

# Manual for the RANTO App

## Rhythm ANalysis TOol for Timeseries

Dr. Lara S. Burchardt

01/09/2025

## Contents

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>1. Introduction</b>                             | <b>1</b>  |
| 1.1      | 1.1 Why use this app? . . . . .                    | 2         |
| 1.1.1    | 1.1.1 A little background . . . . .                | 2         |
| 1.1.2    | 1.1.2 Literature . . . . .                         | 2         |
| 1.2      | 1.2 Preparations . . . . .                         | 3         |
| 1.2.1    | 1.2.1 Packages . . . . .                           | 3         |
| 1.2.2    | 1.2.2 Necessary Code on your machine . . . . .     | 4         |
| <b>2</b> | <b>2. How to use the app?</b>                      | <b>5</b>  |
| 2.1      | 2.1 How should the input data look like? . . . . . | 5         |
| 2.2      | 2.2 Starting the app . . . . .                     | 5         |
| 2.3      | 2.3 Input parameters, Options . . . . .            | 6         |
| <b>3</b> | <b>3 Analysis and Output</b>                       | <b>9</b>  |
| 3.1      | 3.1 Results Table . . . . .                        | 9         |
| 3.2      | 3.2 Visualization of Results . . . . .             | 11        |
| 3.3      | 3.3 Recurrence Plots . . . . .                     | 11        |
| <b>4</b> | <b>4. Download Buttons</b>                         | <b>11</b> |

## 1 1. Introduction

This is the manual for the shiny application “RANTO - Rhythm ANalysis TOol for Timeseries”. The following section will guide you through the first steps and will show you, how to work with the app.

The [code](#) for the app lies on my github repository.

Who am I? My name is Lara, I am a trained Biologist and did my PhD on rhythm in animals' acoustic signals and how to analyse them at the Museum für Naturkunde Berlin and the Freie Universität Berlin.

I am currently working at the Humboldt-Universität zu Berlin, where I have my own DFG-project to develop rhythm analysis methodology further. Currently I am working on different projects to analyse the temporal structures of communication signals, i.e. during development, across languages or time while further improving rhythm analysis methods and their accessibility.

## **1.1 1.1 Why use this app?**

### **1.1.1 1.1.1 A little background**

The temporal structure of animal communication is a potentially crucial parameter, that historically was often overlooked, especially in mammals and birds. With the development of new methods in recent years, research on rhythm analysis in animal communication is an ever-growing field, with yet many knowledge gaps: we want to identify the adaptive function of rhythms and add this piece of the puzzle to ongoing discussions about the evolution of language and music. Apart from that, the temporal structure of vocalizations can inform about species identity, communication context, arousal state, health conditions, and many more, which has wide implications for a lot of research fields ranging from behavioral ecology and cognitive science to conversation biology, and animal welfare. (Anichini et al., 2020; Honing & Ploeger, 2012; Honing et al., 2015; Manser, 2001; Ravignani et al., 2019).

