Neuroinformatics Toolbox Documentation

Release 0.1

Jacob Huth

November 19, 2013

CONTENTS

1 model Package **3**

1.1 pointprocess Module 3
1.2 designmatrix Module 8
1.3 ip Module 10
1.4 net_sim Module 14
1.5 create_design_matrix_vk Module 16
1.6 create_splines Module 17
1.7 backend_elasticnet Module 17
1.8 backend_glm Module 18

2 tools Package **19**

2.1 bootstrap Module 19
2.2 progressbar Module 20
2.3 project Module 20
2.4 html_view Module 25
2.5 strap Module 26
2.6 statcollector Module 28

3 data Package **31**

3.1 data Module 31
3.2 decoding_data Module 33
3.3 monkey Module 34

4 List of ToDo Items **35**

5 Indices and tables **37**

Python Module Index **39**

Index **41**

The NI Toolbox contains python versions of commonly used functions to deal with spike data. Get the pdf version here

Contents:

CHAPTER

ONE

MODEL PACKAGE

1.1 `pointprocess` Module

class `ni.model.pointprocess.`**`PointProcess`**(*dimensionality*)

A Point Process container.

Usually generated by loading from a file or via `ni.model.pointprocess.createPoisson()`

`addSpike`(*t*)

adds a spike to the point process, if it falls in the allowed range.

`getCounts`()

Gives a (in most cases binary) time series of the point process.

`getProbability`(*t_from*, *t_to*)

Undocumented

`plot`(*y=0*, *marker='|'*)

Plots the pointprocess as points at line *y*.

marker determines the color and shape of the marker. Default is a vertical line '|'

`plotGaussed`(*width*)

Plots the pointprocess as a smoothed time series

`ni.model.pointprocess.`**`PointProcessFromSpikeTimes`**(*times*)

class `ni.model.pointprocess.`**`SimpleFiringRateModel`**

Uses just the firing rate as a predictor

`compare`(*Data*, *Prediction*)

`fit`(*data*)

`loglikelihood`(*Data*, *Prediction*)

`predict`(*Data*)

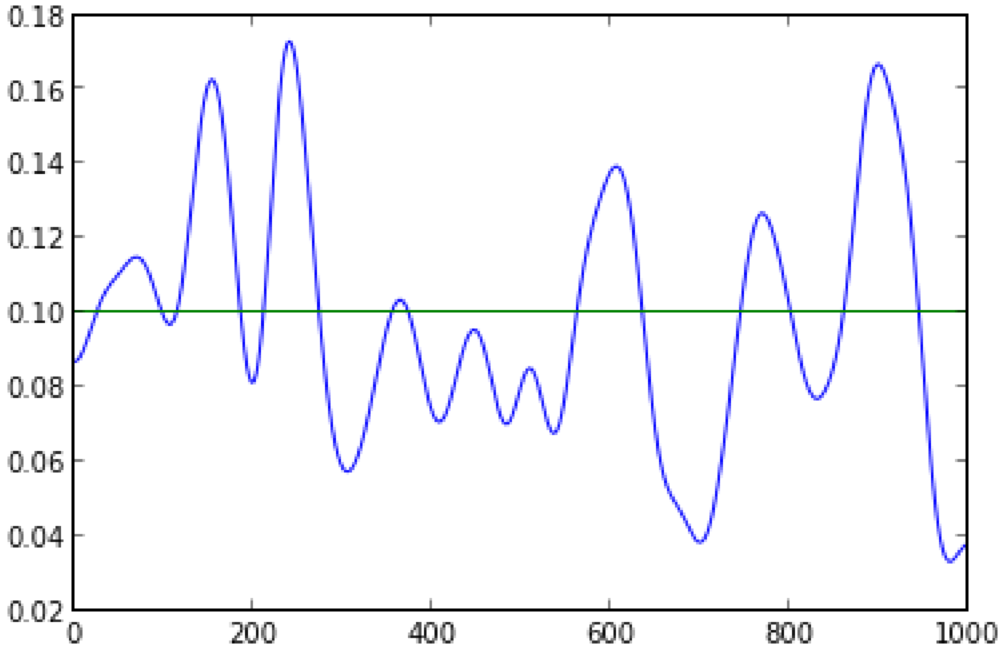
`ni.model.pointprocess.`**`createPoisson`**(*p*, *l*)

This generates a spike sequence of length *l* according to either a fixed firing rate *p*, or a repeated sequence of firing rates if *type(p) == np.ndarray*.

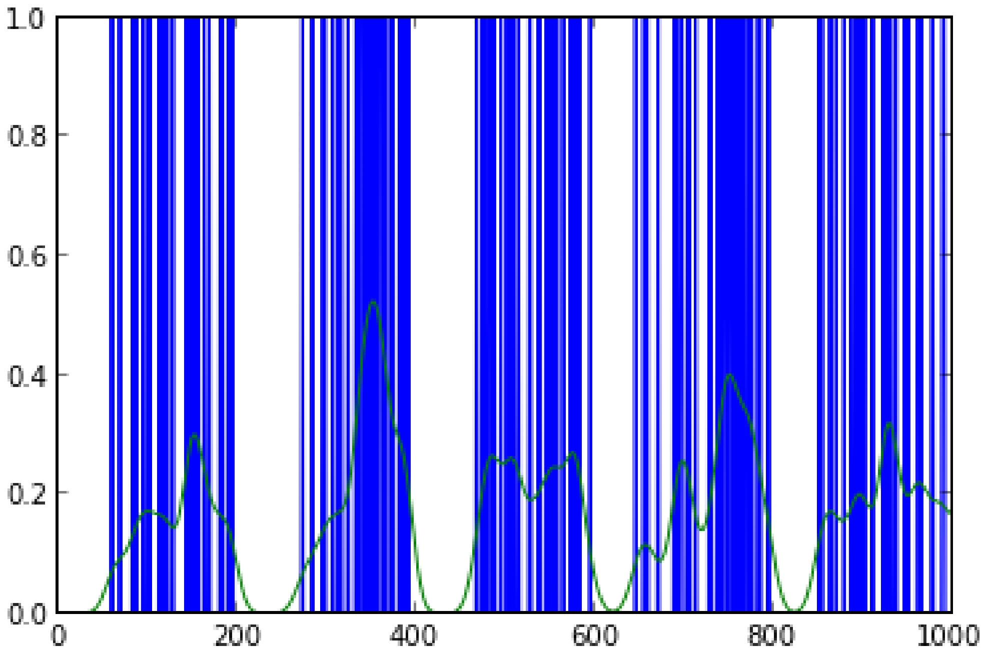
It creates a `ni.model.pointprocess.PointProcess`

Example 1:

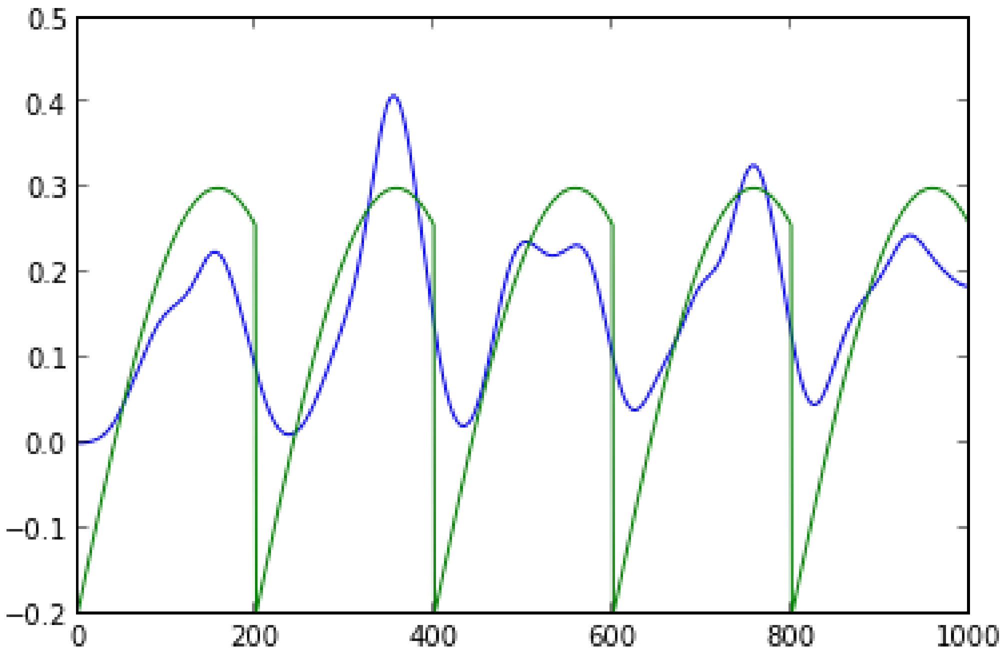
```
p1 = ni.model.pointprocess.createPoisson(0.1,1000)
p1.plotGaussed(20)
plot(p1.frate)
```



```
p2 = ni.model.pointprocess.createPoisson(sin(numpy.array(range(0,200))*0.01)*0.5- 0.2,1000)
p2.plot()
p2.plotGaussed(10)
```



```
p2.plotGaussed(20)
plot(p2.frate)
```

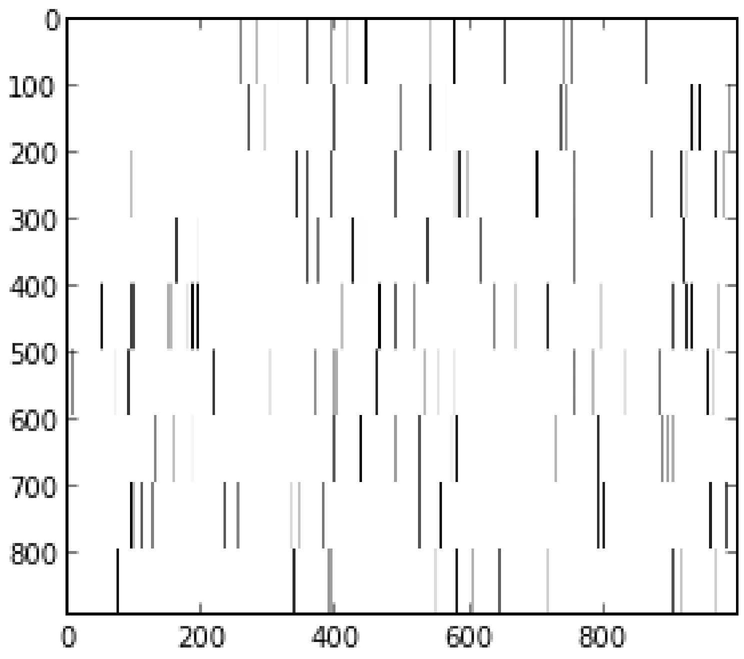


Example with multiple channels:

