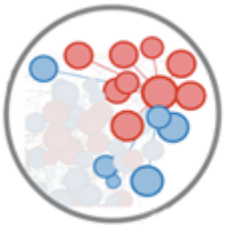

pyunicorn

pyunicorn Documentation

Release 0.6.1

Jonathan F. Donges and pyunicorn authors

Jan 20, 2023

CONTENTS

CHAPTER

ONE

INTRODUCTION

pyunicorn (**Uni**fied **Co**mplex Network and **R**ecurre**N**ce analysis toolbox) is a fully object-oriented Python package for the advanced analysis and modeling of complex networks. Above the standard measures of complex network theory such as degree, betweenness and clustering coefficient it provides some **uncommon but interesting statistics** like Newman's random walk betweenness. pyunicorn features novel **node-weighted (node splitting invariant)** network statistics as well as measures designed for analyzing **networks of interacting/interdependent networks**.

Moreover, pyunicorn allows to easily **construct networks from uni- and multivariate time series and event data** (functional (climate) networks and recurrence networks). This involves linear and nonlinear measures of time series analysis for constructing functional networks from multivariate data (e.g. Pearson correlation, mutual information, event synchronization and event coincidence analysis). pyunicorn also features modern techniques of nonlinear analysis of single and pairs of time series such as recurrence quantification analysis (RQA), recurrence network analysis and visibility graphs.

For example, to generate a recurrence network with 1000 nodes from a sinusoidal signal and compute its network transitivity you simply need to type

```
import numpy as np
from pyunicorn.timeseries import RecurrenceNetwork

x = np.sin(np.linspace(0, 10 * np.pi, 1000))
net = RecurrenceNetwork(x, recurrence_rate=0.05)
print(net.transitivity())
```

The package provides special tools to analyze and model **spatially embedded** complex networks.

pyunicorn is **fast** because all costly computations are performed in compiled C, C++ and Fortran code. It can handle **large networks** through the use of sparse data structures. The package can be used interactively, from any Python script and even for parallel computations on large cluster architectures.

CHAPTER

TWO

DOWNLOAD

2.1 Code

Stable releases (https://github.com/pik-copan/pyunicorn/releases), Development version (https://github.com/pik-copan/pyunicorn)

Changelog, Contributions

2.2 Documentation

For extensive HTML documentation, jump right to the pyunicorn homepage (http://www.pik-potsdam.de/~donges/pyunicorn/). Recent PDF versions (http://www.pik-potsdam.de/~donges/pyunicorn/docs/) are also available.

On a local development version, HTML and PDF documentation can be generated using `Sphinx`:

```
$> pip install --user -e .
$> cd docs; make clean html latexpdf
```

2.3 Dependencies

`pyunicorn` is written in Python 3.7. The software is quite flexible, we have it running on Linux and MacOSX machines, the institute's IBM iDataPlex cluster and even on Windows. It relies on the following open source or freely available packages which have to be installed on your machine.

Required:

- Numpy (http://www.numpy.org/) 1.14+
- Scipy (http://www.scipy.org/) 1.0+
- igraph, python-igraph (http://igraph.org/) 0.7+

Optional *(used only in certain classes and methods)*:

- PyNGL (http://www.pyngl.ucar.edu/Download/) (for class NetCDFDictionary)
- netcdf4-python (http://unidata.github.io/netcdf4-python/) (for classes Data and NetCDFDictionary)
- Matplotlib (http://matplotlib.org/) 2.0+
- Matplotlib Basemap Toolkit (http://matplotlib.org/basemap/) (for drawing maps)
- mpi4py (https://bitbucket.org/mpi4py/mpi4py) (for parallelizing costly computations)
- Sphinx (http://sphinx-doc.org/) (for generating documentation)
- Cython (http://cython.org/) 0.27+ (for compiling code during development)

`Numpy`, `Scipy`, `Matplotlib`, `igraph` and other packages should be available via a package management system on Linux or MacOSX. All packages can be downloaded, compiled and installed following the instructions on their homepages.

An easy way to go may be a Python distribution like Anaconda (https://www.anaconda.com/distribution/) that already includes many libraries.

2.4 Installation

Stable release

Via the Python Package Index:

```
$> pip install pyunicorn
```

Development version

For a simple system-wide installation:

```
$> pip install -r requirements.txt .
```

Depending on your system, you may need root privileges. On UNIX-based operating systems (Linux, Mac OS X etc.) this is achieved with `sudo`.

For development, especially if you want to test `pyunicorn` from within the source directory:

```
$> pip install -r requirements.txt --user -e .
```

2.5 Reference

Please acknowledge and cite the use of this software and its authors when results are used in publications or published elsewhere. You can use the following reference:

> J.F. Donges, J. Heitzig, B. Beronov, M. Wiedermann, J. Runge, Q.-Y. Feng, L. Tupikina, V. Stolbova, R.V. Donner, N. Marwan, H.A. Dijkstra, and J. Kurths, **Unified functional network and nonlinear time series analysis for complex systems science: The pyunicorn package**, Chaos 25, 113101 (2015), doi:10.1063/1.4934554, (http://dx.doi.org/10.1063/1.4934554) Preprint: arxiv.org:1507.01571 [physics.data-an]. (http://arxiv.org/abs/1507.01571)

2.6 Funding

The development of `pyunicorn` has been supported by various funding sources, notably the German Federal Ministry for Education and Research (https://www.bmbf.de/en/index.html) (projects GOTHAM (http://belmont-gotham.org/) and CoSy-CC2 (http://cosy.pik-potsdam.de/)), the Leibniz Association (https://www.leibniz-gemeinschaft.de/en/home/) (projects ECONS (http://econs.pik-potsdam.de/) and DominoES (https://www.pik-potsdam.de/research/projects/activities/dominoes)), the German National Academic Foundation (https://www.studienstiftung.de/en/), and the Stordalen Foundation (http://www.stordalenfoundation.no/) via the Planetary Boundary Research Network (http://www.pb-net.org) (PB.net) among others.

CHAPTER

THREE

TUTORIALS

The tutorials are designed to be self-explanatory. For further details on the used classes and methods please refer to the *API*.

3.1 Constructing and analyzing a climate network

This tutorials illustrates the use of `climate` for constructing a climate network from given data in a commonly used format, performing a statistical analysis of the network and finally plotting the results on a map.

For example, our software can handle data from the NCEP/NCAR reanalysis 1 project like this monthly surface air temperature data set (a NetCDF file): ftp://ftp.cdc.noaa.gov/Datasets/ncep.reanalysis.derived/surface/air.mon.mean.nc

You can use `PyNgl` for plotting the results on maps (http://www.pyngl.ucar.edu/Download/). Alternatively, the tutorial saves the results as well as the grid information in text files which can be used for plotting in your favorite software.

