phasik

Phasik Developers

Aug 27, 2023

HOME

1 What is Phasik? 3

2 Install Phasik 5

3 Getting Started 7

4 How to Contribute 9

5 How to Cite 11

6 Developers 13

7 Contact 15

8 License 17

9 Other Resources 19

10 Temporal networks 21

11 Inferring the multiscale temporal organisation of temporal networks 23

12 Workflow 25

13 Building a temporal network 27
13.1 Example from static network and node time series . . . 28
13.2 To go further . . . 30

14 Inferring phases 31
14.1 Building the temporal network . . . 31
14.2 Inferring phases . . . 32

15 Classes 35
15.1 Temporal Network . . . 35
15.2 Partially Temporal Network . . . 45
15.3 TemporalData . . . 46
15.4 DistanceMatrix . . . 48
15.5 ClusterSet . . . 49
15.6 ClusterSets . . . 52

16 Drawing 55
16.1 Clusters . . . 55

16.2 Networks . . . 58
16.3 Drawing . . . 59
16.4 Utils . . . 61

17 Utils **63**
17.1 Clusters . . . 63
17.2 Graphs . . . 64
17.3 Paths . . . 65

Python Module Index **67**

Index **69**

The Phasik package was created to infer temporal phases in temporal networks. It contains various utility classes and functions that can be divided into two main parts:

1. Build, analyse, and visualise temporal networks from time series data.
2. Infer temporal phases by clustering the snapshots of the temporal network.

Code on Gitlab: https://gitlab.com/habermann_lab/phasik
PiPy: https://pypi.org/project/phasik/

CHAPTER

ONE

WHAT IS PHASIK?

Phasik is a Python library for identifying temporal phases in complex systems modeled as temporal networks. The library contains classes and functions that can be divided into two main parts:

1. Build, analyse, and visualise temporal networks from time series data.
2. Identify temporal phases, over multiple scales, by clustering the snapshots of the temporal network.

- Source
- PiPy
- Bug reports
- Documentation
- Tutorials
- Notebooks

CHAPTER

TWO

INSTALL PHASIK

Phasik runs on Python 3.7 or higher. Install the latest version of *phasik* with *pip*:

```
pip install phasik
```

Alternatively, you can clone the repository manually or with *git*, go to the repository and then install locally with *pip*:

```
pip install .
```

You can also simply try *phasik* by cloning the repository without installing the package.

To install for development purposes, first clone the repository and then execute

```
pip install -e .['all']
```

If that command does not work, you may try the following instead

```
pip install -e .\[all\]
```

CHAPTER

THREE

GETTING STARTED

To get started, take a look at the tutorials illustrating the library's basic functionality.

CHAPTER

FOUR

HOW TO CONTRIBUTE

If you want to contribute to this project, please make sure to read the contributing guidelines. We expect respectful and kind interactions by all contributors and users as laid out in our code of conduct.

The Phasik community always welcomes contributions, no matter how small. We're happy to help troubleshoot Phasik issues you run into, assist you if you would like to add functionality or fixes to the codebase, or answer any questions you may have.

Some concrete ways that you can get involved:

- **Spread the word** when you use Phasik by sharing with your colleagues and friends.
- **Request a new feature or report a bug** by raising a new issue.
- **Create a Pull Request (PR)** to address an open issue or add a feature.

CHAPTER

FIVE

HOW TO CITE

We acknowledge the importance of good software to support research, and we note that research becomes more valuable when it is communicated effectively. To demonstrate the value of Phasik, we ask that you cite Phasik in your work. Currently, the best way to cite Phasik is to go cite the paper where it was first used:

"Inferring cell cycle phases from a partially temporal network of protein interactions"
Lucas, M., Morris, A., Townsend-Teague, A., Tichit, L., Habermann, B. H., & Barrat, A.
Cell Reports Methods, 3(2), 2023

In addition to the package, this repository contains the notebooks necessary to reproduce the analysis of the paper. Version of the library associated to the paper:

CHAPTER

SIX

DEVELOPERS

- Maxime Lucas (lead)
- Arthur Morris
- Matteo Neri
- Simone Poetto
- Laurent Tichit
- Alex Townsend-Teague

CHAPTER

SEVEN

CONTACT

maxime.lucas.work[at]gmail[dot]com

CHAPTER

EIGHT

LICENSE

Released under the GNU GENERAL PUBLIC LICENSE v3 (see LICENSE)

CHAPTER

NINE

OTHER RESOURCES

This library may not meet your needs and if this is this case, consider checking out these other resources:

- Raphtory: A package, written in Rust, with a Python interface, for representing, analyzing, and visualizing temporal and static graphs and hypergraphs.
- Reticula: A package with a Python wrapper of C++ functions for representing, analyzing, and visualizing temporal and static graphs and hypergraphs.
- Tacoma: A package in Python for representing, analyzing, and visualizing temporal networks.
- Teneto: A package in Python for representing, analyzing, and visualizing temporal networks.

CHAPTER

TEN

TEMPORAL NETWORKS

Phasik works with *temporal networks*, sometimes called *dynamic networks* or *time-varying networks*. Temporal networks are networks with time-varying edges. They are a natural representation of systems that contain many interacting units (nodes), whose interactions change over time. Temporal networks are often used to model social contact networks, or brain networks, for example.

Temporal networks are a generalisation of the (static) networks in which there is no time dimension: they are composed of nodes and (constant) edges. In temporal networks, however, time is resolved and edges can vary in time. Hence, whereas static networks can be described by adjacency matrix A_{ij}, the adjacency matrix to describe temporal networks needs to be time-varying $A_{ij}(t)$. Phasik implements temporal networks as a dedicated class `TemporalNetwork`.

Phasik provides functions to build a TemporalNetwork by combining a static network to temporal data in several forms:

- time series of nodes: e.g., time series of protein concentrations
- time series of edges: e.g., time series relative to protein-protein interactions
- a list of *t-edges*, sometimes called timestamped interactions, of the form (i,j,t,weight), where i and j are two nodes
- a list of adjacency matrices (or snapshots), representating the time-evolution of the adjacency matrix

A particularity of Phasik is that it can build *partially* temporal networks, i.e. for which we have temporal information about only part of the edges. This is useful in many situations when using experimental data from systems in which the recording of all variables is not always possible. Phasik has a dedicated class `PartiallyTemporalNetwork` for this too.

Phasik also provides several functions to visualise temporal networks, either as static images or as animations.

For more details, see Reference and *Tutorial*.

Further reading: Holme, P., & Saramäki, J. (2012). Temporal networks. Phys. Rep., 519(3), 97-125.

CHAPTER

ELEVEN

INFERRING THE MULTISCALE TEMPORAL ORGANISATION OF TEMPORAL NETWORKS

Systems often go through various phases, or states, over time. A good example of this is the cell cycle, which is typically divided into 4 main phases and multiple subphases. This multiscale temporal organisation in phases of a system can yield insight into its functioning but also its fate. That is why inferring and predicting theses phases can further our understanding of these systems.

The function and dynamics of a network is often linked to its structure, or topology. Hence, if snapshots of the temporal network at 2 different times are very similar, they can be grouped into a same *phase*. On the contrary, if two snapshots are very different, the temporal network can be said in two different phases. This idea can be formalised by clustering snapshots of a given temporal network. Since snapshots have a 1:1 correspondence to timepoints, clusters of snapshots have a can be related to time intervals. These clusters can then be linked back to those edges most active at those times to interpret them from a microscopic standpoint.

Further reading:

- Masuda, N., & Holme, P. (2019). Detecting sequences of system states in temporal networks. Sci. Rep., 9(1) , 1-11.
- Lucas, M., Morris, A., Townsend-Teague, A., Tichit, L., Habermann, B. H., & Barrat, A. (2023). Inferring cell cycle phases from a partially temporal network of protein interactions. Cell Reports Methods, 3(2).

CHAPTER

TWELVE

WORKFLOW

The Phasik workflow is based on the following classes: first, build a `TemporalNetwork` from which to build a `DistanceMatrix`, from which to then build a `ClusterSet` or `ClusterSets`.

Get started:

- Build temporal networks
- Infer phases

To go further, check out the Jupyter Notebooks available on Gitlab.They can be downloaded and run locally.

CHAPTER

THIRTEEN

BUILDING A TEMPORAL NETWORK

Phasik allows the user to build a temporal network from timeseries data in one of the following forms:

- **node time series**, e.g. protein concentration over time, as a pandas Dataframe. Indices must be node names (strings or integers), and columns must be times.
- **edge time series**, e.g. protein-protein interactions over time, as a pandas DataFrame. Indices must be edge names (e.g. 'A-B' for nodes 'A' and 'B'), and columns must be times.
- **t-edges**, also called *timestamped interactions* of the form (i,j,t,weight), as a pandas DataFrame. Columns must be 'i', 'j', 't' and optionally 'weight'. Each row represents one t-edge.

The user can choose to build a fully connected temporal network from this data, with the following methods:

TemporalNetwork.from_tedges(tedges[, normalise])	Creates a TemporalNetwork from a dataframe of tedges
TemporalNetwork.from_edge_timeseries(...[, ...])	Creates a TemporalNetwork from a DataFrame of edge timeseries
TemporalNetwork.from_node_timeseries(...[, ...])	Creates a temporal network by combining node timeseries into edge timeseries.

Or to combine the time series data with a static network, with the following methods:

TemporalNetwork.from_static_network_and_tedges(...)	Creates a temporal network by combining a static network with tedges
TemporalNetwork.from_static_network_and_edge_timeseries(...)	Creates a temporal network by combining a static network with edge timeseries
TemporalNetwork.from_static_network_and_node_timeseries(...)	Creates a temporal network by combining a static network with node timeseries

Note: The set of nodes/edges present in the time series do **not** need to be the same as those present in the static network. In the resulting temporal network, the nodes and edges are those of the static network, and the methods only add temporal information for the edges among those present in the time series (as an edge, or as two nodes).

Note: In Phasik, timepoints do not need to be evenly spaced in a temporal network. This opens the possibility to use timeseries associated to a pseudotime for example.

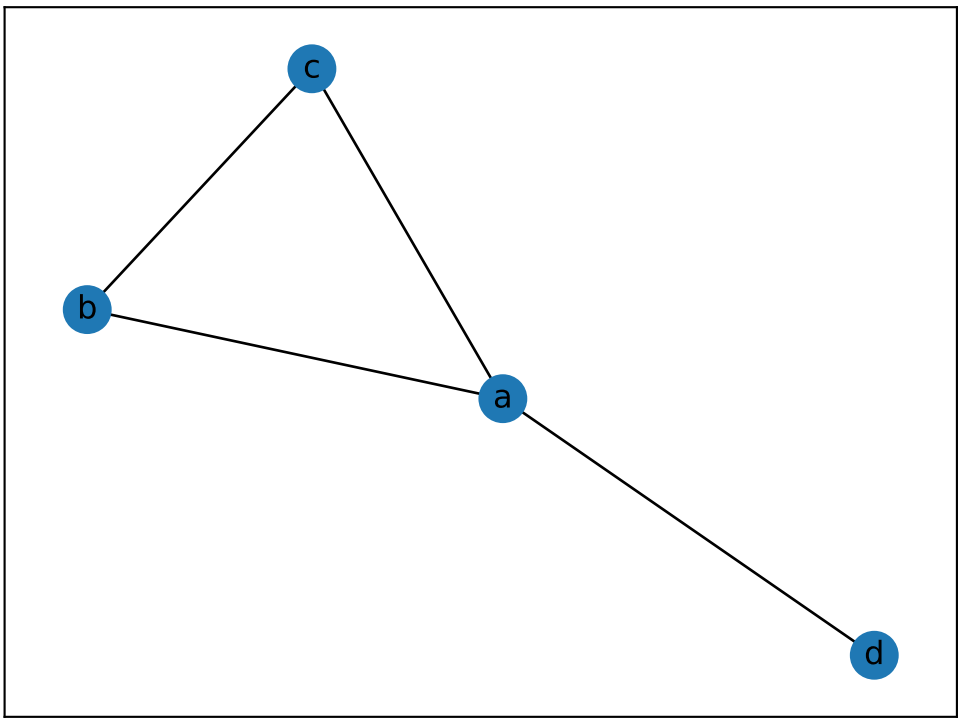
13.1 Example from static network and node time series

Before anything, start by importing Phasik and a few base packages. Numpy is only used to generate random data for the example, and Maplotlib to illustrate it.

```
>>> import matplotlib.pyplot as plt
>>> import networkx as nx # for the static network
>>> import numpy as np
>>> import pandas as pd # for the temporal data
>>> # import Phasik
>>> import phasik as pk
```

First, we generate an example static network

```
>>> edges = [('a', 'b'), ('a', 'c'), ('b', 'c'), ('a', 'd')]
>>> static_network = nx.Graph(edges)
>>>
>>> nx.draw_networkx(static_network)
>>> plt.show()
```



Note: The nodes can be either strings or integers.

