Estimation of History Dependence in Neural Spiking Data

The `history dependence estimator` tool provides a set of routines that facilitate the estimation of history dependence in neural spiking data, using estimators based on information-theoretical measures, as presented in (Rudelt et al. in prep.). There, one can read more about the methods, both their derivation and practical aspects of their computation. This guide offers a reference for the usage of the tool.

The tool takes spike data as input and produces an image as output (and the data required to plot the image, in case the user prefers to create the graphs herself). The image includes visualizations of the history dependence, of the auto mutual informal, of the neuron's activity, as well as a table with results.

This guide is structured as follows: First, a quick-start section is meant to help test whether the tool works on the user's computer. For that purpose, a file containing sample spike times is provided. Afterwards, the sections are dedicated to the steps required to perform the analysis on one's own data: Section 2 explains how to set up the analysis; Sec. 3 outlines the internal routines involved in the analysis; finally, Sec. 4 presents a sample analysis, in which the estimator is applied to openly available data.

Contents

1. Quick Start

1. Download the code by cloning the repository from GitHub

```
$ git clone https://github.com/Priesemann-Group/hdestimator.git
```

2. Change into the tool's directory

```
$ cd hdestimator
```

3. Test that all dependencies for the tool are met, and that it is ready for use, by running:

```
$ python3 estimate.py --help
```

   (Installation instructions can be found in the project's README file.) You should now see a help text for the tool printed on screen.

4. A file containing sample spike data is provided in the `sample_data` folder. Run the analysis on that file, using the `test.yaml` settings file, which is also provided. Save the output to `sample_analysis.pdf`. This should run for less than a minute.

```
$ python3 estimate.py sample_data/spike_times.dat \
--settings-file settings/test.yaml \
--output sample_analysis.pdf
```

If the analysis is taking too long, check the output for a warning message that the Cython module couldn't be loaded. In that case, please read Appendix B for help on how to set up the Cython module.

5. When the analysis is done, open the resulting image, `sample_analysis.pdf`. It should look like that in Fig. 1.

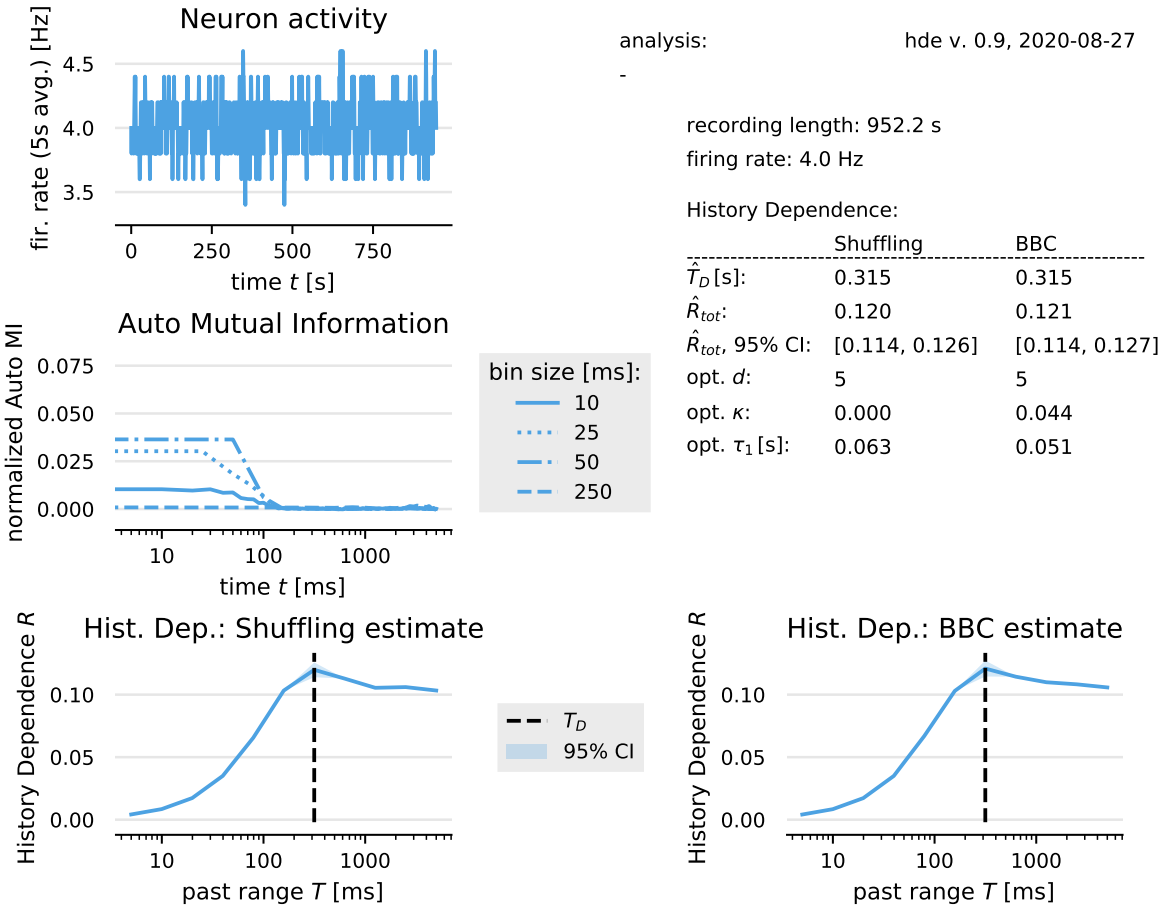

Figure 1: Output of the tool applied to the sample data and using the `test.yaml` settings file. The settings define that only the shuffling estimator should be run, therefore the plot for the BBC estimate is empty. For guidance on how to interpret the results, please see (Rudelt et al. in prep.).

