pyret Documentation

Release 0.6.0

Benjamin Naecker, Niru Maheswaranathan

May 08, 2018

Contents

pyret is a library for pre-processing, analyzing, and visualizing data from retina electrophysiology experiments. It was written because we think sharing code and data withing the scientific community is easiest when there are open standards to adhere to.

CHAPTER 1

Installation

1.1 Basic

The fastest way to install is by grabbing the code from Github:

```
$ git clone https://github.com/baccuslab/pyret.git
$ cd pyret
$ python setup.py install
```

Pyret supports Python2.7 and Python3.4+.

1.2 Dependencies

Pyret requires the following dependencies:

- `numpy`
- `scipy`
- `scikit-image`
- `scikit-learn`
- `matplotlib`

1.3 Development

To contribute to `pyret`, you'll need to also install `sphinx` and `numpydoc` for documentation and `pytest` for testing. We adhere to the NumPy/SciPy documentation standards.

CHAPTER 2

Quickstart

2.1 Overview

Pyret is a Python package that provides tools for analyzing stimulus-evoked neurophysiology data. The project grew out of work in a retinal neurophsyiology and computation lab (hence the name), but its functionality should be applicable to any neuroscience work in which you wish to characterize how neurons behave in response to an input.

Pyret's functionality is broken into modules.

- stimulustools: Functions for manipulating input stimuli.
- spiketools: Tools to characterize spikes.
- filtertools: Tools to estimate and characterize linear filters fitted to neural data.
- nonlinearities: Classes for estimating static nonlinearities.
- visualizations: Functions to visualize responses and fitted filters/nonlinearities.

Pyret works on Python3.4+ and Python2.7.

2.2 Demo

2.2.1 Importing pyret

Let's explore how pyret might be used in a very common analysis pipeline. First, we'll import the relevant modules.

```
>>> import pyret
>>> import numpy as np
>>> import matplotlib.pyplot as plt
```

For this demo, we'll be using data from a retinal ganglion cell (RGC), whose spike times were recorded using a multi-electrode array. (Data courtesy of Lane McIntosh.) We'll load the stimulus used in the experiment, as well as the spike times for the cell.

The data is stored in the GitHub repository, in both HDF5 and NumPy's native `.npz` formats. The following code snippets show how to download and load the data in those formats, respectively. Note that using data in the HDF5 format imposes the additional dependency of the `h5py` package, which is available on PyPI.

2.2.2 Loading data from HDF5

To download the data in HDF5 format, use the shell command:

```
$ wget https://github.com/baccuslab/pyret/raw/master/docs/tutorial-data.h5
```

To use `curl` instead of `wget`, run:

```
$ curl -L -o tutorial-data.h5 https://github.com/baccuslab/pyret/raw/master/docs/
↪tutorial-data.h5
```

To load the data, back in the Python shell, run:

```
>>> data_file = h5py.File('tutorial-data.h5', 'r')
>>> spikes = data_file['spike-times']   # Spike times for one cell
>>> stimulus = data_file['stimulus'].value.astype(np.float64)
>>> frame_rate = data_file['stimulus'].attrs.get('frame-rate')
```

2.2.3 Loading data from `.npz`

To download and the data in the `.npz` file format, use the following command:

```
$ wget https://github.com/baccuslab/pyret/raw/master/docs/tutorial-data.npz
```

Or, using `curl`:

```
$ curl -L -o tutorial-data.npz https://github.com/baccuslab/pyret/raw/master/docs/
↪tutorial-data.npz
```

Then, in the Python shell, load the data with:

```
>>> arrays = np.load('tutorial-data.npz')
>>> spikes, stimulus, frame_rate = arrays['spikes'], arrays['stimulus'].astype(np.
↪float64), arrays['frame_rate'][0]
```

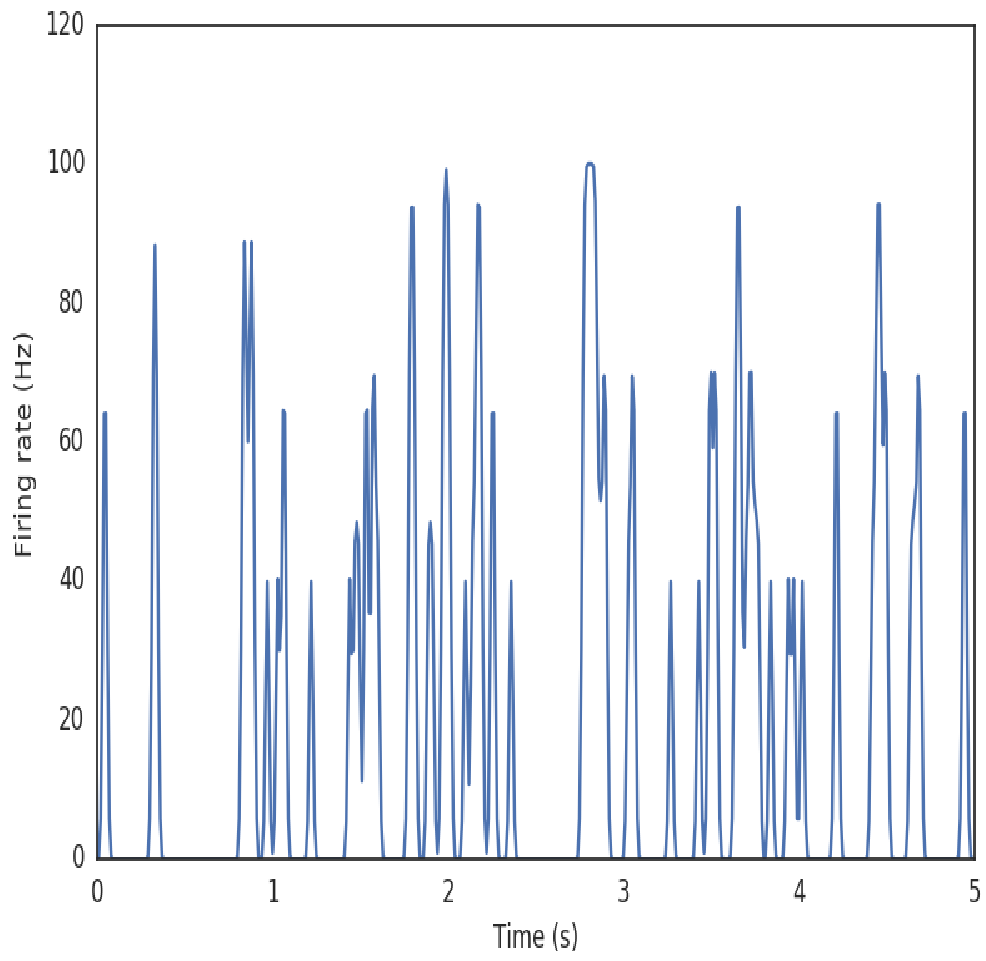
2.2.4 Estimating firing rates

The stimulus is a spatio-temporal gaussian white noise checkboard, with shape `(time, nx, ny)`. Each spatial position is drawn independently from a normal distribution on each temporal frame. We will z-score the stimulus, and create a time-axis for it, to reference the spike times for this cell to the frames of the stimulus.

```
>>> stimulus -= stimulus.mean()
>>> stimulus /= stimulus.std()
>>> time = np.arange(stimulus.shape[0]) * frame_rate
```

To begin, let's look at the spiking behavior of the RGC. We'll create a peri-stimulus time histogram, by binning the spike times and smoothing a bit. This is an estimate of the firing rate of the RGC over time.

