

---

# **EAdetection Documentation**

***Release 1.0***

**Katharina Heining**

**Jan 27, 2021**



# CONTENTS

<b>1</b>	<b>Getting started</b>	<b>3</b>
1.1	Download the toolbox . . . . .	3
1.2	Set up a virtual environment (recommended) . . . . .	4
1.3	Install requirements . . . . .	4
<b>2</b>	<b>Setting parameters</b>	<b>5</b>
2.1	How parameters are handled . . . . .	5
2.2	Create a template . . . . .	6
2.3	Create a parameter file for a recording . . . . .	8
<b>3</b>	<b>Preprocessing</b>	<b>11</b>
3.1	Load and resample . . . . .	11
3.2	Assign polarity . . . . .	12
3.3	Detect artifacts . . . . .	14
<b>4</b>	<b>EA detection and classification</b>	<b>17</b>
4.1	Spike detection . . . . .	18
4.2	Burst detection and classification . . . . .	26
<b>5</b>	<b>Output</b>	<b>29</b>
5.1	Results (HDF5) . . . . .	29
5.2	Quality control figures . . . . .	30
<b>6</b>	<b>Accessing results</b>	<b>39</b>
6.1	Python . . . . .	39
6.2	Matlab . . . . .	44
<b>7</b>	<b>Accessing metadata and diagnostics metrics</b>	<b>45</b>
7.1	General . . . . .	45
7.2	odML . . . . .	46
7.3	YAML . . . . .	50
<b>8</b>	<b>Todolist</b>	<b>53</b>

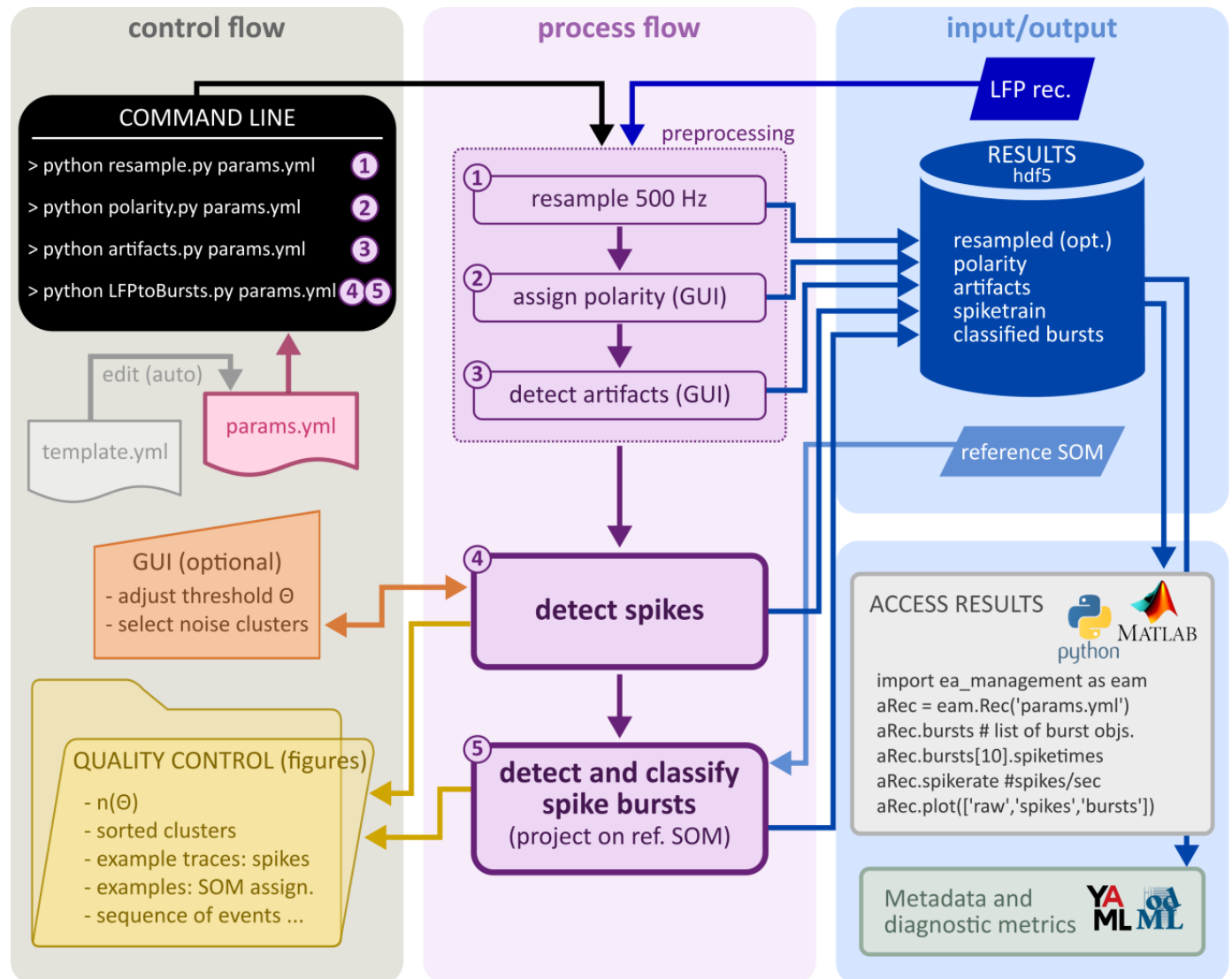


EAdetection is a tool that allows you to automatically **detect and classify epileptiform activity (EA) patterns**. It transforms a local field potential (LFP) recording in a sequence of classified spike bursts and solitary spikes. The tool is python-based and you can execute it via the command-line.

The EA patterns EAdetection identifies are:

- *high-load bursts*, these comprise large behavioral seizures and paroxysmal and seizure-like activity (e.g. high-voltage sharp waves and hyperparoxysmal discharges)
- *medium-load bursts*, a category bridging the continuum between high-load and low-load bursts
- *low-load bursts*, which are looser trains of epileptiform spikes

Three **major analysis steps** are involved: I) preprocessing (resampling, assigning the polarity, semiautomatic artifact detection), II) transforming the LFP into a train of epileptiform spikes, and III) transforming the spike train into a sequence of classified spike bursts.



The tool was originally developed for the **intrahippocampal kainate mouse model**, but we have seen promising results for other animal models of epilepsy as well. It has been used in several publications, but you will find the method best described in *Heining et al. (2021)*.

In this documentation you will learn how to *set everything up*, *set the parameters*, run the *preprocessing* and the *actual*

*analyses*, *harvest the results*, and *retrieve metadata and diagnostic metrics*. You can download the example data and the files successively generated in this tutorial from *here*\_. If can also download a *pdf-version*\_ of this documentation.

---

### Todo:

- 1) Make a link to Heining et al. 2021 once the paper is out.
  - 2) Make a link to the tutorial data file.
  - 3) Make a link to the pdf.
-

## GETTING STARTED

### 1.1 Download the toolbox

#### 1.1.1 Using git

Navigate to where you want to store the toolbox. For example:

```
> cd EAdetection_tutorial/
```

Then clone the repository from git using the command-line:

```
> git clone https://github.com/biomicrotechnology/EAdetection.git
```

The output on the command-line should look like this:

```
remote: Enumerating objects: 362, done.
remote: Counting objects: 100% (362/362), done.
remote: Compressing objects: 100% (151/151), done.
remote: Total 362 (delta 216), reused 340 (delta 201), pack-reused 0
Receiving objects: 100% (362/362), 298.16 KiB | 0 bytes/s, done.
Resolving deltas: 100% (216/216), done.
Checking connectivity... done.
```

Now you have a sub-folder called *EAdetection* in the directory where you executed git clone. *EAdetection* contains all the code.

#### Updating to a newer version

You can later update the toolbox to its newest version using git. Navigate to the toolbox directory:

```
> cd EAdetection
```

And then update to the newest version of the code:

```
> git pull
```

But for now, let's stay one level above the folder *EAdetection*.

#### 1.1.2 Alternative: download archive

If you do not want to use git, you can also download the [zip archive](#) from github. Extract this to a place where you want to store the toolbox.

## 1.2 Set up a virtual environment (recommended)

I strongly recommend installing all requirements for *EAdetection* in a new virtual environment and always run the program from there. This keeps all dependencies nice and tidy and prevents that stuff does not work anymore due to an annoying update. Here is how to set up a virtual environment. To use virtual environments, use this command on Linux:

```
> pip install virtualenvwrapper
```

Or this on for a Windows machine:

```
> pip install virtualenvwrapper-win
```

Then create a virtual environment and let's name it *EAdetectionEnv*:

```
> mkvirtualenv --python=python3 EAdetectionEnv
```

To enter this environment, type:

```
> workon EAdetectionEnv
```

You can leave this environment by typing:

```
> deactivate
```

But for now, let's stay in *EAdetectionEnv*.

## 1.3 Install requirements

If you do not have python 3 (code also works with python version  $\geq 2.5$ ) and pip 3 on your computer. You can find instructions about how to install these two at [python](#) and [pip](#).

If you decided to use a virtual environment for EA detection, make sure you are in it:

```
> workon EAdetectionEnv
```

Install the requirements:

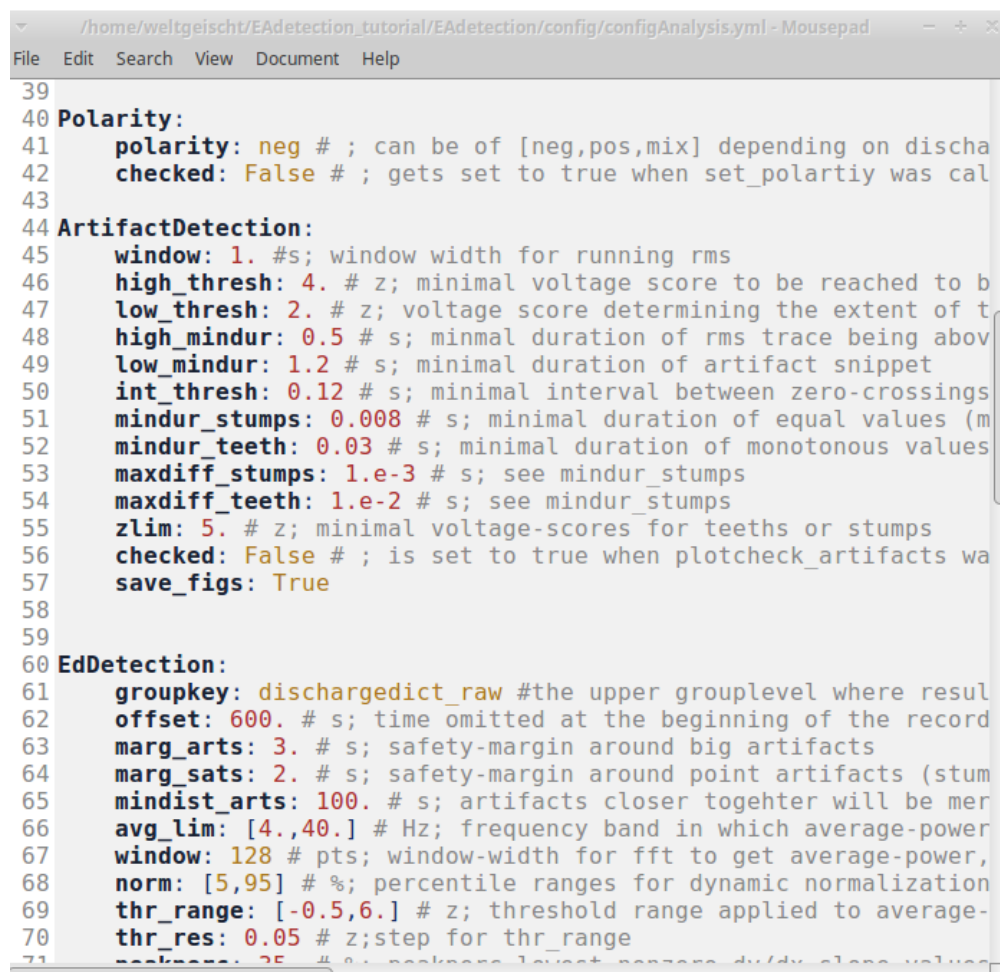
```
> pip3 install -r EAdetection/requirements.txt
```



## SETTING PARAMETERS

### 2.1 How parameters are handled

Parameters are handled at two different levels. The **default parameters** are located at *EAdetection/config/configAnalysis.yml* and look like this:



```

39
40 Polarity:
41     polarity: neg # ; can be of [neg,pos,mix] depending on discha
42     checked: False # ; gets set to true when set_polartiy was cal
43
44 ArtifactDetection:
45     window: 1. #s; window width for running rms
46     high_thresh: 4. # z; minimal voltage score to be reached to b
47     low_thresh: 2. # z; voltage score determining the extent of t
48     high_mindur: 0.5 # s; minmal duration of rms trace being abov
49     low_mindur: 1.2 # s; minimal duration of artifact snippet
50     int_thresh: 0.12 # s; minimal interval between zero-crossings
51     mindur_stumps: 0.008 # s; minimal duration of equal values (m
52     mindur_teeth: 0.03 # s; minimal duration of monotonous values
53     maxdiff_stumps: 1.e-3 # s; see mindur_stumps
54     maxdiff_teeth: 1.e-2 # s; see mindur_stumps
55     zlim: 5. # z; minimal voltage-scores for teeths or stumps
56     checked: False # ; is set to true when plotcheck_artifacts wa
57     save_figs: True
58
59
60 EdDetection:
61     groupkey: dischargedict_raw #the upper grouplevel where resul
62     offset: 600. # s; time omitted at the beginning of the record
63     marg_arts: 3. # s; safety-margin around big artifacts
64     marg_sats: 2. # s; safety-margin around point artifacts (stum
65     mindist_arts: 100. # s; artifacts closer togehter will be mer
66     avg_lim: [4.,40.] # Hz; frequency band in which average-power
67     window: 128 # pts; window-width for fft to get average-power,
68     norm: [5,95] # %; percentile ranges for dynamic normalization
69     thr_range: [-0.5,6.] # z; threshold range applied to average-
70     thr_res: 0.05 # z;step for thr_range
71     peakres: 25 # s; peakres: lowest nonzero du/dy slope values

```

Behind each parameter and default value there is an explanation of what it is good for. Make a change to this file only if you are absolutely sure that you want this change for all for all of your analyses. For now, leave it untouched. In the next sections we will create experiment and recording specific parameter files where you can overwrite the defaults.

## 2.2 Create a template

We found it very useful and time-saving to create parameter templates for whole experiments, i.e. a larger number of recordings to which the same parameters should be applied. To work with templates, first create a directory where you would like to store your templates:

```
> mkdir my_templates
```

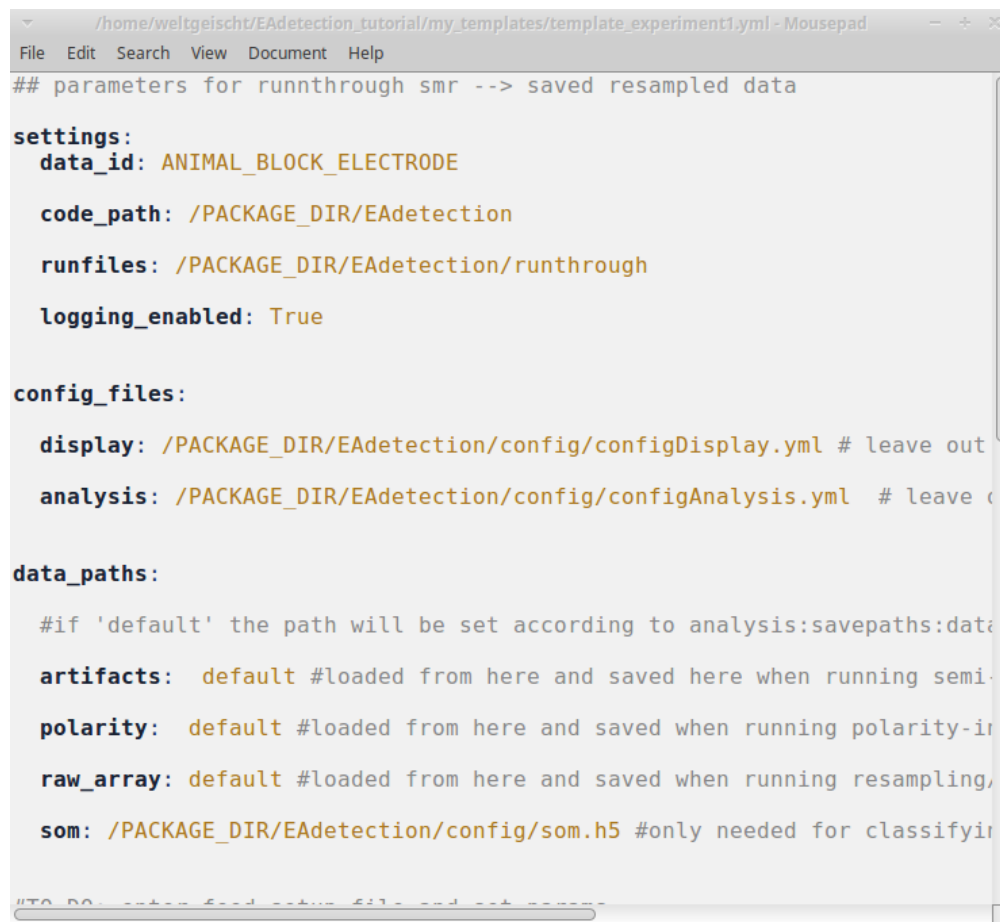
Then copy a templates from EAdetection/templates to your directory. On Linux:

```
> cp EAdetection/templates/template__runparams.yml my_templates/template_experiment1.  
↪.yml
```

On Windows:

```
> cp EAdetection/templates/template_runparamsWin.yml my_templates/template_  
↪experiment1.yml
```

