Modelling Higher-Order Network Dynamics in the Presence of Triadic Interactions

Generated by Doxygen 1.9.8

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

Namespace Documentation

2.1 triadic_interaction Namespace Reference

Namespaces

- namespace computation
- namespace model
- namespace visualization

2.1.1 Detailed Description

```
Triadic Interactions Package.

This package contains a Node Dynamics with Triadic Interaction class and
a set of functions for computations and visualization.
```

2.2 triadic_interaction.computation Namespace Reference

Functions

- create_node_edge_incidence_matrix (list edge_list)
- extract_by_std (np.ndarray data, float n_std=3.)
- freedman_diaconis_rule (np.ndarray data, float power=1./3., float factor=2.)
- _check_data_shape (np.ndarray data)
- _generate_bins (np.ndarray data, str or np.ndarray or int or list or tuple bins)
- estimate_pdf (np.ndarray data, str or int or np.ndarray bins='fd', str method='kde')
- estimate_pdf_joint (np.ndarray data, str or np.ndarray or int or list or tuple bins='fd', str method='kde')
- estimate_pmf (np.ndarray data, str or int or np.ndarray bins='fd', str method='kde')
- estimate_pmf_joint (np.ndarray data, str or np.ndarray or int or list or tuple bins='fd', str method='kde')
- estimate_mutual_information (np.ndarray X, np.ndarray Y, str or int bins='fd', str pmf_method='kde', str method='kl-div')
- covariance (np.ndarray data)
- conditional_correlation (np.ndarray X, np.ndarray Y, np.ndarray Z, str or int bins='fd')
- conditional_mutual_information (np.ndarray X, np.ndarray Y, np.ndarray Z, str or int bins='fd', str pmf_↩ method='kde', str method='kl-div')

2.2.1 Detailed Description

```
Computation module.

This module provides functions for computations.
```

2.2.2 Function Documentation

2.2.2.1 _check_data_shape()

```
triadic_interaction.computation._check_data_shape (
            np.ndarray data )  [protected]
```

```
Check the shape of the data.

Parameters
----------
data : numpy.ndarray
    The data.

Raises
------
ValueError
    If the data is not of shape (n_observations,) or (1, n_observations).
```

2.2.2.2 _generate_bins()

```
triadic_interaction.computation._generate_bins (
            np.ndarray data,
            str or np.ndarray or int or list or tuple bins )  [protected]
```

```
Generate the bins.

Parameters
----------
data : numpy.ndarray of shape (n_observations,) or (1, n_observations) or (n_variables, n_observations)
    The data.
bins : str or a sequence of int or int or list or tuple
    The number of bins or the method to compute the number of bins.
        - 'fd' (str) : The number of bins is computed using the Freedman-Diaconis rule.
        - n (int) : The number of bins for the variable.
        - list or tuple (list or tuple of numpy.ndarray) : The bin edges for each variable.

Returns
-------
_bins : list of numpy.ndarray of shape (n_bins_{i},) [i = 1, ..., n_variables]
    The bins edges for each variable.
n_bins : int
    The number of bins for each variable.

Raises
------
ValueError
    If the bin edges are not monotonically increasing.
    If the length of bins is not equal to the number of variables.
    If the bin type is invalid.
```

2.2.2.3 conditional_correlation()

```
triadic_interaction.computation.conditional_correlation (
            np.ndarray X,
            np.ndarray Y,
            np.ndarray Z,
            str or int bins = 'fd' )
```

```
Compute the conditional variance.

Parameter
---------
X : numpy.ndarray of shape (n_observations,) or (1, n_observations)
    The time series data.
Y : numpy.ndarray of shape (n_observations,) or (1, n_observations)
    The time series data.
Z : numpy.ndarray of shape (n_observations,) or (1, n_observations)
    The time series data. (condition)
bins : int or str, optional
    (default = 'fd')
    The number of bins or the method to compute the number of bins.
    - 'fd' : Freedman-Diaconis rule
    - n (int) : The number of bins for the variable.

Returns
-------
corr_cond : numpy.ndarray of shape (n_bins,)
    The conditional correlation.
z : numpy.ndarray of shape (n_bins,)
    The bin values of the conditional variable.
corr_cond_err : numpy.ndarray of shape (n_bins,)
    The standard error of the conditional correlation.
```

