A Regression Based Approach for Onset Detection

Swapnaneel Bhattacharyya(BS2105)
Srijan Chattopadhyay(BS2126)

Summer, 2023

Course: Statistical Methods IV
B.Stat 2nd Year

Supervisor: Dr. Arnab Chakraborty, ASU, ISIK

Indian Statistical Institute, Kolkata

Contents

1 Introduction

The onset refers to the beginning of a musical note. So onset detection is used to identify the time points where a new sound starts in an audio signal.
When a new musical note begins, a certain amount of increase in energy occurs. So by keeping track of energy flow and analyzing the sound data, we can come up with the onsets.
In modern technology, onset detection is used in several audio applications including music information retrieval, speech recognition, and sound classification.
In this project, we develop a regression-based method of extracting onsets from audio data.

2 Methodology

Our method is based on three steps:

1. **Noise Removal**
2. **Regression and Smoothing**
3. **Peak Detection**

The steps are explained below:

2.1 Noise Removal:

Usually, the audio files contain a lot of noise mixed with the original sound. At first, we want to get pure sound, i.e. the sound without any noise. For that,

- We apply the **Discrete Fourier transform(DFT)** (using **Fast Fourier Transform(FFT)**) on the data, which is on the time domain to convert to the frequency domain. So, it gives the characterization of the different frequencies with their corresponding contribution in the time series.

- Then to remove noise, we remove those **frequencies** which have a very **low contribution** to the data (i.e. remove the frequencies which are less than a certain threshold).
- Then again applying the **inverse DFT** on the frequency domain data, we get the **noiseless** time domain data.

This completes the process of noise removal. The following code in R does that:

```
#applying FFT
z = fft(s@left)/sqrt(length(s))
#creating a new vector in f.d
zclean = z
zclean[abs(z)<100]=0 #removing low frequency
plot(abs(zclean),ty="l")
sclean = s #creating a new vector in t.d
#applying inverse FFT
snd  = fft(zclean, inv=T)*sqrt(length(s))
tmp = Re(snd)#taking real part
sclean@left = 30000*tmp/max(abs(tmp))
sclean@right = rep(0,length(s))
play(sclean)
```

Figure 1: R Codes For removing noise using FFT

However if the original sound does not contain much noise, it's recommended to skip the noise removal step.

2.2 Regression and Smoothing:

2.2.1 Regression:

We apply **Ordinary Least Squares Linear Regression** as follows: We fix a $k > 0$.

- Take the following sets $\{1, 2, .., k\}, \{2, .., k+1\}$, etc. up to wherever feasible according to the length of the dataset.

- Fit a regression model with $\{X_i, X_{i+1}, .., X_{i+k-1}\}$ as response variable with $\{1, 2, .., k\}$ as predictors.
- Let the length of the sample be t, so we got $t-k+1$ many linear regressions and each having a sum of squares of the absolute value of residuals. Now in place of the square, if we try some larger power, then we can discriminate the peaks more easily because the larger power of a larger quantity will blow up. But also, if we try much larger power that can make storage problems or computational problems. Considering these, we took the third power of the absolute value of residuals. Let **res** be the vector defined as

  $$\textbf{res}[i] = \text{sum of cubes of the absolute value of residuals for the model } \{X_i, X_{i+1}, ..., X_{i+k-1}\} \sim \{1, 2, .., k\}$$

  So, we got a vector of length $t - k + 1$.
- We plot the vector **res**.
- Now, in the original sound, whenever the energy flow increases, a sharp peak in the plot of the data in the time domain occurs. For this sudden increase of the value of X_t, the linear regression fit around the actual peak is not good which results in a considerable increment of the sum of residuals, for which we observe a peak in the plot of **res** near the original peaks. So if we try to find the peaks in the plot of **res**, we can detect the original time points for the peak of the original data.
- In the actual dataset, before a peak and after a peak, points are gradually increasing or gradually decreasing in a linear fashion. Hence, we can expect the residuals to be relatively less in the neighborhood of the peaks.
- We took k to be 1000 for our all applications, and it worked very well, however any number in the range $\{500, 3000\}$ can be a good choice.

