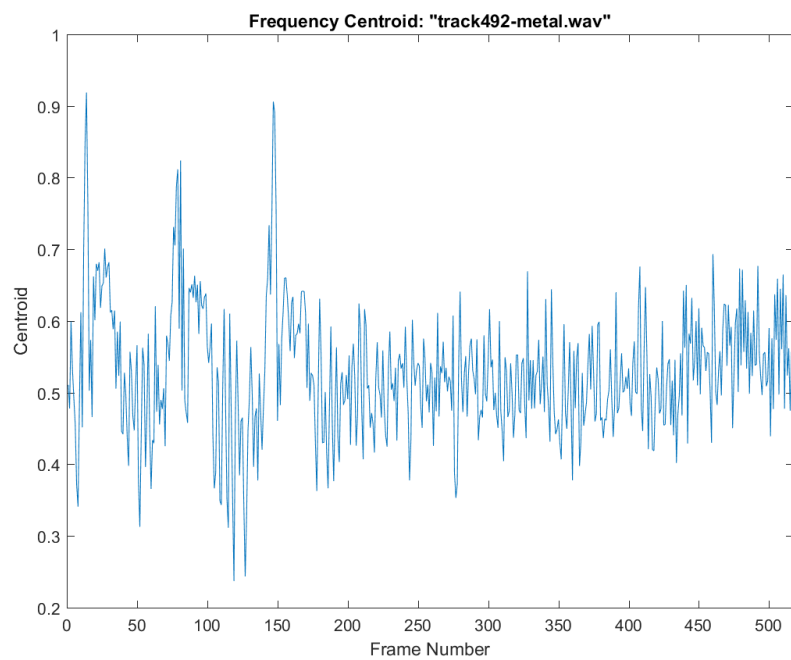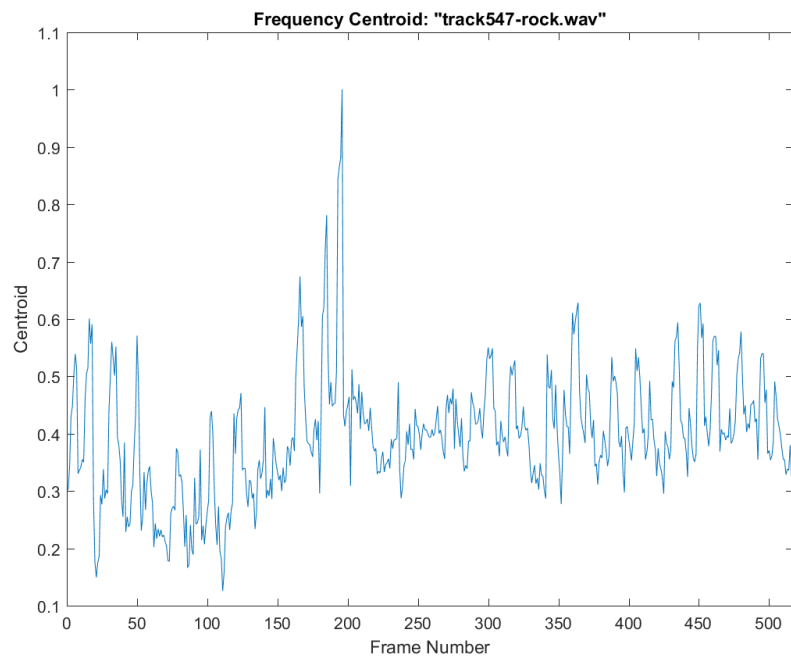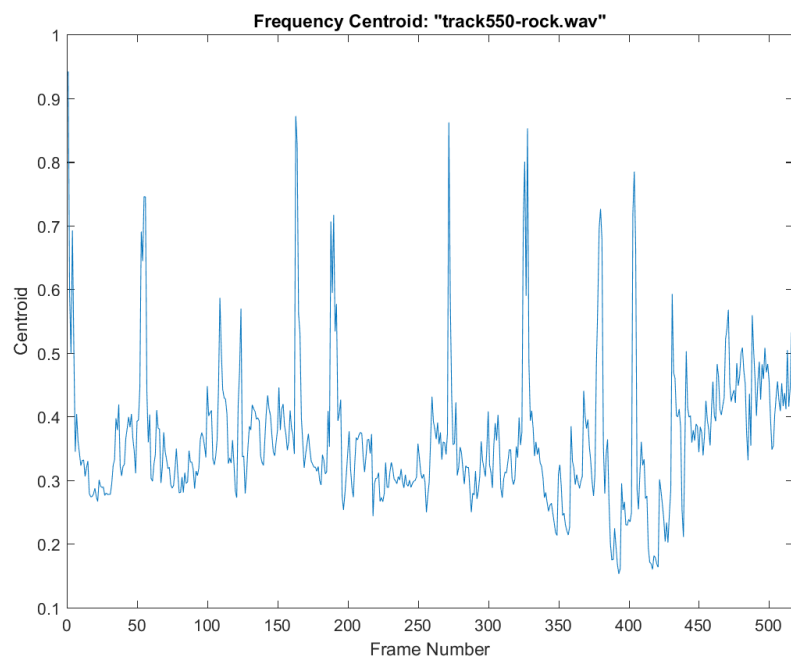