WorkshopIII - Project 3 Report

Jack 1930026143

Exercise 1

Step1: Basic analysis

1. In order to duel with the sounds in the wave form, we should install and use the package of "*tuner*" and "*seewave*" to process the wave data.

```
install.packages("seewave")
install.packages("tuneR")
library(seewave)
library(tuneR)
```

2. Read and get the .wav file by the command *readwave*(path/filename.wav) and the information of s is shown as follow:

```
Wave Object
        Number of Samples:      61740
        Duration (seconds):     1.4
        Samplingrate (Hertz):   44100
        Channels (Mono/Stereo): Mono
        PCM (integer format):   TRUE
        Bit (8/16/24/32/64):    16
```

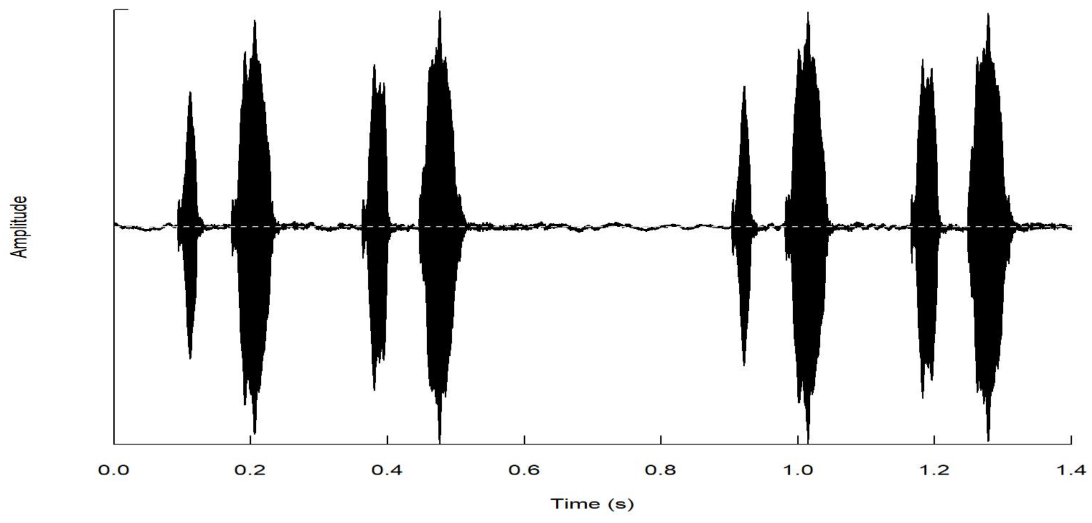
3. Get the class and duration of the wave, we can find the wave is from the package *tuneR*.

```
> class(s)
[1] "Wave"
attr(,"package")
[1] "tuneR"
```

```
> duration(s)
[1] 1.4
```

Step2: Oscillogram

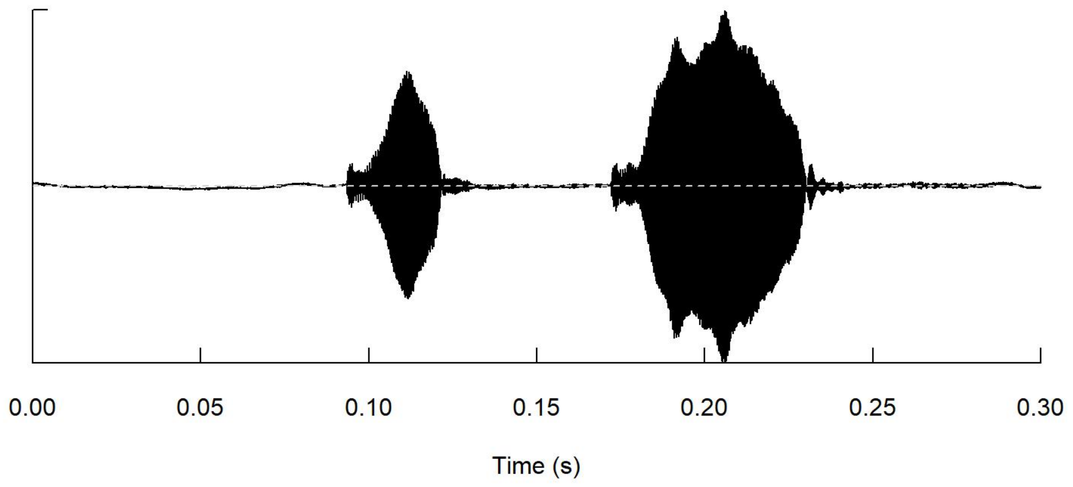
1. In this step, we should extract the wave of the first call according to the time. From the original wave which has 4 calls by the command *oscillo*(s), and the first call is approximately from 0 to 0.3 second.