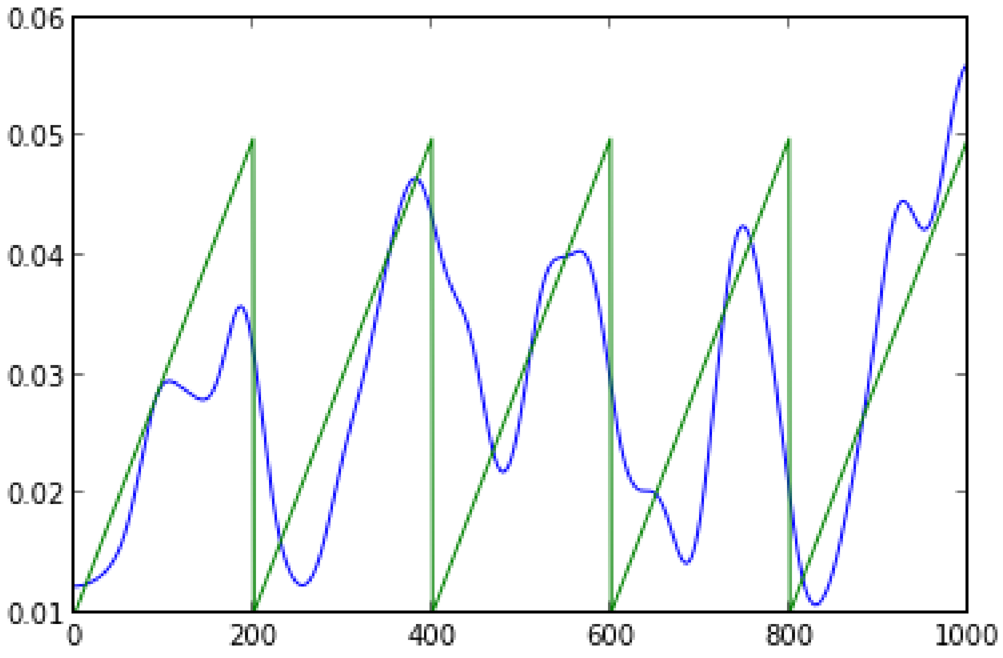
```
frate = (numpy.array(range(0,200))*0.001)*0.2+0.01
channels = 9

dists = [ni.model.pointprocess.createPoisson(frate,1000) for i in range(0,channels)]
#for i in range(0,9): dists[i].plotGaussed(10)
import itertools
spks = np.array([dists[i].getCounts() for i in range(0,channels) for j in range(0,99) ])
imshow(-1*spks)
set_cmap('gray')
```

Will generate:

```
(A plot of spikes)
```



```
ni.model.pointprocess.plotGaussed(np.array([dists[i].getCounts() for i in range(0,channels)]).me
plot(dists[0].frate)
```

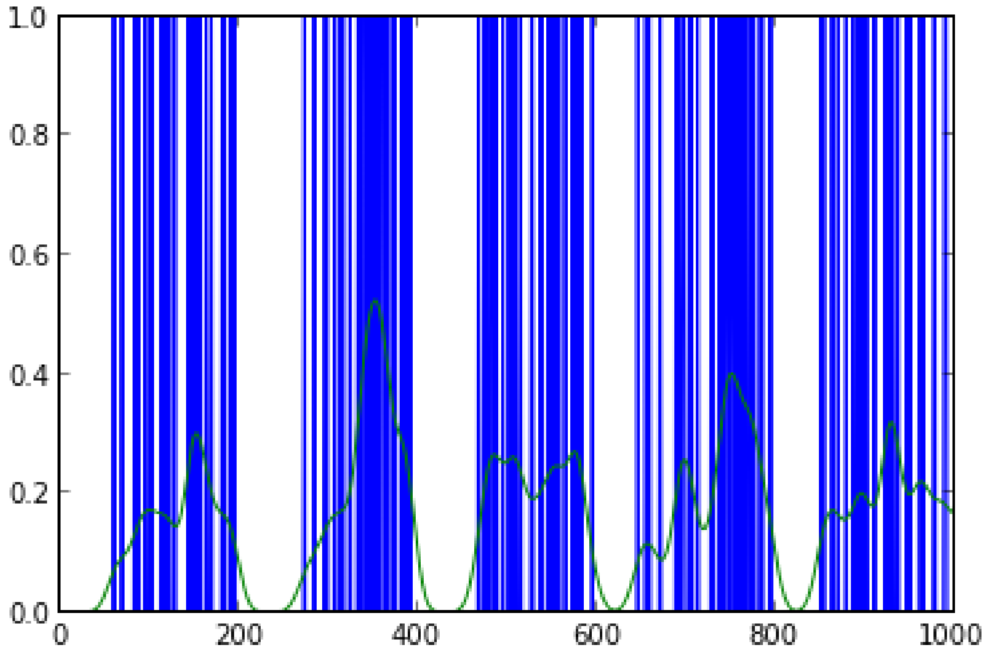


ni.model.pointprocess.**getBinary**(*spikes*, *min_length=1*)
: Gives back a binary array from an array of spike times. The maximum for each bin is 1.

ni.model.pointprocess.**getCounts**(*spikes*)
: Gives back an array of spike counts from an array of spike times. If the output is suppsed to be a Binomial, use *getBinary* instead.

ni.model.pointprocess.**interspike_interval**(*spikes_a*, *spikes_b=False*) ni.model.pointprocess.**plotGaussed**(*data*, *width*)

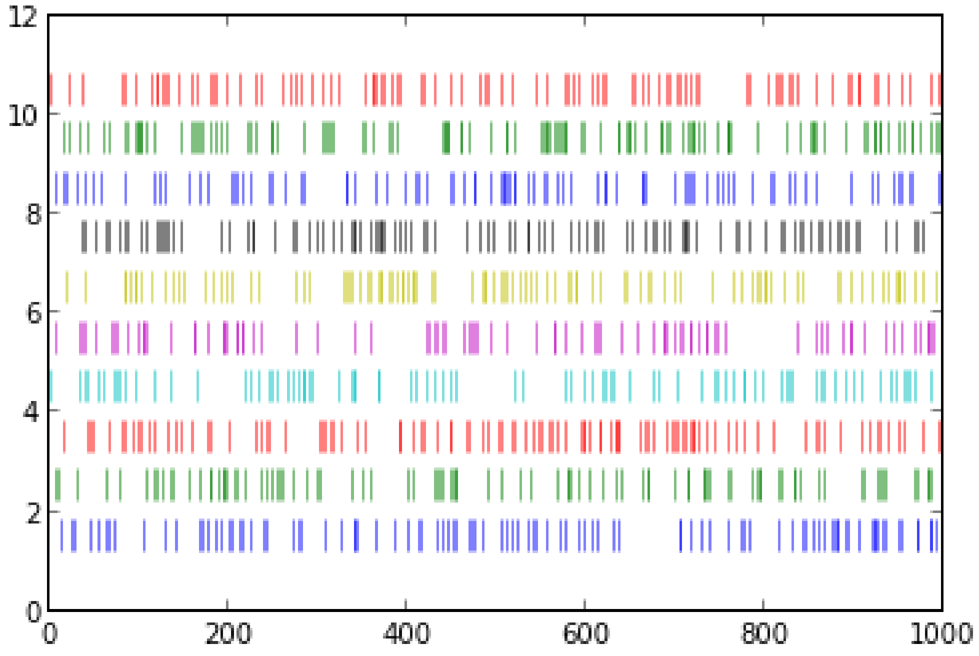
```
p2 = ni.model.pointprocess.createPoisson(sin(numpy.array(range(0,200))*0.01)*0.5- 0.2,1000)
p2.plot()
p2.plotGaussed(10)
```



ni.model.pointprocess.**plotMultiSpikes**(*spikes*)

- *spikes* is a binary 2d matrix

Generates something like:



`ni.model.pointprocess.`**`reverse_correlation`**(*spikes_a*, *spikes_b=False*)

1.2 `designmatrix` Module

class `ni.model.designmatrix.`**`AdaptiveRateComponent`**(*header='rate'*, *rate=False*, *knots=10*, *length=1000*, *kernel=False*)

Bases: `ni.model.designmatrix.Component`

Rate Design Matrix Component

header: name of the kernel component rate: a rate function that determines knots: Number of knots length: length of the component. Will be multiplied

kernel: use this kernel instead of a newly created one

`getSplines`(*data=[ ]*)

class `ni.model.designmatrix.`**`Component`**(*header='Undefined'*, *kernel=0*)

Bases: `ni.tools.pickler.Picklable`

Design Matrix Component

header: name of the kernel component kernel: kernel that will be tiled to fill the design matrix

`getSplines`(*data=[ ]*)

class `ni.model.designmatrix.`**`DesignMatrix`**(*length*, *width=1*)

Bases: `ni.tools.pickler.Picklable` Deprecated since version 0.1.

`add`(*splines*, *header*)

`addLinSpline`(*knots*, *header*, *length=0*)

`addLogSpline`(*knots*, *header*, *length=0*)

`clip`()

`get`(*filt*)

`getIndex`(*filt*)

`getMask`(*filt*)

`plot`(*filt=''*)

`setMask`(*mask*)

class `ni.model.designmatrix.`**`DesignMatrixTemplate`**(*length*, *trial_length=0*)

Bases: `ni.tools.pickler.Picklable`

Most important class for Design Matrices

Uses components that are then combined into an actual design matrix:

```
>>> DesignMatrixTemplate(data.nr_trials * data.time_bins)
>>> kernel = cs.create_splines_logspace(self.configuration.history_length, self.configuration.kn
>>> design_template.add(designmatrix.HistoryComponent('autohistory', kernel=kernel))
>>> design_template.add(designmatrix.HistoryComponent('crosshistory'+str(2), channel=2, kernel =
>>> design_template.add(designmatrix.RateComponent('rate',self.configuration.knots_rate,trial_le
>>> design_template.add(designmatrix.Component('constant',np.ones((1,1))))
>>> design_template.combine(data)
```

`add`(*component*)

`combine`(*data*)

get (*filt*)

getIndex (*filt*)

getMask (*filt*)

get_components (*filt*)

setMask (*mask*)

class `ni.model.designmatrix.`**HistoryComponent** (*header='autohistory'*, *channel=0*, *history_length=100*, *knot_number=4*, *order_flag=1*, *kernel=False*, *delete_last_spline=True*)

Bases: `ni.model.designmatrix.Component`

History Design Matrix Component

Will be convolved with spikes before fitting

header: name of the kernel component channel: which channel the kernel should be convolved with (default 0) history_length: length of the kernel knot_number: number of knots (will be logspaced) order_flag: default 0 (no higher order interactions)

kernel: use this kernel instead of a newly created one

Atm only order 1 interactions

getSplines (*channels=*[ ])

class `ni.model.designmatrix.`**HistoryDesignMatrix** (*spikes*, *history_length=100*, *knot_number=5*, *order_flag=1*, *kernel=False*)

Internal helper class - will be turned into a function

class `ni.model.designmatrix.`**RateComponent** (*header='rate'*, *knots=10*, *length=1000*, *kernel=False*)

Bases: `ni.model.designmatrix.Component`

Rate Design Matrix Component

header: name of the kernel component knots: Number of knots length: length of the component. Will be multiplied

kernel: use this kernel instead of a newly created one

getSplines (*data=*[ ])

class `ni.model.designmatrix.`**SecondOrderHistoryComponent** (*header='autohistory'*, *channel_1=0*, *channel_2=0*, *history_length=100*, *knot_number=4*, *order_flag=1*, *kernel_1=False*, *kernel_2=False*, *delete_last_spline=True*)

Bases: `ni.model.designmatrix.Component`

History Design Matrix Component with Second Order Kernels

Will be convolved with spikes before fitting

header: name of the kernel component channel: which channel the kernel should be convolved with (default 0) history_length: length of the kernel knot_number: number of knots (will be logspaced) order_flag: default 0 (no higher order interactions)

kernel: use this kernel instead of a newly created one

Atm only order 1 interactions

getSplines (*channels=[ ]*, *get_1d_splines=False*, *beta=False*)

`ni.model.designmatrix.`**convolve_spikes** (*spikes*, *kernel*)

`ni.model.designmatrix.`**convolve_spikes_2d** (*spikes_a*, *spikes_b*, *kernel_a*, *kernel_b*)

