Audio Comparison - Two Pieces

Anna Yanchenko

5/19/2019

Abstract

This vignette demonstrates the audio processing and distance decomposition code used to process the Beethoven symphonies adapted to compare two pieces only. The various data representations and decompositions are described and calculated.

1. Data Representations

1.1 Audio Recordings

The sample recordings considered here consist of 2 different recordings of *Twinkle, Twinkle, Little Star*. The *reference recording* is a constant volume and tempo of 95 bpm. The *query recording* changes tempo every 4 bars, from 120 bpm, to 95 bpm to 60 bpm and also decreases in volume every 4 measures. The reference recording is assumed to be tuned to A4 = 440 Hz and the query to A4 = 442 Hz. Both recordings are originally piano "MIDI" recordings.

1.2 Code Notes

The main script for the analysis is `data_processing2.R`, which contains code to processes the audio and calculate the various data decompositions. The data processing uses the packages `tuneR` and `seewave`, the audio alignment is performed using the package `dtw` and the distance calculations use `philentropy`.

```
## Load packages and functions
library(tuneR)
library(reshape2)
require(gridExtra)
library(ggplot2)
library(dtw)
library(abind)
library(seewave)
library(readr)
library(magrittr)
library(philentropy)
library(car)

source("data_processing2.R")
```

1.3 Data Representations

The two main data representations used are the spectrogram and the chromagram. A MIDIgram representation is also calculated as an intermediate representation in the calculation of the chromagram. All data representations and decompositions considered rely on converting the raw audio files to a spectrogram representation using a Fast Fourier Transform (FFT). The spectrogram represents time and frequency

amplitudes across the entire spectrum. The FFT trades off frequency resolution for time resolution. The frequency resolution for the FFT is set to 5 Hz with a temporal resolution of 0.1 s, which corresponds to a window length of 4410*2, though other resolutions are possible.

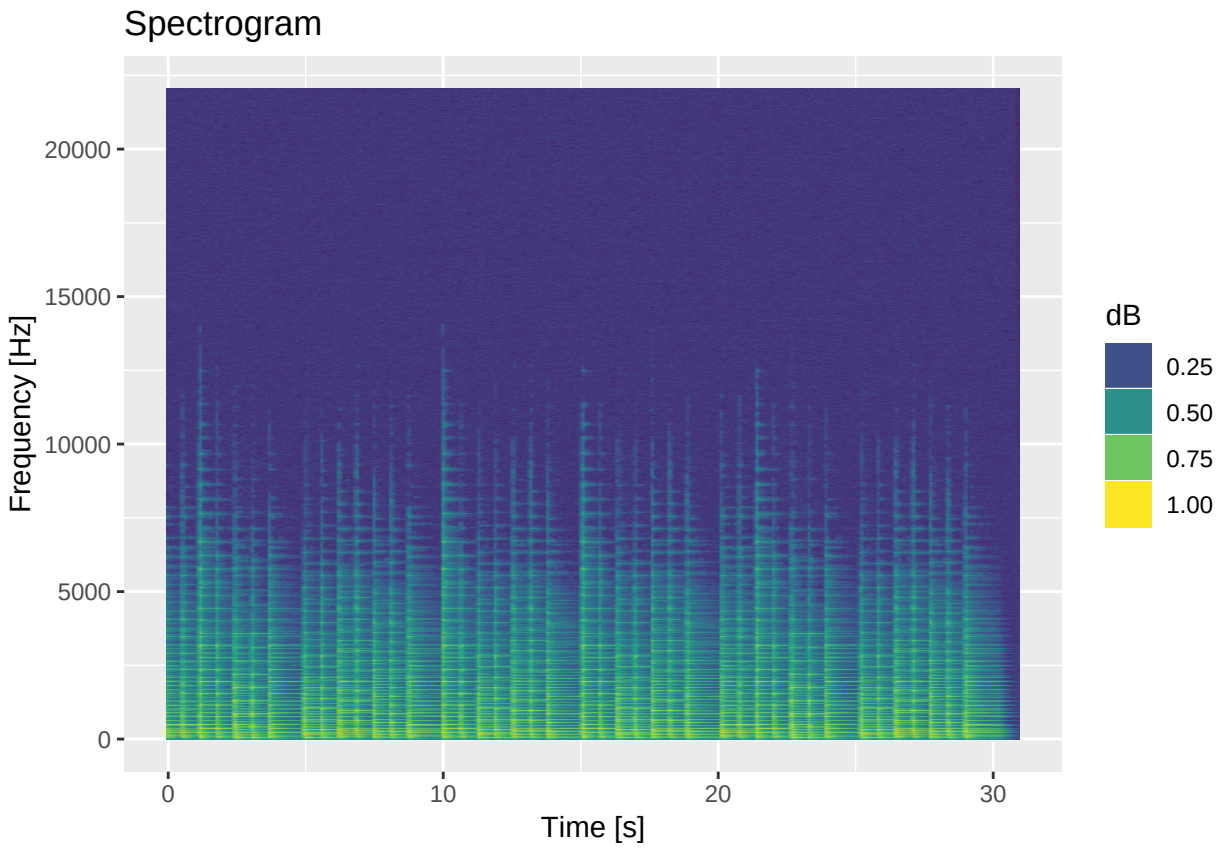
The MIDIgram representation converts the spectrogram frequencies to MIDI pitch numbers to aid in the calculation of the chromagram. In this step, the different tunings of the orchestras can be taken into account by converting the appropriate frequencies to each note pitch (standard tuning is A4 = 440 Hz).