Open the copied template (in this case *my\_templates/template\_experiment1.yml*), it looks like this:



```

/home/weltgeist/EAdetection_tutorial/my_templates/template_experiment1.yml - Mousepad
File Edit Search View Document Help
## parameters for runthrough smr --> saved resampled data

settings:
  data_id: ANIMAL_BLOCK_ELECTRODE

  code_path: /PACKAGE_DIR/EAdetection

  runfiles: /PACKAGE_DIR/EAdetection/runthrough

  logging_enabled: True

config_files:

  display: /PACKAGE_DIR/EAdetection/config/configDisplay.yml # leave out
  analysis: /PACKAGE_DIR/EAdetection/config/configAnalysis.yml # leave c

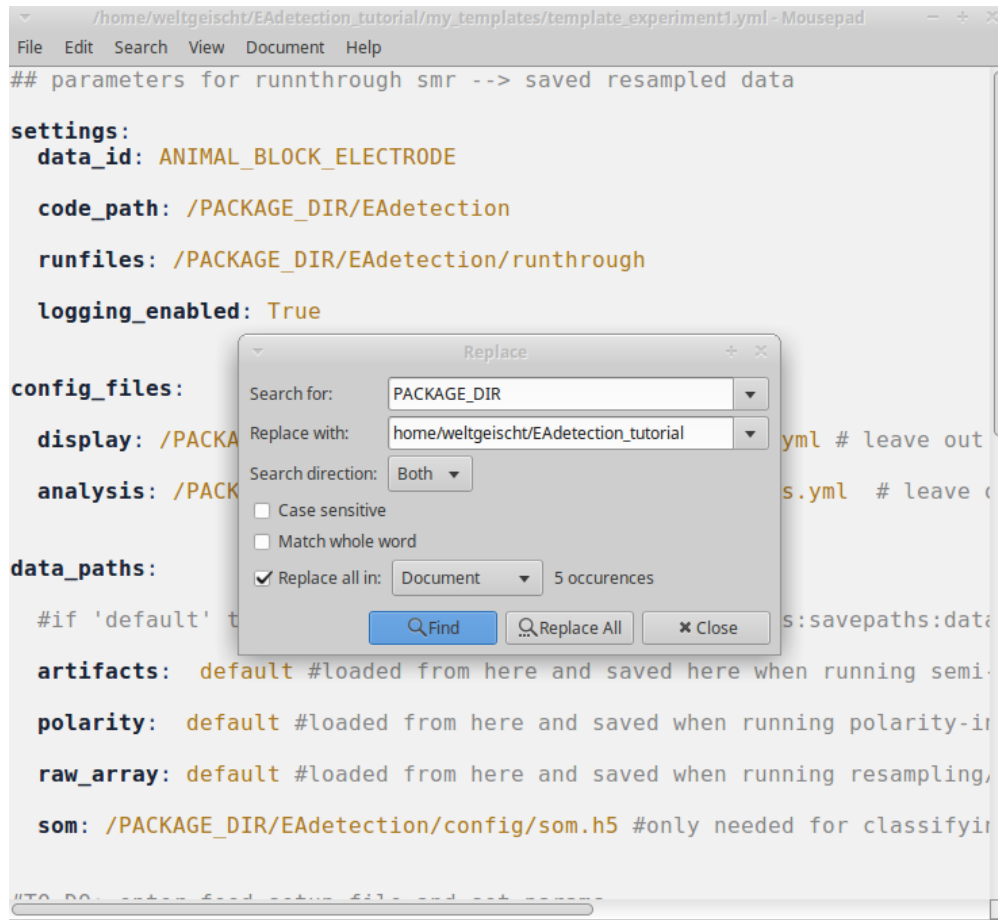
data_paths:

  #if 'default' the path will be set according to analysis:savepaths:date

  artifacts: default #loaded from here and saved here when running semi-
  polarity: default #loaded from here and saved when running polarity-in
  raw_array: default #loaded from here and saved when running resampling,
  som: /PACKAGE_DIR/EAdetection/config/som.h5 #only needed for classifyin

#TO DO: enter feed-back file and set param-
```

All entries in CAPITALS are place-holders, which we will now successively replace according to our setup and preferences:



Make the following replacements:

- `PACKAGE_DIR`: where you cloned/downloaded the toolbox, e.g.: `home/weltgeischt/EAdetection_tutorial`
- `DATADIR` → where you want your results saved, e.g.: `home/weltgeischt/EAdetection_tutorial/my_results/data`
- `FIGDIR`: where you want the figures saved, e.g.: `home/weltgeischt/EAdetection_tutorial/my_results/figures`
- `SOURCEDIR`: directory in which you keep your raw data files, e.g.: `home/weltgeischt/EAdetection_tutorial/example_data`

### 2.2.1 Overwriting defaults

You can also overwrite default parameters at the template level. You do this by specifying the parameter as a sub-entry below *setparams* (red arrows in example below). Here we set the offset for spike detection (*EdDetection*, *Ed* for epileptiform discharge) to 300 s, i.e. we exclude the first 5 min of the recording from spike detection:

```

run: [EdDetection, SpikeSorting, BurstClassification, Diagnostics]

save_figs: True

EdDetection:
  interactive: True
  setparams:
    offset: 300

SpikeSorting:
  interactive: True
  setparams:

BurstClassification:
  setparams:

StateAnalysis:
  setparams:

Diagnostics:
  setparams:

preprocessing:
  loading:
    source: /home/weltgeischt/EAdetection_tutorial/example_data/THIS_SOURCE_FILE
    channel: interactive # either the channel ID if known or interactive

```

At this stage we have created a template for a big batch of recordings.

## 2.3 Create a parameter file for a recording

Now we will create a parameter file specific for a single recording session. This will be the input for the commandline tool. Here you will learn how to generate a specific parameter file from the template and how to edit it. But first, let's create a directory where you can store all your specific parameter files:

```
> mkdir run_params
```

Using `generate_yamlsetup` you can automatically generate the recording specific parameter file and set the recording id and source file from which to import the raw data (it basically does a find-replace for `ANIMAL_BLOCK_ELECTRODE` and `THIS_SOURCE_FILE` in your template). First, you enter the toolbox and call `python` or `ipython`:

```
> cd EAdetection > ipython
```

In python you can now generate your recording specific parameter file:

```

from core.helpers import generate_yamlsetup
template_path = '/home/weltgeischt/EAdetection_tutorial/my_templates/template_
↳ experiment1.yml'
rec_id = 'my_recording' # I recommend using animalID_elecID_recID (eg AN115_El2_rec5)
datasource = 'my_example_data.smr'
setuppath = '/home/weltgeischt/EAdetection_tutorial/run_params/%s_params.yml'%rec_id

```

(continues on next page)

(continued from previous page)

```
generate_yamlsetup(rec_id, datasource, template_path, setuppath=setuppath)
exit() #to leave python
```

In this case *rec\_id* is the id of the recording. I recommend to give it some meaningful name specifying the animal, electrode and number of recording. You can *download this example .smr file\_* to replicate what happens in this tutorial. The variable *setuppath* specifies where the recording specific parameter file will be saved. Et voilà, now you have created your recording specific parameter file at *run\_params/my\_recording\_params.yml*.

You can edit further edit your recording specific parameter file by hand in the same way as :ref:‘ described previously for the template <overwrite\_defaults>’.

---

**Note:** Have a look at *EAdetection/examples/generate\_paramfile.py* for creating several parameter files in a loop.

---

---

**Todo:** Make a link to *download this example .smr file\_*.

---

**Alternative:** If you don’t like python, you can of course find/replace `ANIMAL_BLOCK_ELECTRODE` and `THIS_SOURCE_FILE` by hand too.



## PREPROCESSING

Both preprocessing and *EA detection and classification* work as a command line tool. The basic command structure is like this:

```
> python <path/to/codefile.py> <path/to/paramfile.yml>
```

### 3.1 Load and resample

The goal of this is to transfer obtain a resampled version of the raw data in hdf5 format. To date the loading supports .smr and .edf files.

Just run:

```
> python runthrough/rawToResampled.py run_params/my_recording_params.yml
```

Depending on what you put in your specific parameter file, loading and resampling either just run through *automatically*, which is nice when processing several recordings in a loop, or you can select the channel *interactively*.

#### 3.1.1 Automatic mode

For this you need to know the channel you want to access and enter it directly in the parameter file:



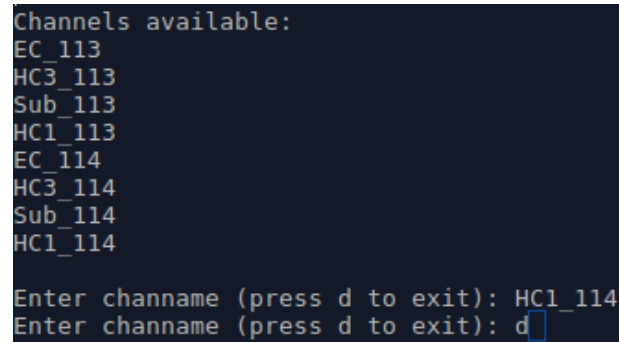
```
64
65 preprocessing:
66
67   loading:
68     source: /home/weltgeischt/EAdetection_tutorial/example_data/my_example_data.smr
69
70     channel: HC1_114 # either the channel ID if known or interactively select the ch
71
72
73
```

#### 3.1.2 Alternative: Interactive mode

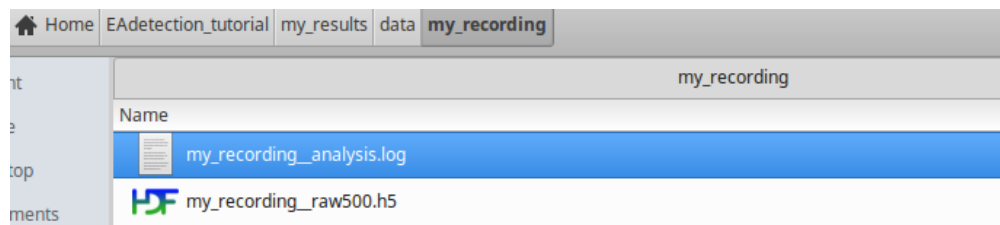
In case you do not know the channel name and want to select a channel interactively, *channel* should be set to *interactive* in the specific parameter file:

```
65 preprocessing:
66
67     loading:
68         source: /home/weltgeischt/EAdetection_tutorial/example_data/my_example_data.smr
69
70         channel: interactive # either the channel ID if known or interactively select th
71
72 --
```

In interactive mode you then type the name of the channel you want to select and exit with d:

A terminal window with a dark background. It displays a list of available channels: EC\_113, HC3\_113, Sub\_113, HC1\_113, EC\_114, HC3\_114, Sub\_114, and HC1\_114. Below the list, it prompts 'Enter channame (press d to exit):' and the user has entered 'HC1\_114'. The next prompt is 'Enter channame (press d to exit): d' with the cursor on the 'd'.

You now have created a resampled .hdf5 at `/my_results/data/my_recording/` and a log-file of the analysis (which is nice for handing in, in case your analysis is not working properly):



---

**Note:** You can skip this loading and resampling routine and use your own resampled hdf5 file. Take care, however, to adhere to the format given in `EAdetection_tutorial/my_results/data/my_recording/my_recording__raw500.h5`

---

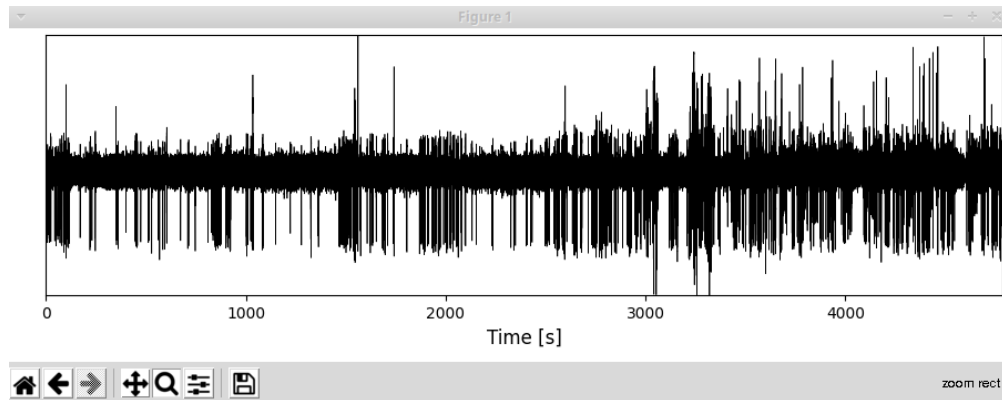
## 3.2 Assign polarity

Polarity refers to the direction of the spike component in EA. To interactively determine and set the polarity of your recording, run the following command:

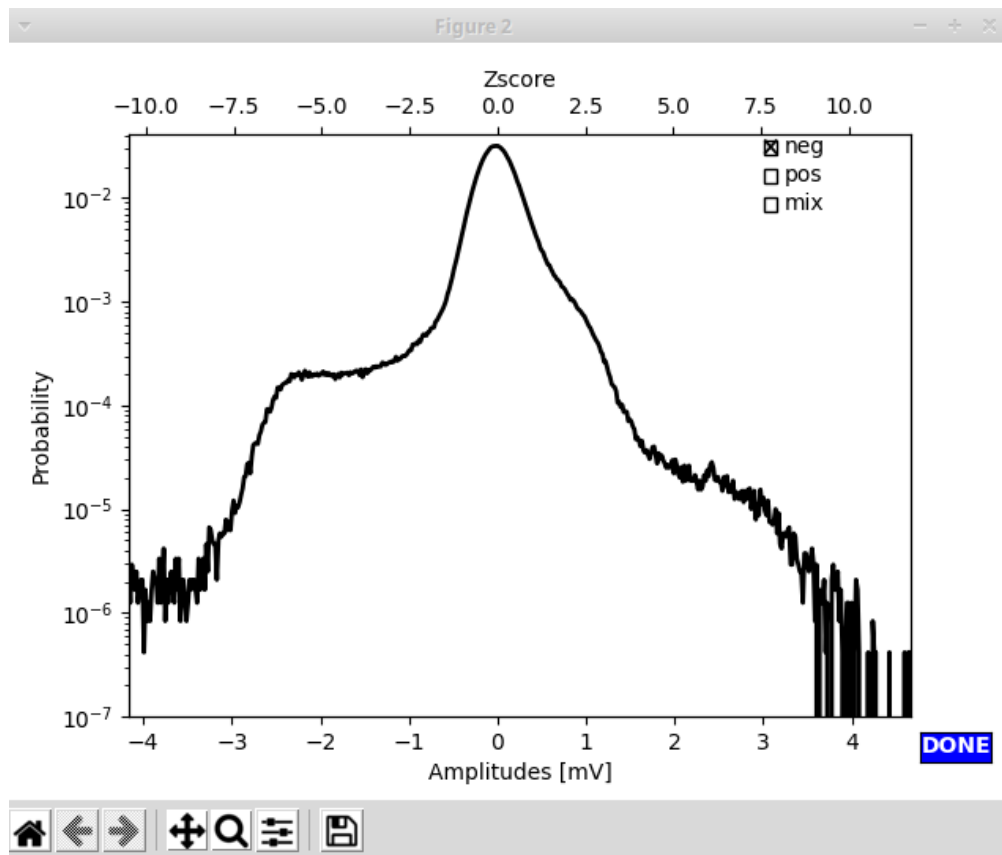
```
> python runthrough/polarityCheck.py run_params/my_recording_params.yml
```

Two windows will pop up: A LFP trace of the whole recording...

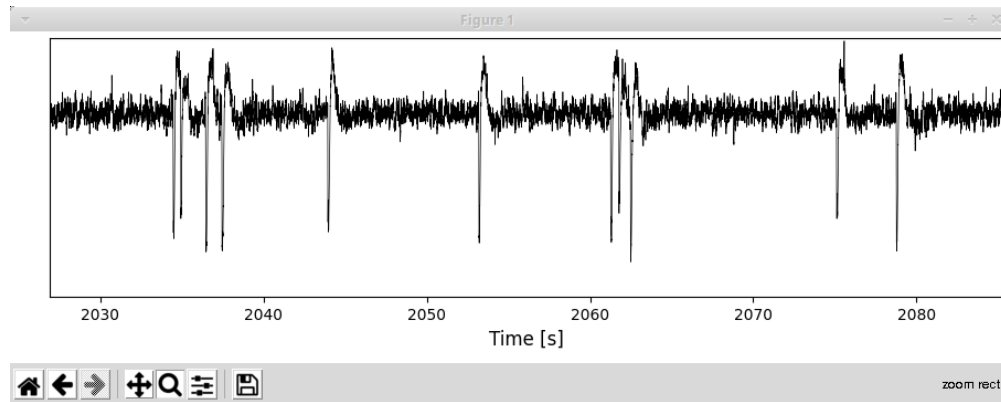




... and an amplitude distribution, with checkboxes:



In this example you can see a shoulder at negative amplitudes, this strongly suggests, that the polarity of this example recording is negative. To be sure you could also zoom around in the LFP trace that just popped up. As you can see, the polarity indeed appears to be negative (the spike component goes down).



By marking a checkbox in the upper right corner of the window with the amplitude distribution, you select a polarity. Clicking **Done** (bottom right) ends the whole procedure, and a simple `.txt` file is created at `EAdetection_tutorial/my_results/data/my_recording/my_recording__polarity.txt`:

```
File Edit Search View Document Help
1 neg
2 checked=True
```

---

**Note:** If you know the polarity of your recordings anyway, you can yourself create a file `my_recording__polarity.txt` and do not need to follow the interactive routine to determine the polarity. If the file `my_recording__polarity.txt` is not present, later analyses will assume default polarity (negative).

