connectivipy Documentation

Release 0.36a

Dominik Krzeminski

Sep 22, 2021

Contents

Python connectivity module. It is a part of GSOC 2015 project.

You can find it and download from GitHub.

Connectivity estimation is one of the most important problem in EEG/MEG studies. Many estimators work correctly in different applications, so it's convenient to have them all in one tool. ConnectiviPy is light and extendable python module open for many data formats. In opposite to other packages it allows you to work with all types of data, not only biomedical. Calculations base on numpy and scipy what provides good efficiency.

Tested under Python >=2.7 and >=3.3.

CHAPTER 1

Contents:

1.1 Data

Data module - main class governing your data and wrapper for all other **ConnectiviPy** functions.

class `connectivipy.data.`**`Data`**(*data*, *fs=1.0*, *chan_names=[]*, *data_info=''*)

Class governing the communication between data array and connectivity estimators.

Args:

data [numpy.array or str]

- **array with data (kXNxR, k - channels nr, N - data points,** R - nr of trials)
- str - path to file with appropieate format

***fs* = 1: int** sampling frequency

***chan_names* = []: list** names of channels

***data_info* = '': string** other information about the data

`select_channels`(*channels=None*)

Selecting channels to plot or further analysis.

Args:

channels [list(int)] List of channel indices. If None all channels are taken into account.

`filter`(*b*, *a*)

Filter each channel of data using forward-backward filter *filtfilt* from *scipy.signal*.

Args:

b, a [np.array] Numerator *b* / denominator *a* polynomials of the IIR filter.

`resample`(*fs_new*)

Signal resampling to new sampling frequency *new_fs* using *resample* function from *scipy.signal* (basing on Fourier method).

Args:

fs_new [int] new sampling frequency

fit_mvar (*p=None*, *method='yw'*)

Fitting MVAR coefficients.

Args:

***p* = None** [int] estimation order, default None

***method* = 'yw'** [str {'yw', 'ns', 'vm'}] method of MVAR parameters estimation all avaiable methods you can find in *fitting_algorithms*

conn (*method*, ***params*)

Estimate connectivity pattern.

Args:

***p* = None** [int] estimation order, default None

method [str] method of connectivity estimation all avaiable methods you can find in *conn_estim_dc*

short_time_conn (*method*, *nfft=None*, *no=None*, ***params*)

Short-time connectivity.

Args:

***method* = 'yw'** [str {'yw', 'ns', 'vm'}] method of estimation all avaiable methods you can find in *fitting_algorithms*

***nfft* = None** [int] number of data points in window; if None, it is signal length N/5.

***no* = None** [int] number of data points in overlap; if None, it is signal length N/10.

params other parameters for specific estimator

significance (*Nrep=100*, *alpha=0.05*, *verbose=True*, ***params*)

Statistical significance values of connectivity estimation method.

Args:

***Nrep* = 100** [int] number of resamples

***alpha* = 0.05** [float] type I error rate (significance level)

***verbose* = True** [bool] if True it prints dot on every realization

Returns:

***signi*: numpy.array** matrix in shape of (k, k) with values for each pair of channels

short_time_significance (*Nrep=100*, *alpha=0.05*, *nfft=None*, *no=None*, *verbose=True*, ***params*)

Statistical significance values of short-time version of connectivity estimation method.

Args:

***Nrep* = 100** [int] number of resamples

***alpha* = 0.05** [float] type I error rate (significance level)

***nfft* = None** [int] number of data points in window; if None, it is taken from `Data.short_time_conn()` method.

***no* = None** [int] number of data points in overlap; if None, it is taken from *short_time_conn* method.

***verbose* = True** [bool] if True it prints dot on every realization

Returns:

***signi*: numpy.array** matrix in shape of (k, k) with values for each pair of channels

plot_data (*trial=0*, *show=True*)

Plot data in a subplot for each channel.

Args:

***trial* = 0** [int] if there is multichannel data it should be a number of trial you want to plot.

***show* = True** [boolean] show the plot or not

plot_conn (*name=''*, *ylim=None*, *xlim=None*, *signi=True*, *show=True*)

Plot connectivity estimation results.

Args:

***name* = ''** [str] title of the plot

***ylim* = None** [list] range of y-axis values shown, e.g. [0,1] *None* means that default values of given estimator are taken into account

***xlim* = None** [list [from (int), to (int)]] range of y-axis values shown, if None it is from 0 to Nyquist frequency

***signi* = True** [boolean] if significance levels are calculated they are shown in the plot

***show* = True** [boolean] show the plot or not

plot_short_time_conn (*name=''*, *signi=True*, *percmax=1.0*, *show=True*)

Plot short-time version of estimation results.

Args:

***name* = ''** [str] title of the plot

***signi* = True** [boolean] reset irrelevant values; it works only after short time significance calculation using *short_time_significance*

***percmax* = 1.** [float (0,1)] percent of maximal value which is maximum on the color map

***show* = True** [boolean] show the plot or not

export_trans3d (*mod=0*, *filename='conntrans3d.dat'*, *freq_band=[]*)

Export connectivity data to trans3D data file in order to make 3D arrow plots. Args:

***mod* = 0** [int] 0 - `Data.conn()` results 1 - `Data.short_time_conn()` results

***filename* = 'conn_trnas3d.dat'** [str] title of the plot

***freq_band* = []** [list] frequency range [from_value, to_value] in Hz.

fill_nans (*values*, *borders*)

Fill nans where *values* < *borders* (independent of frequency).

