itng toolbox

Release 0.1.4

Abolfazl Ziaeemehr

Mar 10, 2020

CONTENTS

1 Overview: **1**
1.1 Requirements: 1
1.2 Contents: 1
1.3 Indices and tables 10

Python Module Index **11**

Index **13**

CHAPTER

ONE

OVERVIEW:

itng (iasbs theoretical neuroscience group toolbox) is a collection of tools for visualising and analysing the time series, measuring the spike synchronies and also graph analysis. This is a collaborative work with complex network and theoretical neuroscience group of Institute of advanced studied in basic science in Zanjan.

1.1 Requirements:

itng project depend on the following packages:

1. numpy
2. scipy
3. networkx
4. python-igraph
5. matplotlib
6. bctpy
7. cloud_sptheme (for documentation)

1.2 Contents:

1.2.1 itng toolbox tutorials

This is a short description about how to use itng toolbox.

Graph generator

networks.Generators.complete_graph()

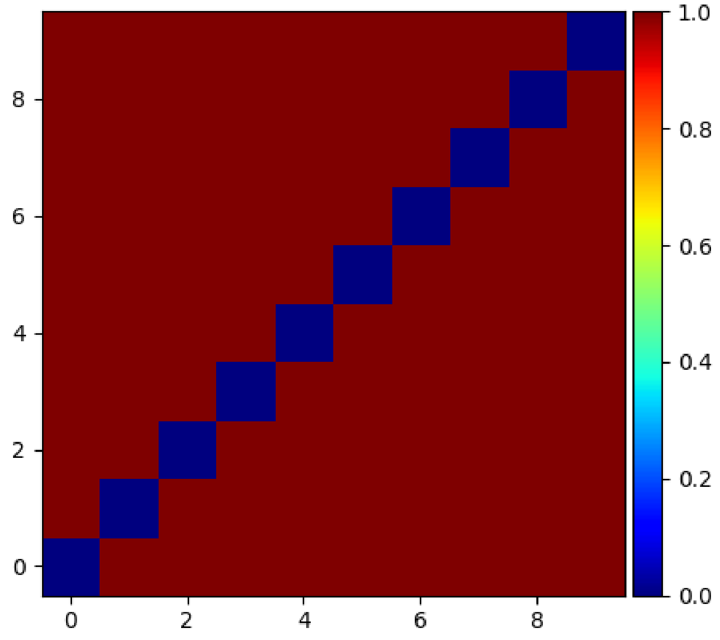
```
import itng
from itng import networks
import networkx as nx
from itng.drawing import Drawing

g = itng.networks.Generators()
G = g.complete_graph(10)
```

(continues on next page)

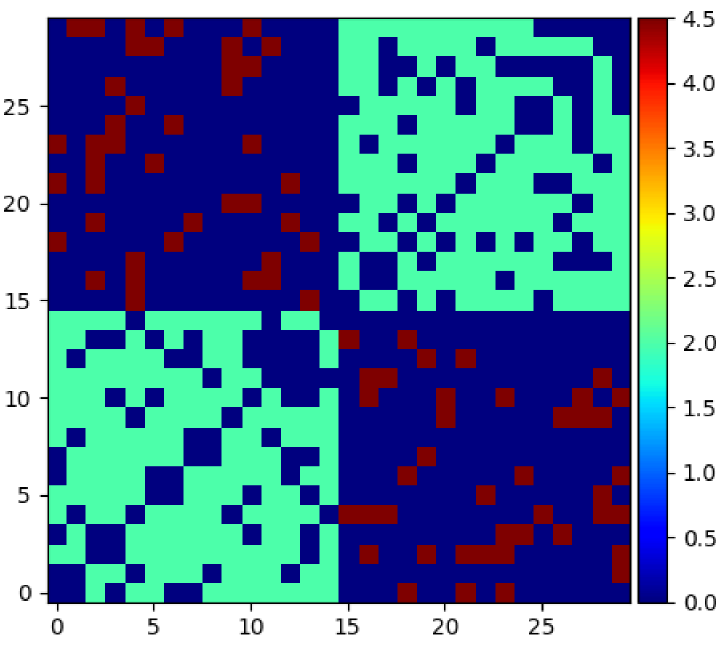
(continued from previous page)

```
Drawing.plot_adjacency_matrix(G,
                              fileName="complete_graph.png",
                              cmap="jet")
```



networks.Generators.modular_graph()