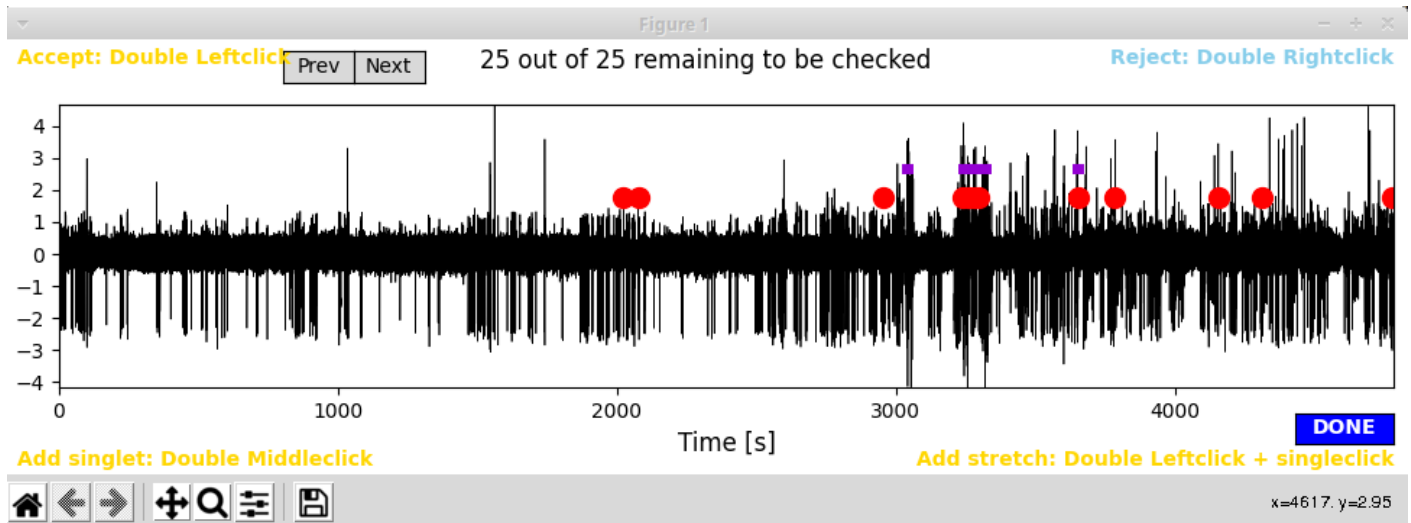
---

### 3.3 Detect artifacts

To run the semi-automatic artifact detection, execute this:

```
> python runthrough/artifactCheck.py run_params/my_recording_params.yml
```

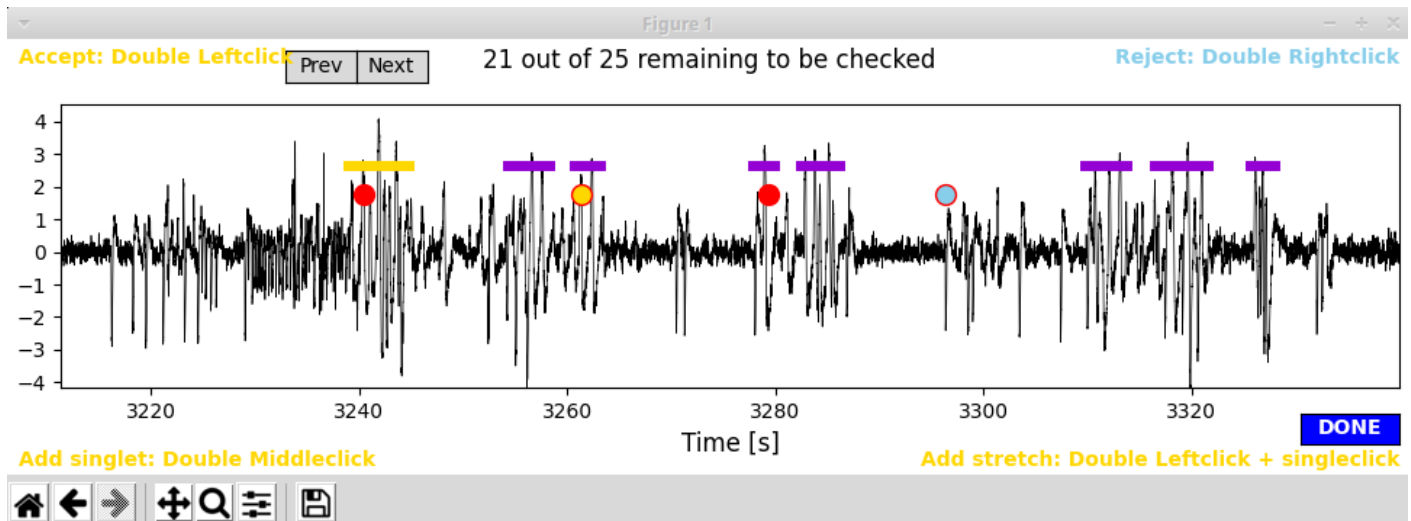
A window displaying the whole extent of the recording session will pop up. In it single events the algorithm detected as potential *saturation artifacts* are marked by red dots and potential *longer stretches of artifacts* are marked by purple lines:



**Accepting suggested artifacts:** Zoom around to have a look whether you want to accept any of the proposed artifacts. Accepting an artifact means that this stretch of data (plus a margin for the *saturation artifact*) will be masked for further analyses. To accept an artifact, double left click on its marker. Once you accepted the artifact its marker will turn yellow.

**Rejecting suggested artifacts:** double right click to reject a suggested artifact. Once rejected, the marker of the artifact will turn blue:

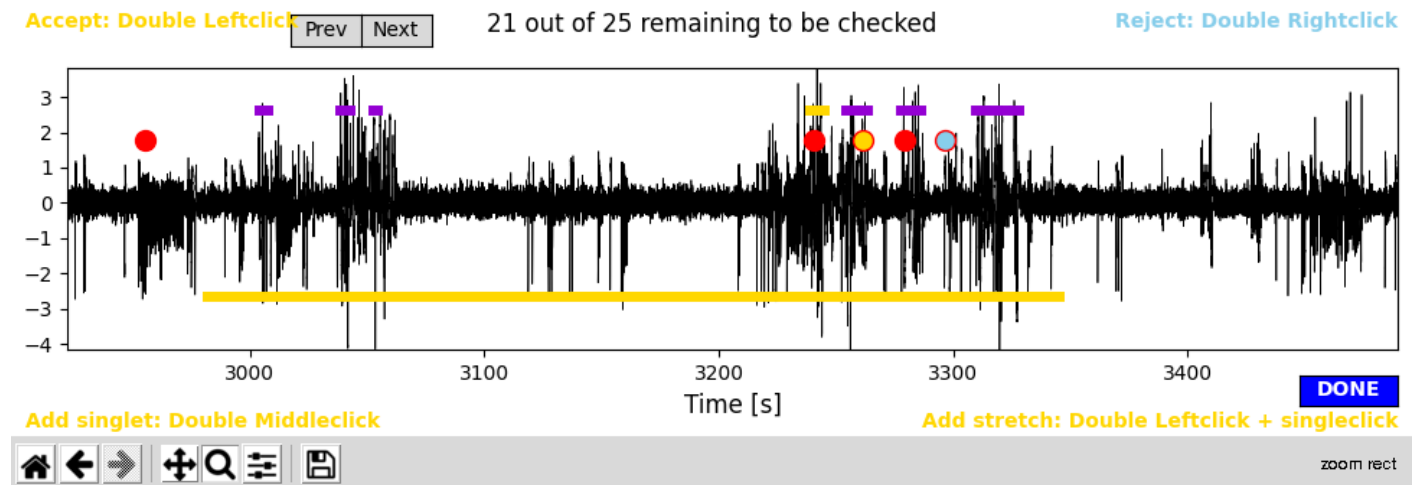
**Warning:** Only artifacts marked in yellow, i.e. accepted artifacts, will be saved as artifacts. The functionality of rejecting artifacts is just there to better keep track of which artifacts you accepted. By default, all potential artifacts the algorithm highlights will not be masked in later analyses unless you explicitly accept them.



**Note:** Make sure to release the zoom tool (by clicking on it), once you try to accept/reject artifacts. Otherwise selection will not work.

**Adding artifacts yourself:** To add a single artifact event double middle click at the position where you have identified it. A yellow dot will appear. For adding longer stretches of artifacts double left click at the position where you think the artifact starts and then single left click where you think the artifact ends. This

artifact stretch will be indicated by a yellow line. We typically exclude large stretches of data in that way when they appear to be peppered by artifacts.



**Important:** For adding your own artifacts, always **click below  $y=0$** . This serves to keep self-identified and automatically suggested artifacts separate.

Once you are finished with artifact hunting, click on the blue Done button. Again a simple .txt file is created at `EAdetection_tutorial/my_results/data/my_recording/my_recording__artifacts.txt`:

```

/home/weltgeischt/EAdetection_tutorial/my - + x
File Edit Search View Document Help
1 (artstart,artstop)
2 (3238.87, 3244.73)
3 (2981.61, 3345.16)
4
5 saturation artifacts
6 3261.34
7 4155.07
8

```

Below the header `(artstart,artstop)` the start and end points of large artifact stretches are indicated (yellow lines in the GUI, see above). Below the header `saturation artifacts` time points of single artifact events are given (yellow dots in the GUI). Don't worry if some artifact stretches overlap in time (as shown in this example) - the tool can resolve this automatically.

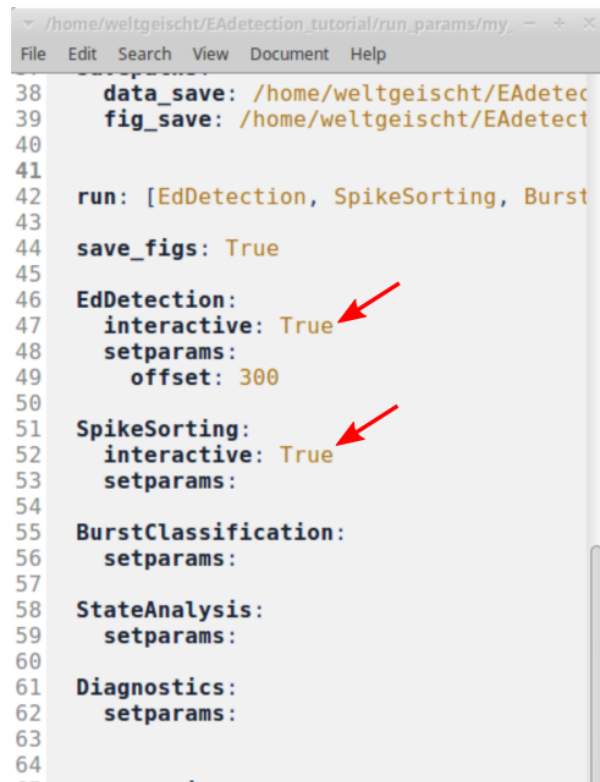
**Note:** You can edit the file `my_recording__artifacts.txt` by hand or altogether avoid the interactive routine described here and create such a file yourself. If the file `my_recording__artifacts.txt` is not present, later analyses will assume that there are no artifacts.

## EA DETECTION AND CLASSIFICATION

Spike detection and burst detection/classification are called with a single command. If you are not interested in bursts and prefer to just run the spike detection you can do so by excluding *BurstClassification* from the list of *analysis:run* in the parameter file at the template or recording specific level (see [Setting parameters](#)):

```
34
35 analysis:
36
37   savepaths:
38     data_save: /home/weltgeischt/EAdetection_tutorial/my_results/data/
39     fig_save: /home/weltgeischt/EAdetection_tutorial/my_results/figure:
40
41
42   run: [EdDetection, SpikeSorting, BurstClassification, Diagnostics]
43
44   save_figs: True
45
46   EdDetection:
47     interactive: True
48     setparams:
49       offset: 300
50
51   SpikeSorting:
52     interactive: True
53     setparams:
54
```

*Spike detection* is followed by *burst classification*. Spike detection consists of three consecutive steps: *spectral spike detection* (the main step) complemented by *amplitude based spike detection* for reducing false negatives and *spike sorting* for reducing false positives. Amplitude based spike detection and burst classification runs fully automatically. For spectral spike detection and spike sorting there is a **GUI** that allows you to set parameters manually. Depending on whether you want to use the GUI or rely on default parameters, have *interactive* set to True or False in the parameter file:



```
38 data_save: /home/weltgeischt/EAdetec
39 fig_save: /home/weltgeischt/EAdetec
40
41
42 run: [EdDetection, SpikeSorting, Burst
43
44 save_figs: True
45
46 EdDetection:
47   interactive: True
48   setparams:
49     offset: 300
50
51 SpikeSorting:
52   interactive: True
53   setparams:
54
55 BurstClassification:
56   setparams:
57
58 StateAnalysis:
59   setparams:
60
61 Diagnostics:
62   setparams:
63
64
```

Execute the spike detection and burst classification through the command-line:

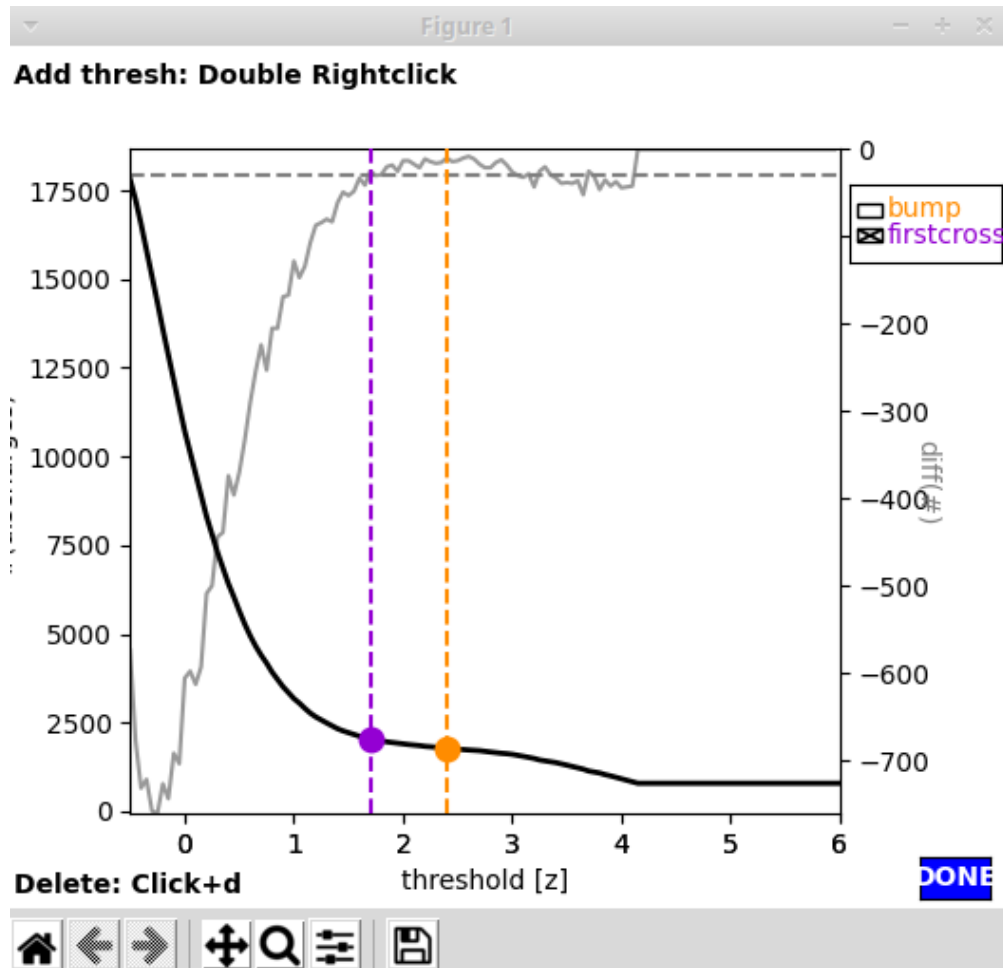
```
> python runthrough/LFPtoBursts.py run_params/my_recording_params.yml
```

Now your PC may be busy for a few minutes (about 3 min for a 2h recording on an ok Laptop). If you chose not to use a GUI (`interactive:False`) nothing will pop up and you can skip forward to [Output](#):

## 4.1 Spike detection

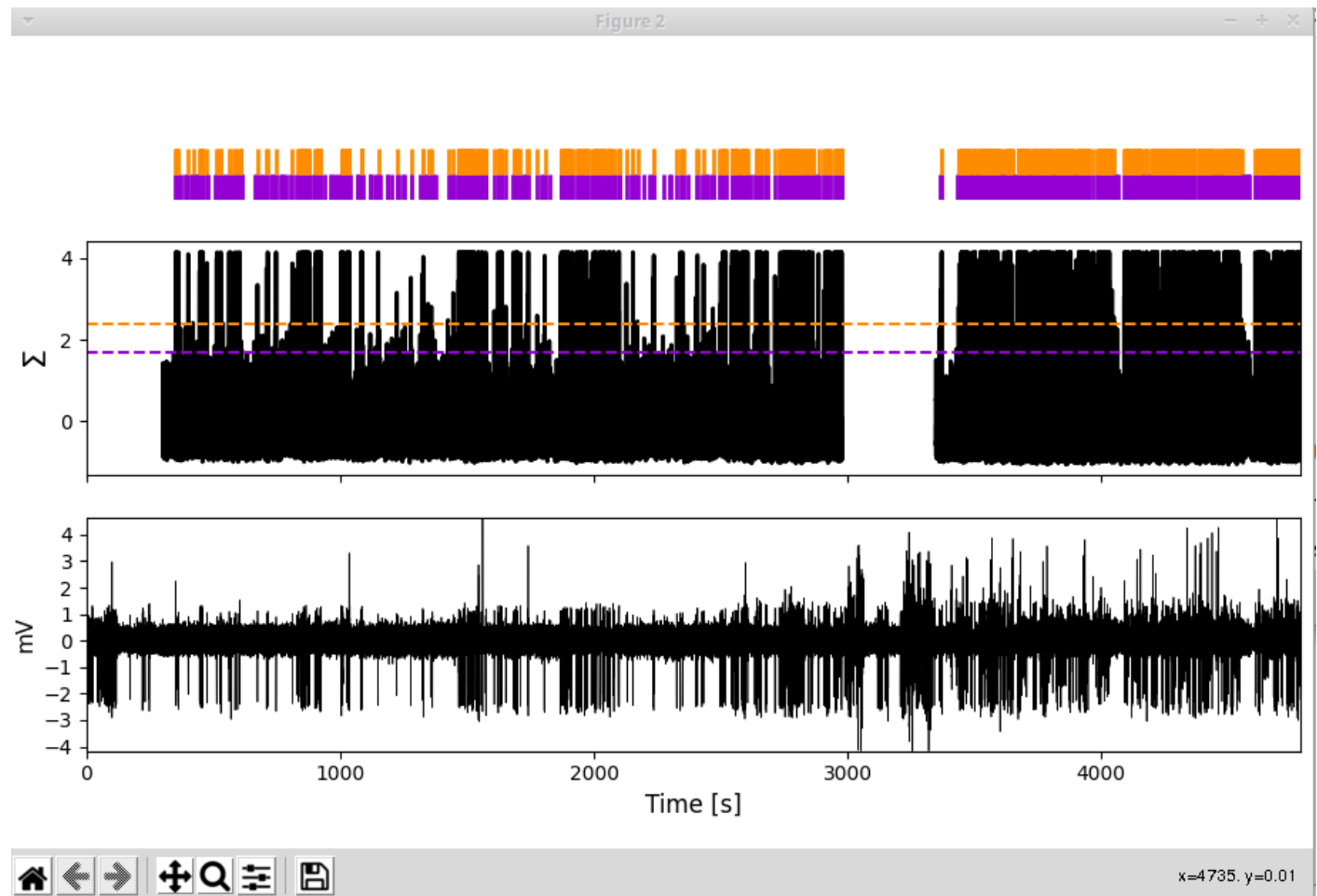
### 4.1.1 Spectral spike detection (interactive)

Using the GUI for spectral spike detection (`EdDetection:interactive:True` in the parameter file) two windows will pop up. The first window shows  $n(\theta)$ , i.e. the number of spikes as a function of the spectral threshold. The two lines indicate what the algorithm interprets as sensible thresholds. By default, *firstcross* (the beginning of the plateau region, corresponding to  $\theta_a$  in the *paper*\_) would be chosen, but also *bump* (the middle of the plateau, corresponding to  $\theta_b$ ) might be a good choice.