Different parameters can be measured in the context of rhythm analysis, many of which I studied and developed during my doctoral thesis (Burchardt, Briefer, et al., 2021; Burchardt & Knörnschild, 2020). For example, we can calculate simple distributional parameters, such as mean and coefficient of variation of Inter-Onset-Intervals (IOI), the duration between the start of a sound element in a sequence and the start of the next element in that sequence (Figure 1). Furthermore, we can look at ratios between pairs of IOIs in a sequence, this approach has gained a lot of attention in the last years and is used extensively (Roeske et al., 2020; de Gregorio et al., 2021; Osiecka et al., 2024; Eleuteri et al. 2024; Van der Vleuten et al. 2024). A possible next step is to describe a single sequence with one particular rhythm or beat in hertz (Hz). A sequence with a rhythm of 5 Hz would have 5 equally distributed sound elements per second, so roughly one sound element every 200 ms. If it is a perfect fit, it is exactly every 200 ms, but in biological systems such 'perfection' seldom happens. Beat precision can then be used to describe how good the fit of the actual elements to the calculated theoretical beat is. I will analyze exact beat frequencies to describe a sequence with the so-called IOI approach (Burchardt & Knörnschild, 2020) and I will use the universal goodness-of-fit value, which I developed during my doctoral thesis, to determine the beat precision (Figure 1, called bp throughout). The striking advantages of the bp parameter include that it can be calculated independent of the method used to calculate the exact beats; it is independent of the number of elements, the rhythm it is describing, or any other parameter. Other goodness-of-fit values were correlated to i.e., the number of elements or the rhythm they were describing, making them unfit for comparative work or the general description of beat precision. Another important aspect of the bp parameter is that it can be calculated per sound element and can therefore be investigated on very different levels ranging from single sound elements to long sound sequences. It can be calculated for a single individual, or several individuals of the same species in comparison with another species, making it potentially useful for very different kinds of questions. Individual bp values can range from 0 to 1, with 0 meaning a perfect fit and 1 meaning the element is at the maximum possible deviation it can have from a beat. On a summary scale, a symmetry of values around 0.5 exists, where a summary value of 0.1 means the same as a summary value of 0.9, just in case of a 0.9 bp summary, elements would be phase shifted. Therefore, for the summary per sequence values are transformed accordingly to 0 to 0.5 (Burchardt et al., 2025).

The app calculates all these values and makes them available as results tables to download and plots.

### **1.1.2 1.1.2 Literature**

The methods were described and used by me for example in the following peer-reviewed publications:

Methods:

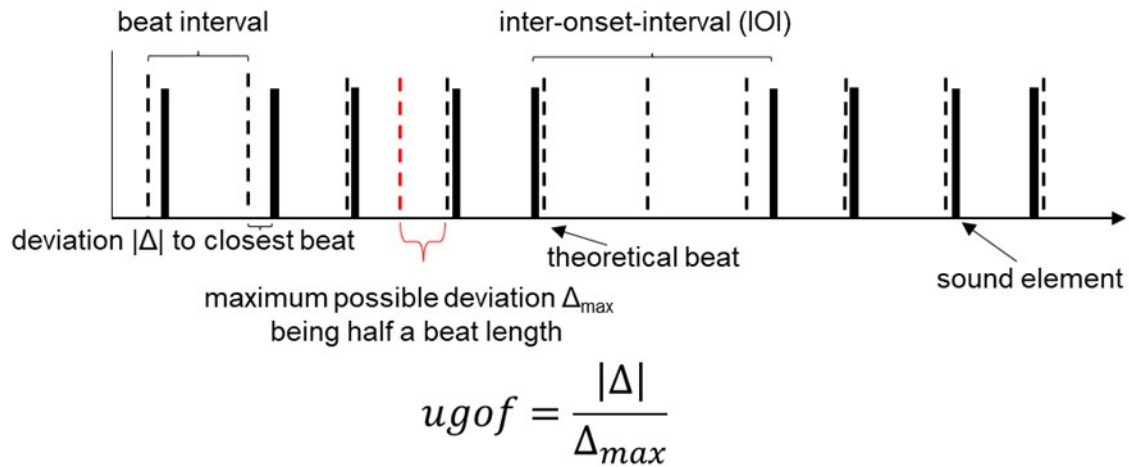


Figure 1: An element sequence with the best fitting theoretical beat overlaid and a depiction how the beat precision (ugof) value is calculated. If applicable, please cite [Burchardt et al, 2021](#)

[Paper 1](#): “Comparison of methods for rhythm analysis of complex animals’ acoustic signals” by Burchardt & Knörnschild, 2020

[Paper 2](#): “Novel ideas to further expand the applicability of rhythm analysis” by Burchardt, Briefer and Knörnschild, 2021

[Paper 3](#): “Robust rhythm reporting will advance ecological and evolutionary research” by Hersh, Ravignani and Burchardt, 2024

[Paper 4](#): Preprint of upcoming book chapter: “What is rhythm and how can we study it?” by Burchardt, Hoeschele, Honing, Large, Merker and Schneider, 2025