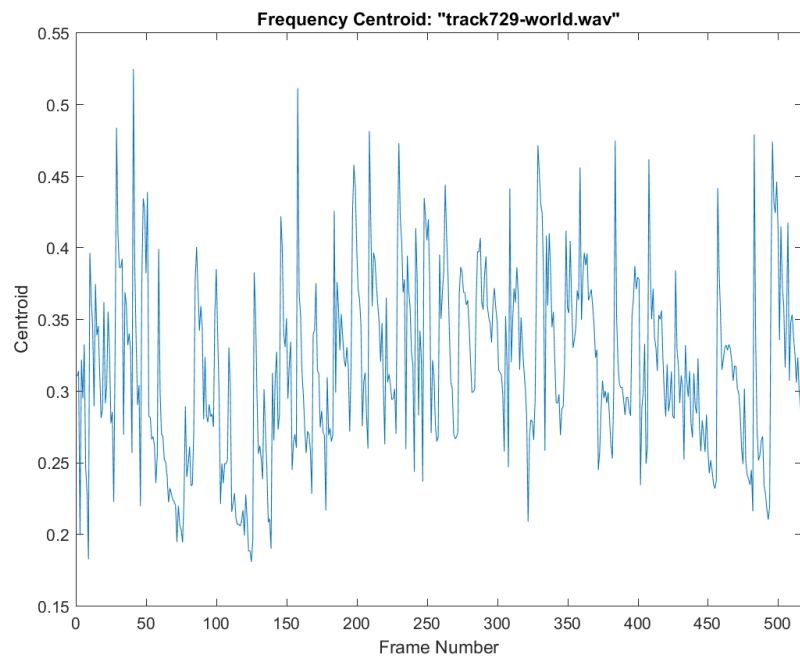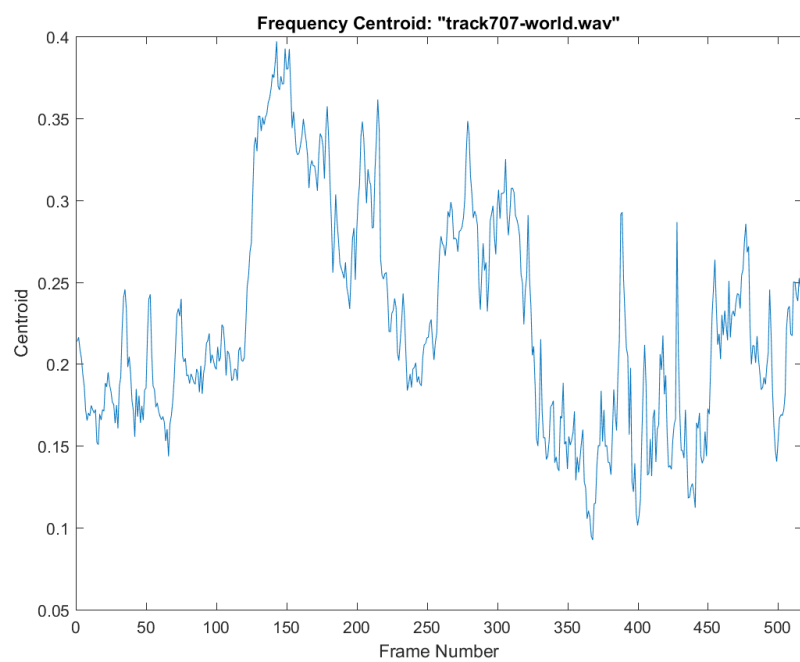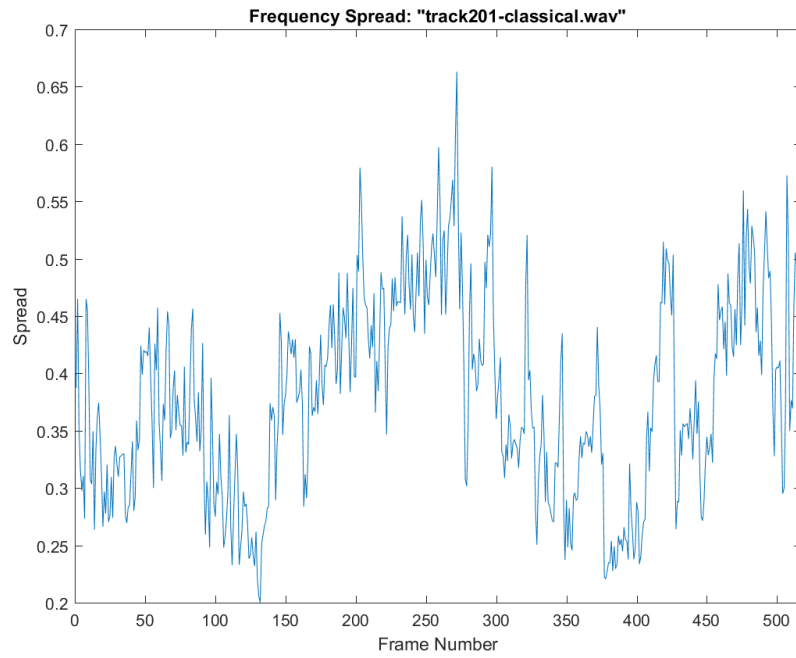