Phonetic fieldwork research and experiments with the R package *phonfieldwork*[*]

George Moroz
HSE University
agricolamz@gmail.com

1. Introduction

Olga Krivnova was a prominent phonetician. She was enormously influential in Russian linguistics and inspired, taught, and helped many phonologists and phoneticians in Russia. She contributed to research on Russian suprasegmental units like prosody and rhytmization, and their applications in automatic speech synthesis. This paper introduces the R package `phonfieldwork`, which is intended to perform better-quality phonetic research. I devote this paper to the memory of Olga Krivnova, whom I hold in high esteem.

From my point of view phonetic research should ideally consist of the following steps:

- Formulating a research question; thinking of what kind of data is necessary to answer the question, what is the appropriate amount of data, what kind of annotation is needed, what kind of statistical models and visualizations will be used, etc.
- Creating a list of stimuli.
- Eliciting the list of stimuli from speakers who signed an Informed Consent statement, thus agreeing to participate in the experiment and to be recorded on audio and/or video; monitoring recording settings: sampling rate, resolution (bit), and number of channels should be the same for all recordings.
- Annotating the collected data.
- Extracting the collected data.
- Creating visualizations and evaluating your statistical models.
- Reporting your results.
- Publishing your data.

[*] This article is the output of a research project implemented as part of the Basic Research Program at the National Research University Higher School of Economics (HSE University).

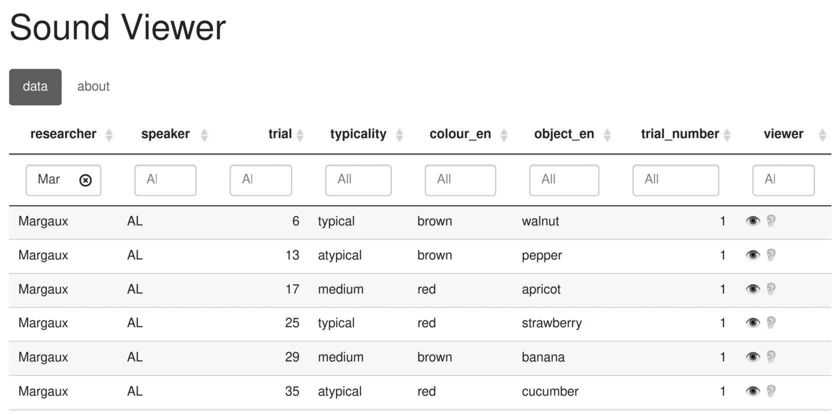
There are multiple ways of following this procedure. To make the automatic annotation of data easier, I usually record each stimulus as a separate file. While recording, I carefully listen to my consultants to make sure that they are producing the kind of linguistic units I want: three isolated pronunciations of the same stimulus separated by a pause and contained in a carrier phrase. In case a speaker does not produce three clear repetitions, I ask them to repeat the task, so that as a result of my fieldwork session I will have:

- a collection of small soundfiles (video) with the same sampling rate, resolution (bit), and number of channels;
- a list of successful and unsuccessful attempts to produce a stimulus according to my requirements (usually I keep this list in a regular notebook).

There are some phoneticians who prefer to record everything, for language documentation purposes. I think that this should be a separate task: you can't have your cake and eat it too. But if you or your research team insist on recording everything, you can 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 a special software to record your stimuli automatically on a computer (e.g. SpeechRecorder [Draxler 2011] or PsychoPy [Peirce 2007]).

You can show a native speaker your stimuli one by one or not show them the stimuli but ask them to pronounce a certain stimulus or its translation. I use presentations to collect all stimuli in a particular order without the risk of omissions.

Since each stimulus is recorded as a separate audiofile, it is possible to merge them into one file automatically and make an annotation in a Praat TextGrid (the same result can be achieved with the `Concatenate recoverably` command in Praat). After this step, the researcher needs to annotate the data according to their goals. When the annotation part is finished, the annotated parts can be extracted to Sound viewer. Sound viewer is a useful tool that combines your annotations and makes them searchable. It also produces a ready to go `.html` file that could be uploaded on the server (e.g., to Github Pages) and be available for anyone. In Sound viewer you can see a table, where each annotated object is a row characterized by some features (stimulus, repetition, speaker, etc.). You can play the soundfile and view its oscillogram and spectrogram (see Figure 1 for an example of a Sound viewer).