2.2.2.4 conditional_mutual_information()

```
triadic_interaction.computation.conditional_mutual_information (
            np.ndarray X,
            np.ndarray Y,
            np.ndarray Z,
            str or int bins = 'fd',
            str pmf_method = 'kde',
            str method = 'kl-div' )
```

```
Calculate the conditional mutual information between X and Y given Z.

Parameters
----------
X : numpy.ndarray of shape (n_observations, ) or (1, n_observations)
    The data.
Y : numpy.ndarray of shape (n_observations, ) or (1, n_observations)
    The data.
Z : numpy.ndarray of shape (n_observations, ) or (1, n_observations)
    The data to be conditioned.
bins : str or a sequence of int or int, optional
    (default = None)
    The number of bins or the method to compute the number of bins.
    - 'fd' : The number of bins is computed using the Freedman-Diaconis rule.
    - n_1, n_2, ..., n_n : The number of bins for each variable.
    - n : The number of bins for all variables.
pmf_method : str, optional
    (default = 'kde')
    The method to estimate the probability mass function.
    - 'hist' : The probability density function is estimated by the histogram method.
```

```
    - 'kde' : The probability density function is estimated by the kernel density estimation.
method : str, optional
    (default = 'kl-div')
    The method to estimate the mutual information.
    - 'kl-div' : The mutual information is calculated by the Kullback-Leibler divergence.
    - 'entropy' : The mutual information is calculated from entropies.

Returns
-------
cmi : numpy.ndarray of shape (n_bins,)
    The conditional mutual information between X and Y given Z=z for each z in Z.
pmf_Z : numpy.ndarray of shape (n_bins,)
    The probability mass function of Z.
z : numpy.ndarray of shape (n_bins,)
    The z values of the corresponding bins.
```

2.2.2.5 covariance()

```
triadic_interaction.computation.covariance (
            np.ndarray data )
```

```
Calculate the covariance matrix.

Parameters
----------
data : numpy.ndarray of shape (n_nodes, n_timesteps, n_samples)
    The time series data.

Returns
-------
cov_np : numpy.ndarray of shape (n_samples, n_nodes * n_nodes)
    The covariance matrix.
```

2.2.2.6 create_node_edge_incidence_matrix()

```
triadic_interaction.computation.create_node_edge_incidence_matrix (
            list edge_list )
```

```
Create a node-edge incidence matrix B from a given edge list.

Parameters
----------
edge_list : list of tuples (i, j) where i < j
    The list of edges (i, j).

Returns
-------
B : numpy.ndarray of shape (n_nodes, n_edges)
    The node-edge incidence matrix.
```

2.2.2.7 estimate_mutual_information()

```
triadic_interaction.computation.estimate_mutual_information (
            np.ndarray X,
            np.ndarray Y,
            str or int bins = 'fd',
            str pmf_method = 'kde',
            str method = 'kl-div' )
```

```
Calculate the mutual information between X and Y.

Parameters
----------
X : numpy.ndarray of shape (n_observations,) or (1, n_observations)
    The data.
Y : numpy.ndarray of shape (n_observations,) or (1, n_observations)
    The data.
bins : str or a sequence of int or int, optional
    (default = None)
    The number of bins or the method to compute the number of bins.
    - 'fd' : The number of bins is computed using the Freedman-Diaconis rule.
    - n : The number of bins for the variable.
pmf_method : str, optional
    (default = 'kde')
    The method to estimate the probability mass function.
    - 'hist' : The probability density function is estimated by the histogram method.
    - 'kde' : The probability density function is estimated by the kernel density estimation.
method : str, optional
    (default = 'kl-div')
    The method to estimate the mutual information.
    - 'kl-div' : The mutual information is calculated by the Kullback-Leibler divergence.
    - 'entropy' : The mutual information is calculated from entropies.

Returns
-------
mi : float
    The mutual information between X and Y.
```

