exana Documentation

Release 0.2.0+27.g6715fde

Mikkel E. Lepperød, Milad H. Mobarhan, Alessio Buccino, Tristan S

Nov 12, 2018

Contents

Exana is a python module for neuroscientific data analysis.

CHAPTER 1

Installation

1.1 Pre-configured installation (recommended)

It's strongly recommended that you use Anaconda to install exdir along with its compiled dependencies.

With Anaconda or Miniconda:

```
conda install -c cinpla exana
```

1.2 Python installation

To install with setuptools clone the repo and install with python:

```
git clone https://github.com/CINPLA/exana.git
python setup.py develop
```

CHAPTER 2

Developers' guide

Please fork and send pull requests for contributions.

2.1 Requirements for code contributions

Code contributions to exana need to adhere to the following requirements:

- All added code needs to be useful to more than just you. If only you need the functionality in your project, then add it to your project instead.
- Each function needs to have a corresponding test.
- Each function needs to have an example in the docstring.
- **Plotting functions must be placed in *examples*. Rationale: Everyone has an opinion on how to plot everything.**
  - Every function in examples must also be used in the documentation - see the *docs* folder.
- Create specialized functions. Don't create one function that is general and takes 1000 parameters. Make 1000 functions instead.
- Prioritize functions over classes - that keeps the scope minimal. Use classes only when you really need it for the data or the API.
- Keep it simple. Sometimes it is better to be a bit verbose to make the logic simpler. In other words, use simple functions and avoid classes if you can.
- Avoid wrapping simple functions from other libraries to be "helpful". Just show the function you wanted to wrap in your examples or docstrings instead.

Please conform to pep8 and write docstrings in numpy style

2.2 Testing

To run tests, simply run in the command line:

```
$ py.test --doctest-modules --ignore setup.py
```

Test your code as much as possible, and preferably make a proper test case runnable with pytest, as a minimum write runnable examples in the docstring; see example below:

```
def has_ten_spikes(spiketrain):
    """
    Parameters
    ----------
    spiketrain : neo.SpikeTrain
        A NEO spike train with spike times.

    Returns
    -------
    bool
        True if successful, False otherwise.

    Example
    -------
    >>> import neo
    >>> spiketrain = neo.SpikeTrain(times=list(range(10)), t_stop=10, units='s')
    >>> has_ten_spikes(spiketrain)
    True
    """
    if len(spiketrain) == 10:
        return True
    else:
        return False
```

2.3 Dependencies

Exana depends on *neo* and *numpy*, to keep the global dependencies to a minimum please import special dependencies inside functions.

2.4 Getting the source code

We use the Git version control system. The best way to contribute is through GitHub. You will first need a GitHub account, and you should then fork the repository.

2.5 Working on the documentation

The documentation is written in reStructuredText, using the Sphinx documentation system. To build the documentation:

```
$ cd exana/docs
$ sphinx-apidoc -o . ../exana/
$ make html
```

Then open *doc/build/html/index.html* in your browser.

2.6 Committing your changes

Once you are happy with your changes, **run the test suite again to check that you have not introduced any new bugs**. Then you can commit them to your local repository:

```
$ git commit -m 'informative commit message'
```

If this is your first commit to the project, please add your name and affiliation/employer to `doc/source/authors.rst`

You can then push your changes to your online repository on GitHub:

```
$ git push
```

Once you think your changes are ready to be included in the main Neo repository, open a pull request on GitHub (see https://help.github.com/articles/using-pull-requests).

CHAPTER 3

Examples

statistics_plot.**plot_isi_hist**(*sptr*, *alpha=1*, *ax=None*, *binsize=array(2.0) * ms*, *time_limit=array(100.0) * ms*, *color='b'*, *edgecolor=None*)

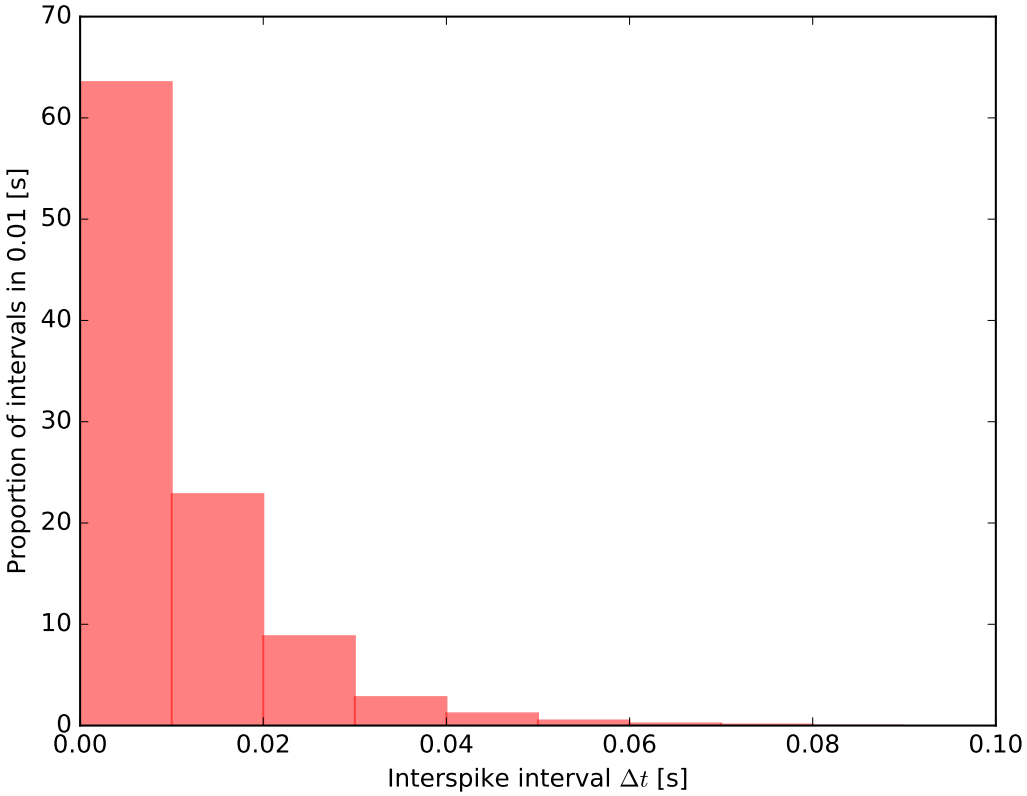
Bar plot of interspike interval (ISI) histogram