2. Then we can cut the pitch of first call and name it $c1$.

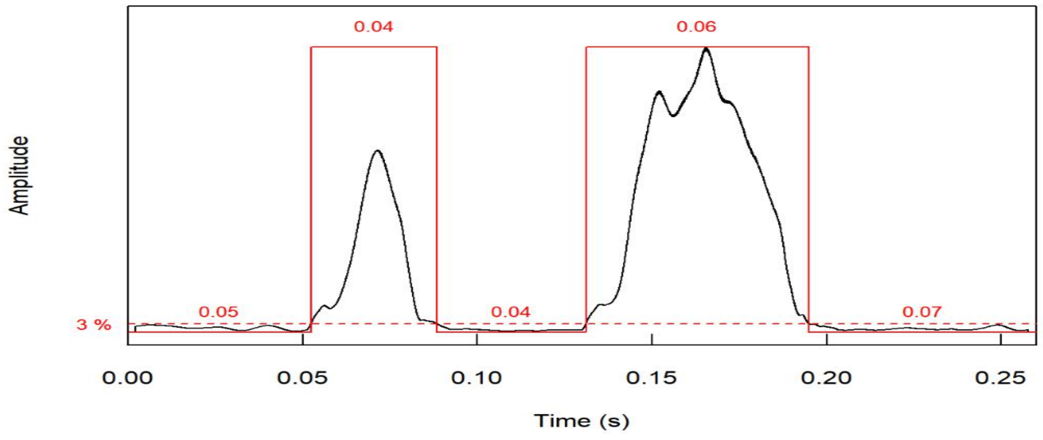
```
c1 <- extractWave(s, from=0, to=.3, xunit="time")
```

Show the *oscillo* gram of $c1$ and we can see the patch:



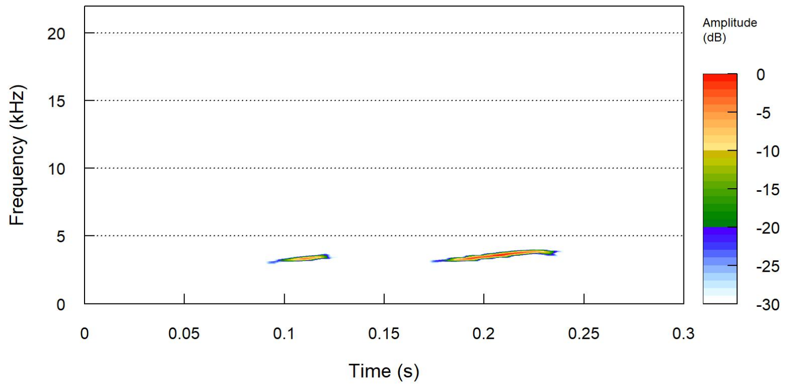
Step3: Temporal Analysis

In this part, we should take manual time measurements to c1 by the *timer*() method, and we set the parameter *threshold* to 3 which mean if the fluctuation of Amplitude expand the 3%, it will be marked out by the red line.

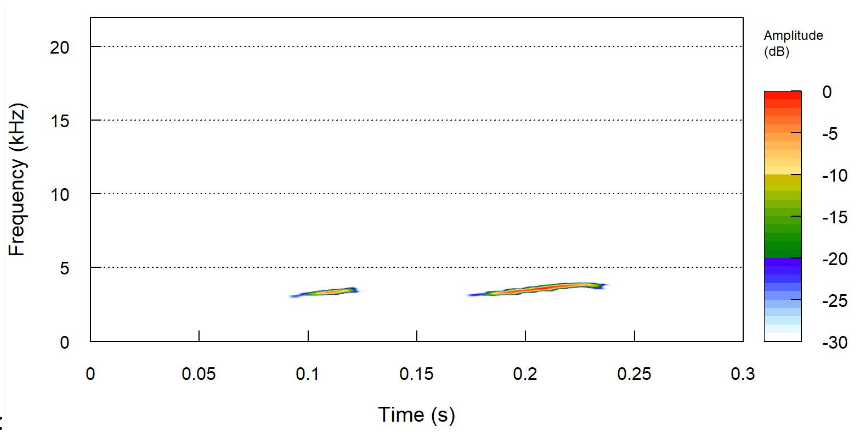


Step4. Dominant Frequency

1. In this step, we should to visualize the first call wave which was cut in the Step2 with the function *spectro*(). Firstly, plot the normal spectrogram of $c1$:

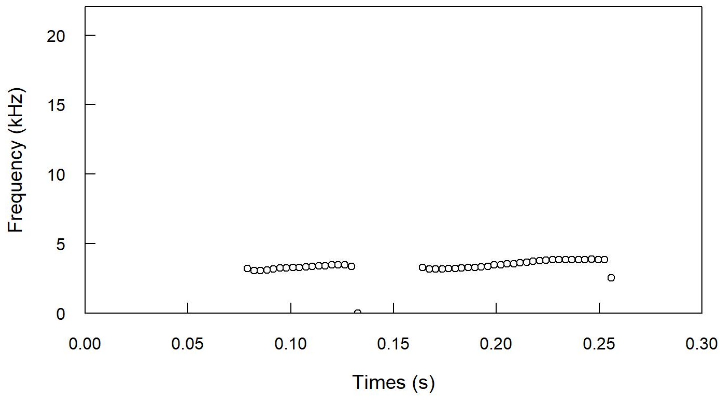


And then adjust the parameter $l = 1024, ovlp = 87.5$ which means to use the Fourier window:

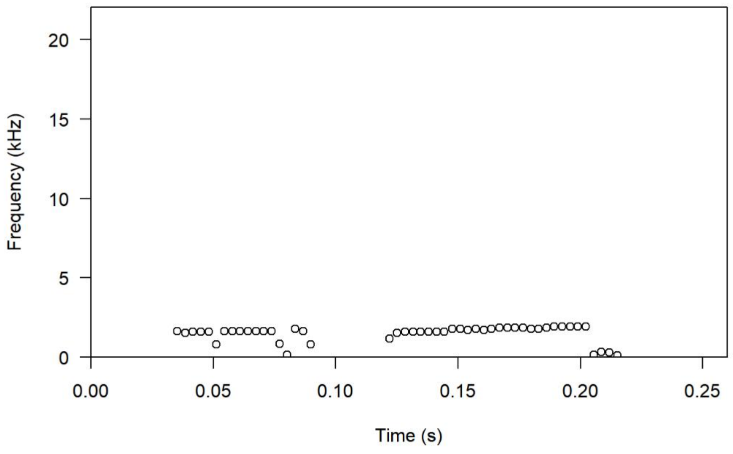


Actually, there is a slight difference between the two figures.

2. Get the dominant frequency and fundamental frequency of $c1$:



by the $freq()$



by the $fund()$

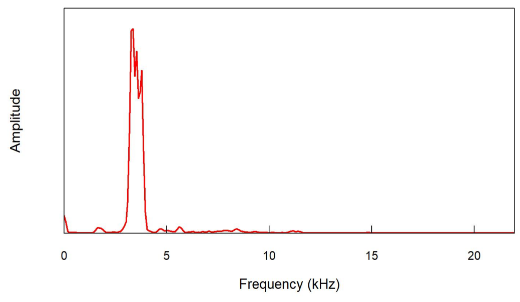
Step5: Spectral Analysis

In this step, we should calculate the average spectrum of the first note of $c1$, and get the main feature of the average spectrum, the peak frequency.

1. Get the timer of the $c1$ which contains the summary information: each start time, end time and other basic feathers.

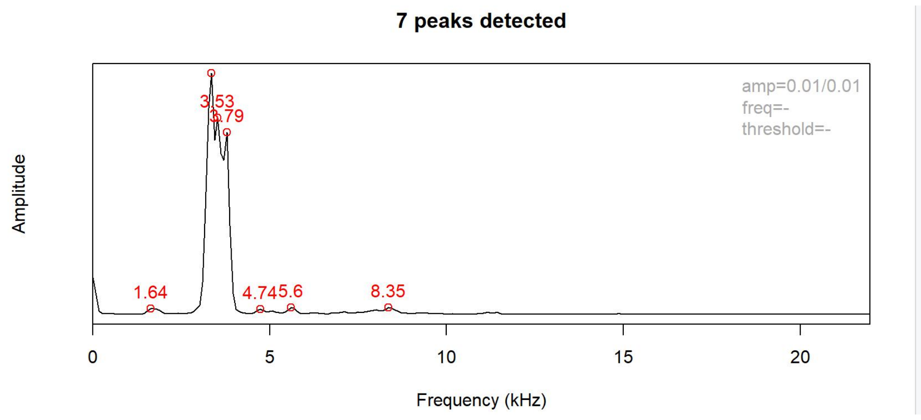
```
> timer
$s
[1] 0.03696425 0.06558319

$p
[1] 0.09166226 0.04163580 0.06415451

$r
[1] 0.5193522

$s.start
[1] 0.09166226 0.17026230

$s.end
[1] 0.1286265 0.2358455

$first
[1] "pause"
```