Sound Viewer

data about

researcher	speaker	trial	typicality	colour_en	object_en	trial_number	viewer
Mar	Al	Al	All	All	All	All	Al
Margaux	AL	6	typical	brown	walnut	1	👁 👂
Margaux	AL	13	atypical	brown	pepper	1	👁 👂
Margaux	AL	17	medium	red	apricot	1	👁 👂
Margaux	AL	25	typical	red	strawberry	1	👁 👂
Margaux	AL	29	medium	brown	banana	1	👁 👂
Margaux	AL	35	atypical	red	cucumber	1	👁 👂

Figure 1. Example of Sound viewer. By clicking on the eye icon, the user can see the spectrogram and oscillogram. By clicking on the ear icon, the user can hear extracted sound fragment.
The content, inventory, and names of columns are determined by the researcher

In the following section, I will show how to use the `phonfieldwork` R package to conduct phonetic fieldwork research.

2. Phonetic research steps with `phonfieldwork`

2.1. Reading a list of stimuli in R

There are several ways to read your stimuli list in R. First, one can just list all stimuli within the `c()` command:

```
my_stimuli <- c("tip", "tap", "top")
```

The most convenient way is to store your stimuli list in a `.csv` or `.xlsx` table and read it using standard tools for reading tables in R such as the `read.csv()` function, `read_csv()`/`read_tsv()` and others from the `readr` package, or the `read_xlsx()` function from the `readxl` package.

```
my_stimuli <- read.csv("your_file.csv")
library(readxl)
my_stimuli <- read_xlsx("your_file.xlsx")
```

If you upload your data from a `.csv` or `.xlsx` file, you will probably have some column names in your table. Throughout the article I will use the following table:

```
my_stimuli

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

In this table, we can see a list of three stimuli (*tip*, *tap*, *top*) and the target vowels (ɪ, æ, ɒ). In most cases, users more information is included, e.g., stimuli translations, syllable type, syllable number, or any other type of information. The content, inventory, and names of columns are determined by the researcher.

If you are not familiar with R, it is important to note that one can visualize the content of a certain column by using the dollar sign and the column name:

```
my_stimuli$stimuli

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

2.2. Creating a presentation based on a list of stimuli

When the list of stimuli is loaded into R, you can create a presentation for elicitation. It is important to define an output directory, so in the following example I use the `getwd()` function, which returns the path to the current working directory. You can set any directory as your current one by using the `setwd()` function (run `?setwd`, `?getwd` to see the documentation). It is also possible to provide a path to your intended output directory with the `output_dir` argument (e.g. `"/home/user_name/…"`). This command (unlike `setwd()`) does not change your working directory.

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

As a result, a file named `first_example.html` was created in the output folder, see Figure 2. You can change the name of this file by changing the `output_file` argument. It is also possible to change the output format by using the `output_format` argument. By default it is `html`, but you can also use `pptx`. The additional argument `translations` can be used to provide translation vector and make them appeared near the stimuli on the slide.

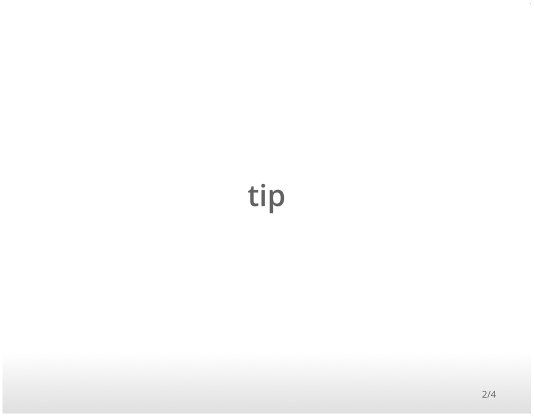


Figure 2. Example of an automatically compiled presentation based on a list of stimuli. You can use arrows for scrolling back and forth. It is possible to add translations, pictures, or animated `.gif` images.