Different versions of the app were used in the following papers in action:

[Paper 5](#): “Rhythmic properties of *Sciaena umbra* calls across space and time in the Mediterranean Sea.” by Picciulin, Bolgan and Burchardt, 2024

[Paper 6](#): Preprint “Shared Duration of Speech Intervals Across 49 Human Languages” by Burchardt, Paschen and Fuchs, 2025

## 1.2 1.2 Preparations

The app runs independent of platforms on Windows, Linux and Mac. No adjustments are needed for any of these. A sessionInfo file with detailed information about R versioning and package versions can be found in the Git repository as well.

### 1.2.1 1.2.1 Packages

For the app to work a bunch of packages need to be installed. The app will automatically check, whether all necessary packages are installed on your local machine, and if not, will install them. This is an overview of the needed packages, just for your information.

```
install.load
shiny
shiny.Files
shinybusy
shinyjs
```

```
shinyWidgets
tidyverse
readxl
vegan
corrplot
plotly
DT
scales
strings
Rcpp
```

## 1.2.2 1.2.2 Necessary Code on your machine

To run the app it is easiest to do so in RStudio.

In the RStudio app, set up new project (file -> New project). If you have never used R projects before, see [here](#) for a short explanation of the use and advantageous of R Projects. If you are using a different shell just makes sure that all codes are in the same folder and this folder is set as working directory.

You now need to download the code from [github](#) into this project folder. This will ensure, that RStudio can access all necessary files and data without any further steps.

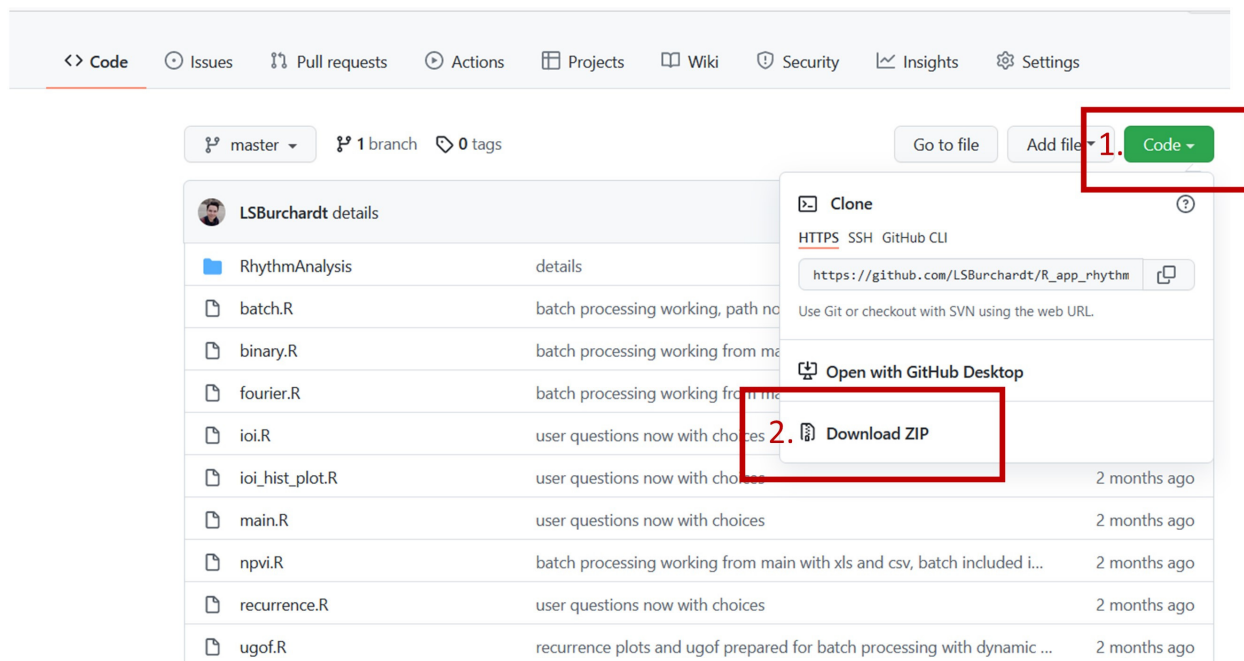


Figure 2: Screenshot of github page to guide code download.