1.3 `ip` Module

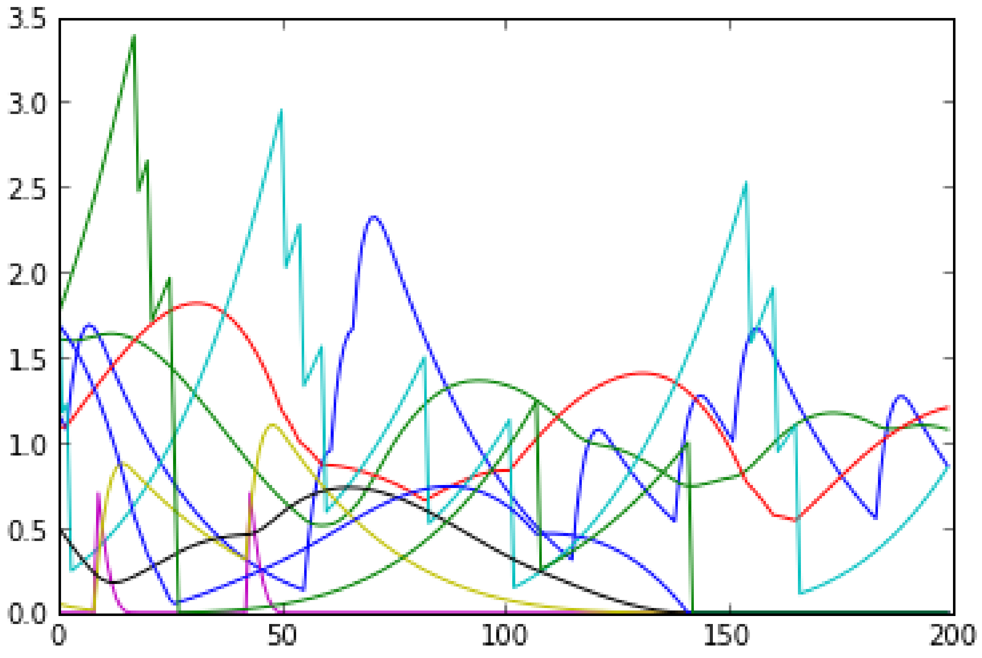
1.3.1 Inhomogeneous Pointprocess Generalized Linear Model

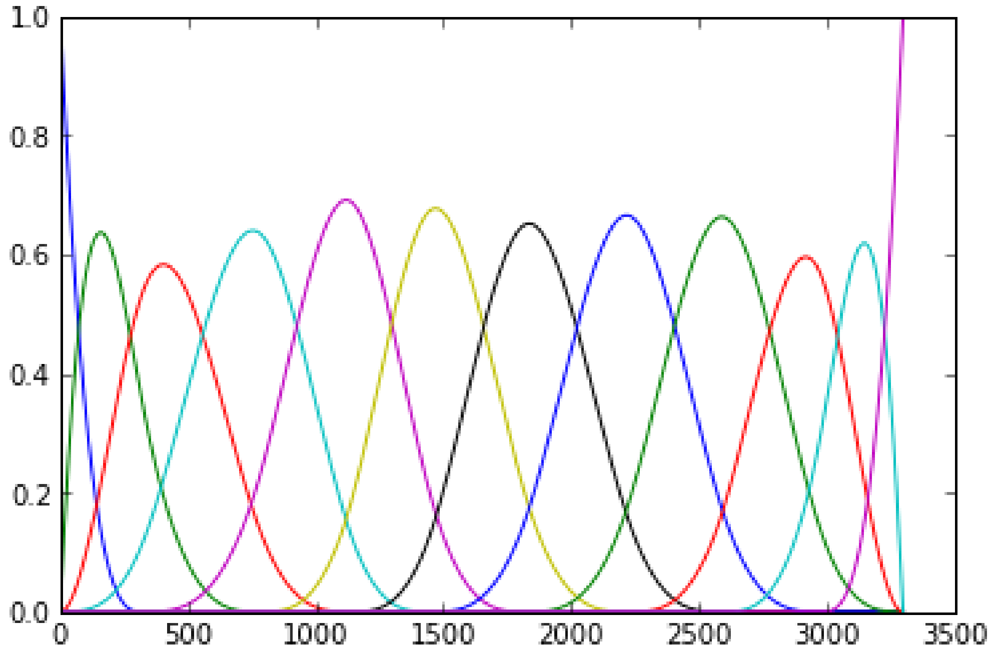
Adapted from FMTP by Robert Costa

A generalized linear model predicts a variable Y with a linear predictor and a link function. The linear predictor of the form:

$$\eta = X \cdot \beta$$

Where **X** is a matrix consisting of rows of values that correspond to a specific point in time of the modeled process. Each row my model a certain aspect (ie. time in trial, time after spike of some neuron) and is then weighted by the corresponding β parameter value. Each aspect may be scaled arbitrarily and shifted. This weighted matrix is then added up into a firing probability that is passed on to the link function.





The link function in our case of pointprocesses (ie. a poisson, bernoulli or binomial distribution, depending on notation) it is either the *log* or *logit* function ($ln\left(\frac{\mu}{(1-\mu)}\right)$).

Uses one of two backends `ni.model.backend_glm` and `ni.model.backend_elasticnet`

class `ni.model.ip.`**`Configuration`**(*c=False*)

Bases: `ni.tools.pickler.Picklable`

The following values are the defaults used:

self.backend = "glm"

The backend used. Valid options: "glm" and "elasticnet"

self.history_length = 100

Length of the history kernel

self.knot_number = 3

Number of knots in ?the history kernel?

self.order_flag = 2

Something

Todo

Find out what this is

self.knots_rate = 10

Knots of the firing rate kernel (knots/second)

Look at the [source] for a full list of defaults.

class `ni.model.ip.`**`FittedModel`**(*model*)

Bases: `ni.tools.pickler.Picklable`

When initialized via Model.fit() it contains a copy of the configuration, a link to the model it was fitted from and fitting parameters:

FittedModel. **fit**

modelFit Output

FittedModel. **design**

The DesignMatrix used. Use *design.matrix* for the actual matrix or design.get('...') to extract only the rows that correspond to a keyword.

`compare`(*data*)
Using the model this will predict a firing probability function according to a design matrix.

Returns:

Deviance_all: dv, **LogLikelihood_all**: ll, **Deviance**: dv/nr_trials, **LogLikelihood**: ll/nr_trials, **llf**: Likelihood function over time **ll**: np.sum(ll)/nr_trials

`complexity`

`dumps`()

`family_fitted_function`(*p*)
only implemented family: Binomial

`firing_rate_model`()

`generate`(*bins=-1*)
Generates new spike trains from the extracted staistics

Currently uses rate model and autohistory.

bins

How many bins should be generated (should be multiples of trial_length)

`getParams`()

`getPvalues`()

`history_model`(*n='autohistory'*)
TODO: sort out what is saved where

`html_view`()

`plotParams`(*x=-1*)

`plot_firing_rate_model`()

`plot_prototypes`()

`predict`(*data*)
Using the model this will predict a firing probability function according to a design matrix.

`prototypes`()

`pvalues_by_component`()

`read_pickle`(*path*)

`to_pickle`(*path*)

class `ni.model.ip.`**`Model`**(*configuration=None, nr_bins=0*)
Bases: `ni.tools.pickler.Picklable`

`backend`

compare (*data*, *p*, *nr_trials=1*)

dm (*in_spikes*, *design=False*)
Creates a design matrix from data and self.design

in_spikes *ni.data.data.Data* instance

fit (*data=None*, *beta=None*, *x=None*, *dm=None*, *nr_trials=None*)
Fits the model

in_spikes *ni.data.data.Data* instance

example:

```
from scipy.ndimage import gaussian_filter
import ni
model = ni.model.ip.Model(ni.model.ip.Configuration({'crosshistory':False}))
data = ni.data.monkey.Data()
data = data.condition(0).trial(range(int(data.nr_trials/2)))
dm = model.dm(data)
x = model.x(data)
from sklearn import linear_model
betas = []
fm = model.fit(data)
betas.append(fm.beta)
print "fitted."
for clf in [linear_model.LinearRegression(), linear_model.RidgeCV(alphas=[0.1, 1.0, 10.0])]:
        clf.fit(dm,x)
        betas.append(clf.coef_)

        figure()
        plot(clf.coef_.transpose(),'.')
        title('coefficients')
        prediction = np.dot(dm,clf.coef_.transpose())
        figure()
        plot(prediction)
        title('prediction')
        ll = x * log(prediction) + (len(x)-x)*log(1-prediction)
        figure()
        plot(ll)
        title('ll')
        print np.sum(ll)
```

fit_with_design_matrix (*fittedmodel*, *spike_train_all_trial*, *dm*)

generateDesignMatrix (*data*, *trial_length*)

html_view ()

predict (*beta*, *data*)

read_pickle (*path*)

to_pickle (*path*)

x (*in_spikes*)

class `ni.model.ip.`**MultiChannelModel** (*configuration={}*)
Bases: `ni.tools.pickler.Picklable`

append (*m*)

ni.model.ip.**generate_to_file**(*path*, *data*, *eval_trials*, *use_cells=[0]*, *eval_bootstrap_repetitions=10*)

This function fits two models and generates data for each saved to path + 'data0.pkl' and path + 'data1.pkl'

Todo

split into more usefull and modular functions

1.4 net_sim Module

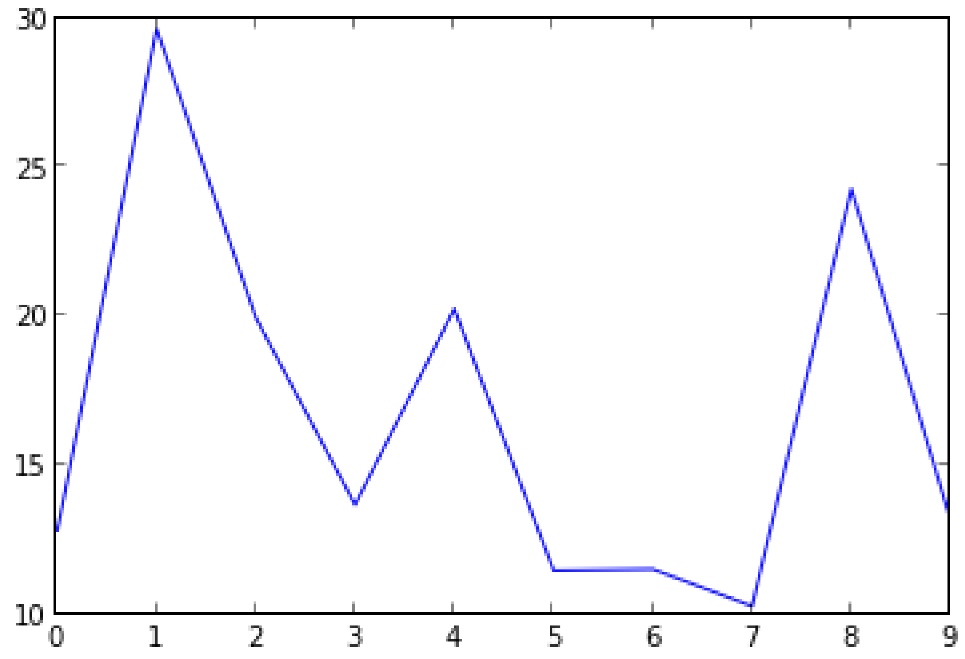
The Net Simulator is divided into a Configuration, Net and a Result object.

After configuration of the network it can be instantiated by calling *Net(conf)* with a valid configuration *conf*. This creates eg. random connectivity so that the simulation with the same network can be repeated multiple times.

