
eeglib Documentation

Release 0.4

Luis Cabañero Gómez

Apr 05, 2021

CONTENTS

1	Documentation of eeglib.eeg module	3
2	Documentation of eeglib.helpers module	15
3	Documentation of eeglib.wrapper module	21
4	Documentation of eeglib.features module	23
5	Documentation of eeglib.preprocessing module	27
6	Indices and tables	29
	Bibliography	31
	Python Module Index	33
	Index	35

This module is a library with some tools and functions for EEG signal analysis.

Author: Luis Cabañero Gómez

This reference manual contains the documentation about the modules, classes and functions included in eeglib. For learning how to use this library, use the [Quickstart Guide](#). Next, there are the submodules API.

DOCUMENTATION OF EEGLIB.EEG MODULE

This module define the basic data structure that are used in this library

class `eeplib.eeg.EEG` (*windowSize*, *sampleRate*, *channelNumber*, *names=None*)

Bases: `object`

This class apply signal analysis functions over the data stored in its attribute window.

Attributes

windowSize: int The maximum samples the window will store.

sampleRate: int The number of samples per second

channelNumber: int The number of channels of samples the window will handle.

window: SampleWindow It stores the data.

Methods

<i>CCC</i> ([channels, allPermutations])	Computes the Cross Correlation Coefficient between the data in c1 and the data in c2.
<i>DFA</i> ([i])	Applies Detrended Fluctuation Analysis algorithm to the given data.
<i>DFT</i> ([i, windowFunction, output, ...])	Returns the Discrete Fourier Transform of the data at a given index of the window.
<i>DTW</i> ([channels, allPermutations, normalize, ...])	Computes the Dynamic Time Warping algorithm between the data of the given channels.
<i>HFD</i> ([i, kMax])	Returns the Higuchi Fractal Dimension at the given index of the window.
<i>LZC</i> ([i])	Returns the Lempel-Ziv Complexity at the given channel/s.
<i>PFD</i> ([i])	Returns the Petrosian Fractal Dimension at the given index of the window.
<i>PSD</i> ([i, windowFunction, nperseg, retFrequencies])	Returns the Power Spectral Density of the data at a given index of the window using the Welch method.
<i>bandPower</i> ([i, bands, spectrumFrom, ...])	Returns the power of each band at the given index.
<i>engagementLevel</i> ()	Returns the engagement level, which is calculated with this formula: $\beta/(\alpha+\theta)$, where α , β and θ are the average of the average band values between all the channels.
<i>getBoundsForBand</i> (bandBounds)	Returns the bounds of each band depending of the sample rate and the window size.

continues on next page

Table 1 – continued from previous page

<code>getChannel([i])</code>	Returns the raw data stored at the given index of the windows.
<code>getSignalAtBands([i, bands])</code>	Rebuilds the signal from a component <i>i</i> but only in the specified frequency bands.
<code>hjorthActivity([i])</code>	Returns the Hjorth Activity at the given channel
<code>hjorthComplexity([i])</code>	Returns the Hjorth Complexity at the given channel
<code>hjorthMobility([i])</code>	Returns the Hjorth Mobility at the given channel
<code>sampEn([i])</code>	Returns Multiscale Sample Entropy at the given channel/s.
<code>set(samples[, columnMode])</code>	Sets multiple samples at a time into the window.
<code>synchronizationLikelihood([channels, ...])</code>	Returns the Synchronization Likelihood value applied over the <i>i1</i> and <i>i2</i> channels by calling <code>synchronizationLikelihood()</code> .

CCC (*channels=None, allPermutations=False*)

Computes the Cross Correlation Coefficient between the data in *c1* and the data in *c2*.

Parameters

channels: Variable type, optional In order to understand how this parameter works go to the doc of `eeglib.eeg.EEG._applyFunctionTo2C()`

allPermutations: bool, optional

In order to understand how this parameter works go to the doc of

`eeglib.eeg.EEG._applyFunctionTo2C()`

Returns

float or dict If a tuple is passed the it returns the result of applying the function to the channels specified in the tuple. If another valid value is passed, the method returns a dictionary, being the key the two channels used and the value the result of applying the function to those channels.