2. Setup the Analysis

An analysis with the history dependence estimator is performed by calling the main script, `estimate.py`, and providing it with an array of spike times (cf Sec. 2.1). The outcome of the analysis is determined not only by the data, but also by some settings (cf Sec. 2.2) and by the part of the analysis that is performed (per default, all parts implemented in the tool are run, cf Sec. 3). This section addresses the setup of the analysis, ie how to provide the data and how to tweak the analysis by changing the settings. The section afterwards is concerned with the parts that make up the analysis.

2.1. Import Data

First, the data needs be presented in a format readable by the script. The `estimate.py` script has implemented two methods to import data. It can take a file, in which one spike time (in seconds) is written per line. A sample file in this format can be found in the `sample_data` folder, which can be passed as follows:

```
$ python3 estimate.py sample_data/spike_times.dat
```

It can also be provided with a file in the `hdf5` format, by using the `--hdf5-dataset` (or `-h5` for short) argument followed by the path to the dataset within the `hdf5` file. Here, too, spike times should be provided in units of seconds. Again, a sample file can be found in `sample_data`, which can be analysed with the following command:

```
$ python3 estimate.py sample_data/spike_times.h5 -h5 spt
```

Data sources can be diverse and might be stored in a way other than those mentioned above. For users who do not want to re-store their data in new files, we recommend to write a script that reads their data and outputs it in the required format. Let's say you did so with a script named `read_and_output_data.py`. You can then redirect its output to the history dependence estimator tool as follows:

```
$ python3 read_and_output_data.py | python3 estimate.py /dev/stdin
```

(The vertical bar '|' redirects ("pipes") the output from the program on the left hand side to the input on the right hand side; `/dev/stdin` is the name of a special file in GNU/ Linux operating systems that here allows to pass such an input as if were a file.) Make sure that your script is in the same folder as the tool, or else provide appropriate paths to the programs. An example on how this is applied to real data is presented in Sec. 4.

2.1.1. Data Requirements

For the analysis to be meaningful and comparable across different recordings, the data should meet a few requirements. First and foremost, the data needs to be stationary; ie it should not be explicitly time-dependent. One would obtain non-stationary data for example in a trial-based experiment, for which one expects a systematic dependency of the neural activity on the time within a trial. Second, recordings should be sufficiently long; as a rule of thumb we recommend a minimum length of 5 minutes. Last, if one compares the history dependence for recordings of different systems or brain areas, the length of the recordings should be fairly similar. A rationale for this can be given as follows: The shorter the recording, both the higher the variance of the estimate and the lower the dimension that an embedding can attain without resulting in biased estimates. These two factors tend to lead to a lower history dependence and a lower temporal depth estimate (cf Sec. 3.1). In order for this effect to be similar across the analysed systems or brain areas, the recording lengths should thus be similar.

2.1.2. Non-contiguous Data

The tool also allows jointly to analyse several non-contiguous spike trains. This might be useful eg when the data originate from trial-based experiments and one is interested in the response to a certain stimulus. In this case each recording related to this stimulus taken individually might be too short. If the recordings are taken together however, data might be sufficient for a meaningful analysis. To analyse several recordings jointly, either provide several files to the tool, eg as follows:

```
$ python3 estimate.py spike_times_part_1.dat spike_times_part_2.dat
```

Provide several `hdf5` files all with the same dataset name:

```
$ python3 estimate.py spike_times_part_1.h5 spike_times_part_2.h5 \
  -h5 spt
```

Provide several `hdf5` files each with an individual dataset name:

```
$ python3 estimate.py spike_times_part_1.h5 spike_times_part_2.h5 \
  -h5 spt1 spt2
```

Or provide an `hdf5` file with several dataset names:

```
$ python3 estimate.py spike_times.h5 -h5 spt1 spt2
```

If a script is used to pipe the data to the tool as described at the end of Sec. 2.1, divide the parts by printing a line with '----------' (10 dashes, without quotation marks) as its content.

2.2. Settings

The analysis can be tweaked with several parameters. The most important ones certainly are those that define how the spike times are embedded, ie turned into vectors of zeros and ones. Other settings define eg how many bootstrap samples should be drawn to compute confidence intervals around the estimates, but also more trivial things such as which color and font size to use for the resulting image. A table detailing the available options can be found in Appendix A.

The settings and parameters for the analysis are read from the `default.yaml` settings file. (Information on the syntax of `yaml` files can be found online, but in-depth knowledge should not be required to adapt the tool's settings file.) This file is provided with the tool and for most use cases we recommend leaving it untouched. If you want to change any setting, create a new file and include only those settings you want to change with respect to the `default.yaml` settings file. As in the quick start guide, you pass the new settings with the `--settings-file` argument, eg as follows:

```
$ python3 estimate.py sample_data/spike_times.dat \
  --settings-file settings/custom.yaml
```

All the settings that you did not define in the `custom.yaml` settings file are read from the default settings file. If you accidentally changed something in the default settings file and want to revert it, simply delete the file and it will be re-created into its original state with the next run of the tool. (You can also use `git checkout -- settings/default.yaml` to revert it.)

We provide two sample custom settings files, called `test.yaml` and `exhaustive.yaml`. The `test.yaml` settings are meant to be used for a quick run to either test the tool itself or eg its application to a new data set. The `exhaustive.yaml` settings could be used in addition to the `default.yaml` settings, if the user wants a more thorough analysis that includes higher-dimension embeddings and both estimation methods. These files also exemplify how custom settings files should look like: only the settings that deviate from the default one should be set. For most cases, the default settings should be used: these provide a good deal of performance without compromising on adequacy.

For a few settings, there is also the possibility to modify the settings by passing them via the command line, eg modify the label for your analysis by passing:

```
$ python3 estimate.py sample_data/spike_times.dat \
  --settings-file settings/custom.yaml \
  --label ‘‘sample analysis’’
```

Some settings can only be passed via the command line. Eg if you only want to plot the results (assuming you previously ran the analysis and saved the analysis to file using the `--persistent` argument, `-p` for short, cf Sec 2.3):

```
$ python3 estimate.py sample_data/spike_times.dat \
  --settings-file settings/custom.yaml \
  --label ‘‘sample analysis’’ -p --task plots
```

If you want help with the command line parameters, please run

```
$ python3 estimate.py -h
```

The tool prioritizes passed settings as follows: if provided, a command-line setting is used; else the one from the custom settings file is used; and if it is not found there either, the one from the default settings file is loaded.