Args:

values [numpy.array] array of shape (time, freqs, channels, channels) to fill nans

borders [numpy.array] array of shape (time, channels, channels) with limes values

Returns:

values_nans [numpy.array] array of shape (time, freq, channels, channels) with nans where values were less than appropieate value from *borders*

mvar_coefficients
Returns mvar coefficients if calculated

mvarcoef
Returns mvar coefficients if calculated

1.1.1 Data loading

Additonal function which enable other data formats loading

`connectivipy.load.loaders.`**`signalml_loader`**(*file_name*)
It returns data and dictionary from SignalML files.

Args:

file_name [str] must be the same for .xml and .raw files.

Returns:

***data*: np.array** eeg data from raw file

xmlinfo [dict] dcitionary with keys: samplingFrequency, channelCount, firstSampleTimestamp, channelNames, calibrationCoef which means the same as in SML file

`connectivipy.load.loaders.`**`give_xml_info`**(*path*)
It returns dictionary from SignalML file.

Args:

path [str] SML file eg. 'test.xml'

Returns:

xml_data [dict] dcitionary with keys: samplingFrequency, channelCount, firstSampleTimestamp, channelNames, calibrationCoef which means the same as in SML file

1.1.2 Additional functions

Plot tools which are separate from Data class

`connectivipy.`**`plot_conn`**(*values*, *name=''*, *fs=1*, *ylim=None*, *xlim=None*, *show=True*)
Plot connectivity estimation results. Allows to plot your results without using *Data* class.

Args:

values [numpy.array] connectivity estimation values in shape (fq, k, k) where fq - frequency, k - number of channels

***name* = ''** [str] title of the plot

***fs* = 1** [int] sampling frequency

***ylim* = None** [list] range of y-axis values shown, e.g. [0,1] *None* means that default values of given estimator are taken into account

***xlim* = None** [list [from (int), to (int)]] range of y-axis values shown, if None it is from 0 to Nyquist frequency

***show* = True** [boolean] show the plot or not

1.2 Connectvity

Connectivity methods classes.

`connectivipy.conn.`**`spectrum`**(*acoef*, *vcoef*, *fs=1*, *resolution=100*)

Generating data point from matrix *A* with MVAR coefficients. Args:

acoef [numpy.array] array of shape (k, k, p) where *k* is number of channels and *p* is a model order.

vcoef [numpy.array] prediction error matrix (k, k)

***fs* = 1** [int] sampling rate

***resolution* = 100** [int] number of spectrum data points

Returns:

A_z [numpy.array] z-transformed A(f) complex matrix in shape (*resolution*, k, k)

H_z [numpy.array] inversion of *A_z*

S_z [numpy.array] spectrum matrix (*resolution*, k, k)

References: .. [1] K. J. Blinowska, R. Kus, M. Kaminski (2004) "Granger causality

and information flow in multivariate processes" Physical Review E 70, 050902.

`connectivipy.conn.`**`spectrum`**(*acoef*, *vcoef*, *fs=1*, *resolution=100*)

Generating data point from matrix *A* with MVAR coefficients. Args:

acoef [numpy.array] array of shape (k, k, p) where *k* is number of channels and *p* is a model order.

vcoef [numpy.array] prediction error matrix (k, k)

***fs* = 1** [int] sampling rate

***resolution* = 100** [int] number of spectrum data points

Returns:

A_z [numpy.array] z-transformed A(f) complex matrix in shape (*resolution*, k, k)

H_z [numpy.array] inversion of *A_z*

S_z [numpy.array] spectrum matrix (*resolution*, k, k)

References: .. [1] K. J. Blinowska, R. Kus, M. Kaminski (2004) "Granger causality

and information flow in multivariate processes" Physical Review E 70, 050902.

`connectivipy.conn.`**`spectrum_inst`**(*acoef*, *vcoef*, *fs=1*, *resolution=100*)

Generating data point from matrix *A* with MVAR coefficients taking into account zero-lag effects. Args:

acoef [numpy.array] array of shape (k, k, p+1) where *k* is number of channels and *p* is a model order. acoef[0] - is (k, k) matrix for zero lag, acoef[1] for one data point lag and so on.

vcoef [numpy.array] prediction error matrix (k, k)

***fs* = 1** [int] sampling rate

***resolution* = 100** [int] number of spectrum data points

Returns:

A_z [numpy.array] z-transformed A(f) complex matrix in shape (*resolution*, k, k)

H_z [numpy.array] inversion of *A_z*

S_z [numpy.array] spectrum matrix (*resolution*, k, k)

References: .. [1] Erla S. et all, Multivariate Autoregressive Model with

Instantaneous Effects to Improve Brain Connectivity Estimation, Int. J. Bioelectromagn. 11, 74–79 (2009).

`connectivipy.conn.`**`spectrum_inst`**(*acoef*, *vcoef*, *fs=1*, *resolution=100*)

Generating data point from matrix *A* with MVAR coefficients taking into account zero-lag effects. Args:

acoef [numpy.array] array of shape (k, k, p+1) where *k* is number of channels and *p* is a model order. acoef[0] - is (k, k) matrix for zero lag, acoef[1] for one data point lag and so on.

vcoef [numpy.array] prediction error matrix (k, k)

***fs* = 1** [int] sampling rate

***resolution* = 100** [int] number of spectrum data points

Returns:

A_z [numpy.array] z-transformed A(f) complex matrix in shape (*resolution*, k, k)

H_z [numpy.array] inversion of *A_z*

S_z [numpy.array] spectrum matrix (*resolution*, k, k)

References: .. [1] Erla S. et all, Multivariate Autoregressive Model with

Instantaneous Effects to Improve Brain Connectivity Estimation, Int. J. Bioelectromagn. 11, 74–79 (2009).

class `connectivipy.conn.`**`Connect`**

Abstract class governing calculation of various connectivity estimators with concrete methods: *short_time*, *significance* and abstract *calculate*.