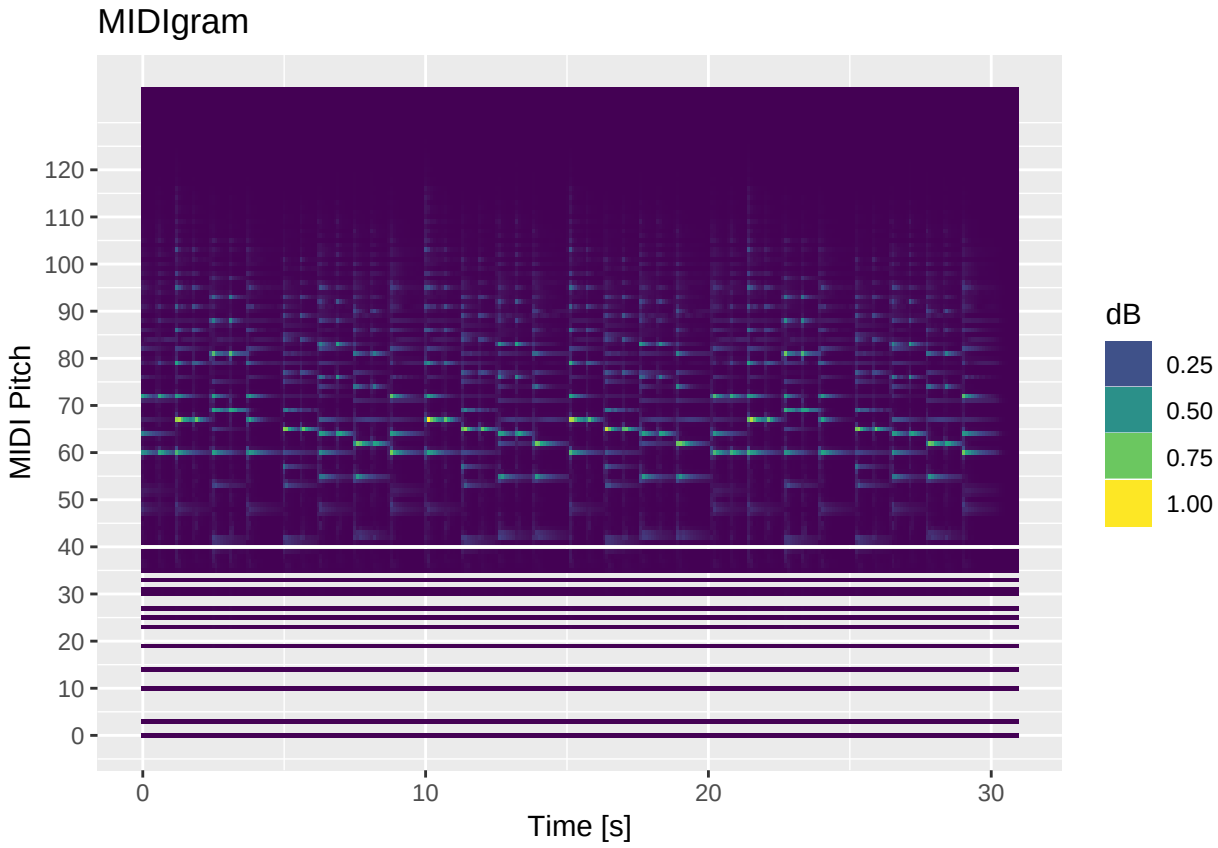
The chromagram representation plots the 12-dimensional chroma values vs. time, where the intensity is the volume (sum of the amplitudes from the spectrogram representation). These chroma vectors represent the note pitch with the octave removed and the pitches are thus collapsed across octaves. There are 12 chroma dimensions, one corresponding to each of the 12 semitones in an octave. The spectrogram, MIDIgram and chromagram for the original recording of *Twinkle, Twinkle, Little Star* are plotted below. The chroma are very distinct in time for this recording, as each note has a clear onset in this simple, piano recording. Note onsets are much less clear in true orchestral audio recordings.