2.2.2.8 estimate_pdf()

```
triadic_interaction.computation.estimate_pdf (
            np.ndarray data,
            str or int or np.ndarray bins = 'fd',
            str method = 'kde' )
```

```
Estimate the probability density function of the data.

Parameters
----------
data : numpy.ndarray of shape (n_observations,) or (1, n_observations)
    The data.
bins : str or a sequence of int or int, optional
    (default = None)
    The number of bins or the method to compute the number of bins.
        - 'fd' (str) : The number of bins is computed using the Freedman-Diaconis rule.
        - n (int) : The number of bins for the variable.
        - numpy.ndarray : The bin edges for the variable.
method : str, optional
    (default = 'kde')
    The method to estimate the probability density function.
        - 'hist' : The probability density function is estimated by the histogram method.
        - 'kde' : The probability density function is estimated by the kernel density estimation.

Returns
```

```
-------
pdf : numpy.ndarray of shape (n_bins, )
    The estimated probability density function.
x : numpy.ndarray of shape (n_bins, )
    The x values.
```

2.2.2.9 estimate_pdf_joint()

```
triadic_interaction.computation.estimate_pdf_joint (
             np.ndarray data,
             str or np.ndarray or int or list or tuple bins = 'fd',
             str method = 'kde' )
```

```
Estimate the joint probability density function of the data by the histogram method.

Parameters
----------
data : numpy.ndarray of shape (n_variables, n_observations)
    The data.
bins : str or a sequence of int or int or list, optional
    (default = None)
    The number of bins or the method to compute the number of bins.
        - 'fd' (str) : The number of bins is computed using the Freedman-Diaconis rule.
        - n_{1}, n_{2}, ..., n_{n_variables} (sequence of int) : The number of bins for each variable.
        - n (int): The number of bins for all variables.
        - list (list of numpy.ndarray) : The bin edges for each variable.
method : str, optional
    (default = 'kde')
    The method to estimate the probability density function.
        - 'hist' : The probability density function is estimated by the histogram method.
        - 'kde' : The probability density function is estimated by the kernel density estimation.

Returns
-------
pdf_joint : numpy.ndarray of shape (n_bins_{1}, ..., n_bins_{n_variables})
    The estimated probability density function.
x : list of numpy.ndarray of shape (n_bins_{i},) [i = 1, ..., n_variables]
    The x values of the corresponding bins.
```

2.2.2.10 estimate_pmf()

```
triadic_interaction.computation.estimate_pmf (
             np.ndarray data,
             str or int or np.ndarray bins = 'fd',
             str method = 'kde' )
```

```
Estimate the probability mass function of the data by the histogram method.

Parameters
----------
data : numpy.ndarray of shape (n_observations,) or (1, n_observations)
    The data.
bins : str or a sequence of int or int, optional
    (default = None)
    The number of bins or the method to compute the number of bins.
    - 'fd' : The number of bins is computed using the Freedman-Diaconis rule.
    - n : The number of bins for the variable.
    - np.ndarray : The bin edges for the variable.
method : str, optional
    (default = 'kde')
    The method to estimate the probability density function.
```

```
    - 'hist' : The probability density function is estimated by the histogram method.
    - 'kde' : The probability density function is estimated by the kernel density estimation.

Returns
-------
pmf : numpy.ndarray of shape (n_bins, )
    The estimated probability mass function of the data.
X : numpy.ndarray of shape (n_bins, )
    The bin centers for variables.
```

2.2.2.11 estimate_pmf_joint()

```
triadic_interaction.computation.estimate_pmf_joint (
            np.ndarray data,
            str or np.ndarray or int or list or tuple bins = 'fd',
            str method = 'kde' )


Estimate the joint probability mass function of the data.

Parameters
----------
data : numpy.ndarray of shape (n_variables, n_observations)
    The data.
bins : str or a sequence of int or int or list, optional
    (default = None)
    The number of bins or the method to compute the number of bins.
    - 'fd' : The number of bins is computed using the Freedman-Diaconis rule.
    - n_{1}, n_{2}, ..., n_{n_variable} : The number of bins for each variable.
    - n : The number of bins for all variables.
    - list : The bin edges for each variable.
method : str, optional
    (default = 'kde')
    The method to estimate the probability density function.
    - 'hist' : The probability density function is estimated by the histogram method.
    - 'kde' : The probability density function is estimated by the kernel density estimation.

Returns
-------
pmf_joint : numpy.ndarray of shape (n_bins^{1}, n_bins^{2}, ..., n_bins^{n_variables})
    The estimated probability mass function of the data.
x : list of numpy.ndarray of shape (n_bins^{i},) [i = 1, ..., n_variables]
    The x values of the corresponding bins.
```