2. Then we can Compute the mean frequency spectrum of c1:



```
> head(specmean, 10)
               x           y
 [1,] 0.00000000 0.153097870
 [2,] 0.08613281 0.090987004
 [3,] 0.17226563 0.014145029
 [4,] 0.25839844 0.004755947
 [5,] 0.34453125 0.004049366
 [6,] 0.43066406 0.003745740
 [7,] 0.51679688 0.004074558
 [8,] 0.60292969 0.002905195
 [9,] 0.68906250 0.001924478
[10,] 0.77519531 0.002034710
```

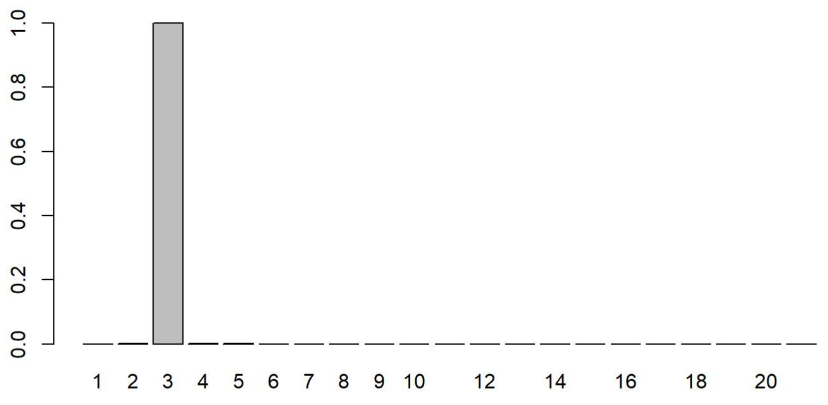
3. Find the main frequency peaks:



4. Get the main properties and dominant frequent:

```
> domfreq
      x       y
3.35918 1.00000
```

5. Plot the spectrogram of frequencies:



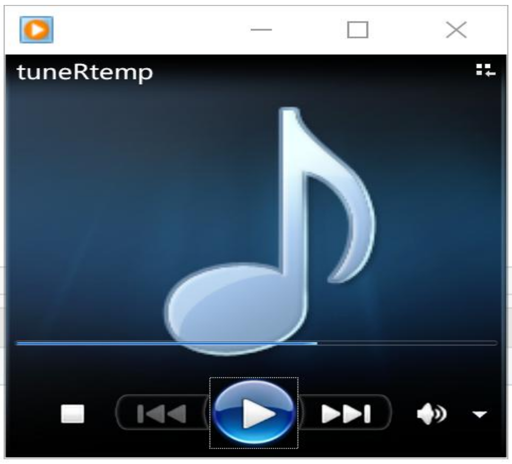
Exercise 2:

Step1: Basic operation

1 . Install and load the libraries and set the path

```
library(tuneR)
library(seewave)
# setwd("E:/Desktop_xsj/Desktop/Second Semester of Junior/WorkshopIII/Week8-9 - Project3-20220215/data/")
path <- "E:/Desktop_xsj/Desktop/Second Semester of Junior/WorkshopIII/Week8-9 - Project3-20220215/data/"
```

2. Play a file by the command $tuneR::play(paste0(path, "Allobates\_femoralis.wav"))$



Step2: Generate a Sound

1. Define the duration and the Hz and generalize the wave

```
# A duration of 3 seconds
t = seq(0, 3, 1/8000)
# 440 Hz sine wave
u = (2^15 - 1) * sin(2 * pi * 440 * t)
# Generate the wave
w = Wave(u, samp.rate = 8000, bit=16)
```

2. Summary the wave w and u, we can find the results are the same.

```
> summary(u)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
 -32767  -23170       0       0   23170   32767 
> summary(w) # Same as u

Wave Object
	Number of Samples:      24001
	Duration (seconds):     3
	Samplingrate (Hertz):   8000
	Channels (Mono/Stereo): Mono
	PCM (integer format):   TRUE
	Bit (8/16/24/32/64):    16

Summary statistics for channel(s):

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
 -32767  -23170       0       0   23170   32767 
```