The step **Regression** is done by the following R-code in Figure 2.

```
res= numeric(length(s)-1000)

for(i in 1:(length(res))){
  z=sl[seq(i,i+999,1)] #taking k points at a time
  q = seq(1,1000,1) #the indices
  p = lm(z~q) #plotting the model
  res[i]= sum((abs(p$residuals)^3))#residuals
  print(i)#to see the progress
  }
```

Figure 2: R Code for k consecutive regression

So we are left with detecting the peaks of the plot of res. To do that we will perform Smoothing to our residual vector res as follows:

2.2.2 Smoothing:

The plot of the residual is much less noisy than the actual time series plot, but still not enough clear to detect the peaks easily. So, for that, we apply a smoothing technique to detect the peaks easily. Here we use the fact that **mean is a non-robust statistic**. Then

- First fix a number $l > 0$, then divide the whole index set into consecutive l length blocks, up to the nearest multiple of l which is less than the length of the index set, i.e. $[1, l], [l + 1, 2l], [2l + 1, 3l]..$, etc.
- Then for each such interval we replace all the values of **res** by the mean of the values of **res** from that interval and plot it. We call the plot **Step Plot**. Now by appropriately choosing l, if we can take at most one peak in each block, then the task will be very easy. As, the mean is non-robust, when an interval will contain a peak, the mean of that interval will automatically increase and the neighbor blocks will have a relatively small mean. So

peaks will occur in those points only, where there were peaks in the actual data.

- We choose l to be 5000 in all of our applications. However any number between $\{1000, 5000\}$ may be a good choice.

This completes the Smoothing step. Now, the final step is the peak detection from the step plot. The following code does the smoothing:

```
a = floor(length(res)/5000)*5000
#Finding the precise intervals
mres = numeric(length(res))
c = seq(1,a-4999,5000)
for(i in c){
  y = res[seq(i,i+4999,1)]
  mres[seq(i,i+4999,1)]=mean(y)
  }
```

Figure 3: R Code for smoothing

2.3 Peak Detection:

Now we have a step plot, we first fix a number α based upon data.

- For a particular block (say s^{th} block), if the difference between the **means of sample values from the block**($\bar{z}_s$) and the **previous**($\bar{z}_{s-1}$) and **next block**($\bar{z}_{s+1}$) both are greater than α, i.e. $\bar{z}_s - \bar{z}_{s-1} > \alpha$ and $\bar{z}_s - \bar{z}_{s+1} > \alpha$ then that block contains a peak. Since the blocks are designed to have at most one peak(roughly) so, we consider the peak to be at the median of the sample values from the block s.

- Applying the above criterion for all the blocks, we get the location of the peaks.

- From the **.wav file**, we know the sampling rate of the sound. So, to get the exact time stamps, we can just divide the peaks by the sampling rate and we will get our desired output.
- However, here the choice of α may be quite subjective and may depend on the plot after smoothing. More experience will lead to a better selection of α, which in turn will do a perfect note arrival detection. However, experimentally we noticed that if the maximum value after smoothing is of order 10^m, then a value between $\{10^{[\frac{m}{2}]+1}, 10^{\frac{m+1}{2}}, .., 10^{m-1}\}$ works well, however, it's better to apply a supervised learning method here to choose the α appropriately.

The following code does this task:

```
k=0 #counter
peak = numeric() #null vector

for(i in c[-c(1,length(c))]){
  y = mres[seq(i-5000,i-1,1)] #previous block
  z = mres[seq(i,i+4999,1)] #current block
  w = mres[seq(i+5000,i+9999,1)] #next block
  if(z[1]-y[1]>10^7 && z[1]-w[1]>10^7){
    k=k+1 #increase the counter
    peak[k]=median(seq(i,i+4999,1)) #detecting the peak
  }
}
onset = peak/s@samp.rate
#final time stamps
```

Figure 4: R Code for detecting peaks

3 Applications

3.1 Application 1:

The data can be downloaded from here.