DFA (*i=None, *args, **kwargs*)

Applies Detrended Fluctuation Analysis algorithm to the given data.

Parameters

i: Variable type, optional

- int: the index of the channel.
- str: the name of the channel.
- list of strings and integers: a list of channels that will be used.
- slice: a slice selecting the range of channels that will be used.
- None: all the channels will be used

fit_degree: int, optional Degree of the polynomial used to model de local trends. Default: 1.

min_window_size: int, optional Size of the smallest window that will be used. Default: `signalSize//2`.

fskip: float, optional Fraction of the window that will be skipped in each iteration for each window size. Default: 1.

Returns

float or array The resulting value. If more than one channel was selected the return object will be a 1D array containing the result of the processing.

DFT (*i=None, windowFunction=None, output='complex', onlyPositiveFrequencies=False*)

Returns the Discrete Fourier Transform of the data at a given index of the window.

Parameters

i: Variable type, optional

- int: the index of the channel.
- str: the name of the channel.
- list of strings and integers: a list of channels that will be used.
- slice: a slice selecting the range of channels that will be used.
- None: all the channels will be used.

windowFunction: str, tuple or numpy.ndarray, optional This can be a string with the name of the function, a tuple with a str with the name of the function in the first position and the parameters of the function in the nexts or a numpy array with a size equals to the window size. In the first case an array with the size of windowSize will be created. The created array will be multiplied by the data in the window before FFT is used.

output: str, optional

- “complex”: default output of the FFT, $x+yi$
- “magnitude”: computes the magnitude of the FFT, $\sqrt{x^2+y^2}$
- “phase”: computes the phase of the FFT, $\text{atan2}(yi,x)$

Returns

numpy.ndarray An array with the result of the Fourier Transform. If more than one channel was selected the array will be of 2 Dimensions.

DTW (*channels=None, allPermutations=False, normalize=False, returnOnlyDistances=True, *args, **kwargs*)

Computes the Dynamic Time Warping algorithm between the data of the given channels. It uses the FastDTW implementation given by the library fastdtw.

Parameters

channels: Variable type, optional In order to understand how this parameter works go to the doc of `eeglib.eeg.EEG._applyFunctionTo2C()`

allPermutations: bool, optional

In order to understand how this parameter works go to the doc of

`eeglib.eeg.EEG._applyFunctionTo2C()`

normalize: bool optional If True the result of the algorithm is divided by the window size. Default: True.

returnOnlyDistances: bool, optional If True, the result of the function will include only the distances after applying the DTW algorithm. If False it will return also the path. Default: True.

radius [int, optional] size of neighborhood when expanding the path. A higher value will increase the accuracy of the calculation but also increase time and memory consumption. A radius equal to the size of x and y will yield an exact dynamic time warping calculation.

dist [function or int, optional] The method for calculating the distance between $x[i]$ and $y[j]$. If dist is an int of value $p > 0$, then the p-norm will be used. If dist is a function then $\text{dist}(x[i], y[j])$ will be used. If dist is None then $\text{abs}(x[i] - y[j])$ will be used.

Returns

tuple, float, dict of floats or dict of tuples If a tuple is passed the it returns the result of applying the function to the channels specified in the tuple. If another valid value is passed, the method returns a dictionary, being the key the two channels used and the value the result of applying the function to those channels.

HFD ($i=None, kMax=None$)

Returns the Higuchi Fractal Dimension at the given index of the window.

Parameters

i: Variable type, optional

- int: the index of the channel.
- str: the name of the channel.
- list of strings and integers: a list of channels that will be used.
- slice: a slice selecting the range of channels that will be used.
- None: all the channels will be used.

kmax: int, optional By default it will be $\text{windowSize}/2$.

Returns

float or array The resulting value. If more than one channel was selected the return object will be a 1D array containing the result of the processing.