`calculate`()

Abstract method to calculate values of estimators from specific parameters

`short_time`(*data*, *nfft=None*, *no=None*, ***params*)

Short-tme version of estimator, where data is windowed into parts of length *nfft* and overlap *no*. *params* catch additional parameters specific for estimator. Args:

data [numpy.array] data matrix

***nfft* = None** [int] window length (if None it's N/5)

***no* = None** [int] overlap length (if None it's N/10)

***params* :** additional parameters specific for chosen estimator

Returns:

stvalues [numpy.array] short time values (time points, frequency, k, k), where k is number of channels

`short_time_significance`(*data*, *Nrep=10*, *alpha=0.05*, *nfft=None*, *no=None*, *verbose=True*, ***params*)

Significance of short-tme versions of estimators. It base on bootstrap `Connect.bootstrap()` for multitrial case and surrogate data `Connect.surrogate()` for one trial. Args:

data [numpy.array] data matrix

***Nrep* = 100** [int] number of resamples

***alpha* = 0.05** [float] type I error rate (significance level)

***nfft* = None** [int] window length (if None it's N/5)

***no* = None** [int] overlap length (if None it's N/10)

***verbose* = True** [bool] if True it prints dot on every realization, if False it's quiet.

***params* :** additional parameters specific for chosen estimator

Returns:

signi_st [numpy.array] short time significance values in shape of - (tp, k, k) for one sided estimator - (tp, 2, k, k) for two sided where k is number of channels and tp number of time points

significance(*data*, *Nrep=10*, *alpha=0.05*, *verbose=True*, ***params*)

Significance of connectivity estimators. It base on bootstrap `Connect.bootstrap()` for multitrial case and surrogate data `Connect.surrogate()` for one trial. Args:

data [numpy.array] data matrix

***Nrep* = 100** [int] number of resamples

***alpha* = 0.05** [float] type I error rate (significance level)

***verbose* = True** [bool] if True it prints dot on every realization, if False it's quiet.

***params* :** additional parameters specific for chosen estimator

Returns:

signific [numpy.array] significance values, check `Connect.levels()`

levels(*signi*, *alpha*, *k*)

Levels of significance Args:

signi [numpy.array] bootstraped values of each channel

alpha [float] type I error rate (significance level) - from 0 to 1 - (1-*alpha*) for onesided estimators (e.g. class:*DTF*) - *alpha* and (1-*alpha*) for twosided (e.g. class:*PSI*)

k [int] number of channels

Returns:

ficance [numpy.array] maximal value throughout frequency of score at percentile at level 1-*alpha* - (k, k) for one sided estimator - (2, k, k) for two sided

bootstrap(*data*, *Nrep=100*, *alpha=0.05*, *verbose=True*, ***params*)

Bootstrap - random sampling with replacement of trials. Args:

data [numpy.array] multichannel data matrix

***Nrep* = 100** [int] number of resamples

***alpha* = 0.05** [float] type I error rate (significance level)

***verbose* = True** [bool] if True it prints dot on every realization, if False it's quiet.

***params* :** additional parameters specific for chosen estimator

Returns:

levelsigni [numpy.array] significance values, check `Connect.levels()`

surrogate(*data*, *Nrep=100*, *alpha=0.05*, *verbose=True*, ***params*)

Surrogate data testing. Mixing data points in each channel. Significance level in calculated over all *Nrep* surrogate sets. Args:

data [numpy.array] multichannel data matrix

***Nrep* = 100** [int] number of resamples

***alpha* = 0.05** [float] type I error rate (significance level)

***verbose* = True** [bool] if True it prints dot on every realization, if False it's quiet.

***params* :** additional parameters specific for chosen estimator

Returns:

levelsigni [numpy.array] significance values, check `Connect.levels()`

class connectivipy.conn.**ConnectAR**

Inherits from *Connect* class and governs calculation of various connectivity estimators basing on MVAR model methods. It overloads *short_time*, *significance* methods but *calculate* remains abstract.

short_time(*data*, *nfft=None*, *no=None*, *mvarmethod='yw'*, *order=None*, *resol=None*, *fs=1*)

It overloads `ConnectAR` method `Connect.short_time()`. Short-tme version of estimator, where data is windowed into parts of length *nfft* and overlap *no*. *params* catch additional parameters specific for estimator. Args:

data [numpy.array] data matrix

***nfft* = None** [int] window length (if None it's N/5)

***no* = None** [int] overlap length (if None it's N/10)

***mvarmethod* = 'yw' :**

MVAR parameters estimation method

all avaiable methods you can find in *fitting_algorithms*

***order* = None:** MVAR model order; it None, it is set automatically basing on default criterion.

***resol* = None:** frequency resolution; if None, it is 100.

***fs* = 1 :** sampling frequency

Returns:

stvalues [numpy.array] short time values (time points, frequency, k, k), where k is number of channels

short_time_significance(*data*, *Nrep=100*, *alpha=0.05*, *method='yw'*, *order=None*, *fs=1*, *resolution=None*, *nfft=None*, *no=None*, *verbose=True*, ***params*)