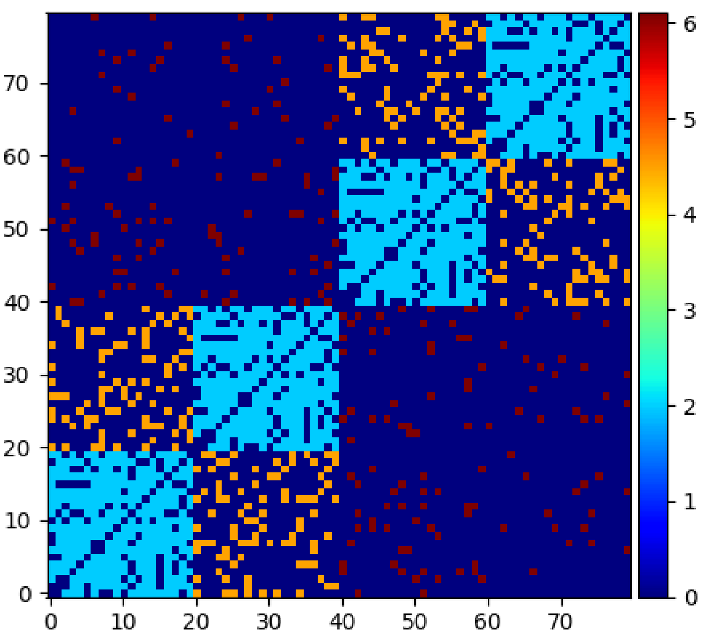
```
G = g.modular_graph(30, 0.7, 0.2, [15] * 2, 2.0, 4.5)
Drawing.plot_adjacency_matrix(G,fileName="modular.png", cmap="jet")
```



networks.Generators.hierarchical_modular_graph()

```
G = g.hierarchical_modular_graph(20, 3, 0.8, 0.25, 1.0, [2.0, 4.5, 6.1])
Drawing.plot_adjacency_matrix(G, fileName="hmn.png", cmap="jet")
print (nx.info(G))

# output
# Name:
# Type: Graph
# Number of nodes: 80
# Number of edges: 886
# Average degree:  22.1500
```



1.2.2 Auto Generated Documentation

`graphUtility.`**`binarize`**(*adj*, *threshold*)

binarize the given 2d numpy array

Parameters

- **`data`** – [2d numpy array] given array.
- **`threshold`** – [float] threshold value.

Returns [2d int numpy array] binarized array.

`graphUtility.`**`calculate_NMI`**(*self*, *comm1*, *comm2*, *method='nmi'*)

Compares two community structures

Parameters

- **`comm1`** – the first community structure as a membership list or as a Clustering object.
- **`comm2`** – the second community structure as a membership list or as a Clustering object.
- **`method`** – [string] defaults to ["nmi"] the measure to use. "vi" or "meila" means the variation of information metric of Meila (2003), "nmi" or "danon" means the normalized mutual information as defined by Danon et al (2005), "split-join" means the split-join distance of van Dongen (2000), "rand" means the Rand index of Rand (1971), "adjusted_rand" means the adjusted Rand index of Hubert and Arabie (1985).

Returns [float] the calculated measure.

Reference:

- Meila M: Comparing clusterings by the variation of information. In: Scholkopf B, Warmuth MK (eds). Learning Theory and Kernel Machines: 16th Annual Conference on Computational Learning Theory and 7th Kernel Workship, COLT/Kernel 2003, Washington, DC, USA. Lecture Notes in Computer Science, vol. 2777, Springer, 2003. ISBN: 978-3-540-40720-1.
- Danon L, Diaz-Guilera A, Duch J, Arenas A: Comparing community structure identification. J Stat Mech P09008, 2005.
- van Dongen D: Performance criteria for graph clustering and Markov cluster experiments. Technical Report INS-R0012, National Research Institute for Mathematics and Computer Science in the Netherlands, Amsterdam, May 2000.
- Rand WM: Objective criteria for the evaluation of clustering methods. J Am Stat Assoc 66(336):846-850, 1971.
- Hubert L and Arabie P: Comparing partitions. Journal of Classification 2:193-218, 1985.