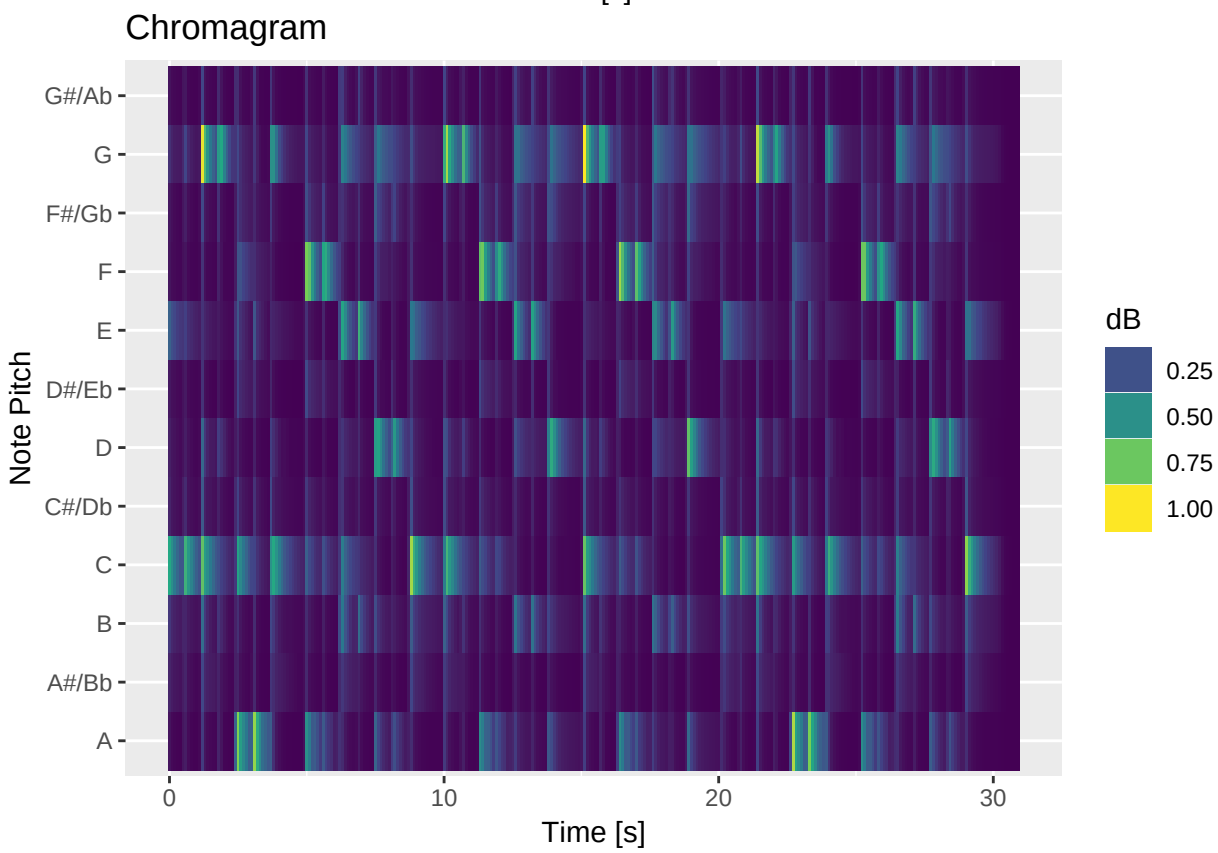
```
freq2midi <- function(freq, A4 = 440){
  p <- round(69 + 12*log2(freq/A4))
  ## Check for 0
  if(any(p <= 0)){
    p[which(p <=0)] = 0
  }
  return(p)
}

## Convert MIDI pitch to frequency
midi2freq <- function(pitch, A4 = 440){
  f <- A4*2^((pitch - 69)/12)
  return(f)
}

## Convert MIDI pitch (0-127) to chroma number,
## ref  is MIDI pitch of lowest note that chroma starts with, default is A0
# A0 = 21
# B0 = 23
# C1 = 24
# D1 = 26
# E1 = 28
# F1 = 29
# G1 = 31
midi2chroma <- function(midi, ref = 21){
  chrom <- (midi - ref)%%12 + 1
  labs <- c("A", "A#/Bb", "B", "C", "C#/Db", "D", "D#/Eb", "E", "F",
            "F#/Gb", "G", "G#/Ab")
  ind <- ref - 21 + 1
  ## Reorder labels
  labels <- c(labs[ind:length(labs)], labs[1:ind-1])
  return(list(chroma = chrom, labels = labels))
}
```

```
curr <- readMP3("Music/Twinkle-Twinkle/Twinkle-Twinkle-two-piece-ref.mp3")
sr <- curr@samp.rate
pieces <- curr@left
ind <- which(pieces > 0 )
ind1 <- ind[1] ## remove silence before piece
ind2 <- tail(ind, n = 1) ## remove silence from end of piece
```

```
curr <- pieces[ind1:ind2]
e.twinkle <- encode(curr, sr, window.length = 4410*2, A4 = 440, plot = TRUE)
```




MIDIgram
MIDI Pitch
120
110
100
90
80
70
60
50
40
30
20
10
0
0
10
20
30
Time [s]
dB
0.25
0.50
0.75
1.00


Chromagram
Note Pitch
G#/Ab
G
F#/Gb
F
E
D#/Eb
D
C#/Db
C
B
A#/Bb
A
0
10
20
30
Time [s]
dB
0.25
0.50
0.75
1.00

```
# Can aditionall plot each reprsentation individually
#plot.chrom(e.twinkle$chromagram, e.twinkle$labels, e.twinkle$spec$t)
```

2. Distance Decompositions

Orchestral performances can differ across several different dimensions. The musical dimensions across which orchestras can differ that are considered here are:

- Tempo
- Volume
- Balance
- Timbre

Each decomposition is performed to find a density over time (normalized between 0 and 1) so that density distance metrics can be calculated to compare the various decompositions. `process.curves` is the main function to encode each recording and calculate the tempo, volume and balance data decompositions, to explore balance and volume by frequency group and to calculate the timbre based decompositions.

```
folder <- "Music/Twinkle-Twinkle/"

file.list <- c("Twinkle-Twinkle-two-piece-ref.mp3",
               "Twinkle-Twinkle-two-piece-query.mp3")
A4.list <- c(440, 442) ## tunings by orchestra


## Select Octaves and Upper Harmonics Chuncked -  account for tuning
## Octaves starting from A0 to A7, then three bins upper harmonics
freq.list <- seq(0, 22045, 5)

## Start and End frequencies
f.inds1 <- which(freq.list %in% c(30, 55, 110, 220, 440, 880, 1760, 3520, 5000, 10000))
f.inds2 <- c((f.inds1 - 1)[-1], length(freq.list))

nMFCC <- 13 # Number of MFCC coefficients
## Read in each recording and process different decompositions
twinkle.process <- process.curves(file.list, folder, A4.list, ref.word = "ref",
                                  f.inds1, f.inds2, nMFCC, wintime = 0.05, hoptime = 0.1,
                                  window.length = 4410*2)
```

```
## [1] "Twinkle-Twinkle-two-piece-ref.mp3"
## [1] "Twinkle-Twinkle-two-piece-query.mp3"
```

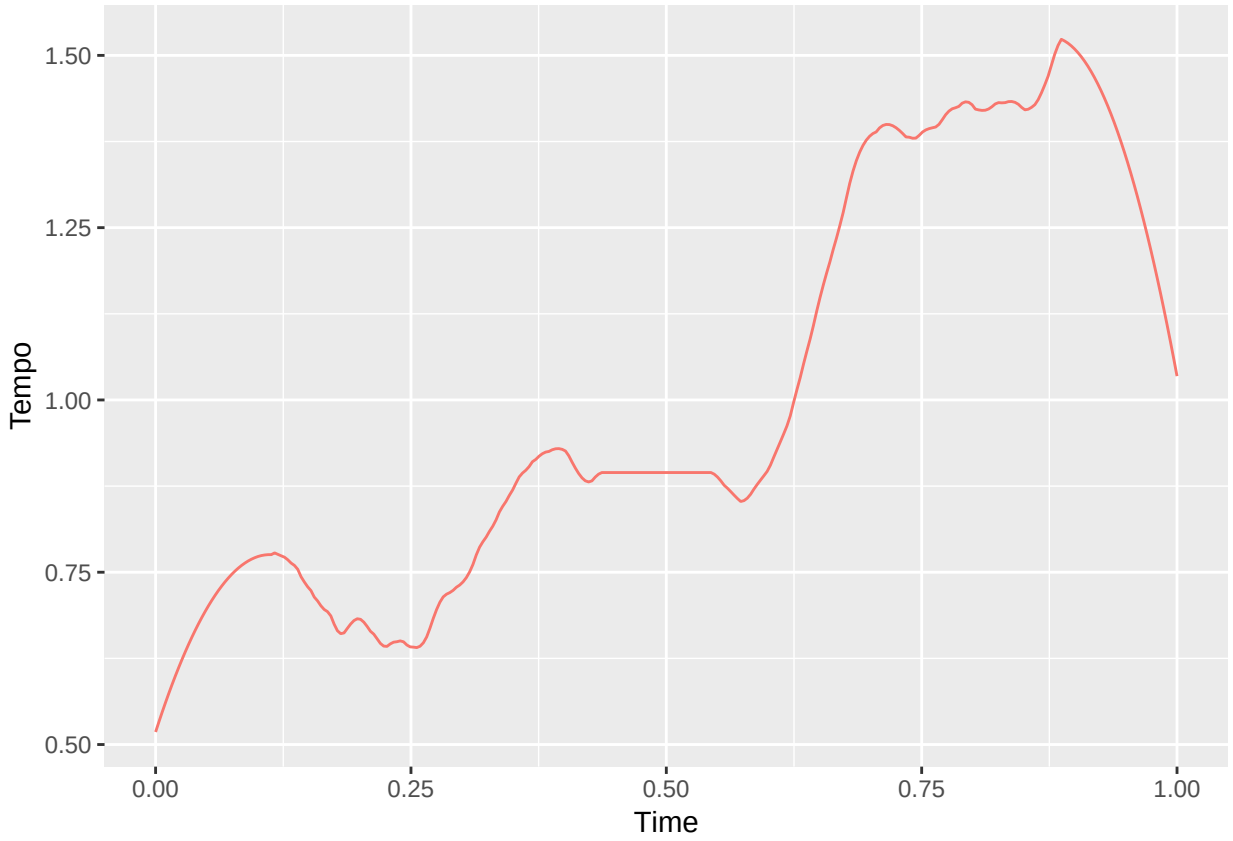
2.1 Alignment

Before any data decompositions are calculated, the recordings need to be aligned temporally so that each decomposition only measures the feature of interest and is invariant to timing differences. As each recording is of the same piece and thus must include the same notes, Dynamic Time Warping is used to temporally align the different recordings before further processing. The alignment is performed on the chromagram data representation.

2.2 Tempo

The alignment indices from the dynamic time warping alingment step are used to calculate differences in tempo curves between different orchestras. The derivative of the indices is found using the Savitzky-Golay filter to yield the tempo density curves.

