Phonetic fieldwork and experiments with the `phonfieldwork` package for R

George Moroz

Linguistic Convergence Laboratory, NRU HSE, Moscow, Russia

15 November 2019

NRU HSE, School of linguistics, Moscow

Presentation is available here: tinyurl.com/yzd2yjr5

Margaux Dubuis

Most phonetic research consists of the following steps:

1. Formulate a research question. Think of what kind of data is necessary to answer this question, what is the appropriate amount of data, what kind of annotation you will do, what kind of statistical models and visualizations you will use, etc.
2. Create a list of stimuli.
3. Elicite list of stimuli from speakers who signed an Informed Consent statement, agreeing to participate in the experiment to be recorded on audio and/or video.
4. Annotate the collected data.
5. Extract the information from annotated data.
6. Create visualizations and evaluate your statistical models.
7. Report your results.
8. Publish your data.

Most phonetic research consists of the following steps:

1. Formulate a research question. Think of what kind of data is necessary to answer this question, what is the appropriate amount of data, what kind of annotation you will do, what kind of statistical models and visualizations you will use, etc.
2. Create a list of stimuli.
3. Elicite list of stimuli from speakers who signed an Informed Consent statement, agreeing to participate in the experiment to be recorded on audio and/or video.
4. Annotate the collected data.
5. Extract the information from annotated data.
6. Create visualizations and evaluate your statistical models.
7. Report your results.
8. Publish your data.

The `phonfieldwork` package is created for helping with items 3, partially with 4, and 5 and 8.

Why/when do you need the `phonfieldwork` package?

These ideal plan hides some technical subtasks:

- creating a presentation for elicitation task
- renaming and concatenating multiple sound files recorded during a session
- automatic annotation in Praat TextGrids [Boersma and Weenink 2019]
- creating a searchable `.html` table with annotations, spectrograms and ability to hear sound
- converting multiple formats (Praat, ELAN [Brugman et al. 2004] and EXMARaLDA [Schmidt and Wörner 2009])

Why/when do you need the `phonfieldwork` package?

These ideal plan hides some technical subtasks:

- creating a presentation for elicitation task
- renaming and concatenating multiple sound files recorded during a session
- automatic annotation in Praat TextGrids [Boersma and Weenink 2019]
- creating a searchable `.html` table with annotations, spectrograms and ability to hear sound
- converting multiple formats (Praat, ELAN [Brugman et al. 2004] and EXMARaLDA [Schmidt and Wörner 2009])

All of these tasks can be solved by a mixture of different tools:

- any programming language can handle automatic file renaming
- Praat contains scripts for concatenating and renaming files

Why/when do you need the `phonfieldwork` package?

These ideal plan hides some technical subtasks:

- creating a presentation for elicitation task
- renaming and concatenating multiple sound files recorded during a session
- automatic annotation in Praat TextGrids [Boersma and Weenink 2019]
- creating a searchable `.html` table with annotations, spectrograms and ability to hear sound
- converting multiple formats (Praat, ELAN [Brugman et al. 2004] and EXMARaLDA [Schmidt and Wörner 2009])

All of these tasks can be solved by a mixture of different tools:

- any programming language can handle automatic file renaming
- Praat contains scripts for concatenating and renaming files

`phonfieldwork` provides a functionality that will make it easier to solve those tasks independently of any additional tools. You can also compare the functionality with other R packages:
`rPraat` [Bořil and Skarnitzl 2016] and `textgRid` [Reidy 2016], `pympi` [Lubbers and Torreira 2013]

Philosophy of the `phonfieldwork` package

- each stimulus as a separate file
- researcher carefully listens to consultants to make sure that they are producing the kind of speech they wanted
- in case a speaker does not produce three clear repetitions, researcher ask them to repeat the task

There are some phoneticians who prefer to record everything, for language documentation purposes. I think that should be a separate task. If you insist on recording everything, it is possible to run two recorders at the same time: one could run during the whole session, while the other is used to produce small audio files. You can also use special software to record your stimuli automatically on a computer (e.g. `PsychoPy` [Peirce et al. 2009]).