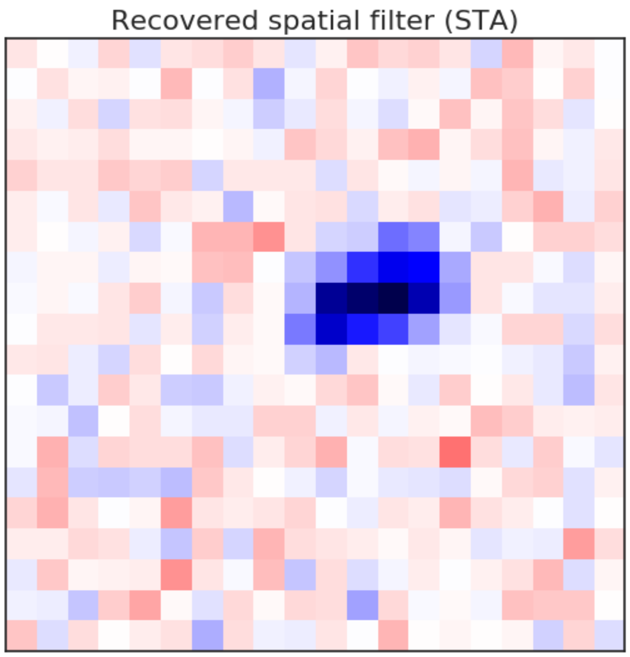
```
>>> binned = pyret.spiketools.binspikes(spikes, time)
>>> rate = pyret.spiketools.estfr(binned, time)
>>> plt.plot(time[:500], rate[:500])
>>> plt.xlabel('Time (s)')
>>> plt.ylabel('Firing rate (Hz)')
```

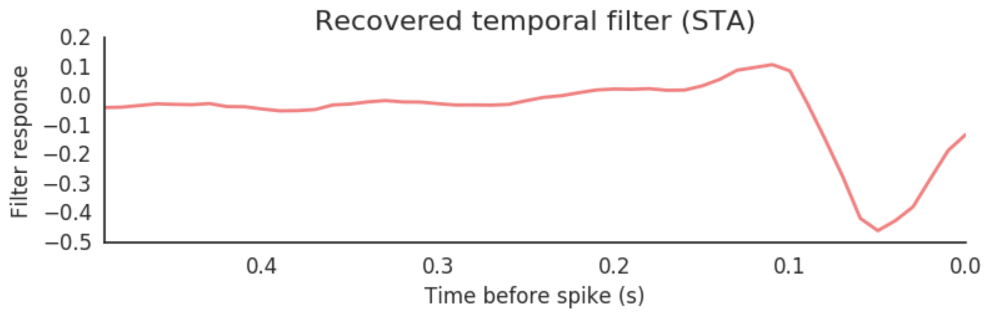


2.2.5 Estimating a receptive field

One widely-used and informative description of the cell is its receptive field. This is a linear approximation to the function of the cell, and captures the average visual feature to which it responds. Because our data consists of spike times, we'll compute the *spike-triggered average* (STA) for the cell.

```
>>> filter_length_seconds = 0.5  # 500 ms filter
>>> filter_length = int(filter_length_seconds / frame_rate)
>>> sta, tax = pyret.filtertools.sta(time, stimulus, spikes, filter_length)
>>> fig, axes = pyret.visualizations.plot_sta(tax, sta)
>>> axes[0].set_title('Recovered spatial filter (STA)')
>>> axes[1].set_title('Recovered temporal filter (STA)')
>>> axes[1].set_xlabel('Time relative to spike (s)')
>>> axes[1].set_ylabel('Filter response')
```





Important: It is common to hear the terms "STA", "linear filter", and "receptive field" used interchangeably. However, this is technically incorrect. The STA is an unbiased estimate of the time-reverse of a best-fitting linear filter (in the least-squares sense), *assuming the stimulus is uncorrelated*. If the stimulus contains correlations, those will appear in the arrays returned by both `filtertools.sta` and `filtertools.revcorr`. As Gaussian white noise, which is uncorrelated, is an exceedingly common stimulus, practioners often loosely refer to the STA as the linear filter, keeping the time-reversing process implicit. The `pyret` methods and docstrings strive for the maximal amount of clarity when refering to these objects, and the documentation should be heeded about whether a filter or STA is expected.

2.2.6 Estimating a nonlinearity

While the STA gives a lot of information, it is not the whole story. Real RGCs are definitely *not* linear. One common way to correct for this fact is to fit a single, time-invariant (static), point-wise nonlinearity to the data. This is a mapping between the linear response to the real spiking data; in other words, it captures the difference between how the cell *would response if it were linear* and how the cell actually responds.

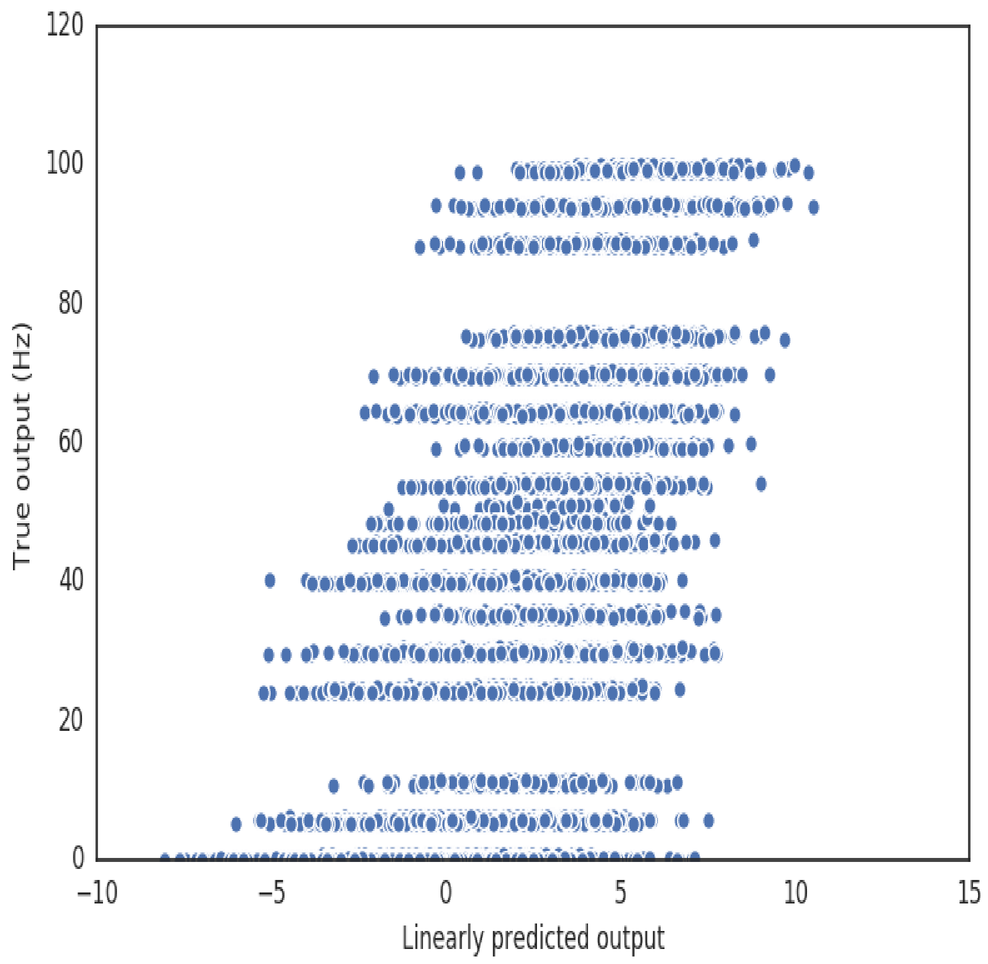
The first step in computing a nonlinearity is to compute how the recovered linear filter responds to the input stimulus. This is done via convolution of the linear filter with the stimulus.

```
>>> pred = pyret.filtertools.linear_response(sta[::-1], stimulus)
>>> stimulus.shape
(30011, 20, 20)
>>> pred.shape
(30011,)
```

Important: Note here that we're *flipping* the STA before passing it to the `linear_response` function. This function expects a true *linear filter*, while the arrays returned by `sta` and `revcorr` are reverse- correlations. This must be flipped along the time (zero-th) axis to arrive at a filter.