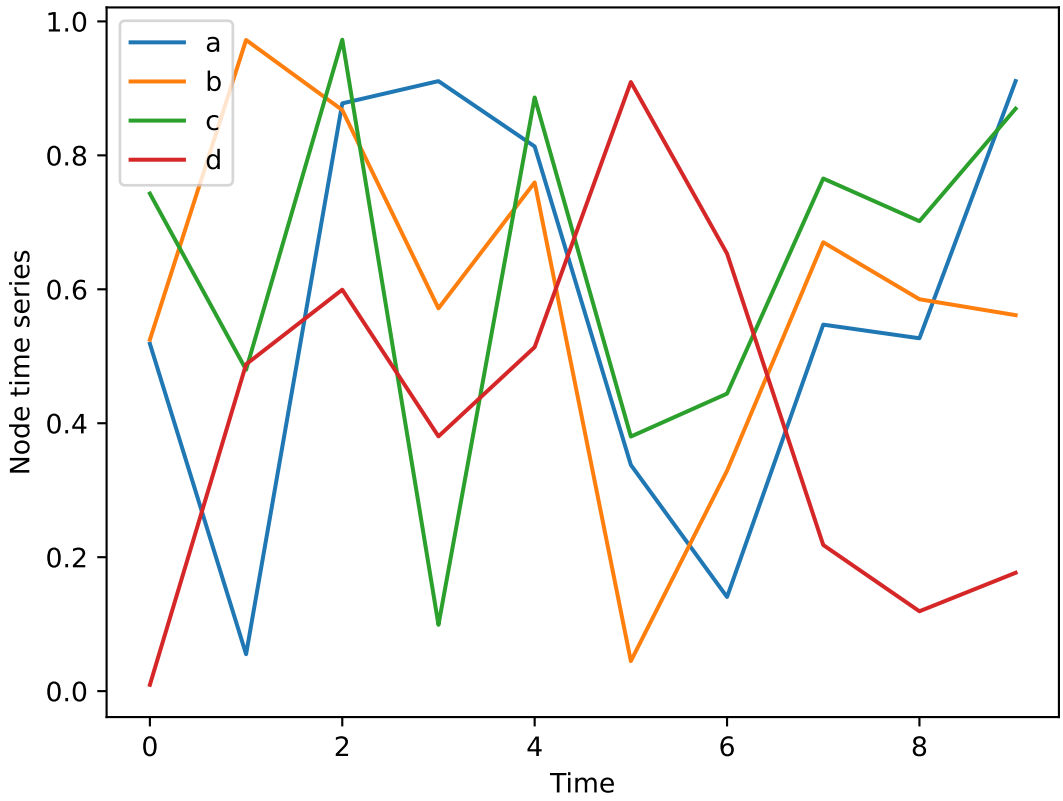
Note: The static network can be created any other method provided by NetworkX. It can be created from a file containing interactions (edgelist) or and adjacency file, for example.

Second, we generate example time series for the nodes

```
>>> nodes = list(static_network.nodes)
>>> N = static_network.number_of_nodes()
>>> T = 10 # number of timepoints
>>>
>>> node_series_arr = np.random.random((N, T)) # random time series
>>> node_series = pd.DataFrame(node_series_arr, index=nodes)
>>> node_series
```

The node time series must be in DataFrame as above. The indexes represent node names, and the columns are times. Each value in the DataFrame is that relative to a given node at a given time. For example, the 1st row could could represent the concentration of protein 'a' from time 0 to time 9. The time series can be visualised as follows:

```
>>> node_series.T.plot()
>>> plt.xlabel("Time")
>>> plt.ylabel("Node time series")
>>> plt.show()
```



Finally, we generate the temporal network by combining the static network and the node time series. Several options

are available to choose the normalisation of the resulting edge time series and how to deal with edges without temporal information.

```
>>> # create the temporal network by combining
>>> # the static network with the node timeseries
>>> temporal_network = pk.TemporalNetwork.from_static_network_and_node_timeseries(
...     static_network,
...     node_series,
...     static_edge_default_weight=1,
...     normalise='minmax', # method to normalise the edge weights
...     quiet=False # if True, prints less information
... )
```

```
>>> print(temporal_network)
<class 'phasik.classes.TemporalNetwork.TemporalNetwork'> with 4 nodes and 10 times
```

13.2 To go further

More examples of how to build a temporal network with Phasik can be found on the gitlab repository.

CHAPTER

FOURTEEN

INFERRING PHASES

Examples of how to infer phases with Phasik can be found on the gitlab repository, for example notebook 1_c_Infer_phases_by_clustering_snapshots.

```
>>> import matplotlib.pyplot as plt
>>> import networkx as nx # for the static network
>>> import numpy as np
>>> import pandas as pd # for the temporal data
>>> # import Phasik
>>> import phasik as pk
```

14.1 Building the temporal network

First, we generate an example static network

```
>>> edges = [('a', 'b'), ('a', 'c'), ('b', 'c'), ('a', 'd')]
>>> static_network = nx.Graph(edges)
```

Second, we generate example time series for the nodes

```
>>> nodes = list(static_network.nodes)
>>> N = static_network.number_of_nodes()
>>> T = 10 # number of timepoints
>>>
>>> node_series_arr = np.random.random((N, T)) # random time series
>>> node_series = pd.DataFrame(node_series_arr, index=nodes)
```

Finally, we generate the temporal network

```
>>> # create the temporal network by combining
>>> # the static network with the node timeseries
>>> temporal_network = pk.TemporalNetwork.from_static_network_and_node_timeseries(
...     static_network,
...     node_series,
...     static_edge_default_weight=1,
...     normalise='minmax', # method to normalise the edge weights
...     quiet=False # if True, prints less information
... )
```

14.2 Inferring phases

Set the parameters for the phase inference

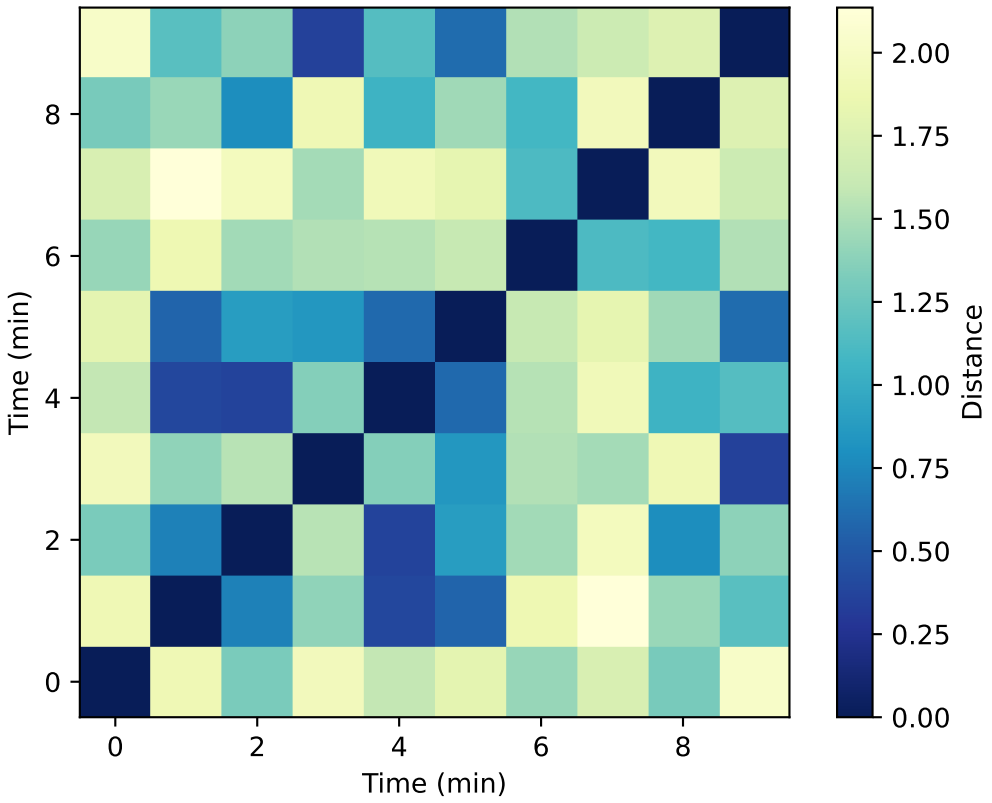
```
>>> distance_metric = 'euclidean' # used to compute distance between snapshots
>>> clustering_method = 'ward' # used to compute the distance between clusters
>>> n_max_type = 'maxclust' # set number of clusters by maximum number of clusters wanted
>>> n_max = 3 # max number of clusters
>>> n_max_range = range(2,6) # range of numbers of clusters to compute
```

First, compute the distance matrix between snapshots, from the temporal network:

```
>>> distance_matrix = pk.DistanceMatrix.from_temporal_network(temporal_network, distance_
↪metric)
```

Plot this distance matrix :

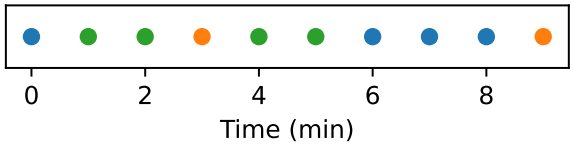
```
>>> fig, ax = plt.subplots()
>>>
>>> im = ax.imshow(distance_matrix.distance_matrix, aspect="equal", origin="lower", cmap=
↪"YlGnBu_r")
>>>
>>> ax.set_ylabel("Time (min)")
>>> ax.set_xlabel("Time (min)")
>>>
>>> cb = fig.colorbar(im)#, cax=cax)
>>> cb.set_label("Distance")
>>>
>>> plt.show()
```



Second, compute a cluster set with a given number of clusters 'n_max':

```
>>> cluster_set = pk.ClusterSet.from_distance_matrix(distance_matrix, n_max_type, n_max,␣
↪clustering_method)
```

```
>>> fig, ax = plt.subplots(figsize=(7, 1))
>>>
>>> cluster_set.plot(ax=ax, y_height=0)
>>>
>>> ax.set_aspect(10)
>>> ax.set_yticks([])
>>> ax.set_xlabel("Time (min)")
>>> plt.tight_layout()
>>> plt.show()
```