`graphUtility.`**`check_edge_between_clusters`**(*e*, *nodes*)
check if given edge is between clusters.

Parameters

- **`e`** – [int, int] given edge
- **`nodes`** – nested list (list of list of int) including the index of nodes in each cluster.

Returns [bool] True if edge be between the clusters

`graphUtility.`**`clusters_info_modular`**(*G*, *clusters*, *verbosity=True*)
Returns the properties of clusters in a modular network.

Parameters

- **`G`** – undirected Graph of modular network
- **`clusters`** – list of list of int inclusing the index of nodes in each cluster.
- **`verbosity`** – [bool] if True print the results on the screen.

Returns [int, int] total number of edges and number of edges between clusters.

`graphUtility.`**`extract_attributes`**(*G*, *attr=None*)
extract the matrix of given atributes from graph

Parameters

- **`G`** – [networkx graph object]
- **`attr`** – [string] given attribute

Returns [ndarray] matrix of given attribute

`graphUtility.`**`find_indices_with_distance`**(*DiGraph*, *distance*, *source*)
return index of nodes in given distance from source node.

Parameters

- **`DiGraph`** – networkx directed graph.
- **`distance`** – [int] given distance from source node.
- **`source`** – index of given nodes as source, distance of other nodes calculated from this node.

Returns index of nodes in given distance.

`graphUtility.`**`find_leader_node`**(*self*, *adj*, *directed=True*)

find the leader node in adj matrix of a directed acyclic graph leader node is the node with in degree of zero.

Parameters **`adj`** –

Directed [bool] if True consider a directed graph

Returns index of leader node

`graphUtility.`**`lambdan_over_lambda2`**(*G*)

calculate the fraction of lambda_n over lambda_2. lambda_i s are the eigen values of laplacian matrix

`graphUtility.`**`multilevel`**(*adj_matrix*, *directed=False*, *return_levels=False*)

find communities of given weighted adjacency network

Parameters

- **`adjMatrix`** – [2d array] adjacency matrix
- **`directed`** – [bool(default=False)] choose directed or undirected network
- **`return_levels`** – if True, the communities at each level are returned in a list. If False, only the community structure with the best modularity is returned.

Returns a list of VertexClustering objects, one corresponding to each level (if return_levels is True), or a VertexClustering corresponding to the best modularity.

`graphUtility.`**`print_adj_matrix`**(*G*, *file_name=None*, *fmt='%d'*, *binary_file=False*)

print the adjacency matrix of given graph

Parameters

- **`G`** – networkx graph
- **`file_name`** – optional(default=None), if given save to file
- **`fmt`** – optional, format of numbeers in weighted adjacency matrix, "%d" for integer and e.g. "%10.3f", "%g", … for float numbers.
- **`binary_file`** – [bool] if True, save npz file in binary format.

`graphUtility.`**`topological_sort`**(*self*, *adj_matrix*)

return topologicaly sorted network of given directed acyclic graph.

A topological sort is a nonunique permutation of the nodes such that an edge from u to v implies that u appears before v in the topological sort order.

Parameters **`adj_atrix`** – [ndarray] given adjacency matrix of directed acyclic graph (DAG).

Returns [ndarray] sorted adjacency matrix

`statistics.`**`cv`**(*self*, *ts*, *gids*)

Compute the coefficient of variation.

!todo

`statistics.`**`isi`**(*self*, *spiketimes*, *gids*)

calculate interspike interval of given spike train

Parameters

- **`spikeTimes`** – time of spikes, 1 dimensional array, list or tuple
- **`gids`** – global ID of neurons

Returns inter spike interval 1d numpy array `statistics.`**`mean_firing_rate`**(*self*, *spiketrain*, *t_start=None*, *t_stop=None*)

Return the firing rate of the spike train.