Significance of short-tme versions of estimators. It base on bootstrap `ConnectAR.bootstrap()` for multitrial case and surrogate data `ConnectAR.surrogate()` for one trial. Args:

data [numpy.array] data matrix

***Nrep* = 100** [int] number of resamples

***alpha* = 0.05** [float] type I error rate (significance level)

***method* = 'yw': str** method of MVAR parameters estimation all avaiable methods you can find in *fitting_algorithms*

***order* = None** [int] MVAR model order, if None, it's chosen using default criterion

***fs* = 1** [int] sampling frequency

***resolution* = None** [int] resolution (if None, it's 100 points)

***nfft* = None** [int] window length (if None it's N/5)

***no* = None** [int] overlap length (if None it's N/10)

***verbose* = True** [bool] if True it prints dot on every realization, if False it's quiet.

***params* :** additional parameters specific for chosen estimator

Returns:

signi_st [numpy.array] short time significance values in shape of - (tp, k, k) for one sided estimator - (tp, 2, k, k) for two sided where k is number of channels and tp number of time points

significance(*data*, *method*, *order=None*, *resolution=None*, *Nrep=10*, *alpha=0.05*, *verbose=True*, ***params*)

Significance of connectivity estimators. It base on bootstrap `ConnectAR.bootstrap()` for multitrial case and surrogate data `ConnectAR.surrogate()` for one trial. Args:

data [numpy.array] data matrix

***method* = 'yw': str** method of MVAR parametersestimation all avaiable methods you can find in *fitting_algorithms*

***order* = None** [int] MVAR model order, if None, it's chosen using default criterion

***Nrep* = 100** [int] number of resamples

***alpha* = 0.05** [float] type I error rate (significance level)

***resolution* = None** [int] resolution (if None, it's 100 points)

***verbose* = True** [bool] if True it prints dot on every realization, if False it's quiet.

***params* :** additional parameters specific for chosen estimator

Returns:

signi_st [numpy.array] significance values, check `Connect.levels()`

bootstrap(*data*, *method*, *order=None*, *Nrep=10*, *alpha=0.05*, *fs=1*, *verbose=True*, ***params*)

Bootstrap - random sampling with replacement of trials for *ConnectAR*. Args:

data [numpy.array] multichannel data matrix

method [str] method of MVAR parametersestimation all avaiable methods you can find in *fitting_algorithms*

***Nrep* = 100** [int] number of resamples

***alpha* = 0.05** [float] type I error rate (significance level)

***order* = None** [int] MVAR model order, if None, it's chosen using default criterion

***verbose* = True** [bool] if True it prints dot on every realization, if False it's quiet.

***params* :** additional parameters specific for chosen estimator

Returns:

levelsigni [numpy.array] significance values, check `Connect.levels()`

surrogate(*data*, *method*, *Nrep=10*, *alpha=0.05*, *order=None*, *fs=1*, *verbose=True*, ***params*)

Surrogate data testing for *ConnectAR* . Mixing data points in each channel. Significance level in calculated over all *Nrep* surrogate sets. Args:

data [numpy.array] multichannel data matrix

method [str] method of MVAR parameters estimation all avaiable methods you can find in *fitting_algorithms*

***Nrep* = 100** [int] number of resamples

***alpha* = 0.05** [float] type I error rate (significance level)

***order* = None** [int] MVAR model order, if None, it's chosen using default criterion

***verbose* = True** [bool] if True it prints dot on every realization, if False it's quiet.

***params* :** additional parameters specific for chosen estimator

Returns:

levelsigni [numpy.array] significance values, check `Connect.levels()`

connectivipy.conn.**dtf_fun**(*Acoef*, *Vcoef*, *fs*, *resolution*, *generalized=False*)

Directed Transfer Function estimation from MVAR parameters. Args:

Acoef [numpy.array] array of shape (k, k, p) where *k* is number of channels and *p* is a model order.

Vcoef [numpy.array] prediction error matrix (k, k)

***fs* = 1** [int] sampling rate

***resolution* = 100** [int] number of spectrum data points

***generalized* = False** [bool] generalized version or not

Returns:

DTF [numpy.array] matrix with estimation results (*resolution*, k, k)

References: .. [1] M. Kaminski, K.J. Blinowska. A new method of the description

of the information flow. Biol.Cybern. 65:203-210, (1991).

connectivipy.conn.**pdc_fun**(*Acoef*, *Vcoef*, *fs*, *resolution*, *generalized=False*)

Partial Directed Coherence estimation from MVAR parameters. Args:

Acoef [numpy.array] array of shape (k, k, p) where *k* is number of channels and *p* is a model order.

Vcoef [numpy.array] prediction error matrix (k, k)

***fs* = 1** [int] sampling rate

***resolution* = 100** [int] number of spectrum data points

***generalized* = False** [bool] generalized version or not

Returns:

PDC [numpy.array] matrix with estimation results (*resolution*, k, k)

References: .. [1] Sameshima, K., Baccala, L. A., Partial directed

coherence: a new concept in neural structure determination., 2001, Biol. Cybern. 84, 463–474.

class connectivipy.conn.**PartialCoh**

PartialCoh - class inherits from `ConnectAR` and overloads `Connect.calculate()` method.

calculate(*Acoef=None*, *Vcoef=None*, *fs=None*, *resolution=None*)

Partial Coherence estimation from MVAR parameters. Args:

Acoef [numpy.array] array of shape (k, k, p) where *k* is number of channels and *p* is a model order.