We can get a sense for how poor our linear prediction is, simply by plotting the predicted versus the actual response at each time point.

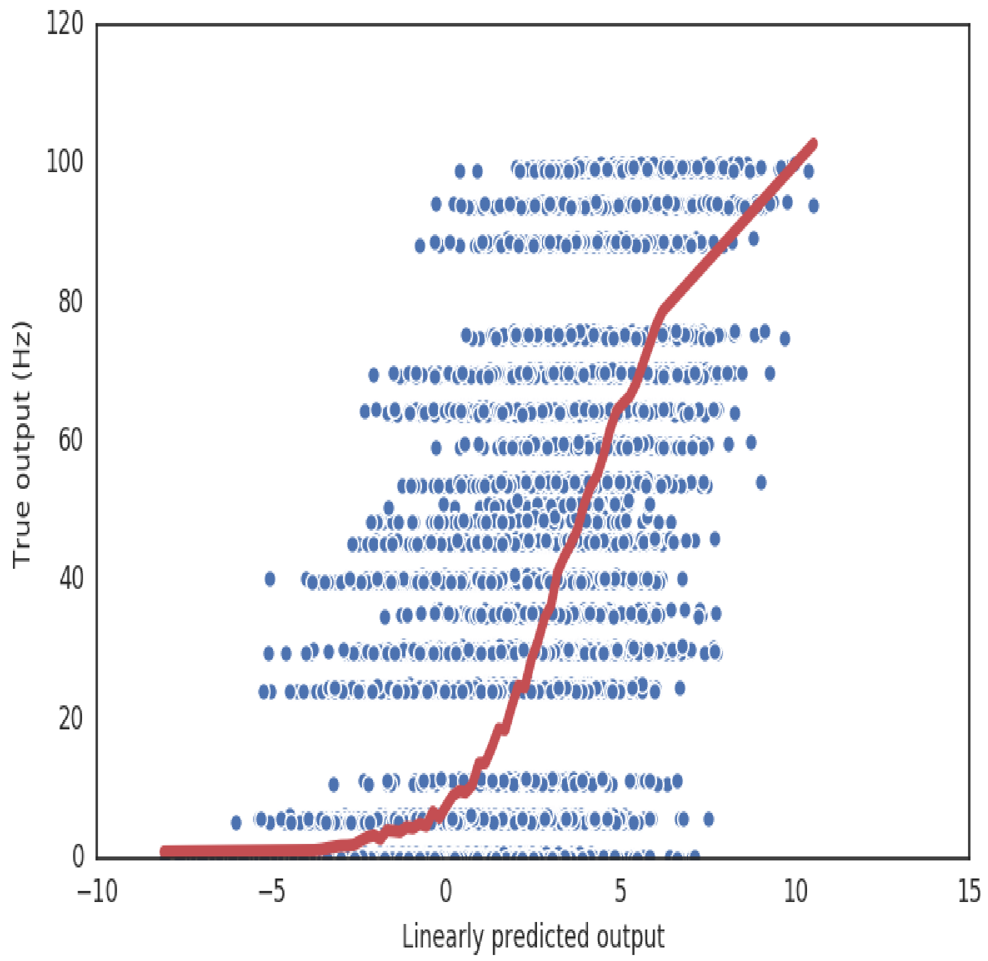
```
>>> plt.plot(pred, rate, linestyle='none', marker='o', mew=1, mec='w')
>>> plt.xlabel('Linearly predicted output')
>>> plt.ylabel('True output (Hz)')
```



It's clear that there is at least some nonlinear behavior in the cell. For one thing, firing rates can never be negative, but our linear prediction definitely is.

`pyret` contains several classes for fitting nonlinearities to data. The simplest is the `Binterp` class (a portmanteau of “bin” and “interpolate”), which computes the average true output in specified bins along the input axis. It uses variable-sized bins, so that each bin has roughly the same number of data points.

```
>>> nbins = 50
>>> binterp = pyret.nonlinearities.Binterp(nbins)
>>> binterp.fit(pred, rate)
>>> nonlin_range = (pred.min(), pred.max())
>>> binterp.plot(nonlin_range, linewidth=5, label='Binterp')   # Plot nonlinearity␣
↪over the given range
```



One can also fit sigmoidal nonlinearities, or a nonlinearity using a Gaussian process (which has some nice advantages, and returns errorbars automatically). More information about these can be found in the full documentation.

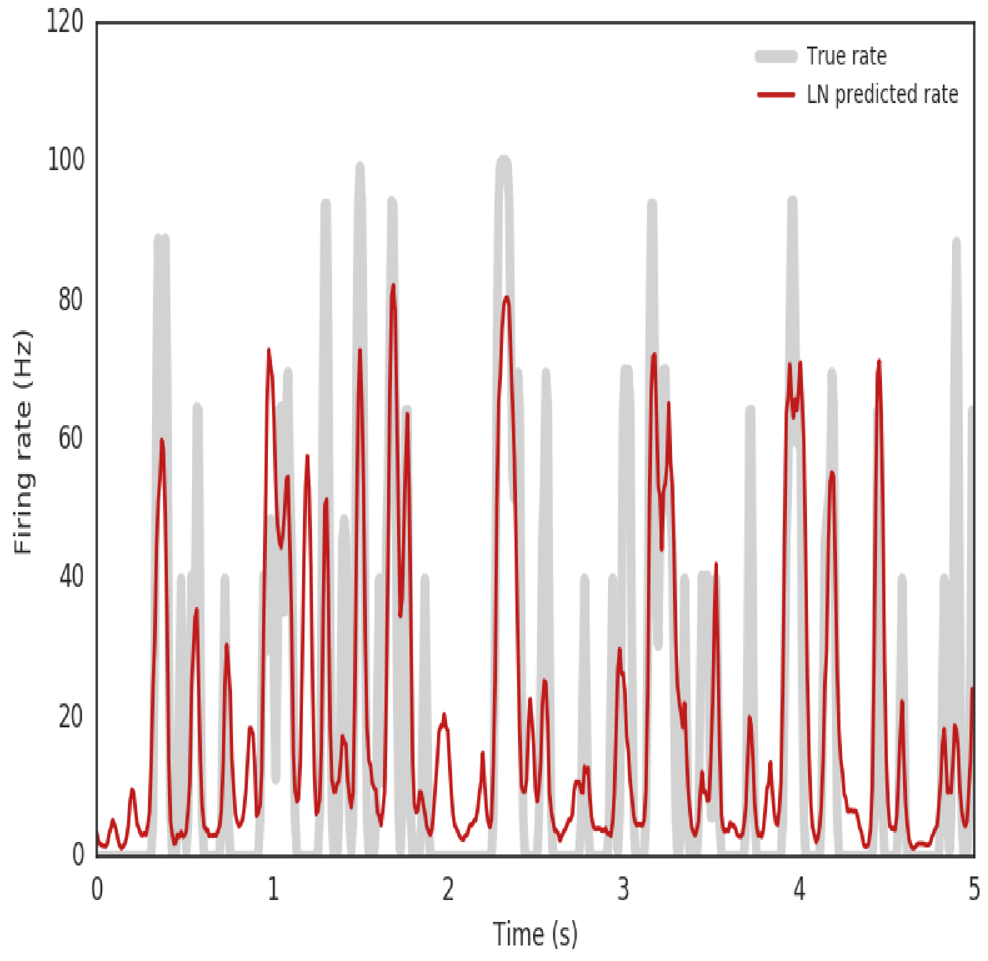
We can now compare how well the full LN model captures the cell’s response characteristics.

```
>>> predicted_rate = binterp.predict(pred)
>>> plt.figure()
>>> plt.plot(time[:500], rate[:500], linewidth=5, color=(0.75,) * 3, alpha=0.7, label=
↪'True rate')
>>> plt.plot(time[:500], predicted_rate[:500], linewidth=2, color=(0.75, 0.1, 0.1),␣
↪label='LN predicted rate')
>>> plt.legend()
>>> plt.xlabel('Time (s)')
```

(continues on next page)

(continued from previous page)

```
>>> plt.ylabel('Firing rate (Hz)')
>>> np.corrcoef(rate, predicted_rate)[0, 1] # Correlation coefficient on training data
0.70315310866999448
```



CHAPTER 3

API Reference

3.1 filtertools

Tools and utilities for computing spike-triggered averages (filters), finding spatial and temporal components of spatiotemporal filters, and basic filter signal processing.