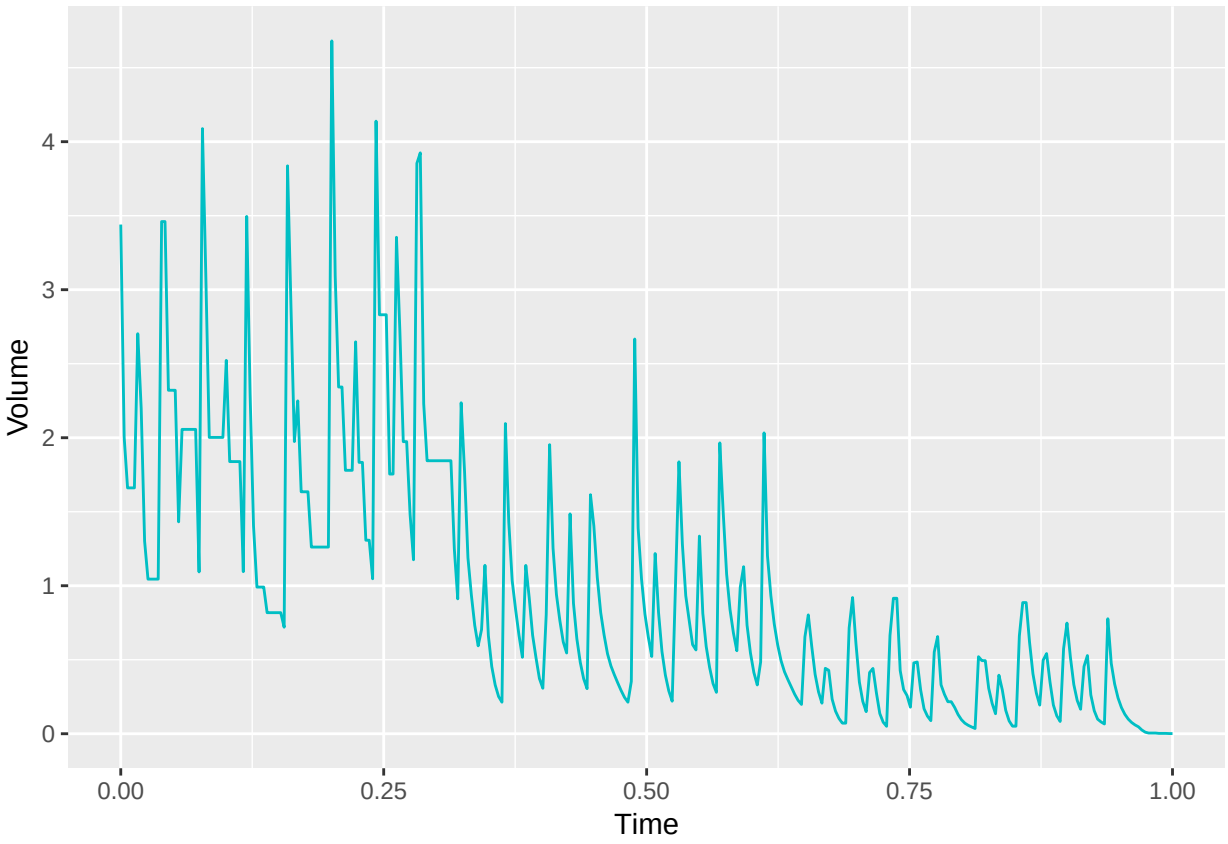
```
plot.df <- data.frame(do.call(cbind, twinkle.process[1:3]))
colnames(plot.df) <- c("Tempo", "Volume", "SF")
plot.df$Time <- seq(0, 1, length.out = nrow(plot.df))

ggplot(plot.df, aes(x = Time, y = Tempo)) +
  geom_line(color = "#F8766D")
```



2.3 Volume

The volume curves are calculated by finding the overall volume at each point in time for the aligned chromagrams and then normalizing by the average volume for the entire piece. The sharp peaks in the volume curves are from the note onsets, which are quite evident in the piano MIDI recordings considered here.