This tutorial is also available as an ipython notebook.

```
# -*- coding: utf-8 -*-

"""
Tutorial on analyzing climate networks using Python.

Uses the Python packages ``core`` and ``climate`` providing all kinds of tools
related to climate networks. Written as part of a diploma / phd thesis in
Physics by Jonathan F. Donges (donges@pik-potsdam.de) at University of Potsdam
/ Humboldt University Berlin and Potsdam Institute of Climate Impact Research
(PIK),

Copyright 2008-2019.
"""

import numpy as np

from pyunicorn import climate


#
#  Settings
#

#  Related to data

#  Path and filename of NetCDF file containing climate data
```

(continues on next page)

(continued from previous page)

```
DATA_FILENAME = "../../../Daten/Reanalysis/NCEP-NCAR/air.mon.mean.nc"

#  Type of data file ("NetCDF" indicates a NetCDF file with data on a regular
#  lat-lon grid, "iNetCDF" allows for arbitrary grids - > see documentation).
#  For example, the "NetCDF" FILE_TYPE is compatible with data from the IPCC
#  AR4 model ensemble or the reanalysis data provided by NCEP/NCAR.
FILE_TYPE = "NetCDF"

#  Indicate data source (optional)
DATA_SOURCE = "ncep_ncar_reanalysis"

#  Name of observable in NetCDF file ("air" indicates surface air temperature
#  in NCEP/NCAR reanalysis data)
OBSERVABLE_NAME = "air"

#  Select a subset in time and space from the data (e.g., a particular region
#  or a particular time window, or both)
WINDOW = {"time_min": 0., "time_max": 0., "lat_min": 0, "lon_min": 0,
          "lat_max": 30, "lon_max": 0}  # selects the whole data set

#  Indicate the length of the annual cycle in the data (e.g., 12 for monthly
#  data). This is used for calculating climatological anomaly values
#  correctly.
TIME_CYCLE = 12

#  Related to climate network construction

#  For setting fixed threshold
THRESHOLD = 0.5

#  For setting fixed link density
LINK_DENSITY = 0.005

#  Indicates whether to use only data from winter months (DJF) for calculating
#  correlations
WINTER_ONLY = False

#
#  Print script title
#

print("\n")
print("Tutorial on how to use climate")
print("------------------------------")
print("\n")

#
#  Create a ClimateData object containing the data and print information
#

data = climate.ClimateData.Load(
    file_name=DATA_FILENAME, observable_name=OBSERVABLE_NAME,
    data_source=DATA_SOURCE, file_type=FILE_TYPE,
    window=WINDOW, time_cycle=TIME_CYCLE)

#  Print some information on the data set
```

(continues on next page)

(continued from previous page)

```
print(data)

#
#  Create a MapPlots object to manage 2D-plotting on the sphere
#

#  Comment this if you are not using pyngl for plotting!!!
map_plots = climate.MapPlots(data.grid, DATA_SOURCE)

#
#  Generate climate network using various procedures
#

#  One of several alternative similarity measures and construction mechanisms
#  may be chosen here

#  Create a climate network based on Pearson correlation without lag and with
#  fixed threshold
net = climate.TsonisClimateNetwork(
    data, threshold=THRESHOLD, winter_only=WINTER_ONLY)

#  Create a climate network based on Pearson correlation without lag and with
#  fixed link density
# net = climate.TsonisClimateNetwork(
#     data, link_density=LINK_DENSITY, winter_only=WINTER_ONLY)

#  Create a climate network based on Spearman's rank order correlation without
#  lag and with fixed threshold
# net = climate.SpearmanClimateNetwork(
#     data, threshold=THRESHOLD, winter_only=WINTER_ONLY)

#  Create a climate network based on mutual information without lag and with
#  fixed threshold
# net = climate.MutualInfoClimateNetwork(
#     data, threshold=THRESHOLD, winter_only=WINTER_ONLY)

#
#  Some calculations
#

print("Link density:", net.link_density)

#  Get degree
degree = net.degree()
#  Get closeness
closeness = net.closeness()
#  Get betweenness
betweenness = net.betweenness()
#  Get local clustering coefficient
clustering = net.local_clustering()
#  Get average link distance
ald = net.average_link_distance()
#  Get maximum link distance
mld = net.max_link_distance()

#
```

(continues on next page)

(continued from previous page)

```
#  Save results to text file
#

#  Save the grid (mainly vertex coordinates) to text files
data.grid.save_txt(filename="grid.txt")

#  Save the degree sequence. Other measures may be saved similarly.
np.savetxt("degree.txt", degree)

#
#  Plotting
#

#  Comment everything below if you are not using pyngl for plotting!

#  Add network measures to the plotting queue
map_plots.add_dataset("Degree", degree)
map_plots.add_dataset("Closeness", closeness)
map_plots.add_dataset("Betweenness (log10)", np.log10(betweenness + 1))
map_plots.add_dataset("Clustering", clustering)
map_plots.add_dataset("Average link distance", ald)
map_plots.add_dataset("Maximum link distance", mld)

#  Change the map projection
map_plots.resources.mpProjection = "Robinson"
map_plots.resources.mpCenterLonF = 0

#  Change the levels of contouring
map_plots.resources.cnLevelSelectionMode = "EqualSpacedLevels"
map_plots.resources.cnMaxLevelCount = 20

# map_plots.resources.cnRasterSmoothingOn = True
# map_plots.resources.cnFillMode = "AreaFill"

map_plots.generate_map_plots(file_name="climate_network_measures",
                             title_on=False, labels_on=True)
```

3.2 Recurrence network analysis of the logistic map

This tutorial demonstrates how to use `timeseries` for a nonlinear time series analysis of a realization of the chaotic logistic map.

This tutorial is also available as an ipython notebook.

```
# -*- coding: utf-8 -*-

"""
Tutorial on how to handle recurrence plots and recurrence networks using
Python, based on the timeseries package.

Written as part of a PhD thesis in Physics by Jonathan F. Donges
(donges@pik-potsdam.de) at the Potsdam Institute of Climate Impact Research
(PIK) and Humboldt University Berlin,

Copyright 2008-2019.
```

(continues on next page)

(continued from previous page)

```
"""

# array object and fast numerics
import numpy as np

# plotting facilities
import pylab

from pyunicorn.timeseries import RecurrencePlot, RecurrenceNetwork


#
#  Functions
#
def logistic_map(x0, r, T):
    """
    Returns a time series of length T using the logistic map
    x_(n+1) = r*x_n(1-x_n) at parameter r and using the initial condition x0.

    INPUT: x0 - Initial condition, 0 <= x0 <= 1
           r - Bifurcation parameter, 0 <= r <= 4
           T - length of the desired time series
    TODO: Cythonize
    """
    #  Initialize the time series array
    timeSeries = np.empty(T)

    timeSeries[0] = x0
    for i in range(1, len(timeSeries)):
        xn = timeSeries[i-1]
        timeSeries[i] = r * xn * (1 - xn)

    return timeSeries


def logistic_map_lyapunov_exponent(timeSeries, r):
    """
    Returns the Lyapunov exponent of the logistic map for different r.

    INPUT: timeSeries - The time series generated by a logistic map
                   r - the bifurcation parameter
    """
    lyap = np.log(r) + (np.log(np.abs(1 - 2 * timeSeries))).mean()

    return lyap

#
#  Settings
#

#  Parameters of logistic map
r = 3.679  # Bifurcation parameter
x0 = 0.7   # Initial value

#  Length of the time series
T = 150
```

(continues on next page)

(continued from previous page)

```
#  Settings for the embedding
DIM = 1  # Embedding dimension
TAU = 0  # Embedding delay

#  Settings for the recurrence plot
EPS = 0.05  # Fixed threshold
RR = 0.05   # Fixed recurrence rate
# Distance metric in phase space ->
# Possible choices ("manhattan","euclidean","supremum")
METRIC = "supremum"

#
#  Main script
#
#  Create a time series using the logistic map
time_series = logistic_map(x0, r, T)

#  Print the time series
print(time_series)
#  Plot the time series
pylab.plot(time_series, "r")
#  You can include LaTex labels...
pylab.xlabel("$n$")
pylab.ylabel("$x_n$")

#  Generate a recurrence plot object with fixed recurrence threshold EPS
rp = RecurrencePlot(time_series, dim=DIM, tau=TAU, metric=METRIC,
                    normalize=False, threshold=EPS)

#  Show the recurrence plot
pylab.matshow(rp.recurrence_matrix())
pylab.xlabel("$n$")
pylab.ylabel("$n$")
pylab.show()

#  Calculate and print the recurrence rate
print("Recurrence rate:", rp.recurrence_rate())

#  Calculate some standard RQA measures
DET = rp.determinism(l_min=2)
LAM = rp.laminarity(v_min=2)

print("Determinism:", DET)
print("Laminarity:", LAM)

#  Generate a recurrence plot object with fixed recurrence rate RR
rp = RecurrencePlot(time_series, dim=DIM, tau=TAU, metric=METRIC,
                    normalize=False, recurrence_rate=RR)

#  Calculate and print the recurrence rate again to check if it worked...
RR = rp.recurrence_rate()
print("Recurrence rate:", RR)

#  Calculate some standard RQA measures
DET = rp.determinism(l_min=2)
```