3.1.1 Time Domain Plot and Noise Removal:

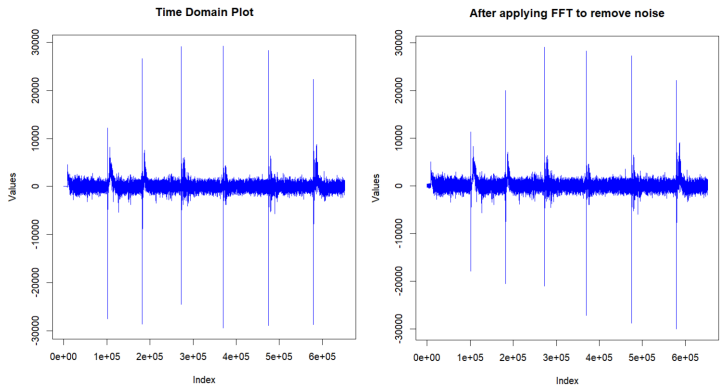

Figure 5: Time Domain Plot of the Data before and after noise removal

3.1.2 Regression and Smoothing:

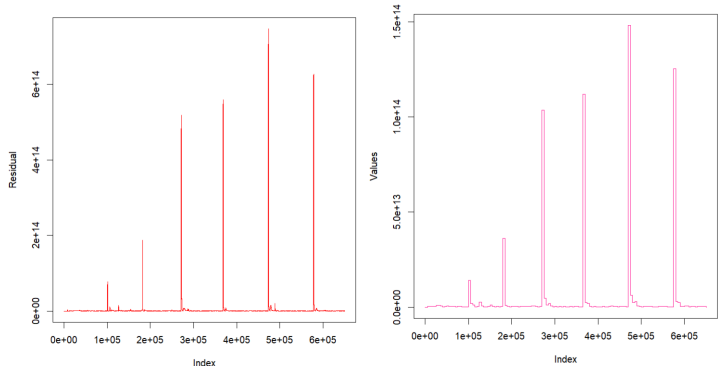

Figure 6: Plot of the Res vector and Plot after Smoothing

3.1.3 Peak Detection

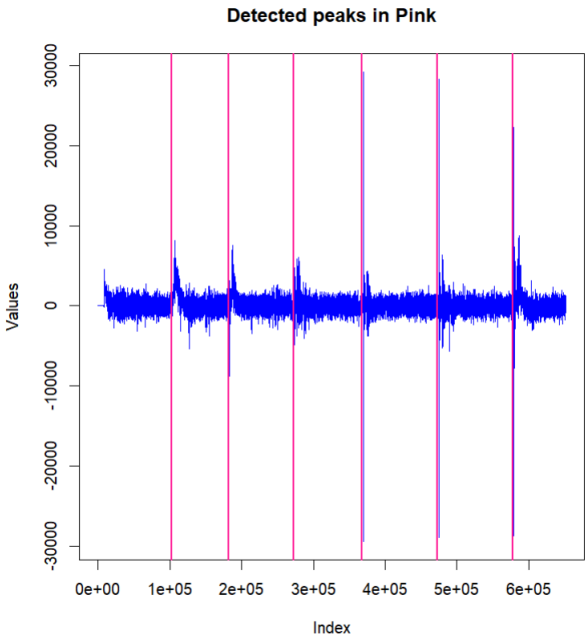

Figure 7: Detected Peaks

```
[1]  2.135427  3.802094  5.677094  7.656260  9.843760 12.031260
```

Figure 8: Output

Here, we get the timestamps of changing notes to be 2.135427, 3.802094, 5.677094, 7.656260, 9.843760, and 12.031260 respectively. And, clearly from the time domain plot, there were 6 peaks and we got each of those. So the algorithm is working pretty nicely in case of perfect ideal data.

3.2 Application 2:

The data can be downloaded from here.