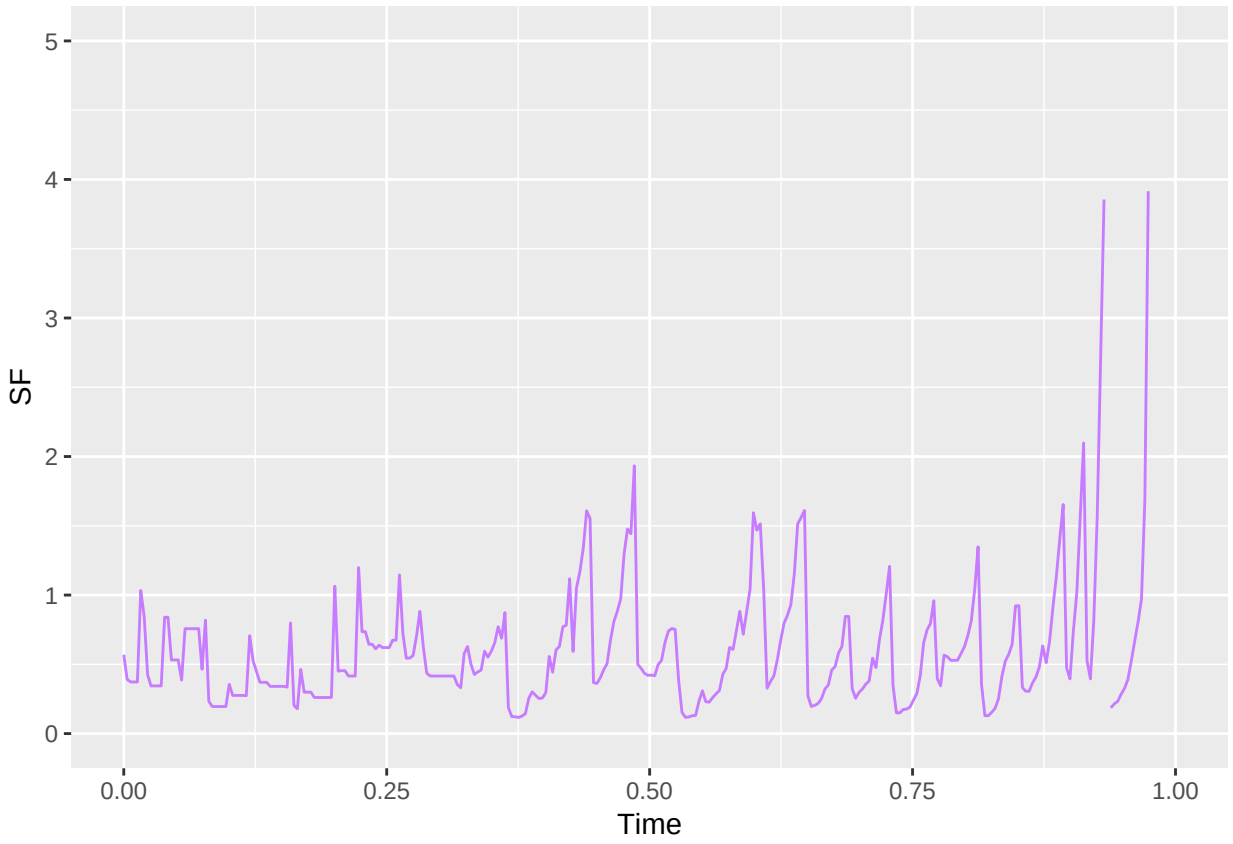
```
ggplot(plot.df, aes(x = Time, y = Volume)) +
  geom_line(color = "#00BFC4")
```



2.4 Spectral Flatness - Overall

Spectral flatness is the ratio of the geometric and arithmetic means of the amplitude of a signal and is a measure of the tonality of a recording. This measure can give an indication of the timbre of different orchestras, and is normalized to be a density.

```
ggplot(plot.df, aes(x = Time, y = SF)) +
  geom_line(color = "#C77CFF") +
  ylim(0, 5)
```