(continues on next page)

(continued from previous page)

```
LAM = rp.laminarity(v_min=2)

print("Determinism:", DET)
print("Laminarity:", LAM)

#  Generate a recurrence network at fixed recurrence rate
rn = RecurrenceNetwork(time_series, dim=DIM, tau=TAU, metric=METRIC,
                       normalize=False, recurrence_rate=RR)

#  Calculate average path length, transitivity and assortativity
L = rn.average_path_length()
T = rn.transitivity()
C = rn.global_clustering()
R = rn.assortativity()

print("Average path length:", L)
print("Transitivity:", T)
print("Global clustering:", C)
print("Assortativity:", R)
```

CHAPTER

FOUR

METHODS

A brief introduction to the methods, measures and algorithms provided by `pyunicorn`.

4.1 General complex networks

Many standard complex network measures, network models and algorithms are supported, most of them inherited from the `igraph` package, e.g., degree, closeness and betweenness centralities, clustering coefficient and transitivity or commmunity detection algorithms and network models such as Erdos-Renyi or Barabasi-Albert. Moreover, a number of less common network statistics like Newman's or Arenas' random walk betweenness can be computed. Reading and saving network data from and to many common data formats is possible.

- *core.network*

4.2 Spatially embedded networks

`pyunicorn` includes measures and models specifically designed for spatially embedded networks (or simply spatial networks) via the GeoNetwork and Grid classes.

- *core.geo_network*
- *core.grid*

4.3 Interacting/interdependent/multiplex networks / networks of networks

The InteractingNetworks class provides a rich collection of network measures and models specifically designed for investigating the structure of networks of networks (also called interacting networks, interdependent networks or multiplex networks in different contexts). Examples include the cross-link density of connections between different subnetworks or the cross-shortest path betweenness quantifying the importance of nodes for mediating interactions between different subnetworks. Models of interacting networks allow to assess the degree of organization of the cross-connectivity between subnetworks.

- *core.interacting_networks*

4.4 Node-weighted network measures / node-splitting invariance

Node-weighted networks measures derived following the node-splitting invariance approach are useful for studying systems with nodes representing subsystems of heterogeneous size, weight, area, volume or importance, e.g., nodes representing grid cells of widely different area in climate networks or voxels of differing volume in functional brain networks. `pyunicorn` provides node-weighted variants of most standard and non-standard measures for networks as well as interacting networks.

- *core.network*
- *core.interacting_networks*

4.5 Climate networks / Coupled climate networks

`pyunicorn` provides classes for the easy construction and analysis of the statistical interdependency structure within and between fields of time series (functional networks) using various similarity measures such as Pearson and Spearman correlation, lagged linear correlation, mutual information and event synchronization. Climate networks allow the analysis of single fields of time series, whereas coupled climate networks focus on studying the interrelationships between two fields of time series. While there is a historical focus on applications to climate data, those methods can also be applied to other sources of time series data such as neuroscientific (e.g., FMRI and EEG data) or financial data (e.g., stock market indices).

- *climate.climate_network*
- *climate.coupled_climate_network*
- *climate.climate_data*

4.6 Recurrence networks / recurrence quantification analysis / recurrence plots

Recurrence analysis is a powerful method for studying nonlinear systems, particularly based on univariate and multivariate time series data. Recurrence quantification analysis (RQA) and recurrence network analysis (RNA) allow to classify different dynamical regimes in time series and to detect regime shifts, dynamical transitions or tipping points, among many other applications. Bivariate methods such as joint recurrence plots/networks, cross recurrence plots or inter system recurrence networks allow to investigate the coupling structure between two dynamical systems based on time series, including methods to detect the directionality of coupling. Recurrence analysis is applicable to general time series data from many fields such as climatology, paleoclimatology, medicine, neuroscience or economics.

- *timeseries.recurrence_plot*
- *timeseries.recurrence_network*
- *timeseries.joint_recurrence_plot*
- *timeseries.joint_recurrence_network*
- *timeseries.cross_recurrence_plot*
- *timeseries.inter_system_recurrence_network*

4.7 Visibility graph analysis

Visibility graph analysis is an alternative approach to nonlinear time series analysis, allowing to study among others fractal properties and long-term memory in time series. As a special feature, `pyunicorn` provides time-directed measures such as advanced and retarded degree/clustering that can be used for designing tests for time-irreversibility (time-reversal asymmetry) of processes.

- *timeseries.visibility_graph*

4.8 Surrogate time series

Surrogate time series are useful for testing hypothesis on observed time series properties, e.g., on what features of a time series are expected to arise with high probability for randomized time series with the same autocorrelation structure. `pyunicorn` can be used to generate various types of time series surrogates, including white noise surrogates, Fourier surrogates, amplitude adjusted Fourier (AAFT) surrogates or twin surrogates (conserving the recurrence structure of the underlying time series).

- *timeseries.surrogates*

CHAPTER

FIVE

API

Release
0.6.1

Date
Jan 20, 2023

`pyunicorn` consists of six subpackages, where the `core` and `utils.mpi` namespaces are to be accessed by calling `import pyunicorn`. The subpackage `eventseries` only contains one function and will be extended in one of the next versions. All subpackages except for `utils` directly export the classes defined in their submodules.

5.1 core

General network analysis and modeling.

5.1.1 core.data

5.1.2 core.geo_network

5.1.3 core.grid

5.1.4 core.interacting_networks

5.1.5 core.netcdf_dictionary

5.1.6 core.network

5.1.7 core.resistive_network

5.2 climate

Constructing and analysing climate networks, related climate data analysis.

5.2.1 climate.climate_data

5.2.2 climate.climate_network

5.2.3 climate.coupled_climate_network

5.2.4 climate.coupled_tsonis

5.2.5 climate.havlin

5.2.6 climate.hilbert

5.2.7 climate.map_plots

5.2.8 climate.mutual_info

5.2.9 climate.partial_correlation

5.2.10 climate.rainfall

5.2.11 climate.spearman

5.2.12 climate.tsonis

5.3 timeseries

Recurrence plots, recurrence networks, multivariate extensions and visibility graph analysis of time series. Time series surrogates for significance testing.

5.3.1 timeseries.cross_recurrence_plot

5.3.2 timeseries.inter_system_recurrence_network

5.3.3 timeseries.joint_recurrence_network

5.3.4 timeseries.joint_recurrence_plot

5.3.5 timeseries.recurrence_network

5.3.6 timeseries.recurrence_plot

5.3.7 timeseries.surrogates

5.3.8 timeseries.visibility_graph

5.4 funcnet

Constructing and analysing general functional networks.

5.4.1 funcnet.coupling_analysis

5.4.2 funcnet.coupling_analysis_pure_python

5.5 eventseries

Analysis of event series.

5.5.1 eventseries.eca

5.6 utils

Parallelization, interactive network navigator, helpers.

5.6.1 utils.mpi

Module for parallelization using mpi4py.

Allows for easy parallelization in master/slaves mode with one master submitting function or method calls to slaves. Uses mpi4py if available, otherwise processes calls sequentially in one process.