Vcoef [numpy.array] prediction error matrix (k, k)

***fs* = 1** [int] sampling rate

***resolution* = 100** [int] number of spectrum data points

***generalized* = False** [bool] generalized version or not

Returns:

PC [numpy.array] matrix with estimation results (*resolution*, k, k)

References: .. [1] G. M. Jenkins, D. G. Watts. Spectral Analysis and its

Applications. Holden-Day, USA, 1969

class connectivipy.conn.**PDC**

PDC - class inherits from `ConnectAR` and overloads `Connect.calculate()` method.

calculate(*Acoef=None*, *Vcoef=None*, *fs=None*, *resolution=100*)

More in `pdc_fun()`.

class connectivipy.conn.**gPDC**

gPDC - class inherits from `ConnectAR` and overloads `Connect.calculate()` method.

calculate(*Acoef=None*, *Vcoef=None*, *fs=None*, *resolution=100*)

More in `pdc_fun()`

class connectivipy.conn.**DTF**

DTF - class inherits from `ConnectAR` and overloads `Connect.calculate()` method.

calculate(*Acoef=None*, *Vcoef=None*, *fs=None*, *resolution=100*)

More in `dtf_fun()`.

class connectivipy.conn.**gDTF**

gDTF - class inherits from `ConnectAR` and overloads `Connect.calculate()` method.

calculate(*Acoef=None*, *Vcoef=None*, *fs=None*, *resolution=100*)

More in `dtf_fun()`.

class connectivipy.conn.**ffDTF**

ffDTF - class inherits from `ConnectAR` and overloads `Connect.calculate()` method.

calculate(*Acoef=None*, *Vcoef=None*, *fs=None*, *resolution=100*)

full-frequency Directed Transfer Function estimation from MVAR parameters. Args:

Acoef [numpy.array] array of shape (k, k, p) where *k* is number of channels and *p* is a model order.

Vcoef [numpy.array] prediction error matrix (k, k)

***fs* = 1** [int] sampling rate

***resolution* = 100** [int] number of spectrum data points

***generalized* = False** [bool] generalized version or not

Returns:

ffDTF [numpy.array] matrix with estimation results (*resolution*, k, k)

References: .. [1] Korzeniewska, A.et. all. Determination of information flow direction

among brain structures by a modified directed transfer function (dDTF) method. J. Neurosci. Methods 125, 195–207 (2003).

class connectivipy.conn.**dDTF**

dDTF - class inherits from `ConnectAR` and overloads `Connect.calculate()` method.

calculate(*Acoef=None*, *Vcoef=None*, *fs=None*, *resolution=100*)

direct Directed Transfer Function estimation from MVAR parameters. dDTF is a DTF multiplied in each frequency by Patrial Coherence. Args:

Acoef [numpy.array] array of shape (k, k, p) where *k* is number of channels and *p* is a model order.

Vcoef [numpy.array] prediction error matrix (k, k)

***fs* = 1** [int] sampling rate

***resolution* = 100** [int] number of spectrum data points

***generalized* = False** [bool] generalized version or not

Returns:

dDTF [numpy.array] matrix with estimation results (*resolution*, k, k)

References: .. [1] Korzeniewska, A.et. all. Determination of information flow direction

among brain structures by a modified directed transfer function (dDTF) method. J. Neurosci. Methods 125, 195–207 (2003).

class connectivipy.conn.**Coherency**

Coherency - class inherits from `Connect` and overloads `Connect.calculate()` method and *values_range* attribute.

calculate(*data*, *cnfft=None*, *cno=None*, *window=<function hanning>*, *im=False*)

Coherency calculation using FFT mehtod. Args:

data [numpy.array] array of shape (k, N) where *k* is number of channels and *N* is number of data points.

***cnfft* = None** [int] number of data points in window; if None, it is N/5

***cno* = 0** [int] overlap; if None, it is N/10

***window* = np.hanning** [<function> generating window with 1 arg *n*] window function

***im* = False** [bool] if False it return absolute value, otherwise complex number

Returns:

COH [numpy.array] matrix with estimation results (*resolution*, k, k)

References: .. [1] M. B. Priestley Spectral Analysis and Time Series.

Academic Press Inc. (London) LTD., 1981

class connectivipy.conn.**PSI**

PSI - class inherits from `Connect` and overloads `Connect.calculate()` method.

calculate(*data*, *band_width=4*, *psinfft=None*, *psino=0*, *window=<function hanning>*)

Phase Slope Index calculation using FFT mehtod. Args:

data [numpy.array] array of shape (k, N) where *k* is number of channels and *N* is number of data points.

***band_width* = 4** [int] width of frequency band where PSI values are summed

***psinfft* = None** [int] number of data points in window; if None, it is N/5

***psino* = 0** [int] overlap; if None, it is N/10

***window* = np.hanning** [<function> generating window with 1 arg *n*] window function

Returns:

COH [numpy.array] matrix with estimation results (*resolution*, k, k)

References: .. [1] Nolte G. et all, Comparison of Granger Causality and

Phase Slope Index. 267–276 (2009).

class connectivipy.conn.**GCI**

GCI - class inherits from `Connect` and overloads `Connect.calculate()` method.

calculate(*data*, *gcimethod='yw'*, *gciorder=None*)

Granger Causality Index calculation from MVAR model. Args:

data [numpy.array] array of shape (k, N) where *k* is number of channels and *N* is number of data points.

***gcimethod* = 'yw'** [int] MVAR parameters estimation model

***gciorder* = None** [int] model order, if None appropiate value is chosen basic on default criterion

Returns:

gci [numpy.array] matrix with estimation results (*resolution*, k, k)

References: .. [1] Nolte G. et all, Comparison of Granger Causality and

Phase Slope Index. 267–276 (2009).