Step3: Play a Series of Notes:

1. Initialization

```
sr = 8000;    # The sampling rate used in playing function
tatum= .1;    # The length of time (secs) of the smallest musical time unit
```

2. Generate a series of the pitch in random probabilities, we set the probabilities to 0.8, which means that if this time have the 80% to generate note lo, then the next time have the 80% to generate note hi.

```
lo <- 80;
hi <- 100;
p <- .8
notes <- 50
s <- rep(0,notes);
s[1] <- lo;
up <- 1

# fill out the list s
for (i in 2:notes){
    if (up){  # up==1
        s[i] <- min(s[i-1]+1, hi);
    }  # if up move upward if possible
    else{
        s[i] <- max(lo, s[i-1]-1);
    }
    # Generate the random music
    if (runif(1) > p)
        up <- 1-up;
}
```

Step4: Convert the sound file to digit

1. Build the note-midi table and based on the table, we can generate the specific music. Here we just need the 5th symphony: $sol\, sol\, sol\, mi\, fa\, fa\, fa\, re$ by matching the notation of notes.

```
table1 <-data.frame("Midivalue" = seq(68,100), "American notation" =
                       c("G","A","A","B","C","C","D","D","E","F",
                         "F","G","G","A","A","B","C","C","D","D",
                         "E","F","F","G","G","A","A","B","C","C",
                         "D","D","E"))
```

2. At the comment part, we can see the relationship of notes and the midi values and store in a list.

```
# Note:       C   D   E   F   G   A   B
# Syllable: DO  RE  MI  FA  SOL  LA  SI
playit(c(91, 91, 1, 91, 91, 1, 91, 91, 1, 87, 87, 87, 87, 87,
         87, 87, 1, 89, 89, 1, 89, 89, 1, 89, 89, 1, 86, 86,
         86, 86, 86, 86, 86) , paste0(path,"beethoven.wav"))
```

3. Then we can write a function to Convert the sound file to digit (specific value).

```
transcribeMusic <- function(wavFile, widthSample = 4096) {
    perioWav <- periodogram(wavFile, width = widthSample )
    freqWav <- FF(perioWav)
    noteWav <- noteFromFF(freqWav)
    noteWavNames <- noteWav[!is.na(noteWav)]
    print(noteWavNames)
    print(notenames(noteWavNames))
    return(notenames(noteWavNames))
}
```

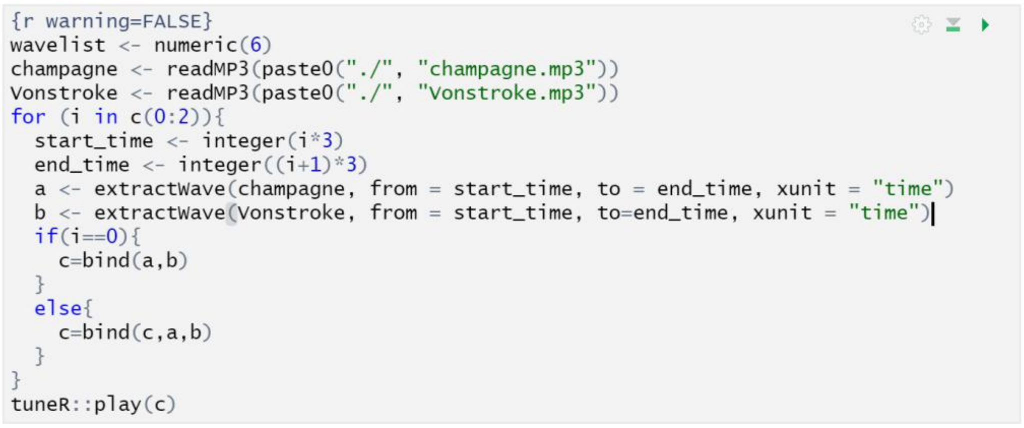
Here we have four steps:

1. Get the object of wave spec.
2. Obtain the fundamental frequencies
3. Acquire the notes from the frequencies in the second step.
4. Remove the NaN data in the end

Step4: Music Mixture

In this part, we should splice the two music with the MP 3 form.