2.3. Storing the Results

If, as in the Quick Start in Sec. 1, you use the `test.yaml` settings, the tool shows you a visualization of the results of the analysis —either on screen or saved to a file if the `--output` argument is used—, as well as some statistics that are printed to the terminal. The data created in the process of the analysis are deleted when the tool is closed. It is also possible, and this is the default behaviour of the tool, to store the data to files that are not automatically deleted; here, we outline what happens behind the scenes.

The tool takes a `persistent_analysis` setting, which determines both whether results of the analysis are stored, and whether the tool should search for existing results in order to avoid redundant computations. For that purpose, either set the `persistent_analysis` setting to `True` or pass the `--persistent` argument. Per default, data is stored in the `analysis` directory within the parent directory of the tool. If you want to select a different one, please set the `ANALYSIS_DIR` setting accordingly. Note that the tool only searches for existing analyses within that directory.

The `ANALYSIS_DIR` setting defines a parent directory. Within there, each neuron (ie each spike times file) gets a dedicated directory, in which a few files are created. One is created to identify the spike times file that is linked to the analysis. Another, called `analysis_file.h5`, contains detailed results stored in a structured way. Finally, the CSV files (cf Sec. 3.4), and output image (cf Sec. 3.5) are stored there too.

You might want to start with a fast run, in which you try out only a few parameters. Then, depending on the result, you might want to run the analysis again with a more broad set of parameters. If the analysis is stored to file, the tool automatically detects an existing analysis and does not re-compute the estimates for the parameters for which an analysis was found.

Furthermore, if you want to use the CSV files (cf Sec. 3.4) to to perform additional steps on the data or visualize the data in your own way, you need to tell the tool to store them by having the `persistent_analysis` set to `True`.

Settings

`ANALYSIS_DIR` sets the path to a parent directory for persistent storage, under which a directory is created per analysis of a spiking times file.

`persistent_analysis` sets whether results produced by the analysis should be stored persistently, ie to the hard drive. This includes raw results (stored in `analysis_file.h5`), summary statistics and sufficient data to generate the output image (stored in CSV files, cf Sec. 3.4), as well as the output image itself. This parameter also determines whether stored data should be read from file. Its value can be either `True` or `False`.

3. Perform the Analysis

Once you have your data ready (cf Sec. 2.1) and set up the desired settings (cf Sec. 2.2 and Appendix A), you are ready to perform the analysis. The analysis can be split up into a number of tasks, which you can run individually; or you can tell the tool to run the full analysis (which is the default when no argument is provided, as in the Quick Start). Available tasks are

- `history-dependence`: Estimate the history dependence in the spiking data (the main analysis).
- `confidence-intervals`: Compute confidence intervals for the history dependence estimates.
- `auto-mi`: Estimate the auto mutual information in the spiking data (a related measure).
- `csv-files`: Save the results as comma separated values (CSV) files.
- `plots`: Produce plots, a visual representation of the analysis.
- `full-analysis`: Perform all of the tasks above.

Running the full analysis should be the way to go in most cases. If you nonetheless want to run them individually, you should proceed as follows: first run the desired analyses (you always have to start with `history-dependence`, the other ones are optional). Then create the `csv-files` and visualize the results by producing the `plots`.

The task to be performed is passed with the `--task` argument via the command line. The default task is `full-analysis`. If you want to run any other task other than the `full-analysis`, you should make sure that the `persistent_analysis` setting is set to `True` (or pass the `--persistent` argument, `-p` for short); see Sec. 2.3 for details. Please note that it is not required to write out the full name of the task, as long as the first letters uniquely identify the task.

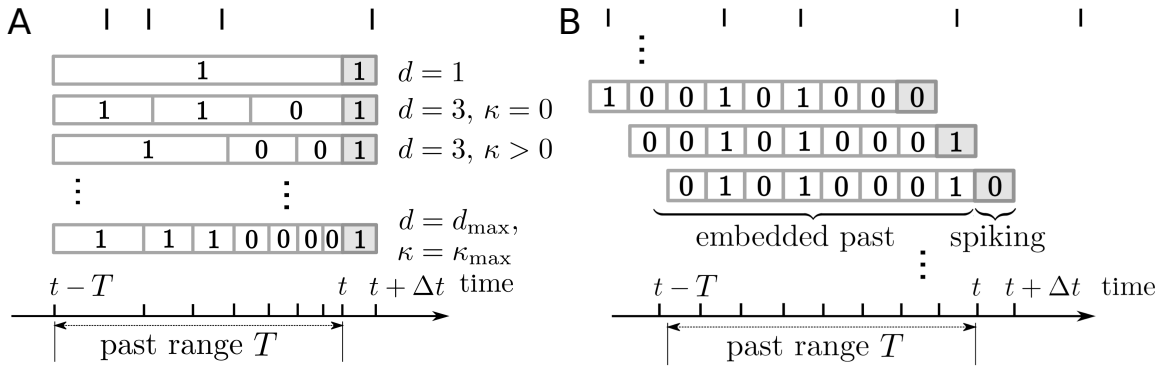

Figure 2: A window of length T consisting of d bins is used to embed the spike time data into vectors of length d, with elements 0 and 1, called symbols. **A**: An exhaustive search is performed to find embeddings that optimally capture the history dependence. For that purpose, the past range T, the number of bins d and the scaling exponent κ are varied. **B**: For each combination of embedding parameters, the window is slid through the spike time data in steps of size Δt and the resulting symbols are counted. Image taken from (Rudelt et al. in prep.).