LZC ($i=None, *args, **kwargs$)

Returns the Lempel-Ziv Complexity at the given channel/s.

Parameters

i: Variable type, optional

- int: the index of the channel.
- str: the name of the channel.
- list of strings and integers: a list of channels that will be used.
- slice: a slice selecting the range of channels that will be used.
- None: all the channels will be used

threshold: numeric, optional A number use to binarize the signal. The values of the signal above threshold will be converted to 1 and the rest to 0. By default, the median of the data.

normalize: bool If True the resulting value will be between 0 and 1, being 0 the minimal possible complexity of a sequence that has the same length of data and 1 the maximal possible complexity. By default, False.

Returns

float or array The resulting value. If more than one channel was selected the return object will be a 1D array containing the result of the processing.

PFD ($i=None$)

Returns the Petrosian Fractal Dimension at the given index of the window.

Parameters**i: Variable type, optional**

- int: the index of the channel.
- str: the name of the channel.
- list of strings and integers: a list of channels that will be used.
- slice: a slice selecting the range of channels that will be used.
- None: all the channels will be used.

Returns

float or array The resulting value. If more than one channel was selected the return object will be a 1D array containing the result of the processing.

PSD (*i=None, windowFunction='hann', nperseg=None, retFrequencies=False*)

Returns the Power Spectral Density of the data at a given index of the window using the Welch method.

Parameters**i: Variable type, optional**

- int: the index of the channel.
- str: the name of the channel.
- list of strings and integers: a list of channels that will be used.
- slice: a slice selecting the range of channels that will be used.
- None: all the channels will be used.

windowFunction: str or tuple or array_like, optional Desired window to use. If window is a string or tuple, it is passed to `get_window` to generate the window values, which are DFT-even by default. See `get_window` for a list of windows and required parameters. If window is array_like it will be used directly as the window and its length must be `nperseg`. Defaults to a Hann window.

nperseg: int, optional Length of each segment. Defaults to None, but if window is str or tuple, is set to 256, and if window is array_like, is set to the length of the window.

retFrequencies: bool, optional If True two arrays will be returned for each channel, one with the frequencies and another with the spectral density of those frequencies.

Returns

numpy.ndarray An array with the result of the Fourier Transform. If more than one channel was selected the array will be of 2 Dimensions.

__init__ (*windowSize, sampleRate, channelNumber, names=None*)

Parameters

windowSize: int The maximum samples the window will store.

sampleRate: int The number of samples per second

channelNumber: int The number of channels of samples the window will handle.

names: list of strings, optional The optional names that can be used to refer to each channel.

bandPower (*i=None, bands={'alpha': (8, 12), 'beta': (12, 30), 'delta': (1, 4), 'theta': (4, 7)}, spectrumFrom='DFT', windowFunction='hann', nperseg=None, normalize=False*)
Returns the power of each band at the given index.

Parameters

i: Variable type, optional

- int: the index of the channel.
- str: the name of the channel.
- list of strings and integers: a list of channels that will be used.
- slice: a slice selecting the range of channels that will be used.
- None: all the channels will be used.

bands: dict, optional This parameter is used to indicate the bands that are going to be used. It is a dict with the name of each band as key and a tuple with the lower and upper bounds as value.

spectrumFrom: str, optional

- “DFT”: uses the spectrum from the DFT of the signal.
- “PSD”: uses the spectrum from the PSD of the signal.

windowFunction: str or tuple or array_like, optional Desired window to use. If window is a string or tuple, it is passed to `get_window` to generate the window values, which are DFT-even by default. See `get_window` for a list of windows and required parameters. If window is `array_like` it will be used directly as the window and its length must be `nperseg`. Defaults to a Hann window.

nperseg: int, optional This parameter is only relevant when `powerFrom` is “PSD”, else it is ignored. Length of each segment. Defaults to None, but if window is str or tuple, is set to 256, and if window is `array_like`, is set to the length of the window.

normalize: bool, optional If True the each band power is divided by the total power of the spectrum. Default False.