`pyret.filtertools.`**`ste`**(*time*, *stimulus*, *spikes*, *nsamples_before*, *nsamples_after=0*)

Constructs an iterator over spike-triggered stimuli.

Parameters

- **`time`** (*`ndarray`*) – The time array corresponding to the stimulus.
- **`stimulus`** (*`ndarray`*) – A spatiotemporal or temporal stimulus array, where time is the first dimension.
- **`spikes`** (*`iterable`*) – A list or ndarray of spike times.
- **`nsamples_before`** (*`int`*) – Number of samples to include in the STE before the spike.
- **`nsamples_after`** (*`int`*) – Number of samples to include in the STE after the spike, which defaults to 0.

Returns **ste** – A generator that yields samples from the spike-triggered ensemble.

Return type generator

Notes

The spike-triggered ensemble (STE) is the set of all stimuli immediately surrounding a spike. If the full stimulus distribution is p(s), the STE is p(s | spike).

`pyret.filtertools.`**`sta`**(*time*, *stimulus*, *spikes*, *nsamples_before*, *nsamples_after=0*)

Compute a spike-triggered average.

Parameters

- **time** (*ndarray*) – The time array corresponding to the stimulus
- **stimulus** (*ndarray*) – A spatiotemporal or temporal stimulus array (where time is the first dimension)
- **spikes** (*iterable*) – A list or ndarray of spike times
- **nsamples_before** (*int*) – Number of samples to include in the STA before the spike
- **nsamples_after** (*int*) – Number of samples to include in the STA after the spike (default: 0)

Returns

- **sta** (*ndarray*) – The spatiotemporal spike-triggered average. Note that time increases with increasing array index, i.e. time of the spike is at the index for which `tax == 0`.
- **tax** (*ndarray*) – A time axis corresponding to the STA, giving the time relative to the spike for each time point of the STA.

Notes

The spike-triggered average (STA) is the averaged stimulus feature conditioned on the presence of a spike. This is widely-used method for estimating a neuron's receptive field, and captures the average stimulus feature to which the neuron responds.

Formally, the STA is defined as the function [1]:

$$C(\tau) = \frac{1}{N} \sum_{i=1}^{N} s(t_i - \tau)$$

where τ is time preceding the spike, and t_i is the time of the ith spike.

The STA is often used to estimate a linear filter which captures a neuron's responses. If the stimulus is uncorrelated (spherical), the STA is unbiased and proportional to the time-reverse of the linear filter.

Note that the `tax` time values returned by this method are formally given by `t_i - \tau`, i.e., they are the actual time relative to the spike of each corresponding point in the STA.

References

[1] Dayan, P. and L.F. Abbott. Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems. 2001.

`pyret.filtertools.`**`stc`**(*time*, *stimulus*, *spikes*, *nsamples_before*, *nsamples_after=0*)

Compute the spike-triggered covariance.

Parameters

- **time** (*ndarray*) – The time array corresponding to the stimulus, where time is the first dimension.
- **stimulus** (*ndarray*) – A spatiotemporal or temporal stimulus array.
- **spikes** (*iterable*) – A list or ndarray of spike times.
- **nsamples_before** (*int*) – Number of samples to include in the STC before the spike.
- **nsamples_after** (*int*) – Number of samples to include in the STC after the spike, which defaults to 0.

Returns **stc** – The spike-triggered covariance (STC) matrix.

Return type ndarray

`pyret.filtertools.`**`lowranksta`**`(`*sta_orig*, *k=10*`)`

Constructs a rank-k approximation to the given spatiotemporal STA. This is useful for estimating a spatial and temporal kernel for an STA or for denoising.

Parameters

- **`sta_orig`** (*`array_like`*) – 3D STA to be separated, shaped as `(time, space, space)`.
- **`k`** (*`int`*) – Number of components to keep (rank of the reduced STA).

Returns

- **sk** (*array_like*) – The rank-k estimate of the original STA.
- **u** (*array_like*) – The top `k` temporal components (each column is a component).
- **s** (*array_like*) – The top `k` singular values.
- **v** (*array_like*) – The top `k` spatial components (each row is a component). These components have all spatial dimensions collapsed to one.

Notes

This method requires that the STA be 3D. To decompose a STA into a temporal and 1-dimensional spatial component, simply promote the STA to 3D before calling this method.

Despite the name this method accepts both an STA or a linear filter. The components estimated for one will be flipped versions of the other.

`pyret.filtertools.`**`decompose`**`(`*sta*`)`

Decomposes a spatiotemporal STA into a spatial and temporal kernel.

Parameters **`sta`** (*`array_like`*) – The full 3-dimensional STA to be decomposed, of shape `(t, nx, ny)`.

Returns

- **s** (*array_like*) – The spatial kernel, with shape `(nx * ny,)`.
- **t** (*array_like*) – The temporal kernel, with shape `(t,)`.

`pyret.filtertools.`**`filterpeak`**`(`*sta*`)`

Find the peak (single point in space/time) of a smoothed STA or linear filter.

Parameters **`sta`** (*`array_like`*) – STA or filter for which to find the peak. It should be shaped as `(time, ...)`, where ellipses indicate any spatial dimensions to the array.

Returns

- **linear_index** (*int*) – Linear index of the maximal point, i.e. treating the array as flattened.
- **sidx** (*1- or 2-element tuple*) – Spatial index of the maximal point. This returns a tuple with the same number of elements as the filter has spatial dimensions.
- **tidx** (*int*) – Temporal index of the maximal point.

`pyret.filtertools.`**`smooth`**`(`*f*, *spacesig=0.5*, *timesig=1*`)`

Smooths a 3D spatiotemporal STA or linear filter using a multi-dimensional Gaussian filter with the given properties.

Parameters

- **f** (*array_like*) – 3D STA or filter to be smoothed.
- **spacesig** (*float*) – The standard deviation of the spatial Gaussian smoothing kernel.
- **timesig** (*float*) – The standard deviation of the temporal Gaussian smoothing kernel.

Returns **fsmooth** – The smoothed filter, with the same shape as the input.

Return type array_like

`pyret.filtertools.`**`cutout`**(*arr*, *idx=None*, *width=5*)

Cut out a chunk of the given stimulus or filter.

Parameters

- **arr** (*array_like*) – Stimulus, STA, or filter array from which the chunk is cut out. The array should be shaped as `(time, spatial, spatial)`.
- **idx** (*array_like, optional*) – 2D array specifying the row and column indices of the center of the section to be cut out (if None, the indices are taken from `filterpeak`).
- **width** (*int, optional*) – The size of the chunk to cut out from the start indices. Defaults to 5 samples.

Returns **cut** – The cut out section of the given stimulus, STA, or filter.

Return type array_like

Notes

This method can be useful to reduce the space and time costs of computations involving stimuli and/or filters. For example, a neuron's receptive field is often much smaller than the stimulus, but this method can be used to only compare the relevant portions of the stimulus and receptive field.

`pyret.filtertools.`**`resample`**(*arr*, *scale_factor*)

Resamples a 1-D or 2-D array by the given scale.

Parameters

- **arr** (*array_like*) – The original array to be resampled.
- **scale_factor** (*int_like*) – The factor by which `arr` will be resampled. For example, a factor of 2 results in an of twice the size in each dimension, with points interpolated between existing points.

Returns **res** – The resampled array. If `arr` has shape (M,N), `res` has shape `(scale_factor*M, scale_factor*N)`.