3.1. History Dependence

To run the `history-dependence` task on the sample data, please use

```
$ python3 estimate.py sample_data/spike_times.dat -t hist
```

Note that it is sufficient to write `hist` here, because it uniquely identifies the task.

The history dependence estimation task constitutes the main part of the present analysis. Several steps are performed in its course. At this stage, the data available are a list of spike times. In the first step, a form of discretization is applied to the data. To do so, a window is defined with which the spike times are embedded into a vector of zeros and ones. We will also refer to this window as the *embedding* and to the vectors as *symbols*. This is illustrated in Fig. 2.

The embedding is a window of length T that is divided into d bins. The window is slid through the spike train in steps of size Δt, in each step finding a symbol, the occurrence of which is counted. The bin sizes τ_j can be all equal ("uniform embedding") or increase exponentially ("exponential embedding"). Using the exponential embedding results in a higher precision close to the response x_t in the interval $[t,\, t + \Delta t]$ and a lower precision at times further in the past. The scaling exponent κ defines how the size of the bins τ_j should change, according to $\tau_\mathrm{j} = \tau_1 10^{\kappa(\mathrm{j}-1)}$, where τ_1 is the bin closest to the response (the activity at the next time step) and τ_d is the bin furthest in the past. Please note that the value for τ_1 is uniquely determined for fixed T, d and κ, and thus needs not and cannot be set explicitly by the user.

First, the tool estimates the history dependence for each embedding defined by the user. The tool provides two method for this estimation, one called BBC Estimator

(because it uses a Bayesian Bias Criterion to guarantee an unbiased estimate) and one called Shuffling Estimator (because it shuffles information to correct a possible bias). Users interested in how the history dependence is estimated are referred to (Rudelt et al. in prep.). Then, for each T only the embedding for which the history dependence R is maximal is taken into account. The curves at the bottom plots of Fig. 1 show these maximal values of R at each T.

In this toolbox, we quantify history dependence both in terms of the total history dependence R_{tot}, and the temporal depth T_{D}. The temporal depth is the past range T for which the estimated R saturates within errorbars, $T_{\mathrm{D}} = \min\{T \mid R(T) > R_{\max} - \mathrm{se}(R_{\max})\}$ (cf Sec. 3.2 on the estimation of errorbars and Rudelt et al. in prep. for details). The total history dependence is the history dependence for the given temporal depth, $R_{\mathrm{tot}} = R(T_{\mathrm{D}})$. In the plot, these values can be located through the dashed vertical bar and are provided in the results table.

In order to avoid overfitting when optimizing the estimate for different embeddings, it is possible to apply cross-validation to the analysis. If `cross_validated_optimization` is set to `True`, the embedding optimization is performed on the first half of the data, while R_{tot} is obtained by applying the resulting embedding to the other half of the data.

Settings

`embedding_step_size` sets the step size Δt (in seconds) with which the window is slid through the data.

`embedding_past_range_set` sets the values for T, the past range (in seconds) to be used for embeddings and should be an array of floating-point values.

`embedding_number_of_bins_set` sets the values for d, the number of bins in the embedding, and should be an array of integer values.

`embedding_scaling_exponent_set` sets the values for κ, the scaling exponent for the bins in the embedding, and can be defined in two ways:

- One is to set `embedding_scaling_exponent_set` to an array of floating-point values, the values for κ.
- The other, used in the default settings, is to set it not to an array but to a python-dictionary, with entries `number_of_scalings`, `min_first_bin_size` and `min_step_for_scaling`. If these are set, up to `number_of_scalings` are used, where the smallest value of κ is always 0, corresponding to a uniform embedding. The largest value of κ is such, that the resulting value for τ_1 is equal to `min_first_bin_size`. Values in between the smallest and largest κ are chosen to be linearly spaced. However, very small differences in κ, especially when T is relatively small, might not make a big difference in the resulting embedded symbols. Thus, to save computation time, the effective number of values for κ that are used

for the embeddings is reduced, such that the resulting values are spaced at least `min_step_for_scaling` apart.

`estimation_method` sets the method to be used to estimate the history dependence, one of `bbc`, `shuffling` or `all`.

`bbc_tolerance` sets the tolerance for the Bayesian Bias Criterion (cf Rudelt et al. in prep.). and influences which embeddings are discarded from the analysis.

`number_of_bootstraps_R_max` sets the number of bootstrap re-shuffles (cf Sec. 3.2) that should be used to compute $\mathrm{se}(R_{\mathrm{max}})$ which is used to locate T_{D} and thus obtain R_{tot}. If `persistent_analysis` is set to `True`, previous re-shuffles are stored. Thus if you initially bootstrapped 50 times and then set the parameter to 100, only 50 additional estimates are computed.

`cross_validated_optimization` sets whether cross validation should be used for the embedding optimization, to avoid overfitting.

3.2. Confidence Intervals

To run the `confidence-intervals` task on the sample data, please use

```
$ python3 estimate.py sample_data/spike_times.dat -t conf
```

Each estimate of the history-dependence carries some degree of uncertainty. To quantify this uncertainty, the tool includes a task to compute bootstrap confidence intervals (cf Efron and Tibshirani 1994). We apply a blocks of blocks bootstrap (Davison and Hinkley 1997), for which we draw blocks of symbols (each symbol itself represents a block in the time series). The `block_length_l` can be defined by the user, but we recommend to leave it at `None`, in which case a heuristic is applied to determine the length of the block (cf Rudelt et al. in prep.). In all inspections, we observed the distribution of bootstrap replications to be normally distributed, so per default their standard deviation is used to obtain 95% confidence intervals. This behaviour can be changed by setting `bootstrap_CI_use_sd` to `False`, in which case confidence intervals are defined via percentiles of the replications, as defined via `bootstrap_CI_percentile_lo` and `bootstrap_CI_percentile_hi`. Note that this requires significantly more bootstrap replications to be accurate.