2.3. Renaming the collected data

Let us imagine that we collected data from two speakers, `s1` and `s2`. After collecting data and removing soundfiles with unsuccessful elicitations, one could end up with the following file structure:

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

As shown in the scheme above, there are two separate folders, and in both of them files are named `01.wav`, `02.wav` and `03.wav`. Sound recorders often provide more meaningful names by adding a prefix with the date of the recording, but the rest is simple enumeration, since the sound recorder cannot retrieve information on what has been recorded and rename the files accordingly. So, this task is up to the researcher. It is important to keep in mind that numbering corresponds to the order in which the stimuli were presented to the speaker and not to alphabetic order. So, both for the analysis, for language documentation, and for simple file management, it makes sense to rename files by using more meaningful names. In our example, we have just three stimuli, but in projects with hundreds or even thousands of stimuli manual file renaming can be a highly time-consuming task.

For each speaker, `s1` and `s2`, there is a folder that contains three audiofiles. It is possible to rename soundfiles in phonfieldwork using the `rename_soundfiles()` command:

```
rename_soundfiles(stimuli = my_stimuli_df$stimuli,
                  prefix = "s1_",
                  path = "s1/")
## You can find change correspondences in the following file:
## .../s1/backup/logging.csv
```

As a result, we obtain the following file structure:

```
├── s1
│   ├── 1_s1_tip.wav
│   ├── 2_s1_tap.wav
│   ├── 3_s1_top.wav
│   └── backup
│       ├── 01.wav
│       ├── 02.wav
│       ├── 03.wav
│       └── logging.csv
└── s2
    ├── 01.wav
    ├── 02.wav
    └── 03.wav
```

The `rename_soundfiles()` function created a backup folder with non-renamed files, and renamed all files by using the prefix provided in the prefix argument (“s1_” in our example). The additional argument `backup` in the `rename_soundfiles()` function can be set to `FALSE` (it is `TRUE` by default) in case you are sure that the renaming function will work properly with your files and stimuli, and you do not need a backup of the unrenamed files. A message after the function run informs us that a `logging.csv` file was created as a result. This file contains the correspondences between old and new file names. Here are the contents of the `logging.csv` file:

```
    from           to
1 01.wav 1_s1_tip.wav
2 02.wav 2_s1_tap.wav
3 03.wav 3_s1_top.wav
```

The additional argument logging in the rename_soundfiles() function (TRUE by default) that creates a logging.csv file in the backup folder (or in the original folder if the backup argument has value FALSE).

Users might be surprised by the fact that the default behavior of the function is to create a backup folder and a logging.csv file. However, in case something goes wrong, users will need to check and rename all the files manually (and it could be hundreds or thousands of files). So, to prevent users from incidentally corrupting their files, I made this backup creation the default behavior. However, if a user has a large number of files (e.g., 10GB), this backup will harm, since it might be the case that there is not enough space on the user's hard drive. This is one the possible scenarios in which it makes sense to set the backup argument to FALSE.

2.4. Merging the data

Once all files have been renamed, you can merge them into one long file. Remember that sampling rate, resolution (bit), and number of channels should be the same across for recordings. It is possible to resample files with the resample() function from bioacoustics.

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

As we can see from Figure 3, this command creates a new soundfile s1_all.wav and an associated Praat TextGrid s1_all.TextGrid (the same result can be achieved with the Concatenate recoverably command in Praat). As a result, we obtain the following file structure:

```
├── s1
│   ├── 1_s1_tip.wav
│   ├── 2_s1_tap.wav
│   ├── 3_s1_top.wav
│   ├── backup
│   │   ├── 01.wav
│   │   ├── 02.wav
│   │   ├── 03.wav
│   │   └── logging.csv
│   ├── s1_all.TextGrid
│   └── s1_all.wav
└── s2
    ├── 01.wav
    ├── 02.wav
    └── 03.wav
```