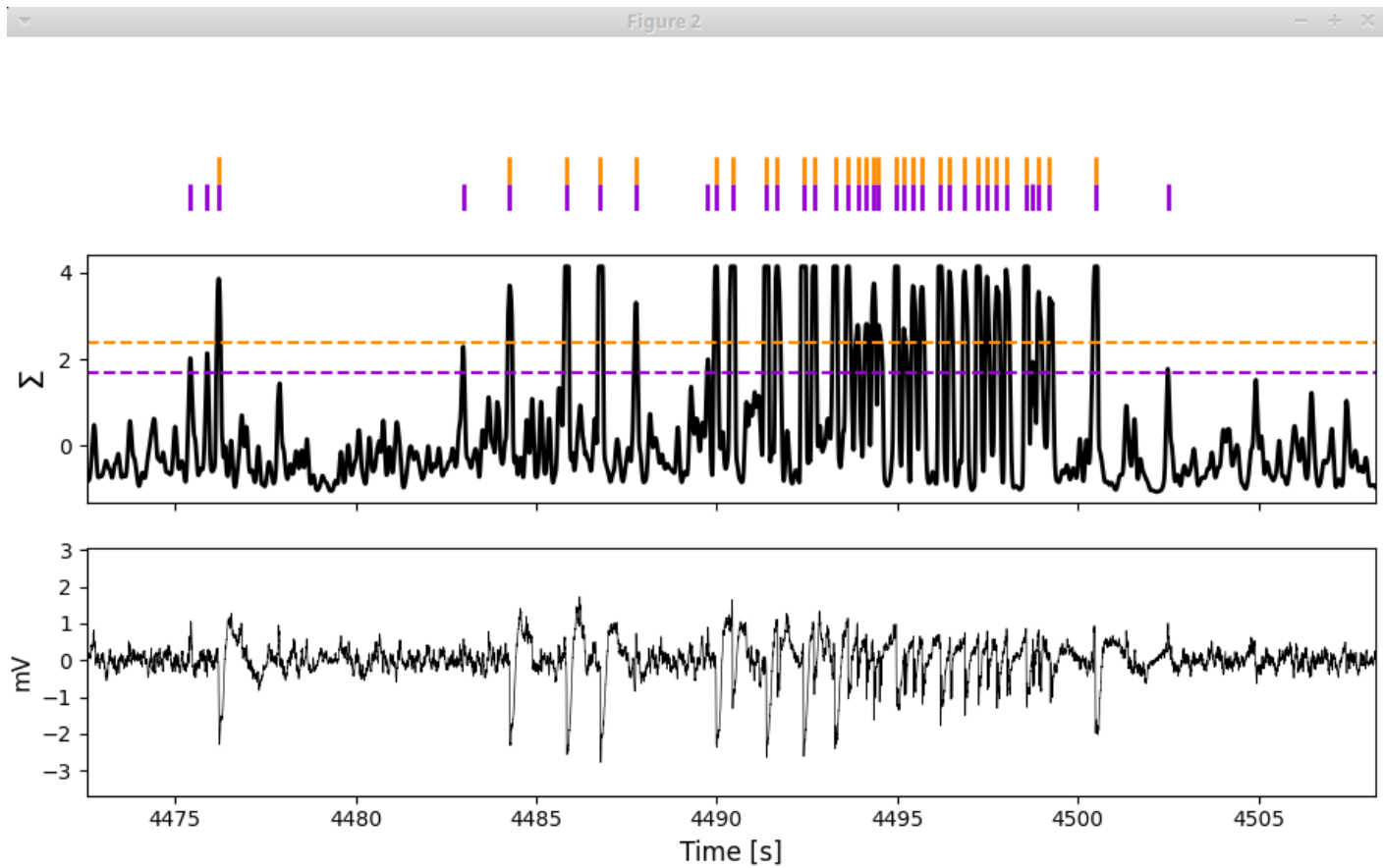
**Todo:** Link to the *paper*\_

The second window displays the *LFP* (bottom), the *normalized spectral sum* to which the threshold is applied (middle, thresholds shown as horizontal lines) and the *spikes detected* when applying the threshold (top).

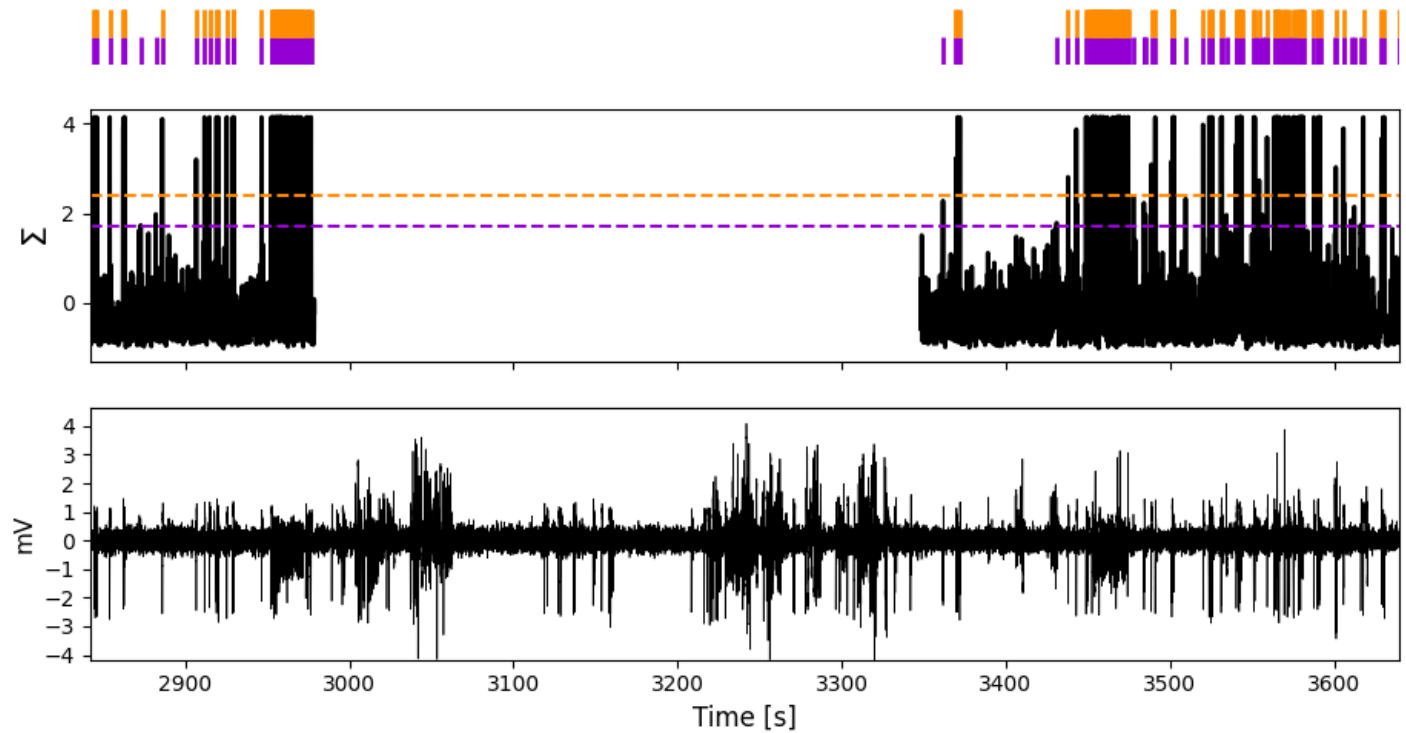


Zoom around in this window to see how happy you are with the performances of each of the thresholds:

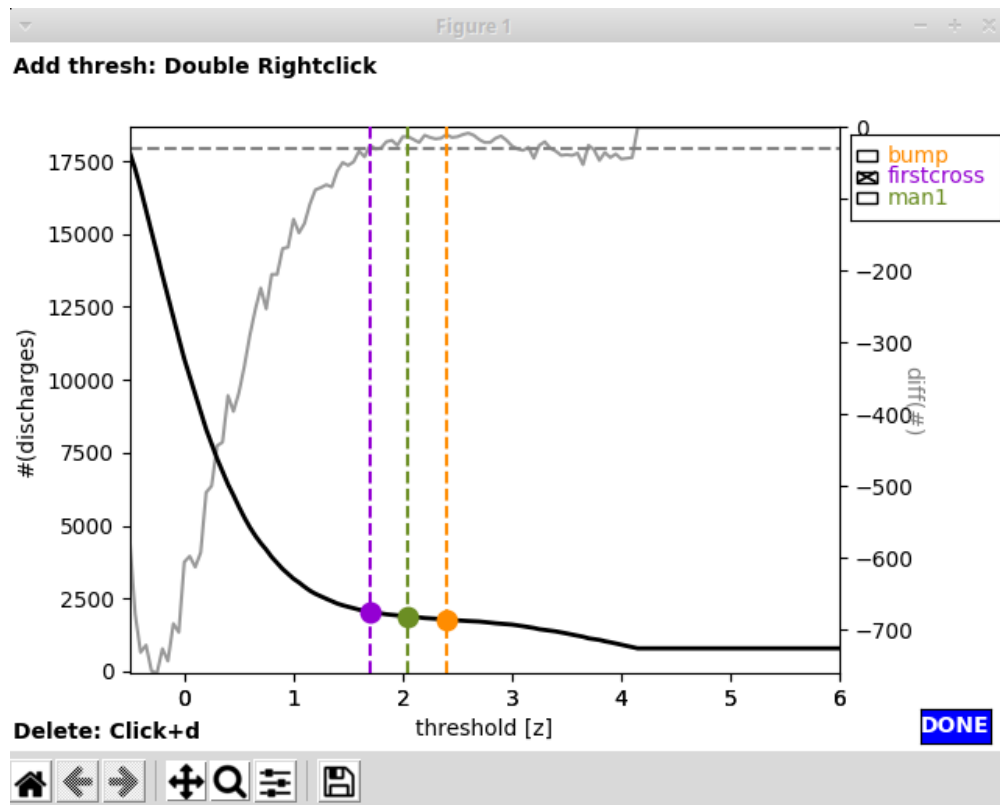




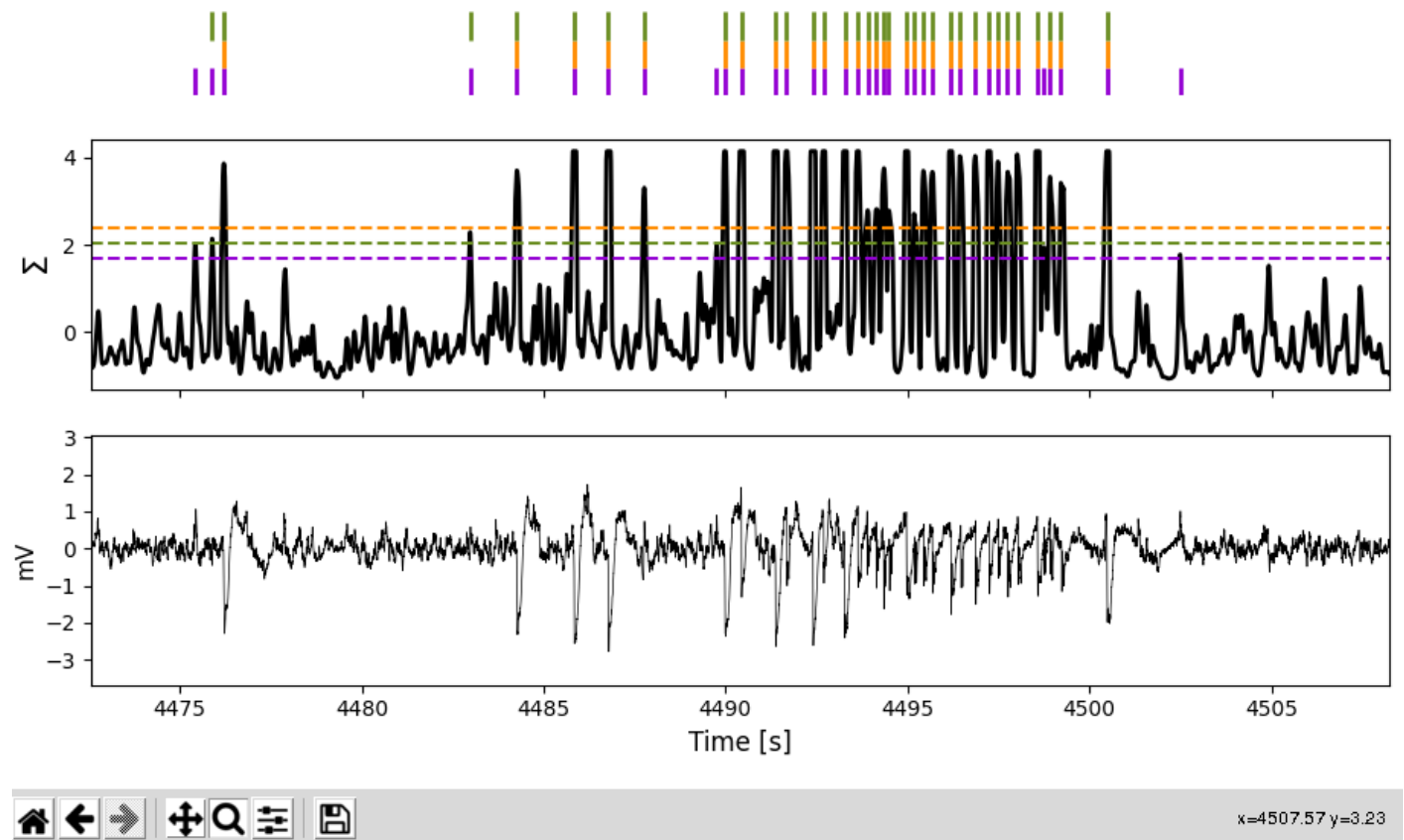
And by the way, the missing values in the normalized spectral sum in this example are due to the artifact we had assigned in this region:



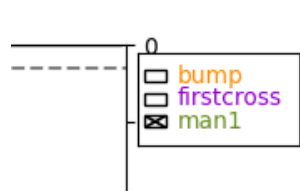
**Manually adding a threshold:** In case you are unhappy with the performance of *firstcross* and *bump*, you can add a custom threshold yourself by a *double right click* on  $n(\theta)$  in the first window:



In this case we added a threshold (green) between *firstcross* and *bump*. Upon adding a new threshold for inspection it will also show up in the second window, so you can judge its performance:



**Picking a threshold:** For setting a threshold click the respective checkbox. Here we pick our manual threshold *man1*:



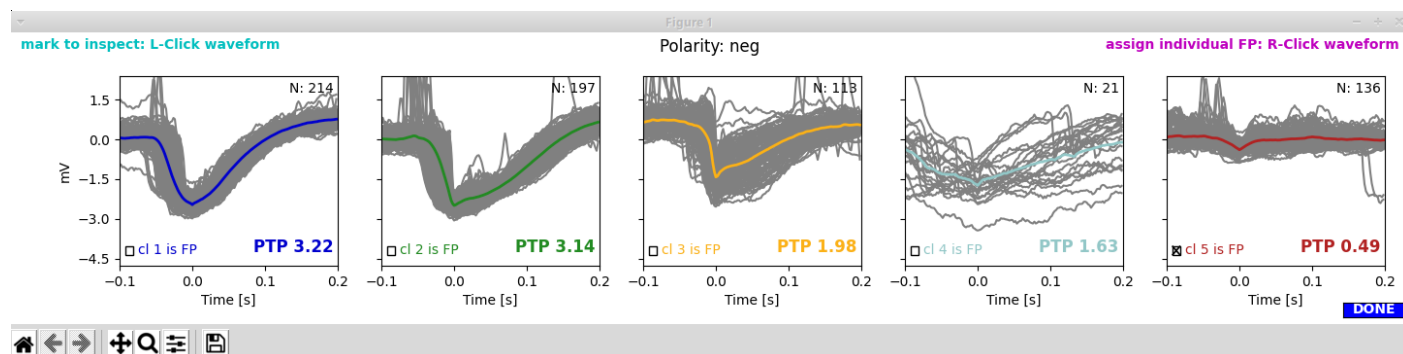
To finish the spectral spike detection, click the blue *Done* button in the first window.

**Important:** I highly recommend to be rather liberal with the threshold at this stage, ie. opt for a rather low threshold such as *firstcross* and don't worry about some false positives. False positives will later be removed through *Spike sorting (interactive)*.

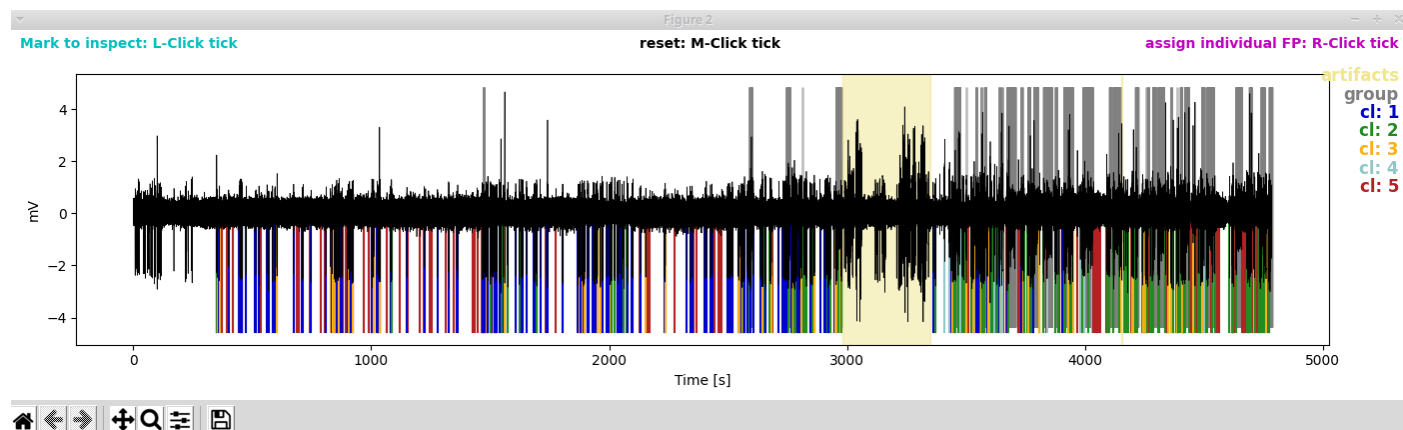
### 4.1.2 Spike sorting (interactive)

With spike sorting we try to detect (and later remove) false positives through clustering waveforms. If you are in interactive mode for spike sorting (`SpikeSorting:interactive:True` in the parameter file), two more windows will pop up now. The panels of the first window show the clusters (waveforms of individual spikes as thin yellowish lines, average waveforms in colors). The clusters are arranged descending from highest to lowest peak-to-peak (PTP) amplitude of

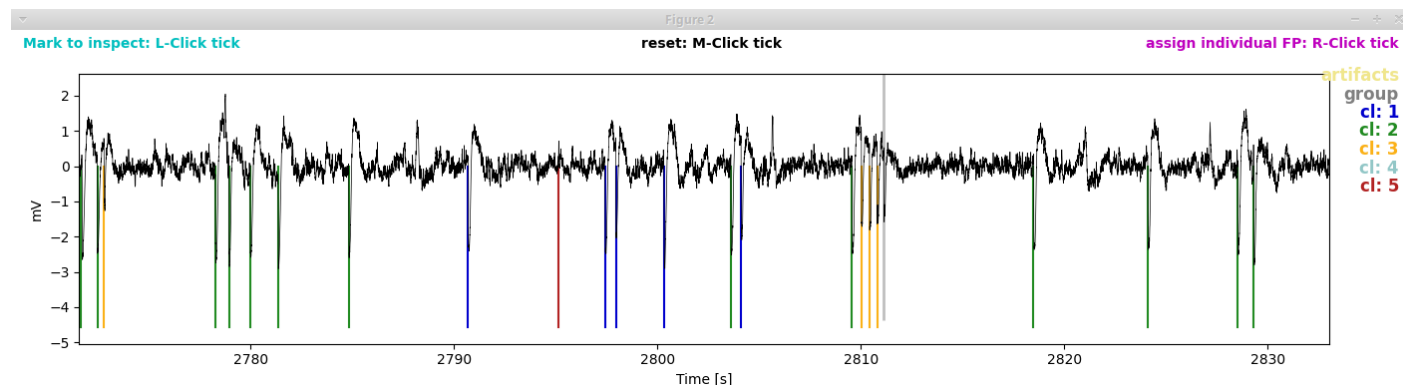
the average waveform. By default the cluster with the lowest PTP amplitude would be discharged (cluster 5 shown in red):



The second window shows the LFP trace with spikes annotated (colored vertical lines) according to the cluster to which they were assigned. The grey lines indicate spikes that had not been subjected to clustering because they occurred in dense *groups* and thus their waveforms could not be separated (we do not worry too much about false positive detections in dense bursts, as they have only very minor influence on burst classification in the end). The area shaded in yellow marks the region we excluded from analyses by assigning an *artifact*.