Let's go through **phonfieldwork** functionality

Let's go through **phonfieldwork** functionality:

- install the package
- create a presentation based on a list of stimuli
- rename collected data
- merge all data together
- automatically annotate your data
- extract annotated fragments
- visualize your data
- create an **.html** viewer
- cite the package

Install the package

- Install the package from CRAN:

```
install.packages("phonfieldwork")
```

- ...or install it from Github:

```
install.packages("devtools")
devtools::install_github("agricolamz/phonfieldwork")
```

Install the package

- Install the package from CRAN:

```
install.packages("phonfieldwork")
```

- ...or install it from Github:

```
install.packages("devtools")
devtools::install_github("agricolamz/phonfieldwork")
```

- Load the package:

```
library("phonfieldwork")
packageVersion("phonfieldwork")

## [1] '0.0.3'
```

Create a presentation based on a list of stimuli

There are several ways to enter information about a list of stimuli into R:

- listing stimuli with the `c()` function inside R

```
c("tip", "tap", "top")
```

```
[1] "tip" "tap" "top"
```

- importing `.csv` file inside R using the `read.csv()` function:

```
read.csv("my_stimuli_df.csv")
```

```
  stimuli vowel
1     tip     ɪ
2     tap     æ
3     top     ɒ
```

- importing data from `.xls` or `.xlsx` file inside R using the `read.csv()` function:

```
read.csv("my_stimuli_df.csv")
```

```
  stimuli vowel
1     tip     ɪ
2     tap     æ
3     top     ɒ
```

Create a presentation based on a list of stimuli

- Now we are ready for creating a presentation for elicitation:

```
create_presentation(stimuli = my_stimuli$stimuli,
                    output_file = "first_example",
                    output_dir = getwd())
```

  Here is the result.

- Here is another example with translations:

```
create_presentation(stimuli = my_stimuli$stimuli,
                    translation = c("чаевые", "кран", "верхушка"),
                    output_file = "first_example",
                    output_dir = getwd())
```

  Here is the result.

Rename collected data

After collecting data and removing soundfiles with unsuccesful elicitations, one could end up with the following structure:

```
## ├── s1
## │   ├── 01.wav
## │   ├── 02.wav
## │   └── 03.wav
## └── s2
##     ├── 01.wav
##     ├── 02.wav
##     └── 03.wav
```

Rename collected data

Let's rename the files:

```
rename_soundfiles(stimuli = my_stimuli$stimuli,
                  prefix = "s1_",
                  path = "s1/")
```

```
## ├── s1
## │   ├── backup
## │   │   ├── 01.wav
## │   │   ├── 02.wav
## │   │   └── 03.wav
## │   ├── s1_tap.wav
## │   ├── s1_tip.wav
## │   └── s1_top.wav
## └── s2
##     ├── 01.wav
##     ├── 02.wav
##     └── 03.wav
```

Rename collected data

```
rename_soundfiles(stimuli = my_stimuli$stimuli,
                  prefix = "s2_",
                  suffix = paste0("_", 1:3),
                  path = "s2/",
                  backup = FALSE)
```

```
## ├── s1
## │   ├── backup
## │   │   ├── 01.wav
## │   │   ├── 02.wav
## │   │   └── 03.wav
## │   ├── s1_tap.wav
## │   ├── s1_tip.wav
## │   └── s1_top.wav
## └── s2
##     ├── s2_tap_2.wav
##     ├── s2_tip_1.wav
##     └── s2_top_3.wav
```