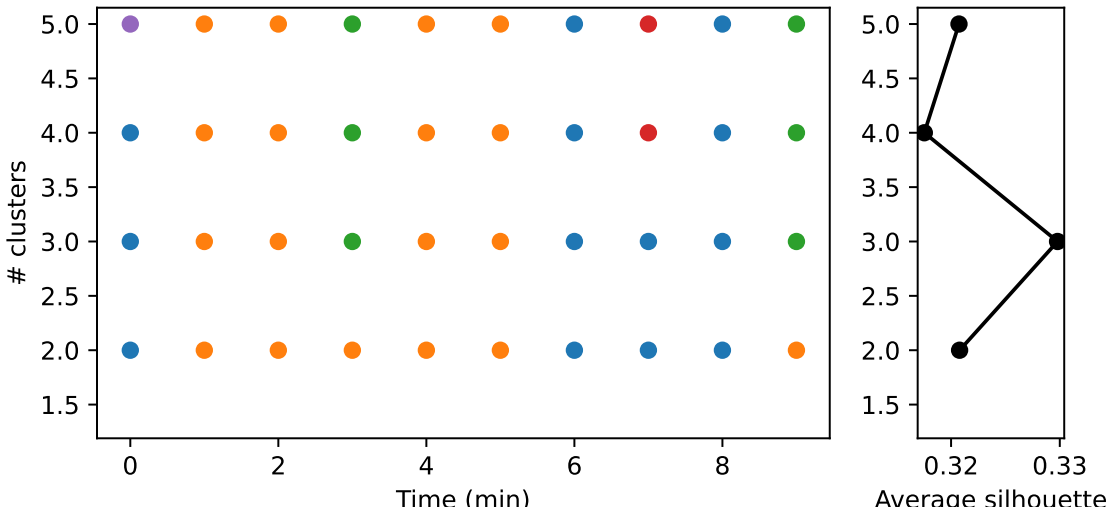


We can also compute a range of numbers of clusters

```
>>> cluster_sets = pk.ClusterSets.from_distance_matrix(distance_matrix, n_max_type, n_
↪max_range, clustering_method)
```

and plot them as follows, with the associated silhouette scores:

```
>>> gridspec_kw = {"width_ratios": [5, 1]}
>>> fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(7, 3.5), gridspec_kw=gridspec_kw,␣
↪sharey='all')
>>>
>>> cluster_sets.plot(axs=(ax1, ax2), with_silhouettes=True)
>>> pk.adjust_margin(ax=ax1, bottom=0.2)
>>>
>>> ax1.set_xlabel("Time (min)")
>>> ax1.set_axisbelow(True)
>>> ax1.set_ylabel("# clusters")
>>>
>>> ax2.set_xlabel("Average silhouette")
>>> ax2.yaxis.set_tick_params(labelleft=True)
>>>
>>> plt.subplots_adjust(wspace=0.2, top=0.8)
>>> plt.show()
```



CHAPTER

FIFTEEN

CLASSES

15.1 Temporal Network

class TemporalNetwork

Base class for temporal networks

Temporal networks are networks with time-varying edges. They consist of nodes and edges, and latter can have time-varying weights.

Variables

- **nodes** (*list of (str or int)*) – Sorted list of node names. Node names can be either strings or integers, but they all need to be of the same type.
- **times** (*list of (int or float)*) – Sorted list of times for which we have temporal information
- **tedges** (*pandas.DataFrame*) – Dataframe containing tedges, also called timestamped data (potentially weighted). Columns are ['i', 'j', 't', ('weight')] and each row represents a tedge.
- **snapshots** (*numpy array*) – Array of shape (T, N, N) storing the instantaneous values of the adjacency matrix A_{ij}(t).

15.1.1 Constructors

TemporalNetwork()	Base class for temporal networks
TemporalNetwork.from_tedges(tedges[, normalise])	Creates a TemporalNetwork from a dataframe of tedges
TemporalNetwork.from_edge_timeseries(...[, ...])	Creates a TemporalNetwork from a DataFrame of edge timeseries
TemporalNetwork.from_node_timeseries(...[, ...])	Creates a temporal network by combining node timeseries into edge timeseries.
TemporalNetwork.from_static_network_and_tedges(...)	Creates a temporal network by combining a static network with tedges
TemporalNetwork.from_static_network_and_edge_timeseries(...)	Creates a temporal network by combining a static network with edge timeseries
TemporalNetwork.from_static_network_and_node_timeseries(...)	Creates a temporal network by combining a static network with node timeseries

phasik.classes.TemporalNetwork.TemporalNetwork

class TemporalNetwork

Base class for temporal networks

Temporal networks are networks with time-varying edges. They consist of nodes and edges, and latter can have time-varying weights.

Variables

- **nodes** (*list of (str or int)*) – Sorted list of node names. Node names can be either strings or integers, but they all need to be of the same type.
- **times** (*list of (int or float)*) – Sorted list of times for which we have temporal information
- **tedges** (*pandas.DataFrame*) – Dataframe containing tedges, also called timestamped data (potentially weighted). Columns are ['i', 'j', 't', ('weight')] and each row represents a tedge.
- **snapshots** (*numpy array*) – Array of shape (T, N, N) storing the instantaneous values of the adjacency matrix A_{ij}(t).

__init__()

Methods

N()	Returns the number of nodes
T()	Returns the number of times
__init__()	
add_tedges(tedges_to_add)	Adds multiple tedges (optionally weighted)
aggregated_network([time_indices, output])	Returns a time-aggregated network as a networkx.Graph
discard_temporal_info_from_edge(edge[, ...])	Discards temporal information from 'edge' by setting its weight to a constant
discard_temporal_info_from_node(node[, ...])	Discards temporal information from 'node' by setting the weight of its edges to a constant
edge_timeseries([edges])	Returns dict of edge time series.
edges_aggregated()	Returns a list of edges in the aggregated network
from_edge_timeseries(edge_timeseries[, ...])	Creates a TemporalNetwork from a DataFrame of edge timeseries
from_node_timeseries(node_timeseries[, ...])	Creates a temporal network by combining node timeseries into edge timeseries.
from_static_network_and_edge_timeseries(...)	Creates a temporal network by combining a static network with edge timeseries
from_static_network_and_node_timeseries(...)	Creates a temporal network by combining a static network with node timeseries
from_static_network_and_tedges(...[, ...])	Creates a temporal network by combining a static network with tedges
from_tedges(tedges[, normalise])	Creates a TemporalNetwork from a dataframe of tedges
has_node(node)	Returns True if node is in the TemporalNetwork
has_tedge(tedge)	Returns True if tedge is in the TemporalNetwork, regardless of its weight
has_time(time)	Returns True if time is in the TemporalNetwork
is_weighted()	Returns True if tedges are weighted
neighbors()	Returns a dictionary of neighboring nodes in the aggregate network.
network_at_time(time_index[, output])	Returns the temporal network at time 'time' as a networkx.Graph
number_of_edges()	Returns the number of edges in the aggregated network
shape()	Returns the shape (N,T) of the TemporalNetwork
tedges_of_edge(edge[, return_mask, reverse])	Returns a filtered DataFrame containing only the tedges of edge 'edge'.
tedges_of_node(node[, return_mask, reverse])	Returns a filtered DataFrame containing only the tedges of node 'node'.
to_partially_temporal()	Returns a copy of the temporal network as a PartiallyTemporalNetwork

Attributes

nodes	Returns a list of nodes in the TemporalNetwork
snapshots	Returns a numpy array of snapshots in the Temporal-Network
tedges	Returns a DataFrame of tedges the TemporalNetwork
times	Returns a list of times in the TemporalNetwork

phasik.classes.TemporalNetwork.TemporalNetwork.from_tedges

classmethod TemporalNetwork.**from_tedges**(*tedges*, *normalise=None*)

Creates a TemporalNetwork from a dataframe of tedges

Parameters

- **tedges** (*pandas.DataFrame or list of tuples*) – List of tedges with 'i', 'j', 't', and optionally 'weight' If DataFrame, these are the name of the columns, and each row contains a tedge
- **normalise** (*{'max', 'minmax', "standardise", None}*) – Choice of normalsation of the edge timeseries

Returns TN

Return type *TemporalNetwork*

phasik.classes.TemporalNetwork.TemporalNetwork.from_edge_timeseries

classmethod TemporalNetwork.**from_edge_timeseries**(*edge_timeseries*, *normalise='max'*)

Creates a TemporalNetwork from a DataFrame of edge timeseries

All edges in the network are those of the timeseries, and nodes are extracted from edge names

Parameters

- **edge_timeseries** (*pandas.DataFrame*) – Dataframe where each row is a timeseries, with index as edge names and columns as times
- **normalise** (*{'max', 'minmax', "standardise", None}*) – Choice of normalsation of the edge timeseries

Return type *TemporalNetwork*

phasik.classes.TemporalNetwork.TemporalNetwork.from_node_timeseries

classmethod TemporalNetwork.**from_node_timeseries**(*node_timeseries*, *normalise='max'*)

Creates a temporal network by combining node timeseries into edge timeseries.

By construction, the underlying static network created is always fully connected.

Parameters

- **node_timeseries** (*pandas.DataFrame*) – Timeseries of nodes, indexed by node name and times as columns
- **normalise** (*{'max', 'minmax', "standardise", None}*) – Choice of normalsation of the edge timeseries

Return type *TemporalNetwork*

phasik.classes.TemporalNetwork.TemporalNetwork.from_static_network_and_tedges

classmethod TemporalNetwork.**from_static_network_and_tedges**(*static_network*, *tedges*, *static_edge_default_weight=None*, *normalise='max'*, *quiet=True*)

Creates a temporal network by combining a static network with tedges

If all edges of the static network are represented in the tedges, create a temporal network by setting time-varying edge weights from the tedges. Raises an Exception if not all edges have temporal information.

Parameters

- **static_network** (*networkx.Graph*) – Static network into which to integrate the temporal information
- **tedges** (*pandas.DataFrame or list of tuples*) – Tedges must be of the form (i, j, t, weight)
- **static_edge_default_weight** (*float*) – Weight to use for edges without temporal information
- **normalise** (*{'max', 'minmax', "standardise", None}*) – Choice of normalsation of the edge timeseries
- **quiet** (*bool*) – If True (default), print minimum informative messages

phasik.classes.TemporalNetwork.TemporalNetwork.from_static_network_and_edge_timeseries

classmethod TemporalNetwork.**from_static_network_and_edge_timeseries**(*static_network*, *edge_timeseries*, *static_edge_default_weight=None*, *normalise=None*, *quiet=False*)

Creates a temporal network by combining a static network with edge timeseries

If all edges of the static network are represented in the timeseries, create a temporal network by setting time-varying edge weights from the tedges. If not all edges of the static network, creates a partially temporal network.

Parameters

- **static_network** (*nx.Graph*) – Static network into which to integrate the temporal information
- **edge_timeseries** (*Dataframe*) – Dataframe with indexed (rows) by edge names (formatted as 'A-B') and with columns as times. Entries of the Dataframe represent the weight of that edge at that time.
- **static_edge_default_weight** (*float*) – Weight to use for edges without temporal information
- **normalise** (*{'max', 'minmax', "standardise", None}*) – Choice of normalsation of the edge timeseries
- **quiet** (*bool*) – If True (default), print minimum informative messages

Return type *TemporalNetwork*

phasik.classes.TemporalNetwork.TemporalNetwork.from_static_network_and_node_timeseries

classmethod TemporalNetwork.**from_static_network_and_node_timeseries**(*static_network*, *node_timeseries*, *combine_node_weights=<function TemporalNetwork.<lambda>>*, *static_edge_default_weight=None*, *normalise=None*, *quiet=False*)

Creates a temporal network by combining a static network with node timeseries

Edge time series are generated for the subset of edges in the 'static_network' that have both nodes in the 'node_timeseries', by combining their time series. These edge times series are used to set the time-varying weights of the corresponding edges in the temporal network. If not all edges have temporal information, creates a partially temporal network.