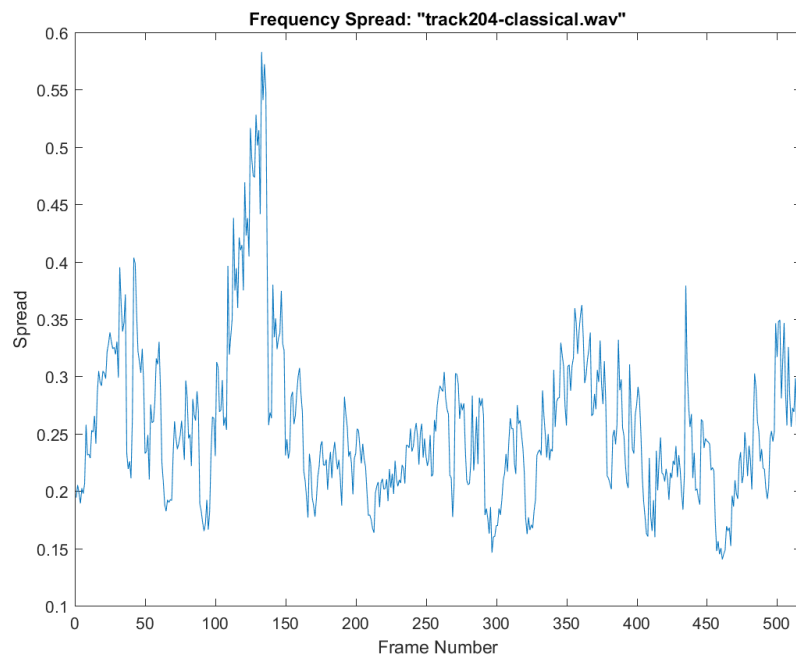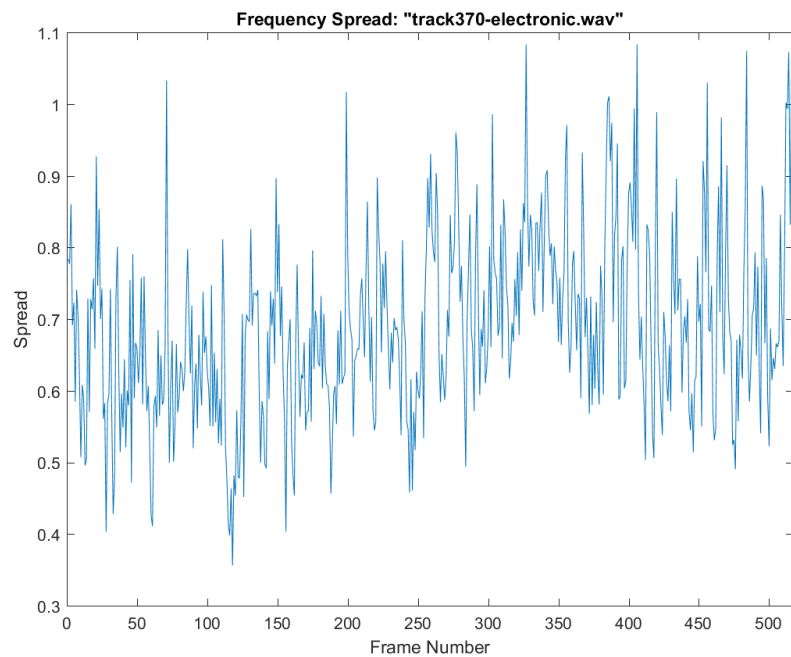