On the project webpage, you click the green button “Code” and then choose “Download ZIP” (Figure 1) This will download the main app codes:

```
global.R
ui.R
server.R
```

and the folders R, containing helper functions for the app; www and images\_manual containing necessary files for the app and manual to work. You will also find the folder test\_data\csv with ten test csv files to analyze also showing the necessary structure of input files. More on that later.

Everything, that you save within the app (i.e., data) will also be saved into the project folder.

## 2 2. How to use the app?

### 2.1 2.1 How should the input data look like?

The input data needs to include the relevant event time points you want to calculate rhythms for. This could be starting points of sound elements, end points of sound elements, or time point with peak energy of sound elements, depending on your data and approach. These time points need to be in the first column of a data table input. The app only takes .csv files.

The data table needs a second column including respective “endpoints” of sound elements, if applicable. If you do not have endpoints, this column still needs to be present, it can contain pseudo numbers or NAs, as it is currently not used specifically. A third column should contain element types. If you supply element types, it is possible to filter for element types and calculate rhythms element type wise. Elements need to be labeled with small letters from a-z for this to work. You can distinguish up to 26 element types (a-z). For most datasets distinguishing between 10 element types is enough (a-j), per default only these elements are shown. If you need more element types, click the corresponding button “Show more elements”. If you do not know or do not care about different elements, just declare everything as “a” for example, then you do not have to adjust anything else.

A possible example of an input data table is shown below.

Table 1: Exemplary input data table with time points of interest in the first column (here start points), the respective end points and the element type.

| start | end  | element type |
|-------|------|--------------|
| 0.1   | 0.12 | a            |
| 0.2   | 0.23 | b            |
| 0.3   | 0.31 | b            |
| 0.4   | 0.42 | a            |
| 0.5   | 0.51 | d            |
| 0.6   | 0.62 | c            |
| 0.7   | 0.71 | a            |
| 0.8   | 0.82 | a            |
| 0.9   | 0.91 | a            |
| 1     | 1.01 | g            |

### 2.2 2.2 Starting the app

Before you start the app, your environment should be empty, no data sets should be loaded or values be stored. To make sure it is, run the following code `rm(list = ls)` in the console. WARNING: Only do this, when the respective project is opened! Otherwise you might loose data from other projects, you do not want to remove.

If you are using RStudio, now open one of the following scripts: `global.R`, `ui.R` or `server.R`. All three scripts together form the app (together with some helper files as described above). You can start the app from any of the three files.

To avoid confusion, it might be easiest to open the `global.R` file, as it is the shortest. Once you open it, on the top right corner you will see the option Run App with a green triangle in front of it. If you click on that option, the app will start.

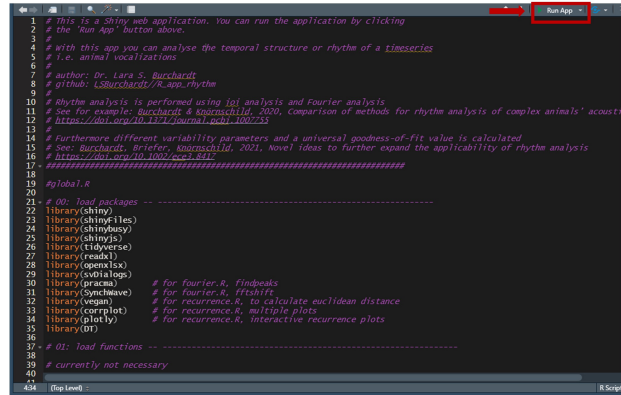


Figure 3: Where to find the “Run App button” when, i.e. the global.R script is opened.