In the `history-dependence` task (cf Sec. 3.1), bootstrap replications are computed to obtain the standard error $\mathrm{se}(R_{\mathrm{max}})$ of R_{max}, which is used to locate T_{D} and thus obtain R_{tot}. In this task, bootstrap replications are computed to obtain a confidence interval for $\mathrm{se}(R_{\mathrm{tot}})$.

The parameter `number_of_bootstraps_R_tot` sets the number of bootstrap replications to be used for R_{tot}. It is also possible to compute confidence intervals for the

embeddings for other values of T, but these are not required for the main analysis, and thus the parameter is called `number_of_bootstraps_nonessential` and its default value is 0.

The confidence intervals (including the nonessential ones) can be indicators of whether there are sufficient data available for the analysis to be meaningful. If the recording length is short, symbols may occur only rarely, potentially yielding highly variable estimates depending on the bootstrap re-shuffle. This, however, also depends on the dimension d of the embedding (the number of bins). As a rule of thumb, if the confidence intervals do not allow for enough confidence about the estimate, either record longer data or use a smaller number of bins. In practice this might not be needed, as too high a number of bins for too short a recording produces a biased estimate, which is rejected by the BBC estimator (if it does not, the BBC tolerance might be set too high; we recommend a value of 0.05) and corrected for by the Shuffling estimator, respectively.

Settings

`number_of_bootstraps_R_tot` sets the number of bootstrap re-shuffles that should be used for the confidence interval of R_{tot}. If `persistent_analysis` is set to `True`, previous re-shuffles are stored. Thus if you initially bootstrapped 50 times and then set the parameter to 100, only 50 additional estimates are computed.

`number_of_bootstraps_nonessential` sets the number of bootstrap re-shuffles that should be used to estimate the confidence intervals for embeddings other than the optimal one. As above, if `persistent_analysis` is set to `True`, previous re-shuffles are stored.

`block_length_l` sets the number of symbols that should be drawn in each block for bootstrap resampling. If it is set to `None` (recommended), the length is automatically chosen, based on heuristics.

`bootstrap_CI_use_sd` sets whether confidence intervals should be computed based on the standard deviation of the bootstrap replications.

`bootstrap_CI_percentile_lo` sets the lower percentile for the confidence interval. This has no effect if `bootstrap_CI_use_sd` is set to `True`.

`bootstrap_CI_percentile_hi` sets the upper percentile for the confidence interval. This has no effect if `bootstrap_CI_use_sd` is set to `True`.

`estimation_method` sets the method to be used to estimate the history dependence, one of `bbc`, `shuffling` or `all`.

`bbc_tolerance` sets the tolerance for the Bayesian Bias Criterion (cf Rudelt et al. in prep.). and influences which embeddings are discarded from the analysis.

`cross_validated_optimization` sets whether cross validation should be used for the embedding optimization, to avoid overfitting.

3.3. Auto Mutual Information

To run the `auto-mi` task on the sample data, please use

```
$ python3 estimate.py sample_data/spike_times.dat -t auto
```

The auto mutual information is a measure closely related to the history dependence; they both are based on information theory. The auto mutual information measures the mutual information of only one bin and a response in the future (so in the history dependence framework, we would have $d = 1$). Unlike for the history dependence, the response is not necessarily in the imminent future of the past bin, but there might be a delay in between. In fact, the analysis here consists of a systematic increase of this delay, as well as a systematic variation of the size of the bins. The delay is increased in steps of the size of the bin, from a delay of zero until the max delay set by the user. As with the history dependence, the auto mutual information is normalized to a value between 0 and 1, by dividing by H_{spiking} (cf Rudelt et al. in prep.).

Settings

`auto_MI_bin_size_set` sets the values for the sizes of the bins (in seconds) and should be an array of floating-point values.

`auto_MI_max_delay` sets the maximum delay (in seconds) between the past bin and the response.

3.4. CSV Files

To run the `csv-files` task on the sample data, please use

```
$ python3 estimate.py sample_data/spike_times.dat -t csv
```

This task exports the results to comma-separated values (CSV) files, which can then be read either by the tool itself using the `plots` task. Or if you want to perform additional steps and visualize the data in your own way, you can do so by writing a script that reads from these files.

Three files are created by the tool. One with summary statistics, mainly the ones printed in form of a table in the output image, such as in Fig. 1. One with the information needed to plot the history dependence over T. And one for the auto mutual information.

A script is provided with the tool to merge several of the summary-statistics CSV files, to compare the results across multiple neurons. The usage of this tool is demonstrated in the sample analysis on real data, in Sec. 4.

3.5. Plots

To run the `plots` task on the sample data, please use

```
$ python3 estimate.py sample_data/spike_times.dat -t plot
```

The tool reads from the CSV files (see previous section) to produce an image as in Fig. 1. For details on where this image is stored, please see Sec. 2.3.

A few tweaks can be made to the plots. The `plot_settings` that are read from the settings file are used to update the pyplot-rcParams using the `rcParams.update` function. Therefore, in principle, all settings that can be modified in matplotlib in this way, can be modified by adding the desired value to the settings file. However, this is untested behaviour and might produce unwanted results. The color used for the plots can be changed via the `plot_color` setting.

Settings

`plot_settings` sets the settings to be used for the `rcParams.update` function of the matplotlib.pyplot plotting framework, eg to change the format of the output image or to change the font size therein.

`plot_color` sets the color to be used for the output image.

4. Sample Analysis

To exemplify how the tool could be used in practice, we demonstrate the first steps of a potential analysis of electrophysiological recordings. These are openly available and can be obtained from `https://janelia.figshare.com/articles/Eight-probe_Neuropixels_recordings_during_spontaneous_behaviors/7739750` (Steinmetz et al. 2019).

Our demonstration consists of four steps:

- First we set up the analysis
- Then we run the estimator for a few recordings.
- Next we merge the results from each recording into one table.
- Finally we import the results into a script for further processing.