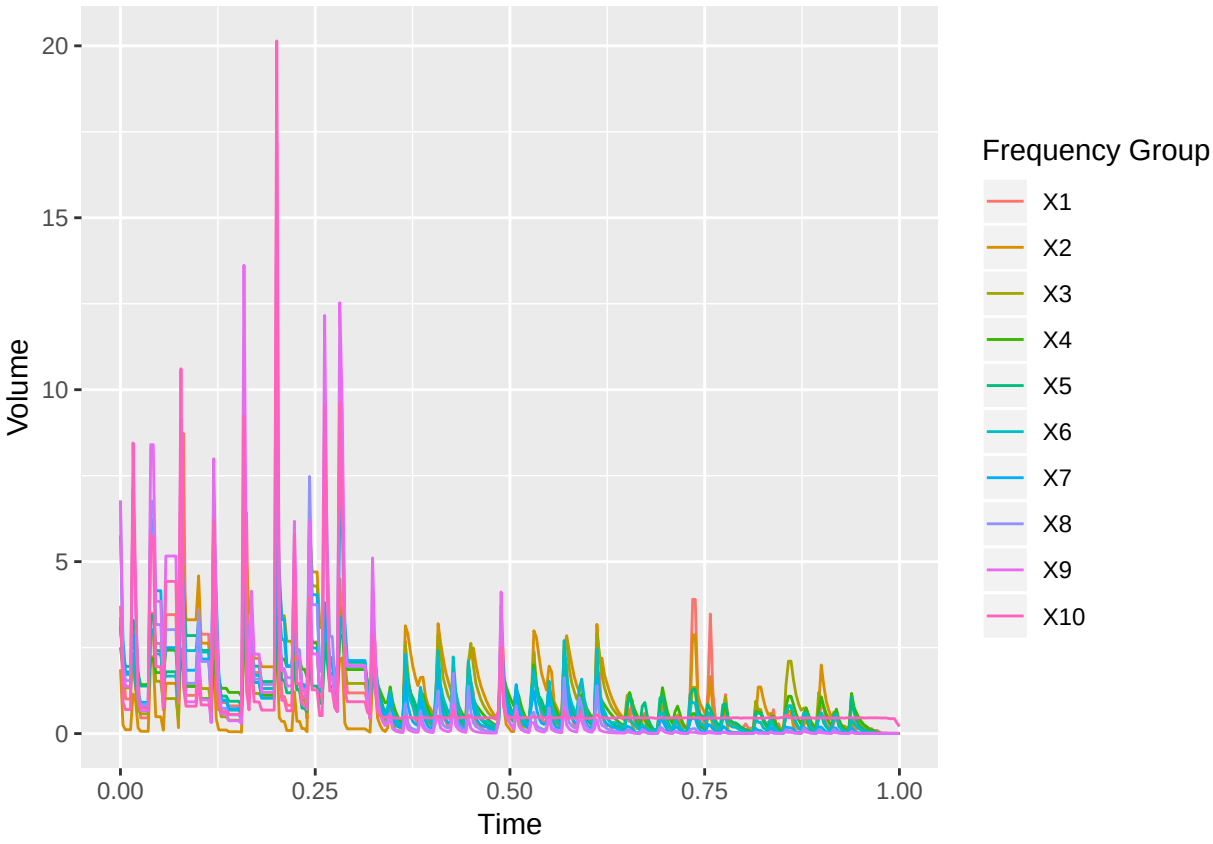


2.5 Balance - Volume by Frequency

The balance of different orchestras is apparent in the relative volume of the different instruments and sections. Ideally, this could be explored by extracting each instrument's volume over time from the spectrogram. However, the decomposition of an audio signal to each instrument is very challenging and an open research problem. To approximate the volume of each instrument, the volume is calculated over a frequency range, approximating the volume of different groups of instruments.

f.inds1 indicates the frequency indices in the spectrogram at which to start a grouping, while *f.inds2* indicates the ending indices for each frequency group. 10 groupings are considered here, 1 for each octave, A0 - A7, and three groupings for upper harmonics. Different orchestra tunings can be accounted for by selecting the appropriate frequencies to correspond to the desired range of note pitches.

```
vol.df <- data.frame(t(twinkle.process$vol.freq))
vol.df$Time <- seq(0, 1, length.out = nrow(vol.df))


ggplot(melt(vol.df, id = "Time"), aes(x = Time, y = value, color = variable)) +
  geom_line() +
  labs(y = "Volume", color = "Frequency Group")
```



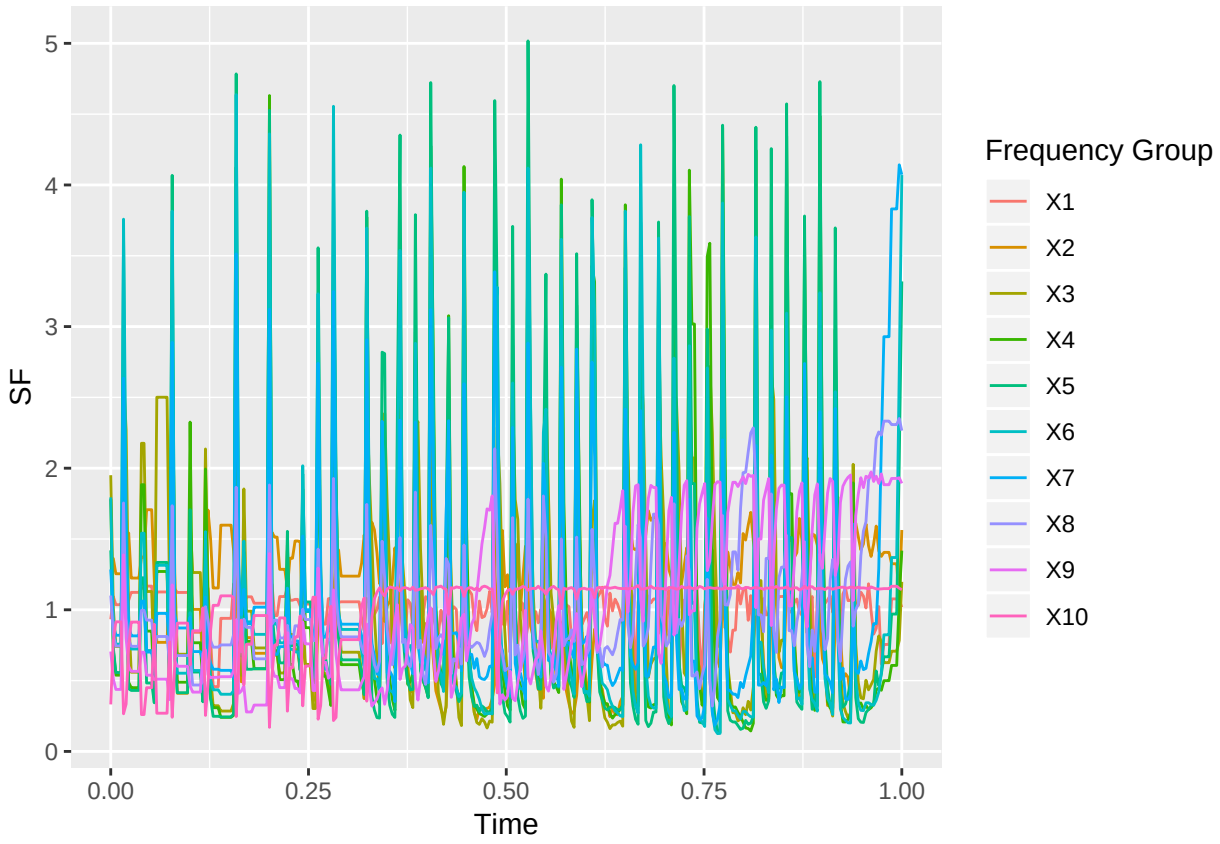
2.6 Timbre

Timbre can be described as the "color" of a sound and allows listeners to distinguish between different instruments playing the same note pitch. Timbre is a subjective concept and is difficult to measure from an audio file. However, it is one of the principal dimensions along which recordings by different orchestras can differ, as many orchestras have a distinctive "sound". Some aspects of timbre are evident in the upper harmonics of a spectrum, as more "resonant" tones lead to more higher harmonics above the fundamental frequency pitch.

