pyelectro Documentation

Release 0.1.8

Mike Vella

Oct 03, 2017

Contents

Tool for analysis of electrophysiology in Python.

This package was originally developed by Mike Vella. This has been updated by Padraig Gleeson and others (and moved to NeuralEnsemble) to continue development of pyelectro and Neurotune for use in OpenWorm, Open Source Brain and other projects

Contents:

CHAPTER 1

Source documentation

pyelectro Package

analysis Module

Module for mathematical analysis of voltage traces from electrophysiology.

AUTHOR: Mike Vella vellamike@gmail.com

class pyelectro.analysis.**IClampAnalysis**(*v*, *t*, *analysis_var*, *start_analysis=0*, *end_analysis=None*, *target_data_path=None*, *smooth_data=False*, *show_smoothed_data=False*, *smoothing_window_len=11*, *max_min_method=<function max_min>*, *verbose=False*)

Bases: *pyelectro.analysis.TraceAnalysis*

Analysis class for data from whole cell current injection experiments

This is designed to work with simulations of spiking cells or current clamp experimental data.

A lot of the logic here is hardcoded to work well with Cortical Layer II/III Pyramidal cells in Rats.

Parameters

- **v** – time-dependent variable (usually voltage)
- **t** – time-vector
- **analysis_var** – dictionary containing parameters to be used in analysis such as delta for peak detection
- **start_analysis** – time t where analysis is to start
- **end_analysis** – time in t where analysis is to end

analysable_data

analyse()
If data is analysable analyses and puts all results into a dict

plot_results()
Method represents the results visually.

class pyelectro.analysis.**NetworkAnalysis**(*volts*, *t*, *analysis_var*, *start_analysis=0*, *end_analysis=None*, *smooth_data=False*, *show_smoothed_data=False*, *smoothing_window_len=11*, *verbose=False*)
Bases: object

Analysis class for networks of spiking cells, mainly simulation data

Parameters

- **v** – time-dependent variable (usually voltage)
- **t** – time-vector
- **analysis_var** – dictionary containing parameters to be used in analysis such as delta for peak detection
- **start_analysis** – time t where analysis is to start
- **end_analysis** – time in t where analysis is to end

analyse(*targets=None*, *extra_targets=None*)
Analyses and puts all results into a dict

class pyelectro.analysis.**TraceAnalysis**(*v*, *t*, *start_analysis=0*, *end_analysis=None*)
Bases: object

Base class for analysis of electrophysiology data

Constructor for TraceAnalysis base class takes the following arguments:

Parameters

- **v** – time-dependent variable (usually voltage)
- **t** – time-array (1-to-1 correspondence with v_array)
- **start_analysis** – time in v,t where analysis is to start
- **end_analysis** – time in v,t where analysis is to end

plot_trace(*save_fig=False*, *trace_name='voltage_trace.png'*, *show_plot=True*)
Plot the trace and save it if requested by user.

pyelectro.analysis.**ap_integrals**(*v*, *t*)
TODO:explain this fn

pyelectro.analysis.**broadening_index**(*v*, *t*)
TODO:explain this fn TODO:add logging to this module

pyelectro.analysis.**burst_analyser**(*t*)
Pearson's correlation coefficient applied to interspike times

Parameters **t** – Rank-1 array containing spike times

Returns pearson's correlation coefficient of interspike times

pyelectro.analysis.**centered_slice**(*v*, *index*, *length=5*)
Retruns slice of given length centred on index.

pyelectro.analysis.**elburg_bursting**(*spike_times*)
bursting measure B as described by Elburg & Ooyen 2004

Parameters **spike_times** – sequence of spike times

Returns bursting measure B as described by Elburg & Ooyen 2004

pyelectro.analysis.**exp_fit**(*t*, *y*)
Fits data to an exponential.

Returns K for a formula of the type y=A*exp(K*x)

param t time vector

param y variable which varies with time (such as voltage)

pyelectro.analysis.**filter**(*t*, *v*)

pyelectro.analysis.**inflexion_spike_detector**(*v*, *t*, *threshold=0.4*, *indices=False*, *max_data_points=2000*, *voltage_threshold=-30*)
Computes spike start and stop times based on extent of voltage deflection.

This function requires some familiarity with Python to understand.

Parameters **indices** – whether to return tuples of indices for each spike or times

:return list of tuples with start and end indices of every AP

pyelectro.analysis.**linear_fit**(*t*, *y*)
Fits data to a line

Parameters

- **t** – time vector
- **y** – variable which varies with time (such as voltage)

Returns Gradient M for a formula of the type y=C+M*x

pyelectro.analysis.**load_csv_data**(*file_path*, *delimiter=','*, *plot=False*)
Extracts time and voltage data from a csv file

Data must be in a csv and in two columns, first time and second voltage. Units should be SI (Volts and Seconds).

Parameters **file_path** – full file path to file e.g /home/mike/test.csv

Returns two lists - time and voltage

pyelectro.analysis.**max_min**(*a*, *t*, *delta=0*, *peak_threshold=0.0*, *verbose=False*)
Find the maxima and minima of a voltage trace.