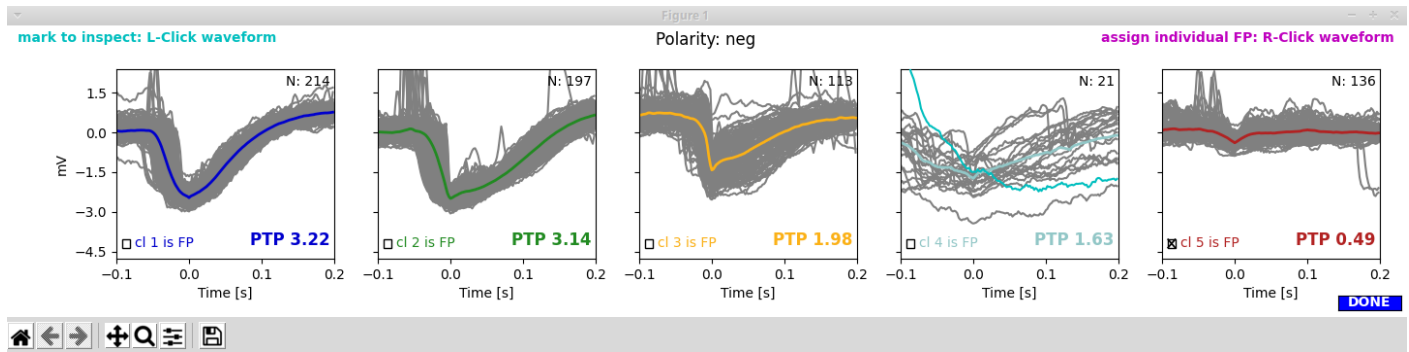


To have a closer look at the spikes in their natural habitat, let's zoom around in the second window:



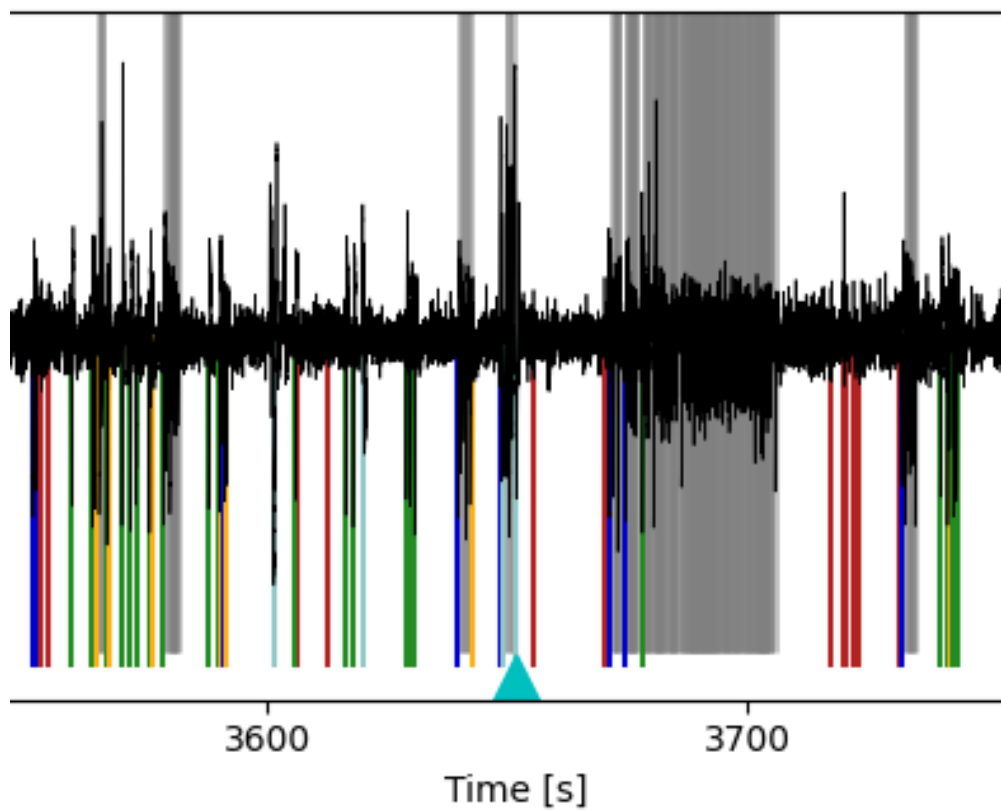
I think clusters 1 to 3 look really decent and cluster 5 clearly is a false positive. But what about cluster 4?

**Marking waveforms for further inspection:** There are really few cluster 4 spikes, and it is hard to spot them in the LFP trace (second window). So let's mark one of those suspicious waveforms by a *left click* in the first window. The waveform now turns cyan:

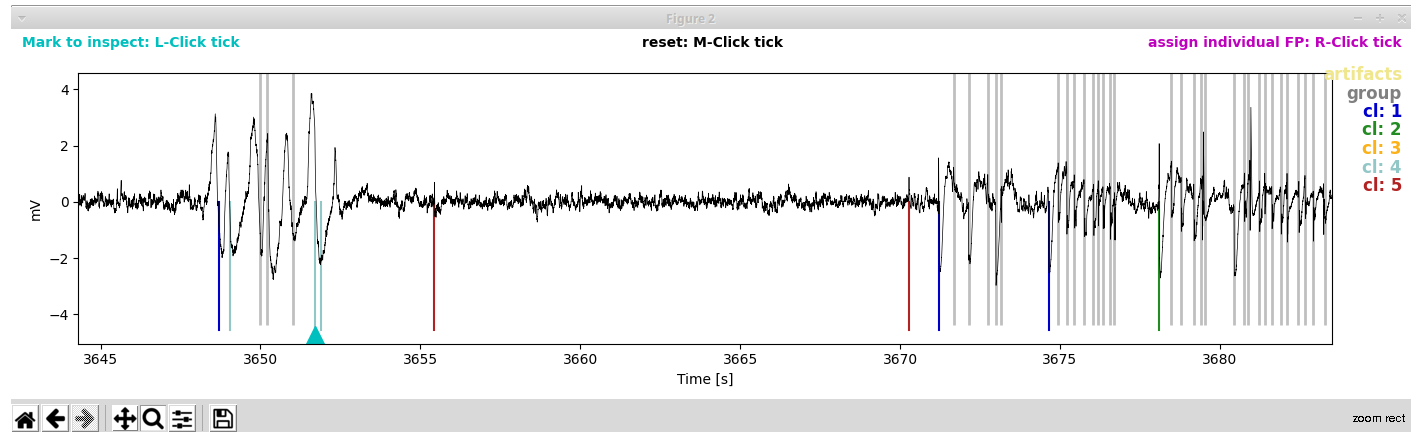


... and the waveform gets marked by a cyan triangle in the second window:

**reset: M-Click tick**

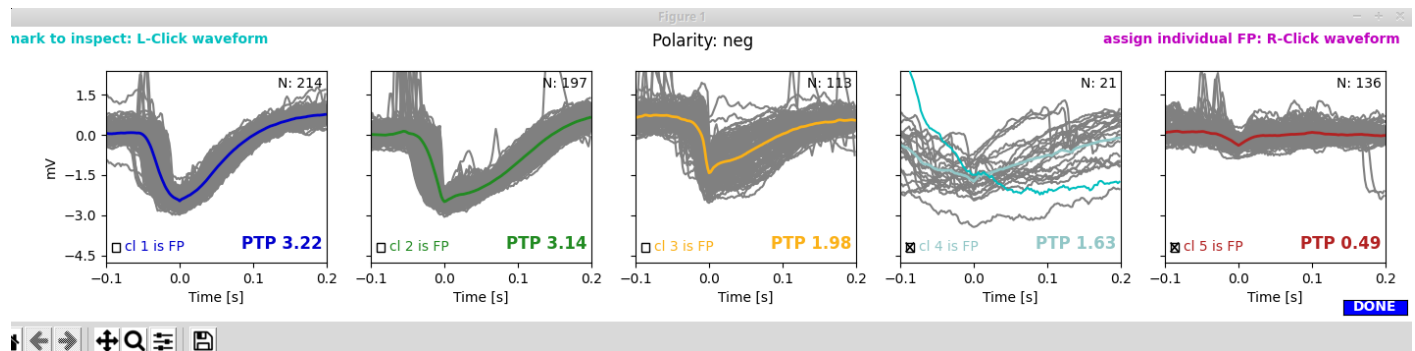


Let's zoom in on this:



Oh dear, the spikes of cluster 4 appear to belong to an artifact that we did not exclude. To be super correct, we should now *edit the artifact file manually*. But as this serves for demonstration purposes only, let's content ourselves with annotating the spikes belonging to cluster 4 as false positives too.

**Picking noise clusters:** You can remove clusters as noise by checking the corresponding check boxes in the first window. In our case, we select clusters 4 and 5 to be discarded as false positives.

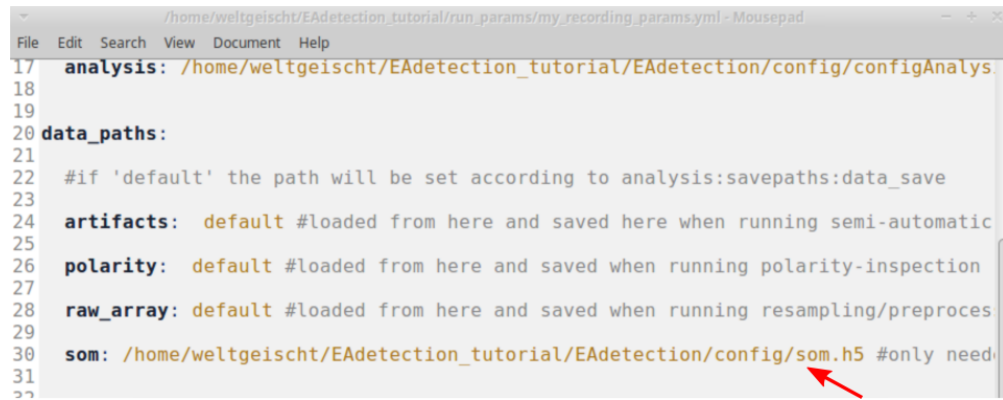


By clicking the blue *Done* button, we save our choice and finish this procedure.

**Note:** If you selected **mixed polarity** when *assigning the polarity*, you will see two rows of clusters, one for positive and one for negative polarity. Likewise, the LFP trace will be annotated with spikes belonging to positive clusters (above the zero line) and spikes belonging to negative clusters (below the zero line). By default the fifth cluster of both negative and positive clusters will be discarded. You can choose separately for positive and negative clusters which clusters to discard as false positives.

## 4.2 Burst detection and classification

The burst classification runs fully automatically by projecting detected spike bursts on a predefined *self-organizing map* (SOM). For the *intrahippocampal kainate mouse model* we once calculated a **reference SOM** on which we project all new datasets. This worked well so far and we included the reference SOM in the toolbox (path: *EAdetection/config/som.h5*). If you use a different kind of animal model with different types of bursts, you might want to select other features and derive your own SOM. To calculate the SOM itself we used the *SOMz-package* by Carrasco Kind and Brunner (“SOMz: photometric redshift PDFs with self-organizing maps and random atlas”, doi:10.1093/mnras/stt2456). You can use your own SOM for the projection method by setting the path to your SOM in the parameter file:



```
17 analysis: /home/weltgeischt/EAdetection_tutorial/EAdetection/config/configAnalys.
18
19
20 data_paths:
21
22 #if 'default' the path will be set according to analysis:savepaths:data_save
23
24 artifacts: default #loaded from here and saved here when running semi-automatic
25
26 polarity: default #loaded from here and saved when running polarity-inspection
27
28 raw_array: default #loaded from here and saved when running resampling/preproces
29
30 som: /home/weltgeischt/EAdetection_tutorial/EAdetection/config/som.h5 #only need
31
32
```





## OUTPUT

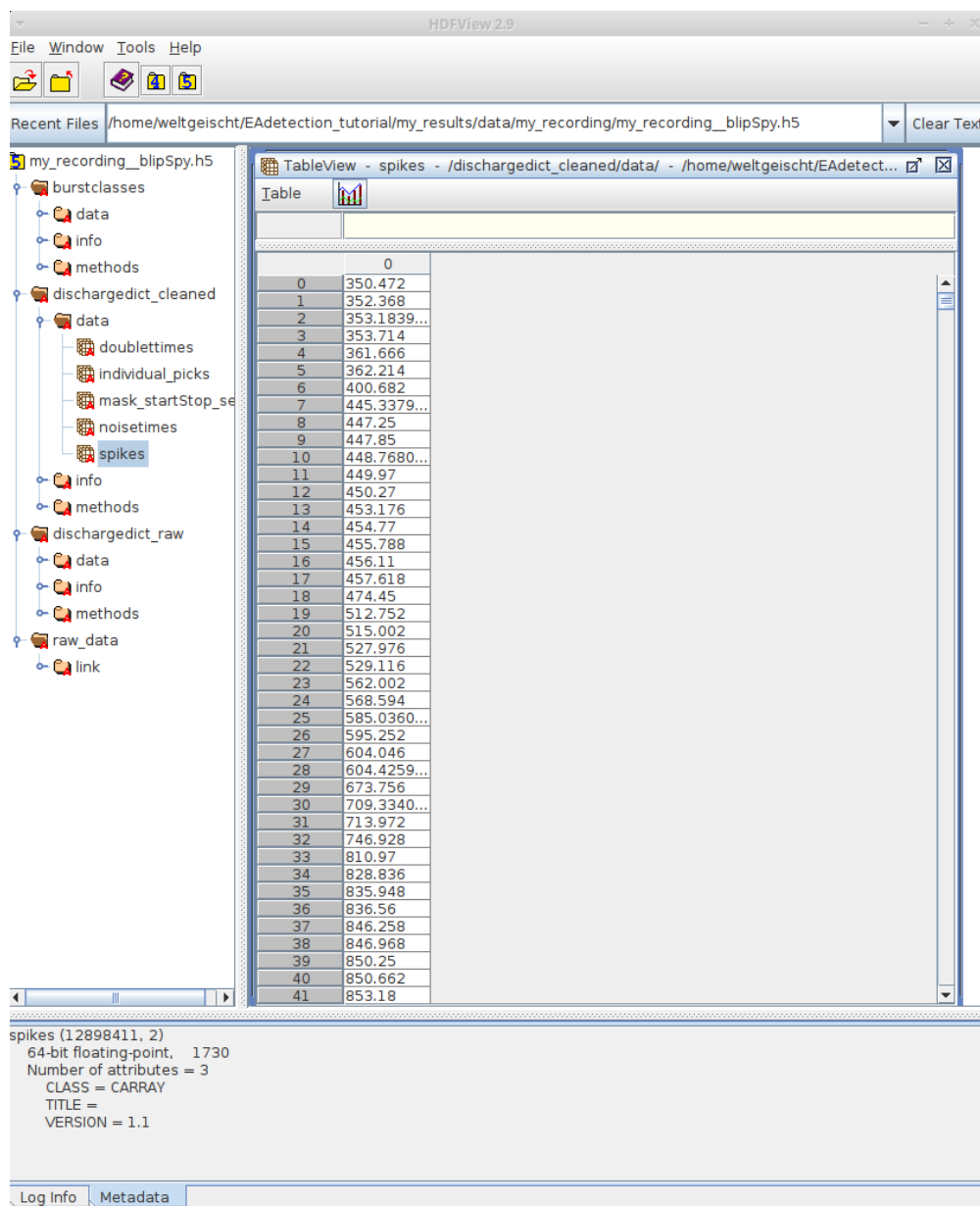
During this analysis run the following folders and files were generated:

```
my_results
├── data
│   └── my_recording
│       ├── my_recording__analysis.log
│       ├── my_recording__artifacts.txt
│       ├── my_recording__blipSpy.h5
│       ├── my_recording__polarity.txt
│       └── my_recording__raw500.h5
└── figures
    └── my_recording
        ├── my_recording__burstClassification
        │   ├── my_recording__examplesOnMap_thisSOM.png
        │   ├── my_recording__mapIds_to_ROIIds.txt
        │   └── my_recording__traceClassified_thisSOM.png
        ├── my_recording__spikeDetection
        │   ├── my_recording__ampDetection.png
        │   ├── my_recording__detExamples.png
        │   └── my_recording__FofThresh.png
        └── my_recording__spikesorting
            ├── my_recording__clusterremoval_ncomp3.png
            ├── my_recording__fpRemoval_ex1.png
            ├── my_recording__fpRemoval_ex2.png
            └── my_recording__fpRemoval_ex3.png
```

The exact location of the figure and data files depend on the location you entered when *editing the parameter files* (specifically how you replaced *DATADIR* and *FIGDIR* in the template).