Parameters

- **static_network** (*nx.Graph*) – Static network into which to integrate the temporal information
- **node_timeseries** (*Dataframe*) – Dataframe with indexed (rows) by node names and with columns as times. Entries of the Dataframe represent the value of that node at that time.
- **combine_node_weights** (*function*) – Function that determines how two node timeseries are combined to generate and edge timeseries. By default, the two node timeseries are multiplied.
- **static_edge_default_weight** (*float*) – Weight to use for edges without temporal information
- **normalise** (*{'max', 'minmax', "standardise", None}*) – Choice of normalsation of the edge timeseries
- **quiet** (*bool*) – If True (default), print minimum informative messages

Return type *TemporalNetwork*

15.1.2 Utils

TemporalNetwork.N()	Returns the number of nodes
TemporalNetwork.T()	Returns the number of times
TemporalNetwork.shape()	Returns the shape (N,T) of the TemporalNetwork
TemporalNetwork.number_of_edges()	Returns the number of edges in the aggregated network
TemporalNetwork.is_weighted()	Returns True if tedges are weighted
TemporalNetwork.has_node(node)	Returns True if node is in the TemporalNetwork
TemporalNetwork.has_time(time)	Returns True if time is in the TemporalNetwork
TemporalNetwork.has_tedge(tedge)	Returns True if tedge is in the TemporalNetwork, regardless of its weight

phasik.classes.TemporalNetwork.TemporalNetwork.N

TemporalNetwork.**N**()
Returns the number of nodes

phasik.classes.TemporalNetwork.TemporalNetwork.T

TemporalNetwork.**T**()
Returns the number of times

phasik.classes.TemporalNetwork.TemporalNetwork.shape

TemporalNetwork.**shape**()
Returns the shape (N,T) of the TemporalNetwork

phasik.classes.TemporalNetwork.TemporalNetwork.number_of_edges

TemporalNetwork.**number_of_edges**()
Returns the number of edges in the aggregated network

phasik.classes.TemporalNetwork.TemporalNetwork.is_weighted

TemporalNetwork.**is_weighted**()
Returns True if tedges are weighted

phasik.classes.TemporalNetwork.TemporalNetwork.has_node

TemporalNetwork.**has_node**(*node*)
Returns True if node is in the TemporalNetwork

phasik.classes.TemporalNetwork.TemporalNetwork.has_time

TemporalNetwork.**has_time**(*time*)
Returns True if time is in the TemporalNetwork

phasik.classes.TemporalNetwork.TemporalNetwork.has_tedge

TemporalNetwork.**has_tedge**(*tedge*)
Returns True if tedge is in the TemporalNetwork, regardless of its weight

15.1.3 Methods for nodes and edges

TemporalNetwork.add_tedges(tedges_to_add)	Adds multiple tedges (optionally weighted)
TemporalNetwork.neighbors()	Returns a dictionary of neighboring nodes in the aggregate network.
TemporalNetwork.edge_timeseries([edges])	Returns dict of edge time series.
TemporalNetwork.tedges_of_edge(edge[, ...])	Returns a filtered DataFrame containing only the tedges of edge 'edge'.
TemporalNetwork.tedges_of_node(node[, ...])	Returns a filtered DataFrame containing only the tedges of node 'node'.
TemporalNetwork.edges_aggregated()	Returns a list of edges in the aggregated network

phasik.classes.TemporalNetwork.TemporalNetwork.add_tedges

TemporalNetwork.**add_tedges**(*tedges_to_add*)

Adds multiple tedges (optionally weighted)

Parameters **tedges_to_add** (*DataFrame or list of tuples*)

Return type None

phasik.classes.TemporalNetwork.TemporalNetwork.neighbors

TemporalNetwork.**neighbors**()

Returns a dictionary of neighboring nodes in the aggregate network.

Returns A dictionary where keys represent the nodes in the aggregate network, and values are lists of neighboring nodes.

Return type dict

Notes

This function relies on the *aggregated_network* method to obtain the aggregate network graph.

phasik.classes.TemporalNetwork.TemporalNetwork.edge_timeseries

TemporalNetwork.**edge_timeseries**(*edges=None*)

Returns dict of edge time series. Keys are edge names and values are timeseries

Parameters **edges** (*list of tuples*) – List of edges wanted, e.g. [('A, B')]. If None (default), all edges in the temporal network are used.

Returns **all_series** – Dictionary with edge names as keys (as 'A-B'), and timeseries as values.

Return type dict

phasik.classes.TemporalNetwork.TemporalNetwork.tedges_of_edge

TemporalNetwork.**tedges_of_edge**(*edge*, *return_mask=True*, *reverse=False*)

Returns a filtered DataFrame containing only the tedges of edge 'edge'.

Optionally, return the boolean mask to filter the original DataFrame.

Parameters

- **edge** (*tuple*) – Edge used to filter tedges
- **return_mask** (*bool, optional*) – If True (default), return boolean mask to filter the original DataFrame
- **reverse** (*bool, optional*) – If True, return the Dataframe obtained by filtered with logically opposite mask, i.e. all tedges except those of edge 'edge'.

Return type None

phasik.classes.TemporalNetwork.TemporalNetwork.tedges_of_node

TemporalNetwork.**tedges_of_node**(*node*, *return_mask=True*, *reverse=False*)

Returns a filtered DataFrame containing only the tedges of node 'node'.

Optionally, return the boolean mask to filter the original DataFrame.

Parameters

- **node** (*str or int*) – Node used to filter tedges
- **return_mask** (*bool, optional*) – If True (default), return boolean mask to filter the original DataFrame
- **reverse** (*bool, optional*) – If True, return the Dataframe obtained by filtered with logically opposite mask, i.e. all tedges except those of node 'node'.

phasik.classes.TemporalNetwork.TemporalNetwork.edges_aggregated

TemporalNetwork.**edges_aggregated**()

Returns a list of edges in the aggregated network

Parameters **None**

Return type list of tuples

15.1.4 Methods

TemporalNetwork.aggregated_network([...])	Returns a time-aggregated network as a networkx.Graph
TemporalNetwork.to_partially_temporal()	Returns a copy of the temporal network as a PartiallyTemporalNetwork
TemporalNetwork.discard_temporal_info_from_edges()	Discards temporal information from 'edge' by setting its weight to a constant
TemporalNetwork.discard_temporal_info_from_nodes()	Discards temporal information from 'node' by setting the weight of its edges to a constant

phasik.classes.TemporalNetwork.TemporalNetwork.aggregated_network

TemporalNetwork.**aggregated_network**(*time_indices=None*, *output='weighted'*)

Returns a time-aggregated network as a networkx.Graph

Parameters

- **time_indices** (*list of int, optional*) – Indices of times over which to aggregate the network (default: all times).
- **output** (*{'weighted', 'averaged', 'binary', 'normalised'}, optional*) – Determines the type of output edge weights

Returns **G_agg** – Aggregated network

Return type networkx Graph

phasik.classes.TemporalNetwork.TemporalNetwork.to_partially_temporal

TemporalNetwork.**to_partially_temporal**()

Returns a copy of the temporal network as a PartiallyTemporalNetwork

phasik.classes.TemporalNetwork.TemporalNetwork.discard_temporal_info_from_edge

TemporalNetwork.**discard_temporal_info_from_edge**(*edge*, *default_weight=1*, *reverse=False*)

Discards temporal information from 'edge' by setting its weight to a constant

Returns a copy of the temporal network with the new edge weights

Parameters

- **edge** (*tuple of int or str*) – Edge from which to discard temporal information
- **default_weight** (*float, optional*) – Value used for the edges with no temporal information
- **reverse** (*bool, optional*) – If True, discard temporal info from all edges except 'edge'.

Returns **TN_modified**

Return type *TemporalNetwork*

phasik.classes.TemporalNetwork.TemporalNetwork.discard_temporal_info_from_node

TemporalNetwork.**discard_temporal_info_from_node**(*node*, *default_weight=1*, *reverse=False*)

Discards temporal information from 'node' by setting the weight of its edges to a constant

Returns a copy of the temporal network with the new edge weights

Parameters

- **node** (*int or str*) – Node from which to discard temporal information
- **default_weight** (*float, optional*) – Value used for the edges with no temporal information
- **reverse** (*bool, optional*) – If True, discard temporal info from all nodes except 'node'.

Returns **TN_modified**

Return type *TemporalNetwork*

15.2 Partially Temporal Network

A PartiallyTemporalNetwork is a TemporalNetwork for which we do not have temporal information about a subset of the edges. In some sense, it is in between a static and a temporal network.

class PartiallyTemporalNetwork

Base class for partially temporal networks

Partially temporal networks are temporal networks for which we do not have temporal information about all edges.

Variables

- **nodes** (*list of (str or int)*) – Sorted list of node names. Node names can be either strings or integers, but they all need to be of the same type.
- **times** (*list of (int or float)*) – Sorted list of times for which we have temporal information
- **tedges** (*pandas.DataFrame*) – Dataframe containing tedges (potentially weighted). Columns are ['i', 'j', 't', ('weight')] and each row represents a tedge.
- **snapshots** (*numpy array*) – Array of shape (T, N, N) storing the instantaneous values of the adjacency matrix A_{ij}(t).
- **temporal_nodes** (*list of (str or int)*) – List of nodes that are part of a temporal edge
- **temporal_edges** (*list of tuples*) – List of edges for which we have temporal information

15.2.1 Specific methods

It is implemented as a child class of the TemporalNetwork class, and contains the additional attributes *temporal_nodes* and *temporal_edges* and the following methods:

PartiallyTemporalNetwork.number_of_temporal_edges()	Returns the number of temporal edges in the temporal network
PartiallyTemporalNetwork.number_of_temporal_nodes()	Returns the number of temporal nodes in the temporal network
PartiallyTemporalNetwork.fraction_of_temporal_nodes()	Returns the fraction of temporal edges in the temporal network
PartiallyTemporalNetwork.fraction_of_temporal_edges()	Returns the fraction of temporal edges in the temporal network
PartiallyTemporalNetwork.temporal_neighbors()	Returns a dict of neighbors in the aggregated network that are temporal nodes

phasik.classes.PartiallyTemporalNetwork.PartiallyTemporalNetwork.number_of_temporal_edges

PartiallyTemporalNetwork.**number_of_temporal_edges**()

Returns the number of temporal edges in the temporal network

phasik.classes.PartiallyTemporalNetwork.PartiallyTemporalNetwork.number_of_temporal_nodes

PartiallyTemporalNetwork.**number_of_temporal_nodes**()

Returns the number of temporal nodes in the temporal network

phasik.classes.PartiallyTemporalNetwork.PartiallyTemporalNetwork.fraction_of_temporal_nodes

PartiallyTemporalNetwork.**fraction_of_temporal_nodes**()

Returns the fraction of temporal edges in the temporal network

phasik.classes.PartiallyTemporalNetwork.PartiallyTemporalNetwork.fraction_of_temporal_edges

PartiallyTemporalNetwork.**fraction_of_temporal_edges**()

Returns the fraction of temporal edges in the temporal network

phasik.classes.PartiallyTemporalNetwork.PartiallyTemporalNetwork.temporal_neighbors

PartiallyTemporalNetwork.**temporal_neighbors**()

Returns a dict of neighbors in the aggregated network that are temporal nodes

15.3 TemporalData

class TemporalData(*temporal_data*, *variables*, *times*, *true_times*)

Class representing any sort of temporal data for a specified list of variables

Since teneto's TemporalNetwork class requires all times to start at zero, we again have the concept of 'true' times as well as times offset to start at zero.