:note This method does not appear to be very robust when comparing to experimental data

Parameters

- **a** – time-dependent variable (usually voltage)
- **t** – time-vector
- **delta** – the value by which a peak or trough has to exceed its neighbours to be considered outside of the noise
- **peak_threshold** – peaks below this value are discarded

Returns turning_points, dictionary containing number of max, min and their locations

Note: minimum value between two peaks is in some ways a better way of obtaining a minimum since it guarantees an answer, this may be something which should be implemented.

pyelectro.analysis.**max_min_interspike_time**(*t*)
Calculate the maximum & minimum interspike interval from the list of maxima times

Parameters **t** – a list of spike times in ms

Returns (max, min) interspike time

pyelectro.analysis.**max_min_simple**(*a*, *times*, *delta=0*, *peak_threshold=0.0*, *verbose=False*)

pyelectro.analysis.**mean_spike_frequency**(*t*)
Find the average frequency of spikes

Parameters **t** – a list of spike times in ms

Returns mean spike frequency in Hz, calculated from mean interspike time

pyelectro.analysis.**minima_phases**(*max_min_dictionary*)
Find the phases of minima.

Minima are found by finding the minimum value between sets of two peaks. The phase of the minimum relative to the two peaks is then returned. i.e the fraction of time elapsed between the two peaks when the minimum occurs is returned.

It is very important to make sure the correct delta is specified for peak discrimination, otherwise unexpected results may be returned.

Parameters **max_min_dictionary** – max_min_dictionary

Returns phase of minimum relative to peaks.

pyelectro.analysis.**phase_plane**(*t*, *y*, *plot=False*)
Return a tuple with two vectors corresponding to the phase plane of the tracetarget

pyelectro.analysis.**pptd**(*t*, *y*, *bins=10*, *xyrange=None*, *dvdt_threshold=None*, *plot=False*)
Returns a 2D map of x vs y data and the xedges and yedges. in the form of a vector (H,xedges,yedges) Useful for the PPTD method described by Van Geit 2007.

pyelectro.analysis.**pptd_error**(*t_model*, *v_model*, *t_target*, *v_target*, *dvdt_threshold=None*)
Returns error function value from comparison of two phase pptd maps as described by Van Geit 2007.

pyelectro.analysis.**print_comment**(*text*, *print_it=False*, *warning=False*)

pyelectro.analysis.**print_comment_v**(*text*, *warning=False*)

pyelectro.analysis.**single_spike_width**(*y*, *t*, *baseline*)
Find the width of a spike at a fixed height

calculates the width of the spike at height baseline. If the spike shape does not intersect the height at both sides of the peak the method will return value 0. If the peak is below the baseline 0 will also be returned.

The input must be a single spike or nonsense may be returned. Multiple-spike data can be handled by the interspike_widths method.

Parameters

- **y** – voltage trace (array) corresponding to the spike
- **t** – time value array corresponding to y
- **baseline** – the height (voltage) where the width is to be measured.

Returns width of spike at height defined by baseline

pyelectro.analysis.**smooth**(*x*, *window_len=11*, *window='hanning'*)
Smooth the data using a window with requested size.

This function is useful for smoothing out experimental data. This method utilises the convolution of a scaled window with the signal. The signal is prepared by introducing reflected copies of the signal (with the window size) in both ends so that transient parts are minimized in the begining and end part of the output signal.

Parameters

- **x** – the input signal
- **window_len** – the dimension of the smoothing window; should be an odd integer
- **window** – the type of window from 'flat', 'hanning', 'hamming', 'bartlett', 'blackman', flat window will produce a moving average smoothing.

Returns smoothed signal

example:

```
t=linspace(-2,2,0.1)
x=sin(t)+randn(len(t))*0.1
y=smooth(x)
```

See also:

numpy.hanning numpy.hamming numpy.bartlett numpy.blackman numpy.convolve scipy.signal.lfilter

pyelectro.analysis.**spike_broadening**(*spike_width_list*)
Returns the value of the width of the first AP over the mean value of the following APs.

pyelectro.analysis.**spike_covar**(*t*)
Calculates the coefficient of variation of interspike times

Parameters **t** – Rank-1 array containing spike times

Returns coefficient of variation of interspike times

pyelectro.analysis.**spike_frequencies**(*t*)
Calculate frequencies associated with interspike times

Parameters **t** – a list of spike times in ms

Returns list of frequencies in Hz associated with interspike times and times associated with the frequency (time of first spike in pair)

pyelectro.analysis.**spike_widths**(*y*, *t*, *max_min_dictionary*, *baseline=0*, *delta=0*)
Find the widths of each spike at a fixed height in a train of spikes.

Returns the width of the spike of each spike in a spike train at height baseline. If the spike shapes do not intersect the height at both sides of the peak the method will return value 0 for that spike. If the peak is below the baseline 0 will also be returned for that spike.

Parameters

- **y** – voltage trace (array) corresponding to the spike train
- **t** – time value array corresponding to y
- **max_min_dictionary** – precalculated max_min_dictionary
- **baseline** – the height (voltage) where the width is to be measured.

Returns width of spike at height defined by baseline

pyelectro.analysis.**three_spike_adaptation**(*t*, *y*)
Linear fit of amplitude vs time of first three AP spikes

Initial action potential amplitudes may very substaintially in amplitude and then settle down.

Parameters

- **t** – time vector (AP times)
- **y** – corresponding AP amplitude

Returns Gradient M for a formula of the type y=C+M*x for first three action potentials

pyelectro.analysis.**voltage_plot**(*t*, *v*, *title=None*)
Plot electrophysiology recording.

pyelectro.analysis.**window_peak_detector**(*v*, *delta=0.01*)
Detects peak by comparing mean of either side of peak and deciding whether it exceeds some threshold.

Returns Boolean, True if a peak is detected in that window

pyelectro.analysis.**y_from_x**(*y*, *x*, *y_to_find*)
Returns list of x values corresponding to a y after a doing a univariate spline interpolation

Parameters

- **x** – x-axis numerical data
- **y** – corresponding y-axis numerical data
- **y_to_find** – x value for desired y-value, interpolated from nearest two measured x/y value pairs

Returns interpolated y value

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index