Parameters

- **`spiketrain`** – [np.ndarray] the spike times
- **`t_start`** – [float] The start time to use for the interval
- **`t_stop`** – [float] The end time to use for the interval

Returns The firing rate of the spiketrain

#!todo

`synchrony.`**`spike_synchrony`**(*self*, *ts*, *gids*, *threshold_num_spikes=10*)

calculate spike synchrony.

Parameters

- **`ts`** – time of spikes 1 dimensional array, list or tuple.
- **`gids`** – global ID of neurons.
- **`threshold_num_spikes`** – minimum number of spikes required for calculation of measure.

Returns [float] spike synchrony in [0, 1].

Reference

- Tiesinga, P. H. & Sejnowski, T. J. Rapid temporal modulation of synchrony by competition in cortical interneuron networks. Neural computation 16, 251–275 (2004).

`synchrony.`**`voltage_synchrony`**(*self*, *voltages*)

calculate voltage synchrony.

Parameters **`voltages`** – [ndarray, nested list or nested tuple (number of nodes by number of time steps)] votages of n nodes.

Returns [float] voltage synchrony in [0, 1]

Reference:

- Lim, W. & Kim, S. Y. Coupling-induced spiking coherence in coupled subthreshold neurons. Int. J. Mod. Phys. B 23, 2149–2157 (2009)

`randomSequence.`**`power_law`**(*self*, *N*, *e*, *xmin*, *xmax*)

generate a power law distribution of integers from uniform distribution

Parameters

- **`N`** – [int] number of data in powerlaw distribution (pwd).
- **`e`** – [int, float] exponent of the pwd.
- **`xmin`** – [int] min value in pwd.
- **`xmax`** – [int] max value in pwd.

Returns [numpy array of int] the power law distribution

becuse the numbers will use as degree of nodes, the sum of degrees should be an even number.

Reference:

- http://mathworld.wolfram.com/RandomNumber.html `signal_processing.`**`fft_1d_real`**(*signal*, *fs*)

fft from 1 dimensional real signal

Parameters

- **`signal`** – [np.array] real signal
- **`fs`** – [float] frequency sampling in Hz

Returns [np.array, np.array] frequency, normalized amplitude

- example:

```
>>> B = 30.0  # max freqeuency to be measured.
>>> fs = 2 * B
>>> delta_f = 0.01
>>> N = int(fs / delta_f)
>>> T = N / fs
>>> t = np.linspace(0, T, N)
>>> nu0, nu1 = 1.5, 22.1
>>> amp0, amp1, ampNoise = 3.0, 1.0, 1.0
>>> signal = amp0 * np.sin(2 * np.pi * t * nu0) + amp1 * np.sin(2 * np.pi * t *␣
↪nu1) +
        ampNoise * np.random.randn(*np.shape(t))
>>> freq, amp = fft_1d_real(signal, fs)
>>> pl.plot(freq, amp, lw=2)
>>> pl.show()
```

`signal_processing.`**`filter_butter_bandpass`**(*signal*, *fs*, *lowcut*, *highcut*, *order=5*)

Butterworth filtering function

Parameters

- **`signal`** – [np.array] Time series to be filtered
- **`fs`** – [float] Frequency sampling in Hz
- **`lowcut`** – [float] Lower value for frequency to be passed in Hz
- **`highcut`** – [float] Higher value for frequency to be passed in Hz
- **`order`** – [int] The order of the filter.

Returns [np.array] filtered frequncy

`signal_processing.`**`fwhm2sigma`**(*fwhm*)

Convert a FWHM in a Gaussian kernel to a sigma value

The FWHM is the width of the kernel, at half of the maximum of the height of the Gaussian. Thus, for the standard Gaussian above, the maximum height is ~0.4. The width of the kernel at 0.2 (on the Y axis) is the FWHM. As x = -1.175 and 1.175 when y = 0.2, the FWHM is roughly 2.35.