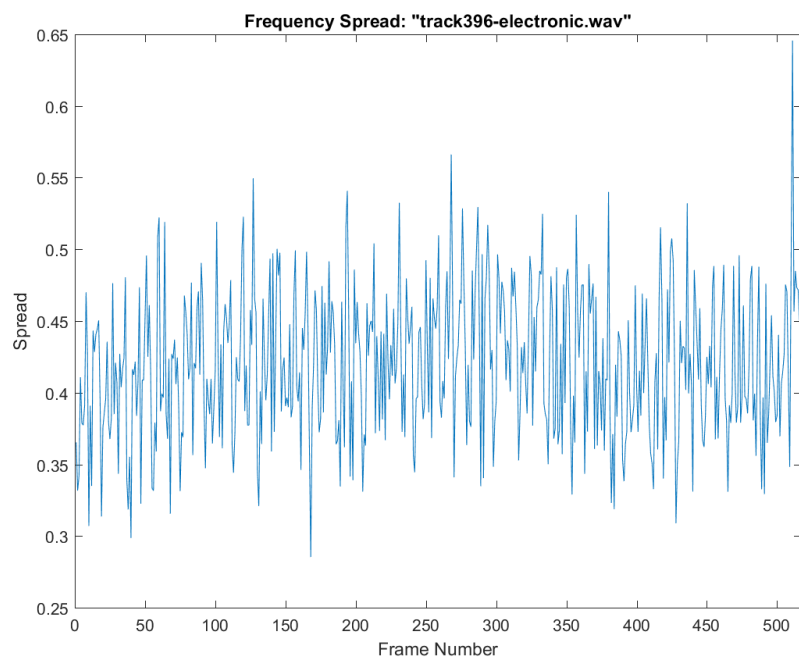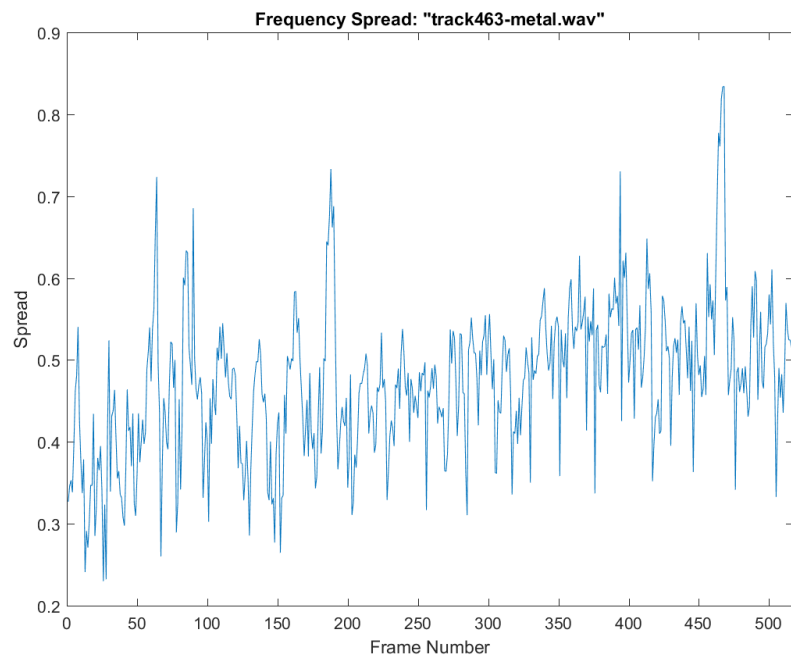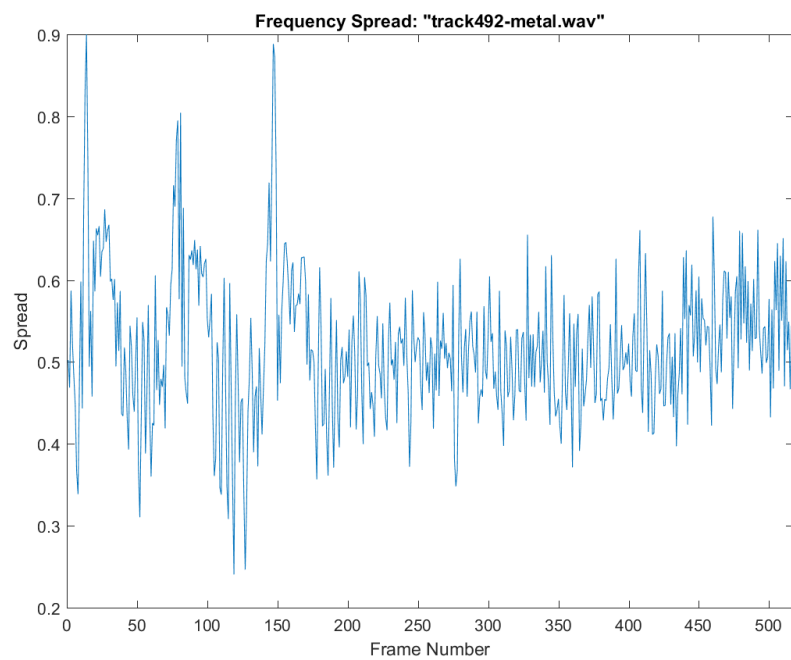