Returns

dict or list of dict The keys are the name of each band and the values are the mean of the magnitudes. If more than one channel was selected the return object will be a list containing the dict for each channel selected

engagementLevel ()

Returns the engagement level, which is calculated with this formula: $\text{beta}/(\text{alpha}+\text{theta})$, where alpha, beta and theta are the average of the average band values between all the channels.

Returns

float The engagement level.

getBoundsForBand (*bandBounds*)

Returns the bounds of each band depending of the sample rate and the window size.

Parameters

bandbounds: tuple A tuple containing the lower and upper bounds of a frequency band.

Returns

tuple A tuple containing the new bounds of the given band.

getChannel (*i=None*)

Returns the raw data stored at the given index of the windows.

Parameters

i: Variable type, optional

- int: the index of the channel.
- str: the name of the channel.
- list of strings and integers: a list of channels that will be returned as a 2D ndarray.
- slice: a slice selecting the range of channels that will be returned as a 2D ndarray.
- None: all the data returned as a 2D ndarray

Returns

list The list of values of a specific channel.

getSignalAtBands (*i=None, bands={'alpha': (8, 12), 'beta': (12, 30), 'delta': (1, 4), 'theta': (4, 7)}*)

Rebuilds the signal from a component *i* but only in the specified frequency bands.

Parameters

i: Variable type, optional

- int: the index of the channel.
- str: the name of the channel.
- list of strings and integers: a list of channels that will be used.
- slice: a slice selecting the range of channels that will be used.
- None: all the channels will be used.

bands: dict, optional This parameter is used to indicate the bands that are going to be used. It is a dict with the name of each band as key and a tuple with the lower and upper bounds as value.

Returns

dict of numpy.ndarray (1D or 2D) The keys are the same keys the bands dictionary is using. The values are the signal filtered in every band at the given index of the window. If more than one channel is selected the return object will be a dict containing 2D arrays in which each row is a signal filtered at the corresponding channel.

hjorthActivity (*i=None*)

Returns the Hjorth Activity at the given channel

Parameters

i: int or string, optional Index or name of the channel.

Returns

float or array The resulting value. If more than one channel was selected the return object will be a 1D array containing the result of the processing.

hjorthComplexity (*i=None*)

Returns the Hjorth Complexity at the given channel

Parameters

i: Variable type, optional

- **int**: the index of the channel.
- **str**: the name of the channel.
- **list of strings and integers**: a list of channels that will be used.
- **slice**: a slice selecting the range of channels that will be used.
- **None**: all the channels will be used.

Returns

float or array The resulting value. If more than one channel was selected the return object will be a 1D array containing the result of the processing.

hjorthMobility (*i=None*)

Returns the Hjorth Mobility at the given channel

Parameters**i: Variable type, optional**

- **int**: the index of the channel.
- **str**: the name of the channel.
- **list of strings and integers**: a list of channels that will be used.
- **slice**: a slice selecting the range of channels that will be used.
- **None**: all the channels will be used.

Returns

float or array The resulting value. If more than one channel was selected the return object will be a 1D array containing the result of the processing.

sampEn (*i=None, *args, **kwargs*)

Returns Multiscale Sample Entropy at the given channel/s.

Parameters**i: Variable type, optional**

- **int**: the index of the channel.
- **str**: the name of the channel.
- **list of strings and integers**: a list of channels that will be used.
- **slice**: a slice selecting the range of channels that will be used.
- **None**: all the channels will be used

m: int, optional Size of the embedded vectors. By default 2.

l: int, optional Lag between elements of embedded vectors. By default 1.

r: float, optional Tolerance. By default $\text{fr} \cdot \text{std}(\text{data})$

fr: float, optional Fraction of $\text{std}(\text{data})$ used as tolerance. If **r** is passed, this parameter is ignored. By default, 0.2.