Parameters

- **temporal_data** (*ndarray*) – 2D array whose columns are the variables we're interested in and whose rows are the temporal data for these variables
- **variables** (*list of str*) – List (not a numpy array) of variable names
- **times** (*ndarray*) – 1D array based on true_times but with values shifted to start at zero
- **true_times ndarray** – 1D array of time points

downsample(*skip*)

Return a downsampled copy of the TemporalData object, by skipping ever 'skip' elements

classmethod from_df(*df*, *times*, *true_times*, *i_tend=None*)

Create a TemporalData from a DataFrame with variables as columns and timepoints as rows

classmethod from_dict(*dict_*, *times*, *true_times*, *i_tend=None*)

Create a TemporalData from a dict with variables as keys and time series as values

normalise()

Return new TemporalData where each time series is normalised by its maximum

plot_relative_optima(*variable*, *optima_type*, *ax=None*, *use_true_times=True*, *plot_var=True*)

Plot relative optima for a prticular variable

Parameters

- **variable** (*str*) – The name of the variable whose optima we want to plot
- **optima_type** (*{'minima', 'maxima'}*)
- **ax** (*matplotlib.Axes, optional*) – Axes on which to plot
- **use_true_times** (*bool, optional*) – Whether to use the 'actual' times or offset the times so that they start at zero

Return type None

plot_series(*variables=None*, *ax=None*, *norm=False*, *add_labels=True*, *labels_xvals=None*, *use_true_times=True*)

Plot particular variables' values over time

Parameters

- **variables** (*list of str*) – Names of the variable whose values we want to plot
- **ax** (*matplotlib.Axes, optional*) – Axes on which to plot
- **norm** (*bool, optional*) – Whether or not to normalise the time series by dividing through by the max (default False)
- **add_labels** (*bool, optional*) – Whether to label the variables when plotting (default True)
- **labels_xvals** (*list of float, optional*) – The positions along the x-axis at which to place the variables' labels (if using). If set to None (default), labels will be placed at regular intervals along x-axis.
- **use_true_times** (*bool, optional*) – Whether to use the 'actual' times or offset the times so that they start at zero (default True)

Return type None

relative_optima(*variable*, *optima_type*)

Get relative optima for a particular variable

Parameters

- **variable** (*str*) – Name of the variable whose optima we want
- **optima_type** (*{'minima', 'maxima'}*) – Type of optima to look for

Returns

- **optima** (*list_like*) – Value of the variable at its optima
- **optima_times** (*list_like*) – Times at which these optima occur

save_to_csv(*filepath*, *index=False*)

Save temporal data to .csv file, with variables as rows and time points as columns

`series`(*variable*)

Get temporal data for a particular variable

`to_df`()

Convert temporal data to pandas.DataFrame format, with variables as rows and time points as columns

`to_dict`()

Convert temporal data to dict format, with variables as keys and time series as values

15.4 DistanceMatrix

`class DistanceMatrix`(*snapshots*, *times*, *distance_metric*)

Base class for matrix of pairwise distance/similarity between snapshots of a temporal network.

Variables

- **times** (*list of (int or float)*) – Times corresponding to each of the T snapshots
- **snapshots** (*numpy array*) – Array of dim (T, N, N) representing instantaneous adjacency matrices. Snapshots can also be inputed as vectors of dim (T, N).
- **snapshots_flat** (*numpy array*) – Snapshots (flattened into vectors if originals are matrices) from which the distance matrix is computed
- **distance_metric** (*str*) – Distance metric used to compute the distance between snapshots, e.g. 'euclidean' with sklearn.metrics.pairwise.paired_distances. It must be one of the options allowed by scipy.spatial.distance.pdist for its metric parameter (e.g. 'chebyshev', 'cityblock', 'correlation', 'cosine', 'euclidean', 'hamming', 'jaccard', etc.), or a metric listed in pairwise.PAIRWISE_DISTANCE_FUNCTIONS.
- **distance_matrix** (*numpy array*) – Array of dim (T, T)
- **distance_matrix_flat** (*numpy array*) – Flattened distance matrix of dim (T,)

Base class for a distance matrix, i.e. a matrix where each entry is the distance/similarity between two snapshots in 'snapshots'.

Parameters

- **snapshots** (*numpy array*) – Array of dim (T, N, N) representing instantaneous adjacency matrices. Snapshots can also be inputed as vectors of dim (T, N).
- **times** (*list of (float or int)*) – Times corresponding to each snapshot
- **distance_metric** (*str*) – Distance metric used to compute the distance between snapshots, e.g. 'euclidean', with sklearn.metrics.pairwise.paired_distances. It must be one of the options allowed by scipy.spatial.distance.pdist for its metric parameter (e.g. 'chebyshev', 'cityblock', 'correlation', 'cosine', 'euclidean', 'hamming', 'jaccard', etc.), or a metric listed in pairwise.PAIRWISE_DISTANCE_FUNCTIONS.

`property distance_matrix`

Returns the distance matrix as a numpy array

`property distance_matrix_flat`

Returns the distance matrix flattened for easier use in clustering

`property distance_metric`

Returns the distance metric used to compute the distance matrix

classmethod from_temporal_network(*temporal_network*, *distance_metric*)

Generates a distance matrix from a temporal network

Each entry of the matrix is the distance between two snapshots of the temporal network.

Parameters

- **temporal_network** (*TemporalNetwork*) – Temporal network from which to compute the distance matrix
- **distance_metric** (*str*) – Distance metric used to compute the distance between snapshots, e.g. 'euclidean', with sklearn.metrics.pairwise.paired_distances. It must be one of the options allowed by scipy.spatial.distance.pdist for its metric parameter (e.g. 'chebyshev', 'cityblock', 'correlation', 'cosine', 'euclidean', 'hamming', 'jaccard', etc.), or a metric listed in pairwise.PAIRWISE_DISTANCE_FUNCTIONS.

Return type *DistanceMatrix*

classmethod from_timeseries(*timeseries*, *distance_metric*)

Generates a distance matrix from time series

Each entry of the matrix is the distance between two 'snapshots' of the timeseries, i.e. the vector with instantaneous values of the N timeseries at time t.

Parameters

- **timeseries** (*pandas.Dataframe*) – Timeseries relative to nodes, edges, or both. Each row is a timeseries, with index as series name and columns as times.
- **distance_metric** (*str*) – Distance metric used to compute the distance between snapshots, e.g. 'euclidean', with sklearn.metrics.pairwise.paired_distances. It must be one of the options allowed by scipy.spatial.distance.pdist for its metric parameter (e.g. 'chebyshev', 'cityblock', 'correlation', 'cosine', 'euclidean', 'hamming', 'jaccard', etc.), or a metric listed in pairwise.PAIRWISE_DISTANCE_FUNCTIONS.

Return type *DistanceMatrix*

property snapshots

Returns the snapshots (matrix or vectors) from which the distance matrix is computed

property snapshots_flat

Returns the snapshots (flattened into vectors if original are matrices) from which the distance matrix is computed

property times

Returns the sorted list of times corresponding to the snapshots

15.5 ClusterSet

class ClusterSet(*clusters*, *times*, *linkage*, *distance_matrix*, *distance_metric*, *cluster_method*, *n_clusters_max*, *n_max_type*)

Base class for a set of clusters (partition) of timepoints

Variables

- **clusters** (*list of int*) – Clusters as a list of cluster labels
- **times** (*list of (int or float)*) – Sorted list of time associated to each clustered snapshot

- **n_clusters** (*int*) – Number of clusters in the cluster set (partition)
- **cluster_method** (*float*) – Method used to cluster the snapshots . Examples : 'k_means', 'centroid', 'average', 'complete', 'weighted', 'median', 'single', 'ward'
- **n_max_type** (*float*) – Method that was used to determine when to stop clustering when creating this cluster set. e.g. A cluster set can be created by clustering until a particular number of clusters has been reached ('maxclust'), or until every cluster is at least a certain distance away from each other ('distance').
- **n_max** (*int*) – Value corresponding to the n_max_type described above.
- **distance_metric** (*str*) – Distance metric used to compute the distance between snapshots, e.g. 'euclidean', with sklearn.metrics.pairwise.paired_distances. It must be one of the options allowed by scipy.spatial.distance.pdist for its metric parameter (e.g. 'chebyshev', 'cityblock', 'correlation', 'cosine', 'euclidean', 'hamming', 'jaccard', etc.), or a metric listed in pairwise.PAIRWISE_DISTANCE_FUNCTIONS.

Parameters

- **clusters** (*list of int*) – Clusters as a list of cluster labels
- **times** (*list of (int or float)*) – Sorted list of time associated to each clustered snapshot
- **linkage** – Linkage of the clustering
- **distance_matrix** (*phasik.DistanceMatrix*) – Distance matrix from which the clusters were computed
- **cluster_method** (*float*) – Method used to cluster the snapshots . Examples : k_means', 'centroid', 'average', 'complete', 'weighted', 'median', 'single', 'ward'
- **n_max_type** (*float*) – Method that was used to determine when to stop clustering when creating this cluster set. e.g. A cluster set can be created by clustering until a particular number of clusters has been reached ('maxclust'), or until every cluster is at least a certain distance away from each other ('distance').
- **n_clusters_max** (*int*) – Value corresponding to the n_max_type described above.
- **distance_metric** (*str*) – Distance metric used to compute the distance between snapshots, e.g. 'euclidean', with sklearn.metrics.pairwise.paired_distances. It must be one of the options allowed by scipy.spatial.distance.pdist for its metric parameter (e.g. 'chebyshev', 'cityblock', 'correlation', 'cosine', 'euclidean', 'hamming', 'jaccard', etc.), or a metric listed in pairwise.PAIRWISE_DISTANCE_FUNCTIONS.

`property cluster_method`

Returns the clustering method used to cluster the temporal data

`property clusters`

Returns the clusters, i.e. a list of cluster labels (int)

`property distance_metric`

Returns the distance metric used to compute the distance between snapshots, e.g. 'euclidean'

`distance_threshold()`

Calculate the distance at which clustering stops

Parameters None

Returns Smallest number d such that the distance between any two clusters is < d.

Return type int

classmethod from_distance_matrix(*distance_matrix*, *n_max_type*, *n_clusters_max*, *cluster_method*)

Generates a ClusterSet from a distance matrix

Parameters

- **distance_matrix** (*phasik.DistanceMatrix*) – Distance matrix from which to cluster
- **cluster_method** (*str*) – Clustering method used to cluster the temporal network snapshots. Examples : 'k_means', 'centroid', 'average', 'complete', 'weighted', 'median', 'single', 'ward'
- **n_max_type** (*str*) – The method that determines when to stop clustering. For example, cluster set can be created by clustering until a particular number of clusters has been reached ('maxclust'), or until every cluster is at least a certain distance away from each other ('distance').
- **n_clusters_max** (*int*) – Value corresponding to the n_max_type described above.