If that is not working or you are not using RStudio, you can use this code to start the app: `shiny::runApp()`. For this to work, the app directory needs to be the working directory. If you are working with an R project, the project folder automatically is set as working directory. If you are not working with R projects, specify the working directory with `setwd('RhythmAnalysis')`.

## 2.3 Input parameters, Options

When you start the app, a new window will open, that looks like this:

Figure 4: Overview of the app after launch.

The app is separated in four tabs: “Input and Data” for setting input parameters and choosing the data to analyse; “Analysis Results” which shows the results in tables and plots; “Recurrence Plots” which shows recurrence plots of each individual file that was analysed (more on recurrence plots below.) and last, the “Help” tab, which links to this manual and shows some important remarks.

You can change several inputs in “Input and Data” tab.

### 1. Analysis Settings

1.a: Method for IOI Beat; here you can choose between median (default) and mean. This decides, whether the IOI Beat is calculated. Either, the median of IOIs over a sequence is taken to be transformed into a frequency, or the mean is taken.

1.b: Phase Shift Step Size (s); after an IOI beat has been calculated based on median or mean IOI duration in the sequence, we test which phase shift fits the whole sequence best. We shift more than one beat, the step size of which phase shifts to test is chosen here. It can be set in steps of 0.001 in a range from 0.001 to 10 seconds and depends on the sequence. The smaller the step size, the longer the calculations will run. Phase shift optimizations are saved and also plotted in “Analysis Results”.

1.c: Integer ratio method; Integer Ratios are calculated across the sequence for all pairs, not only adjacent pairs. As this can be very time consuming for longer sequences, you can choose the option “random pairs” instead of “All pairs”. If you choose “random pairs” two more options appear. You can then choose the number of random pairs to calculate. This will always calculate adjacent and non adjacent pairs in a ratio to 50% each, so in the default case of 20 pairs, 10 random adjacent pairs will be used and 10 non adjacent pairs. The raw integer ratios as well as which pairs were compared is saved. For reproducibility, you can set (and report) the seed used for the random choosing for interval pairs. The default seed is 123.

## 2. Input Data

2.a: Column names present?; Can be either TRUE or FALSE. Does your dataset have column headers such as “start”, “end”, or similar? If so, you have to choose TRUE, for the analysis to work correctly. If your first row already contains data, choose FALSE. You can check with the check button.

2.b: Select Folder; click this button to choose any folder on your machine for analysis. The folder you choose should only contain the .csv-files you want to analyse to avoid errors or crashes. In the left field you have to choose the folder as depicted, so that the contained csv files are shown in the right field. If you only see a folder on the right side, this won't work. See the picture for how this should look like:

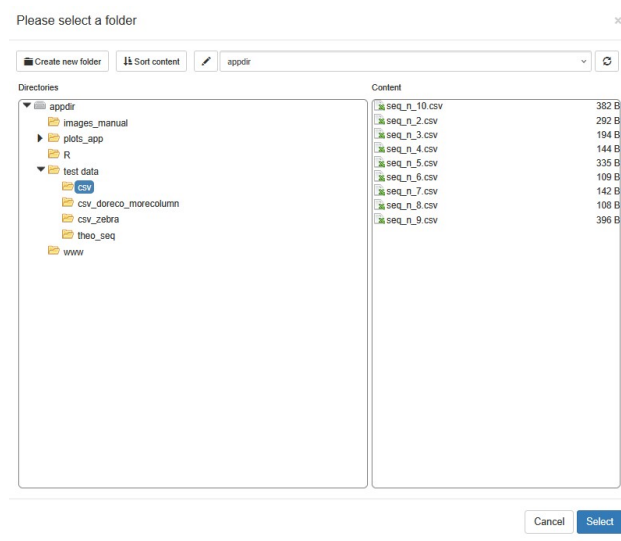


Figure 5: Folder selection: This is what the window will look like to choose the data to analyse. Like in the picture, the csv files need to be visible on the right side of the window.