1.3 Mvarmodel

1.3.1 MVAR class

Tools for MVAR parameters fitting

class connectivipy.mvarmodel.**Mvar**

Static class *Mvar* to multivariete autoregressive model fitting. Possible methods are in *fitting_algorithms* where key is acronym of algorithm and value is a function from *mvar.fitting*.

classmethod fit(*data*, *order=None*, *method='yw'*)

Mvar model fitting. Args:

data [numpy.array] array with data shaped (k, N), k - channels nr, N-data points)

***order* = None** [int] model order, when default None it estimates order using akaike order criteria.

***method* = 'yw': str** name of mvar fitting algorithm, default Yule-Walker all avaiable methods you can find in *fitting_algorithms*

Returns:

Av [numpy.array] model coefficients (kXkXorder)

Vf [numpy.array] reflection matrix (kXk)

classmethod order_akaike(*data*, *p_max=None*, *method='yw'*)

Akaike criterion of MVAR order estimation.

Args:

data [numpy.array] multichannel data in shape (k, n) for one trial case and (k, n, tr) for multitrial k - nr of channels, n -data points, tr - nr of trials

***p_max* = 5** [int] maximal model order

***method* = 'yw'** [str] name of the mvar calculation method

Returns:

best_order [int] minimum of *crit* array

crit [numpy.array] order criterion values for each value of order *p* starting from 1

References: .. [1] Blinowska K. J., Zygierewicz J., (2012) Practical

biomedical signal analysis using MATLAB. Boca Raton: Taylor & Francis.

classmethod order_hq(*data*, *p_max=None*, *method='yw'*)

Hannan-Quin criterion of MVAR order estimation.

Args:

data [numpy.array] multichannel data in shape (k, n) for one trial case and (k, n, tr) for multitrial k - nr of channels, n -data points, tr - nr of trials

***p_max* = 5** [int] maximal model order

***method* = 'yw'** [str] name of the mvar calculation method

Returns:

best_order [int] minimum of *crit* array

crit [numpy.array] order criterion values for each value of order *p* starting from 1

References: .. [1] Blinowska K. J., Zygierewicz J., (2012) Practical

biomedical signal analysis using MATLAB. Boca Raton: Taylor & Francis.

classmethod order_schwartz(*data*, *p_max=None*, *method='yw'*)

Schwartz criterion of MVAR order estimation.

Args:

data [numpy.array] multichannel data in shape (k, n) for one trial case and (k, n, tr) for multitrial k - nr of channels, n -data points, tr - nr of trials

***p_max* = 5** [int] maximal model order

***method* = 'yw'** [str] name of the mvar calculation method

Returns:

best_order [int] minimum of *crit* array

crit [numpy.array] order criterion values for each value of order *p* starting from 1

References: .. [1] Blinowska K. J., Zygierewicz J., (2012) Practical

biomedical signal analysis using MATLAB. Boca Raton: Taylor & Francis.

classmethod order_fpe(*data*, *p_max=None*, *method='yw'*)

Final Prediction Error criterion of MVAR order estimation. (not recommended) Args:

data [numpy.array] multichannel data in shape (k, n) for one trial case and (k, n, tr) for multitrial
k - nr of channels, n -data points, tr - nr of trials

***p_max* = 5** [int] maximal model order

***method* = 'yw'** [str] name of the mvar calculation method

Returns:

best_order [int] minimum of *crit* array

crit [numpy.array] order criterion values for each value of order *p* starting from 1

References: .. [1] Akaike H, (1970), Statistical predictor identification,

Ann. Inst. Statist. Math., 22 203–217.

1.3.2 Algorithms

connectivipy.mvar.fitting.**mvar_gen**(*Acf*, *npoints*, *omit=500*)

Generating data point from MVAR coefficiencs matrix *Acf*. Args:

Acf [numpy.array] array in shape of (p,k,k) where *k* is number of channels and *p* is a model order.

npoints [int] number of data points.

Returns:

y [numpy.array] (k, npoints) data points

connectivipy.mvar.fitting.**mvar_gen_inst**(*Acf*, *npoints*, *omit=500*)

Generating data point from matrix *A* with MVAR coefficients but it takes into account also zerolag interactions. So here Acf[0] means instantenous interaction not as in *mvar_gen* one data point lagged. Args:

Acf [numpy.array] array in shape of (p,k,k) where *k* is number of channels and *p* is a model order.

npoints [int] number of data points.

Returns:

y [np.array] (k, n) data points

connectivipy.mvar.fitting.**meanncov**(*x*, *y=[]*, *p=0*, *norm=True*)

Wrapper to multichannel case of new covariance *ncov*. Args:

x [numpy.array] multidimensional data (channels, data points, trials).

***y* = []** [numpy.array]

multidimensional data. If not given the autocovariance of *x* will be calculated.

***p* = 0: int** window shift of input data. It can be negative as well.

***norm* = True: bool** normalization - if True the result is divided by length of *x*, otherwise it is not.

Returns:

mcov [np.array] covariance matrix

`connectivipy.mvar.fitting.`**`ncov`**(*x*, *y=[]*, *p=0*, *norm=True*)

New covariance. Args:

x [numpy.array] onedimensional data.

***y* = []** [numpy.array]

onedimensional data. If not given the autocovariance of *x* will be calculated.