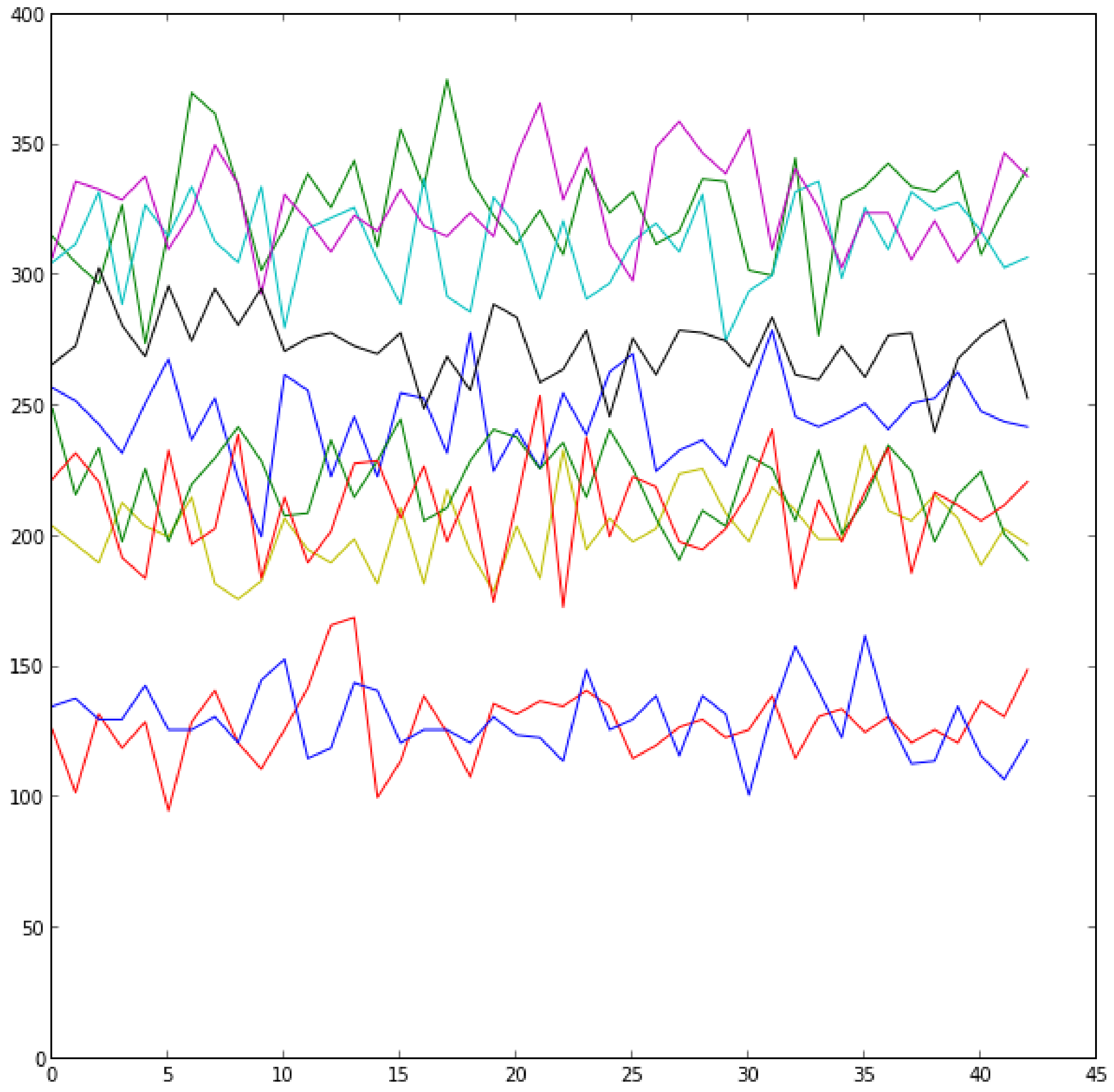
Todo

Add options for random number generator seeds, so that the *exact* same trial can be run over and over again.

```
c = ni.model.net_sim.SimulationConfiguration()
c.Nneur = 10
net = ni.model.net_sim.Net(c)
print net
net.plot_firing_rates()

'ni.model.net_sim' Simulation Setup
Timerange: (250, 10250)
 10 channels with firing rates:
        [12.815928361, 29.6328550796, 19.9415819867, 13.6710936491, 20.242131795, 11.4661487294, 11.5
Firing Rates plot
```



```
for i in range(1,11):
    print i
    res1 = net.simulate()
    res1.plot_firing_rates()

plot(numpy.array([r.num_spikes_per_channel for r in net.results]))
plot([0]*len(net.results))
```



class `ni.model.net_sim.`**`Net`**(*config*)

Undocumented

`load`(*filename*)

Undocumented

`plot_firing_rates`()

Undocumented

plot_interaction()
Undocumented

save(*filename*)
Undocumented

simulate()
Undocumented

class ni.model.net_sim.**SimulationConfiguration**
Undocumented

class ni.model.net_sim.**SimulationResult**
Undocumented

plot()
Undocumented

plot_firing_rates()
Undocumented

plot_firing_rates_per_channel()
Undocumented

stopTimer()
Undocumented

store(*data*)
Undocumented

ni.model.net_sim.**simulate**(*config*)
Undocumented

1.5 create_design_matrix_vk Module

ni.model.create_design_matrix_vk.**computeCovariate**(*index*, *o*, *C*, *V1*)
Computes a row of the designMatrix corresponding to a certain covariate.

ni.model.create_design_matrix_vk.**create_design_matrix_vk**(*V1*, *o*)
Fills free rows in the current design matrix, deduced from size(mD) and len(freeCov), corresponding to a single covariate according to the spline bases of Volterra kernels. The current kernel(s) and the respective numbers of covariates that will be computed for each kernel is deduced from masterIndex by determining the position in hypothetical upper triangular part of hypercube with number of dimensions corresponding to current kernel order. Using only the 'upper triangular part' of the hypercube reflects the symmetry of the kernels which stems from the fact that only a single spline is used as basis function.

saves covariate information in cell array 'covariates', format is {kernelOrder relativePositionInKernel productTermsOfV1}

Anpassung für Gordon: masterIndex, log, C, mD, freeCov werden berechnet statt übergeben.

ni.model.create_design_matrix_vk.**detKernels**(*freeCov*, *masterIndex*, *oCov*, *mOrder*, *C*)
Determines from the number of free slots in Designmatrix len(freeCov) and the current masterIndex how many covariates for which Volterra coefficient can be computed. Updates model order mOrder.

ni.model.create_design_matrix_vk.**detModelOrder**(*masterIndex*, *C*)
Determines model order and corresponding number of covariates.

ni.model.create_design_matrix_vk.**numCov**(*C*, *complexity*)
Computes number of covariates in a model for which len(complexity) symmetric kernels are assumed.

`ni.model.create_design_matrix_vk.`**`upTriHalf`**(*C*, *cDim*)
Computes number of elements in upper triangular half of hybercube.

1.6 `create_splines` Module

`ni.model.create_splines.`**`N`**(*u*, *i*, *p*, *knots*)
Compute Spline Basis

Evaluates the spline basis of order p defined by knots at knot i and point u.

`ni.model.create_splines.`**`augknt`**(*knots*, *order*)
Augment knot sequence such that some boundary conditions are met.

`ni.model.create_splines.`**`create_splines`**(*length*, *nr_knots*, *remove_last_spline*, *fn_knots*)
Generates B-spline basis functions based on the length and number of knots of the ongoing iteration. fn_knots is a function that computes the knots.

`ni.model.create_splines.`**`create_splines_linspace`**(*length*, *nr_knots*, *remove_last_spline*)
Generates B-spline basis functions based on the length and number of knots of the ongoing iteration

`ni.model.create_splines.`**`create_splines_logspace`**(*length*, *nr_knots*, *remove_last_spline*)
Generates B-spline basis functions based on the length and number of knots of the ongoing iteration

`ni.model.create_splines.`**`spcol`**(*x*, *knots*, *spline_order*)
Computes the spline colocation matrix for knots in x.

The spline collocation matrix contains all m-p-1 bases defined by knots. Specifically it contains the ith basis in the ith column.

Input: x: vector to evaluate the bases on knots: vector of knots spline_order: order of the spline

Output:

colmat: m x m-p matrix The colocation matrix has size m x m-p where m denotes the number of points the basis is evaluated on and p is the spline order. The colums contain the ith basis of knots evaluated on x.

`ni.model.create_splines.`**`spline`**(*x*, *knots*, *p*, *i=0.0*)
Evaluates the ith spline basis given by knots on points in x

1.7 `backend_elasticnet` Module

class `ni.model.backend_elasticnet.`**`Configuration`**
Default Values:

crossvalidation = True

If true, alpha and l1_ratio will be calculated by crossvalidation.

alpha = 0.5

l1_ratio = 1

be_memory_efficient = True Does not keep the data with which it is fitted.

class `ni.model.backend_elasticnet.`**`Fit`**(*f*, *m*)

`predict`(*X=False*)

class `ni.model.backend_elasticnet.`**`Model`**(*c=False*)

> **`fit`**(*x*, *dm*)

`ni.model.backend_elasticnet.`**`compare`**(*x*, *p*, *nr_trials=1*)

`ni.model.backend_elasticnet.`**`predict`**(*x*, *dm*)

1.8 `backend_glm` Module

class `ni.model.backend_glm.`**`Configuration`**

> Default Values:
>
> **be_memory_efficient = True** Does not keep the data with which it is fitted.

class `ni.model.backend_glm.`**`Fit`**(*f*, *m*)

> **`predict`**(*X=False*)

class `ni.model.backend_glm.`**`Model`**(*c=False*)

> **`fit`**(*y*, *X*)

`ni.model.backend_glm.`**`compare`**(*x*, *p*, *nr_trials=1*)

`ni.model.backend_glm.`**`predict`**(*x*, *dm*)

TOOLS PACKAGE

2.1 `bootstrap` Module

`ni.tools.bootstrap.`**`bootstrap`**(*bootstrap_repetitions*, *model*, *data*, *other_data*)

`ni.tools.bootstrap.`**`evaluate`**(*Model*, *Data*, *bootstrap_repetitions*, *return_all=False*)

Executes a certain number of bootstrap repetitions to calculate the bias of the likelihood he model computes

Model

A model object that is capable of loglikelihood estimation

Data

Data that is to be reshuffled. A bootstrap sample is drawn from this Data of the same length with each Element of Data being equally probable of being included.

bootstrap_repetitions

Number of repetitions

return_all

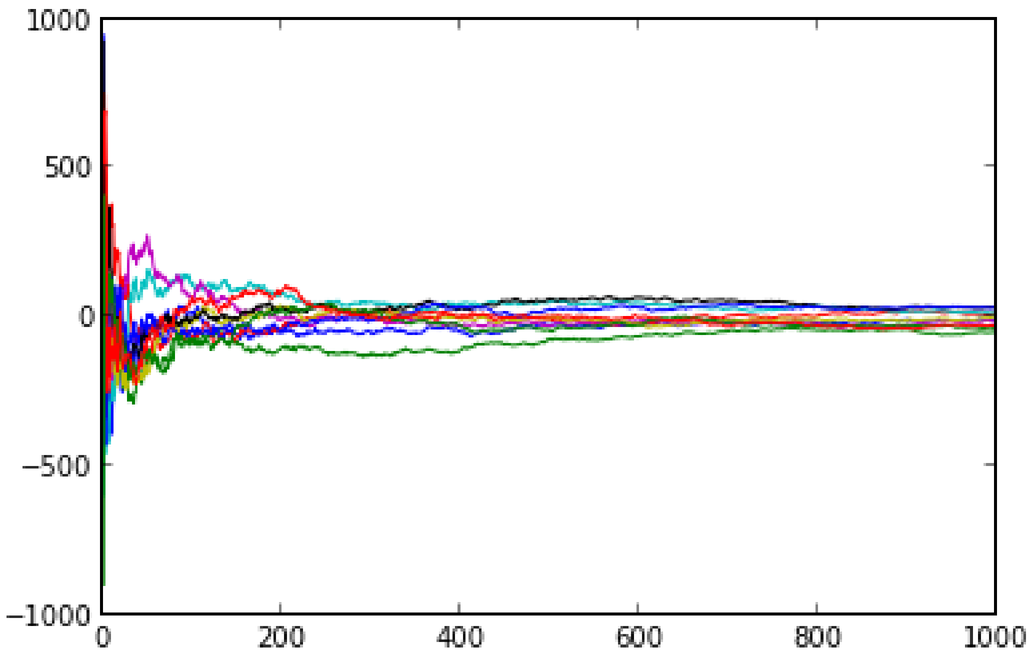
Default: False. Whether an array of all bootstrap biases should be returned or just the mean.