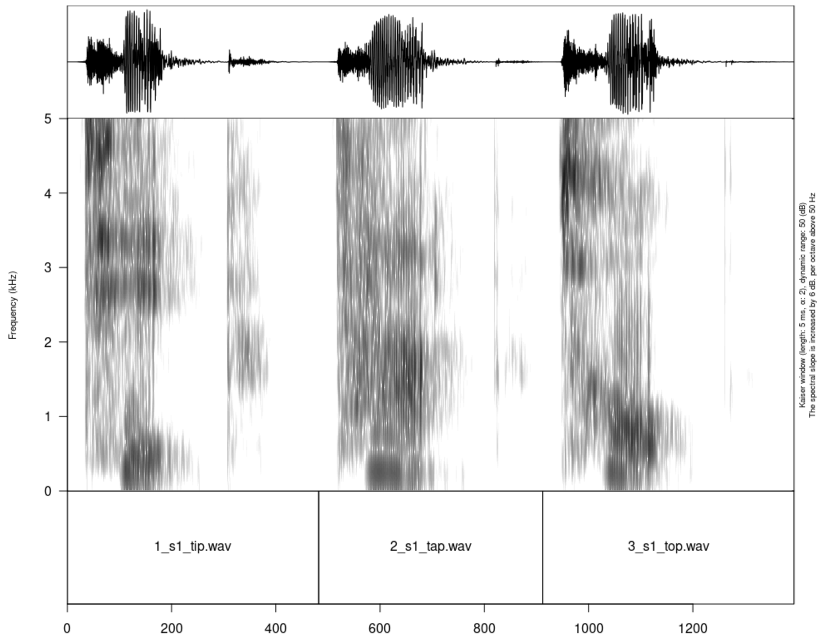


Figure 3. Result of the `concatenate_soundfiles()` function. All files are merged into one long file and an associated Praat TextGrid contains all file names.

Sometimes recorded sounds do not have fragments of silence at the beginning or at the end, so, after merging, the resulting utterances will be too close to each other. This problem can be fixed by using the argument `separate_duration` of the `concatenate_soundfiles()` function: just put the desired duration of the separator in seconds.

Although this is not the kind of task envisaged by the `phonfield-work` philosophy, multiple `.TextGrids` with the same tier structure can in principle be merged by using the `concatente_textgrids()` function.

2.5. Annotating the data

As shown in Figure 3, the annotation provided so far is just file names. We can improve it by using the existing annotation that we provided earlier:

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

The result can be seen in Figure 4 on the left side. As you can see from the example, the `annotate_textgrid()` function creates a backup of the tier and adds a new tier on top of the previous one. It is possible to prevent the function from doing so by setting the backup argument to `FALSE`. Imagine that we are interested in annotation of vowels. The most common solution is to open Praat and create new annotations manually. But it is also possible to create them in advance by using subannotations. The idea is that you choose some baseline tier that later will be automatically cut into smaller pieces in the other tier. In the following example we have one utterance per word, so we want to create three subannotations for each annotation in the word tier.

```
create_subannotation(textgrid = "s1/s1_all.TextGrid",
                     tier = 1, # this is a baseline tier
                     n_of_annotations = 3) # how many
empty annotations per unit?
```

After creating the subannotations, we can annotate the new tier. However, we need to pass a complex vector with the following values:

```
"" "ɪ" "" "" "æ" "" "" "ɒ" ""
```

In our example, we could have provided it just as a simple vector manually. However, if there are hundreds or thousands of stimuli, this might be an inconvenient solution. To create such a vector, we can use the following code:

```
vowel_annotation <- unlist(lapply(my_stimuli$vowel,
function(x) c("", x, "")))
vowel_annotation

[1] ""  "ɪ" ""  ""  "æ" ""  ""  "ɒ" ""
```

Note that this code is based on `my_stimuli$vowel`, so this works based on the data that we provided at the beginning. Now we are finally ready to perform automatic annotation:

```
annotate_textgrid(annotation = vowel_annotation,
                  textgrid = "s1/s1_all.TextGrid",
                  tier = 3,
                  backup = FALSE)
```

We created a third tier with annotation, which is shown in Figure 4 on the right side. The only passage left is moving annotation boundaries in Praat. This task can not be automated unless you use some forced alignment, e.g., MAUS (Munich Automatic Segmentation, [Kisler et al. 2012]). The output of the `annotate_textgrid()` and `create_sub-annotation()` functions is a `.TextGrid` in which the only task for the annotator is to move boundaries. First, this is faster for the annotator, since they do not need to type. Second, this strategy prevents the annotator from incidentally pasting something wrong in the annotation (e.g., spaces, new lines, or similar commands from other keyboard layouts). In case you want to create an empty `.TextGrid`, you can use the `create_empty_textgrid()` function, which takes duration as an argument. It is also possible to remove a tier from the `.TextGrid` by using the `remove_textgrid_tier()` function.