Examples:

Save the following lines in `demo_mpi.py` and run:

```
> mpirun -n 10 python demo_mpi.py
```

1. Use master/slaves parallelization with the Network class:

```
from pyunicorn import Network, mpi


def master():
    net = Network.BarabasiAlbert(n_nodes=1000, n_links_each=10)
    print(net.newman_betweenness())
    mpi.info()

mpi.run()
```

2. Do a Monte Carlo simulation as master/slaves:

```
from pyunicorn import Network, mpi


def do_one():
    net = Network.BarabasiAlbert(n_nodes=100, n_links_each=10)
    return net.global_clustering()


def master():
    n = 1000
    for i in range(0, n):
        mpi.submit_call("do_one", ())
    s = 0
```

(continues on next page)

(continued from previous page)

```
    for i in range(0, n):
        s += mpi.get_next_result()
    print(s/n)
    mpi.info()

mpi.run()
```

3. Do a parameter scan without communication with a master, and just save the results in files:

```
import numpy
from pyunicorn import Network, mpi

offset = 10
n_max = 1000
s = 0
n = mpi.rank + offset
while n <= n_max + offset:
    s += Network.BarabasiAlbert(n_nodes=n).global_clustering()
    n += mpi.size

numpy.save("s"+str(mpi.rank), s)
```

exception pyunicorn.utils.mpi.**MPIException**(*value*)

Bases: Exception

__init__(*value*)

__str__()

Return str(self).

__weakref__

list of weak references to the object (if defined)

pyunicorn.utils.mpi.**abort**()

Abort execution on all MPI nodes immediately.

Can be called by master and slaves.

pyunicorn.utils.mpi.**am_master = True**

(bool) indicates that this MPI node is the master.

pyunicorn.utils.mpi.**am_slave = False**

(bool) indicates that this MPI node is a slave.

pyunicorn.utils.mpi.**assigned = {}**

(dictionary) assigned[id] is the slave assigned to the call with that id.

pyunicorn.utils.mpi.**available = False**

(bool) indicates that slaves are available.

pyunicorn.utils.mpi.**get_next_result**()

Return result of next earlier submitted call whose result has not yet been got.

Can only be called by the master.

If the call is not yet finished, waits for it to finish.

Return type
object

Returns
return value of call, or None of there are no more calls in the queue.

pyunicorn.utils.mpi.**get_result**(*id*)

Return result of earlier submitted call.

Can only be called by the master.

If the call is not yet finished, waits for it to finish. Results should be collected in the same order as calls were submitted. For each slave, the results of calls assigned to that slave must be collected in the same order as those calls were submitted. Can only be called once per call.

Parameters
id (*object*) – id of an earlier submitted call, as provided to or returned by submit_call().

Return type
object

Returns
return value of call.

pyunicorn.utils.mpi.**info**()

Print processing statistics.

Can only be called by the master.

pyunicorn.utils.mpi.**n_processed = array([0])**

(list of ints) n_processed[rank] is the total number of calls processed by MPI node rank. On slave i, only total_time[i] is available.

pyunicorn.utils.mpi.**n_slaves = 0**

(int) no. of slaves available.

pyunicorn.utils.mpi.**queue = []**

(list) ids of submitted calls

pyunicorn.utils.mpi.**rank = 0**

(int) rank of this MPI node (0 is the master).

pyunicorn.utils.mpi.**results = {}**

(dictionary) if mpi is not available, the result of submit_call(..., id=a) will be cached in results[a] until get_result(a).

pyunicorn.utils.mpi.**run**(*verbose=False*)

Run in master/slaves mode until master() finishes.

Must be called on all MPI nodes after function master() was defined.

On the master, run() calls master() and returns when master() returns.

On each slave, run() calls slave() if that is defined, or calls serve() otherwise, and returns when slave() returns, or when master() returns on the master, or when master calls terminate().

Parameters
verbose (*bool*) – whether processing information should be printed.

pyunicorn.utils.mpi.**size = 1**

(int) number of MPI nodes (master and slaves).

pyunicorn.utils.mpi.**slave_queue = [[]]**

(list of lists) slave_queue[i] contains the ids of calls assigned to slave i.

pyunicorn.utils.mpi.**start_time = 1674233325.3783672**

(float) starting time of this MPI node.

pyunicorn.utils.mpi.**stats = []**

(list of dictionaries) stats[id] contains processing statistics for the last call with this id. Keys:

- "id": id of the call

- "rank": MPI node who processed the call
- "this_time": wall time for processing the call
- "time_over_est": quotient of actual over estimated wall time
- "n_processed": no. of calls processed so far by that slave, including this
- "total_time": total wall time until this call was finished

pyunicorn.utils.mpi.**submit_call**(*name_to_call*, *args=()*, *kwargs={}*, *module='__main__'*, *time_est=1*, *id=None*, *slave=None*)

Submit a call for parallel execution.

If called by the master and slaves are available, the call is submitted to a slave for asynchronous execution.

If called by a slave or if no slaves are available, the call is instead executed synchronously on this MPI node.

Examples:

1. Provide ids and time estimate explicitly:

```
for n in range(0,10):
    mpi.submit_call("doit", (n,A[n]), id=n, time_est=n**2)

for n in range(0,10):
    result[n] = mpi.get_result(n)
```

2. Use generated ids stored in a list:

```
for n in range(0,10):
    ids.append(mpi.submit_call("doit", (n,A[n])))

for n in range(0,10):
    results.append(mpi.get_result(ids.pop()))
```

3. Ignore ids altogether:

```
for n in range(0,10):
    mpi.submit_call("doit", (n,A[n]))

for n in range(0,10):
    results.append(mpi.get_next_result())
```

4. Call a module function and use keyword arguments:

```
mpi.submit_call("solve", (), {"a":a, "b":b},
    module="numpy.linalg")
```

5. Call a static class method from a package:

```
mpi.submit_call("Network._get_histogram", (values, n_bins),
    module="pyunicorn")
```

Note that it is module="pyunicorn" and not module="pyunicorn.network" here.

Parameters

- **name_to_call** (*str*) – name of callable object (usually a function or static method of a class) as contained in the namespace specified by module.
- **args** (*tuple*) – the positional arguments to provide to the callable object. Tuples of length 1 must be written (arg,). Default: ()

- **kwargs** (*dict*) – the keyword arguments to provide to the callable object. Default: {}
- **module** (*str*) – optional name of the imported module or submodule in whose namespace the callable object is contained. For objects defined on the script level, this is "__main__", for objects defined in an imported package, this is the package name. Must be a key of the dictionary sys.modules (check there after import if in doubt). Default: "__main__"
- **time_est** (*float*) – estimated relative completion time for this call; used to find a suitable slave. Default: 1
- **id** (*object or None*) – unique id for this call. Must be a possible dictionary key. If None, a random id is assigned and returned. Can be re-used after get_result() for this is. Default: None
- **slave** (*int > 0 and < mpi.size, or None*) – optional no. of slave to assign the call to. If None, the call is assigned to the slave with the smallest current total time estimate. Default: None

Return object
id of call, to be used in get_result().

pyunicorn.utils.mpi.**terminate**()

Tell all slaves to terminate.

Can only be called by the master.

pyunicorn.utils.mpi.**total_time = array([0.])**

(list of floats) total_time[rank] is the total wall time until that node finished its last call. On slave i, only total_time[i] is available.

pyunicorn.utils.mpi.**total_time_est = array([inf])**

(numpy array of ints) total_time_est[i] is the current estimate of the total time MPI slave i will work on already submitted calls. On slave i, only total_time_est[i] is available.

5.6.2 utils.navigator

CHAPTER

SIX

DEVELOPMENT

6.1 Test suite

Before committing changes to the code base, please make sure that all tests pass. The test suite is managed by tox (http://tox.readthedocs.io/) and configured to use system-wide packages when available. Thus to avoid frequent waiting, we recommend you to install the current versions of the following packages:

```
$> pip install networkx matplotlib basemap Sphinx
$> pip install tox pylint pytest pytest-xdist pytest-flake8
```

The test suite can be run from anywhere in the project tree by issuing:

```
$> tox
```

To expose the defined test environments and target them independently:

```
$> tox -l
$> tox -e units,style
```

To test individual files:

```
$> py.test                   tests/test_core/TestNetwork.py  # unit tests
$> py.test --doctest-modules pyunicorn/core/network.py       # doctests
$> py.test --flake8          pyunicorn/core/network.py       # style
$> pylint                    pyunicorn/core/network.py       # code analysis
```

6.2 Mailing list

Not implemented yet.