Return type *ClusterSet*

classmethod from_temporal_network(*temporal_network*, *distance_metric*, *cluster_method*, *n_max_type*, *n_clusters_max*)

Generates a ClusterSet from a temporal network

Parameters

- **temporal_network** (*TemporalNetwork*) – Temporal network from which to compute the distance matrix
- **distance_metric** (*str*) – Distance metric used to compute the distance between snapshots, e.g. 'euclidean', with sklearn.metrics.pairwise.paired_distances. It must be one of the options allowed by scipy.spatial.distance.pdist for its metric parameter (e.g. 'chebyshev', 'cityblock', 'correlation', 'cosine', 'euclidean', 'hamming', 'jaccard', etc.), or a metric listed in pairwise.PAIRWISE_DISTANCE_FUNCTIONS.
- **clustering_method** (*str*) – Clustering method used to cluster the temporal network snapshots. Examples : 'k_means', 'centroid', 'average', 'complete', 'weighted', 'median', 'single', 'ward'
- **n_max_type** (*str*) – The method that determines when to stop clustering. For example, cluster set can be created by clustering until a particular number of clusters has been reached ('maxclust'), or until every cluster is at least a certain distance away from each other ('distance').
- **n_clusters_max** (*int*) – Value corresponding to the n_max_type described above.

Return type *ClusterSet*

property n_max

Returns the value corresponding to the n_max_type described above.

property n_max_type

Returns the method (str) that determines when to stop clustering

plot(*ax=None*, *y_height=0*, *cmap=<matplotlib.colors.ListedColormap object>*, *number_of_colors=10*, *colors=None*)

Plots this cluster set as a scatter graph

Parameters

- **ax** (*matplotlib.Axes, optional*) – Axes on which to plot
- **y_height** (*int or float, optional*) – Height at which to plot (default 0)

- **cmap** (*matplotlib.cm, optional*) – Desired colour map (default 'tab10')
- **number_of_colors** (*int, optional*) – Desired number of colours to use for the colormap (default 10)
- **colors** (*list of int, optional*) – If None (default), cluster label 0 is assigned its automatic color "C0" and so on. If *colors* is a list (e.g. [3,1,2]), it relabels the clusters in that order and assigns them the new corresponding colors.

Return type None

`plot_dendrogram`(*ax=None*, *distance_threshold=None*, *leaf_rotation=90*, *leaf_font_size=6*)

Plot this cluster set as a dendrogram

Parameters

- **ax** (*matplotlib.Axes, optional*) – Axes on which to plot
- **leaf_rotation** (*int or float, optional*) – Rotation to apply to the x-axis (leaf) labels (default 90)
- **leaf_font_size** (*int or str, optional*) – Desired size of the x-axis (leaf) labels (default 6)

Return type None

`plot_silhouette_samples`(*ax=None*)

Plot the silhouette samples from this cluster set

Parameters **ax** (*matplotlib.Axes, optional*) – Axes on which to plot

Return type None

`property times`

Returns the list of times corresponding to datapoints clustered

15.6 ClusterSets

`class ClusterSets`(*cluster_sets*, *n_max_type*, *ns_max*)

Base class for sets of clusters (partition) of timepoints

Variables

- **cluster_sets** (*iterable of phasik.ClusterSet*) – List of ClusterSets
- **clusters** (*numpy array of int*) – Summary array of the cluster labels, with dim (len(ns_max), len(times))
- **n_clusters** (*list of int*) – Number of clusters in the cluster set (partition)
- **times** (*list of (int or float)*) – Sorted list of time associated to each clustered snapshot
- **distance_metric** (*str*) – Distance metric used to compute the distance between snapshots, e.g. 'euclidean', with sklearn.metrics.pairwise.paired_distances. It must be one of the options allowed by scipy.spatial.distance.pdist for its metric parameter (e.g. 'chebyshev', 'cityblock', 'correlation', 'cosine', 'euclidean', 'hamming', 'jaccard', etc.), or a metric listed in pairwise.PAIRWISE_DISTANCE_FUNCTIONS.
- **n_max_type** (*float*) – Method that was used to determine when to stop clustering when creating this cluster set. e.g. A cluster set can be created by clustering until a particular

number of clusters has been reached ('maxclust'), or until every cluster is at least a certain distance away from each other ('distance').

- **ns_max** (*list of int*) – List of values corresponding to the n_max_type described above, in other words, list of numbers clusters to be computed. The number of elements in this list is the number of ClusterSet computed.
- **silhouettes_average** (*numpy array*) – Value of average silouette for each clustering

Parameters

- **cluster_sets** (*iterable of ClusterSet*)
- **n_max_type** (*str*) – Method that was used to determine when to stop clustering when creating these cluster sets. e.g. A cluster set can be created by clustering until a particular number of clusters has been reached ('maxclust'), or until every cluster is at least a certain distance away from each other ('distance')
- **ns_max** (*list of int*) – List of values corresponding to the n_max_type described above, in other words, list of numbers clusters to be computed. The number of elements in this list is the number of ClusterSet computed.

property clusters_sets

Returns the list of ClusterSet

classmethod from_distance_matrix(*distance_matrix*, *n_max_type*, *ns_clusters_max*, *cluster_method*)

Generates ClusterSets from a distance matrix

Parameters

- **distance_matrix** (*phasik.DistanceMatrix*) – Distance matrix from which to cluster
- **cluster_method** (*str*) – Clustering method used to cluster the temporal network snapshots. Examples : 'k_means', 'centroid', 'average', 'complete', 'weighted', 'median', 'single', 'ward'
- **n_max_type** (*str*) – The method that determines when to stop clustering. For example, cluster set can be created by clustering until a particular number of clusters has been reached ('maxclust'), or until every cluster is at least a certain distance away from each other ('distance').
- **ns_clusters_max** (*list of int*) – List of values corresponding to the n_max_type described above, in other words, list of numbers clusters to be computed. The number of elements in this list is the number of ClusterSet computed.

Return type *ClusterSets*

plot(*axs=None*, *coloring='consistent'*, *translation=None*, *with_silhouettes=False*, *with_n_clusters=False*)

Plots these cluster sets as a scatter graph

Parameters

- **ax** (*matplotlib.Axes, optional*) – Axes on which to plot
- **coloring** (*{'ascending', 'consistent', None}*)
- **translation** (*dict, optional*)
- **If None (default), has no effect. Elsee, dictionary that determines which label**
- **should be replaced by which other label**
- **For example {1** (*2, 2: 3, 3: 1}*)
- **It is applied after the order relabling from `method`.**

- **with_silhouettes** (*bool*) – If True, also plot the average silhouettes on a 2nd axis. Defaults to False.
- **with_n_clusters** (*bool*) – If True, also plot the actual number of clusters on a 3rd axis. Defaults to False.

Return type None

`plot_and_format_with_average_silhouettes`(*axs*, *events*, *phases*, *time_ticks=None*, *coloring='consistent'*)

Plot and format these cluster sets as a scatter graph, along with the average silhouettes and cluster set sizes

Our pattern generally has been to leave all formatting in the jupyter notebooks, but this method is used by several different notebooks, so it makes sense to put it somewhere common.

Parameters

- **axs** (*list of matplotlib.Axes*) – Axes on which to plot; should be an indexable object with at least three items
- **events** – Any events that should be plotted on the scatter graph
- **phases** – Any phases that should be plotted on the scatter graph
- **time_ticks** (*list or array*) – The ticks that should be displayed along the x-axis (time axis)

Return type None

`plot_silhouette_samples`(*axs*, *coloring='consistent'*)

Plot the average silhouettes across this range of cluster sets

Parameters **axs** (*list of matplotlib.Axes*) – Axes on which to plot; should be an iterable object with at least as many items as there are cluster sets in this class.

Return type None

CHAPTER

SIXTEEN

DRAWING

16.1 Clusters

Useful functions to draw clusters

plot_average_silhouettes(*cluster_sets*, *ax=None*)

Plot the average silhouettes across this range of cluster sets

Parameters

- **cluster_sets** (*ClusterSets*)
- **ax** (*matplotlib.Axes, optional*) – Axes on which to plot

Return type None

plot_cluster_set(*cluster_set*, *ax=None*, *y_height=0*, *cmap=<matplotlib.colors.ListedColormap object>*, *number_of_colors=10*, *colors=None*)

Plots this cluster set as a scatter graph

Parameters

- **cluster_set** (*pk.ClusterSet*) – ClusterSet object containing partitions to plot
- **ax** (*matplotlib.Axes, optional*) – Axes on which to plot
- **y_height** (*int or float, optional*) – Height at which to plot (default 0)
- **cmap** (*matplotlib.cm, optional*) – Desired colour map (default 'tab10')
- **number_of_colors** (*int, optional*) – Desired number of colours to use for the colormap (default 10)
- **colors** (*list of int, optional*) – If None (default), cluster label 0 is assigned its automatic color "C0" and so on. If *colors* is a list (e.g. [3,1,2]), it relabels the clusters in that order and assigns them the new corresponding colors.

Return type None

plot_cluster_sets(*cluster_sets*, *axs=None*, *coloring='consistent'*, *translation=None*, *with_silhouettes=False*, *with_n_clusters=False*)

Plot these cluster sets as a scatter graph

Parameters

- **cluster_sets** (*phasik ClusterSets*) – ClusterSets object containing partitions to plot
- **axs** (*matplotlib.Axes, optional*) – Matplotlib axes on which to plot
- **coloring** (*{'ascending', 'consistent', None}, optional*)

- **translation** (*dict, optional*) – If None (default), has no effect. Elsee, dictionary that determines which label should be replaced by which other label For example {1: 2, 2: 3, 3: 1} It is applied after the order relabling from *method*.
- **with_silhouettes** (*bool, optional*)
- **with_n_clusters** (*bool, optional*)

Return type None

`plot_dendrogram`(*cluster_set*, *ax=None*, *distance_threshold=None*, *leaf_rotation=90*, *leaf_font_size=6*)

Plot this cluster set as a dendrogram

Parameters

- **ax** (*matplotlib.Axes, optional*) – Axes on which to plot
- **leaf_rotation** (*int or float, optional*) – Rotation to apply to the x-axis (leaf) labels (default 90)
- **leaf_font_size** (*int or str, optional*) – Desired size of the x-axis (leaf) labels (default 6)

Return type None

`plot_ns_clusters`(*cluster_sets*, *ax=None*)

Plot the average cluster set sizes across this range of cluster sets

Parameters

- **cluster_sets** (*ClusterSets*) – Cluster sets information to plot
- **ax** (*matplotlib.Axes, optional*) – Axes on which to plot

Return type None

`plot_randindex_bars_over_methods_and_sizes`(*valid_cluster_sets*, *reference_method='ward'*, *ax=None*, *plot_ref=False*)

Plot Rand Index as bars, to compare any method to a reference method.

This compares all combinations of methods and number of clusters.

Parameters

- **valid_cluster_sets** (*list*) – A list of tuples representing valid cluster sets. Each tuple contains the ClusterSet and the clustering method name.
- **reference_method** (*str, optional*) – The reference method to compare other methods to. Defaults to "ward".
- **ax** (*matplotlib.axes.Axes, optional*) – The axes to plot the bars on. If not provided, the current axes will be used.
- **plot_ref** (*bool, optional*) – Determines whether to plot the reference method bars (will be one). Defaults to False.

Returns The axes object containing the plotted bars.