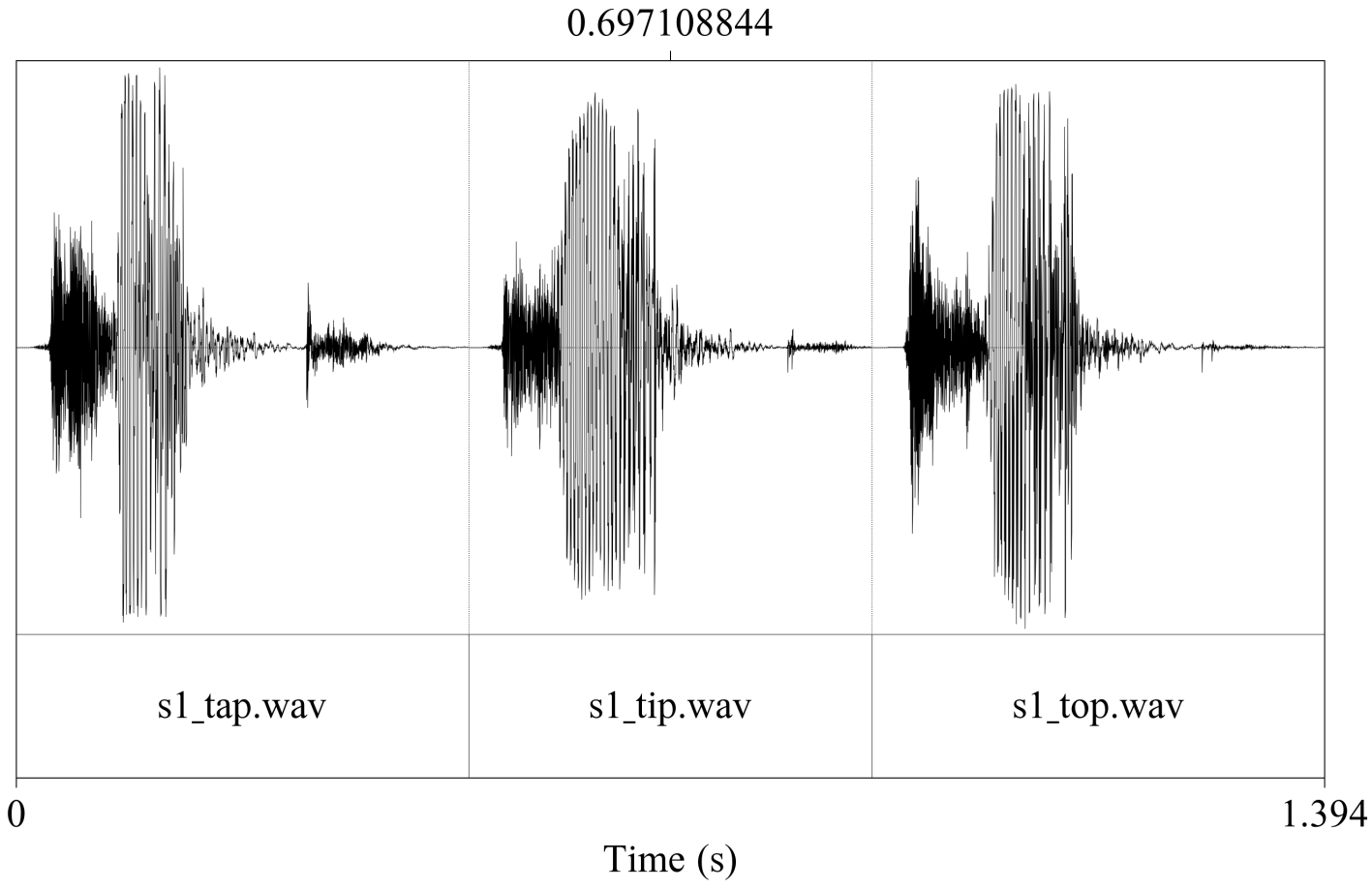
Merge all data together

```
concatenate_soundfiles(file_name = "s1_all",
                       path = "s1/")
```

```
## ├── s1
## │   ├── backup
## │   │   ├── 01.wav
## │   │   ├── 02.wav
## │   │   └── 03.wav
## │   ├── s1_all.TextGrid
## │   ├── s1_all.wav
## │   ├── s1_tap.wav
## │   ├── s1_tip.wav
## │   └── s1_top.wav
## └── s2
##     ├── s2_tap_2.wav
##     ├── s2_tip_1.wav
##     └── s2_top_3.wav
```