1. Read the mp3 file
2. Use a for loop to divide the two music into $\frac{n}{3}$.
3. Stack each slice with the $bind()$.



```
{r warning=FALSE}
wavelist <- numeric(6)
champagne <- readMP3(paste0("./", "champagne.mp3"))
Vonstroke <- readMP3(paste0("./", "Vonstroke.mp3"))
for (i in c(0:2)){
  start_time <- integer(i*3)
  end_time <- integer((i+1)*3)
  a <- extractWave(champagne, from = start_time, to = end_time, xunit = "time")
  b <- extractWave(Vonstroke, from = start_time, to=end_time, xunit = "time")
  if(i==0){
    c=bind(a,b)
  }
  else{
    c=bind(c,a,b)
  }
}
tuneR::play(c)
```

Step5. Stack two slices of music

1. Read the MP3 file first.
2. Read it as an MP3 using the $writeWave$ function
3. Write the entire object as wav.
4. Use $extractWave$ to get part of music,
5. Stack the pieces together.

And we can see the two example to stack the two parts of music.

```
wav_ch <- readWave( paste0(path, "champagne.wav"))
wav_vo <- readWave( paste0(path, "Vonstroke.wav"))
wav_ch_1 <- extractWave(wav_ch, from = 0, to = 3, xunit = "time")
wav_ch_2 <- extractWave(wav_ch, from = 3, to = 6, xunit = "time")
wav_ch_3 <- extractWave(wav_ch, from = 6, to = 9, xunit = "time")
wav_vo_1 <- extractWave(wav_vo, from = 0, to = 3, xunit = "time")
wav_vo_2 <- extractWave(wav_vo, from = 3, to = 6, xunit = "time")
wav_vo_3 <- extractWave(wav_vo, from = 6, to = 9, xunit = "time")

combi <- bind( wav_ch_1 , wav_vo_1, wav_ch_2, wav_vo_2, wav_ch_3, wav_vo_3)
tuneR::play(combi)
```

```
# Harmonious music
wav_sa_mp <- readMP3("satie_gymnopedie_no_1.mp3")
wav_lo_mp <- readMP3("bass-lounge.mp3")
writeWave(wav_sa_mp, filename = paste0(path, "satie_gymnopedie_no_1.wav") )
writeWave(wav_lo_mp, filename = paste0(path, "bass-lounge.wav") )

wav_sa <- readWave( paste0(path, "satie_gymnopedie_no_1.wav") )
wav_lo <- readWave( paste0(path, "bass-lounge.wav") )
wav_ch_1 <- extractWave(wav_sa, from = 400001, to = 1400000)
wav_ch_2 <- extractWave(wav_lo, from = 1, to = 1000000)
www = wav_ch_1+wav_ch_2
```

Step6. Mix the two music just use the above example.

1. The first part:

```
combi = bind( wav_ch_mp , wav_vo_mp )
tuneR::play(combi)
writeWave(combi, filename = paste0(path, "two_my.wav"))

wav_ch     <- readWave( paste0(path, "electro.wav") )
wav_vo     <- readWave( paste0(path, "Vonstroke.wav") )
wav_ch_1   <- extractWave(wav_ch, from = 0, to = 3, xunit = "time")
wav_ch_2   <- extractWave(wav_ch, from = 3, to = 6, xunit = "time")
wav_ch_3   <- extractWave(wav_ch, from = 6, to = 9, xunit = "time")
wav_vo_1   <- extractWave(wav_vo, from = 0, to = 3, xunit = "time")
wav_vo_2   <- extractWave(wav_vo, from = 3, to = 6, xunit = "time")
wav_vo_3   <- extractWave(wav_vo, from = 6, to = 9, xunit = "time")
combi      <- bind( wav_ch_1 , wav_vo_1, wav_ch_2, wav_vo_2, wav_ch_3, wav_vo_3)
tuneR::play(combi)
```

2. The second part:

```
wav_ch_mp <- readMP3("Vonstroke.mp3")
wav_el_mp <- readMP3("electro.mp3")
writeWave(wav_ch_mp, filename = paste0(path, "Vonstroke.wav") )
writeWave(wav_el_mp, filename = paste0(path, "electro.wav") )
wav_ch    <- readWave( paste0(path, "Vonstroke.wav") )
wav_el    <- readWave( paste0(path, "electro.wav") )  # this is mono, only has left channel
wav_ch_1  <- extractWave(wav_ch, from = 1, to = 500000)
wav_ch_2  <- extractWave(wav_el, from = 1, to = 500000)
www       <- Wave(wav_ch_1 @ left + wav_ch_2 @ left, samp.rate= 44100, bit=16) #if one is mono
tuneR::play(normalize(www, unit=c("16")))
```