The first step is quickly done: we create a file called `neuropixel_settings.yaml` (Listing 1) with only one line that defines a directory dedicated to the analysis files of the tool for this particular analysis. All other settings are therefore read from the `default.yaml` settings file, which should be fine for most purposes.

Next we want to run the estimator for recordings from a few neurons. In a preliminary step, we created a list of recordings that we want to analyse, an excerpt of which is shown in Listing 2. The columns denote the name of the mouse, the number of the probe, the number of the cluster (or neuron) and the area of the brain in which the recorded neuron is located. We will later pass this information to the tool, so that it appears in the output table.

The data is stored in a Matlab format which the tool cannot interpret. Therefore, we first write a script, provided in Listing 3, that takes the name of a mouse, a probe number and cluster number as input, and prints one spike time per line, as required for the tool. Furthermore, the script makes sure that data requirements as described in 2.1.1 are met: we analyse only the spontaneous activity, for which we assume stationarity; we make sure recordings are sufficiently long and trim recordings that are significantly longer.

Now we run the estimator for each of the recordings, eg for the first one listed in Listing 3, as follows:

```
$ python3 print_spike_times_for_neuron.py Krebs 4 175 | python3 \
  estimate.py /dev/stdin -s neuropixel_settings.yaml \
  --label Krebs;4;175;V1
```

(You need to provide more explicit paths to the respective files, but these are left out here for clarity.) After running the estimator for each of the six recordings, there are six directories within the `ANALYSIS_DIR` labeled `ANALYSIS0000` through `ANALYSIS0005`, and containing the analysis data for each neuron. Next, we want to merge the results for these six neurons into one table.

To merge the results, we first change directory into that of the estimator:

```
$ cd hdestimator
```

There, a script is provided that does the job for us. We run the script with the path to the `ANALYSIS_DIR` as argument:

```
$ python3 merge_csv_stats_files.py \
  /scratch/projects/analysis_files_neuropixels
```

This looks for the CSV files within each subdirectory of `ANALYSIS_DIR` and creates a file called `statistics_merged.csv` within `ANALYSIS_DIR`. This file can now be read for further processing.

In Listing 4 we import the `statistics_merged.csv` file. Each column (cf Table 4 in Appendix A) can be accessed by referencing its name using eg the `pandas` module. We split up the label we passed to the tool into four columns: `mouse_name`, `probe_num`, `cluster_num` and `area`, add new columns representing the logarithms of R_{tot} and T_{D}, respectively, and produce two plots to describe the data.

```
ANALYSIS_DIR : /scratch/projects/analysis_files_neuropixels
```

Listing 1: neuropixel_settings.yaml

```
Krebs    4    175 V1
Krebs    4    176 V1
Krebs    4    177 V1
Krebs    4    178 V1
Krebs    4    179 V1
Krebs    4    180 V1
```

Listing 2: neuron_list.tsv

```
from sys import argv, exit
from scipy.io import loadmat

### settings
data_dir = '/scratch/data'

mouse_names = ['Krebs','Waksman','Robbins']

t_start = [3811, 3633, 3323] # start of spontaneous activity
t_vid_end = [5150, 6153, 5574] # Krebs, Waksman, Robbins

min_rec_len = 800  # don't output recordings below around 13 min
max_rec_len = 1200 # trim recordings longer than 20 min


def print_spike_times(mouse_name,
                      probe_num_1,
                      cluster_num_1):
    assert mouse_name in mouse_names
    probe_num_0 = probe_num_1 - 1
    mouse_num = mouse_names.index(mouse_name)
    t_0 = t_start[mouse_num]
    t_max = t_vid_end[mouse_num]

    spks = loadmat('{}/spks/spks{}_Feb18.mat'.format(data_dir,
                                                     mouse_name),
                   squeeze_me=True)

    # spks consists of
    # spks['spks'][probe_num][0] : spike times (in seconds)
    # spks['spks'][probe_num][1] : clusters
    # spks['spks'][probe_num][2] : cluster heights (in microns)

    st = spks['spks'][probe_num_0][0]
    cl = spks['spks'][probe_num_0][1]

    spike_times = sorted([t for (t, c) in zip(st, cl) if c ==
       cluster_num_1])

```

```
    if len(spike_times) == 0:
        exit()

    spt_i_0 = 0
    while spike_times[spt_i_0] < t_0:
        spt_i_0 += 1

    spt_i_max = -1
    while spike_times[spt_i_max] > t_max:
        spt_i_max -= 1

    rec_len = spike_times[spt_i_max] - spike_times[spt_i_0]

    if rec_len < min_rec_len:
        exit()

    if rec_len > max_rec_len:
        diff = rec_len - max_rec_len
        while spike_times[spt_i_max] > t_max - diff:
            spt_i_max -= 1

    for spt in spike_times[spt_i_0:spt_i_max]:
        print(spt)

if __name__ == "__main__":
    if not len(argv) == 4 or not argv[2].isdecimal() or not argv[3].
       isdecimal():
        print('usage is python3 {} mouseName probeNum clusterNum'.
           format(argv[0]))
        print('probeNum and clusterNum are indexed starting with 1, to
           be consistent with matlab')
        exit()
    exit(print_spike_times(mouse_name=argv[1],
                           probe_num_1=int(argv[2]),
                           cluster_num_1=int(argv[3])))
```