Example:

```
import ni.tools.bootstrap
reload(ni.tools.bootstrap)
import ni.model.pointprocess
reload(ni.model.pointprocess)
p1 = np.array([ni.model.pointprocess.createPoisson(0.1,1000).getCounts() for i in range(0,10)])
p2 = np.array([ni.model.pointprocess.createPoisson(sin(numpy.array(range(0,200))*0.01)*0.5- 0.2,

m1 = ni.model.pointprocess.SimpleFiringRateModel()

ni.tools.bootstrap.evaluate(m1,p1,10000)
```

Or to see the effect of increasing bootstrap size:

```
[plot(np.cumsum(ni.tools.bootstrap.evaluate(m1,p1,1000,return_all=True))/range(1,1001)) for i in
```



`ni.tools.bootstrap.`**`likelihood_Fun`**(*y*, *x*, *mu*)
    Calculates the likelihood for a binary vector and a predicted firing rate

$$-(size(y)/2) \cdot log(2 \cdot \pi \cdot x^2) - (1/x^2 \cdot (y - mu)^2)$$

`ni.tools.bootstrap.`**`plotBootstrap`**(*res*, *path*)

`ni.tools.bootstrap.`**`plotCompareBootstrap`**(*reses*, *path*)

2.2 `progressbar` Module

`ni.tools.progressbar.`**`progress`**(*a*, *b*)
    Undocumented

`ni.tools.progressbar.`**`progress_end`**()
    Undocumented

`ni.tools.progressbar.`**`progress_init`**()
    Undocumented

2.3 `project` Module

NI Project Management

- All steps in a configuration / simulation process will be logged to some folder structure
- after the simulation and even after changing the original code, the results should still be viewable / interpretable with a project viewer
- batches of runs should be easy to batch interpret (characteristic plots etc.)
- **metadata should contain among others:** date software versions configuration options manual comments

- saving of plots/data should be done by the project manager

class `ni.tools.project.`**`CodeManager`**(*project*)
Provides Management for Project Code Files

`appendCode`(*code*)

`appendFile`(*file*)

`report`()

class `ni.tools.project.`**`DataManager`**(*project*)
Provides Management for Project Data Files

`appendFile`(*file*, *options*)

`appendXML`(*xml_root*)

`report`()

class `ni.tools.project.`**`Figure`**(*path*, *display=False*, *close=True*)
Figure Context Manager

Can be used with the **with** statement:

```
import ni
x = np.arange(0,10,0.1)
with ni.figure("some_test.png"):
    plot(cos(x))        # plots to a first plot
    with ni.figure("some_other_test.png"):
        plot(-1*np.array(x)) # plots to a second plot
    plot(sin(x))        # plots to the first plot again
```

Or if they are to be used in an interactive console:

```
import ni
x = np.arange(0,10,0.1)
with ni.figure("some_test.png",close=False):
        plot(cos(x))    # plots to a first plot
                with ni.figure("some_other_test.png",close=False):
                        plot(-1*np.array(x)) # plots to a second plot
        plot(sin(x))    # plots to the first plot again
```

Both figures will be displayed, but the second one will remain available after the code. (But keep in mind that in the iPython pylab console, after every input, all figures will be closed)

class `ni.tools.project.`**`FigureManager`**(*project*)
Provides Management for Figures

class `ni.tools.project.`**`HistoryManager`**(*project*)
Provides Management for Project HistoryManager

class `ni.tools.project.`**`Job`**(*project*, *session*, *path*, *job_name=''*, *job_number=''*, *file=''*, *status='initializing...'*, *dependencies=[ ]*)

`can_run`()

`get_status`()

`html_view`()

`run`(*parameters=[ ]*)

`save`()

set_activity(*msg=''*)

set_status(*msg=''*)

update()

class ni.tools.project.**ListContainer**

append(*msg_type*, *priority*, *date*, *job*, *txt*)

clear()

class ni.tools.project.**LogContainer**(*f*)

append(*msg_type*, *priority*, *date*, *job*, *txt*)

clear()

class ni.tools.project.**PickleContainer**(*f*)

append(*msg_type*, *priority*, *date*, *job*, *txt*)

clear()

class ni.tools.project.**Project**(*folder='unsaved_project'*, *name=''*)

Project Class

loads a Project folder (containing eg. a main.py file or a project_manifest.xml)

abandon()

autorun()

clear()

dbg(*txt*, *priority=-1*)

do_log(*b*)

dumpheap()

err(*txt*, *priority=0*)

execute(*code*, *local_vars={}*, *session=False*)

find_sessions()

get_parameters_from_job_file()

get_session_status(*r=False*)

html_view()

job(*j*)

TODO: rename to something else

job_activate(*j*, *msg='running...'*)

job_done(*j*)

last_run()

less_running_than(*N*)

log(*txt*, *priority=0*)

msg(*msg_type*, *txt*, *priority=0*)

next()

next_job(*ignore_dependencies=False*)

parse_job_file(*filename*, *session*)

print_job_status()

print_long_job_status()

report(*silent=False*)

reportHTML()

require_job(*j*)

reset_failed_jobs()

run(*parameters=*[ ], *job=False*)

save(*name*, *val*)

save_html(*path='project.html'*)

select_session(*path*)

set_session_status(*msg='running...'*)

setup_jobs(*parameter_string='‘*)

sibjob(*j*)

Sibling Job

Is on the same level as the previous job (ie. a child of its parent)

subjob(*j*)

superjob()

update_job_status()

class ni.tools.project.**Session**(*project*, *path='‘*, *parameter_string='‘*)

abandon()

add_job(*job_name='‘*, *job_number='‘*, ***kwargs*)

execute(*code*, *local_vars={}*)

find_jobs()

get_status()

html_view()

next_job(*retry_failed=False*, *ignore_dependencies=False*)

parse_job_file(*filename*, *parameter_string='‘*)

print_job_status()

print_long_job_status()

reset_failed_jobs(*which='failed.'*, *to='pending'*)

save_html(*path='session.html'*)

set_status(*msg='‘*)

setup_jobs(*source_file*, *parameter_string='‘*)

update_job_files (*source_file='‘, parameter_string='‘*)

update_jobs ()

class ni.tools.project.**TemporaryJob** (*project, session, job_name*)
Bases: ni.tools.project.Job

class ni.tools.project.**TemporarySession** (*project*)
Bases: ni.tools.project.Session

class ni.tools.project.**VariableContainer**

ni.tools.project.**atoi** (*text*)
converts text containing numbers into ints / used by natural_keys()

ni.tools.project.**dbg** (*txt, priority=-1*)

ni.tools.project.**do_log** (*b*)

ni.tools.project.**dumpheap** ()

ni.tools.project.**err** (*txt, priority=0*)

ni.tools.project.**figure** (*path, display=False, close=True*)

Todo

integrate with projects, write to default paths

Can be used with the **with** statement:

```
import ni
x = np.arange(0,10,0.1)
with ni.figure("some_test.png"):
    plot(cos(x))         # plots to a first plot
    with ni.figure("some_other_test.png"):
        plot(-1*np.array(x)) # plots to a second plot
    plot(sin(x))         # plots to the first plot again
```

Or if they are to be used in an interactive console:

```
import ni
x = np.arange(0,10,0.1)
with ni.figure("some_test.png",display=True):
    plot(cos(x))         # plots to a first plot
    with ni.figure("some_other_test.png",close=False):
        plot(-1*np.array(x)) # plots to a second plot
    plot(sin(x))         # plots to the first plot again
```

Both of these figures will be displayed, but the second one will remain open and can be activated again.

ni.tools.project.**job** (*j*)

ni.tools.project.**load** (*path*)

ni.tools.project.**log** (*txt, priority=0*)

ni.tools.project.**natural_keys** (*text*)
alist.sort(key=natural_keys) sorts in human order (See Toothy’s implementation in the comments of http://nedbatchelder.com/blog/200712/human_sorting.html)

ni.tools.project.**natural_sorted**(*l*)
sorts a sortable in human order (0 < 20 < 100)

ni.tools.project.**report**(*silent=False*)

ni.tools.project.**require_job**(*j*)

ni.tools.project.**run**()

ni.tools.project.**save**(*name*, *val*)

ni.tools.project.**sibjob**(*j*)

ni.tools.project.**subjob**(*j*)

ni.tools.project.**superjob**()

2.4 `html_view` Module

This module can generate HTML output from text or objects that provide a .html_view() function:

```
import ni
view = ni.View()        # this is a shortcut for ni.tools.html_view.View
view.add("#1/title","This is a test")
view.add("#2/Some Example Models/tabs/",ni.model.ip.Model({'name': 'Basic Model'}))
view.add("#2/Some Example Models/tabs/",ni.model.ip.Model({'autohistory_2d':True, 'name': 'Model witl
view.add("#2/Some Example Models/tabs/",ni.model.ip.Model({'rate':False, 'name': 'Model without Rate
view.add("#3/Some Example Data/tabs/1",ni.data.monkey.Data())
view.render("this_is_a_test.html")
```

class ni.tools.html_view.**Figure**(*view*, *path*, *close=True*, *figsize=False*)
Figure Context Manager

Can be used with the **with** statement:

```
import ni
v = ni.View()
x = np.arange(0,10,0.1)
with ni.tools.html_view.Figure(v,"some test"):
    plot(cos(x))        # plots to a first plot
    with ni.tools.html_view.Figure(v,"some other test"):
        plot(-1*np.array(x)) # plots to a second plot
    plot(sin(x))        # plots to the first plot again
v.render("context_manager_test.html")
```

class ni.tools.html_view.**View**(*path='‘*)

add(*path*, *obj*)

figure(*path='‘*, *close=True*, *figsize=False*)
Provides a Context Manager for figure management

Should be used if plots are to be used in

Example:

```
import ni
v = ni.View()
x = np.arange(0,10,0.1)
with v.figure("some test"):
    plot(cos(x))                # plot to a first plot
    with v.figure("some other test"):
        plot(-1*np.array(x))    # plot to a second plot
    plot(sin(x))                # plot to the first plot again
v.render("context_manager_test.html")
```

has(*path*)

html_view()

load(*filename*)

loadList(*filenames*)

load_glob(*filename_template*)

load_list(*filenames*)

parse(*tree*)

process(*obj*, *mode='text'*)

render(*path*, *include_files=True*)

save(*filename*)

savefig(*p='‘*, *fig='‘*, *close=True*)

ni.tools.html_view.**atoi**(*text*)

ni.tools.html_view.**natural_keys**(*text*)
alist.sort(key=natural_keys) sorts in human order http://nedbatchelder.com/blog/200712/human_sorting.html (See Toothy's implementation in the comments)

ni.tools.html_view.**natural_sorted**(*l*)
sorts a sortable in human order (0 < 20 < 100)

2.5 strap Module

ni.tools.strap.**bootstrap**(*bootstrap_repetitions*, *model*, *data*, *test_data=[ ]*, *shuffle=True*, *prefix='‘*, *bootstrap_data=[ ]*)
A helper function that performs bootstrap evaluation of models.

A Model *model* is fitted with some data *data*, called "actual data" or "D" and subsequently on all of a number of bootstrap samples "D*_n" for n in range(*bootstrap_repetitions*). This yields an *actual fit* and *bootstrap_repetitions* times *boot fit* (or *fit**) for each sample.

Use bootstrap_results for explanations on the dimensions of the result.

bootstrap_repetitions

model

data

'test_data'=[]

'shuffle'=True

'prefix'='‘ String that is prefixed to the results.

'bootstrap_data'=[] If new data instead of trial shuffling is to be used as bootstrap data, this data should be passed here. The *ni.data.data.Data* Instance should contain an additional index *Bootstrap Sample* `ni.tools.strap.`**`bootstrap_samples`**(*bootstrap_data*, *model*, *data*, *test_data=[ ]*, *shuffle=False*, *prefix='‘*, *boot_dim='Bootstrap Sample'*)

Performs bootstrap evaluation with bootstrap data.

A Model *model* is fitted with some data *data*, called "actual data" or "D" and subsequently on all of a number of bootstrap samples "D*_n" for n in range(*bootstrap_repetitions*). This yields an *actual fit* and *bootstrap_repetitions* times *boot fit* (or *fit**) for each sample.