Returns

float or array The resulting value. If more than one channel was selected the return object will be a 1D array containing the result of the processing.

set (*samples*, *columnMode=False*)

Sets multiple samples at a time into the window. The sample number must be the same as the window size.

Parameters

samples: array_like of 2 dimensions Samples of data with a size equals to window.

columnMode: boolean, optional By default it is assumed that the shape of the data given is nSamples X nchannels. If the given data is the inverse, columnMode should be True.

synchronizationLikelihood (*channels=None*, *allPermutations=False*, *m=None*, *l=None*, *w1=None*, *w2=None*, *pRef=0.05*, ***kargs*)

Returns the Synchronization Likelihood value applied over the i1 and i2 channels by calling `synchronizationLikelihood()`.

Parameters

channels: Variable type, optional In order to understand how this parameter works go to the doc of `eeglib.eeg.EEG._applyFunctionTo2C()`

allPermutations: bool, optional

In order to understand how this parameter works go to the doc of

`eeglib.eeg.EEG._applyFunctionTo2C()`

m: int, optional Numbers of elements of the embedded vectors.

l: int, optional Separation between elements of the embedded vectors.

w1: int, optional Theiler correction for autocorrelation effects

w2: int, optional A window that sharpens the time resolution of the Synchronization measure

pRef: float, optional The p Ref param of the synchronizationLikelihood. Default 0.05

epsilonIterations: int, optional Number of iterations used to determine the value of epsilon

Returns

float or dict If a tuple is passed the it returns the result of applying the function to the channels specified in the tuple. If another valid value is passed, the method returns a dictionary, being the key the two channels used and the value the result of applying the function to those channels.

class `eeglib.eeg.SampleWindow` (*windowSize*, *channelNumber*, *names=None*)

Bases: `object`

This class is a data structure that stores the signal data and works like a sliding window.

Attributes

windowSize: int The maximum samples the window will store.

channelNumber: int The number of channels of samples the window will handle.

window: list of lists It stores the data.

Methods

<code>getChannel(i)</code>	Returns an array containing the data of the the selected channel/s.
<code>getIndicesList(i)</code>	Returns a list of numeric indices from a combined type of indices.
<code>getPairOfChannels([channels, allPermutations])</code>	Applies a function that uses two signals by selecting the channels to use.
<code>getPairsIndicesList(i[, allPermutations])</code>	Returns a list of tuples of numeric indices from a combined type of indices.
<code>set(samples[, columnMode])</code>	Sets multiple samples at a time.

`__init__` (*windowSize*, *channelNumber*, *names=None*)

Creates a SampleWindow wich stores the value of *windowSize* as the number of samples and the value of *channelNumber* as the number of channels.

Parameters

windowSize: int The size of the sliding window.

channelNumber: int The number of channels recording simultaneously.

names: list of strings, optional The optional names that can be used to refer to each channel.

`getChannel` (*i=None*)

Returns an array containing the data of the the selected channel/s.

Parameters

i: Variable type, optional

- int: the index of the channel.
- str: the name of the channel.
- list of strings and integers: a list of channels that will be returned as a 2D ndarray.
- slice: a slice selecting the range of channels that will be returned as a 2D ndarray.
- None: all the data returned as a 2D ndarray

Returns

numpy.ndarray Can be a one or a two dimension matrix, depending of the parameters.

`getIndicesList` (*i*)

Returns a list of numeric indices from a combined type of indices.

`getPairOfChannels` (*channels=None*, *allPermutations=False*)

Applies a function that uses two signals by selecting the channels to use. It will apply the function to different channels depending on the parameters. Note: a single channel can be selected by using an int or a string if a name for the channel was specified.

Parameters

channels: Variable type, optional

- tuple of lenght 2 containing channel indexes: applies the function to the two channels specified by the tuple.
- list of tuples(same restrictions than the above): applies the function to every tuple in the list.