## 5.1 Results (HDF5)

The folder `my_results/data/my_recording` contains the results of the analyses, most importantly `my_recording__blipSpy.h5`. In this file all important results and intermediate steps (like the normalized averaged spectrogram) are stored. `my_recording__blipSpy.h5` looks like this:

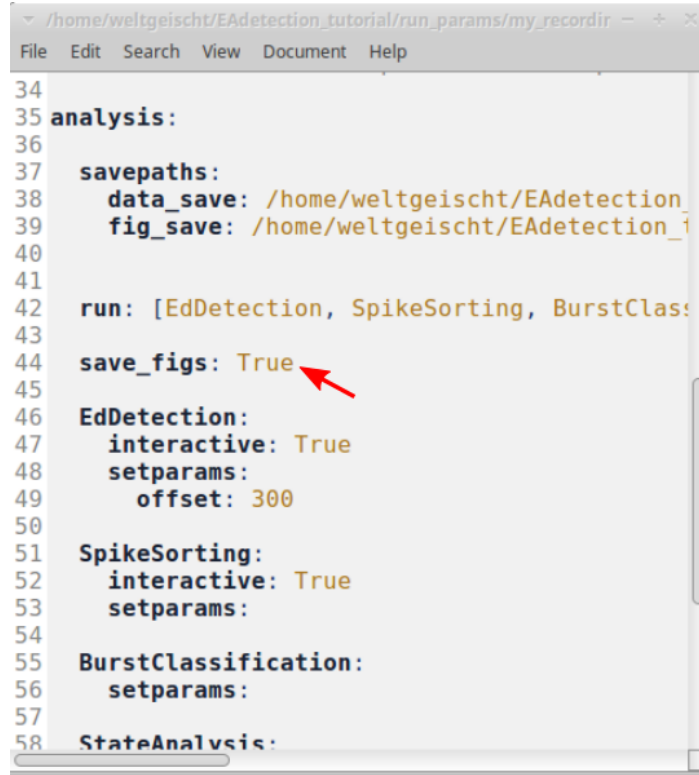


This file contains the timings and classes of bursts (field *burstclasses*), the train of spikes after spikesorting (*dischargedict\_cleaned*), the train of spikes before spikesorting (i.e. the spikes resulting from spectral and amplitude based spike detection, *dischargedict\_raw*) and a link to the raw\_data contained in *my\_recording\_\_raw500.h5* (field *raw\_data/link*). Most of these fields contain three subfields - *data*, where the actual results are stored - *info*, where you find metadata (e.g. the version of the code used and the paths to the figures generated by these analyses) - *methods*, giving the parameters used for a particular analysis step

You can either directly extract the data from these hdf5 files yourself (have a look at [this](#) for inspiration), or use the *data handling tool* we provide to conveniently access and visualize results.

## 5.2 Quality control figures

To assess whether the algorithm performed sensibly, figures were created and saved in the folder *my\_results/data/my\_recording*. You can turn this off in the parameter file (*analysis:save\_figs: False*) ...



```

34
35 analysis:
36
37   savepaths:
38     data_save: /home/weltgeischt/EAdetection
39     fig_save: /home/weltgeischt/EAdetection
40
41
42   run: [EdDetection, SpikeSorting, BurstClass
43
44   save_figs: True
45
46   EdDetection:
47     interactive: True
48     setparams:
49       offset: 300
50
51   SpikeSorting:
52     interactive: True
53     setparams:
54
55   BurstClassification:
56     setparams:
57
58   StateAnalysis:

```

... but I highly recommend you keep saving these figures and routinely check them to see whether everything works well. In the following you will learn how to interpret these figures.

### 5.2.1 Spike detection

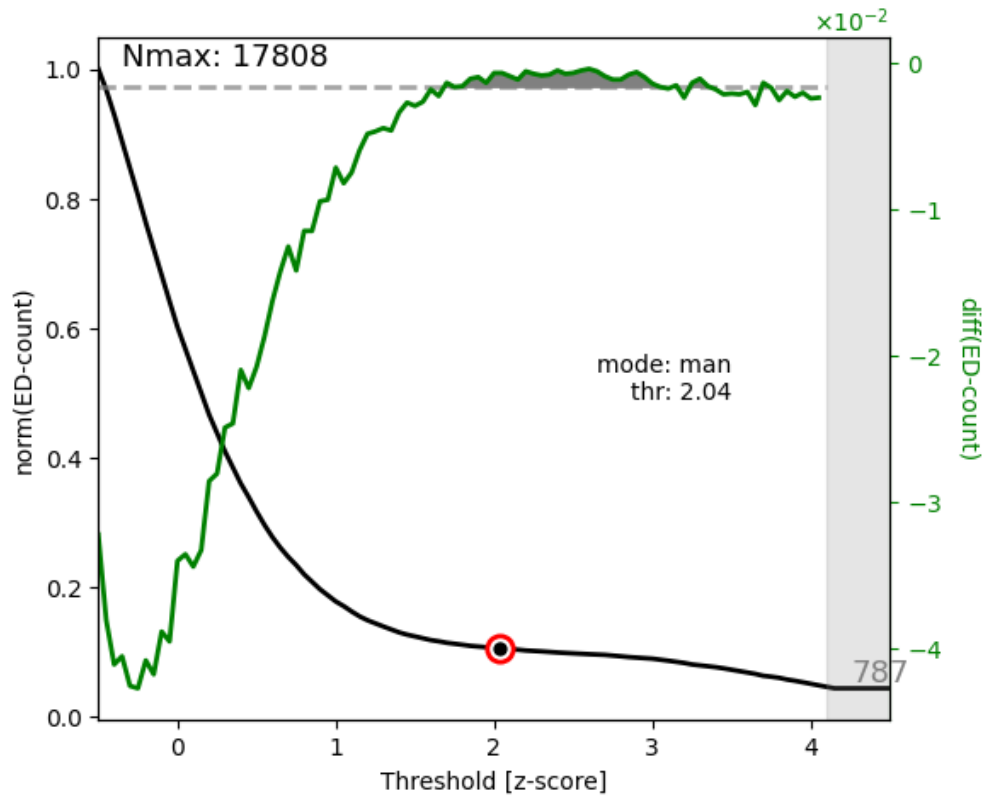
The folder **my\_recording\_\_spikeDetection** contains three figure files:

```

my_recording__spikeDetection
├── my_recording__ampDetection.png
├── my_recording__detExamples.png
└── my_recording__FofThresh.png

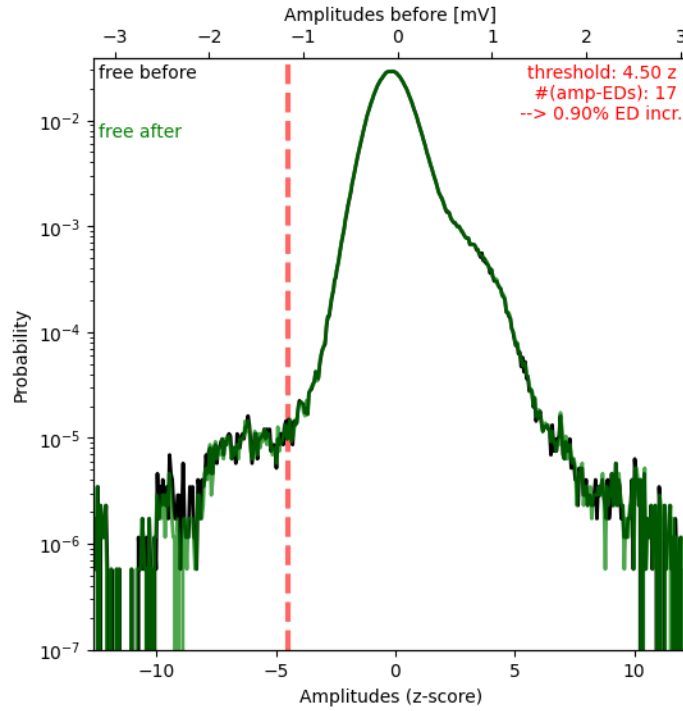
```

The figure *my\_recording\_\_FofThresh.png* displays the number of spikes as a function of threshold ( $n(\theta)$ , see also [spectral spike detection](#)). The circle indicates the threshold selected. As we manually picked the threshold in this example, the *mode* is *man*. The green line displays the derivative of  $n(\theta)$  based on which the plateau region was determined:



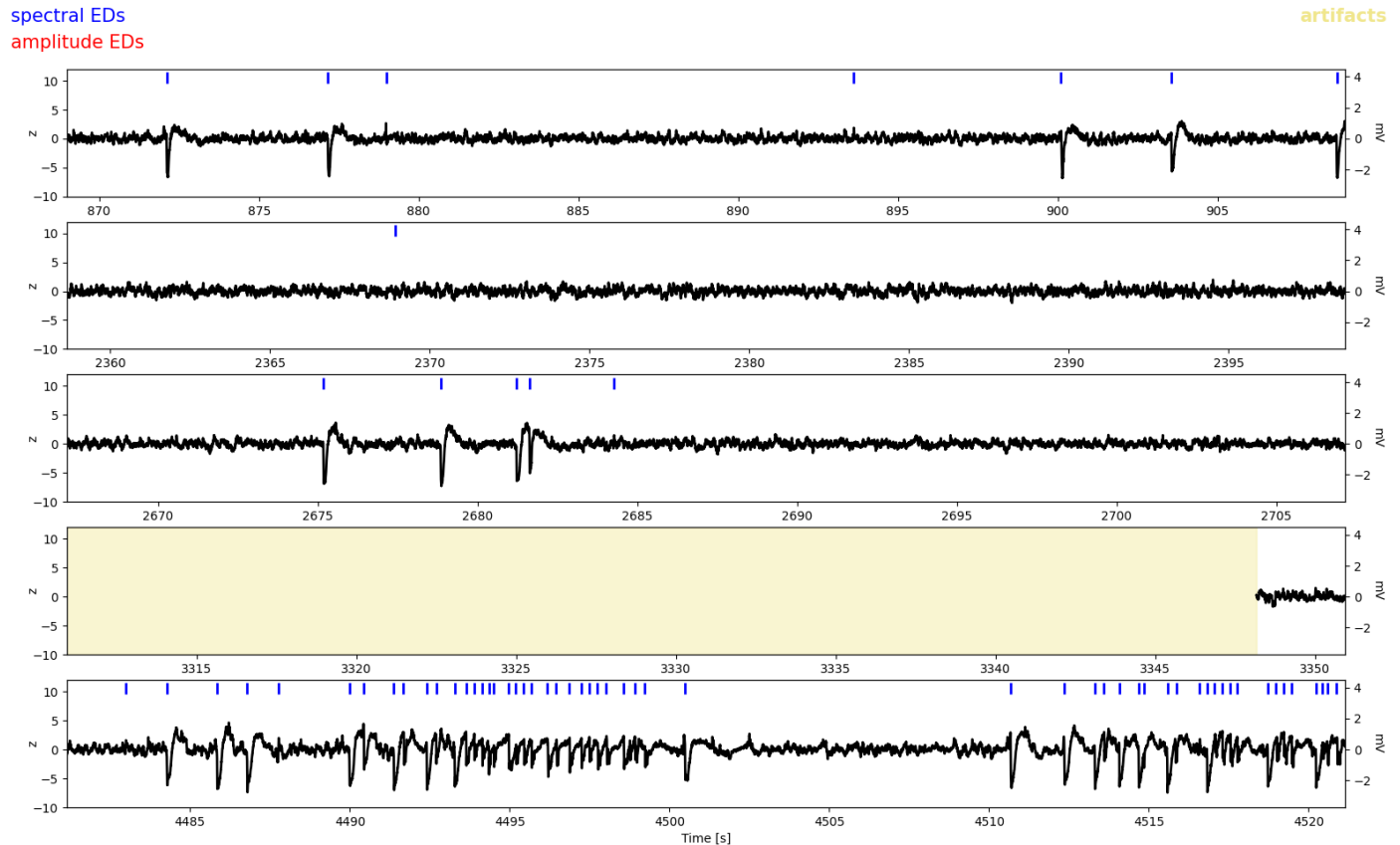
For a proper spike detection we need a plateau region (*paper\_Figure XXX\_*), that is a region where  $n(\theta)$  increases only very slowly. Non-epileptic mice lack such a plateau. Here we are content, because a plateau is clearly visible and the threshold is located nicely within the plateau region.

The figure *my\_recording\_ampDetection.png* displays the amplitude distribution of the LFP with spectral spikes masked (black) and spectral + amplitude spikes masked (green, *paper\_Figure XXX\_*). The red line indicates the value where the amplitude threshold was automatically set (-4.5 due to negative *polarity*) and the red numbers in the upper right corner state again the threshold, the absolute number of amplitude spikes # (amp-EDs), and how the number of spikes increased through amplitude based spike detection:



In this case the shoulder for negative values is very small (note the logarithmic y-axis) and consequently very few spikes were detected as amplitude spikes (0.9% increase only).

The figure *my\_recording\_\_detExamples.png* shows five randomly selected 40s snippets of LFP with spectral and amplitude spikes annotated by blue and red ticks respectively. Regions *annotated as artifacts* are shaded in yellow.



Because there were so few amplitude spikes (i.e. spectral spike detection did a good job), we don't see any red ticks here. But we notice some false positives, eg. in the first and second row. Luckily these were probably removed by *spike sorting*.

## 5.2.2 Spike sorting

The folder **my\_recording\_\_spikesorting** contains one file again displaying the spike clusters and three figures of LFP traces annotated with putative false positives detected through spike sorting:

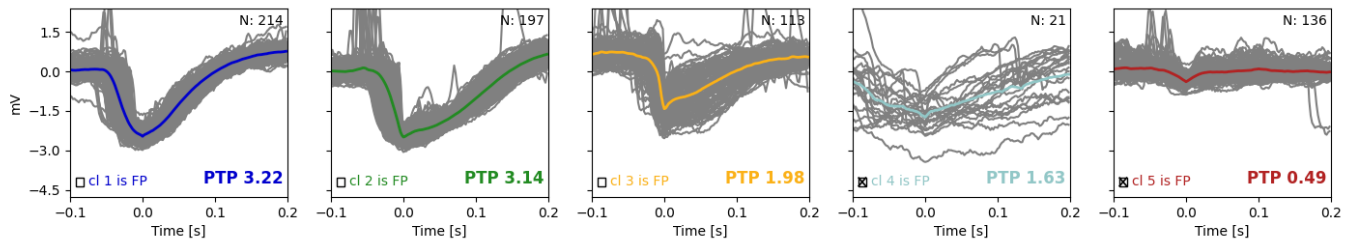
```
my_recording__spikesorting
├── my_recording__clusterremoval_ncomp3.png
├── my_recording__fpRemoval_ex1.png
├── my_recording__fpRemoval_ex2.png
└── my_recording__fpRemoval_ex3.png
```

The figure *my\_recording\_\_clusterremoval\_ncomp3.png* (ncomp3 refers to the default of three principal components used for sorting) have already been seen when running *spike sorting in interactive mode*:

FP clusters are: [4, 5]

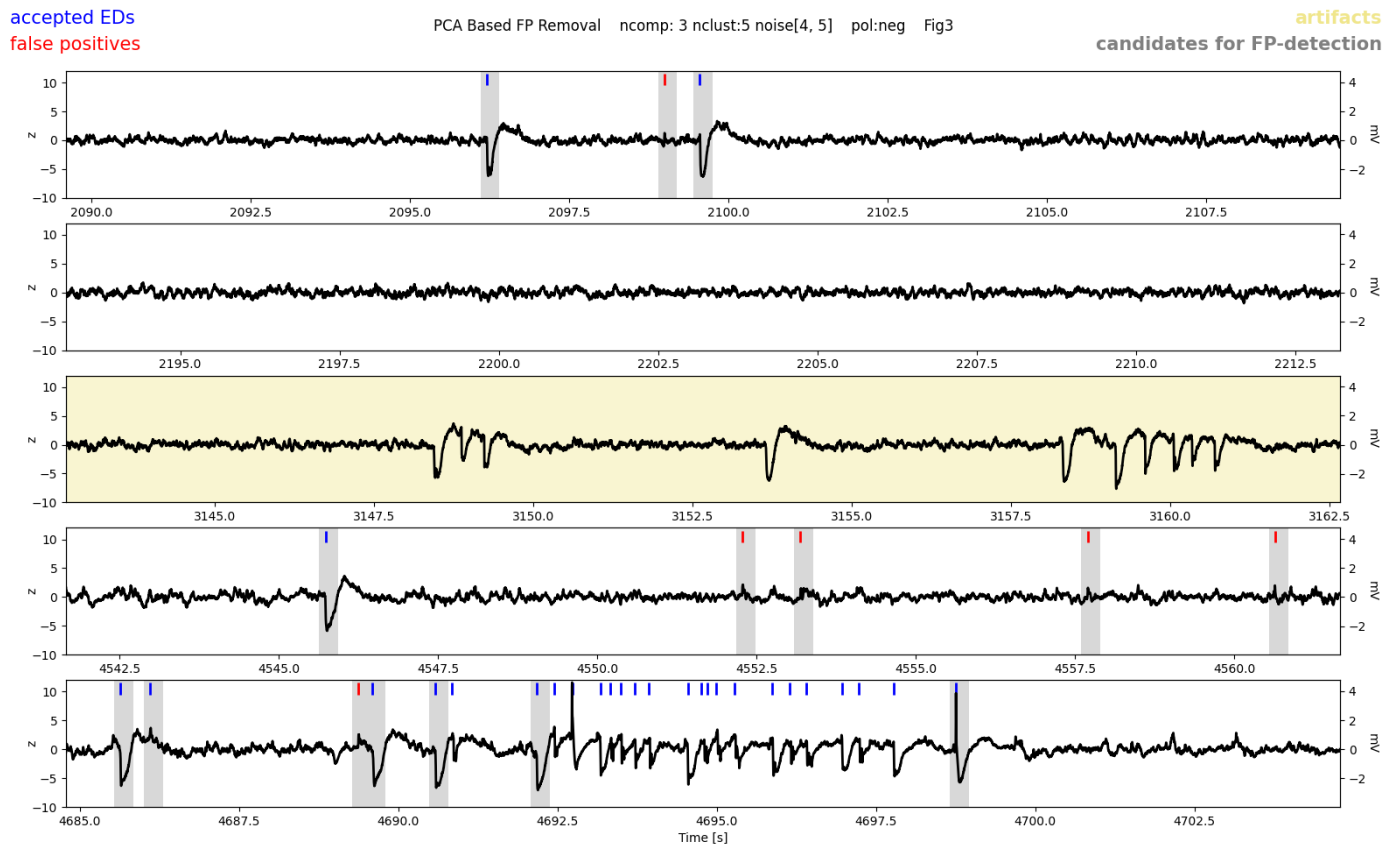
Polarity: neg

individually assigned FPs



If you ran everything automatically (by default only cluster 5 is removed) you would get suspicious and now run the sorting again to remove cluster 4 as well. But as indicated by the *checkmarks*, we already removed cluster 4 as well and are very content, because clusters 1-3 show nice spike-like waveforms, while clusters 4 and 5 do not look like spikes.