Parameters **`fwhm`** – [float] fwhm in gaussian kernel

Returns sigma in gaussian kernel

see also: https://matthew-brett.github.io/teaching/smoothing_intro.html

`signal_processing.`**`kuramoto_correlation_matrix`**(*x*)

claculate the Kuramoto correlation

Parameters **`x`** – [np.array] array of phases

Returns [np.ndarray] calculated correlation matrix signal_processing.**pearson_correlation_matrix**(*data*, *axis=1*)

calculate the pearson correlation matrix

Parameters

- **data** – [np.dnarray (n by num_time_step)] Time series of n nodes, each have num_time_step elements
- **axis** – [int] optional, if 1, each row is considered as a time seri. if 0, each column is a time seri.

Returns [np.ndarray] correlation matrix

- example

```
>>> np.random.seed(1)
>>> data = np.random.rand(3, 5)
>>> p = pearson_correlation_matrix(data, axis=1)
>>> print (p)
>>> #[[ 0.          -0.63668568  0.6188195 ]
>>> # [-0.63668568  0.          -0.37240144]
>>> # [ 0.6188195   -0.37240144  0.         ]]
```

signal_processing.**sigma2fwhm**(*sigma*)

Convert a sigma in a Gaussian kernel to a FWHM value

The FWHM is the width of the kernel, at half of the maximum of the height of the Gaussian.

Sigma sigma in gaussian kernel

Returns fwhm in gaussian kernel

```
>>> sigma2fwhm(1)
2.3548200450309493
```

see also https://matthew-brett.github.io/teaching/smoothing_intro.html

signal_processing.**smooth_gaussian**(*x_values*, *y_values*, *sigma*)

smoothing signal by gaussian kernel.

Parameters

- **x_values** – [np.array] x values of given signal
- **y_values** – [np.array] y values if given signal
- **sigma** – [float] sigma in gaussian kernel

Returns [np.array] smoothed signal

Gaussian distribution in 1D has the form : G(x) = 1/(sqrt(2 * pi) sigma) exp(-x^2/(2*sigma^2))

```
>>> FWHM = 4
>>> n_points = 60
>>> x_vals = np.arange(n_points)
>>> y_vals = np.random.normal(size=n_points)
>>> sigma = fwhm2sigma(FWHM)
>>> smoothed_g = smooth_gaussian(x_vals, y_vals, sigma)
>>> plt.plot(x_vals, y_vals, lw=2, label='original')
>>> plt.plot(x_vals, smoothed_g, lw=3, c='r', label='gaussian')
```

1.3 Indices and tables

- genindex
- modindex
- search

PYTHON MODULE INDEX

g

graphUtility, 4

r

randomSequence, 7

s

signal_processing, 7
statistics, 6
synchrony, 7

INDEX

B

binarize() (*in module graphUtility*), 4

C

calculate_NMI() (*in module graphUtility*), 4
check_edge_between_clusters() (*in module graphUtility*), 5
clusters_info_modular() (*in module graphUtility*), 5
cv() (*in module statistics*), 6

E

extract_attributes() (*in module graphUtility*), 5

F

fft_1d_real() (*in module signal_processing*), 7
filter_butter_bandpass() (*in module signal_processing*), 8
find_indices_with_distance() (*in module graphUtility*), 5
find_leader_node() (*in module graphUtility*), 5
fwhm2sigma() (*in module signal_processing*), 8

G

graphUtility (*module*), 4

I

isi() (*in module statistics*), 6

K

kuramoto_correlation_matrix() (*in module signal_processing*), 8

L

lambdan_over_lambda2() (*in module graphUtility*), 6

M

mean_firing_rate() (*in module statistics*), 6
multilevel() (*in module graphUtility*), 6

P

pearson_correlation_matrix() (*in module signal_processing*), 8
power_law() (*in module randomSequence*), 7
print_adj_matrix() (*in module graphUtility*), 6

R

randomSequence (*module*), 7

S

sigma2fwhm() (*in module signal_processing*), 9
signal_processing (*module*), 7
smooth_gaussian() (*in module signal_processing*), 9
spike_synchrony() (*in module synchrony*), 7
statistics (*module*), 6
synchrony (*module*), 7

T

topological_sort() (*in module graphUtility*), 6

V

voltage_synchrony() (*in module synchrony*), 7