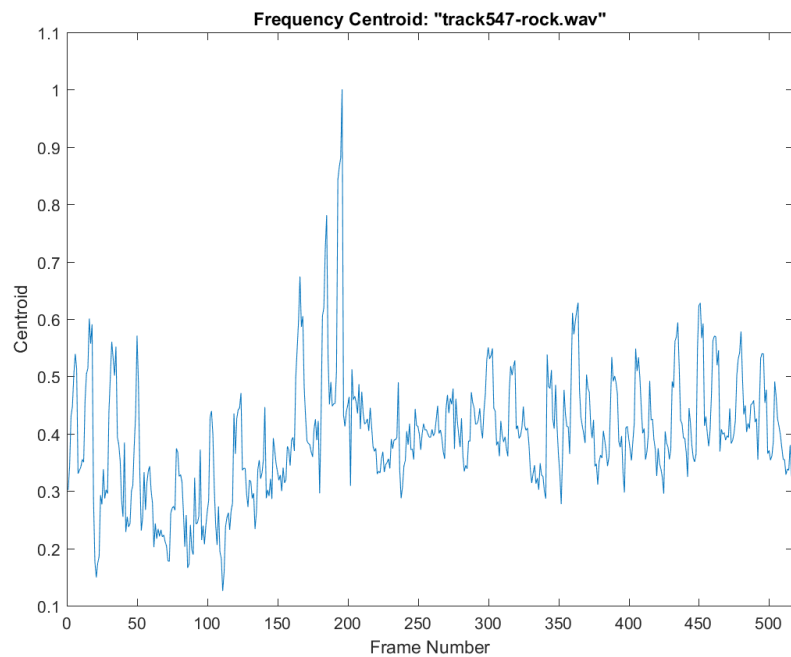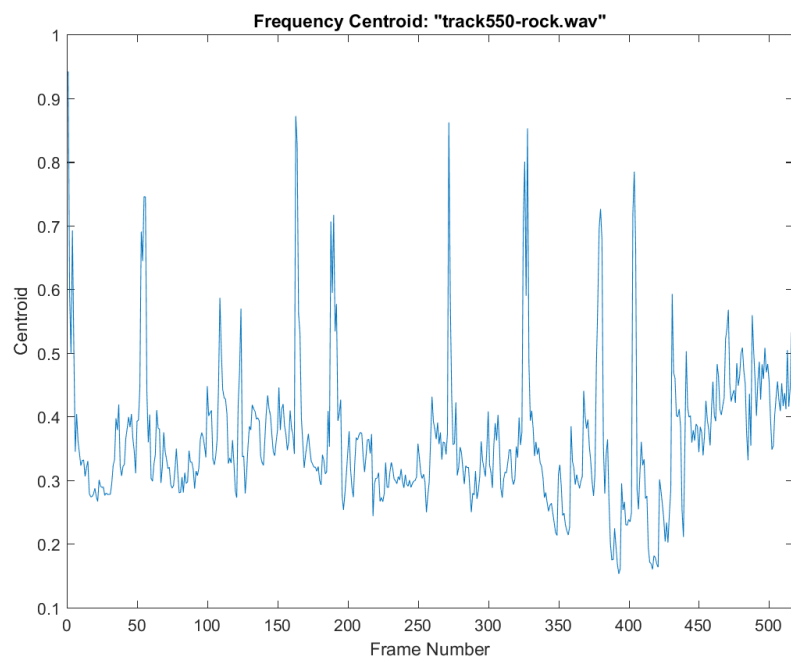Figure 54: Frequency Centroid