2.2.2.12 extract_by_std()

```
triadic_interaction.computation.extract_by_std (
            np.ndarray data,
            float n_std = 3.  )


Extract the data within a given number of standard deviations from its mean.

Parameters
----------
data : numpy.ndarray of shape (n_observations, )
    The data.
n_std : float, optional
    (default = 3.0)
    The number of standard deviations to extract.

Returns
-------
```

```
data_min : float
    The minimum value in the extracted range of data .
data_max : float
    The maximum value in the extracted range of data.

Raises
------
ValueError
    If the data is not of shape (n_observations, ).
```

2.2.2.13 freedman_diaconis_rule()

```
triadic_interaction.computation.freedman_diaconis_rule (
             np.ndarray data,
             float power = 1./3.,
             float factor = 2.  )
```

```
Compute the number of bins using the Freedman-Diaconis rule.

Parameters
----------
data : numpy.ndarray of shape (n_variables, n_observations)
    The data matrix.
power : float, optional
    (default = 1./3.)
    The power of the number of observations in the denominator.
factor : float, optional
    (default = 2.)
    The factor to multiply the width of bins.

Returns
-------
bins_edges : numpy.ndarray of shape (n_bins,) or list of numpy.ndarray of shape (n_bins_{i},) [i = 1, ..., n_v
    The bins edges.

Raises
------
ValueError
    If the data is not a one-dimensional or two-dimensional array.
```

2.3 triadic_interaction.model Namespace Reference

Classes

- class NDwTIs

2.3.1 Detailed Description

```
Node Dynamics with Triadic Interactions Class.

This class implements the model of node dynamics with triadic interactions.
```

2.4 triadic_interaction.visualization Namespace Reference

Functions

- plot_timeseries (np.ndarray X, str output_file, float t_max, int n_samples=1, bool separate=False, theory=None)
- plot_pdf (list probs, list bins, str output_file, f_theory=None, bool logscale=False, bool parallel=False)
- plot_covariance (np.ndarray cov, str output_file, theory=None)
- plot_conditional_correlation (np.ndarray or list Xgrids, np.ndarray or list cond_corr, tuple or list order, str output_file, bool or list std=False, tuple or list Xrange=None, theory=None, f_supplement=None, float or list threshold=None)
- plot_conditional_mutual_information (np.ndarray or list Xgrids, np.ndarray or list cmi, tuple or list order, str output_file, bool or list std=False, tuple or list Xrange=None, theory=None)

2.4.1 Detailed Description

```
Visualization module.

This module contains functions for visualizing the results of the node dynamics with triadic interactions.
```

2.4.2 Function Documentation

2.4.2.1 plot_conditional_correlation()

```
triadic_interaction.visualization.plot_conditional_correlation (
             np.ndarray or list Xgrids,
             np.ndarray or list cond_corr,
             tuple or list order,
             str output_file,
             bool or list std = False,
             tuple or list Xrange = None,
              theory = None,
              f_supplement = None,
             float or list threshold = None )
```

```
Plot the conditional correlation.

Parameter
---------
Xgrids : numpy.ndarray of shape (n_bins,) or list of numpy.ndarray of shape (n_bins,)
    The grid of the conditional variable.
cond_corr : numpy.ndarray of shape (n_bins, n_samples) or list of numpy.ndarray of shape (n_bins, n_samples)
    The conditional correlation.
order : tuples or list of tuples
    The order of the nodes.
output_file : str
    The output file name.
std : bool or  list of numpy.ndarray of shape (n_bins, n_samples), optional
     (default = False)
    If False, do not plot the standard error.
    If list, plot the standard error.
Xrange : bool or list of tuples, optional
         (default = None)
    If None, do not set the range of the x-axis.
    If list, set the range of the x-axis.
theory : function or list of functions, optional
         (default value = None)
```

```
    The theoretical solutions.
f_supplement : function or list of functions, optional
        (default value = None)
    The supplementary functions.
threshold : float or list of float, optional
        (default value = None)
    The threshold value
Returns
-------
None
```