Use bootstrap_results for explanations on the dimensions of the result.

bootstrap_data If new data instead of trial shuffling is to be used as bootstrap data, this data should be passed here. The *ni.data.data.Data* Instance should contain an additional index *Bootstrap Sample*

model

Model to be evaluated. It needs to provide an *x()*, *dm()* and *fit(x=, dm=)*/*fit(data)* method.

data

‘test_data‘=[]

‘shuffle‘=True

‘prefix‘=’‘ String that is prefixed to the results.

boot_dim The *ni.data.data.Data* Instance should contain an additional index *Bootstrap Sample* or be a list. If some other index should be used as bootstrap samples, *boot_dim* can be set to that.

`ni.tools.strap.`**`bootstrap_time`**(*bootstrap_repetitions*, *model*, *data*, *test_data=[ ]*, *prefix='‘*)

Performs bootstrap evaluation of models.

A Model *model* is fitted with some data *data*, called "actual data" or "D" and subsequently on all of a number of bootstrap samples "D*_n" for n in range(*bootstrap_repetitions*). This yields an *actual fit* and *bootstrap_repetitions* times *boot fit* (or *fit**) for each sample.

Use bootstrap_results for explanations on the dimensions of the result.

bootstrap_repetitions

model

Model to be evaluated. It needs to provide an *x()*, *dm()* and *fit(x=, dm=)* method.

data

‘test_data‘=[]

‘prefix‘=’‘ String that is prefixed to the results.

`ni.tools.strap.`**`bootstrap_trials`**(*bootstrap_repetitions*, *model*, *data*, *test_data=[ ]*, *shuffle=True*, *prefix='‘*, *bootstrap_data=[ ]*)

Performs bootstrap evaluation by trial shuffling.

A Model *model* is fitted with some data *data*, called "actual data" or "D" and subsequently on all of a number of bootstrap samples "D*_n" for n in range(*bootstrap_repetitions*). This yields an *actual fit* and *bootstrap_repetitions* times *boot fit* (or *fit**) for each sample.

Use bootstrap_results for explanations on the dimensions of the result.

bootstrap_repetitions

Number of bootstrap repetitions

model

Model to be evaluated. It needs to provide an *x()*, *dm()* and *fit(x=, dm=)*/*fit(data)* method.

data

'test_data'=[]

'shuffle'=True

'prefix'=' String that is prefixed to the results.

'bootstrap_data'=[] If new data instead of trial shuffling is to be used as bootstrap data, this data should be passed here. The *ni.data.data.Data* Instance should contain an additional index *Bootstrap Sample*

`ni.tools.strap.`**`description`**(*prefix='', additional_information=''*)
Describes the common bootstrap output variables as a dictionary. *additional_information* will be appended to each entry, *prefix* will be prepended to each key.

`ni.tools.strap.`**`generate`**(*model, bootstrap_repetitions*)

`ni.tools.strap.`**`merge`**(*stats*)

Todo

implement merge function for bootstrap data that calculates EIC etc.

`ni.tools.strap.`**`plotBootstrap`**(*res, path*)
Deprecated since version 0.1. use the plot capabilities of the `ni.tools.statcollector.StatCollector`.

`ni.tools.strap.`**`plotCompareBootstrap`**(*reses, path*)
Deprecated since version 0.1. use the plot capabilities of the `ni.tools.statcollector.StatCollector`.

2.6 `statcollector` Module

class `ni.tools.statcollector.`**`StatCollector`**(*stat_init={}*)
A class to collect statistics about models. It can be used to analyse nested models, as slashes in the name are interpreted as submodels.

Example:

```
>>> rate model/0
>>> rate model/1
>>> rate model/2
>>> rate model/3
>>> nested model/0
>>> nested model/1
>>> nested model/2
>>> nested model/3
>>> nested model/0/1
>>> nested model/0/2
>>> nested model/0/3
>>> nested model/0/2/1
>>> nested model/0/2/3
>>> nested model/0/2/3/1
```

This example could be generated by fitting a model with a certain number of crosshistory cells. In each iteration the best model is extended by another cell. The nested model can then be evaluated whether it has increasing likelihood and/or eic, aic or other statistics:

```
>>> stats.getModelsOnPath(['nested model',0,2,3,1]).get('eic')
[ -1023, -1020, -900, -950 ]
```

self.stats: a dict of lists, each list containing dicts with: name llf eic, aic, eice, complexity (optional) additional - a dict with more information (ignored for now)

addNode (*name*, *data={}*)
adds the node *name* with the attributes in the dictionary *dic*. If *name* exists, it wll be overwritten.

addToNode (*name*, *dic*)
adds all attributes in the dictionary *dic* to the node *name*

filter (*name*)
returns a StatCollector Object with only a subset of models

get (*dim*)
returns a numpy ndarray with the *dim* attributes of each node that contains *dim*

getChildren ()

getDimensions ()
returns which dimensions are availble for the contained nodes

getList (*dim*)
returns a list with the *dim* attributes of each node that contains *dim*

getModelsOnPath (*name*)
returns models that lead to the node *name*

getNode (*name*)
returns a dictionary with all attributes of the node *name*

getTree (*substitution_patterns*=[ ])
returns a tuple used by plotTree to plot a tree representation of the nodes.

html_view ()
Generates an html_view of this object.

Example:

```
stats.html_view().render('stats_file.html')
```

keys ()
returns the name of all nodes. Synonym of `StatCollector.nodes()`

load (*filename*)
Loads the StatCollector saved to *filename*.

This file should be a pickled dictionary.

loadList (*filenames*)
loads a list of files. (Alias of `StatCollector.load_list()`)

load_glob (*filename_template*)

load_list (*filenames*)
loads a list of files

nodes ()
returns the name of all nodes. Synonym of `StatCollector.keys()`

plotHist (*path*, *width*, *dims*)
plots a histogram of each dimension in *dims*

plotTree (*dim*, *substitution_patterns*=[ ], *line_kwargs={}*, *marker_kwargs={}*, *right_to_left=False*)
Plots a tree of the nodes, using *dim* as the height, if the node contains *dim*.

Slashes in the model names will be used as the different levels in the tree. The order of the parts between the slashes is ignored for now (3/4 and 4/3 are the same).

substitution_patterns may contain substitution patterns (used to connect nodes) for re.sub as a three tuple (pattern, substitute, color), where color is the color that will be assigned to this connection

line_kwargs and *marker_kwargs* can contain arguments in a dictionary to alter the options to set lines or markers. The dictionary will be passed on to the plot function.

right_to_left determines, whether the plot is plotted from left to right (default) or the other way around (*right_to_left* = True).

prefix(*prefix='/'*)
Makes the last node a part of the property name

pull_from_inner_dict(*from_dictionary='statistics'*, *from_key='bic'*, *to_key='BIC'*)
If a dictionary is added as a dimension, this function can pull values from that dictionary and add them to each node that has the specific dictionary. As the bootstrap functions add the statistics dictionary of the model fit to the node, this function has to be used to eg. access the BIC criterion (which is why this is the default from and to keys).

re(*regex*)
returns a StatCollector Object with only a subset of models

rename(*pattern*, *substitution*)
Renames nodes with the regex pattern *pattern* just like `re.sub()`.

Example to rename different number of knots to a tree, where each the increase in knots is counted as a submodel:

```
statsr = stats.rename(r'50','30/50').rename(r'30','20/30').rename(r'20','10/20').rename(r'10
```

save(*filename*)
Saves the StatCollector to *filename*.

This file will be a pickled dictionary.

set(*name*, *key*, *value*)
sets one attribute for node *name*

split(*keys*)
Takes a portion of the property names and makes it a node

`ni.tools.statcollector.`**atoi**(*text*)
converts text containing numbers into ints / used by `natural_keys()`

`ni.tools.statcollector.`**listToPath**(*name*)
Creates a string that joins a list together with slashes. The list can contain strings and numbers.

`ni.tools.statcollector.`**natural_keys**(*text*)
alist.sort(key=natural_keys) sorts in human order (See Toothy's implementation in the comments of http://nedbatchelder.com/blog/200712/human_sorting.html)

CHAPTER

THREE

DATA PACKAGE

Provides easy access to some data.

3.1 `data` Module

Todo

Use different internal representations, depending on use. Ie. Spike times vs. binary array

Todo

Lazy loading and prevention from data duplicates where unnecessary. See also: indexing view versus copy

3.1.1 Storing Spike Data in Python with Pandas

The pandas package allows for easy storage of large data objects in python. The structure that is used by this toolbox is the pandas `pandas.MultiIndexedFrame` which is a `pandas.DataFrame` / pandas.DataFrame with an Index that has multiple levels.

The index contains at least the levels `'Cell'`, `'Trial'` and `'Condition'`. Additional Indizex can be used (eg. `'Bootstrap Sample'` for Bootstrap Samples), but keep in mind that when fitting a model only `'Cell'` and `'Trial'` should remain, all other dimensions will be collapsed as more sets of Trials which may be indistinguishable after the fit.

Condition	Cell	Trial	*t* (Timeseries of specific trial)
0	0	0	0,0,0,0,1,0,0,0,0,1,0...
0	0	1	0,0,0,1,0,0,0,0,1,0,0...
0	0	2	0,0,1,0,1,0,0,1,0,1,0...
0	1	0	0,0,0,1,0,0,0,0,0,0,0...
0	1	1	0,0,0,0,0,1,0,0,0,1,0...
...	...	...	...
1	0	0	0,0,1,0,0,0,0,0,0,0,1...
1	0	1	0,0,0,0,0,1,0,1,0,0,0...
...	...	...	...

To put your own data into a `pandas.DataFrame`, so it can be used by the models in this toolbox create a MultiIndex for example like this:

```
import ni
import pandas as pd
d = []
tuples = []
for con in range(nr_conditions):
        for t in range(nr_trials):
                for c in range(nr_cells):
                                spikes = list(ni.model.pointprocess.getBinary(Spike_times_STC.all_SUA
                                if spikes != []:
                                        d.append(spikes)
                                        tuples.append((con,t,c))
index = pd.MultiIndex.from_tuples(tuples, names=['Condition','Trial','Cell'])
data = ni.data.data.Data(pd.DataFrame(d, index = index))
```

If you only have one trial if several cells or one cell with a few trials, it can be indexed like this:

> from ni.data.data import Data import pandas as pd
>
> index = pd.MultiIndex.from_tuples([(0,0,i) for i in range(len(d))], names=['Condition','Cell','Trial']) data = Data(pd.DataFrame(d, index = index))

To use the data you can use `ni.data.data.Data.filter()`:

```
only_first_trials = data.filter(0, level='Trial')

# filter returns a copy of the Data object

only_the_first_trial = data.filter(0, level='Trial').filter(0, level='Cell').filter(0, level='Conditi

only_the_first_trial = data.condition(0).cell(0).trial(0) # condition(), cell() and trial() are short

only_some_trials  = data.trial(range(3,15))
# using slices, ranges or boolean indexing causes the DataFrame to be indexed again from 0 to N, in t
```

Also ix and xs pandas operations can be useful:

```
plot(data.data.ix[(0,0,0):(0,3,-1)].transpose().cumsum())
plot(data.data.xs(0,level='Condition').xs(0,level='Cell').ix[:5].transpose().cumsum())
```

class `ni.data.data.`**`Data`**(*matrix*, *dimensions=[ ]*, *other_spikes=False*, *key_index='i'*, *resolution=1000*)

Spike data container

Contains a panda Data Frame with MultiIndex. Can save to and load from files.

The Index contains at least Trial, Cell and Condition and can be extended.

`as_list_of_series`(*list_conditions=True*, *list_cells=True*, *list_trials=False*, *list_additional_indizes=True*)

Returns one timeseries, collapsing only certain indizes (on default only trials). All non collapsed indizes

`as_series`()

Returns one timeseries, collapsing all indizes.

The output has dimensions of (N,1) with N being length of one trial x nr_trials x nr_cells x nr_conditions (x additonal indices).

If cells, conditions or trials should be separated, use `as_list_of_series()` instead.