2.c: Separator; depending on the settings on your machine, country and data extraction method, your csv files might be separated by commas or other delimiters. You can choose the separator individually, to accommodate this differences. The delimiters would most commonly be any of , | ; | \t.

2.d: With the “Check” button you can check, whether the chosen input choices work. This will read the first file in your list and display the first few lines. This should look something like this, if the read in works:

**A Input Data**

Column names present?

TRUE

Select folder

Separator

:

Check

```
# A tibble: 6 × 4
  T1    T2 Shape  Voc
<dbl> <dbl> <chr> <dbl>
1  0.11  0.21 b     10
2  0.45  0.49 b     10
3  1.03  1.13 e     10
4  1.36  1.48 b     10
5  1.75  1.93 b     10
6  2.79  2.96 h     10
```

ID for saving (e.g., test\_species)

default01

**B Input Data**

Column names present?

TRUE

Select folder

Separator

,

Check

```
# A tibble: 6 × 1
  `T1;T2;Shape;Voc`
<chr>
1 "0.11;0.21;\b\";10"
2 "0.45;0.49;\b\";10"
3 "1.03;1.13;\e\";10"
4 "1.36;1.48;\b\";10"
5 "1.75;1.93;\b\";10"
6 "2.79;2.96;\h\";10"
```

ID for saving (e.g., test\_species)

default01

Figure 6: Check Button: (A) If the read-in works fine, this is what the output of the check button would look like. (B) If the separator is chosen wrong, this is what the table might look like.

2.e: You can select a savename for your data, that will be included in the filename, when you download the results. A savename is necessary for the app to work properly, so a default savename is present. The savename should not be empty.

Potential issues: Column names - you need to set the column names argument to TRUE or FALSE. If you choose FALSE, but your data does indeed contain specific column headers, the read in will fail, depending on your machine, the app will either crash or just keep running, without any progress, in both cases this error message is likely to be shown in the console → “Error in FUN: nicht-numerisches Argument für binären Operator” (German) or “Error in FUN: non-numeric argument for binary operator”. The app runs into this error the first time it is trying to calculate values, which it then attempts to do on the column name.

### 3. Selected Files and Elements

3.a: Once you chose a folder containing csv files, a list of these files will be shown here with the filename and an index for each file. This is also the order in which the files will be analysed.

| file         | index |
|--------------|-------|
| seq_n_10.csv | 1     |
| seq_n_2.csv  | 2     |
| seq_n_3.csv  | 3     |
| seq_n_4.csv  | 4     |
| seq_n_5.csv  | 5     |
| seq_n_6.csv  | 6     |
| seq_n_7.csv  | 7     |
| seq_n_8.csv  | 8     |
| seq_n_9.csv  | 9     |

Figure 7: A list of csv files that were chosen for analysis. Note that how files are ordered. If no csv files exist in the chosen folder, the message “No CSV Files found” will be displayed.

3.b: Choose elements a-j; You have supplied element types which you can now analyse separatly if you want. For example, you might want to only analyse element types b and c, because they seem to have a different rhythm



than the other elements. If you need to differentiate between more than 10 elements, click the button, to be able to decide between elements from a to z (Figure 9). In the results table for the rhythm analysis parameters, the elements you chose will be saved.

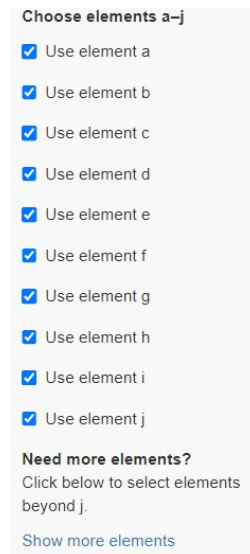


Figure 8: Per default all element types are selected, so that the full sequence is analysed. Select and de-select as needed.

Choosing input elements - if you filter element types in a way, that at least one sequence is of length 0, the app will crash corresponding error message → “Error in :: argument of length 0”

#### 4. GO Button

Once you have selected all input parameters, have chosen a folder to analyse and checked, whether the read in works, have decided on a savename and selected all elements you want to analyse, click the GO button to run the analysis. A spinning circle will appear in the upper right corner, indicating that the app is still calculating. Once it disappears, the calculations are done. You can now move to the “Analysis Results” or “Recurrence Plot” tab and download the results with the download buttons on top.