2.4.2.2 plot_conditional_mutual_information()

```
triadic_interaction.visualization.plot_conditional_mutual_information (
             np.ndarray or list Xgrids,
             np.ndarray or list cmi,
             tuple or list order,
             str output_file,
             bool or list std = False,
             tuple or list Xrange = None,
              theory = None )
```

```
Plot conditional mutual information.

Parameter
---------
Xgrids : numpy.ndarray of shape (n_bins,) or list of numpy.ndarray of shape (n_bins,)
    The grid of the conditional variable.
cmi : numpy.ndarray of shape (n_bins, ) or list of numpy.ndarray of shape (n_bins, )
    The conditional mutual information.
order : tuples or list of tuples
    The order of the nodes.
output_file : str
    The output file name.
std : bool, optional
     (default = False)
    If True, plot the standard deviation.
Xrange : bool or list of tuples, optional
    (default = None)
    If None, do not set the range of the x-axis.
    If list, set the range of the x-axis.
theory : function or list of functions, optional
        (Default value = None)
    The theoretical solutions.

Returns
-------
None
```

2.4.2.3 plot_covariance()

```
triadic_interaction.visualization.plot_covariance (
             np.ndarray cov,
             str output_file,
              theory = None )
```

```
Plot the covariance matrix.

Parameters
----------
```

```
cov : numpy.ndarray of shape (n_nodes, n_nodes)
    The covariance matrix.
output_file : str
    The output file name.
theory : function, optional
     (Default value = None)
    The theoretical solution.

Returns
-------
None
```

2.4.2.4 plot_pdf()

```
triadic_interaction.visualization.plot_pdf (
             list probs,
             list bins,
             str output_file,
              f_theory = None,
             bool logscale = False,
             bool parallel = False )
```

```
Plot the probability distributions.

Parameters
----------
probs : list of numpy.ndarray of shape (n_nodes, n_bins)
    The probability distributions for all nodes.
bins : list of numpy.ndarray of shape (n_nodes, n_bins)
    The bins for all nodes.
output_file : str
    The output file name.
f_theory : function, optional
     (Default value = None)
    The theoretical solution.
logscale : bool, optional
     (Default value = False)
    If True, plot the log scale.
parallel : bool, optional
     (Default value = False)
    If True, plot each node separately.

Returns
-------
None
```

2.4.2.5 plot_timeseries()

```
triadic_interaction.visualization.plot_timeseries (
             np.ndarray X,
             str output_file,
             float t_max,
             int n_samples = 1,
             bool separate = False,
              theory = None )
```

```
Plot the timeseries.

Parameters
----------
X : numpy.ndarray of shape (n_nodes, n_timesteps, n_samples)
    The timeseries data.
output_file : str
    The output file name.
t_max : float
    The maximum time span.
n_samples : int, optional
     (Default value = 1)
    The number of samples.
separate : bool, optional
     (Default value = False)
    If True, plot each node separately.
theory : function, optional
     (Default value = None)
    The theoretical solution.

Returns
-------
None
```

Chapter 3

Class Documentation

3.1 triadic_interaction.model.NDwTIs Class Reference

Public Member Functions

- __init__ (self, np.ndarray B, np.ndarray K, float w_pos, float w_neg, float threshold, float alpha, float noise_std, external_force=None, np.ndarray x_init=None, float dt=0.01, float t_max=1.)
- np.ndarray getLaplacian (self, np.ndarray x)
- np.ndarray derivative (self, np.ndarray x, float t)
- np.ndarray noise (self, np.ndarray x, float t)
- np.ndarray integrate (self, bool deterministic=False)
- np.ndarray run (self, bool deterministic=False)