CHAPTER

SEVEN

CHANGELOG

A summary of major changes made in each release of `pyunicorn`:

0.6.1

- Fixed some bugs and compatibility issues.
- Improved test framework.
- Added pyunicorn description paper reference to all code files.

0.6.0

- Migrated from Python 2.7 to Python 3.7.
- Completed transition from `Weave` to `Cython`.
- Added Event Coincidence Analysis.

0.5.2

- Updated test suite and `Travis CI`.

0.5.1

- Added reference to pyunicorn description paper published in the journal Chaos.

0.5.0

- Substantial update of `CouplingAnalysis`.
- New methods in `RecurrenceNetwork`: `transitivity_dim_single_scale`, `local_clustering_dim_single_scale`.
- Renamed time-directed measures in `VisibilityGraph`: `left/right` -> `retarded/advanced`.
- Improved documentation and extended publication list.
- Began transition from `Weave` to `Cython`.
- Added unit tests and improved Pylint compliance.
- Set up continuous testing with Travis CI.
- Fixed some minor bugs.

0.4.1

- Removed a whole lot of `get_` s from the API. For example, `Network.get_degree()` is now `Network.degree()`.
- Fixed some minor bugs.

0.4.0

- Restructured package (subpackages: `core`, `climate`, `timeseries`, `funcnet`, `utils`).
- Removed dependencies: `Pysparse`, `PyNio`, `progressbar`.
- Added a module for resistive networks.

- Switched to `tox` for test suite management.
- Ensured PEP8 and PyFlakes compliance.

0.3.2

- Fixed some minor bugs.
- Switched to `Sphinx` documentation system.

0.3.1

- First public release of `pyunicorn`.

CHAPTER

EIGHT

PUBLICATIONS

References to peer-reviewed publications, theses and reports describing in detail and applying the methods implemented in the `pyunicorn` package.

8.1 General complex networks

8.1.1 *Review papers*

[Newman2003], [Boccaletti2006], [Costa2007].

8.1.2 *Further network papers*

[Watts1998], [Newman2001], [Newman2002], [Arenas2003], [Newman2005], [Soffer2005], [Holme2007], [Tsonis2008a], [Ueoka2008].

8.2 Spatially embedded networks

[Bartelemy2011].

8.3 Interacting/interdependent networks / networks of networks

8.3.1 *Introduction to structural analysis of interacting networks*

[Donges2011a].

8.4 Node-weighted network measures / node-splitting invariance

8.4.1 *Introduction*

[Heitzig2012].

8.4.2 *Random graph models and network surrogates for interacting networks*

[Schultz2010].

8.4.3 *Analysis of node-weighted interacting networks*

[Wiedermann2011], [Wiedermann2013].

8.5 Climate data analysis (general)

[Bretherton1992].

8.6 Climate networks / Coupled climate networks

8.6.1 *Comparing linear and nonlinear construction of climate networks*

[Donges2009a].

8.6.2 *Studying the dynamical structure of the surface air temperature field*

[Donges2009b], [Radebach2010].

8.6.3 *Introduction to coupled climate networks and applications*

[Schultz2010], [Donges2011a], [Wiedermann2011].

8.6.4 *Review of climate network analysis (in Chinese!)*

[Zou2011].

8.6.5 *Visualization of climate networks*

[Tominski2011].

8.6.6 *Evolving climate networks*

[Radebach2013].

8.6.7 *General*

[Tsonis2004], [Tsonis2006], [Gozolchiani2008], [Tsonis2008b], [Tsonis2008c], [Yamasaki2008], [Donges2009c], [Yamasaki2009].

8.7 Power Grids/Power Networks

8.7.1 *Resistance based networks*

[Schultz2014], [Schultz2014a].

8.8 Time series analysis and synchronization

8.8.1 *General*

[Pecora1998], [Schreiber2000], [Kraskov2004], [Kantz2006], [Thiel2006], [Bergner2008], [Pompe2011], [Runge2012b].

8.8.2 *Event synchronization*

[Quiroga2002], [Boers2014].

8.9 Recurrence networks / quantification analysis / plots

8.9.1 *Review of recurrence plots and RQA*

[Marwan2007].

8.9.2 *Introduction and application of recurrence networks in the context of RQA*

[Marwan2009].

8.9.3 *A thorough introduction to recurrence network analysis*

[Donner2010b].

8.9.4 *Discussion of choosing an appropriate recurrence threshold*

[Donner2010a], [Zou2010].

8.9.5 *Review of various methods for network-based time series analysis*

[Donner2011a].

8.9.6 *Introduction to measures of (fractal) transitivity dimensions*

[Donner2011b].

8.9.7 *Applications of recurrence network analysis to paleoclimate data*

[Donges2011b], [Donges2011c], [Feldhoff2012].

8.9.8 *Theory of recurrence networks*

[Donges2012], [Zou2012].

8.9.9 *Multivariate extensions of recurrence network analysis*

[Feldhoff2012], [Feldhoff2013].

8.9.10 *General*

[Ngamga2007], [Xu2008], [Schinkel2009].

8.10 Visibility graph analysis

8.10.1 *Introduction*

[Lacasa2008].

8.10.2 *Application to geophysical time series*

[Donner2012].

8.10.3 *Tests for time series irreversibility*

[Donges2013].

CHAPTER

NINE

LICENSE

Copyright
© 2008-2019 Jonathan F. Donges and pyunicorn authors.

License
BSD (3-clause)