Return type matplotlib.axes.Axes

`relabel_clusters_sorted`(*clusters*, *final_labels=None*)

Returns array of cluster labels sorted in order of appearance, with clusters unchanged

Parameters

- **clusters** (*array of int*) – Cluster labels

- **final_labels** (*array of int*) – Cluster labels in expected order (has size of the number of clusters)

Example

```
>>> clusters = np.array([2, 2, 2, 3, 3, 1, 1, 1])
>>> relabel_clusters_sorted(clusters)
[ 1 1 1 2 2 3 3 3 ]
```

relabel_clustersets(*cluster_sets*, *method='consistent'*, *translation=None*)

Relabels clusters in each cluster set, so that clusters are labeled consistently at different numbers of clusters.

This is especially useful when plotting cluster sets, to have consistent colouring.

Parameters

- **cluster_sets** (*ClusterSets*)
- **method** (*{'consistent', 'ascending'}, optional*)
- **translation** (*dict, optional*) – If None (default), has no effect. Elsee, dictionary that determines which label should be replaced by which other label For example {1: 2, 2: 3, 3: 1} It is applied after the order relabling from *method*.

Returns **cluster_sets_sorted**

Return type *ClusterSets*

relabel_clustersets_from_dict(*cluster_sets*, *translation*)

Relabels clusters in each cluster set, so that clusters are labeled according to the translation dictionary

This is especially useful when plotting cluster sets, to have consistent colouring. between different figures with cluster sets.

Parameters

- **cluster_sets** (*ClusterSets*)
- **translation** (*dict*) – Dictionary that determines which label should be replaced by which other label For example {1: 2, 2: 3, 3: 1}

Returns **cluster_sets_sorted**

Return type *ClusterSets*

relabel_next_clusterset_sorted(*cluster_sets*, *cluster_sets_sorted*, *i*)

Relabels the clusters in i+1th cluster set so that it is consistent with ith cluster set

This is especially useful when plotting cluster sets, to have consistent colouring.

Parameters

- **cluster_sets** (*ClusterSets*) – Original cluster sets
- **cluster_sets_sorted** (*ClusterSets*) – Cluster sets being sorted, already sorted up to i-1
- **i** (*int*)

Returns **cluster_sets_sorted**

Return type *ClusterSets*

16.2 Networks

Functions to visualise networks and temporal networks

animate_temporal_network(*temporal_network*, *color_temporal='red'*, *color_constant='silver'*, *width_scale=1.5*, *with_labels=True*, *pos=None*, *ax=None*, *interval=20*, *frames=None*)

Return animation of the temporal network evolving over time

Parameters

- **temporal_network** (*phasik.TemporalNetwork*) – Temporal network to visualise
- **color_temporal** (*str*) – Color of the time-varying edges, defaults to 'red'
- **color_constant** (*str*) – Color of the constant edges (defaults to 'silver'), i.e. for which we have no temporal information
- **width_scale** (*float*) – Scale factor for width of the temporal edges compared to the constant ones
- **pos** (*dict*) – Dictionary of node positions
- **ax** (*matplotlib.axis*) – Axes to plot the animation on
- **interval** (*int*) – Interval of time between frames, in ms.
- **frames** (*int*) – Number of frames of the animation (should be at most the number of time-points (default))

Return type matplotlib.animation

draw_graph(*graph*, *ax=None*, *label_nodes=True*, *color='mediumseagreen'*, *pos=None*, *edge_widths=None*, *edge_colors=None*, *edge_cmap=None*, *edge_vmin=None*, *edge_vmax=None*)

Basic graph drawing function

Parameters

- **graph** (*networkx.Graph*) – Graph to visualise
- **ax** (*matplotlib.Axes, optional*) – Axes on which to draw the graph
- **label_nodes** (*bool, optional*) – Whether to label the nodes or just leave them as small circles (default True)
- **color** (*str, optional*) – Color to use for the graph nodes and edges (default 'mediumseagreen')
- **pos** (*dict*) – Dictionary of node positions
- **edge_widths** (*float or array of floats*) – Line width of edges
- **edge_colors** (*color or array of colors*) – Edge color. Can be a single color or a sequence of colors with the same length as edgelist. Color can be string or rgb (or rgba) tuple of floats from 0-1. If numeric values are specified they will be mapped to colors using the edge_cmap and edge_vmin,edge_vmax parameters.
- **edge_cmap** (*Matplotlib colormap, optional*) – Colormap for mapping intensities of edges
- **edge_vmin,edge_vmax** (*floats, optional*) – Minimum and maximum for edge colormap scaling

Return type None

`highlight_subgraphs`(*graphs*, *colors*, *ax=None*, *pos=None*, *label_nodes=True*)

Draw multiple nested subgraphs on the same axes

Parameters

- **graphs** (*list of networkx.Graph*)
- **colors** (*list of str*) – List of colors, one for each of the graphs in 'graphs'
- **ax** (*matplotlib.Axes, optional*) – Axes to plot on
- **label_nodes** (*bool, optional*) – Whether or not to label the graph nodes or leave them as circles
- **pos** (*dict*) – Dictionary of node positions

Return type None

`standard_edge_params`(*color*)

Returns a dictionary containing standard values of edge plotting parameters

`standard_label_params`(*color*)

Returns a dictionary containing standard values of label plotting parameters

`standard_node_params`(*color*)

Returns a dictionary containing standard values of node plotting parameters

`standard_params`(*color*)

Returns a dictionary containing standard values of plotting parameters

16.3 Drawing

Useful drawing functions

`plot_edge_series`(*temporal_network*, *edges*, *ax=None*)

Plot timeseries of edge weights, for the specified edges

Parameters

- **temporal_network** (*pk.TempNet*) – Temporal network
- **edge** (*list of str*) – List of edges to plot
- **ax** (*matplotlib.Axes, optional*) – Axes to use

Return type ax

`plot_events`(*events*, *ax=None*, *text_y_pos=None*, *text_x_offset=0*, *period=None*, *n_periods=1*, *add_labels=True*, *orientation='vertical'*, *zorder=- 1*, *alpha=1*, *va='bottom'*)

Plot events as vertical lines on axes

Parameters

- **events** (*list of tuples (time, name, line_style)*) – time - time at which the event occurred name - the name of the event line_style - any string accepted by matplotlib.lines.Line2D.set_linestyle
- **ax** (*matplotlib.Axes, optional*) – Axes on which to plot the events
- **text_y_pos** (*float, optional*) – Height at which to place the name of the event (default None)
- **text_x_offset** (*float, optional*) – Distance along x-axis by which to offset the placement of the event name (default 0)

- **period** (*float or None, optional*) – Length of time of one period, if events repeat periodically.
- **n_periods** (*int, optional*) – Number of periods to draw, if events repeat periodically.
- **add_labels** (*bool, optional*) – Wether to display the label of each vertical line, True by default.

Return type None

`plot_interval`(*binary_series*, *times*, *y=0*, *peak=None*, *color='k'*, *ax=None*, *zorder=0*)

Plot a binary series as a sequence of coloured intervals

Specifically, when a binary series has value 1, plot it as a continuous rectangular interval. When it has value 0 do nothing.

Parameters

- **binary_series** (*ndarray*) – 2D array of binary data to plot
- **times** (*ndarray*) – 1D array consisting of the corresponding time points
- **y** (*float, optional*) – Height (y-axis value) at which to plot the interval (default 0)
- **color** (*str, optional*) – Color to use for the intervals (default 'k')
- **ax** (*matplotlib.Axes, optional*) – Axes to plot on
- **zorder** (*int, optional*) – Height on the z-axis which to plot the interval (default 0)

Return type None

`plot_phases`(*phases*, *ax=None*, *y_pos=None*, *ymin=0*, *ymax=1*, *t_offset=0*)

Plot phases as shaded regions on axes

Parameters

- **phases** (*list of tuples (start_time, end_time, name)*)
- **ax** (*matplotlib.Axes*) – Axes on which to plot the phases
- **y_pos** (*float or None, optional*) – Height at which to place the name of the phase
- **ymin** (*float, optional*) – Height at which to start shaded region (default 0)
- **ymax** (*float, optional*) – Height at which to stop shaded region (default 1)
- **t_offset** (*float, optional*) – Offset of phase on the time axis

Return type None

`threshold_plot`(*x*, *y*, *threshold*, *color_below_threshold*, *color_above_threshold*, *ax=None*)

Plot values above a certain threshold in a particular colour

Parameters

- **x** (*array*) – 1D array of alues to plot along x-axis
- **y** (*array*) – 1D array of values to plot along y-axis
- **threshold** (*float*) – Only plot in colour the points (x,y) with y >= threshold
- **colour_below_threshold** (*str*) – Colour to use for points below threshold
- **colour_above_threshold** (*list of str*) – Colour to use for points above threshold
- **ax** (*matplotlib.Axes, optional*) – Axes to use

Return type None

16.4 Utils

General utility functions for plots

adjust_margin(*ax=None*, *top=0*, *bottom=0*, *left=0*, *right=0*)

Extend the margin of a plot by a percentage of its original width/height

Parameters

- **ax** (*matplotlib.Axes, optional*) – Axes whose margins to adjust
- **top** (*float, optional*) – Percentage (as decimal) by which to increase top margin
- **bottom** (*float, optional*) – Percentage (as decimal) by which to increase bottom margin
- **left** (*float, optional*) – Percentage (as decimal) by which to increase left margin
- **right** (*float, optional*) – Percentage (as decimal) by which to increase right margin

Return type None

configure_sch_color_map(*cmap*)

Set SciPy's colour palette to use a particular colour map

display_name(*key*)

Get a more user-friendly name for certain keywords.

This function takes a keyword *key* and returns a more user-friendly name for that keyword if it exists in the predefined *names* dictionary. If the keyword is not found in the dictionary, it is returned as is.

Parameters **key** (*str*) – The keyword for which a display name is needed.

Returns The display name for the given keyword, or the original keyword if not found.

Return type str

Examples

```
>>> display_name('maxclust')
'Max # clusters'
>>> display_name('distance')
'Distance threshold'
>>> display_name('unknown')
'unknown'
```

label_subplot_grid_with_shared_axes(*rows*, *columns*, *total_subplots*, *xlabel*, *ylabel*, *fig*, *axes*)

Method to tidy up cases where we have a grid of plots with shared axes, by deleting unused subplots (if number of of subplots is not rectangular) and adding axes ticks

Parameters

- **rows** (*int*) – Number of rows in the grid of subplots
- **columns** (*int*) – Number of columns in the grid of subplots
- **total_subplots** (*int*) – Number of subplots in the grid; need not be a 'rectangular' number
- **xlabel** (*str*) – Label of the x-axis
- **ylabel** (*str*) – Label of the y-axis
- **fig** (*matplotlib.Figure*) – Figure that the subplots are a part of

- **axes** (*list of matplotlib.Axes*) – Axes containing the subplots

Return type None

`palette_20_ordered`(*as_map=False*)

Create an ordered color palette of 20 colors.

The function uses the 'tab20' color palette from seaborn and rearranges the colors in an ordered pattern. By default, the colors are returned as a list, but if *as_map* is set to True, a *ListedColormap* object is returned.

Parameters **as_map** (*bool, optional*) – Whether to return the colors as a *ListedColormap* object (default is False).

Returns The ordered color palette. If *as_map* is True, a *ListedColormap* object is returned.