The figures *my\_recording\_fpRemoval\_ex1-3.png* show five randomly selected 20s snippets of LFP with non-removed (*accepted EDs*, blue) and false positives detected through spike sorting (red) indicated by tickmarks. The grey regions indicates the waveform cutouts subjected to spike sorting. Regions *annotated as artifacts* are shaded in yellow. Here is one of these example figures:



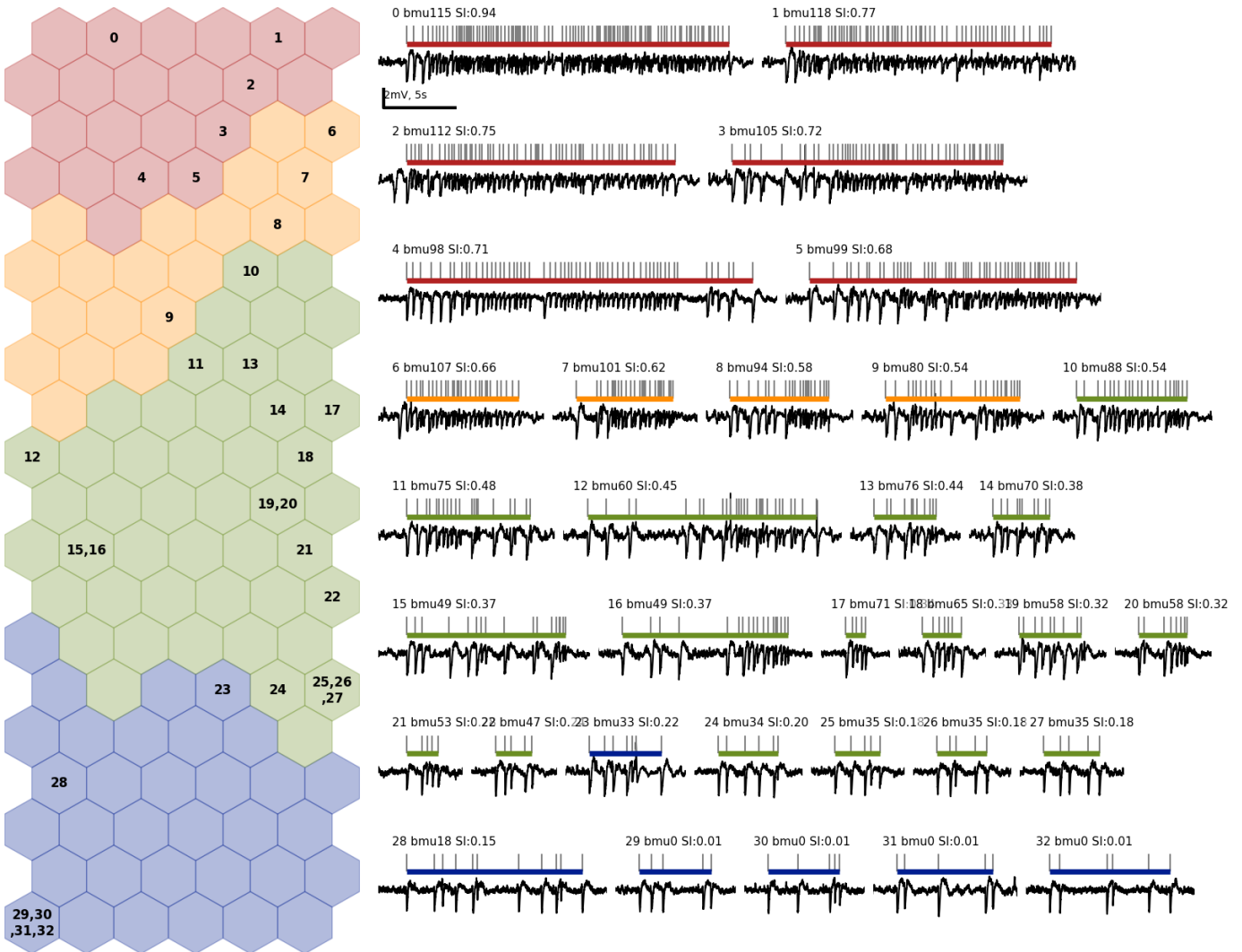
Note that spikes occurring in groups were not subjected to spike sorting (last row). Overall, we are very happy with the performance, most false positives appear to have been detected. If you noticed an awful lot of false positives in these example figures, you might need to *change the threshold for spectral spike detection* or *exclude more/different clusters*. Likewise, if you observed too many false negatives or spikes being wrong- fully labeled false positives here.

### 5.2.3 Burst classification

The folder `my_recording__burstClassification` contains two figures related to burst classification:

```
my_recording__burstClassification
├── my_recording__examplesOnMap_thisSOM.png
├── my_recording__mapIds_to_ROIds.txt
└── my_recording__traceClassified_thisSOM.png
```

The figure `my_recording__examplesOnMap_thisSOM.png` displays the SOM (left) and how spikes bursts (randomly chosen) were assigned to nodes of the SOM (right). The colors indicate the different burst classes. Note that the high-load cluster is sub-differentiated here in *high-load 1* (red) and *high-load 2* (orange):.

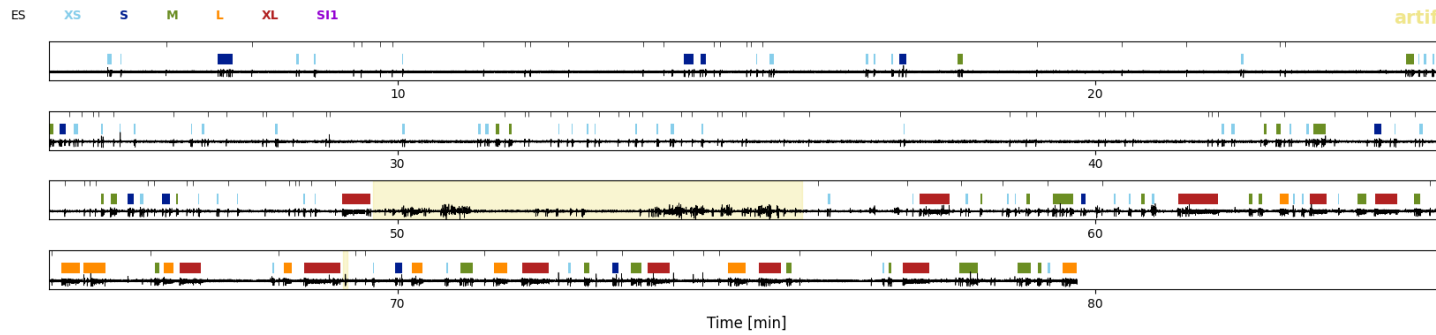


The file `my_recording__mapIds_to_ROIds.txt` translates the burst ids given in this figure to the actual burst ids in the data (in case you want to hunt down a particular burst). Everything looks nice, bursts appear to be assigned properly and according to our intuition of severity. Pay special heed to this figure when applying our algorithm to a different mouse model. If things look strange here, you *may need to use different kinds of features and/or build your own SOM*.

The figure `my_recording__traceClassified_thisSOM.png` shows the whole trace of the LFP with the burst classes (colors) and solitary spikes (black tickmarks) displayed on top. The purpose of this figure is to give you an overview



of the sequence of classified EA events. Note that here the EA is classified in t-shirt sizes to allow for a more detailed classification. The category *low-load* in the *paper\_* comprises all *XS* (bursts with fewer than five spikes which were not classified on the SOM) and *S* (low-load bursts classified through the SOM) bursts shown here. The *medium-load* bursts are category *M* and the category *high-load* comprises all *L* (=‘high-load2’) and *XL* (=‘high-load1’) bursts:



**Todo:** Links for *paper\_Figure XXX\_* and *paper\_*, in the whole section



## ACCESSING RESULTS

Here is how to use the modules and scripts that allow you to conveniently access and visualize the results stored during analyses in the *main output hdf5-file*. We provide access tools for both *Python* and *Matlab*, but note that the access through Python provides more advanced features.

### 6.1 Python

#### 6.1.1 Setting things up

Make sure that you are in the virtual environment for EAdetection (in case you decided to use one):

```
> workon EAdetectionEnv
```

For me, the command line now looks like this:

```
(EAdetectionEnv) weltgeischt@weltgeischt:~/EAdetection_tutorial$
```

Navigate into the directory of EAdetection:

```
> cd EAdetection
```

I am now in (pwd for showing the current directory):

```
> pwd
/home/weltgeischt/EAdetection_tutorial/EAdetection
```

Start python or ipython:

```
> ipython
```

From within python, we first import some useful modules:

```
In [1]: import core.ea_management as eam
In [2]: import core.helpers as hf
In [3]: import matplotlib.pyplot as plt
In [4]: import numpy as np
```

#### 6.1.2 Initialize recording object

There are two ways to initialize a recording object and thereby gain access to the data. First, you can initialize a recording object with its parameter file.

```
In [2]: paramfile = '/home/weltgeischt/EAdetection_tutorial/run_params/my_recording_
↳params.yml'
In [6]: aRec = eam.Rec(init_yamlpath=paramfile)
```

Alternatively, you could initialize it directly with the *results file* results file. If you want to plot colorful bursts, this option also requires you to give the path to the SOM.

```
In [3]: datapath = '/home/weltgeischt/EAdetection_tutorial/my_results/data/my_
↳recording/my_recording__blipSpy.h5'
In [8]: sompath = '/home/weltgeischt/EAdetection_tutorial/EAdetection/config/som.h5'
In [9]: aRec = eam.Rec(init_datapath=datapath, sompath=sompath)
```

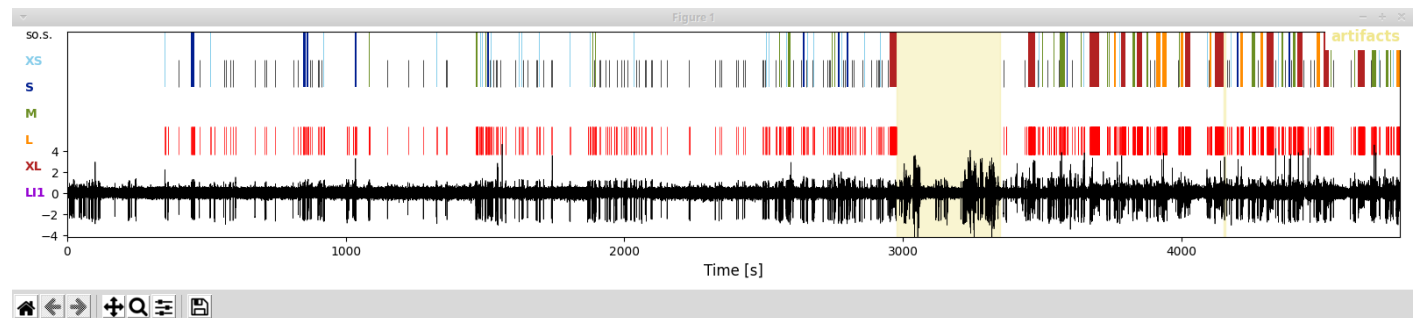
**Note:** If you dislike objects and want to extract the data directly from the hdf5 file, you are allowed to *skip ahead*.

### 6.1.3 Visualize EA sequences

Now let's plot the LFP and the EA we detected:

```
In [4]: aRec.plot(['raw', 'artifacts', 'spikes', 'singlets', 'bursts'], legendOn=True)
In [11]: plt.show()
```

In the resulting plot you see the LFP trace (bottom), all spikes detected (middle, red ticks), and the sequence of bursts (top colored according to category) and solitary spikes (black ticks). The part of the LFP that we annotated as artifact was disregarded for analyses and is shown in a yellow shade:



In this representation you see t-shirt size categorization which allow for a finer distinction than the one given in the *paper*. XS and S bursts correspond to *low-load* bursts, M bursts are *medium-load* bursts, and XL and L bursts correspond to *high-load* bursts. In this representation bursts with a spike load index = 1 (LI1) are marked in purple (technically these are also XL bursts).

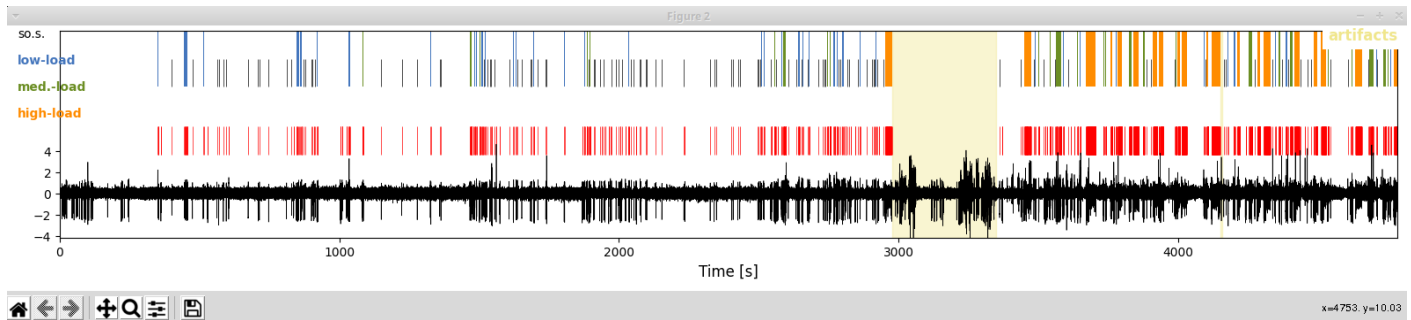
**Todo:** *paper\_* link

If you prefer the more compact, **load-based categorizations** (as in the papers) you can switch the naming scheme and coloring by executing:

```
In [5]: aRec.loadify_bursts()
```

As you can see when plotting now, colors and cluster identity are now given in the load scheme:

```
In [6]: aRec.plot(['raw', 'artifacts', 'spikes', 'singlets', 'bursts'], legendOn=True,
↳ loadlegend=True)
In [14]: plt.show()
```



### 6.1.4 Accessing the results (examples)

The recording object provides a convenient access to everything that was analyzed:

```
In [7]: aRec.dur# duration of whole recording
Out[7]: 4784.398

In [8]: aRec.offset #the time we excluded at the beginning of the recording
Out[8]: 300

In [9]: aRec.artifactTimes #start and stop times of annotated artifacts
Out[9]:
array([[2978.61, 3348.16],
       [4153.07, 4157.07]])

In [10]: aRec.durAnalyzed# duration - offset - artifact times
Out[10]: 4110.848
```

You can directly extract the **spiketimes** and e.g. calculate the overall **spike rate** from it:

```
In [11]: aRec.spiketimes #timepoints of spikes detected
Out[11]: array([ 350.472,  352.368,  353.184, ..., 4783.702, 4784.058, 4784.198])

In [12]: spikerate = len(aRec.spiketimes)/aRec.durAnalyzed

In [13]: print (spikerate, 'spikes/second')
0.4208377444264541 spikes/second
```

Our recording object contains a list of burst objects, that is the bursts that happend during the recording:

```
In [14]: aRec.bursts #list of burst objects
Out[14]:
[<core.ea_management.Burst at 0x7f79bd806c88>,
 <core.ea_management.Burst at 0x7f79b7cd2e48>,
 <core.ea_management.Burst at 0x7f79b7cd2b70>,
 .....
 <core.ea_management.Burst at 0x7f79b7cf16d8>]
```

Each **burst object** has a start and stop point in time (*.start*, *.stop*), a duration (*.dur*), a category (*.cname*), a color (*.color*, useful for plotting), a listing of its constituent spikes (*.spiketimes*), a spike load index (*.si*) and a few other properties.

```

In [15]: B = aRec.bursts[10] # for example lets pick burst number 10 and name it

In [16]: B.start
Out[16]: 919.578

In [17]: B.stop
Out[17]: 923.682

In [18]: B.roi #start and stop time
Out[18]: [919.578, 923.682]

In [19]: B.dur
Out[19]: 4.104000000000042

In [20]: B.cname
Out[20]: 'low-load'

In [21]: B.color
Out[21]: '#4476bd'

In [22]: B.spiketimes
Out[22]: array([919.578, 921.264, 923.258, 923.682])

In [23]: B.si #None because not classified on the SOM (<5 spikes), so lets pick a
↳different example ...

In [24]: aRec.bursts[33].si
Out[24]: 0.17557326391422834

```

You can **collect burst objects** of a certain type and perform computations on them. Let's e.g. collect high-load bursts and calculate their rate:

```

In [25]: hl_bursts = [B for B in aRec.bursts if B.cname=='high-load'] #list of burst
↳objects
In [34]: N_hl = len(hl_bursts) #number of high-load bursts
In [35]: rate_hl = N_hl/aRec.durAnalyzed
In [36]: print ('High-load rate (/min): ', rate_hl*60.) #*60 to get /min
High-load rate (/min): 0.2919105741686387

```

Similarly we can also calculate the absolute **time spent in high-load bursts** and the fraction of time spent in high-load bursts:

```

In [26]: dur_hl = np.sum([B.dur for B in hl_bursts])

In [27]: print ('Total hl-duration (min): ', dur_hl/60.) #/ 60 to get min
Total hl-duration (min): 5.909600000000008

In [28]: print ('Relative time in high-loads: ', dur_hl/aRec.durAnalyzed) #/ 60 to get
↳min
Relative time in high-loads: 0.08625373645534948

```

You can also address **EA-free snippets** as objects, in a way very similar to bursts:

```

In [29]: aRec.freesnips #list of EA-free snippet objects in the recording
Out[29]:
[<core.ea_management.Period at 0x7f79b410cda0>,
 <core.ea_management.Period at 0x7f79b410cdd8>,

```

(continues on next page)

(continued from previous page)

```

<core.ea_management.Period at 0x7f79b410ccc0>,
.....
<core.ea_management.Period at 0x7f79b4114c18>]

In [30]: aFree = aRec.freesnips[10] #10th EA-free snippet

In [31]: aFree.roi #start and stop time
Out[31]: [586.5360000000001, 593.752]

In [32]: aFree.dur #duration
Out[32]: 7.2159999999998945