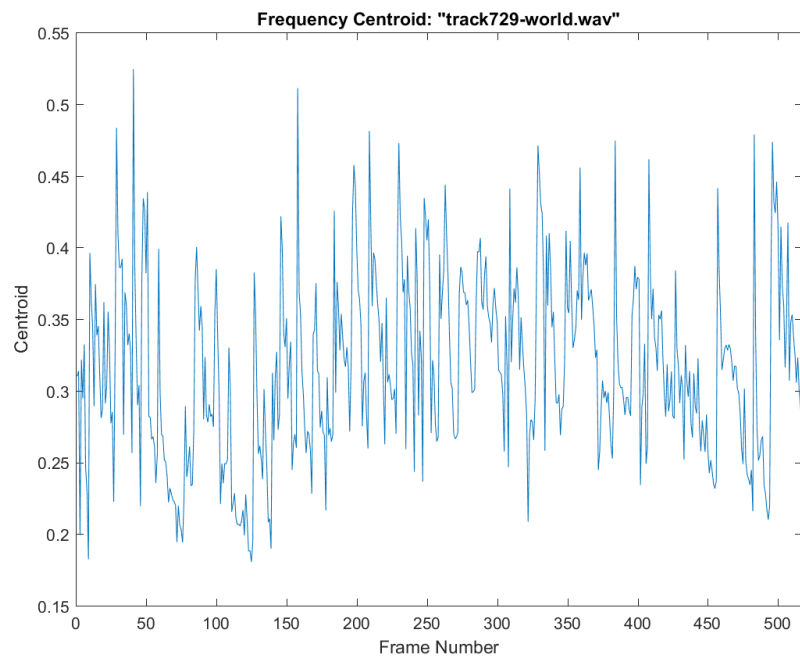
Figure 55: Frequency Centroid
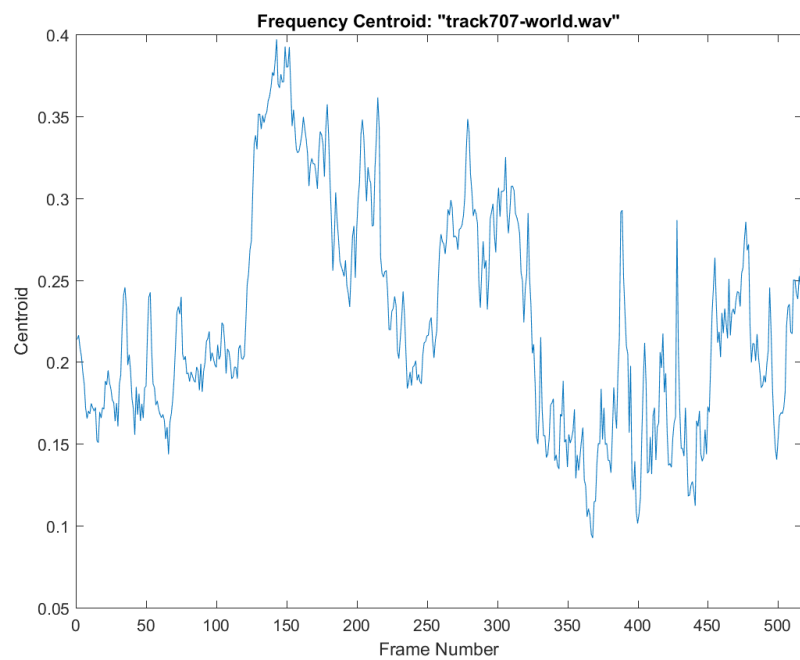