Return type list or ListedColormap

CHAPTER

SEVENTEEN

UTILS

17.1 Clusters

Functions to manipulate and sort clusters

`aggregate_network_by_cluster`(*temporal_network*, *clusters*, *sort_clusters=None*, *output='averaged'*)

Aggregates the temporal network over eacher cluster in a cluster set

Parameters

- **temporal_network** (*phasik.TemporalNetwork*) – Temporal network to aggregate
- **clusters** (*array of int*) – Cluster labels
- **sort_clusters** (*bool*) – If True, sort cluster labels based on ascending times
- **output** (*{'weighted', 'averaged', 'binary', 'normalised'}, optional*) – Determines the type of output edge weights

Returns **aggregates** – Dict each key is a cluster label and each value is a tuple of the form (networkx.Graph, list of time indices of cluster).

Return type dict

`cluster_sort`(*clusters*, *final_labels=None*)

Sorts an array of cluster labels in order of appearance, and returns the sorted array while leaving the original clusters unchanged.

Parameters

- **clusters** (*numpy.ndarray*) – An array of cluster labels.
- **final_labels** (*list or None, optional*) – A list of final labels (as integers) to replace the original cluster labels, by default None.

Returns An array of cluster labels sorted in order of appearance. If *final_labels* is not None, it will return a list of final labels with the same length as *clusters*.

Return type numpy.ndarray or list

Examples

```
>>> clusters = np.array([2, 2, 2, 3, 3, 1, 1, 1])
>>> cluster_sort(clusters)
array([1, 1, 1, 2, 2, 3, 3, 3])
```

```
>>> final_labels = [4, 5, 6]
>>> cluster_sort(clusters, final_labels)
[4, 4, 4, 5, 5, 6, 6, 6]
```

convert_cluster_labels_to_dict(*clusters*)

Returns dictionary where each key is a cluster label and each value is list of the time indices composing the cluster

Parameters **clusters** (*list of int*) – List of cluster labels

rand_index_over_methods_and_sizes(*valid_cluster_sets*, *reference_method='ward'*)

Compute the Rand Index to compare any clustering method to a reference method, for all combinations of methods and number of clusters.

Parameters

- **valid_cluster_sets** (*list*) – List of tuples (cluster_object, method_name) representing the clustering object and the name of the clustering method used to obtain it.
- **reference_method** (*str, optional*) – The name of the reference method to compare against. The default is "ward".

Returns **rand_scores** – Array of dimension (n_sizes, n_methods) with Rand Index scores.

Return type ndarray

Notes

The Rand Index is a measure of the similarity between two clusterings. It is based on the number of pairs of samples that are assigned to the same or different clusters in the two clusterings. The adjusted Rand Index is a modification of the Rand Index that takes into account chance agreements.

17.2 Graphs

Utility functions for static graphs

graph_size_info(*graph*)

Return basic size info on about graph

weighted_edges_as_df(*network*, *keep_static=True*, *temporal_edges=None*)

Returns a pandas.Dataframe of weighted edges sorted by weight, from a networkx.Graph.

Columns are ['i', 'j', 'weight'] and each row represents a different edge

Parameters

- **network** (*networkx.Graph*) – A network from which to get weighted edges
- **keep_static** (*bool or np.nan, optional*) – If True (default), keep all edges. If False, discard the static edges (those not in *temporal_edges*). If np.nan, keep the static edges, but set their weight to np.nan. If keep_static is not False, *temporal_edges* must be provided.

- **temporal_edges** (*list of tuples*) – List of edges for which there is temporal information.

Return type pandas.DataFrame

Notes

When the static network is derived from a temporal network, some edges might static (no temporal info) and have a default constant edge weight. That is when the arguments *keep_static* and *temporal_edges* are useful.

17.3 Paths

Functions to deal with system paths

`slugify`(*text*, *keep_characters=None*)

Turn any text into a string that can be used in a filename

Parameters

- **text** (*str*) – text to slugify
- **keep_characters** (*list of str*) – characters in this iterable will be kept in the final string. Defaults to ['_']. Any other non-alphanumeric characters will be removed.

Returns **slug**

Return type str

PYTHON MODULE INDEX

p

phasik.drawing.drawing, 59
phasik.drawing.drawing_clusters, 55
phasik.drawing.drawing_networks, 58
phasik.drawing.utils, 61
phasik.utils.clusters, 63
phasik.utils.graphs, 64
phasik.utils.paths, 65

INDEX

Symbols

__init__() (*TemporalNetwork method*), 36

A

add_tedges() (*TemporalNetwork method*), 42
adjust_margin() (*in module phasik.drawing.utils*), 61
aggregate_network_by_cluster() (*in module phasik.utils.clusters*), 63
aggregated_network() (*TemporalNetwork method*), 44
animate_temporal_network() (*in module phasik.drawing.drawing_networks*), 58

C

cluster_method (*ClusterSet property*), 50
cluster_sort() (*in module phasik.utils.clusters*), 63
clusters (*ClusterSet property*), 50
clusters_sets (*ClusterSets property*), 53
ClusterSet (*class in phasik.classes.clustering.ClusterSet*), 49
ClusterSets (*class in phasik.classes.clustering.ClusterSets*), 52
configure_sch_color_map() (*in module phasik.drawing.utils*), 61
convert_cluster_labels_to_dict() (*in module phasik.utils.clusters*), 64

D

discard_temporal_info_from_edge() (*TemporalNetwork method*), 44
discard_temporal_info_from_node() (*TemporalNetwork method*), 44
display_name() (*in module phasik.drawing.utils*), 61
distance_matrix (*DistanceMatrix property*), 48
distance_matrix_flat (*DistanceMatrix property*), 48
distance_metric (*ClusterSet property*), 50
distance_metric (*DistanceMatrix property*), 48
distance_threshold() (*ClusterSet method*), 50
DistanceMatrix (*class in phasik.classes.DistanceMatrix*), 48
downsample() (*TemporalData method*), 46
draw_graph() (*in module phasik.drawing.drawing_networks*), 58

E

edge_timeseries() (*TemporalNetwork method*), 42
edges_aggregated() (*TemporalNetwork method*), 43

F

fraction_of_temporal_edges() (*PartiallyTemporalNetwork method*), 46
fraction_of_temporal_nodes() (*PartiallyTemporalNetwork method*), 46
from_df() (*TemporalData class method*), 46
from_dict() (*TemporalData class method*), 46
from_distance_matrix() (*ClusterSet class method*), 50
from_distance_matrix() (*ClusterSets class method*), 53
from_edge_timeseries() (*TemporalNetwork class method*), 38
from_node_timeseries() (*TemporalNetwork class method*), 38
from_static_network_and_edge_timeseries() (*TemporalNetwork class method*), 39
from_static_network_and_node_timeseries() (*TemporalNetwork class method*), 40
from_static_network_and_tedges() (*TemporalNetwork class method*), 39
from_tedges() (*TemporalNetwork class method*), 38
from_temporal_network() (*ClusterSet class method*), 51
from_temporal_network() (*DistanceMatrix class method*), 48
from_timeseries() (*DistanceMatrix class method*), 49

G

graph_size_info() (*in module phasik.utils.graphs*), 64

H

has_node() (*TemporalNetwork method*), 41
has_tedge() (*TemporalNetwork method*), 41
has_time() (*TemporalNetwork method*), 41
highlight_subgraphs() (*in module phasik.drawing.drawing_networks*), 58

I

is_weighted() (*TemporalNetwork method*), 41

L

label_subplot_grid_with_shared_axes() (*in module phasik.drawing.utils*), 61

M

module
    phasik.drawing.drawing, 59
    phasik.drawing.drawing_clusters, 55
    phasik.drawing.drawing_networks, 58
    phasik.drawing.utils, 61
    phasik.utils.clusters, 63
    phasik.utils.graphs, 64
    phasik.utils.paths, 65

N

N() (*TemporalNetwork method*), 41
n_max (*ClusterSet property*), 51
n_max_type (*ClusterSet property*), 51
neighbors() (*TemporalNetwork method*), 42
normalise() (*TemporalData method*), 47
number_of_edges() (*TemporalNetwork method*), 41
number_of_temporal_edges() (*PartiallyTemporalNetwork method*), 46
number_of_temporal_nodes() (*PartiallyTemporalNetwork method*), 46

P

palette_20_ordered() (*in module phasik.drawing.utils*), 62
PartiallyTemporalNetwork (*class in phasik.classes.PartiallyTemporalNetwork*), 45
phasik.drawing.drawing
    module, 59
phasik.drawing.drawing_clusters
    module, 55
phasik.drawing.drawing_networks
    module, 58
phasik.drawing.utils
    module, 61
phasik.utils.clusters
    module, 63
phasik.utils.graphs
    module, 64
phasik.utils.paths
    module, 65
plot() (*ClusterSet method*), 51
plot() (*ClusterSets method*), 53
plot_and_format_with_average_silhouettes() (*ClusterSets method*), 54
plot_average_silhouettes() (*in module phasik.drawing.drawing_clusters*), 55
plot_cluster_set() (*in module phasik.drawing.drawing_clusters*), 55
plot_cluster_sets() (*in module phasik.drawing.drawing_clusters*), 55
plot_dendrogram() (*ClusterSet method*), 52
plot_dendrogram() (*in module phasik.drawing.drawing_clusters*), 56
plot_edge_series() (*in module phasik.drawing.drawing*), 59
plot_events() (*in module phasik.drawing.drawing*), 59
plot_interval() (*in module phasik.drawing.drawing*), 60
plot_ns_clusters() (*in module phasik.drawing.drawing_clusters*), 56
plot_phases() (*in module phasik.drawing.drawing*), 60
plot_randindex_bars_over_methods_and_sizes() (*in module phasik.drawing.drawing_clusters*), 56
plot_relative_optima() (*TemporalData method*), 47
plot_series() (*TemporalData method*), 47
plot_silhouette_samples() (*ClusterSet method*), 52
plot_silhouette_samples() (*ClusterSets method*), 54

R

rand_index_over_methods_and_sizes() (*in module phasik.utils.clusters*), 64
relabel_clusters_sorted() (*in module phasik.drawing.drawing_clusters*), 56
relabel_clustersets() (*in module phasik.drawing.drawing_clusters*), 57
relabel_clustersets_from_dict() (*in module phasik.drawing.drawing_clusters*), 57
relabel_next_clusterset_sorted() (*in module phasik.drawing.drawing_clusters*), 57
relative_optima() (*TemporalData method*), 47

S

save_to_csv() (*TemporalData method*), 47
series() (*TemporalData method*), 47
shape() (*TemporalNetwork method*), 41
slugify() (*in module phasik.utils.paths*), 65
snapshots (*DistanceMatrix property*), 49
snapshots_flat (*DistanceMatrix property*), 49
standard_edge_params() (*in module phasik.drawing.drawing_networks*), 59
standard_label_params() (*in module phasik.drawing.drawing_networks*), 59
standard_node_params() (*in module phasik.drawing.drawing_networks*), 59
standard_params() (*in module phasik.drawing.drawing_networks*), 59

T

T() (*TemporalNetwork method*), 41
tedges_of_edge() (*TemporalNetwork method*), 43
tedges_of_node() (*TemporalNetwork method*), 43
temporal_neighbors() (*PartiallyTemporalNetwork method*), 46
TemporalData (*class in phasik.classes.TemporalData*), 46
TemporalNetwork (*class in phasik.classes.TemporalNetwork*), 35, 36
threshold_plot() (*in module phasik.drawing.drawing*), 60
times (*ClusterSet property*), 52
times (*DistanceMatrix property*), 49
to_df() (*TemporalData method*), 48
to_dict() (*TemporalData method*), 48
to_partially_temporal() (*TemporalNetwork method*), 44

W

weighted_edges_as_df() (*in module phasik.utils.graphs*), 64