```

If you want to concentrate your analysis on a particular **cutout** of data you can do so too. The cutout can then be analysed and visualized in the same way as the whole recording

```

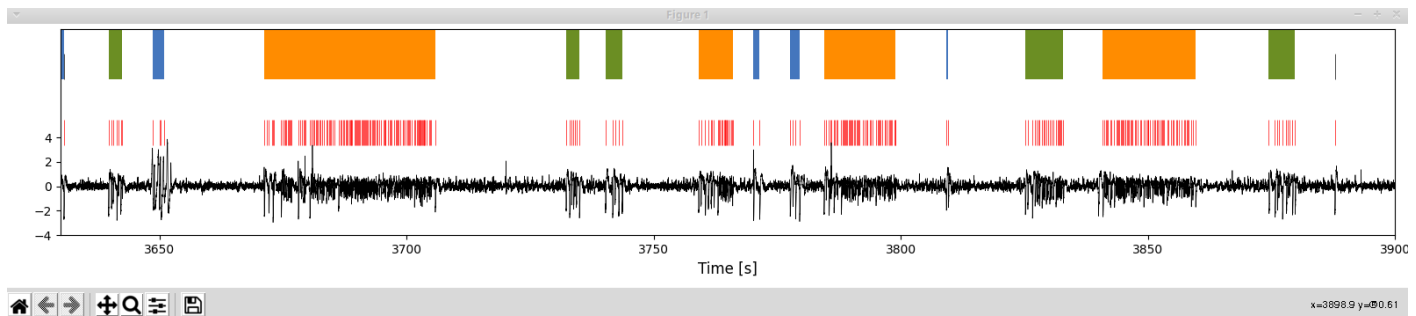
In [33]: cutRec = eam.EAPeriod(3630.,3900.,parentobj=aRec) #cut out from recording_
↪object

In [34]: cutRec.bursts[10].dur #duration of the 10th burst in the cutout
Out[34]: 0.43039182282791444

In [35]: print ('Cutout spikerate: ',len(cutRec.spiketimes)/cutRec.dur,'spikes/second'
↪')
Cutout spikerate: 1.1814814814814816 spikes/second

In [36]: cutRec.plot(['raw','artifacts','spikes','singlets','bursts']) #plot cutout
In [48]: plt.show()

```



## 6.1.5 Directly extract data from hdf5

You can also directly extract the data directly from the *main results file*. If you want to operate with data-arrays directly this might be a more efficient solution. In the following, *datapath* refers to the path of the main results file.

```

In [37]: datapath = '/home/weltgeischt/EAdetection_tutorial/my_results/data/my_
↪recording/my_recording__blipSpy.h5'

In [38]: results = hf.open_hdf5(datapath) #this opens the main results file

In [39]: print (results.keys()) # gives you the sub-fields of the resultsfile
dict_keys(['burstclasses', 'dischargedict_cleaned', 'dischargedict_raw', 'raw_data'])

In [40]: burstdata = results['burstclasses']['data']

```

(continues on next page)

(continued from previous page)

```
In [41]: print (burstdata.keys()) # whats in the burst data?
dict_keys(['params', 'values'])

In [42]: burstdata['params'] #these are whats in the 'values'
Out[42]: ['key_id', 'start', 'stop', 'clustid', 'seizidx', 'bmu']

In [43]: burstdata['values'].shape # (number of bursts) x (number of parameters)
Out[43]: (125, 6)

In [44]: starttimes = burstdata['values'][:,burstdata['params'].index('start')] #
↳starttimes of all bursts

In [45]: starttimes
Out[45]:
array([ 445.338      ,  846.258      ,  860.334      , 1031.31      ,
        1081.86      , 1467.002      , 1500.36      , 1509.08      ,
        .....
        4346.366      , 4454.184      , 4617.296      , 4710.278      ,
        4759.026      ])
```

As you have seen, you can get out a lot from the data. Happy hacking and exploring!

## 6.2 Matlab

Accessing the results in matlab is very similar to accessing them in python (e.g. `aRec.spiketimes` for getting the spiketrain, see [previous section](#)). Have a look at the example in `EAdetection/examples/matlab/use_matlab.m`. As some of the fancier features (e.g. plotting) are not developed yet, you are very welcome to expand the reader for matlab access at `EAdetection/examples/matlab/READEA.m`.



## ACCESSING METADATA AND DIAGNOSTICS METRICS

You can render metadata and diagnostics metrics automatically in human readable form using odML or yaml. A single metadata and/or diagnostics file is created for a specific recording.

**Metadata** includes the parameters used for analyses and details about who analyzed the data when and the paths to the figures created during the different analysis steps. Therefore metadata is sub-section into the fields:

- *EdDetection*, for spectral and amplitude based spike detection
- *SpikeSorting*, for sorting of spikes to remove false positives
- *BurstClassification*, for burst detection and classification

**Diagnostics metrics** include metrics we commonly use to address the severity of epilepsy. These include:

- *rate(spikes)*, the rate of spikes detected during the recording (/s)
- *rate(high-load)*, the rate of high-load bursts (/min)
- *tfrac(high-load)*, the fraction of time spent in high-load bursts
- *tfrac(bursts)*, the fraction of time spent in bursts of any kind
- *tfrac(free)*, the fraction of time spent outside of bursts
- *burstiness*,  $(\text{std}(\text{ISI}) - \text{mean}(\text{ISI})) / (\text{std}(\text{ISI}) + \text{mean}(\text{ISI}))$ ; ISI being interspike interval, see Goh and Barabási (2008)

### 7.1 General

Before you start, make sure that you are in the virtual environment for EAdetection if you have *set up one*:

```
> workon EAdetectionEnv
```

... and make sure you are in the folder of EAdetection. My command-line would look like this:

```
(EAdetectionEnv) weltgeischt@weltgeischt:~/EAdetection_tutorial/EAdetection$
```

The general **command structure** for rendering metadata and diagnostics in both formats looks like this:

```
> python runthrough/metadata_diagnostics.py --format=<format> --<options> <path/to/paramfile.yml>
```

<format> can be either *yaml* or *odml*, depending on your taste of filetype

<options> can be:

- *d* for rendering diagnostic metrics

- *m* for rendering metadata metrics
- *md* for rendering metadata and diagnostic metrics in a single file

## 7.2 odML

If you want to know more about odML, have a look at [this](#). To be able to render something in odML format, you need to have odml installed. You can do so using pip:

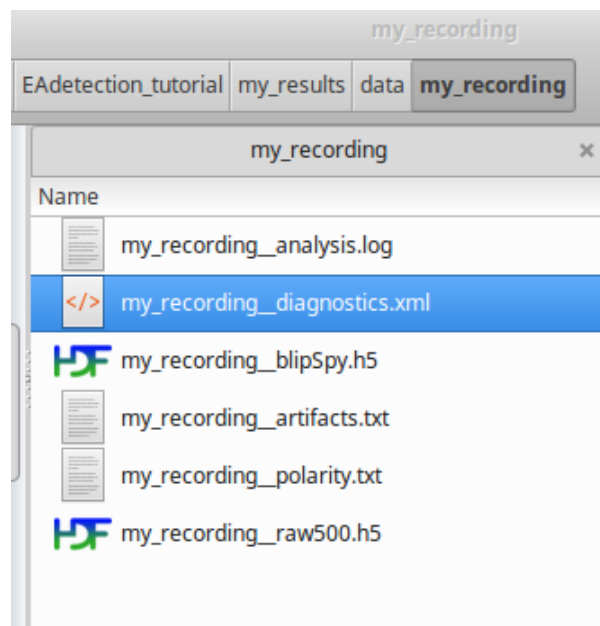
```
> pip3 install odml
```

### 7.2.1 Example 1

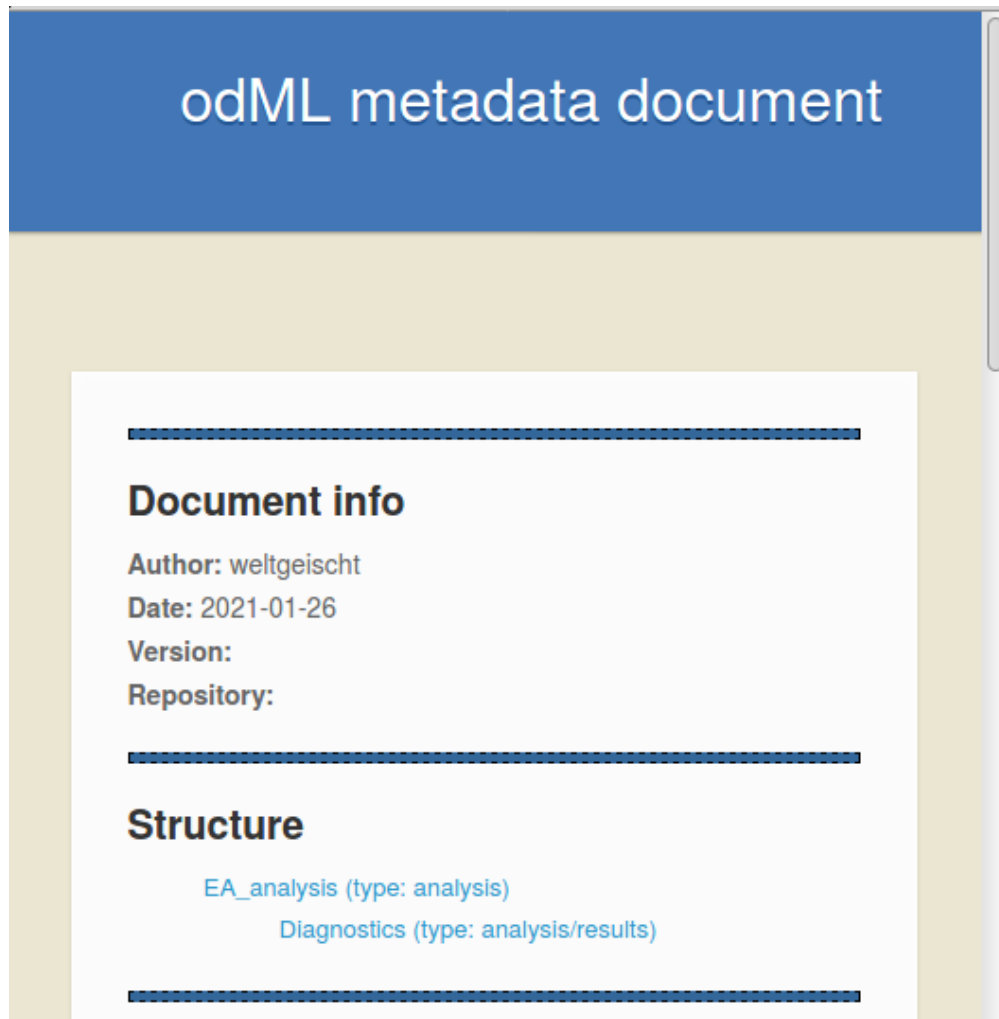
Now, let's render **diagnostic metrics** of our example recording in **odML format**:

```
> python runthrough/metadata_diagnostics.py --format=odml -d /home/weltgeischt/  
↪EAdetection_tutorial/run_params/my_recording_params.yml
```

This generates a file *my\_recording\_diagnostics.xml* at *the folder we had assigned to store the data*, in my case */home/weltgeischt/EAdetection\_tutorial/my\_results/data/my\_recording*:



Open the file in your browser (e.g. right click on the file and select *Open With* > *Firefox*). It should look like this:



If you now click at the blue *Diagnostics* tab, it will bring you to all the beautiful diagnostic metrics:

odML metadata document				
Include:				
Definition:				
Name	Value	Unit	Type	Uncertain
rate(spikes)	0.4208377444264541	1/s	float	
rate(high-load)	0.2919105741686387	1/min	float	
tfrac(high-load)	0.08625373645534948		float	
tfrac(bursts)	0.16576813955858705		float	
tfrac(free)	0.6610617505273513		float	

### 7.2.2 Example 2

Now let's render **metadata** in **odML format**. You type the same as in *the previous example*, only that you now use option `-m` instead of `-d`:

```
> python runthrough/metadata_diagnostics.py --format=odml -m /home/weltgeischt/
↳ EAdetection_tutorial/run_params/my_recording_params.yml
```

This generates a file *my\_recording\_\_metadata.xml* in the results folder. Open it in your browser. As you see it displays a section for each analysis step, with subsection called *...\_methods* and *...\_info*. In the former you find the parameters used for running the analyses, in the latter you have links to figures generated and who ran the procedure when. Clicking on the respective subsection will directly bring you to its content.

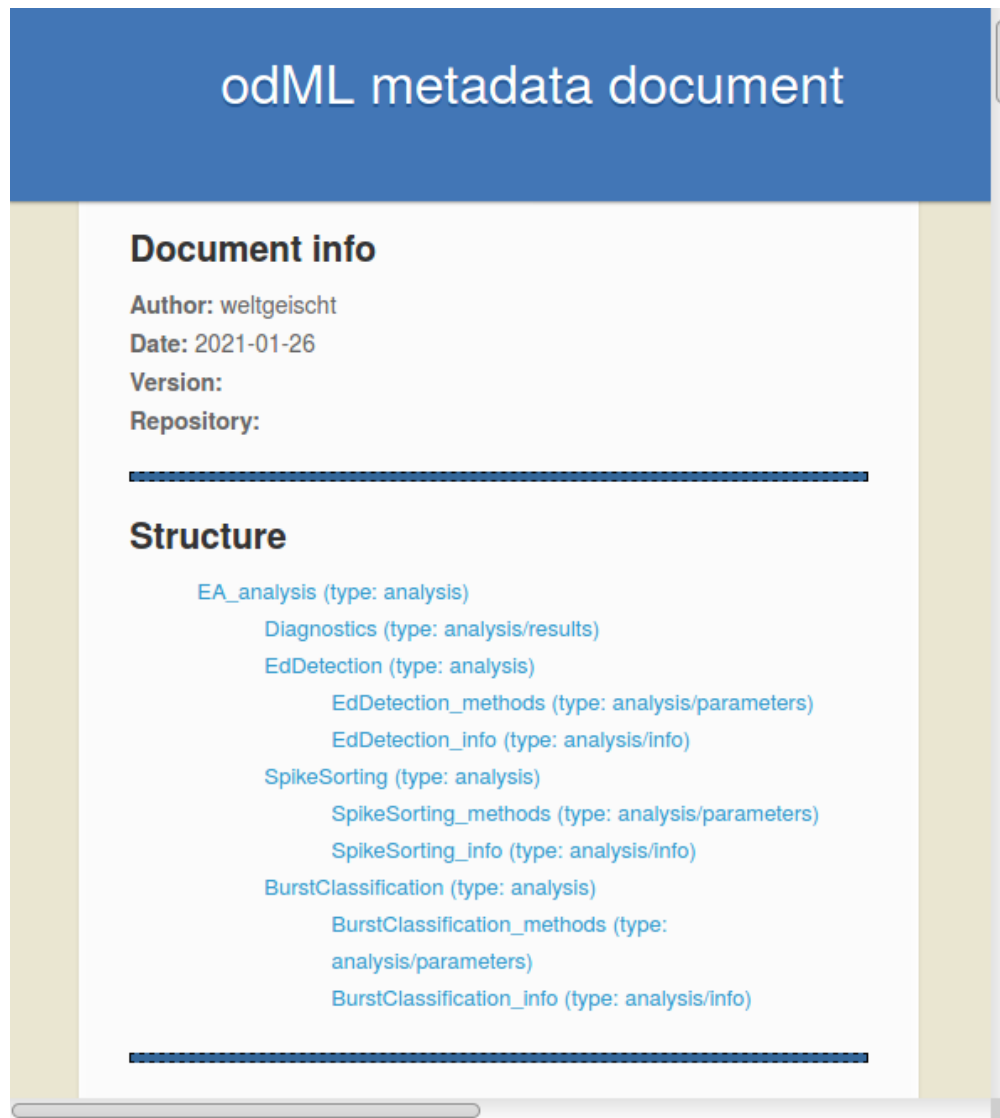


### 7.2.3 Example 3

To produce a single file that contains both **metadata and diagnostic metrics** in **odML format**, use option *-md*:

```
> python runthrough/metadata_diagnostics.py --format=odml -md /home/weltgeischt/
↳ EAdetection_tutorial/run_params/my_recording_params.yml
```

The file generated is called *my\_recording\_\_metadata\_diagnostics.xml* and also resides in the main results folder. As you see, now the file incorporates both, the diagnostic metrics from [Example 1](#), and the metadata from [Example 2](#):



## 7.3 YAML

For the output format **yaml** it works exactly the same as described for odML in the previous section, i.e. you can use option **-d**, **-m** or **-md**. For example, to get the diagnostic metrics you type:

```
> python runthrough/metadata_diagnostics.py --format=yaml -d /home/weltgeischt/  
↳EAdetection_tutorial/run_params/my_recording_params.yml
```

This generates a file `my_recording__diagnostics.yml` in the results folder. Just open this file with some basic text editor (by clicking on it):

```

/home/weltgeischt/EAdetection_tutorial/my_results/data/my_recording/my_recording_diagnositcs.yml - Mousepad
File Edit Search View Document Help
1 Diagnostics:
2   rate(spikes):
3     value: 0.4208377444264541
4     unit: 1/s
5     definition: rate of epileptiform spikes
6   rate(high-load):
7     value: 0.2919105741686387
8     unit: 1/min
9     definition: rate of high-load bursts
10  tfrac(high-load):
11    value: 0.08625373645534948
12    definition: fraction of time spent in high-load bursts
13  tfrac(bursts):
14    value: 0.16576813955858705
15    definition: fraction of time spent in bursts (of any kind)
16  tfrac(free):
17    value: 0.6610617505273513
18    definition: fraction of time spent outside of bursts
19  burstiness:
20    value: 0.5487977010354498
21    definition: burstiness (cp. Goh & Barabasi 2008) of epileptiform spikes
22  durAnalyzed:
23    value: 68.51413333333333
24    unit: min
25    definition: duration of recording analyzed (dur-offset-artifacttimes)
26  durArtifacts:
27    value: 6.225833333333329
28    unit: min
29    definition: total duration of artifacts in recording
30

```

If you are not in the mood of *accessing the results through matlab or python*, you can conveniently read some essential diagnostic metrics from this file.





## TODOLIST

---

**Todo:** *paper\_ link*

---

(The [original entry](#) is located in /home/weltgeischt/workspace/EAdet\_doc/doc/access\_results.rst, line 103.)

---

**Todo:** *Link to the paper\_*

---

(The [original entry](#) is located in /home/weltgeischt/workspace/EAdet\_doc/doc/LFP\_to\_bursts.rst, line 57.)

---

**Todo:** *Links for paper\_Figure XXX\_ and paper\_, in the whole section*

---

(The [original entry](#) is located in /home/weltgeischt/workspace/EAdet\_doc/doc/output.rst, line 201.)

---

**Todo:** *Make a link to download this example .smr file\_.*

---

(The [original entry](#) is located in /home/weltgeischt/workspace/EAdet\_doc/doc/setting\_parameters.rst, line 134.)

---

**Todo:**

- 1) Make a link to Heining et al. 2021 once the paper is out.
  - 2) Make a link to the tutorial data file.
  - 3) Make a link to the pdf.
- 

(The [original entry](#) is located in /home/weltgeischt/workspace/EAdet\_doc/doc/index.rst, line 42.)