- list of index: creates combinations of channels specifies in the list depending of the allPermutations parameter.
- None: Are channels are used in the same way than in the list above. This is the default value.

allPermutations: bool, optional Only used when channels is a list of index or None. If True all permutations of the channels in every order are used; if False only combinations of different channels are used. Example: with the list [0, 2, 4] and allPermutations = True, the channels used will be (0,2), (0,4), (2,0), (2,4), (4,0), (4,2); meanwhile, with allPermutations = False, the channels used will be: (0,2), (0,4), (2,4). Default: False.

Returns

If a tuple is passed the it returns the result of applying the function to the channels specified in the tuple. If another valid value is passed, the method returns a dictionary, being the key the two channels used and the value the result of applying the function to those channels.

getPairsIndicesList (*i*, *allPermutations=False*)

Returns a list of tuples of numeric indices from a combined type of indices.

set (*samples*, *columnMode=False*)

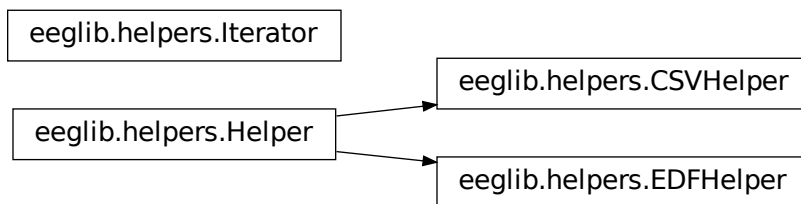
Sets multiple samples at a time. The sample number must be the same as the window size.

Parameters

samples: array_like of 2 dimensions Samples of data with a size equals to window.

columnMode: boolean, optional By default it is assumed that the shape of the data given is nSamples X nchannels. If the given data is the inverse, columnMode should be True.

DOCUMENTATION OF EEGLIB.HELPERS MODULE



This module contains helper classes that are useful to iterating over a EEG data stream. Currently there is support only for CSV files.

class `eeglib.helpers.CSVHelper` (*path*, *args, **kwargs)

Bases: `eeglib.helpers.Helper`

This class is for applying diferents operations using the EEG class over a csv file.

Methods

<code>getNames([indexes])</code>	Returns the names of the specified indexes of channels.
<code>moveEEGWindow(position)</code>	Moves the window to start at position.
<code>prepareEEG(windowSize)</code>	Prepares and creates the EEG object that the iteration will use with the same parameters that an EEG objects is initialized.
<code>prepareIterator([step, startPoint, endPoint])</code>	Prepares the iterator of the helper.
<code>selectSignals(selectedSignals)</code>	

Parameters

`__init__` (*path*, *args, **kwargs)

The rest of parameters can be seen at `Helper.__init__()`

Parameters

path: str The path to the csv file

getNames (*indexes=None*)

Returns the names of the specified indexes of channels.

Parameters

indexes: Iterable of int, optional. The indexes of the channels desired. If None it will return all the channels' names. Default: None.

moveEEGWindow (*position*)

Moves the window to start at position. Also it returns the inner eeg object.

Parameters

position: int, str or datetime.timedelta

- int: position of the sample
- str with format hh:mm:ss.ss: temporal position
- timedelta: temporal position

Returns

EEG

prepareEEG (*windowSize*)

Prepares and creates the EEG object that the iteration will use with the same parameters that an EEG objects is initialized. Also it returns the inner eeg object.

Parameters

windowSize: int The maximun samples the window will store.

Returns

EEG

prepareIterator (*step=None, startPoint=0, endPoint=None*)

Prepares the iterator of the helper.

Parameters

step: int, optional Number of samples to be skipped in each iteration.

startPoint: int, optional The index of first sample from where the iteration will start. By default 0.

endPoint: int, optional The index of the last sample + 1 until where the iteration will go. By default the size of the data.

selectSignals (*selectedSignals*)

Parameters

selectedSignals [iterable of str and/or int] The indexes of the desired selected signals. It must be an iterable type containing either str or int.