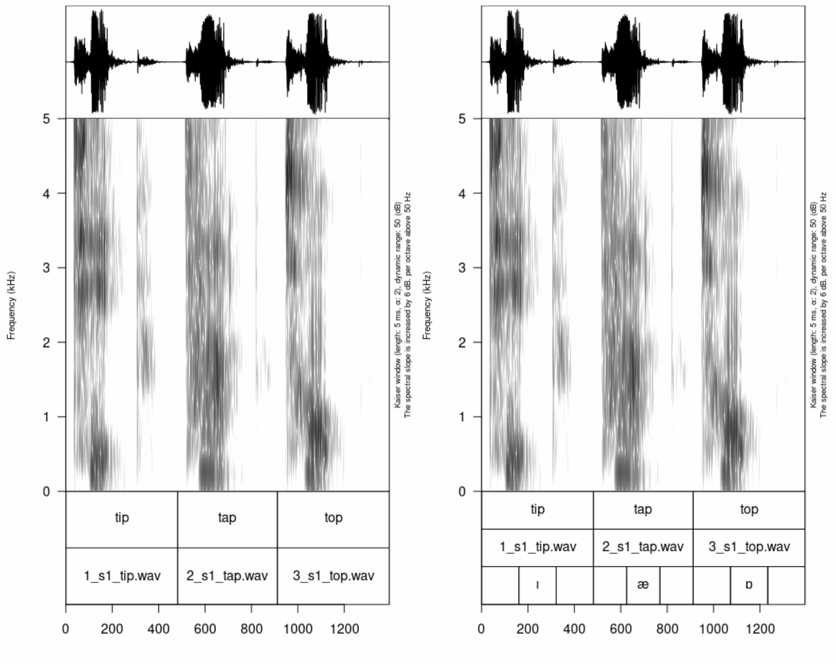


Figure 4. Results of the `annotate_textgrid()` function: adding stimuli (on the left), and vowel annotation (on the right)

2.6. Extracting the data

Once the annotation task is completed, it is possible to extract all annotations. In principle, at this stage, it is possible to extract all relevant information by using Praat scripts. However, this annotation extraction can be useful for creating an .html Sound viewer (see Section 1). First, it is important to create a folder where all the extracted files will be stored. In the following example, this folder is named s1_sounds:

```
dir.create("s1/s1_sounds")
```

Then, we can use the extract_intervals() function to extract of all annotated subfiles based on one of the annotation tiers:

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

As a result, we obtain the following file structure:

```
├── s1
│   ├── 1_s1_tip.wav
│   ├── 2_s1_tap.wav
│   ├── 3_s1_top.wav
│   ├── backup
│   │   ├── 01.wav
│   │   ├── 02.wav
│   │   ├── 03.wav
│   │   └── logging.csv
│   ├── s1_all.TextGrid
│   ├── s1_all.wav
│   └── s1_sounds
│       ├── 1_s1_ɪ.wav
│       ├── 2_s1_æ.wav
│       └── 3_s1_ɒ.wav
└── s2
    ├── 01.wav
    ├── 02.wav
    └── 03.wav
```

As you can see, in the s1_sounds folder there are three new files 1_s1_ɪ.wav, 2_s1_æ.wav and 3_s1_ɒ.wav, which have been extracted based on the annotation provided in the .TextGrid.

2.7. Visualizing the data

The phonfieldwork package allows viewing an oscillogram and spectrogram of any sound file:

```
draw_sound("s1/s1_all.wav",
           "s1/s1_all.TextGrid")
```

The result of this command can be seen on the Figure 4 (right side). It is not better than visualization from Praat, however, this can be useful for creating an `.html` Sound viewer that I introduced at the introduction section. It is also possible to create visualizations of all sound files in a folder. For this purpose you need to specify a source folder with the argument `sounds_from_folder` and a target folder for the images (`pic_folder_name`). The new image folder is automatically created in the upper level folder, so that sound and image folders are on the same level in the tree structure of your directory.

```
draw_sound(sounds_from_folder = "s1/s1_sounds/",
           pic_folder_name = "s1_pics")
```

As a result, we obtain the following file structure:

```
├── s1
│   ├── 1_s1_tip.wav
│   ├── 2_s1_tap.wav
│   ├── 3_s1_top.wav
│   ├── backup
│   │   ├── 01.wav
│   │   ├── 02.wav
│   │   ├── 03.wav
│   │   └── logging.csv
│   ├── 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_tip.png
└── s2
    ├── 01.wav
    ├── 02.wav
    └── 03.wav
```

As we can see, there are three .png files corresponding to each audio file in the s1_sounds folder. It is also possible to use the argument textgrid_from_folder in order to specify the folder containing .TextGrids for annotation (it could be the same folder as the sound one). By default, the draw_sound() function with the sounds_from_folder argument adds a title with the file name to each picture, but it is possible to turn it off by using the argument title_as_filename = FALSE.