## 3 3 Analysis and Output

### 3.1 3.1 Results Table

The results table contains the following information and parameters (example in Figure 10):

1. Index
2. IOI Beat

That is the beat frequency in Hertz, that describes the element sequence best, when using the chosen averaging of IOI durations (median or mean) as the basis for the rhythm calculation. An IOI beat of 5 Hz for example means, if indeed our sequence in question can be described by an isochronous beat, we roughly have 5 beats per second.

3. Unbiased CV

This parameter indicates the variability of IOI durations independent of the sample size and the actual mean it is describing. The CV sets mean and standard deviation into relation and can be interpreted as a percentage. The higher it is, the higher is the variability. Here an unbiased equation is used, suitable for small sample sizes as well. The exact equations can be found [here in Equation 1 and 2](#):

#### 4. npvi

Another variability parameter, originating in linguistics. It indicates how well you can predict one IOI by the preceding IOI. A value of 0 would indicate, that every IOI in a sequence is exactly equal. The exact equations can be found [here in Equation 3](#):

#### 5. Beat Precision (ugof\_ioi)

That is the value, indicating, how well the IOI beat describes the sequence. Beat precision is calculated per element, the raw values can be downloaded as well. Here we report the mean beat precision across a sequence. In contrast to individual values, that can lie between 0 and 1, the summary values can lie between 0 and 0.5, as a symmetry around 0.5 exists. A mean of 0.1 and a mean of 0.9 would technically mean the same thing (just phase shifted), so mean values are transformed according to this symmetry (see [here for a more detailed explanation and equation](#): ). The lower it is, the better the fit between theoretical beat and actual sound elements.

#### 6. Silent Beats

Not every theoretical beat must be accompanied by a sound element, this value indicates how potentially “silent beats” we have. It is calculated as the number of theoretical beat - the number of elements. If elements are spaced very unevenly, it can happen that more than one element is “matched” to the same beat, which is not depicted, but needs to be considered and should not happen too often, otherwise IOI Beats are probably not capturing the event sequence well.

#### 7. npvi beat precision (npvi\_ugof\_ioi)

We calculate the npvi across the beat precision values to see the variability in precision.

#### 8. cv beat precision (cv\_ugof\_ioi)

We calculate the coefficient of variation across the beat precision values to see the variability in precision.

#### 9. averaging method

This just saves the averaging method you chose for calculating the IOI Beat, median or mean.

#### 10. elements used

This saves the elements that you chose to analyse.

#### 11. element sequence

The exact element sequence of the respective file.

#### 12. filename

The filename of the input file for reference.

#### 13. savename

The savename you chose for the analysis run.