Returns

None.

class eeglib.helpers.**EDFHelper** (*path, *args, sampleRate=None, **kargs*)

Bases: [eeglib.helpers.Helper](#)

This class is for applying diferents operations using the EEG class over an edf file.

Methods

<code>getNames([indexes])</code>	Returns the names of the specified indexes of channels.
<code>moveEEGWindow(position)</code>	Moves the window to start at position.
<code>prepareEEG(windowSize)</code>	Prepares and creates the EEG object that the iteration will use with the same parameters that an EEG objects is initialized.
<code>prepareIterator([step, startPoint, endPoint])</code>	Prepares the iterator of the helper.
<code>selectSignals(selectedSignals)</code>	

Parameters

`__init__ (path, *args, sampleRate=None, **kwargs)`

The rest of parameters can be seen at `Helper.__init__()`

Parameters

path: `str` The path to the edf file

`getNames (indexes=None)`

Returns the names of the specified indexes of channels.

Parameters

indexes: `Iterable of int, optional`. The indexes of the channels desired. If `None` it will return all the channels' names. Default: `None`.

`moveEEGWindow (position)`

Moves the window to start at position. Also it returns the inner eeg object.

Parameters

position: `int, str or datetime.timedelta`

- `int`: position of the sample
- `str` with format `hh:mm:ss.ss`: temporal position
- `timedelta`: temporal position

Returns

`EEG`

`prepareEEG (windowSize)`

Prepares and creates the EEG object that the iteration will use with the same parameters that an EEG objects is initialized. Also it returns the inner eeg object.

Parameters

windowSize: `int` The maximun samples the window will store.

Returns

`EEG`

`prepareIterator (step=None, startPoint=0, endPoint=None)`

Prepares the iterator of the helper.

Parameters

step: `int, optional` Number of samples to be skipped in each iteration.

startPoint: int, optional The index of first sample from where the iteration will start. By default 0.

endPoint: int, optional The index of the last sample + 1 until where the iteration will go. By default the size of the data.

selectSignals (*selectedSignals*)

Parameters

selectedSignals [iterable of str and/or int] The indexes of the desired selected signals. It must be an iterable type containing either str or int.

Returns

None.

class eeglib.helpers.**Helper** (*data, sampleRate=None, windowSize=None, names=None, highpass=None, lowpass=None, normalize=False, ICA=False, selectedSignals=None*)

Bases: object

This is an abstract class that defines the way every helper works.

Methods

<i>getNames</i> (<i>[indexes]</i>)	Returns the names of the specified indexes of channels.
<i>moveEEGWindow</i> (<i>position</i>)	Moves the window to start at position.
<i>prepareEEG</i> (<i>windowSize</i>)	Prepares and creates the EEG object that the iteration will use with the same parameters that an EEG objects is initialized.
<i>prepareIterator</i> (<i>[step, startPoint, endPoint]</i>)	Prepares the iterator of the helper.
<i>selectSignals</i> (<i>selectedSignals</i>)	

Parameters

__init__ (*data, sampleRate=None, windowSize=None, names=None, highpass=None, lowpass=None, normalize=False, ICA=False, selectedSignals=None*)

Parameters

data: 2D matrix The signals data in the shape (nChannels, nSamples).

sampleRate: numeric, optional The frequency at which the data was recorded. By default its value is the lenght of the data.

windowSize: int, optional The size of the window in which the calculations will be done. By default its value is the lenght of one second of the data.

names: list of strings A list containing the names of each channel in the same positions than data channels.

highpass: numeric, optional The signal will be filtered above this value.

lowpass: numeric, optional The signal will be filtered bellow this value.

normalize: boolean, optional If True, the data will be normalizing using z-scores. Default = False.

ICA: boolean, optional If True, Independent Component Analysis will be applied to the data. It is applied always after normalization if `normalize = True`. Default: False.

selectedSignals: list of strings or ints If the data file has names associated to each columns, those columns can be selected through the name or the index of the column. If the data file hasn't names in the columns, they can be selected just by the index.

getNames (*indexes=None*)

Returns the names of the specified indexes of channels.