Listing 3: print_spike_times_for_neuron.py

```
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('seaborn-darkgrid')
import pandas as pd
import seaborn as sns
import yaml
with open('neuropixel_settings.yaml', 'r') as settings_file:
    settings = yaml.load(settings_file, Loader=yaml.BaseLoader)

### settings

neuropixels_analysis_dir = settings['ANALYSIS_DIR']
csv_file_name \
    = "{}/statistics_merged.csv".format(neuropixels_analysis_dir)
estimation_method = 'shuffling'
```

```
16
17 areas = [CP, FrCtx, FrMoCtx, HPF, LH, LS,
18          MB, RSP, SC, SSCtx, SomMoCtx, TH, V1, V2]
19
20 ### import data
21
22 data = pd.read_csv(csv_file_name, na_values='-')
23 data.rename(columns={'#analysis_num': 'analysis_num'}, inplace=True)
24 labels = np.array([label.split(';') for label in data['label'].values])
25 data.insert(2, 'mouse_name', labels[:,0])
26 data.insert(3, 'probe_num', labels[:,1])
27 data.insert(4, 'cluster_num', labels[:,2])
28 data.insert(5, 'area', labels[:,3])
29 data.drop('label', axis=1, inplace=True)
30
31 data.insert(6, 'log_Tp', np.log10(data['T_D_{}'.format(
      estimation_method)].values))
32 data.insert(7, 'log_R', np.log10(data['R_tot_{}'.format(
      estimation_method)].values))
33
34 ### analyse
35
36 # plot log-distribution
37 fig0, (ax0, ax1) = plt.subplots(1, 2)
38 data.hist(column='log_R', bins=100, density=True, ax=ax0)
39 data.hist(column='log_T_D', bins=100, density=True, ax=ax1)
40
41 ax0.set_xlabel('history dependence R')
42 ax1.set_xlabel('temporal depth $T_D$')
43
44 plt.show()
45
46 # R vs T_D scatter plot
47 fig0, ax0 = plt.subplots()
48 for area, color in zip(areas, sns.color_palette()):
49     ax0.plot(data.loc[data['area'] == area]['log_T_D'].values,
50              data.loc[data['area'] == area]['log_R'].values,
51              'x', color=color, alpha=0.4)
52
53 ax0.set_ylabel('history dependence R')
54 ax0.set_xlabel('temporal depth $T_D$')
55
56 plt.show()
```

Listing 4: analyse_neuropixels.py

A. Reference Tables

Task	Description
`history-dependence`	Estimate the history dependence in the spiking data (the main analysis).
`confidence-intervals`	Compute confidence intervals for the history dependence estimates .
`auto-mi`	Estimate the auto mutual information in the spiking data (a related measure).
`csv-files`	Save the results as comma separated values (CSV) files.
`plots`	Produce plots, a visual representation of the analysis.
`full-analysis`	Perform all of the steps above.

Table 1: Tasks reference table.

Argument	Description
`-h`, `--help`	Show a help message and exit
`-t`, `--task TASK`	Define task to be performed. One of [`'history-dependence'`, `'confidence-intervals'`, `'auto-mi'`, `'csv-files'`, `'plots'`, `'full-analysis'`]. Per default, the full analysis is performed.
`-e`, `--estimation-method EST_METHOD`	Specify estimation method for the analysis. One of [`'bbc'`, `'shuffling'`, `'all'`].
`-o`, `--output IMAGE_FILE`	Save the output image to file.
`-p`, `--persistent`	Save the analysis to file. If an existing analysis is found, read it from file.
`-s`, `--settings-file SETTINGS_FILE`	Specify yaml file from which to load custom settings.
`-l`, `--label LABEL`	Include a label in the output to classify the analysis.
`-v`, `--verbose`	Print more info at run time.

Table 2: Optional command-line arguments reference table.

Setting	Description
`embedding_step_size`	Step size Δt (in seconds) with which the window is slid through the data.
`embedding_past_range_set`	Set of values for T, the past range (in seconds) to be used for embeddings. Should be an array of floating-point values.
`embedding_number_of_bins_set`	Set of values for d, the number of bins in the embedding. Should be an array of integer values.
`embedding_scaling_exponent_set`	Set of values for κ, the scaling exponent for the bins in the embedding. Should be either an array of floating-point values or a python-dictionary with the three entries below (cf Sec. 3.1):
`number_of_scalings`	(Max.) number of bin scalings for each pair (T, d).
`min_first_bin_size`	The minimal bin size of the first bin, ie its size when κ is maximal ($\kappa = \kappa_{\max}$) for a given pair (T, d).
`min_step_for_scaling`	The minimal difference that two values of κ should have in the linear interpolation between $\kappa = 0$ and $\kappa = \kappa_{\max}$.
`estimation_method`	The method to be used to estimate the history dependence, one of `bbc`, `shuffling` or `all`.
`bbc_tolerance`	The tolerance for the Bayesian Bias Criterion. Influences which embeddings are discarded from the analysis.
`cross_validated_optimization`	If set to `True`, use cross validation for the embedding optimization to avoid overfitting: optimize on the first half of the data only, provide the estimate on the second one.
`number_of_bootstraps_R_max`	The number of bootstrap re-shuffles that should be used to determine the optimal embedding. (Bootstrap the estimates of $R_{\max}$ to determine R_{tot}.)
`number_of_bootstraps_R_tot`	The number of bootstrap re-shuffles that should be used to estimate the confidence interval of the optimal embedding. (Bootstrap the estimates of $R_{\text{tot}} = R(T_{\text{D}})$ to obtain a confidence interval for R_{tot}.)
`number_of_bootstraps_nonessential`	The number of bootstrap re-shuffles that should be used to estimate the confidence interval of embeddings other than the optimal one.

