Node Dynamic with Triadic Interactions

Generated by Doxygen 1.9.7

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

Namespace Documentation

3.1 triadic_interaction Namespace Reference

Namespaces

- namespace computation
- namespace model
- namespace visualization

3.1.1 Detailed Description

```
Triadic Interactions Package.

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

3.2 triadic_interaction.computation Namespace Reference

Functions

- create_node_edge_incidence_matrix (edge_list)
- extract_by_std (X, std=3.0)
- freedman_diaconis_rule (data, power=1./3., factor=2., trim=1)
- discretise (X, n_bins='fd')
- estimate_pdf (data, bins='fd', method='hist')
- estimate_pdf_joint (data, bins='fd', method='hist')
- estimate_pdf_conditional (data, data_cond, val_cond, bins='fd', method='hist')
- pdf_evolution (X, t_max, n_x_resolution=50)
- covariance (data)
- conditional_expectation (X, Y, Z, bins='fd')
- conditional_variance (X, Z, bins='fd')
- conditional_covariance (X, Y, Z, bins='fd')
- conditional_correlation (X, Y, Z, bins='fd', method='default')
- entropy (pdf, x)
- entropy_joint (pdf_joint, x)
- conditional_mutual_information (X, Y, Z, bins='fd', method='hist')

3.2.1 Detailed Description

```
This module provides functions for computations.

Functions
---------
- create_node_edge_incidence_matrix :
    Create a node-edge incidence matrix B from a given edge list.
- extract_by_std :
    Extract the data within a given number of standard deviations from its mean.
- freedman_diaconis_rule :
    Compute the optimal bin width for a histogram.
- discretise :
    Discretise the data.
- estimate_pdf :
    Estimate the probability density function.
- estimate_pdf_joint :
    Estimate the joint probability density function.
- estimate_pdf_conditional :
    Estimate the conditional probability density function.
- pdf_evolution :
    Compute the evolution of the probability density function.
- covariance :
    Compute the covariance.
- conditional_expectation :
    Compute the conditional expectation.
- conditional_variance :
    Compute the conditional variance.
- conditional_covariance :
    Compute the conditional covariance.
- conditional_correlation :
    Compute the conditional correlation.
- entropy :
    Compute the entropy.
- entropy_joint :
    Compute the joint entropy.
- conditional_mutual_information :
    Compute the conditional mutual information.
```

3.2.2 Function Documentation

3.2.2.1 conditional_correlation()

```
triadic_interaction.computation.conditional_correlation (
              X,
              Y,
              Z,
              bins = 'fd',
              method = 'default' )
```

```
Compute the conditional variance.

Parameter
---------
X : numpy.ndarray of shape (n_timesteps, n_samples)
    The time series data.
Y : numpy.ndarray of shape (n_timesteps, n_samples)
    The time series data.
Z : numpy.ndarray of shape (n_timesteps, n_samples)
    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
method : str, optional
```

```
    (default = ’default’)
    The method to compute the conditional correlation.
    - ’default’ : Pearson correlation coefficient
    - ’manual’ : manual computation

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.
```

3.2.2.2 conditional_covariance()

```
triadic_interaction.computation.conditional_covariance (
              X,
              Y,
              Z,
              bins = 'fd' )
```

```
Compute the conditional variance.

Parameter
---------
X : numpy.ndarray of shape (n_timesteps, n_samples)
    The time series data.
Y : numpy.ndarray of shape (n_timesteps, n_samples)
    The time series data.
Z : numpy.ndarray of shape (n_timesteps, n_samples)
    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

Returns
-------
cov_cond : numpy.ndarray of shape (n_bins,)
    The conditional covariance.
z : numpy.ndarray of shape (n_bins,)
    The bin values of the conditional variable.
```

3.2.2.3 conditional_expectation()

```
triadic_interaction.computation.conditional_expectation (
              X,
              Y,
              Z,
              bins = 'fd' )
```

```
Conditional expectation.

Parameter
---------
X : numpy.ndarray of shape (n_timesteps, n_samples)
    The time series data.
Y : numpy.ndarray of shape (n_timesteps, n_samples)
    The time series data.
Z : numpy.ndarray of shape (n_timesteps, n_samples)
```

```
    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.