Figure 56: Frequency Centroid

# B  Code

<div align="center">Listing 5: extractSound.m</div>

```matlab
1  function [ soundExtract,p ] = extractSound( filename, time )
2  %extractSound Extracts time (in seconds) from the middle of the song
3  %    Write a MATLAB function that extract T seconds of music from a
4  %    given track. You will use the MATLAB function audioread to
5  %    read a track and the function play to listen to the track.
6  info = audioinfo(filename);
7  [song,~]=audioread(filename);
8  if time >= info.Duration
9      soundExtract=song;
10     p=audioplayer(soundExtract,info.SampleRate);
11     return;
12 elseif time<= 1/info.SampleRate
13     error('Too small of a time to sample');
14 end
15 samples=time*info.SampleRate;
16 soundExtract=song(floor(info.TotalSamples/2)-floor(samples/2):1: ...
17     floor(info.TotalSamples/2)+floor(samples/2));
18 p=audioplayer(soundExtract,info.SampleRate);
19 end
```

<div align="center">Listing 6: loudness.m</div>

```matlab
1  function [ loudness_data ] = loudness( filename,varargin )
2  %loudness Computes the standard deviation of the original audio divided
3  %into frames of size 255 loudness(filename,[franeSize,time,plotBool])
4  %    The standard deviation of the original audio signal x[n] computed over
5  %    a frame of size N provides a sense of the loudness
6  %
7  %    frameSize = size of frame to compute loudness over; defaults to 255
8  %    time = duration of song to analyze; defaults to 24 seconds
9  %    plotBool = whether or not to plot results; default is true
10 switch nargin
11     case 1
12         frameSize=512;
13         time=24;
14         plotBool=1;
15     case 2
16         frameSize=varargin{1};
17         time=24;
18         plotBool=1;
19     case 3
20         frameSize=varargin{1};
21         time=varargin{2};
22         plotBool=1;
23     case 4
24         frameSize=varargin{1};
25         time=varargin{2};
26         plotBool=varargin{3};
27     otherwise
28         error('loudness:argChk', 'Wrong number of input arguments');
29 end
30 [y,~]=extractSound( filename, time );
31 frames_data = buffer(y,frameSize,ceil(frameSize/2));
32 loudness_data=std(frames_data,0,1);
33 if plotBool==1
34     p=plot(1:length(loudness_data),loudness_data(1,:));
35     title(['Loudness per frame: "' filename '"']);
36     xlim([0 length(loudness_data)+1]);
37     xlabel('Frame Number');
38     ylabel('Loudness');
39     saveas(p,['Loudness_' filename(1:end-4) '.png']);
```

```
40   end
41   end
```

## Listing 7: loudnessPlot.m

```
1   % plot loudness features
2   close all;
3   figure(1)
4   loudness('track201-classical.wav');
5   figure(2)
6   loudness('track204-classical.wav');
7   figure(3)
8   loudness('track370-electronic.wav');
9   figure(4)
10  loudness('track396-electronic.wav');
11  figure(5)
12  loudness('track437-jazz.wav');
13  figure(6)
14  loudness('track439-jazz.wav');
15  figure(7)
16  loudness('track463-metal.wav');
17  figure(8)
18  loudness('track492-metal.wav');
19  figure(9)
20  loudness('track547-rock.wav');
21  figure(10)
22  loudness('track550-rock.wav');
23  figure(11)
24  loudness('track707-world.wav');
25  figure(12)
26  loudness('track729-world.wav');
27  close all;
```

## Listing 8: zeroCross.m

```
1   function [ ZCR_data ] = zeroCross( filename )
2   %zeroCross Computes the zero cross rate of the audio file
3   %    The Zero Crossing Rate is the average number of times the audio signal
4   %    crosses the zero amplitude line per time unit. The ZCR is related to...
5   %    the pitch height, and is also correlated to the noisiness of the...
6   %    signal.
7   frameSize=512;
8   time=24;
9   % info = audioinfo(filename);
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);
14
15  p=plot(1:length(ZCR_data),ZCR_data(1,:));
16  title(['ZCR per frame: "' filename '"']);
17  xlim([0 length(ZCR_data)+1]);
18  xlabel('Frame Number');
19  ylabel('ZCR per frame');
20  saveas(p,['ZCR_' filename(1:end-4) '.png']);
21  end
```

## Listing 9: zeroPlot.m

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

```
 6  zeroCross('track204-classical.wav');
 7  figure(3)
 8  zeroCross('track370-electronic.wav');
 9  figure(4)
10  zeroCross('track396-electronic.wav');
11  figure(5)
12  zeroCross('track437-jazz.wav');
13  figure(6)
14  zeroCross('track439-jazz.wav');
15  figure(7)
16  zeroCross('track463-metal.wav');
17  figure(8)
18  zeroCross('track492-metal.wav');
19  figure(9)
20  zeroCross('track547-rock.wav');
21  figure(10)
22  zeroCross('track550-rock.wav');
23  figure(11)
24  zeroCross('track707-world.wav');
25  figure(12)
26  zeroCross('track729-world.wav');
```