| index | loi_beat | unbiased_cv | npvi  | ugof_loi | silent_beats_loi | npvi_ugof_loi | cv_ugof_loi | averaging | elements     | raw_element_seq       | filename     | savename  |
|-------|----------|-------------|-------|----------|------------------|---------------|-------------|-----------|--------------|-----------------------|--------------|-----------|
| 1     | 2.06     | 1.23        | 76.69 | 0.40     | 18               | 101.58        | 0.62        | median    | abcdefghijkl | bbebbhhhhbbacaacahh   | seq_n_10.csv | default01 |
| 2     | 2.70     | 0.22        | 20.82 | 0.32     | 2                | 89.62         | 0.78        | median    | abcdefghijkl | hghagbcbagbhabgh      | seq_n_2.csv  | default01 |
| 3     | 3.03     | 1.48        | 76.45 | 0.24     | 12               | 83.97         | 0.82        | median    | abcdefghijkl | hhhgchhhgc            | seq_n_3.csv  | default01 |
| 4     | 2.82     | 1.17        | 48.59 | 0.20     | 7                | 90.11         | 0.89        | median    | abcdefghijkl | hgagcci               | seq_n_4.csv  | default01 |
| 5     | 2.94     | 1.03        | 61.79 | 0.28     | 16               | 81.10         | 0.78        | median    | abcdefghijkl | hhbcbhhhhhhhhghhh     | seq_n_5.csv  | default01 |
| 6     | 2.86     | 0.11        | 20.51 | 0.26     | 2                | 76.64         | 0.78        | median    | abcdefghijkl | hhghg                 | seq_n_6.csv  | default01 |
| 7     | 3.17     | 0.06        | 8.15  | 0.23     | 1                | 81.50         | 0.83        | median    | abcdefghijkl | hgbgbg                | seq_n_7.csv  | default01 |
| 8     | 2.74     | 0.08        | 13.51 | 0.22     | 2                | 79.63         | 0.86        | median    | abcdefghijkl | hggbg                 | seq_n_8.csv  | default01 |
| 9     | 2.27     | 0.32        | 19.01 | 0.22     | 4                | 80.41         | 0.83        | median    | abcdefghijkl | fhagaagbhgcgcecaaefff | seq_n_9.csv  | default01 |

Figure 9: Example Results Table

## 3.2 3.2 Visualization of Results

Different rhythm values are plotted for a first overview. In the following order we see:

- A histogram of IOIs
- A density plot of integer ratios, all integer ratios are shown in black, then we separate lines again for only adjacent iois and non-adjacent iois
- (Results Table)
- A histogram of the raw deviations in seconds of elements from the closest theoretical beat
- A Scatter plot of the found IOI Beats
- A Scatter plot of the Beat Precision Values and Coefficients of variation
- A Scatter plot of npvi values
- A line plot showing the phase shift optimization. We search for the phase shift, with the lowest Root-Mean-Square-Deviation between elements and theoretical beats. Raw deviations and beat precision are calculated and saved after phase shift optimization.

## 3.3 3.3 Recurrence Plots

Recurrence Plots are a way of visualizing the temporal structure of sequences, this is especially interesting for longer sequences. The plot is comprised of squares, each square depicts the comparison of one interval in the sequence with another interval in the sequence. The color of the square indicates the difference between the two intervals. The darker a square is, the larger the difference, a white square can be seen as similar intervals (a threshold is applied here, to allow small differences and still plot a white square). To quantify the difference, euclidean distances are calculated between every pair of intervals.

To calculate euclidean distances, the `vegdist` function from the `vegan` package is used. If the euclidean distance is below a data driven threshold, the euclidean distance is set to 0. The threshold is calculated as the mean of IOIs \* 0.1. This is based on a similar approach in [Ravignani & Norton, 2017](#).

In case you want to adjust this threshold, it is set in the server.R file in section 5: Recurrence Plots, line 661-664 (code version 05.09.2025).

## 4 4. Download Buttons

There are four download buttons available (Figure 10). Once the analysis were run, four different datasets with results can be downloaded as a .csv file.



Figure 10: Download Buttons

Rhythm Analysis Results: This contains the results table also shown in the app.

Combined IOIs: This gives a table of all IOIs that were analysed, with file information and the respective IOI beat.

Integer Ratios: This gives a table of all calculated integer ratios. The index of the intervals compared ( $i,j$ ) are given, as well as a boolean column stating, whether these intervals are adjacent (TRUE) or not (FALSE) and the filename.

Raw deviations: This table shows a bunch of parameters connected to how well the element sequence fits the theoretical sequence. We have the filename and the index of the element per sequence. Then we have the time point of the element (`obs_value`) and the timepoint of the matched theoretical beat (`matched_theo`), then we have the raw deviation in seconds between these two points (`raw_deviation`) as well as the absolute deviation (`abs_deviation`). Given in this table is also the best phase shift, that was applied to the sequence before calculating these parameters (`best_shift`) and the beat precision value of the element (`ugof_value_beat`) which is summarized per sequence to form the “`ugof_ioi`” value in the Analysis Results table.