Setting	Description
`block_length_l`	The number of symbols that should be drawn in each block for bootstrap resampling. If it is set to `None` (recommended), the length is automatically chosen, based on heuristics.
`bootstrap_CI_use_sd`	If set to `True`, compute confidence intervals based on the standard deviation of the bootstrap replications.
`bootstrap_CI_percentile_lo`	The lower percentile for the confidence interval. This has no effect if `bootstrap_CI_use_sd` is set to `True`.
`bootstrap_CI_percentile_hi`	The upper percentile for the confidence interval. This has no effect if `bootstrap_CI_use_sd` is set to `True`.
`auto_MI_bin_size_set`	The values for the sizes of the bins (in seconds). Should be an array of floating-point values.
`auto_MI_max_delay`	The maximum delay (in seconds) between the past bin and the response.
`label`	A label for the analysis that appears both in the output image and the CSV file.
`ANALYSIS_DIR`	A parent directory for persistent storage, under which a directory is created per analysis of a spiking times file.
`plot_AIS`	If set to `True`, plot the active information storage (AIS) instead of history dependence (This does not affect the CSV files, both measures are always included there.)
`persistent_analysis`	If set to `True`, save the analysis to file; and if furthermore an existing analysis is found, read it from file.
`verbose_output`	If set to `True`, print more info at run time.
`plot_settings`	A python-dictionary containing the settings to be used for the `rcParams.update` function of the `matplotlib.pyplot` plotting framework.
`plot_color`	The color to be used for the output image.

Table 3: Settings reference table.

Column	Description
`analysis_num`	The analysis number; determines the folder in which the analysis results are stored.
`label`	Label for the analysis; helpful to classify the data.
`T_D_bbc`	Estimated optimal value for the temporal depth T_{D}, based on the BBC estimator.
`R_tot_bbc`	Estimated value for the total history dependence R_{tot}, based on the BBC estimator.
`R_tot_bbc_CI_lo`	Lower bound of the confidence interval of R_{tot}, based on the BBC estimator.
`R_tot_bbc_CI_hi`	Upper bound of the confidence interval of R_{tot}, based on the BBC estimator.
`AIS_tot_bbc`	Estimated value for the total active information storage, based on the BBC estimator.
`AIS_tot_bbc_CI_lo`	Lower bound of the confidence interval of the total AIS, based on the BBC estimator.
`AIS_tot_bbc_CI_hi`	Upper bound of the confidence interval of the total AIS, based on the BBC estimator.
`opt_number_of_bins_d_bbc`	Number of bins d for the embedding that yields (R_{tot}, T_{D}), based on the BBC estimator.
`opt_scaling_k_bbc`	Scaling exponent κ for the embedding that yields (R_{tot}, T_{D}), based on the BBC estimator.
`opt_first_bin_size_bbc`	Size of the first bin τ_1 for the embedding that yields (R_{tot}, T_{D}), based on the BBC estimator.
`T_D_shuffling`	Estimated optimal value for the temporal depth T_{D}, based on the shuffling estimator.
`R_tot_shuffling`	Estimated value for the total history dependence R_{tot}, based on the Shuffling estimator.
`R_tot_shuffling_CI_lo`	Lower bound of the confidence interval of R_{tot}, based on the Shuffling estimator.
`R_tot_shuffling_CI_hi`	Upper bound of the confidence interval of R_{tot}, based on the Shuffling estimator.
`AIS_tot_shuffling`	Estimated value for the total active information storage, based on the Shuffling estimator.
`AIS_tot_shuffling_CI_lo`	Lower bound of the confidence interval of the total AIS, based on the Shuffling estimator.
`AIS_tot_shuffling_CI_hi`	Upper bound of the confidence interval of the total AIS, based on the Shuffling estimator.

Column	Description
`opt_number_of_bins_d_shuffling`	Number of bins d for the embedding that yields (R_{tot}, T_{D}), based on the Shuffling estimator.
`opt_scaling_k_shuffling`	Scaling exponent κ for the embedding that yields (R_{tot}, T_{D}), based on the Shuffling estimator.
`opt_first_bin_size_shuffling`	Size of the first bin τ_1 for the embedding that yields (R_{tot}, T_{D}), based on the Shuffling estimator.
`embedding_step_size`	Step size Δt (in seconds) with which the window is slid through the data.
`bbc_tolerance`	The tolerance for the Bayesian Bias Criterion. Influences which embeddings are discarded from the analysis.
`number_of_bootstraps_bbc`	Number of bootstrap replications performed for the confidence interval of R_{tot}, based on the BBC estimator.
`number_of_bootstraps_shuffling`	Number of bootstrap replications performed for the confidence interval of R_{tot}, based on the Shuffling estimator.
`bs_CI_percentile_lo`	Percentile of the data that is described by the lower bound of the confidence interval (this is set to 2.5 if the CI is computed based on the standard deviation).
`bs_CI_percentile_hi`	Percentile of the data that is described by the lower bound of the confidence interval (this is set to 97.5 if the CI is computed based on the standard deviation).
`firing_rate`	Firing rate of the neuron/ spike train.
`recording_length`	Length of the recording (in seconds).
`H_spiking`	(Unconditional) entropy of the spike train.

Table 4: CSV entries reference table.

B. Performance of the Tool

For better performance of the tool, one should compile the Cython implementation of the embedding module. To do so, (assuming Cython is installed on your computer,) change directory into `src`

```
$ cd src
```

and run

```
$ python3 setup.py build_ext --inplace
```

If all went well you should no longer receive a warning message when running `estimate.py`.

Depending on the data and the settings, running the analysis can take quite some time. As a reference, we provide a few benchmark running times in the table below.

Settings file	Rec. Length [s]	Firing Rate [Hz]	Running Time [h:min:s]
`test.yaml`	952	4	0:00:26
`default.yaml`	952	4	0:05:51
`exhaustive.yaml`	952	4	0:33:11

Table 5: Approximate running times of the full analysis on a 2018 consumer laptop with an `Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz` processor. In all cases, the Cython module was used.

References

Davison, A. C. and D. V. Hinkley (1997). *Bootstrap methods and their application*. Vol. 1. Cambridge university press.

Efron, B. and R. J. Tibshirani (1994). *An introduction to the bootstrap*. CRC press.

Rudelt, L., D. González Marx, M. Wibral, and V. Priesemann (in prep.).

Steinmetz, N. et al. (2019). "Eight-probe Neuropixels recordings during spontaneous behaviors". In: URL: `https://janelia.figshare.com/articles/dataset/Eight-probe_Neuropixels_recordings_during_spontaneous_behaviors/7739750`.