Return type array_like

Raises

- An AssertionError is raised if the scale factor is <= 0.
- A ValueError is raised if the input array is not 1- or 2-dimensional.

`pyret.filtertools.`**`flat2d`**(*x*)

Flattens all dimensions after the first of the given array

Useful for collapsing spatial dimensions in a spatiotemporal stimulus or filter.

`pyret.filtertools.`**`get_ellipse`**(*spatial_filter*, *sigma=2.0*)

Get the parameters of an ellipse fit to a spatial STA or linear filter.

Parameters

- **`spatial_filter`** (*`array_like`*) – The spatial receptive field to which the ellipse should be fit.
- **`sigma`** (*`float, optional`*) – Determines the size of the ellipse contour, in units of standard deviations. (Default: 2)

Returns

- **center** (*(float,float)*) – The receptive field center (location stored as an (x,y) tuple).
- **widths** (*[float,float]*) – Two-element list of the size of each principal axis of the RF ellipse.
- **theta** (*float*) – angle of rotation of the ellipse from the vertical axis, in radians.

`pyret.filtertools.`**`get_regionprops`**(*spatial_filter*, *percentile=0.95*)

Gets region properties of a 2D spatial STA or linear filter.

This returns various attributes of the non-zero area of the given spatial filter, such as its area, centroid, eccentricity, etc.

```
>>> regions = get_regionprops(sta_spatial)
>>> print(regions[0].area) # prints the area of the first region
```

Parameters

- **`spatial_filter`** (*`array_like`*) – The spatial linear filter to which the ellipse should be fit.
- **`percentile`** (*`float, optional`*) – The cutoff percentile at which the contour is taken. Defaults to 0.95.

Returns **regions** – List of region properties (see `skimage.measure.regionprops` for more information).

Return type list

`pyret.filtertools.`**`normalize_spatial`**(*frame*, *scale_factor=1.0*, *clip_negative=False*)

Normalizes a spatial frame, for example of a stimulus or STA, by doing the following:

1. mean subtraction using a robust estimate of the mean (ignoring outliers).
2. scaling such that the std. dev. of the pixel values is 1.0.

Parameters

- **`frame`** (*`array_like`*) – The spatial frame to be normalized.
- **`scale_factor`** (*`float, optional`*) – The given frame is resampled at a sampling rate of this ratio times the original sampling rate (Default: 1.0).
- **`clip_negative`** (*`boolean, optional`*) – Whether or not to clip negative values to 0. (Default: False).

Returns **resampled** – The normalized (and potentially resampled) frame.

Return type array_like

`pyret.filtertools.`**`linear_response`**(*filt*, *stim*, *nsamples_after=0*)

Compute the response of a linear filter to a stimulus.

Parameters

- **filt** (*array_like*) – The linear filter whose response is to be computed. The array should have shape `(t, ...)`, where `t` is the number of time points in the filter and the ellipsis indicates any remaining spatial dimenions. The number of dimensions and the sizes of the spatial dimensions must match that of `stim`.
- **stim** (*array_like*) – The stimulus to which the predicted response is computed. The array should have shape `(T,...)`, where `T` is the number of time points in the stimulus and the ellipsis indicates any remaining spatial dimensions. The number of dimensions and the sizes of the spatial dimenions must match that of `filt`.
- **nsamples_after** (*int, optional*) – The number of acausal points in the filter. Defaults to 0.

Returns **pred** – The predicted linear response, of shape `(t,)`.

Return type array_like

Raises *ValueError : If the number of dimensions of ``stim`` and ``filt`` do not* – match, or if the spatial dimensions differ.

Notes

Note that the first parameter is a *linear filter*. The values returned by `filtertools.sta` and `filtertools.revcorr` are proportional to the time-reverse of the linear filter, so to use those values in this function, they must be flipped along the first dimension.

Both `filtertools.sta` and `filtertools.revcorr` can estimate "acausal" components, such as points in the stimulus occuring *after* a spike. The value passed as parameter `nsamples_after` must match that value used when calling `filtertools.sta` or `filtertools.revcorr`.

`pyret.filtertools.`**`revcorr`**(*stimulus*, *response*, *nsamples_before*, *nsamples_after=0*)

Compute the reverse-correlation between a stimulus and a response.

Parameters

- **stimulus** (*array_like*) – A input stimulus correlated with the `response`. Must be of shape `(t, ...)`, where `t` is the time and `...` indicates any spatial dimensions.
- **response** (*array_like*) – A continuous output response correlated with `stimulus`. Must be one-dimensional, of size `t`, the same size as `stimulus` along the first axis. Note that the first `history` points of the response are ignored, where `history = nsamples_before + nsamples_after`, in order to only return the portion of the correlation during which the `stimulus` and `response` completely overlap.
- **nsamples_before** (*int*) – The maximum negative lag for the correlation between stimulus and response, in samples.
- **nsamples_after** (*int, optional*) – The maximum positive lag for the correlation between stimulus and response, in samples. Defaults to 0.

Returns

- **rc** (*array_like*) – An array of shape `(nsamples_before + nsamples_after, ...)` containing the best-fitting linear filter which predicts the response from the stimulus. The ellipses indicates spatial dimensions of the filter.
- **lags** (*array_like*) – An array of shape `(nsamples_before + nsamples_after, )`, which gives the lags, in samples, between `stimulus` and `response` for the correlation returned in `rc`. This can be converted to an axis of time (like that returned from `filtertools.sta`) by multiplying by the sampling period.

Raises

- ValueError : If the `stimulus` and `response` arrays do not match in
- size along the first dimension.

Notes

The `response` and `stimulus` arrays must share the same sampling rate. As the stimulus often has a lower sampling rate, one can use `stimulustools.upsample` to upsample it.

Reverse correlation is a method analogous to spike-triggered averaging for continuous response variables, such as a membrane voltage recording. It estimates the stimulus feature that most strongly correlates with the response on average.

It is the time-reverse of the standard cross-correlation function, and is defined as:

$$c[-k] = \sum_n s[n]r[n-k]$$

The parameter `k` is the lag between the two signals in samples. The range of lags computed in this method are determined by `nsamples_before` and `nsamples_after`.

Note that, as with `filtertools.sta`, the values (samples) in the `lags` array increase with increasing array index. This means that time is moving forward with increasing array index.

Also note that this method assumes an uncorrelated stimulus. If the stimulus is correlated, those will bias the estimated reverse correlation.

3.2 nonlinearities

Tools for fitting nonlinear functions to data

class `pyret.nonlinearities.`**`Sigmoid`**(*baseline=0.0*, *peak=1.0*, *slope=1.0*, *threshold=0.0*)

Bases: `sklearn.base.BaseEstimator`, `sklearn.base.RegressorMixin`, `pyret.nonlinearities.NonlinearityMixin`

`fit`(*x*, *y*, ***kwargs*)

`predict`(*x*)

class `pyret.nonlinearities.`**`Binterp`**(*nbins*, *method='linear'*, *fill_value='extrapolate'*)

Bases: `sklearn.base.BaseEstimator`, `sklearn.base.RegressorMixin`, `pyret.nonlinearities.NonlinearityMixin`

`fit`(*x*, *y*)

`predict`(*x*)

Placeholder, this method gets overwritten when fit() is called

class `pyret.nonlinearities.`**`GaussianProcess`**(***kwargs*)

Bases: `sklearn.gaussian_process.gpr.GaussianProcessRegressor`, `pyret.nonlinearities.NonlinearityMixin`

`fit`(*x*, *y*)

Fit Gaussian process regression model.

Parameters

- **X** (*array-like, shape = (n_samples, n_features)*) – Training data

- **y** (*array-like, shape = (n_samples, [n_output_dims])*) – Target values

Returns self

Return type returns an instance of self.

predict (*x*, ***kwargs*)

Predict using the Gaussian process regression model

We can also predict based on an unfitted model by using the GP prior. In addition to the mean of the predictive distribution, also its standard deviation (return_std=True) or covariance (return_cov=True). Note that at most one of the two can be requested.