URL
http://www.pik-potsdam.de/members/donges/software

```
Copyright (C) 2008-2019, Jonathan F. Donges (Potsdam-Institute for Climate
Impact Research), pyunicorn authors

All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

    * Redistributions of source code must retain the above copyright notice, this
      list of conditions and the following disclaimer.

    * Redistributions in binary form must reproduce the above copyright notice,
      this list of conditions and the following disclaimer in the documentation
      and/or other materials provided with the distribution.

    * Neither the name of pyunicorn authors and the Potsdam-Institute for
      Climate Impact Research nor the names of its contributors may be used to
      endorse or promote products derived from this software without specific
      prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

CHAPTER

TEN

CONTACT

Please acknowledge and cite the use of this software and its authors when results are used in publications or published elsewhere. You can use the following reference:

> J.F. Donges, J. Heitzig, B. Beronov, M. Wiedermann, J. Runge, Q.-Y. Feng, L. Tupikina, V. Stolbova, R.V. Donner, N. Marwan, H.A. Dijkstra, and J. Kurths, **Unified functional network and nonlinear time series analysis for complex systems science: The pyunicorn package**, Chaos 25, 113101 (2015), doi:10.1063/1.4934554, (http://dx.doi.org/10.1063/1.4934554) Preprint: arxiv.org:1507.01571 [physics.data-an]. (http://arxiv.org/abs/1507.01571)

10.1 Funding

The development of `pyunicorn` has been supported by various funding sources, notably the German Federal Ministry for Education and Research (https://www.bmbf.de/en/index.html) (projects GOTHAM (http://belmont-gotham.org/) and CoSy-CC2 (http://cosy.pik-potsdam.de/)), the Leibniz Association (https://www.leibniz-gemeinschaft.de/en/home/) (projects ECONS (http://econs.pik-potsdam.de/) and DominoES (https://www.pik-potsdam.de/research/projects/activities/dominoes)), the German National Academic Foundation (https://www.studienstiftung.de/en/), and the Stordalen Foundation (http://www.stordalenfoundation.no/) via the Planetary Boundary Research Network (http://www.pb-net.org) (PB.net) among others.

URL
http://www.pik-potsdam.de/members/donges/software

Mail
Jonathan Donges, Potsdam Institute for Climate Impact Research, P.O. Box 60 12 03, D-14412 Potsdam, Germany

Authors
Written as part of a diploma/PhD thesis in physics by Jonathan F. Donges (donges@pik-potsdam.de) at Humboldt University Berlin and the Potsdam Institute for Climate Impact Research (PIK) and completed at the University of Potsdam, Germany. Substantially extended by Jobst Heitzig (heitzig@pik-potsdam.de).

Contributors

- Jakob Runge (extended `core` and `climate`)
- Alexander Radebach
- Hanna Schultz
- Marc Wiedermann (extended `core` and `climate`)
- Alraune Zech (alrauni@web.de) (extended `timeseries` during an internship at PIK)
- Jan Feldhoff (feldhoff@pik-potsdam.de) (extended `timeseries`)
- Aljoscha Rheinwalt
- Hannes Kutza

- Boyan Beronov (beronov@pik-potsdam.de) (restructured, consolidated and updated codebase during an internship at PIK)
- Paul Schultz (pschultz@pik-potsdam.de), Stefan Schinkel (mail@dreeg.org) (supplied `resistive_network` and corresponding tests)
- Wolfram Barfuss (barfuss@pik-potsdam.de) (package extensions and maintenance)
- Nils Harmening (cythonized `weave.inline` functions, extended testing framework, migrated from Python 2.7 to 3.6)
- Jonathan Krönke (extending test framework, package extensions and maintenance)

BIBLIOGRAPHY

[Newman2003] M.E.J. Newman. "The structure and function of complex networks". In *SIAM Review*, vol. 45 (no. 2), p167-256 (2003) doi:10.1137/S003614450342480 (http://dx.doi.org/10.1137/S003614450342480)

[Boccaletti2006] S. Boccaletti, V. Latora, Y. Moreno, M. Chavez, D.U. Hwang. "Complex networks: structure and dynamics". In *Physics Reports*, vol. 424 (no. 4-5), p175-308 (2006) doi:10.1016/j.physrep.2005.10.009 (http://dx.doi.org/10.1016/j.physrep.2005.10.009)

[Costa2007] L.D.F. Costa, F.A. Rodrigues, G. Travieso, P.R. Villas Boas. "Characterization of complex networks: A survey of measurements". In *Advances in Physics*, vol. 56(1), 167-242 (2007) doi:10.1080/00018730601170527 (http://dx.doi.org/10.1080/00018730601170527)

[Watts1998] D.J. Watts and S.H. Strogatz. "Collective dynamics of small-world networks". In *Nature* vol. 393, 440–442 (1998) doi:10.1038/30918 (http://dx.doi.org/10.1038/30918)

[Newman2001] M.E.J. Newman. "Scientific collaboration networks. II. Shortest paths, weighted networks, and centrality". In *Physical Review E* vol. 64.1, 016132 (2001) doi:10.1103/PhysRevE.64.016132 (http://dx.doi.org/10.1103/PhysRevE.64.016132)

[Newman2002] M.E.J. Newman. "Assortative mixing in networks". In *Physical Review Letters*, vol. 89.20, 208701 (2002) doi:10.1103/PhysRevLett.112.068103 (http://dx.doi.org/10.1103/PhysRevLett.112.068103)

[Arenas2003] A. Arenas, A. Cabrales, A. Díaz-Guilera, R. Guimerà, F. Vega-Redondo. "Search and Congestion in Complex Networks". In "Statistical Mechanics of Complex Networks", In *Lecture Notes in Physics*, vol. 625, p175-194 (2003) doi:10.1007/978-3-540-44943-0_11 (http://dx.doi.org/10.1007/978-3-540-44943-0_11)

[Newman2005] M.E.J. Newman. "A measure of betweenness centrality based on random walks". In *Social Networks*, vol 27 (no. 1), p39–54 (2005) doi:10.1016/j.socnet.2004.11.009 (http://dx.doi.org/10.1016/j.socnet.2004.11.009)

[Soffer2005] S.N. Soffer and A. Vázquez "Network clustering coefficient without degree-correlation biases". In *Physical Review E*, vol. 71, 057101 (2005) doi:10.1103/PhysRevE.71.057101 (http://dx.doi.org/10.1103/PhysRevE.71.057101)

[Holme2007] P. Holme, S.M. Park, B.J. Kim, C.R. Edling. "Korean university life in a network perspective: Dynamics of a large affiliation network". In *Physica A: Statistical Mechanics and its Applications*, vol. 373, p821-830 (2007) doi:10.1016/j.physa.2006.04.066 (http://dx.doi.org/10.1016/j.physa.2006.04.066)

[Tsonis2008a] A.A. Tsonis, K.L. Swanson, G. Wang. "Estimating the clustering coefficient in scale-free networks on lattices with local spatial correlation structure". In *Physica A: Statistical Mechanics and its Applications*, vol. 387 (no. 21) p5287-5294 (2008) doi:10.1016/j.physa.2008.05.048 (http://dx.doi.org/10.1016/j.physa.2008.05.048)

[Ueoka2008] Y. Ueoka, T. Suzuki, T. Ikeguchi, Y. Horio. "Efficiency of Statistical Measures to Estimate Network Structure of Chaos Coupled Systems". Proceedings of NOLTA (2008) http://tsuzuki.ise.ibaraki.ac.jp/MyPaper/Meeting/08NOLTA.pdf

[Bartelemy2011] M. Barthelemy. "Spatial networks". In *Physics Reports*, vol. 499 (no. 1-3), p1-101 (2011) doi:10.1016/j.physrep.2010.11.002 (http://dx.doi.org/10.1016/j.physrep.2010.11.002)