Returns
-------
z_bins : numpy.ndarray of shape (n_bins,)
    The bin values of the conditional variable.
mean : tuple of 2 numpy.ndarray of shape (n_bins,)
    The conditional expectation of node i and j
        - X1_mean : numpy.ndarray of shape (n_bins,)
            The conditional expectation of node i.
        - X2_mean : numpy.ndarray of shape (n_bins,)
            The conditional expectation of node j.
std : tuple of 2 numpy.ndarray of shape (n_bins,)
    The standard deviation of node i and j
        - X1_std : numpy.ndarray of shape (n_bins,)
            The conditional standard deviation of node i.
        - X2_std : numpy.ndarray of shape (n_bins,)
            The conditional standard deviation of node j.
X3_dig : numpy.ndarray of shape (n_samples,)
    The digitised data of node v3.
```

3.2.2.4 conditional_mutual_information()

```
triadic_interaction.computation.conditional_mutual_information (
              X,
              Y,
              Z,
              bins = 'fd',
              method = 'hist' )
```

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

Parameters
----------
X : numpy.ndarray of shape (n_observations, )
    The data.
Y : numpy.ndarray of shape (n_observations, )
    The data.
Z : numpy.ndarray of shape (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.
method : str, optional
    (default = 'hist')
    The method to estimate the probability density function.
    - 'hist' : The pdf is estimated by the histogram method.
    - 'kde' : The pdf is estimated by the kernel density estimation method.

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

3.2.2.5 conditional_variance()

```
triadic_interaction.computation.conditional_variance (
              X,
              Z,
              bins = 'fd' )
```

```
Calculate the conditional variance.

Parameter
---------
X : numpy.ndarray of shape (n_timesteps, n_samples)
    The time series data.
Z : numpy.ndarray of shape (n_timesteps, n_samples)
    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

Returns
-------
var_cond : numpy.ndarray of shape (n_bins,)
    The conditional variance.
z : numpy.ndarray of shape (n_bins,)
    The bin values of the conditional variable.
```

3.2.2.6 covariance()

```
triadic_interaction.computation.covariance (
              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.
```

3.2.2.7 create_node_edge_incidence_matrix()

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

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

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

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

3.2.2.8 discretise()

```
triadic_interaction.computation.discretise (
              X,
              n_bins = 'fd' )
```

```
Discretise the time series data.

Parameters
----------
X : numpy.ndarray of shape (n_observation, n_variables)
    The data matrix.
n_bins : int or str, optional
    (default = ’fd’)
    The number of bins or the method to compute the number of bins.
    - ’fd’ : Freedman-Diaconis rule

Returns
-------
X_discrete : numpy.ndarray of shape (n_observation, n_variables)
    The discretised data matrix.
bins : list of numpy.ndarray of shape (n_bins,)
    The list of the bins of the values.
```

3.2.2.9 entropy()

```
triadic_interaction.computation.entropy (
              pdf,
              x )
```

```
Calculate the entropy of the probability density function.

Parameters
----------
pdf : numpy.ndarray of shape (n_bins,)
    The probability density function.
x : numpy.ndarray of shape (n_bins,)
    The x values of the corresponding bins.

Returns
-------
entropy : float
    The entropy of the probability density function.
```

3.2.2.10 entropy_joint()

```
triadic_interaction.computation.entropy_joint (
              pdf_joint,
              x )
```

```
Calculate the joint entropy of the probability density function.

Parameters
----------
pdf_joint : numpy.ndarray of shape (n_bins, n_variables)
    The joint probability density function.
x : list of numpy.ndarray of shape (n_bins,)
    The x values of the corresponding bins.