Parameters

- **X** (*array-like, shape = (n_samples, n_features)*) – Query points where the GP is evaluated
- **return_std** (*bool, default: False*) – If True, the standard-deviation of the predictive distribution at the query points is returned along with the mean.
- **return_cov** (*bool, default: False*) – If True, the covariance of the joint predictive distribution at the query points is returned along with the mean

Returns

- **y_mean** (*array, shape = (n_samples, [n_output_dims])*) – Mean of predictive distribution a query points
- **y_std** (*array, shape = (n_samples,), optional*) – Standard deviation of predictive distribution at query points. Only returned when return_std is True.
- **y_cov** (*array, shape = (n_samples, n_samples), optional*) – Covariance of joint predictive distribution a query points. Only returned when return_cov is True.

3.3 spiketools

Tools for spike train analysis

Includes an object class, SpikingEvent, that is useful for detecting and analyzing firing events within a spike raster. Also provides functions for binning spike times into a histogram (*binspikes*) and a function for smoothing a histogram into a firing rate (*estfr*)

pyret.spiketools.**binspikes** (*spk*, *time*)

Bin spike times at the given resolution. The function has two forms.

Parameters

- **spk** (*array_like*) – Array of spike times
- **time** (*array_like*) – The left edges of the time bins.

Returns **bspk** – Binned spike times

Return type array_like

pyret.spiketools.**estfr** (*bspk*, *time*, *sigma=0.01*)

Estimate the instantaneous firing rates from binned spike counts.

Parameters

- **bspk** (*array_like*) – Array of binned spike counts (e.g. from binspikes)

- **time** (*array_like*) – Array of time points corresponding to bins
- **sigma** (*float, optional*) – The width of the Gaussian filter, in seconds (Default: 0.01 seconds)

Returns **rates** – Array of estimated instantaneous firing rate

Return type array_like

`pyret.spiketools.`**`detectevents`**(*spk*, *threshold=(0.3, 0.05)*)

Detects spiking events given a PSTH and spike times for multiple trials

>> events = detectevents(spikes, threshold=(0.1, 0.005))

Parameters

- **spk** (*array_like*) – An (n by 2) array of spike times, indexed by trial / condition. The first column is the set of spike times in the event and the second column is a list of corresponding trial/cell/condition indices for each spike.
- **threshold** (*(float, float), optional*) – A tuple of two floats that are used as thresholds for detecting firing events. Default: (0.1, 0.005) see *peakdet* for more info

Returns **events** – A list of 'spikingevent' objects, one for each firing event detected. See the *spikingevent* class for more info.

Return type list

`pyret.spiketools.`**`peakdet`**(*v*, *delta*, *x=None*)

Converted from MATLAB script at http://billauer.co.il/peakdet.html

Returns two arrays containing the maxima and minima of a 1D signal

Parameters

- **v** (*array_like*) – The input signal (array) to find the peaks of
- **delta** (*float*) –

  The threshold for peak detection. A point is considered a maxima (or minima) if it is at least delta larger (or smaller) than its neighboring points
- **x** (*array_like, optional*) – If given, the locations of the peaks are given as the corresponding values in *x*. Otherwise, the locations are given as indices

Returns

- **maxtab** (*array_like*) – An (N x 2) array containing the indices or locations (left column) of the local maxima in *v* along with the corresponding maximum values (right column).
- **mintab** (*array_like*) – An (M x 2) array containing the indices or locations (left column) of the local minima in *v* along with the corresponding minimum values (right column).

class `pyret.spiketools.`**`SpikingEvent`**(*start_time*, *stop_time*, *spikes*)

Bases: `object`

`jitter`()

Computes the jitter (standard deviation) in the time to first spike

>> sigma = spkevent.jitter()

`plot`(*sort=False*, *ax=None*, *color='SlateGray'*)

Plots this event, as a spike raster

>> spkevent.plot()

Parameters

- **sort** (*boolean, optional*) – Whether or not to sort by the time to first spike (Default: False)
- **ax** (*matplotlib Axes object, optional*) – If None, creates a new figure (Default: None)
- **color** (*string*) – The color of the points in the raster (Default: 'SlateGray')

sort()

Sort trial indices by the time to first spike

>> sortedspikes = spkevent.sort()

stats()

Compute statistics (mean and standard deviation) across spike counts

>> mu, sigma = spkevent.event_stats()

trial_counts()

Count the number of spikes per trial

ttfs()

Computes the time to first spike for each trial, ignoring trials that had zero spikes

>> times = spkevent.ttfs()

3.4 stimulustools

Tools for dealing with spatiotemporal stimuli

pyret.stimulustools.**upsample**(*stim*, *upsample_factor*, *time=None*)

Upsample the given stimulus by the given factor.

Parameters

- **stim** (*array_like*) – The actual stimulus to be upsampled. dimensions: (time, space, space)
- **upsample_factor** (*int*) – The upsample factor.
- **time** (*array_like, optional*) – The time axis of the original stimulus.

Returns

- **stim_us** (*array_like*) – The upsampled stimulus array
- **time_us** (*array_like*) – the upsampled time vector

pyret.stimulustools.**downsample**(*stim*, *downsample_factor*, *time=None*)

Downsample the given stimulus by the given factor.

Parameters

- **stim** (*array_like*) – The original stimulus array
- **downsample_factor** (*int*) – The factor by which the stimulus will be downsampled
- **time** (*array_like, optional*) – The time axis of the original stimulus

Returns

- **stim_ds** (*array_like*) – The downsampled stimulus array
- **time_ds** (*array_like*) – The downsampled time vector

`pyret.stimulustools.`**`slicestim`**(*stimulus*, *nsamples_before*, *nsamples_after=0*)

Slices a spatiotemporal stimulus array (over time) into overlapping frames.

Parameters

- **`stimulus`** (*`array_like`*) – The spatiotemporal or temporal stimulus to slice. Should have shape `(t, ...)`, so that the time axis is first. The ellipses indicate the spatial dimensions of the stimulus, if any.
- **`nsamples_before`** (*`int`*) – Integer number of time points before a hypothetical center. See Notes section for more details.
- **`nsamples_after`** (*`int, optional`*) – Integer number of time points before a hypothetical center. See Notes section for more details.

Returns **slices** – A view onto the original stimulus array, giving the overlapping slices of the stimulus. The full shape of the returned array is: `(stimulus.shape[0] - history + 1, history ...)`, where `history == nsamples_before + nafter`. As above, the ellipses indicate any spatial dimensions to the stimulus.

Return type array_like

Examples

```
>>> x = np.arange(15).reshape((5, 3))
>>> slicestim(x, 3)
array([[[ 0,  1,  2],
        [ 3,  4,  5]],
```

[[3, 4, 5], [6, 7, 8]],

[[6, 7, 8], [9, 10, 11]],

[[9, 10, 11], [12, 13, 14]]])

Calculate rolling mean of last dimension:

```
>>> np.mean(slicestim(x, 3), -1)
 array([[ 1.,  4.],
        [ 4.,  7.],
        [ 7., 10.],
        [10., 13.]])
```

Notes

`stimulustools.slicestim` is used to create a Toeplitz matrix from a multi-dimensional stimulus. This simplifies performing certain operations such as filtering, as it allows us to express the operation as a matrix product rather than via convolution.

However, this product only works when the sliced stimulus and filter are temporally aligned. Because `filtertools.sta` and `filtertools.revcorr` allow computing acausal components of an STA (points *after* a spike occurs), this method must also allow that in order to keep the temporal alignment.

Practically this means that one must always pass the same value for the `nsamples_after` argument as is passed to `filtertools.sta` or `filtertools.revcorr`.

pyret.stimulustools.**cov**(*stimulus*, *history*, *nsamples=None*, *verbose=False*)
Computes a stimulus covariance matrix

Warning: This is computationally expensive for large stimuli

Parameters

- **stimulus** (*array_like*) – The spatiotemporal or temporal stimulus to slices. Should have shape (t, . . .), where the ellipses indicate any spatial dimensions.
- **history** (*int*) – Integer number of time points to keep in each slice.