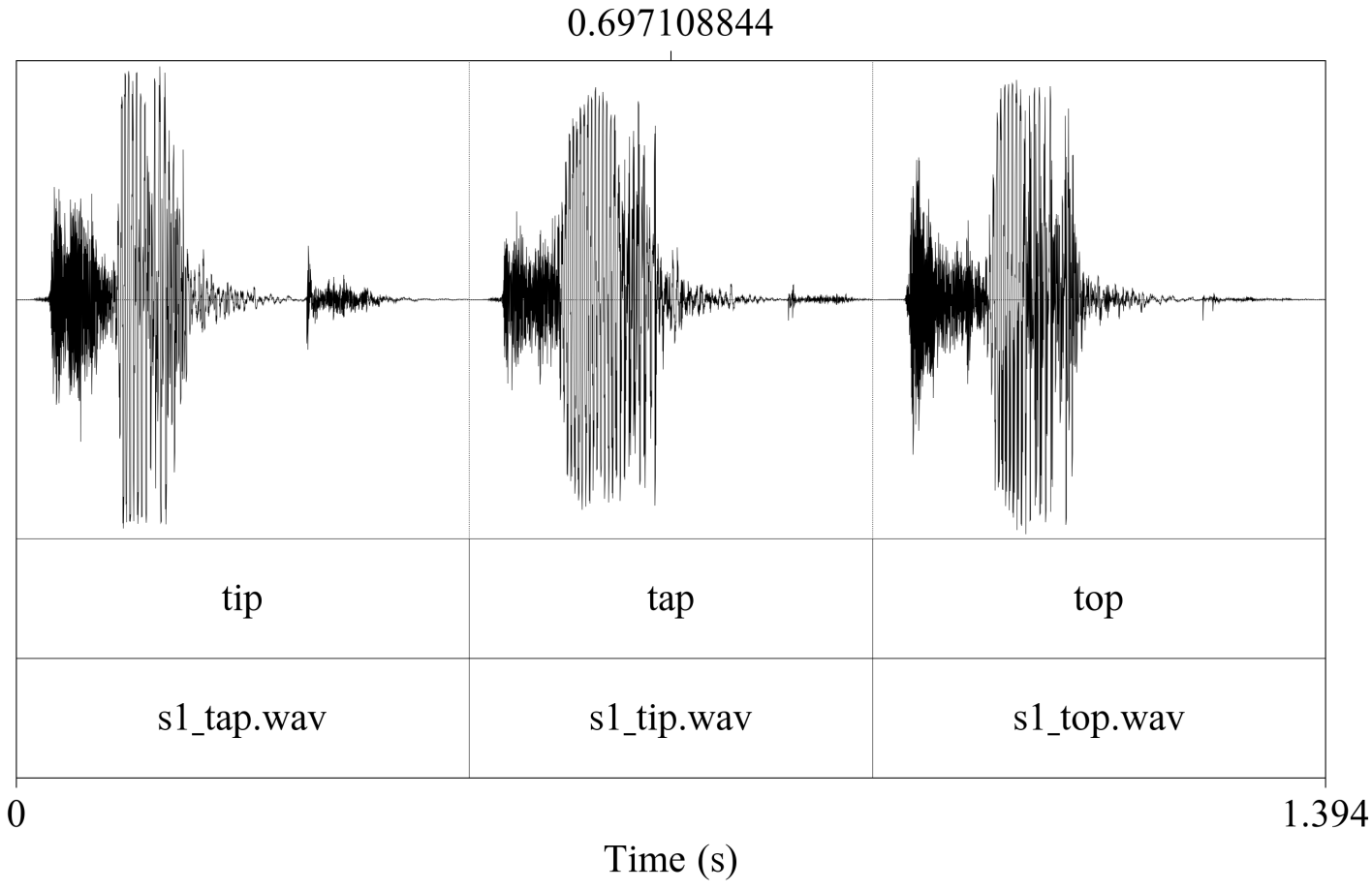
Merge all data together

Here is how `s1_all.TextGrid` and `s1_all.wav` look like:



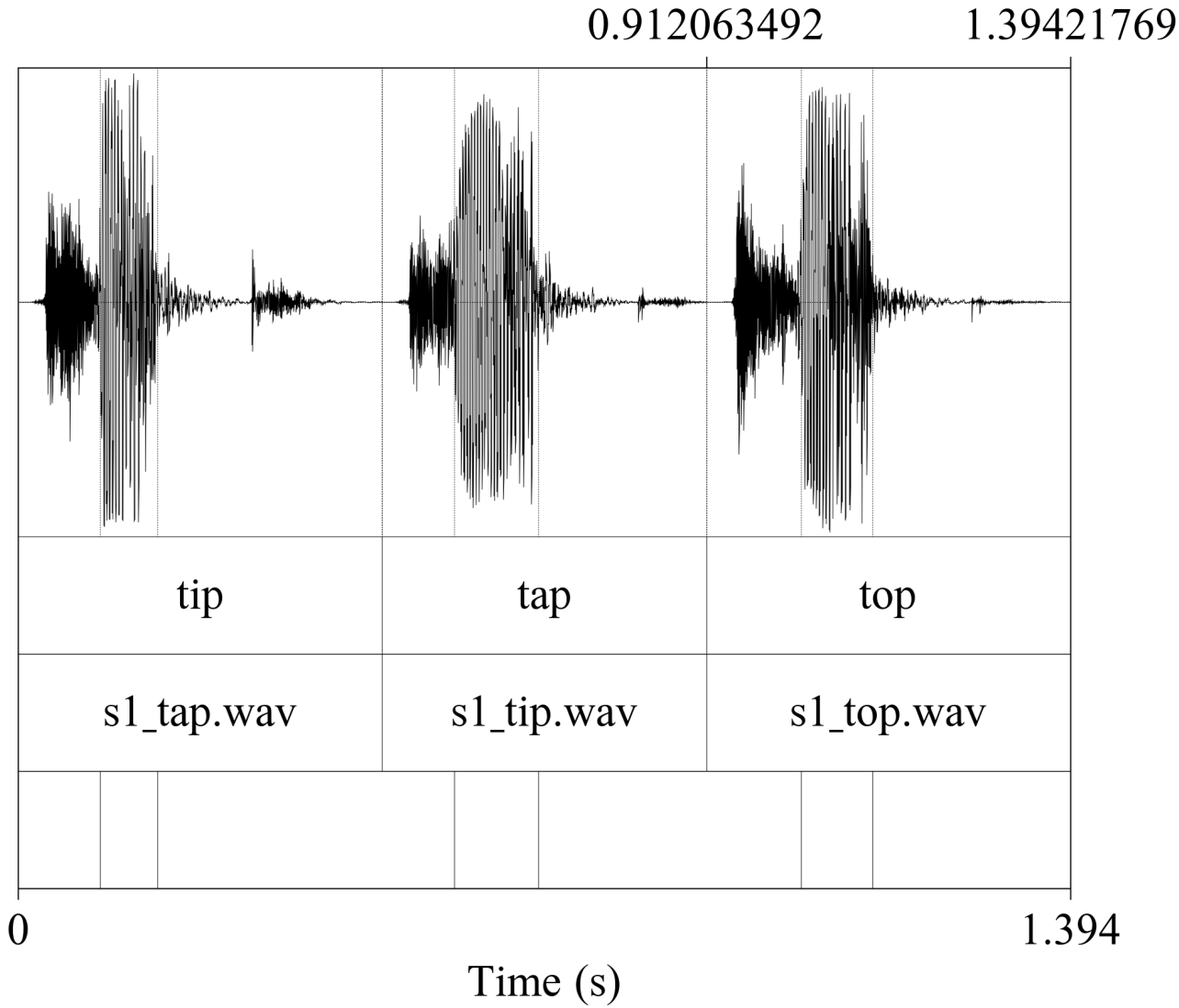
Automatically annotate your data

```
annotate_textgrid(annotation = my_stimuli$stimuli,
                  textgrid = "s1/s1_all.TextGrid")
```