`cell`(*cells=False*)

filters for an array of cells -> see `ni.data.data.Data.filter()`

`condition`(*conditions=False*)

filters for an array of conditions -> see `ni.data.data.Data.filter()`

filter (*array=False*, *level='Cell'*)
: filters for arbitrary index levels *array* a number, list or numpy array of indizes that are to be filtered *level* the level of index that is to be filtered. Default: 'Cell'

firing_rate (*smooth_width=0*, *trials=False*)
: computes the firing rate of the data for each cell separately.

getFlattend (*all_in_one=True*, *trials=False*)
: Deprecated since version 0.1: Use `as_list_of_series()` and `as_series()` instead Returns one timeseries for all trials.

  The *all_in_one* flag determines whether `'Cell'` and `'Condition'` should also be collapsed. If set to *False* and the number of Conditions and/or Cells is greater than 1, a list of timeseries will be returned. If both are greater than 1, then a list containing for each condition a list with a time series for each cell.

html_view ()

interspike_intervals (*smooth_width=0*, *trials=False*)
: computes inter spike intervalls in the data for each cell separately.

read_pickle (*path*)
: Loads a DataFrame from a file

shape (*level*)
: Returns the shape of the sepcified level:

```
>>> data.shape('Trial')
```

100

```
>>> data.shape('Cell') == data.nr_cells
    True
```

to_pickle (*path*)
: Saves the DataFrame to a file

trial (*trials=False*)
: filters for an array of trials -> see `ni.data.data.Data.filter()`

`ni.data.data.`**loadFromFile** (*path*)

`ni.data.data.`**matrix_to_dataframe** (*matrix*, *dimensions*)

`ni.data.data.`**merge** (*datas*, *dim*, *keys=False*)
: merges multiple Data instances into one:

```
data = ni.data.data.merge([ni.data.data.Date(f) for f in ['data1.pkl','data2.pkl','data3.pkl']],
```

`ni.data.data.`**saveToFile** (*path*, *o*)

3.2 `decoding_data` Module

Loads Data into a Panda Data Frame

class `ni.data.decoding_data.`**Cell** (*data*)

class `ni.data.decoding_data.`**DecodingData**
: Loads Data into a Panda Data Frame

class `ni.data.decoding_data.`**`Trial`**

  `addCell`(*data*)

  `getMatrix`()

`ni.data.decoding_data.`**`get`**()

3.3 `monkey` Module

`ni.data.monkey.`**`Data`**(*file_nr='101a03'*, *resolution=1000*, *trial=[ ]*, *condition=[ ]*, *cell=[ ]*)

Loads Data into a Data Frame

Expects a file number. Available file numbers are in ni.data.monkey.available_files:

```
>>> print ni.data.monkey.available_files
        ['101a03', '104a10', '107a03', '108a08', '112a03', '101a03', '104a11', '107a04', '109a04
```

trial

number of trial to load or list of trials to load. Non-existent trial numbers are ignored.

condition

number of condition to load or list of conditions to load. Non-existent condition numbers are ignored.

cell

number of cell to load or list of cells to load. Non-existent cell numbers are ignored.

Example:

```
data = ni.data.monkey.Data(trial_nr = ni.data.monkey.available_trials[3], trial=range(10), condi
```

CHAPTER

FOUR

LIST OF TODO ITEMS

Todo

Use different internal representations, depending on use. Ie. Spike times vs. binary array

(The *original entry* is located in /home/plogic/uni/MT/EIC/py/ni/data/data.py:docstring of ni.data.data, line 7.)

Todo

Lazy loading and prevention from data duplicates where unnecessary. See also: indexing view versus copy

(The *original entry* is located in /home/plogic/uni/MT/EIC/py/ni/data/data.py:docstring of ni.data.data, line 10.)

Todo

Find out what this is

(The *original entry* is located in /home/plogic/uni/MT/EIC/py/ni/model/ip.py:docstring of ni.model.ip.Configuration, line 19.)

Todo

split into more usefull and modular functions

(The *original entry* is located in /home/plogic/uni/MT/EIC/py/ni/model/ip.py:docstring of ni.model.ip.generate_to_file, line 3.)

Todo

Add options for random number generator seeds, so that the *exact* same trial can be run over and over again.

(The *original entry* is located in /home/plogic/uni/MT/EIC/py/ni/model/net_sim.py:docstring of ni.model.net_sim, line 11.)

Todo

integrate with projects, write to default paths

(The *original entry* is located in /home/plogic/uni/MT/EIC/py/ni/tools/project.py:docstring of ni.tools.project.figure, line 1.)

Todo

implement merge function for bootstrap data that calculates EIC etc.

(The *original entry* is located in /home/plogic/uni/MT/EIC/py/ni/tools/strap.py:docstring of ni.tools.strap.merge, line 1.)

CHAPTER

FIVE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

n

ni.data.data *(Unix)*, 31
ni.data.decoding_data *(Unix)*, 33
ni.data.monkey *(Unix)*, 34
ni.model.backend_elasticnet *(Unix)*, 17
ni.model.backend_glm *(Unix)*, 18
ni.model.create_design_matrix_vk, 16
ni.model.create_splines, 17
ni.model.designmatrix *(Unix)*, 8
ni.model.ip *(Unix)*, 10
ni.model.net_sim *(Unix)*, 14
ni.model.pointprocess *(Unix)*, 3
ni.tools.bootstrap, 19
ni.tools.html_view *(Unix)*, 25
ni.tools.progressbar *(Unix)*, 20
ni.tools.project *(Unix)*, 20
ni.tools.statcollector *(Unix)*, 28
ni.tools.strap *(Unix)*, 26

INDEX

A

abandon() (ni.tools.project.Project method), 22
abandon() (ni.tools.project.Session method), 23
AdaptiveRateComponent (class in ni.model.designmatrix), 8
add() (ni.model.designmatrix.DesignMatrix method), 8
add() (ni.model.designmatrix.DesignMatrixTemplate method), 8
add() (ni.tools.html_view.View method), 25
add_job() (ni.tools.project.Session method), 23
addCell() (ni.data.decoding_data.Trial method), 34
addLinSpline() (ni.model.designmatrix.DesignMatrix method), 8
addLogSpline() (ni.model.designmatrix.DesignMatrix method), 8
addNode() (ni.tools.statcollector.StatCollector method), 29
addSpike() (ni.model.pointprocess.PointProcess method), 3
addToNode() (ni.tools.statcollector.StatCollector method), 29
append() (ni.model.ip.MultiChannelModel method), 13
append() (ni.tools.project.ListContainer method), 22
append() (ni.tools.project.LogContainer method), 22
append() (ni.tools.project.PickleContainer method), 22
appendCode() (ni.tools.project.CodeManager method), 21
appendFile() (ni.tools.project.CodeManager method), 21
appendFile() (ni.tools.project.DataManager method), 21
appendXML() (ni.tools.project.DataManager method), 21
as_list_of_series() (ni.data.data.Data method), 32
as_series() (ni.data.data.Data method), 32
atoi() (in module ni.tools.html_view), 26
atoi() (in module ni.tools.project), 24
atoi() (in module ni.tools.statcollector), 30
augknt() (in module ni.model.create_splines), 17
autorun() (ni.tools.project.Project method), 22

B

backend (ni.model.ip.Model attribute), 12
bootstrap() (in module ni.tools.bootstrap), 19
bootstrap() (in module ni.tools.strap), 26
bootstrap_samples() (in module ni.tools.strap), 26
bootstrap_time() (in module ni.tools.strap), 27
bootstrap_trials() (in module ni.tools.strap), 27

C

can_run() (ni.tools.project.Job method), 21
Cell (class in ni.data.decoding_data), 33
cell() (ni.data.data.Data method), 32
clear() (ni.tools.project.ListContainer method), 22
clear() (ni.tools.project.LogContainer method), 22
clear() (ni.tools.project.PickleContainer method), 22
clear() (ni.tools.project.Project method), 22
clip() (ni.model.designmatrix.DesignMatrix method), 8
CodeManager (class in ni.tools.project), 21
combine() (ni.model.designmatrix.DesignMatrixTemplate method), 8
compare() (in module ni.model.backend_elasticnet), 18
compare() (in module ni.model.backend_glm), 18
compare() (ni.model.ip.FittedModel method), 12
compare() (ni.model.ip.Model method), 12
compare() (ni.model.pointprocess.SimpleFiringRateModel method), 3
complexity (ni.model.ip.FittedModel attribute), 12
Component (class in ni.model.designmatrix), 8
computeCovariate() (in module ni.model.create_design_matrix_vk), 16
condition() (ni.data.data.Data method), 32
Configuration (class in ni.model.backend_elasticnet), 17
Configuration (class in ni.model.backend_glm), 18
Configuration (class in ni.model.ip), 11
convolve_spikes() (in module ni.model.designmatrix), 10
convolve_spikes_2d() (in module ni.model.designmatrix), 10
create_design_matrix_vk() (in module ni.model.create_design_matrix_vk), 16
create_splines() (in module ni.model.create_splines), 17
create_splines_linspace() (in module ni.model.create_splines), 17
create_splines_logspace() (in module ni.model.create_splines), 17
createPoisson() (in module ni.model.pointprocess), 3

D

Data (class in ni.data.data), 32
Data() (in module ni.data.monkey), 34
DataManager (class in ni.tools.project), 21
dbg() (in module ni.tools.project), 24
dbg() (ni.tools.project.Project method), 22
DecodingData (class in ni.data.decoding_data), 33
description() (in module ni.tools.strap), 28
DesignMatrix (class in ni.model.designmatrix), 8
DesignMatrixTemplate (class in ni.model.designmatrix), 8
detKernels() (in module ni.model.create_design_matrix_vk), 16
detModelOrder() (in module ni.model.create_design_matrix_vk), 16
dm() (ni.model.ip.Model method), 13
do_log() (in module ni.tools.project), 24
do_log() (ni.tools.project.Project method), 22
dumpheap() (in module ni.tools.project), 24
dumpheap() (ni.tools.project.Project method), 22
dumps() (ni.model.ip.FittedModel method), 12

E

err() (in module ni.tools.project), 24
err() (ni.tools.project.Project method), 22
evaluate() (in module ni.tools.bootstrap), 19
execute() (ni.tools.project.Project method), 22
execute() (ni.tools.project.Session method), 23

F

family_fitted_function() (ni.model.ip.FittedModel method), 12
Figure (class in ni.tools.html_view), 25
Figure (class in ni.tools.project), 21
figure() (in module ni.tools.project), 24
figure() (ni.tools.html_view.View method), 25
FigureManager (class in ni.tools.project), 21
filter() (ni.data.data.Data method), 33
filter() (ni.tools.statcollector.StatCollector method), 29
find_jobs() (ni.tools.project.Session method), 23
find_sessions() (ni.tools.project.Project method), 22
firing_rate() (ni.data.data.Data method), 33
firing_rate_model() (ni.model.ip.FittedModel method), 12
Fit (class in ni.model.backend_elasticnet), 17
Fit (class in ni.model.backend_glm), 18
fit() (ni.model.backend_elasticnet.Model method), 18
fit() (ni.model.backend_glm.Model method), 18
fit() (ni.model.ip.Model method), 13
fit() (ni.model.pointprocess.SimpleFiringRateModel method), 3
fit_with_design_matrix() (ni.model.ip.Model method), 13
FittedModel (class in ni.model.ip), 11