[Donges2011a] J.F. Donges, H.C.H. Schultz, N. Marwan, Y. Zou, J. Kurths. "Investigating the topology of interacting networks - Theory and application to coupled climate subnetworks". In *European Physical Journal B: Condensed Matter and Complex Systems*, vol. 84 (no. 4) p635-652 (2011) doi:10.1140/epjb/e2011-10795-8 (http://dx.doi.org/10.1140/epjb/e2011-10795-8)

[Heitzig2012] J. Heitzig, J. F. Donges, Y. Zou, N. Marwan, J. Kurths. "Node-weighted measures for complex networks with spatially embedded, sampled, or differently sized nodes". In *European Physical Journal B: Condensed Matter and Complex Systems*, vol. 85 p38 (2012) doi:10.1140/epjb/e2011-20678-7 (http://dx.doi.org/10.1140/epjb/e2011-20678-7)

[Schultz2010] H.C.H. Schultz. "Coupled climate networks: Investigating the terrestrial atmosphere's dynamical structure". Diploma thesis, Free University, Berlin (2010)

[Wiedermann2011] M. Wiedermann. "Coupled climate network analysis of multidecadal dynamics in the Arctic". Bachelor's thesis, Humboldt University, Berlin (2011)

[Wiedermann2013] M. Wiedermann, J.F. Donges, J. Heitzig, J. Kurths. "Node-weighted interacting network measures improve the representation of real-world complex systems". In *Europhysics Letters*, vol. 102.2, 28007 (2013) doi:10.1209/0295-5075/102/28007 (http://dx.doi.org/10.1209/0295-5075/102/28007)

[Zemp2014] D.C. Zemp, M. Wiedermann, J. Kurths, A. Rammig, J.F. Donges. "Node-weighted measures for complex networks with directed and weighted edges for studying continental moisture recycling". In *Europhysics Letters*, vol. 107.5, p58005 (2014) doi:10.1209/0295-5075/107/58005 (http://dx.doi.org/10.1209/0295-5075/107/58005)

[Bretherton1992] C.S. Bretherton, C. Smith, J.M. Wallace. "An intercomparison of methods for finding coupled patterns in climate data". In *Journal of Climate*, vol. 5, p541-560 (1992) doi:10.1175/1520-0442(1992)005<0541:AIOMFF>2.0.CO;2 (http://dx.doi.org/10.1175/1520-0442(1992)005%3C0541%3AAIOMFF%3E2.0.CO%3B2)

[Donges2009a] J.F. Donges, Y. Zou, N. Marwan, J. Kurths. "Complex networks in climate dynamics". In *European Physical Journal Special Topics*, vol. 174 (no. 1), p157-179 (2009) doi:10.1140/epjst/e2009-01098-2 (http://dx.doi.org/10.1140/epjst/e2009-01098-2)

[Donges2009b] J.F. Donges, Y. Zou, N. Marwan, J. Kurths. "The backbone of the climate network". In *Europhysics Letters*, vol. 87 (no. 4), 48007 (2009) doi:10.1209/0295-5075/87/48007 (http://dx.doi.org/10.1209/0295-5075/87/48007)

[Radebach2010] A. Radebach. "Evolving climate networks: Investigating the evolution of correlation structure of the Earth's climate system". Diploma thesis, Humboldt University, Berlin (2010)

[Zou2011] Y. Zou, J.F. Donges, J. Kurths. "Recent advances in complex climate network analysis". In *Complex Systems and Complexity Science*, vol. 8 (no. 1), p27-38 (2011)

[Tominski2011] C. Tominski, J.F. Donges, T. Nocke. "Information Visualization in Climate Research". In *Proceedings of the International Conference on Information Visualisation (IV), London*, p298-305 (2011) doi:10.1109/IV.2011.12 (http://dx.doi.org/10.1109/IV.2011.12)

[Radebach2013] A. Radebach, R.V. Donner, J. Runge, J.F. Donges, J. Kurths. "Disentangling different types of El Nino episodes by evolving climate network analysis". In *Physical Review E*, vol. 88, 052807 (2013) doi:10.1103/PhysRevE.88.052807 (http://dx.doi.org/10.1103/PhysRevE.88.052807)

[Tsonis2004] A.A. Tsonis and P.J. Roebber. "The architecture of the climate network". In *Physica A: Statistical Mechanics and its Applications*, vol. 333, p497-504 (2004) doi:10.1016/j.physa.2003.10.045 (http://dx.doi.org/10.1016/j.physa.2003.10.045)

[Tsonis2006] A.A. Tsonis, K.L. Swanson, P.J. Roebber. "What do networks have to do with climate?". In *Bull. Amer. Meteor. Soc.* vol. 87 p585-595 (2006) doi:10.1175/BAMS-87-5-585 (http://dx.doi.org/10.1175/BAMS-87-5-585)

[Gozolchiani2008] A. Gozolchiani, K. Yamasaki, O. Gazit, S. Havlin. "Pattern of climate network blinking links follows El Niño events". In *Europhysics Letters*, vol. 83 (no. 2), 28005 (2008) doi:10.1209/0295-5075/83/28005 (http://dx.doi.org/10.1209/0295-5075/83/28005)

[Tsonis2008b] A. A. Tsonis and K. L. Swanson. "Topology and Predictability of El Niño and La Niña Networks". In *Physical Review Letters* vol 100, 228502 (2008) doi:10.1103/PhysRevLett.100.228502 (http://dx.doi.org/10.1103/PhysRevLett.100.228502)

[Tsonis2008c] A. A. Tsonis, K. L. Swanson, G. Wang. "On the role of atmospheric teleconnections in climate". In *Journal of Climate* vol. 21, p2990-3001 (2008) doi:10.1175/2007JCLI1907.1 (http://dx.doi.org/10.1175/2007JCLI1907.1)

[Yamasaki2008] K. Yamasaki, A. Gozolchiani, S. Havlin. "Climate Networks around the Globe are Significantly Affected by El Niño". In *Physical Review Letters*, vol. 100, 228501 (2008) doi:10.1103/PhysRevLett.100.228501 (http://dx.doi.org/10.1103/PhysRevLett.100.228501)

[Donges2009c] J.F. Donges "Complex networks in the climate system". Diploma thesis, University of Potsdam (2009) Advisor: Prof. Dr. Dr. h.c. Juergen Kurths. URN: `urn:nbn:de:kobv:517-opus-49775`.

[Yamasaki2009] K. Yamasaki, A. Gozolchiani, S. Havlin. "Climate Networks Based on Phase Synchronization Analysis Track El-Niño". In *Progress Of Theoretical Physics Supplement*, vol. 179, p178-188 (2009) doi:10.1143/PTPS.179.178 (http://dx.doi.org/10.1143/PTPS.179.178)

[Schultz2014] P. Schultz "Stability Analysis of Power Grid Networks". *M.Sc. Thesis*, Humboldt-Universität zu Berlin (2014)

[Schultz2014a] P. Schultz, J. Heitzig, J. Kurths A Random Growth Model for Power Grids and Other Spatially Embedded Infrastructure Networks". In *Eur. Phys. J. Special Topics: Resilient Power Grids and Extreme Events* (2014)

[Pecora1998] L.M. Pecora and T.L. Carroll. "Master Stability Functions for Synchronized Coupled Systems". In *Physical Review Letters*, vol. 80, 2109 (1998) doi:10.1103/PhysRevLett.80.2109 (http://dx.doi.org/10.1103/PhysRevLett.80.2109)

[Schreiber2000] T. Schreiber and A. Schmitz. "Surrogate time series". In *Physica D* vol. 142 (no. 3-4), p346-382 (2000) doi:10.1016/S0167-2789(00)00043-9 (http://dx.doi.org/10.1016/S0167-2789(00)00043-9)

[Kraskov2004] A. Kraskov, H. Stögbauer, P. Grassberger. "Estimating mutual information". In *Physical Review E*, vol. 69(6), 066138 (2004) doi:10.1103/PhysRevE.69.066138 (http://dx.doi.org/10.1103/PhysRevE.69.066138)

[Kantz2006] H. Kantz and T. Schreiber. "Nonlinear Time Series Analysis". Cambridge University Press, Cambridge, 2nd edition (2006)

[Thiel2006] M. Thiel, M.C. Romano, J. Kurths, M. Rolfs, R. Kliegl. "Twin surrogates to test for complex synchronization". In *Europhysics Letters*, vol. 75, p535-541 (2006) doi:10.1209/epl/i2006-10147-0 (http://dx.doi.org/10.1209/epl/i2006-10147-0)

[Bergner2008] A. Bergner, R. Meucci, K. Al Naimee, M.C. Romano, M. Thiel, J. Kurths, and F. T. Arecchi. "Continuous wavelet transform in the analysis of burst synchronization in a coupled laser system". In *Physical Review E*, vol. 78, 016211 (2008) doi:10.1103/PhysRevE.78.016211 (http://dx.doi.org/10.1103/PhysRevE.78.016211)