Returns **stim_cov** – Covariance matrix

Return type array_like

pyret.stimulustools.**flat2d**(*x*)
Flattens all dimensions after the first of the given array

Useful for collapsing spatial dimensions in a spatiotemporal stimulus or filter.

3.5 visualizations

Visualization functions for displaying spikes, filters, and cells.

pyret.visualizations.**raster**(**args*, ***kwargs*)
Plot a raster of spike times.

Parameters

- **spikes** (*array_like*) – An array of spike times.
- **labels** (*array_like*) – An array of labels corresponding to each spike in spikes. For example, this can indicate which cell or trial each spike came from. Spike times are plotted on the x-axis, and labels on the y-axis.
- **title** (*string, optional*) – An optional title for the plot (Default: 'Spike raster').
- **marker_string** (*string, optional*) – The marker string passed to matplotlib's plot function (Default: 'ko').
- **ax** (*matplotlib.axes.Axes instance, optional*) – An optional axes onto which the data is plotted.
- **fig** (*matplotlib.figure.Figure instance, optional*) – An optional figure onto which the data is plotted.
- **kwargs** (*dict*) – Optional keyword arguments are passed to matplotlib's plot function.

Returns

- **fig** (*matplotlib.figure.Figure*) – Matplotlib Figure object into which raster is plotted.
- **ax** (*matplotlib.axes.Axes*) – Matplotlib Axes object into which raster is plotted.

pyret.visualizations.**psth**(**args*, ***kwargs*)
Plot a PSTH from the given spike times.

Parameters

- **spikes** (*array_like*) – An array of spike times.

- **`trial_length`** (*`float`*) – The length of each trial to stack, in seconds. If None (the default), a single PSTH is plotted. If a float is passed, PSTHs from each trial of the given length are averaged together before plotting.
- **`binsize`** (*`float`*) – The size of bins used in computing the PSTH.
- **`ax`** (*`matplotlib.axes.Axes instance, optional`*) – An optional axes onto which the data is plotted.
- **`fig`** (*`matplotlib.figure.Figure instance, optional`*) – An optional figure onto which the data is plotted.
- **`kwargs`** (*`dict`*) – Keyword arguments passed to matplotlib's `plot` function.

Returns

- **fig** (*matplotlib.figure.Figure*) – Matplotlib Figure object into which PSTH is plotted.
- **ax** (*matplotlib.axes.Axes*) – Matplotlib Axes object into which PSTH is plotted.

`pyret.visualizations.`**`raster_and_psth`**`(`**args*, ***kwargs*`)`

Plot a spike raster and a PSTH on the same set of axes.

Parameters

- **`spikes`** (*`array_like`*) – An array of spike times.
- **`trial_length`** (*`float`*) – The length of each trial to stack, in seconds. If None (the default), all spikes are plotted as part of the same trial.
- **`binsize`** (*`float`*) – The size of bins used in computing the PSTH.
- **`ax`** (*`matplotlib.axes.Axes instance, optional`*) – An optional axes onto which the data is plotted.
- **`fig`** (*`matplotlib.figure.Figure instance, optional`*) – An optional figure onto which the data is plotted.
- **`kwargs`** (*`dict`*) – Keyword arguments to matplotlib's `plot` function.

Returns

- **fig** (*matplotlib.figure.Figure*) – Matplotlib Figure instance onto which the data is plotted.
- **ax** (*matplotlib.axes.Axes*) – Matplotlib Axes instance onto which the data is plotted.

`pyret.visualizations.`**`spatial`**`(`**args*, ***kwargs*`)`

Plot the spatial component of a full linear filter.

If the given filter is 2D, it is assumed to be a 1D spatial filter, and is plotted directly. If the filter is 3D, it is decomposed into its spatial and temporal components, and the spatial component is plotted.

Parameters

- **`filt`** (*`array_like`*) – The filter whose spatial component is to be plotted. It may have temporal components.
- **`dx`** (*`float, optional`*) – The spatial sampling rate of the STA, setting the scale of the x- and y-axes.
- **`maxval`** (*`float, optional`*) – The value to use as minimal and maximal values when normalizing the colormap for this plot. See `plt.imshow()` documentation for more details.
- **`ax`** (*`matplotlib Axes object, optional`*) – The axes on which to plot the data; defaults to creating a new figure.

Returns

- **fig** (*matplotlib.figure.Figure*) – The figure onto which the spatial STA is plotted.
- **ax** (*matplotlib Axes object*) – Axes into which the spatial STA is plotted.

`pyret.visualizations.`**`temporal`**`(`**args*, ***kwargs*`)`

Plot the temporal component of a full linear filter.

If the given linear filter is 1D, it is assumed to be a temporal filter, and is plotted directly. If the filter is 2 or 3D, it is decomposed into its spatial and temporal components, and the temporal component is plotted.

Parameters

- **`time`** (*`array_like`*) – A time vector to plot against.
- **`filt`** (*`array_like`*) – The full filter to plot. May be than 1D, but must match in size along the first dimension with the `time` input.
- **`ax`** (*`matplotlib Axes object, optional`*) – the axes on which to plot the data; defaults to creating a new figure

Returns

- **fig** (*matplotlib.figure.Figure*) – The figure onto which the temoral STA is plotted.
- **ax** (*matplotlib Axes object*) – Axes into which the temporal STA is plotted

`pyret.visualizations.`**`plot_sta`**`(`*time*, *sta*, *dx=1.0*`)`

Plot a linear filter.

If the given filter is 1D, it is direclty plotted. If it is 2D, it is shown as an image, with space and time as its axes. If the filter is 3D, it is decomposed into its spatial and temporal components, each of which is plotted on its own axis.

Parameters

- **`time`** (*`array_like`*) – A time vector to plot against.
- **`dx`** (*`float, optional`*) – The spatial sampling rate of the STA, setting the scale of the x- and y-axes.
- **`sta`** (*`array_like`*) – The filter to plot.

Returns

- **fig** (*matplotlib.figure.Figure*) – The figure onto which the STA is plotted.
- **ax** (*matplotlib Axes object*) – Axes into which the STA is plotted

`pyret.visualizations.`**`play_sta`**`(`*sta*, *repeat=True*, *frametime=100*, *cmap='seismic_r'*, *clim=None*, *dx=1.0*`)`

Plays a spatiotemporal spike-triggered average as a movie.

Parameters

- **`sta`** (*`array_like`*) – Spike-triggered average array, shaped as `(nt, nx, ny)`.
- **`repeat`** (*`boolean, optional`*) – Whether or not to repeat the animation (default is True).
- **`frametime`** (*`float, optional`*) – Length of time each frame is displayed for in milliseconds (default is 100).
- **`cmap`** (*`string, optional`*) – Name of the colormap to use (Default: `'seismic_r'`).
- **`clim`** (*`array_like, optional`*) – 2-element color limit for animation; e.g. [0, 255].

- **dx** (*float, optional*) – The spatial sampling rate of the STA, setting the scale of the x- and y-axes.

Returns **anim**

Return type matplotlib animation object

`pyret.visualizations.ellipse(*args, **kwargs)`

Plot an ellipse fitted to the given receptive field.

Parameters

- **filt** (*array_like*) – A linear filter whose spatial extent is to be plotted. If this is 2D, it is assumed to be the spatial component of the receptive field. If it is 3D, it is assumed to be a full spatiotemporal receptive field; the spatial component is extracted and plotted.
- **sigma** (*float, optional*) – Determines the threshold of the ellipse contours. This is the standard deviation of a Gaussian fitted to the filter at which the contours are plotted. Default is 2.0.
- **alpha** (*float, optional*) – The alpha blending value, between 0 (transparent) and 1 (opaque) (Default: 0.8).
- **fc** (*string, optional*) – Ellipse face color. (Default: none)
- **ec** (*string, optional*) – Ellipse edge color. (Default: black)
- **lw** (*int, optional*) – Line width. (Default: 3)
- **dx** (*float, optional*) – The spatial sampling rate of the STA, setting the scale of the x- and y-axes.
- **ax** (*matplotlib Axes object, optional*) – The axes onto which the ellipse should be plotted. Defaults to a new figure.