G

generate() (in module ni.tools.strap), 28
generate() (ni.model.ip.FittedModel method), 12
generate_to_file() (in module ni.model.ip), 13
generateDesignMatrix() (ni.model.ip.Model method), 13
get() (in module ni.data.decoding_data), 34
get() (ni.model.designmatrix.DesignMatrix method), 8
get() (ni.model.designmatrix.DesignMatrixTemplate method), 9
get() (ni.tools.statcollector.StatCollector method), 29
get_components() (ni.model.designmatrix.DesignMatrixTemplate method), 9
get_parameters_from_job_file() (ni.tools.project.Project method), 22
get_session_status() (ni.tools.project.Project method), 22
get_status() (ni.tools.project.Job method), 21
get_status() (ni.tools.project.Session method), 23
getBinary() (in module ni.model.pointprocess), 6
getChildren() (ni.tools.statcollector.StatCollector method), 29
getCounts() (in module ni.model.pointprocess), 6
getCounts() (ni.model.pointprocess.PointProcess method), 3
getDimensions() (ni.tools.statcollector.StatCollector method), 29
getFlattend() (ni.data.data.Data method), 33
getIndex() (ni.model.designmatrix.DesignMatrix method), 8
getIndex() (ni.model.designmatrix.DesignMatrixTemplate method), 9
getList() (ni.tools.statcollector.StatCollector method), 29
getMask() (ni.model.designmatrix.DesignMatrix method), 8
getMask() (ni.model.designmatrix.DesignMatrixTemplate method), 9
getMatrix() (ni.data.decoding_data.Trial method), 34
getModelsOnPath() (ni.tools.statcollector.StatCollector method), 29
getNode() (ni.tools.statcollector.StatCollector method), 29
getParams() (ni.model.ip.FittedModel method), 12
getProbability() (ni.model.pointprocess.PointProcess method), 3
getPvalues() (ni.model.ip.FittedModel method), 12
getSplines() (ni.model.designmatrix.AdaptiveRateComponent method), 8
getSplines() (ni.model.designmatrix.Component method), 8
getSplines() (ni.model.designmatrix.HistoryComponent method), 9
getSplines() (ni.model.designmatrix.RateComponent method), 9
getSplines() (ni.model.designmatrix.SecondOrderHistoryComponent method), 10

getTree() (ni.tools.statcollector.StatCollector method), 29

H

has() (ni.tools.html_view.View method), 26
history_model() (ni.model.ip.FittedModel method), 12
HistoryComponent (class in ni.model.designmatrix), 9
HistoryDesignMatrix (class in ni.model.designmatrix), 9
HistoryManager (class in ni.tools.project), 21
html_view() (ni.data.data.Data method), 33
html_view() (ni.model.ip.FittedModel method), 12
html_view() (ni.model.ip.Model method), 13
html_view() (ni.tools.html_view.View method), 26
html_view() (ni.tools.project.Job method), 21
html_view() (ni.tools.project.Project method), 22
html_view() (ni.tools.project.Session method), 23
html_view() (ni.tools.statcollector.StatCollector method), 29

I

interspike_interval() (in module ni.model.pointprocess), 6
interspike_intervals() (ni.data.data.Data method), 33

J

Job (class in ni.tools.project), 21
job() (in module ni.tools.project), 24
job() (ni.tools.project.Project method), 22
job_activate() (ni.tools.project.Project method), 22
job_done() (ni.tools.project.Project method), 22

K

keys() (ni.tools.statcollector.StatCollector method), 29

L

last_run() (ni.tools.project.Project method), 22
less_running_than() (ni.tools.project.Project method), 22
likelihood_Fun() (in module ni.tools.bootstrap), 20
ListContainer (class in ni.tools.project), 22
listToPath() (in module ni.tools.statcollector), 30
load() (in module ni.tools.project), 24
load() (ni.model.net_sim.Net method), 15
load() (ni.tools.html_view.View method), 26
load() (ni.tools.statcollector.StatCollector method), 29
load_glob() (ni.tools.html_view.View method), 26
load_glob() (ni.tools.statcollector.StatCollector method), 29
load_list() (ni.tools.html_view.View method), 26
load_list() (ni.tools.statcollector.StatCollector method), 29
loadFromFile() (in module ni.data.data), 33
loadList() (ni.tools.html_view.View method), 26
loadList() (ni.tools.statcollector.StatCollector method), 29
log() (in module ni.tools.project), 24
log() (ni.tools.project.Project method), 22
LogContainer (class in ni.tools.project), 22
loglikelihood() (ni.model.pointprocess.SimpleFiringRateModel method), 3

M

matrix_to_dataframe() (in module ni.data.data), 33
merge() (in module ni.data.data), 33
merge() (in module ni.tools.strap), 28
Model (class in ni.model.backend_elasticnet), 17
Model (class in ni.model.backend_glm), 18
Model (class in ni.model.ip), 12
msg() (ni.tools.project.Project method), 22
MultiChannelModel (class in ni.model.ip), 13

N

N() (in module ni.model.create_splines), 17
natural_keys() (in module ni.tools.html_view), 26
natural_keys() (in module ni.tools.project), 24
natural_keys() (in module ni.tools.statcollector), 30
natural_sorted() (in module ni.tools.html_view), 26
natural_sorted() (in module ni.tools.project), 24
Net (class in ni.model.net_sim), 15
next() (ni.tools.project.Project method), 22
next_job() (ni.tools.project.Project method), 23
next_job() (ni.tools.project.Session method), 23
ni.data.data (module), 31
ni.data.decoding_data (module), 33
ni.data.monkey (module), 34
ni.model.backend_elasticnet (module), 17
ni.model.backend_glm (module), 18
ni.model.create_design_matrix_vk (module), 16
ni.model.create_splines (module), 17
ni.model.designmatrix (module), 8
ni.model.ip (module), 10
ni.model.net_sim (module), 14
ni.model.pointprocess (module), 3
ni.tools.bootstrap (module), 19
ni.tools.html_view (module), 25
ni.tools.progressbar (module), 20
ni.tools.project (module), 20
ni.tools.statcollector (module), 28
ni.tools.strap (module), 26
nodes() (ni.tools.statcollector.StatCollector method), 29
numCov() (in module ni.model.create_design_matrix_vk), 16

P

parse() (ni.tools.html_view.View method), 26
parse_job_file() (ni.tools.project.Project method), 23
parse_job_file() (ni.tools.project.Session method), 23
PickleContainer (class in ni.tools.project), 22
plot() (ni.model.designmatrix.DesignMatrix method), 8
plot() (ni.model.net_sim.SimulationResult method), 16

plot() (ni.model.pointprocess.PointProcess method), 3
plot_firing_rate_model() (ni.model.ip.FittedModel method), 12
plot_firing_rates() (ni.model.net_sim.Net method), 15
plot_firing_rates() (ni.model.net_sim.SimulationResult method), 16
plot_firing_rates_per_channel() (ni.model.net_sim.SimulationResult method), 16
plot_interaction() (ni.model.net_sim.Net method), 15
plot_prototypes() (ni.model.ip.FittedModel method), 12
plotBootstrap() (in module ni.tools.bootstrap), 20
plotBootstrap() (in module ni.tools.strap), 28
plotCompareBootstrap() (in module ni.tools.bootstrap), 20
plotCompareBootstrap() (in module ni.tools.strap), 28
plotGaussed() (in module ni.model.pointprocess), 6
plotGaussed() (ni.model.pointprocess.PointProcess method), 3
plotHist() (ni.tools.statcollector.StatCollector method), 29
plotMultiSpikes() (in module ni.model.pointprocess), 7
plotParams() (ni.model.ip.FittedModel method), 12
plotTree() (ni.tools.statcollector.StatCollector method), 29
PointProcess (class in ni.model.pointprocess), 3
PointProcessFromSpikeTimes() (in module ni.model.pointprocess), 3
predict() (in module ni.model.backend_elasticnet), 18
predict() (in module ni.model.backend_glm), 18
predict() (ni.model.backend_elasticnet.Fit method), 17
predict() (ni.model.backend_glm.Fit method), 18
predict() (ni.model.ip.FittedModel method), 12
predict() (ni.model.ip.Model method), 13
predict() (ni.model.pointprocess.SimpleFiringRateModel method), 3
prefix() (ni.tools.statcollector.StatCollector method), 30
print_job_status() (ni.tools.project.Project method), 23
print_job_status() (ni.tools.project.Session method), 23
print_long_job_status() (ni.tools.project.Project method), 23
print_long_job_status() (ni.tools.project.Session method), 23
process() (ni.tools.html_view.View method), 26
progress() (in module ni.tools.progressbar), 20
progress_end() (in module ni.tools.progressbar), 20
progress_init() (in module ni.tools.progressbar), 20
Project (class in ni.tools.project), 22
prototypes() (ni.model.ip.FittedModel method), 12
pull_from_inner_dict() (ni.tools.statcollector.StatCollector method), 30
pvalues_by_component() (ni.model.ip.FittedModel method), 12

R

RateComponent (class in ni.model.designmatrix), 9
re() (ni.tools.statcollector.StatCollector method), 30
read_pickle() (ni.data.data.Data method), 33
read_pickle() (ni.model.ip.FittedModel method), 12
read_pickle() (ni.model.ip.Model method), 13
rename() (ni.tools.statcollector.StatCollector method), 30
render() (ni.tools.html_view.View method), 26
report() (in module ni.tools.project), 25
report() (ni.tools.project.CodeManager method), 21
report() (ni.tools.project.DataManager method), 21
report() (ni.tools.project.Project method), 23
reportHTML() (ni.tools.project.Project method), 23
require_job() (in module ni.tools.project), 25
require_job() (ni.tools.project.Project method), 23
reset_failed_jobs() (ni.tools.project.Project method), 23
reset_failed_jobs() (ni.tools.project.Session method), 23
reverse_correlation() (in module ni.model.pointprocess), 7
run() (in module ni.tools.project), 25
run() (ni.tools.project.Job method), 21
run() (ni.tools.project.Project method), 23

S

save() (in module ni.tools.project), 25
save() (ni.model.net_sim.Net method), 16
save() (ni.tools.html_view.View method), 26
save() (ni.tools.project.Job method), 21
save() (ni.tools.project.Project method), 23
save() (ni.tools.statcollector.StatCollector method), 30
save_html() (ni.tools.project.Project method), 23
save_html() (ni.tools.project.Session method), 23
savefig() (ni.tools.html_view.View method), 26
saveToFile() (in module ni.data.data), 33
SecondOrderHistoryComponent (class in ni.model.designmatrix), 9
select_session() (ni.tools.project.Project method), 23
Session (class in ni.tools.project), 23
set() (ni.tools.statcollector.StatCollector method), 30
set_activity() (ni.tools.project.Job method), 21
set_session_status() (ni.tools.project.Project method), 23
set_status() (ni.tools.project.Job method), 22
set_status() (ni.tools.project.Session method), 23
setMask() (ni.model.designmatrix.DesignMatrix method), 8
setMask() (ni.model.designmatrix.DesignMatrixTemplate method), 9
setup_jobs() (ni.tools.project.Project method), 23
setup_jobs() (ni.tools.project.Session method), 23
shape() (ni.data.data.Data method), 33
sibjob() (in module ni.tools.project), 25
sibjob() (ni.tools.project.Project method), 23
SimpleFiringRateModel (class in ni.model.pointprocess), 3

simulate() (in module ni.model.net_sim), 16
simulate() (ni.model.net_sim.Net method), 16
SimulationConfiguration (class in ni.model.net_sim), 16
SimulationResult (class in ni.model.net_sim), 16
spcol() (in module ni.model.create_splines), 17
spline() (in module ni.model.create_splines), 17
split() (ni.tools.statcollector.StatCollector method), 30
StatCollector (class in ni.tools.statcollector), 28
stopTimer() (ni.model.net_sim.SimulationResult method), 16
store() (ni.model.net_sim.SimulationResult method), 16
subjob() (in module ni.tools.project), 25
subjob() (ni.tools.project.Project method), 23
superjob() (in module ni.tools.project), 25
superjob() (ni.tools.project.Project method), 23

T

TemporaryJob (class in ni.tools.project), 24
TemporarySession (class in ni.tools.project), 24
to_pickle() (ni.data.data.Data method), 33
to_pickle() (ni.model.ip.FittedModel method), 12
to_pickle() (ni.model.ip.Model method), 13
Trial (class in ni.data.decoding_data), 33
trial() (ni.data.data.Data method), 33

U

update() (ni.tools.project.Job method), 22
update_job_files() (ni.tools.project.Session method), 23
update_job_status() (ni.tools.project.Project method), 23
update_jobs() (ni.tools.project.Session method), 24
upTriHalf() (in module ni.model.create_design_matrix_vk), 16

V

VariableContainer (class in ni.tools.project), 24
View (class in ni.tools.html_view), 25

X

x() (ni.model.ip.Model method), 13