Returns
-------
entropy_joint : float
    The joint entropy of the probability density function.
```

3.2.2.11 estimate_pdf()

```
triadic_interaction.computation.estimate_pdf (
              data,
              bins = 'fd',
              method = 'hist' )
```

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

Parameters
----------
data : numpy.ndarray of shape (n_observations,) or (n_observations, 1)
    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.
method : str, optional
    (default = 'hist')
    The method to estimate the probability density function (pdf).
    - 'hist' : The pdf is estimated by the histogram method.
    - 'kde' : The pdf is estimated by the kernel density estimation method.

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

3.2.2.12 estimate_pdf_conditional()

```
triadic_interaction.computation.estimate_pdf_conditional (
              data,
              data_cond,
              val_cond,
              bins = 'fd',
              method = 'hist' )
```

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

Parameters
----------
data : numpy.ndarray of shape (n_observations, n_variables)
    The data.
data_cond : numpy.ndarray of shape (n_observations, n_conditional_variables)
    The data to be conditioned on.
val_cond : int
    The value of the variable to condition on.
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.
method : str, optional
    (default = 'hist')
    The method to estimate the probability density function (pdf).
    - 'hist' : The pdf is estimated by the histogram method.
    - 'kde' : The pdf is estimated by the kernel density estimation method.

Returns
```

```
-------
pdf_conditional : numpy.ndarray of shape (n_bins, n_variables)
    The estimated probability density function of the data.
x : list of numpy.ndarray of shape (n_bins-1,)
    The x values of the corresponding bins.
```

3.2.2.13 estimate_pdf_joint()

```
triadic_interaction.computation.estimate_pdf_joint (
              data,
              bins = 'fd',
              method = 'hist' )
```

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

Parameters
----------
data : numpy.ndarray of shape (n_observations, n_variables)
    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 : 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 = 'hist')
    The method to estimate the probability density function (pdf).
    - 'hist' : The pdf is estimated by the histogram method.
    - 'kde' : The pdf is estimated by the kernel density estimation method.

Returns
-------
pdf_joint : numpy.ndarray of shape (n_bins, n_variables)
    The estimated probability density function of the data.
x : list of numpy.ndarray of shape (n_bins-1,)
    The x values of the corresponding bins.
```

3.2.2.14 extract_by_std()

```
triadic_interaction.computation.extract_by_std (
              X,
              std = 3.0 )
```

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

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

Returns
-------
X_min : float
    The minimum value of the core range.
X_max : float
    The maximum value of the core range.
```

3.2.2.15 freedman_diaconis_rule()

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

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

Parameters
----------
data : numpy.ndarray of shape (n_observations,)
    The data.
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.
trim : int, optional
    (default = 1)
    The ratio of the number of observations to trim from each end of the data.

Returns
-------
bins_edges : numpy.ndarray of shape (n_bins,)
    The bins edges.
```

3.2.2.16 pdf_evolution()

```
triadic_interaction.computation.pdf_evolution (
              X,
              t_max,
              n_x_resolution = 50 )
```

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

Parameters
----------
X : numpy.ndarray of shape (n_nodes, n_timesteps, n_variables)
    The data.
t_max : float
    The maximum time.
n_x_resolution : int, optional
    (default = 50)
    The number of bins to estimate the probability density function.

Returns
-------
time_evolution : numpy.ndarray of shape (n_nodes, n_x_resolution, n_timesteps)
    The time evolution of the probability density function.
x_grid : numpy.ndarray of shape (n_x_resolution,)
    The x values of the corresponding bins.
time_grid : numpy.ndarray of shape (n_timesteps,)
    The time steps.
```

3.3 triadic_interaction.model Namespace Reference

Classes

- class NDwTIs

3.3.1 Detailed Description

```
Node Dynamics with Triadic Interactions Class.

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

3.4 triadic_interaction.visualization Namespace Reference

Functions

- plot_timeseries (X, output_file, t_max, n_samples=1, separate=False, theory=None)
- plot_pdf (probs, bins, output_file, f_theory=None, logscale=False, parallel=False)
- plot_covariance (cov, output_file, theory=None)
- plot_conditional_expectation (Xgrids, cond_exps, stds, orders, output_file=None, theory=None)
- plot_conditional_correlation (Xgrids, cond_corr, order, output_file, std=False, Xrange=None, theory=None, f_supplement=None, threshold=None)
- plot_conditional_mutual_information (Xgrids, cmi, order, output_file, std=False, theory=None)
- visualise_evolution (evolution_data, x_grid, time_grid, output_file)

Variables

- dict **MPL_CONFIG**

3.4.1 Detailed Description

```
Visualization module.