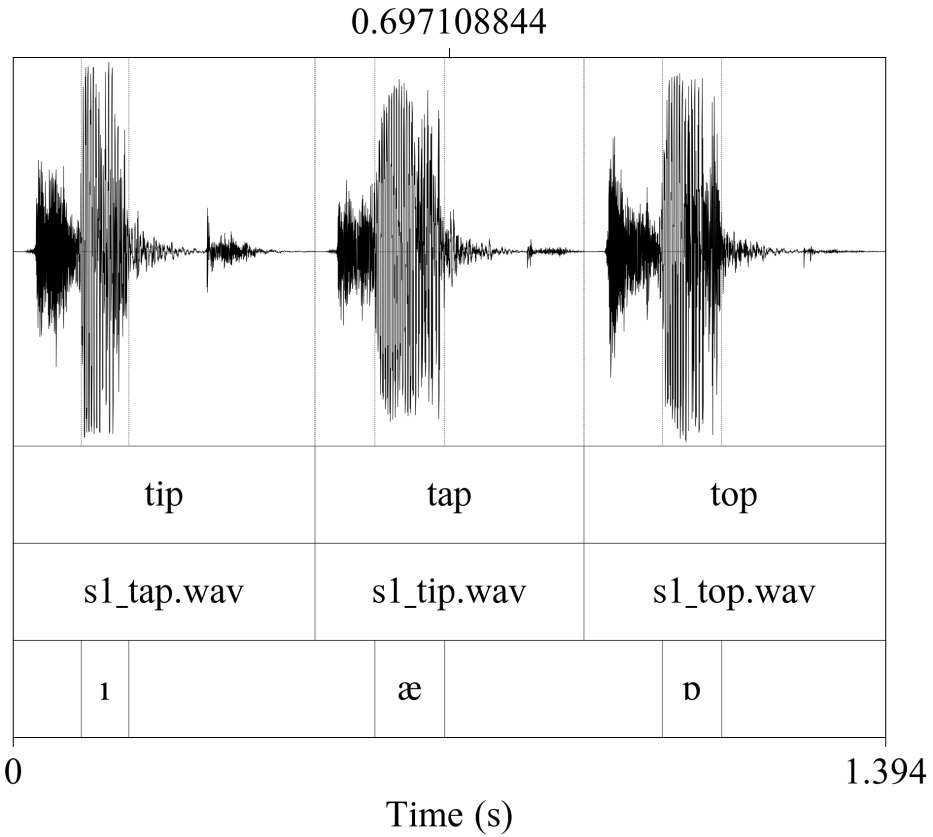
Automatically annotate your data

Imagine that someone manually annotated each vowel in the recording, so the `.TextGrid` will look as follows:



Automatically annotate your data

```
annotate_textgrid(annotation = my_stimuli$vowel,
                  textgrid = "s1/s1_all.TextGrid",
                  each = 2,
                  tier = 3,
                  backup = FALSE)
```



Automatically annotate your data

Why should anybody separately annotate borders and fill gathered annotations with text?
I've participated in several projects with human annotations: there are a lot of typos. So this approach allow

- to avoid typographical problems;

Automatically annotate your data

Why should anybody separately annotate borders and fill gathered annotations with text?
I've participated in several projects with human annotations: there are a lot of typos. So this approach allow

- to avoid typographical problems;
- even if you don't like it, it is possible to automatically annotate words, translations, utterances etc.

Extract annotated fragments

Let's create a folder where all of the extracted files will be stored:

```
dir.create("s1/s1_sounds")
## │   ├── s1_all.TextGrid
## │   ├── s1_all.wav
## │   ├── s1_sounds     ←     we created this folder!
## │   ├── s1_tap.wav
## │   ├── s1_tip.wav
## │   └── s1_top.wav
## └── s2
##     ├──,,,
```