Returns

- **fig** (*matplotlib.figure.Figure*) – The figure onto which the ellipse is plotted.
- **ax** (*matplotlib.axes.Axes*) – The axes onto which the ellipse is plotted.

`pyret.visualizations.plot_cells(*args, **kwargs)`

Plot the spatial receptive fields for multiple cells.

Parameters

- **cells** (*list of array_like*) – A list of spatiotemporal receptive fields, each of which is a spatiotemporal array.
- **dx** (*float, optional*) – The spatial sampling rate of the STA, setting the scale of the x- and y-axes.
- **ax** (*matplotlib Axes object, optional*) – The axes onto which the ellipse should be plotted. Defaults to a new figure.

Returns

- **fig** (*matplotlib.figure.Figure*) – The figure onto which the ellipses are plotted.
- **ax** (*matplotlib.axes.Axes*) – The axes onto which the ellipses are plotted.

`pyret.visualizations.play_rates(rates, patches, num_levels=255, time=None, repeat=True, frametime=100)`

Plays a movie representation of the firing rate of a list of cells, by coloring a list of patches with a color proportional to the firing rate. This is useful, for example, in conjunction with `plot_cells`, to color the ellipses fitted to a set of receptive fields proportional to the firing rate.

Parameters

- **`rates`** (*`array_like`*) – An `(N, T)` matrix of firing rates. `N` is the number of cells, and `T` gives the firing rate at a each time point.
- **`patches`** (*`list`*) – A list of `N` matplotlib patch elements. The facecolor of these patches is altered according to the rates values.

Returns **anim** – The object representing the full animation.

Return type matplotlib.animation.Animation

CHAPTER 4

Changelog

A list of new features, improvements, and bug-fixes in each release.

4.1 v0.6 (Active)

4.1.1 New features

- Adds the ability to extend temporal filters to be acausal (past the time of the spike)
- Adds an `RBF` class for estimating a nonlinearity using tiled radial basis functions.

4.1.2 API changes

- Removes outdated `stimulustools.rolling_window` method.
- In fixing a bug in `linear_response`, then method now returns an array of the same shape as the stimulus input, rather than one shorter by the length of the filter whose response is computed.

4.1.3 Bug fixes

- Fixes a bug in the `Sigmoid` nonlinearity due do shuffled dictionary keys
- Fixes bug in `linear_response`, which was supposed to take a filter, but actually took a reverse-correlation.
- Fixes incorrect documentation for `stimulustools.slicestim`.

4.2 v0.5 (17 Nov 2016)

4.2.1 New features

- Better handling of low-rank STA component signs in `filtertools.lowranksta`.
- Functionality for embedding STA animations into HTML, via `visualizations.anim_to_html()`.
- New classes for estimating nonlinearities: `Binterp`, `Sigmoid` and `GaussianProcess`. These follow the `scikit-learn` interface, meaning they have `fit()` and `predict()` methods, which return `self`.

4.2.2 API changes

- Renamed `filtertools.getsta` -> `filtertools.sta`
- Renamed `filtertools.getste` -> `filtertools.ste`
- Renamed `filtertools.getstc` -> `filtertools.stc`
- Renamed `visualizations.rasterandpsth` -> `visualizations.raster_and_psth`
- Renamed `visualizations.plotcells` -> `visualizations.plot_cells`
- Renamed `visualizations.plotsta` -> `visualizations.plot_sta`
- Renamed `visualizations.playrates` -> `visualizations.play_rates`
- Renamed `visualizations.playsta` -> `visualizations.play_sta`
- `spiketools.binspikes` and `spiketools.estfr` no longer return the time axis. Only the binned spikes and firing rate are returned, respectively.
- Removed `containers` module.
- `filtertools.rolling_window` has been moved to the `stimulustools` module, and is renamed `slicestim`. `rolling_window` is an alias for `slicestim`, for the time being, which raises a warning about future deprecation.
- Renamed `stimulustools.stimcov` -> `stimulustools.cov`.
- Renamed `stimulustools.upsample_stim` -> `stimulustools.upsample`.
- Renamed `stimulustools.downsample_stim` -> `stimulustools.downsample`.

4.3 v0.4 (December 11 2015)

4.3.1 New features

- Adds a *containers* module that contains two classes, and *Experiment* and a *Filter* class, for managing stimuli and spikes (*Experiment*) and spike-triggered averages (*Filter*).
- New and improved ellipse and contour fitting code (*filtertools.rfsize*, *filtertools.get_ellipse*, *visualizations.ellipse*)
- New function *filtertools.resample* which is a thin wrapper around *scipy.signal.resample*

4.3.2 API changes

- Flipped the expected dimensions of stimuli and filters to have the temporal dimension first. E.g. functions now expect (time, space, space) or (time, space) instead of (space, space, time) or (space, time).
- Changes the default value of the argument in *rolling_window* to *time_axis=0*, to be consistent with the rest of pyret (after the flipped dimensions switch)
- Removes the *prinangles* function (does not really belong in the *filtertools* module, or even in pyret at all)
- Updated *pyret.plotsta* function
- Reworked *filtertools.getste* to be a generator, and modified *getsta* and *getstc* to consume that generator.

4.3.3 Issues closed

- #62 bug in filtertools.decompose.
- #63 better ellipse fitting tools.
- #60 custom classes for filter.
- #53 simplifying filtertools.

4.4 v0.3 (June 25 2015)

4.4.1 API changes

- Changed the *filtertools* module's *getste*, *getsta*, and *getstc* to use generators. The *getste* function now returns a generator that yields samples from the spike-triggered ensemble, while *getsta* and *getstc* consume that generator in order to compute their results.

4.5 v0.2 (February 1 2015)

This is a major release with a number of API changes, enhancements, and bug fixes.

The main focus has been on adding thorough documentation of all the packages and functions available.

4.5.1 API changes

- Changed the `filterlength`, `numSamples` and `spatialSmoothing` optional arguments to `filter_length`, `num_samples` and `spatialSmoothing` in `filtertools.py`
- Changed the `numTrials` to `num_trials` in `spiketools.py`
- Changed the `triallength`, `spatialFrame`, `temporalFilter` and `boxdims` optional arguments to `trial_length`, `spatial_filter`, `temporal_filter` and `box_dims` in `visualizations.py`
- Changed the `stim` paramteer to `stimulus` in `stimulustools.py`
- Added a function `sample(rate)` to `spiketools.py` which draws spikes from a Poisson distribution with the given rate.
- Renamed the `spikingevent` class in `spiketools.py` to `SpikingEvent`

- Renamed the attributes `startTime`, `stopTime` and functions `trialCounts`, `eventStats` of `SpikingEvent` to `start_time`, `stop_time` and `trial_counts`, `event_stats`
- Moved the `peakdet` function from the `peakdetect.py` module to `spiketools.py`. Removed the `peakdetect.py` module
- Renamed the functions `getellipseparams` and `getellipse` to `get_ellipse_params` and `fit_ellipse` in `filtertools.py`
- Renamed the functions `upsamplestim` and `downsamplestim` to `upsample_stim` and `downsample_stim` in `stimulustools.py`

4.5.2 General package changes

- Removed the (Igor and Baccus lab specific) module `binary.py`
- Documentation via `sphinx` is included in the `doc/` folder

4.5.3 Known issues

- Installing with `pip` has not been tested.
- Installing with `python setup.py install` is known to not work on some machines.

An example retinal ganglion cell receptive field visualized using pyret:

Please report any bugs you encounter through the GitHub issue tracker.

genindex of all functions.

Python Module Index

Index