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

3.4.2 Function Documentation

3.4.2.1 plot_conditional_correlation()

```
triadic_interaction.visualization.plot_conditional_correlation (
              Xgrids,
              cond_corr,
              order,
              output_file,
              std = False,
              Xrange = None,
              theory = None,
              f_supplement = None,
              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 : list of float, optional
        (default value = None)
    The threshold value
Returns
-------
None
```

3.4.2.2 plot_conditional_expectation()

```
triadic_interaction.visualization.plot_conditional_expectation (
              Xgrids,
              cond_exps,
              stds,
              orders,
              output_file = None,
              theory = None )
```

```
Plot the conditional expectation.

Parameter
---------
Xgrids : numpy.ndarray of shape (n_bins,) or list of numpy.ndarray of shape (n_bins,)
    The grid of the conditional variable.
cond_exps : numpy.ndarray of shape (n_bins, n_samples) or list of numpy.ndarray of shape (n_bins, n_samples)
    The conditional expectations.
std : numpy.ndarray of shape (n_bins, n_samples) or list of numpy.ndarray of shape (n_bins, n_samples)
    The standard deviation of the conditional expectations.
orders : tuples or list of tuples
    The order of the nodes.
output_file : str, optional
    The output file name.
theory : function or list of functions, optional
        (Default value = None)
    The theoretical solutions.

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

3.4.2.3 plot_conditional_mutual_information()

```
triadic_interaction.visualization.plot_conditional_mutual_information (
              Xgrids,
              cmi,
              order,
              output_file,
              std = False,
              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, n_samples) or list of numpy.ndarray of shape (n_bins, n_samples)
    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.
theory : function or list of functions, optional
        (Default value = None)
    The theoretical solutions.

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

3.4.2.4 plot_covariance()

```
triadic_interaction.visualization.plot_covariance (
              cov,
              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
```

3.4.2.5 plot_pdf()

```
triadic_interaction.visualization.plot_pdf (
              probs,
              bins,
              output_file,
              f_theory = None,
              logscale = False,
              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
```

3.4.2.6 plot_timeseries()

```
triadic_interaction.visualization.plot_timeseries (
              X,
              output_file,
              t_max,
              n_samples = 1,
              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
```

3.4.2.7 visualise_evolution()

```
triadic_interaction.visualization.visualise_evolution (
              evolution_data,
              x_grid,
              time_grid,
              output_file )
```

```
Visualise the evolution of probability density function.

Parameters
----------
evolution_data : numpy.ndarray of shape (n_nodes, n_bins, n_times)
    The evolution of probability density function.
x_grid : numpy.ndarray of shape (n_bins,)
    The grid of the variable.
time_grid : numpy.ndarray of shape (n_times,)
    The time grid.
output_file : str
    The output file name.

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

Chapter 4

Class Documentation

4.1 triadic_interaction.model.NDwTIs Class Reference

Public Member Functions

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

Public Attributes

- **B**
- **n_nodes**
- **n_edges**
- **K**
- **w_pos**
- **w_neg**
- **n_hyperedges**
- **n_pos_regulators**
- **n_reg_regulators**
- **alpha**
- **threshold**
- **noise_std**
- **external_force**
- **dt**
- **t_max**
- **n_timesteps**
- **x_init**

4.1.1 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
```

4.1.2 Constructor & Destructor Documentation

4.1.2.1 __init__()

```
triadic_interaction.model.NDwTIs.__init__ (
              self,
              B,
              K,
              w_pos,
              w_neg,
              threshold,
              alpha,
              noise_std,
              external_force = None,
              x_init = None,
              dt = 0.01,
              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.
```

4.1.3 Member Function Documentation

4.1.3.1 derivative()

```
triadic_interaction.model.NDwTIs.derivative (
              self,
              x,
              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.
```

4.1.3.2 getLaplacian()

```
triadic_interaction.model.NDwTIs.getLaplacian (
              self,
              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.
```

4.1.3.3 integrate()

```
triadic_interaction.model.NDwTIs.integrate (
              self,
              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.
```

4.1.3.4 noise()

```
triadic_interaction.model.NDwTIs.noise (
              self,
              x,
              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.
```

4.1.3.5 run()

```
triadic_interaction.model.NDwTIs.run (
              self,
              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.
```

The documentation for this class was generated from the following file:

- model.py

Index