Extract annotated fragments

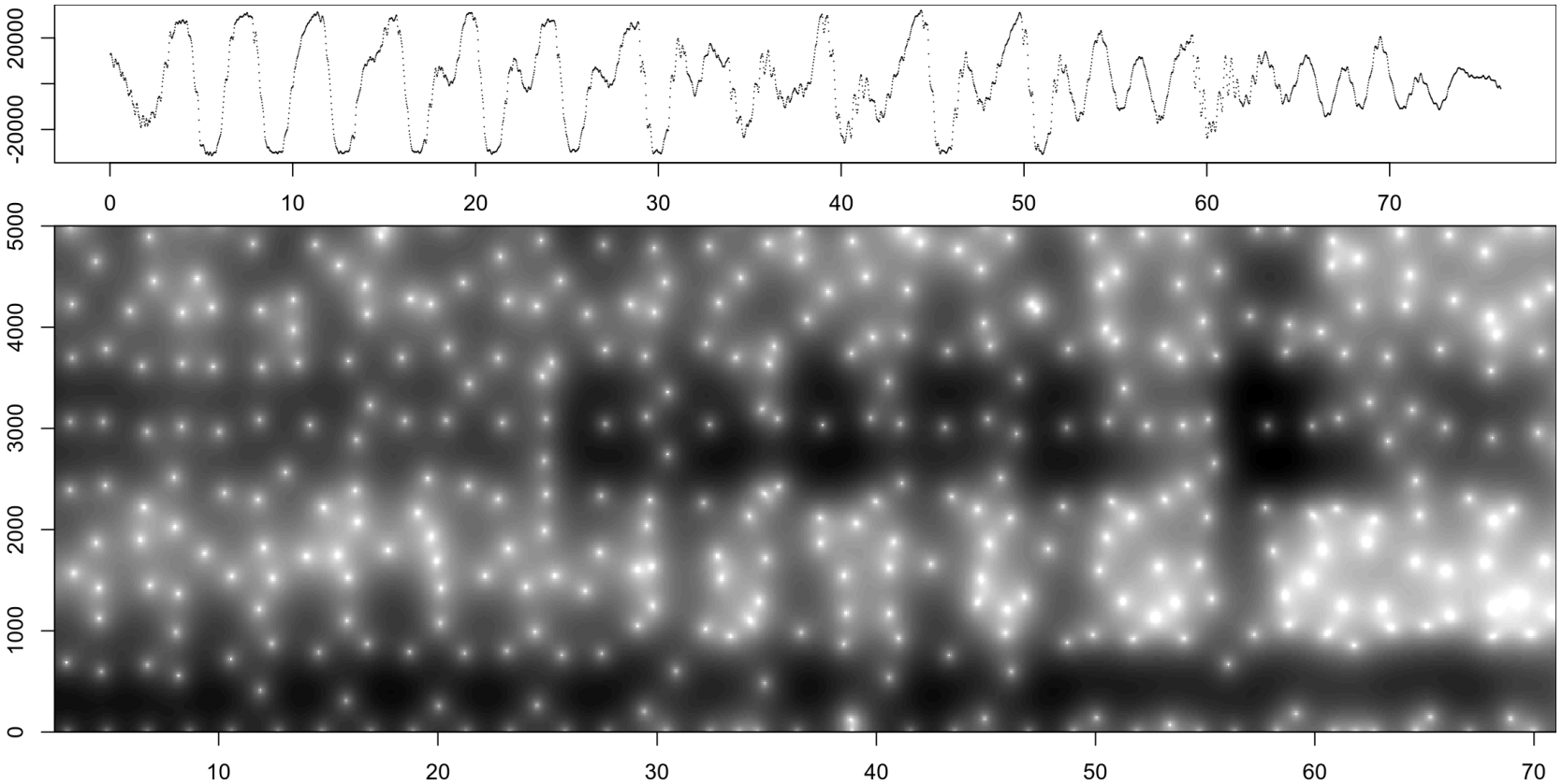
Let's create a folder where all of the extracted files will be stored:

```
extract_intervals(file_name = "s1/s1_all.wav",
                  textgrid = "s1/s1_all.TextGrid",
                  tier = 3,
                  path = "s1/s1_sounds/",
                  prefix = "s1_")
```

```
## |   ├── s1_all.TextGrid
## |   ├── s1_all.wav
## |   ├── s1_sounds
## |   |   ├── 1_s1_ɪ.wav
## |   |   ├── 2_s1_æ.wav
## |   |   └── 3_s1_ɒ.wav
## |   ├── s1_tap.wav
## |   ├── s1_tip.wav
## |   └── s1_top.wav
## └── s2
##     ├──,,,
```

Visualizing your data

```
draw_sound(file_name = "s1/s1_sounds/1_s1_1.wav")
```



Visualizing your data

```
draw_sound(file_name = "s1/s1_sounds/1_s1_1.wav")
```

- `title` — the title for the plot
- `colores` — set to (`TRUE`) for a colored spectogram, or (`FALSE`) for greyscale. It is also possible to provide a vector of custom colors for the spectrogram
- `maximum_frequency` — the maximum frequency to be displayed for the spectrogram
- `dynamic_range` — values greater than this many dB below the maximum will be displayed in the same color
- `window_length` — the desired length in milliseconds for the analysis window
- `output_file` — the name of the output file
- `output_width` — the width of the device
- `output_height` — the height of the device
- `output_units` — the units in which height and width are given. This can be `"px"` (pixels, default), `"in"` (inches), `"cm"` or `"mm"`