***p* = 0: int** window shift of input data. It can be negative as well.

***norm* = True: bool** normalization - if True the result is divided by length of *x*, otherwise it is not.

Returns:

kv [np.array] covariance matrix

`connectivipy.mvar.fitting.`**`vieiramorf`**(*y*, *pmax=1*)

Compute multichannel autoregresive model coefficients using Vieira-Morf algorithm. Args:

y [numpy.array] multichannel data in shape (k, n) for one trial case and (k, n, tr) for multitrial k - nr of channels, n -data points, tr - nr of trials

***pmax*: int >0** model order

Returns:

Ar [np.array] matrix with parameters matrix (p, k, k) where p - model order, k - nr of channels

Vr [np.array] prediction error matrix (k,k)

References: .. [1] Marple, Jr S. L., *Digital Spectral Analysis with

Applications*, Prentice-Hall Signal Processing Series, 1987

`connectivipy.mvar.fitting.`**`nutallstrand`**(*y*, *pmax=1*)

Compute multichannel autoregresive model coefficients using Nutall-Strand algorithm. Args:

y [numpy.array] multichannel data in shape (k, n) for one trial case and (k, n, tr) for multitrial k - nr of channels, n -data points, tr - nr of trials

***pmax*: int >0** model order

Returns:

Ar [np.array] matrix with parameters matrix (p, k, k) where p - model order, k - nr of channels

Vr [np.array] prediction error matrix (k,k)

References: .. [1] Marple, Jr S. L., *Digital Spectral Analysis with

Applications*, Prentice-Hall Signal Processing Series, 1987

`connectivipy.mvar.fitting.`**`yulewalker`**(*y*, *pmax=1*)

Compute multichannel autoregresive model coefficients using Yule-Walker algorithm. Args:

y [numpy.array] multichannel data in shape (k, n) for one trial case and (k, n, tr) for multitrial k - nr of channels, n -data points, tr - nr of trials

***pmax*: int >0** model order

Returns:

Ar [np.array] matrix with parameters matrix (p, k, k) where p - model order, k - nr of channels

Vr [np.array] prediction error matrix (k,k)

References: .. [1] Marple, Jr S. L., *Digital Spectral Analysis with

Applications*, Prentice-Hall Signal Processing Series, 1987

1.3.3 Additional

Additional tools

`connectivipy.mvar.comp.`**`ldl`**`(A)`

LDL decomposition (implementation from *en.wikipedia.org/wiki/Cholesky_decomposition*) Args:

A [numpy.array] matrix kXk

Returns:

L, D, LT [np.array] *L* is a lower unit triangular matrix, *D* is a diagonal matrix and *LT* is a transpose of *L*.

Tutorials:

1.4 Installation

The easiest way is to use *GIT* and just execute:

```
$ git clone https://github.com/dokato/connectivipy.git
$ cd connectivipy
$ python setup.py install
```

1.5 Examples

(tested under Python 2.7 and 3.4)

1.5.1 Loading data

```
import connectivipy as cp

# remember that data should be in a shape (k, N, R),
# where k - number of channels, N - data points, R - number of trials

# for numpy.array simply put that array as a first argument
# when initializing Data class
# fs means sampling frequency
```

(continues on next page)

(continued from previous page)

```
# chan_names is a list with channel names (length of list must be
#             the same as first dimension of data)
# data_info - additional infromation about the data
dt = cp.Data(numpy_array_data, fs=32., chan_names=['Fp1','O1'], data_info='sml')

# Matlab data we can read giving a path to a matlab file
# and in data_info we put Matlab variable name as a string
dd = cp.Data('adata.mat', data_info='bdata')

# similarly for SignalML data, but in data_info you need to point out
# that you want to read 'sml' data from *.raw and *.xml files with the
# same name
dt = cp.Data('cdata.raw', data_info='sml')
```

1.5.2 Data class example

```
# Example 1

import numpy as np
import connectivipy as cp
from connectivipy import mvar_gen

### MVAR model coefficients

"""
MVAR parameters taken from:
Sameshima K. & Baccala L. A., Partial directed coherence : a new
concept in neural structure determination. Biol. Cybern. (2001)
You can compare results with Fig. 3. from that article.
"""

# let's build mvar model matrix
A = np.zeros((2, 5, 5))
# 2 - first dimension is model order
# 5 - second and third dimensions mean number of channels
A[0, 0, 0] = 0.95 * 2**0.5
A[1, 0, 0] = -0.9025
A[0, 1, 0] = -0.5
A[1, 2, 1] = 0.4
A[0, 3, 2] = -0.5
A[0, 3, 3] = 0.25 * 2**0.5
A[0, 3, 4] = 0.25 * 2**0.5
A[0, 4, 3] = -0.25 * 2**0.5
A[0, 4, 4] = 0.25 * 2**0.5

# multitrial signal generation from a matrix above
# let's generate 5-channel signal with 1000 data points
# and 5 trials using function mvar_gen
ysig = np.zeros((5, 10**3, 5))
ysig[:, :, 0] = mvar_gen(A, 10**3)
ysig[:, :, 1] = mvar_gen(A, 10**3)
ysig[:, :, 2] = mvar_gen(A, 10**3)
ysig[:, :, 3] = mvar_gen(A, 10**3)
ysig[:, :, 4] = mvar_gen(A, 10**3)
```

(continues on next page)