3.1.1 Detailed Description

```
Node Dynamics with Triadic Interactions.

Parameters
----------
B : numpy.ndarray of shape (n_nodes, n_edges)
    the boundary operator of the structural network
K : numpy.ndarray of shape (n_edges, n_nodes)
    the regulator network (structure of triadic interactions)
w_pos : float
    the weight of positive regulator
w_neg : float
    the weight of negative regulator
threshold : float
    the threshold parameter
alpha : float
    the coefficient of the triadic Laplacian
noise_std : float
    the standard deviation of the Gaussian noise
external_force : function, default = None
    the external force as a function of time
x_init : numpy.ndarray, default = None
    the initial states of nodes
dt : float, default = 0.01
    the time step size of the evolution
t_max : float, default = 1.
    the time duration of the evolution

Returns
-------
```

```
Attributes
----------
n_nodes : int
    the number of nodes in the structural network

n_edges : int
    the number of edges in the structural network

n_hyperedges : int
    the number of triadic interactions

n_pos_regulators : int
    the number of positive regulators

n_neg_regulators : int
    the number of negative regulators

n_timesteps : int
    the number of timesteps
```

3.1.2 Constructor & Destructor Documentation

3.1.2.1 __init__()

```
triadic_interaction.model.NDwTIs.__init__ (
              self,
             np.ndarray B,
             np.ndarray K,
             float w_pos,
             float w_neg,
             float threshold,
             float alpha,
             float noise_std,
              external_force = None,
             np.ndarray x_init = None,
             float dt = 0.01,
             float t_max = 1.  )


Initialise the triadic interaction null model.

Parameters
----------
B : numpy.ndarray of shape (n_nodes, n_edges)
    The boundary operator of the structural network.
K : numpy.ndarray of shape (n_edges, n_nodes)
    The regulator network.
w_pos : float
    The weight of positive regulator.
w_neg : float
    The weight of negative regulator.
threshold : float
    The threshold parameter.
alpha : float
    The coefficient of the triadic Laplacian.
noise_std : float
    The standard deviation of the Gaussian noise.
external_force : function, optional (default = None)
    The external force as a function of time.
x_init : numpy.ndarray, optional (default = None)
    The initial states of nodes.
dt : float, optional (default = 0.01)
    The time step size of the evolution.
t_max : float, optional (default = 1)
    The time duration of the evolution.
```

3.1.3 Member Function Documentation

3.1.3.1 derivative()

```
np.ndarray triadic_interaction.model.NDwTIs.derivative (
              self,
             np.ndarray x,
             float t )
```

```
The time-derivatives of the states.

Parameters
----------
x : numpy.ndarray of shape (n_nodes,)
    The states of nodes.
t : float
    The time.

Returns
-------
dxdt : numpy.ndarray of shape (n_nodes,)
    The time-derivatives of the states.
```

3.1.3.2 getLaplacian()

```
np.ndarray triadic_interaction.model.NDwTIs.getLaplacian (
              self,
             np.ndarray x )
```

```
Compute the Laplacian of the states.

Parameters
----------
x : numpy.ndarray of shape (n_nodes,)
    The states of nodes.

Returns
-------
L : numpy.ndarray of shape (n_nodes, n_nodes)
    The Laplacian of the states.
```

3.1.3.3 integrate()

```
np.ndarray triadic_interaction.model.NDwTIs.integrate (
              self,
             bool deterministic = False )
```

```
Evolve the system.

Parameters
----------
deterministic : bool, optional (default = False)
    If True, the integration is deterministic. (Default value = False)

Returns
-------
timeseries : numpy.ndarray of shape (n_nodes, n_timesteps)
    The time series of the states.
```

3.1.3.4 noise()

```
np.ndarray triadic_interaction.model.NDwTIs.noise (
              self,
             np.ndarray x,
             float t )
```

```
The coeficients of the noise term.

Parameters
----------
x : numpy.ndarray of shape (n_nodes,)
    The states of nodes.
t : float
    The time.

Returns
-------
noise : numpy.ndarray of shape (n_nodes, n_nodes)
    The coeficients of the noise term.
```

3.1.3.5 run()

```
np.ndarray triadic_interaction.model.NDwTIs.run (
              self,
             bool deterministic = False )
```

```
Run the system.

Parameters
----------
deterministic : bool, optional (default = False)
    If True, the model runs deterministically. (Default value = False)

Returns
-------
timeseries : numpy.ndarray of shape (n_nodes, n_timesteps)
    The time series of the states.
```

Index