3.2.1 Time Domain Plot and Noise Removal:

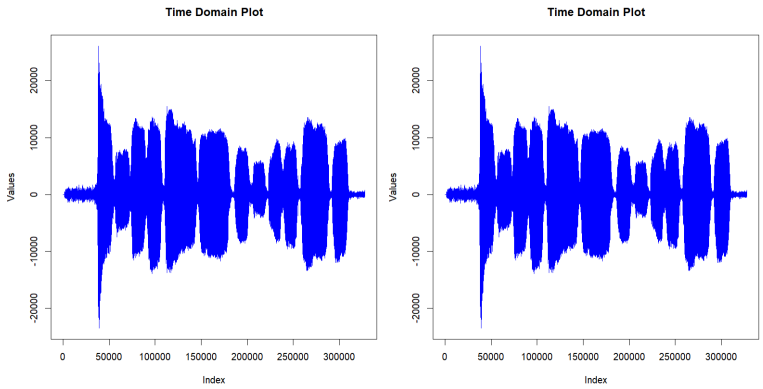

Figure 9: Time Domain Plot of the Data before and after noise removal

3.2.2 Regression and Smoothing:

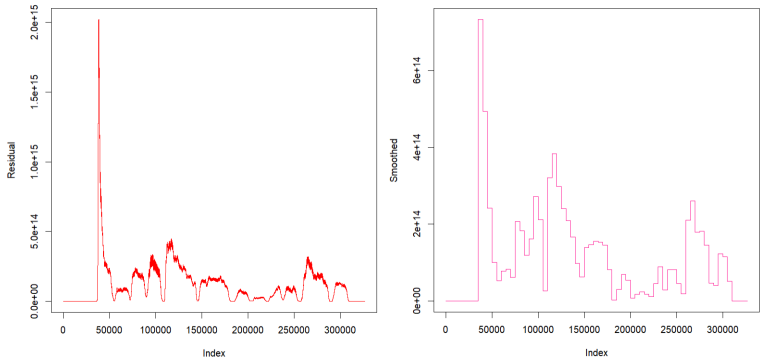

Figure 10: Plot of the Res vector and Plot after Smoothing

3.2.3 Peak Detection:

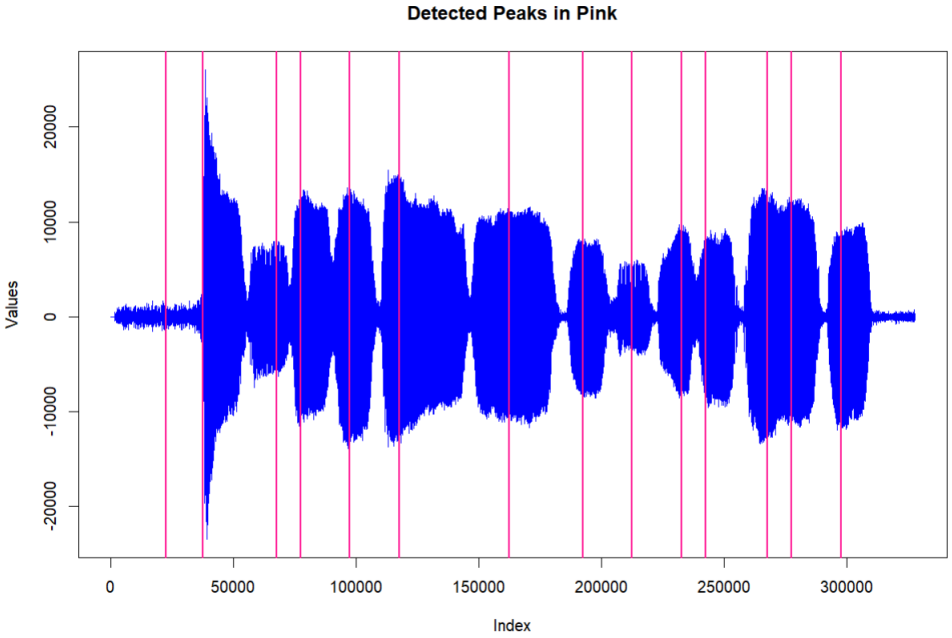

Figure 11: Detected Peaks

```
 [1] 0.5102154 0.8503515 1.5306236 1.7573810 2.2108957 2.6644104
 [7] 3.6848186 4.3650907 4.8186054 5.2721202 5.4988776 6.0657710
[13] 6.2925283 6.7460431
```

Figure 12: Output

These are the timestamps of note change.

4 Advantages

Our method has the following advantages:

- The method is very easy to interpret.
- Without using much advanced statistical tools or packages, the method is fairly able to detect the onsets.

5 Disadvantages

Despite the above advantages the method has the following disadvantages:

- The time complexity of this method is pretty high.
- If the thresholds are not properly chosen, this method may suffer from the possibility of over-detection or under-detection of onsets.