2.8. Creating a Sound viewer

Sound viewer (see Figure 1) is a useful tool that combines your annotations and makes them searchable. It also produces a ready to go .html file that could be uploaded on the server (e.g., to Github Pages) and be available for anyone in the web.

To create a sound viewer, you need three things:

- a folder with sound files (see Section 2.6);
- a folder with pictures (see Section 2.7);
- a table including different types of information (e.g., annotation, utterance number, etc.; see Section 2.1).

Now we can create a Sound viewer:

```
create_viewer(audio_dir = "s1/s1_sounds/",
              picture_dir = "s1/s1_pics/",
              table = my_stimuli,
              output_dir = "s1/",
              output_file = "stimuli_viewer")
## Output created: s1/stimuli_viewer.html
```

As a result, a stimuli_viewer.html was created in the s1 folder. By default, sorting in the resulting annotation viewer will be based on file names in the system. If you want to have another type of sorting, you can specify the columns based on which the resulting table should be sorted by using the sorting_columns argument.

3. Reading linguistic files in R

The phonfieldwork package also provides several methods for reading different file types in R. This makes it possible to analyze them and convert them into .csv or .xlsx files (e.g. by using the write.csv() function or the write_xlsx() function from the writexl package). The main advantage of using those functions is that they return tables with the columns time_start, time_end, content and source, so it is easy to stock multiple files in one table. This makes it easier to manipulate existing annotation, e.g., for the purposes of online corpus creation, or in the process of data analysis. The functions to be used are the following:

- the `textgrid_to_df()` function for Praat `.TextGrid` files;
- the `eaf_to_df()` function for ELAN[1] `.eaf` files;
- the `exb_to_df()` function for EXMARaLDA[2] `.exb` files;
- the `srt_to_df()` function for `.srt` subtitles files;
- the `audacity_to_df()` function for Audacity `.txt` files.

As one can see, these are the most popular sound annotation standards in linguistics. For some of them (like Praat `.TextGrid` files, ELAN `.eaf` files, and EXMARaLDA `.exb` files), there is also a converter from the table format to the original file format. This make it possible, for example, to change something in the data when it is in the table format, and then convert it back to the original format.

4. Conclusions

In this article, I introduced the `phonfieldwork` package, which allows solving a number of common tasks in phonetic research. I described one of the possible ways of working with phonetic material by illustrating each of the steps that such a work requires, and showed how the `phonfiledwork` package can handle some of these tasks. In the last section, I presented a list of functions that allow for conversion between simple tables and different sound annotation formats like Praat `.TextGrid` files, ELAN `.eaf` files, EXMARaLDA `.exb` files, `.srt` subtitles files, and Audacity `.txt` files. The package is conceived as a dynamic project, so any problems or bugs can be fixed with the help of the user community.

References

Boersma P., Weenink D. Praat: Doing phonetics by computer [Computer program]. URL: http:// www.praat.org/, 2022.

Draxler Ch. Speech recorder quick start and user manual. Institute of Phonetics and Speech Processing, University of Munich, Tech. Rep. www.speechrecorder.org, 2011.

Kisler T., Schiel F., Sloetjes H. Signal processing via web services: The use case WebMAUS. Talk presented at Digital Humanities Conference 2012. Hamburg, Germany. 2012.

[1] [Sloetjes, Wittenburg 2008].

[2] [Schmidt, Wörner 2014].

Peirce J.W. PsychoPy — psychophysics software in Python. *Journal of neuroscience methods* 162, 2007. P. 8–13.

R Core Team. R: A Language and Environment for Statistical Computing. Vienna, Austria: R Foundation for Statistical Computing. 2022.

Schmidt Th., Wörner K. EXMARaLDA. Durand J., Gut U., Kristoffersen G. (eds.). *The Oxford handbook of corpus phonology*. Oxford: Oxford University Press, 2014. P. 402–419.

SIL: Speech analyzer [Computer program]. URL: https://software.sil.org/speech-analyzer/, 2011.

Sloetjes H., Wittenburg P. Annotation by category-ELAN and ISO DCR. *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco. European Language Resources Association (ELRA), 2008. P. 816–820.