## Listing 10: windows.m

```
 1  function [Xn] = windows( filename, window )
 2  %windows Implement the computation of the windowed Fourier transform of y
 3  % start=cputime;
 4  frameSize=512;
 5  switch window
 6      case 1
 7          w=bartlett(frameSize); % Bartlett window
 8      case 2
 9          w=hann(frameSize); % Hann (Hanning) window
10      case 3
11          w=kaiser(frameSize,0.5); % Kaiser window
12      otherwise
13          w=kaiser(frameSize,0.5); % defaults to Kaiser window
14  end
15  [x,~]=extractSound(filename,24);
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
```

## Listing 11: windowsTiming.m

```
 1  % Windows timing
 2  fileArray=cellstr(['440Amp1.wav  '; '440Amp5.wav  ';'11025Amp1.wav';...
 3      '11025Amp5.wav'; '14080Amp1.wav'; '14080Amp5.wav']);
 4  timings=zeros(1,length(fileArray)*3);
 5  for i=1:length(fileArray)
 6      for j=1:3
 7          g=@()audioread(['' fileArray{i} '']);
 8          h=@()windows(['' fileArray{i} ''],j);
 9          timings((i-1)*3+j)=timeit(h)-timeit(g);
10      end
11  end
12
```

```
13  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;
```

## Listing 12: freqDist.m

```
1   function [ Xk ] = freqDist( filename )
2   %freqDist Computes the frequency distribution
3   Xn=windows(filename, 1);
4   Xk=zeros(size(Xn));
5   for n=1:length(Xn)
6       for k=1:min(size(Xk))
7           Xk(k,n)=abs(Xn(k,n))./sum(abs(Xn(1:k,n)));
8       end
9   end
10  end
```

## Listing 13: centroidSpread.m

```
1   function [ centroid,sigmaK ] = centroidSpread( filename )
2   %centroid Summary of this function goes here
3   %   The spectral centroid can be used to quantify sound sharpness or
4   %   brightness. The spread quantifies the spread of the spectrum around
5   %   the centroid, and thus helps differentiate between tone-like and
6   %   noise-like sounds.
7   close all;
8   Xk=freqDist(filename);
9   [a,b]=size(Xk);
10  centroid=zeros(1,b);
11  sigmaK=zeros(1,b);
12  for n=1:b
13      for k=1:a
14          centroid(n)=centroid(n)+(k/a*Xk(k,n));
15      end
16  end
17
18  for n=1:b
19      for k=1:a
20          sigmaK(n)=sigmaK(n)+(1/(a-1))*(Xk(k,n)-centroid(n))^2;
21      end
22  end
23  sigmaK=sqrt(sigmaK);
24  x=1:b;
25
26  figure
27  plot(x,sigmaK);
28  title(['Frequency Spread: "' filename '"']);
29  xlabel('Frame Number');
30  ylabel('Spread');
31  xlim([0,b]);
32  saveas(gca,['freqSpread' filename(1:end-4) '.png']);
33
34  figure
```

```
35  plot(x,centroid);
36  title(['Frequency Centroid: "' filename '"']);
37  xlabel('Frame Number');
38  ylabel('Centroid');
39  xlim([0,b]);
40  saveas(gca,['freqCent' filename(1:end-4) '.png']);
41  close all;
42  end
```

Listing 14: centroidPlot.m

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

Listing 15: melBank.m

```
1   function [ fbank ] = melBank(  )
2   %melBank Creates a set of mel filter banks
3   %    Implement the computation of the triangular filterbanks
4   %    Hp, p = 1,...,NB. Your function will return an array fbank of size
5   %    NB x K such that fbank(p,:) contains the filter bank Hp.
6
7   nbanks = 40; %% Number of Mel frequency bands
8   % linear frequencies
9   linFrq = 20:fs/2;
10  % mel frequencies
11  melFrq = log ( 1 + linFrq/700) *1127.01048;
12  % equispaced mel indices
13  melIdx = linspace(1,max(melFrq),nbanks+2);
14  % From mel index to linear frequency
15  melIdx2Frq = zeros (1,nbanks+2);
16  % melIdx2Frq (p) = \Omega_p
17  for i=1:nbanks+2
18      [val indx] = min(abs(melFrq - melIdx(i)));
19      melIdx2Frq(i) = linFrq(indx);
20  end
21
22  end
```