2.6.1 Spectral Flatness by Frequency

As spectral flatness measures the tonality of a sound, it can be used to explore timbre. However, the timbre of an orchestra differs across instruments, so, similar to the volume by frequency approach used to explore balance, the spectral flatness is calculated for each of the specified frequency groups to explore timbre by octave/instrument group.

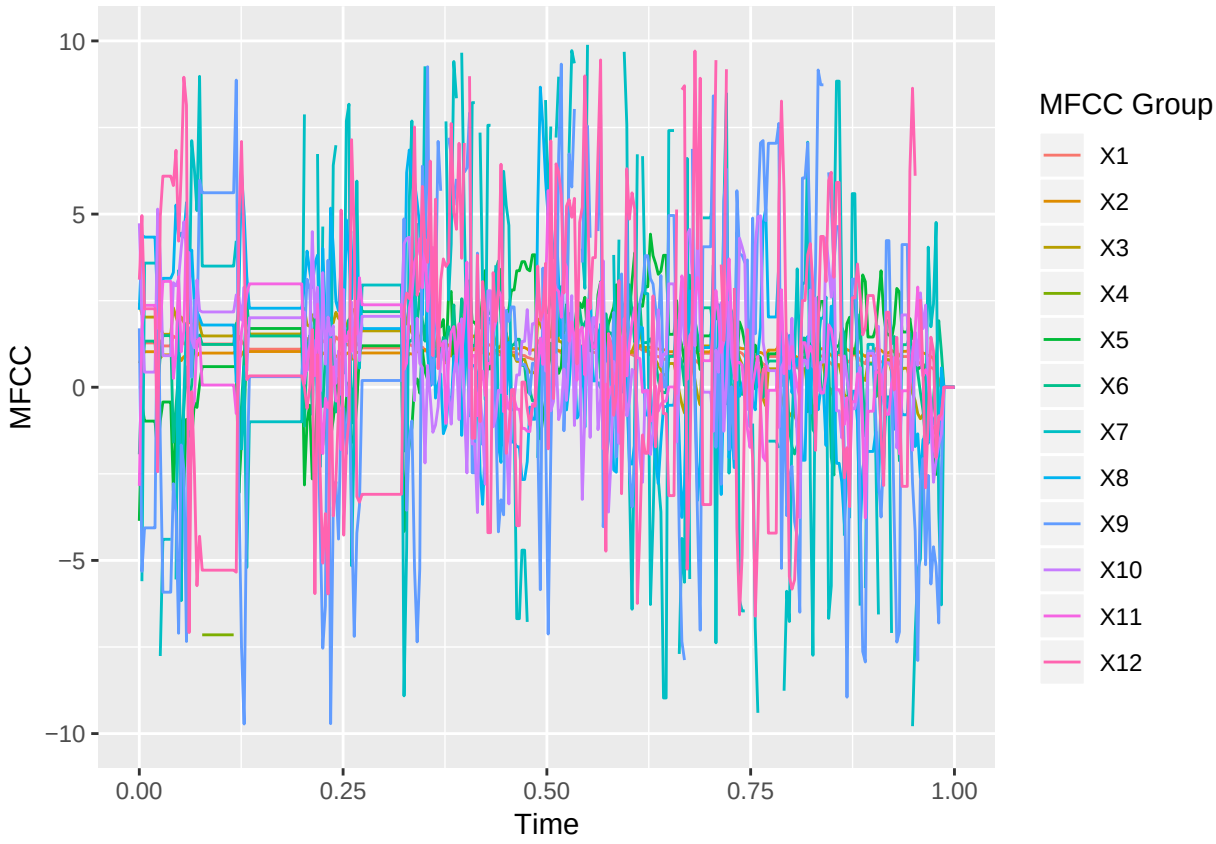
```
sf.df <- data.frame(t(twinkle.process$sf.freq))
sf.df$Time <- seq(0, 1, length.out = nrow(sf.df))


ggplot(melt(sf.df, id = "Time"), aes(x = Time, y = value, color = variable)) +
  geom_line() +
  labs(y = "SF", color = "Frequency Group")
```



2.6.2 Mel-Frequency Cepstrum Coefficients (MFCC)

MFC (mel-frequency cepstrum) coefficients are used often in speech recognition and music information retrieval and also are an approximate measure of timbre. MFCC values are essentially spectrogram values from the FFT warped to the mel-scale and then the discrete cosine transform of the log of these values is taken to give the MFCC. The mel-scale is supposed to better represent how humans perceive frequency, that is the scale places pitches at what would be heard as equal spacing.

```
mfcc.df <- data.frame(t(twinkle.process$mfcc))
mfcc.df$Time <- seq(0, 1, length.out = nrow(mfcc.df))


ggplot(melt(mfcc.df, id = "Time"), aes(x = Time, y = value, color = variable)) +
  geom_line() +
  labs(y = "MFCC", color = "MFCC Group") +
  ylim(-10, 10)
```


MFCC
10
5
0
−5
−10
0.00
0.25
0.50
0.75
1.00
Time
MFCC Group
X1
X2
X3
X4
X5
X6
X7
X8
X9
X10
X11
X12