Visualizing your data

```
draw_sound(file_name = "s1/s1_sounds/1_s1_ɪ.wav", output_file = "s1/s1_tip", title = "s1
## ├── s1
## │   ├── s1_all.TextGrid
## │   ├── s1_all.wav
## │   ├── s1_sounds
## │   │   ├── 1_s1_ɪ.wav
## │   │   ├── 2_s1_æ.wav
## │   │   └── 3_s1_ɒ.wav
## │   ├── s1_tap.wav
## │   ├── s1_tip.png
## │   ├── s1_tip.wav
## │   └── s1_top.wav
## └── s2
## ...
```

Visualizing your data

```
draw_sound(sounds_from_folder = "s1/s1_sounds/", pic_folder_name = "s1_pics")
## ├── s1
## │   ├── s1_all.TextGrid
## │   ├── s1_all.wav
## │   ├── s1_pics
## │   │   ├── 1_s1_ɪ.png
## │   │   ├── 2_s1_æ.png
## │   │   └── 3_s1_ɒ.png
## │   ├── s1_sounds
## │   │   ├── 1_s1_ɪ.wav
## │   │   ├── 2_s1_æ.wav
## │   │   └── 3_s1_ɒ.wav
## │   ├── s1_tap.wav
## │   ├── s1_tip.png
## │   ├── s1_tip.wav
## │   └── s1_top.wav
## └── s2
```

Create an `.html` viewer

```
create_viewer(audio_dir = "s1/s1_sounds/",
              picture_dir = "s1/s1_pics/",
              textgrid = "s1/s1_all.TextGrid",
              tiers = c(1, 3),
              output_dir = "s1/",
              output_file = "stimuli_viewer")
```

- example 1
- example 2: Soqotri emphatics (Vasilisa Zhigulskaya's data)

Cite the package

```
citation("phonfieldwork")
##
## Moroz G (2019). _Phonetic fieldwork and experiments with
## phonfieldwork package_. <URL:
## https://CRAN.R-project.org/package=phonfieldwork>.
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {Phonetic fieldwork and experiments with phonfieldwork package},
##     author = {George Moroz},
##     year = {2019},
##     url = {https://CRAN.R-project.org/package=phonfieldwork},
##   }
```

Send me a letter!

agricolamz@gmail.com

Presentation is available here:

tinyurl.com/y3wtkcbq

References

Boersma, P. and D. Weenink (2019). Praat: doing phonetics by computer (version 5.3.51)[computer program] version 6.0.25, retrieved 15 november 2019 from http://www.praat.org/.

Bořil, T. and R. Skarnitzl (2016). Tools rpraat and mpraat. In P. Sojka, A. Horák, I. Kopeček, and K. Pala (Eds.), *Text, Speech, and Dialogue: 19th International Conference, TSD 2016, Brno, Czech Republic, September 12-16, 2016, Proceedings*, Cham, pp. 367–374. Springer International Publishing.

Brugman, H., A. Russel, and X. Nijmegen (2004). Annotating Multi-media/Multi-modal Resources with ELAN. In *LREC*.

Lubbers, M. and F. Torreira (2013). Pympi-ling: a Python module for processing ELANs EAF and Praats TextGrid annotation files.

Peirce, J. W., J. R. Gray, S. Simpson, M. R. MacAskill, R. Höchenberger, H. Sogo, E. Kastman, and J. Lindeløv (2009). Psychopy2: experiments in behavior made easy. *Behavior Research Methods 51*(1), 195–203.

Reidy, P. (2016). *textgRid: Praat TextGrid Objects in R*. R package version 1.0.1.

Schmidt, T. and K. Wörner (2009). Exmaralda–creating, analysing and sharing spoken language corpora for pragmatic research. *Pragmatics. Quarterly Publication of the International Pragmatics Association (IPrA) 19*(4), 565–582.