Parameters

- **sptr** (*neo.SpikeTrain*)
- **color** (*str*) – color of histogram
- **edgecolor** (*str*) – edgecolor of histogram
- **ax** (*matplotlib axes*)
- **alpha** (*float*) – opacity
- **binsize** (*Quantity(s)*) – binsize of spike rate histogram, default 2 ms
- **time_limit** (*Quantity(s)*) – end time of histogram x limit, default 100 ms

Examples

```
>>> import neo
>>> from numpy.random import rand
>>> spike_train = neo.SpikeTrain(rand(1000) * 10, t_stop=10, units='s')
>>> ax = plot_isi_hist(spike_train, alpha=.1, binsize=10*pq.ms,
...                    time_limit=100*pq.ms, color='r', edgecolor='r')
```



Returns **out**

Return type axes

`statistics_plot.`**`plot_spike_histogram`**(*trials*, *color='b'*, *ax=None*, *binsize=None*, *bins=None*, *output='counts'*, *edgecolor=None*, *alpha=1.0*, *ylabel=None*, *nbins=None*)

histogram plot of trials

Parameters

- **trials** (*list of neo.SpikeTrains*)
- **color** (*str*) – Color of histogram.
- **edgecolor** (*str*) – Color of histogram edges.
- **ax** (*matplotlib axes*)
- **output** (*str*) – Accepts 'counts', 'rate' or 'mean'.
- **binsize** – Binsize of spike rate histogram, default None, if not None then bins are overridden.
- **nbins** (*int*) – Number of bins, defaults to 100 if binsize is None.
- **ylabel** (*str*) – The ylabel of the plot, if None defaults to output type.

Examples

```
>>> import neo
>>> from numpy.random import rand
>>> from exana.stimulus import make_spiketrain_trials
>>> spike_train = neo.SpikeTrain(rand(1000) * 10, t_stop=10, units='s')
>>> epoch = neo.Epoch(times=np.arange(0, 10, 1) * pq.s,
...                   durations=[.5] * 10 * pq.s)
>>> trials = make_spiketrain_trials(spike_train, epoch)
>>> ax = plot_spike_histogram(trials, color='r', edgecolor='b',
...                           binsize=1 * pq.ms, output='rate', alpha=.5)
```

Returns out

Return type axes

statistics_plot.**plot_xcorr**(*spike_trains*, *colors=None*, *edgecolors=None*, *fig=None*, *density=True*, *alpha=1.0*, *gs=None*, *binsize=array(1.0) * ms*, *time_limit=array(1.0) * s*, *split_colors=True*, *xcolor='k'*, *xedgecolor='k'*, *xticksvisible=True*, *yticksvisible=True*, *acorr=True*, *ylim=None*, *names=None*)

Bar plot of crosscorrelation of multiple spiketrians

Parameters

- **spike_trains** (*list of neo.SpikeTrain or neo.SpikeTrain*)
- **colors** (*list or str*) – colors of histogram
- **edgecolors** (*list or str*) – edgecolor of histogram
- **ax** (*matplotlib axes*)
- **alpha** (*float*) – opacity
- **binsize** (*Quantity*) – binsize of spike rate histogram, default 2 ms
- **time_limit** (*Quantity*) – end time of histogram x limit, default 100 ms
- **gs** (*instance of matplotlib.gridspec*)
- **split_colors** (*bool*) – if True splits crosscorrelations into colors from respective autocorrelations
- **xcolor** (*str*) – color of crosscorrelations
- **xedgecolor** (*str*) – edgecolor of crosscorrelations
- **xticksvisible** (*bool*) – show xtics on crosscorrelations, (True by default)
- **yticksvisible** (*bool*) – show ytics on crosscorrelations, (True by default)
- **acorr** (*bool*) – show autocorrelations, (True by default)

Examples

```
>>> import neo
>>> from numpy.random import rand
>>> sptr1 = neo.SpikeTrain(rand(100) * 2, t_stop=2, units='s')
>>> sptr2 = neo.SpikeTrain(rand(100) * 2, t_stop=2, units='s')
```

(continues on next page)

(continued from previous page)

```
>>> sptr3 = neo.SpikeTrain(rand(100) * 2, t_stop=2, units='s')
>>> fig = plot_xcorr([sptr1, sptr2, sptr3])
```

Returns out

Return type fig

CHAPTER 4

Authors

Main authors:

Mikkel Elle Lepperød

Svenn-Arne Dragly

Milad Mobarhan

Alessio Buccino

Tristan Stöber

Our lab : cinpla.org

Contributors:

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index

P

S