[Pompe2011] B. Pompe, J. Runge. "Momentary information transfer as a coupling measure of time series". In *Physical Review E* vol. 83, 051122 (2011) doi:10.1103/PhysRevE.83.051122 (http://dx.doi.org/10.1103/PhysRevE.83.051122)

[Runge2012b] J. Runge, J. Heitzig, N. Marwan, J. Kurths. "Quantifying causal coupling strength: A lag-specific measure for multivariate time series related to transfer entropy". In *Physical Review E*, vol. 86(6), 1-15 (2012) doi:10.1103/PhysRevE.86.061121 (http://dx.doi.org/10.1103/PhysRevE.86.061121)

[Quiroga2002] R.Q. Quiroga, T. Kreuz, P. Grassberger. "Event synchronization: a simple and fast method to measure synchronicity and time delay patterns." In *Physical Review E*, vol. 66(4), 041904 (2002) doi:10.1103/PhysRevE.66.041904 (http://dx.doi.org/10.1103/PhysRevE.66.041904)

[Boers2014] N. Boers, B. Bookhagen, H.M.J. Barbosa, N. Marwan, J. Kurths. J.A. Marengo. "Prediction of extreme floods in the eastern Central Andes based on a complex networks approach". In *Nature communications*, vol. 5, 1–7 (2014) doi:10.1038/ncomms6199 (http://dx.doi.org/10.1038/ncomms6199)

[Marwan2007] N. Marwan, M.C. Romano, M. Thiel, J. Kurths. "Recurrence plots for the analysis of complex systems". In *Physics Reports*, vol. 438 (no. 5–6), p237-329 (2007) doi:10.1016/j.physrep.2006.11.001 (http://dx.doi.org/10.1016/j.physrep.2006.11.001)

[Marwan2009] N. Marwan, J.F. Donges, Y. Zou, R.V. Donner, J. Kurths. "Complex network approach for recurrence analysis of time series". In *Physics Letters A*, vol. 373 (no. 46), p4246-4254 (2009) doi:10.1016/j.physleta.2009.09.042 (http://dx.doi.org/10.1016/j.physleta.2009.09.042)

[Donner2010b] R.V. Donner, Y. Zou, J.F. Donges, N. Marwan, J. Kurths. "Recurrence networks – A novel paradigm for nonlinear time series analysis". In *New Journal of Physics*, vol. 12 (no. 3), 033205 (2010) doi:10.1088/1367-2630/12/3/033025 (http://dx.doi.org/10.1088/1367-2630/12/3/033025)

[Donner2010a] R.V. Donner, Y. Zou, J.F. Donges, N. Marwan, J. Kurths. "Ambiguities in recurrence-based complex network representations of time series". In *Physical Review E*, vol. 81 (no. 1), 015101(R) (2010) doi:10.1103/PhysRevE.81.015101 (http://dx.doi.org/10.1103/PhysRevE.81.015101)

[Zou2010] Y. Zou, R.V. Donner, J.F. Donges, N. Marwan, J. Kurths. "Identifying complex periodic windows in continuous-time dynamical systems using recurrence-based methods". In *Chaos*, vol. 20 (no. 4), 043130 (2010) doi:10.1063/1.3523304 (http://dx.doi.org/10.1063/1.3523304)

[Donner2011a] R.V. Donner, M. Small, J.F. Donges, N. Marwan, Y. Zou, R. Xiang, J. Kurths. "Recurrence-based time series analysis by means of complex network methods". In *International Journal of Bifurcation and Chaos*, vol. 21 (no. 4), p1019-1046 (2011) doi:10.1142/S0218127411029021 (http://dx.doi.org/10.1142/S0218127411029021)

[Donner2011b] R.V. Donner, J. Heitzig, J.F. Donges, Y. Zou, J. Kurths. "The geometry of chaotic dynamics – A complex network perspective". In *European Physical Journal B: Condensed Matter and Complex Systems*, vol. 84 (no. 4), p653-672 (2011) doi:10.1140/epjb/e2011-10899-1 (http://dx.doi.org/10.1140/epjb/e2011-10899-1)

[Donges2011b] J.F. Donges, R.V. Donner, K. Rehfeld, N. Marwan, M.H. Trauth, J. Kurths. "Identification of dynamical transitions in marine palaeoclimate records by recurrence network analysis". In *Nonlinear Processes in Geophysics*, vol. 18 (no. 5), p545-562 (2011) doi:10.5194/npg-18-545-2011 (http://dx.doi.org/10.5194/npg-18-545-2011)

[Donges2011c] J.F. Donges, R.V. Donner, M.H. Trauth, N. Marwan, H.J. Schellnhuber, J. Kurths. "Nonlinear detection of paleoclimate-variability transitions possibly related to human evolution". In *Proceedings of the National Academy of Sciences of the United States of America*, vol. 108 (no. 51), p20422-20427 (2011) doi:10.1073/pnas.1117052108 (http://dx.doi.org/10.1073/pnas.1117052108)

[Donges2012] J.F. Donges, J. Heitzig, R.V. Donner, J. Kurths. "Analytical framework for recurrence network analysis of time series". In *Physical Review E: Statistical, Nonlinear, and Soft Matter Physics*, vol. 85, 046105 (2012) doi:10.1103/PhysRevE.85.046105 (http://dx.doi.org/10.1103/PhysRevE.85.046105)

[Zou2012] Y. Zou, J. Heitzig, R.V. Donner, J.F. Donges, J.D. Farmer, R. Meucci, S. Euzzor, N. Marwan, J. Kurths. "Power-laws in recurrence networks from dynamical systems". In *Europhysics Letters*, vol. 98, 48001 (2012) doi:10.1209/0295-5075/98/48001 (http://dx.doi.org/10.1209/0295-5075/98/48001)

[Feldhoff2012] J.H. Feldhoff, R.V. Donner, J.F. Donges, N. Marwan, J. Kurths. "Geometric detection of coupling directions by means of inter-system recurrence networks". In *Physics Letters A*, vol. 376, 3504-3513 (2012), doi:10.1016/j.physleta.2012.10.008 (http://dx.doi.org/10.1016/j.physleta.2012.10.008)

[Feldhoff2013] J.H. Feldhoff, R.V. Donner, J.F. Donges, N. Marwan, J. Kurths. "Geometric signature of complex synchronisation scenarios". In *Europhysics Letters* vol. 102, 30007 (2013), doi:10.1209/0295-5075/102/30007 (http://dx.doi.org/10.1209/0295-5075/102/30007)

[Ngamga2007] E.J. Ngamga, A. Nandi, R. Ramaswamy, M.C. Romano, M. Thiel, J. Kurths. "Recurrence analysis of strange nonchaotic dynamics". In *Physical Review E*, vol. 75, 036222 (2007) doi:10.1103/PhysRevE.75.036222 (http://dx.doi.org/10.1103/PhysRevE.75.036222)

[Xu2008] X. Xu, J. Zhang, M. Small. "Superfamily phenomena and motifs of networks induced from time series". In *Proceedings of the National Academy of Sciences of the United States of America*, vol. 105 (no. 50) p19601-19605 (2008) doi:10.1073/pnas.0806082105 (http://dx.doi.org/10.1073/pnas.0806082105)

[Schinkel2009] S. Schinkel, N. Marwan, O. Dimigen, J. Kurths. "Confidence bounds of recurrence-based complexity measures". In *Physics Letters A*, vol. 373 (no. 26) p2245–2250 (2009) doi:10.1016/j.physleta.2009.04.045 (http://dx.doi.org/10.1016/j.physleta.2009.04.045)

[Lacasa2008] L. Lacasa, B. Luque, F. Ballesteros, J. Luque, J.C. Nuno. "From time series to complex networks: The visibility graph". In *Proceedings of the National Academy of Sciences of the United States of America*, vol. 105 (no. 13), p4972-4975 (2008) doi:10.1073/pnas.0709247105 (http://dx.doi.org/10.1073/pnas.0709247105)

[Donner2012] R.V. Donner and J.F. Donges. "Visibility graph analysis of geophysical time series: Potentials and possible pitfalls". In *Acta Geophysica*, vol. 60 p589-623 (2012) doi:10.2478/s11600-012-0032-x (http://dx.doi.org/10.2478/s11600-012-0032-x)

[Donges2013] J.F. Donges, R.V. Donner, J. Kurths. "Testing time series irreversibility using complex network methods". In *Europhysics Letters*, vol. 102.1, 10004 (2013) doi:10.1209/0295-5075/102/10004 (http://dx.doi.org/10.1209/0295-5075/102/10004)

PYTHON MODULE INDEX

INDEX

Symbols

A

G

I

M

N

P

Q

R

S

T