(continued from previous page)

```
#### connectivity analysis
data = cp.Data(ysig, 128, ["Fp1", "Fp2", "Cz", "O1", "O2"])

# you may want to plot data (in multitrial case only one trial is shown)
data.plot_data()

# fit mvar using Yule-Walker algorithm and order 2
data.fit_mvar(2, 'yw')

# you can capture fitted parameters and residual matrix
ar, vr = data.mvar_coefficients

# now we investigate connectivity using gDTF
gdtf_values = data.conn('gdtf')
gdtf_significance = data.significance(Nrep=200, alpha=0.05)
data.plot_conn('gDTF')

# short time version with default parameters
pdc_shorttime = data.short_time_conn('pdc', nfft=100, no=10)
data.plot_short_time_conn("PDC")
```

1.5.3 How to use specific classes

```
# Example 2

import numpy as np
import matplotlib.pyplot as plt
import connectivipy as cp
from connectivipy import mvar_gen

"""
In this example we don't use Data class
"""

fs = 256.
acf = np.zeros((3, 3, 3))
# matrix shape meaning
# (p,k,k) k - number of channels,
# p - order of mvar parameters

acf[0, 0, 0] = 0.3
acf[0, 1, 0] = 0.6
acf[1, 0, 0] = 0.1
acf[1, 1, 1] = 0.2
acf[1, 2, 0] = 0.6
acf[2, 2, 2] = 0.2
acf[2, 1, 0] = 0.4

# generate 3-channel signal from matrix above
y = mvar_gen(acf, int(10e4))

# assign static class cp.Mvar to variable mv
mv = cp.Mvar

# find best model order using Vieira-Morf algorithm
```

(continues on next page)

(continued from previous page)

```
best, crit = mv.order_akaike(y, 15, 'vm')
plt.plot(1+np.arange(len(crit)), crit, 'g')
plt.show()
print(best)
# here we know that this is 3 but in real-life cases
# we are always uncertain about it

# now let's fit parameters to the signal
av, vf = mv.fit(y, best, 'vm')

# and check whether values are correct +/- 0.01
print(np.allclose(acf, av, 0.01, 0.01))

# now we can calculate Directed Transfer Function from the data
dtf = cp.conn.DTF()
dtfval = dtf.calculate(av, vf, 128)
# all possible methods are visible in that dictionary:
print(cp.conn.conn_estim_dc.keys())

cp.plot_conn(dtfval, 'DTF values', fs)
```

1.5.4 Instantaneous

```
# Example 3

import numpy as np
import matplotlib.pyplot as plt
import connectivipy as cp

"""
This example reproduce simulation from article:
Erla S et all (2009) "Multivariate autoregressive model with
                      instantaneous effects to improve brain
                      connectivity estimation"
"""

# let's make a matrix from original article

bcf = np.zeros((4, 5, 5))
# matrix shape meaning (k, k, p) k - number of channels,
# p - order of mvar parameters
bcf[1, 0, 0] = 1.58
bcf[2, 0, 0] = -0.81
bcf[0, 1, 0] = 0.9
bcf[2, 1, 1] = -0.01
bcf[3, 1, 4] = -0.6
bcf[1, 2, 1] = 0.3
bcf[1, 2, 2] = 0.8
bcf[2, 2, 1] = 0.3
bcf[2, 2, 2] = -0.25
bcf[3, 2, 1] = 0.3
bcf[0, 3, 1] = 0.9
bcf[1, 3, 1] = -0.6
bcf[3, 3, 1] = 0.3
bcf[1, 4, 3] = -0.3
```

(continues on next page)

(continued from previous page)

```
bcf[2, 4, 0] = 0.9
bcf[2, 4, 3] = -0.3
bcf[3, 4, 2] = 0.6

# now we build a corresponding MVAR process without instantenous effect
L = np.linalg.inv(np.eye(5)-bcf[0])
acf = np.zeros((3, 5, 5))
for i in range(3):
    acf[i] = np.dot(L, bcf[i+1])

# generate 5-channel signals from matrix above
signal_inst = cp.mvar_gen_inst(bcf, int(10e4))
signal = cp.mvar_gen(acf, int(10e4))

# fit MVAR parameters
bv, vfb = cp.Mvar.fit(signal_inst, 3, 'yw')

av, vfa = cp.Mvar.fit(signal, 3, 'yw')

# use connectivity estimators
ipdc = cp.conn.iPDC()
ipdcval = ipdc.calculate(bv, vfb, 1.)

pdc = cp.conn.PDC()
pdcval = pdc.calculate(av, vfa, 1.)

def plot_double_conn(values_a, values_b, name='', fs=1, ylim=None, xlim=None,␣
↪show=True):
    "function to plot two sets of connectivity values"
    fq, k, k = values_a.shape
    fig, axes = plt.subplots(k, k)
    freqs = np.linspace(0, fs*0.5, fq)
    if not xlim:
        xlim = [0, np.max(freqs)]
    if not ylim:
        ylim = [0, 1]
    for i in range(k):
        for j in range(k):
            axes[i, j].fill_between(freqs, values_b[:, i, j], 0, facecolor='red',␣
↪alpha=0.5)
            axes[i, j].fill_between(freqs, values_a[:, i, j], 0, facecolor='black',␣
↪alpha=0.5)
            axes[i, j].set_xlim(xlim)
            axes[i, j].set_ylim(ylim)
    plt.suptitle(name,y=0.98)
    plt.tight_layout()
    plt.subplots_adjust(top=0.92)
    if show:
        plt.show()

plot_double_conn(pdcval**2, ipdcval**2, 'PDC / iPDC')
```

CHAPTER 2

Credits:

- Dominik Krzemiński
- Maciej Kamiński (scientific lead)

Python Module Index

C

Index

P

R

S

V

Y