Parameters

indexes: Iterable of int, optional. The indexes of the channels desired. If None it will return all the channels' names. Default: None.

moveEEGWindow (*position*)

Moves the window to start at position. Also it returns the inner eeg object.

Parameters

position: int, str or datetime.timedelta

- int: position of the sample
- str with format hh:mm:ss.ss: temporal position
- timedelta: temporal position

Returns

EEG

prepareEEG (*windowSize*)

Prepares and creates the EEG object that the iteration will use with the same parameters that an EEG objects is initialized. Also it returns the inner eeg object.

Parameters

windowSize: int The maximum samples the window will store.

Returns

EEG

prepareIterator (*step=None, startPoint=0, endPoint=None*)

Prepares the iterator of the helper.

Parameters

step: int, optional Number of samples to be skipped in each iteration.

startPoint: int, optional The index of first sample from where the iteration will start. By default 0.

endPoint: int, optional The index of the last sample + 1 until where the iteration will go. By default the size of the data.

selectSignals (*selectedSignals*)

Parameters

selectedSignals [iterable of str and/or int] The indexes of the desired selected signals. It must be an iterable type containing either str or int.

Returns

None.

class eeglib.helpers.**Iterator** (*helper, step, auxPoint, endPoint*)
Bases: object

__init__ (*helper, step, auxPoint, endPoint*)
 Initialize self. See help(type(self)) for accurate signature.

DOCUMENTATION OF EEGLIB.WRAPPER MODULE

—

This module contains the Wrapper class that wraps around a Helper and computes every feature in each window configured. It also allows specifying labels for the whole signal or for specific segments.

class `eeglib.wrapper.Wrapper` (*helper, flat=True, flatSeparator='_', store=True, label=None, segmentation=None*)

This class wraps around a helper and allows getting multiple features at once. The main usage of this class is for generating features to use with machine learning techniques.

Methods

<code>addCustomFeature(function[, channels, ...])</code>	Adds a custom feature that will be included in the dataset.
<code>addFeatures(features)</code>	
Parameters	
<code>featuresNames()</code>	Returns the names of the specified features.
<code>getAllFeatures()</code>	Iterates over all the windows in the helper and returns all the values.
<code>getFeatures()</code>	Returns the features in the current window of the helper iterator.
<code>getStoredFeatures()</code>	Returns the stored features if store was set to True, else it returns None.
<code>reset()</code>	Resets the

`__init__` (*helper, flat=True, flatSeparator='_', store=True, label=None, segmentation=None*)

Parameters

helper: `:py:class:`eeglib.helpers.Helper`` The helper object that will be used to get the data and the iteration settings.

flat: **Bool** If True the result of calling `getFeatures()` will be a one dimensional sequence of data. If False, the return value will be a list containing the result of each feature, that can be a float, a dict or an array. Default: True.

separator: **str** It is used to separate the features names when flatten.

store: **Bool** If True, the data will be stored in a `self.storedFeatures`. Default: True.

label: **object, optional** This value will be added as a field in the data except if it is None. Default: None.

segmentation: dict of tuples, optional Parameter format `[((begin, end),label),]`; begin is the begin of the segment, end is the end of the segment and label the label realed to that segment. Begin and end can be either an int, a str or a `datetime.timedelta`, being the same that the parameter of `eeglib.helpers.Helper.moveEEGWindow()`. The label of the unlabelled segments will be 0. If None, no segmentation will be added. Default: None.

addCustomFeature (*function, channels=None, twoChannels=False, name=None*)

Adds a custom feature that will be included in the dataset.

Parameters

function: function The function to apply.

channels: Variable type, optional The channels over which the function will be applied. * int: the index of the channel. * str: the name of the channel. * list of strings and integers: a list of channels. * slice: a slice selecting the range of channels. * None: all the channels. The function to apply to the data to obtain the feature.

twoChannels: bool If function receives two channels of data this parameter should be True. Default: False.

name: str A custom name for the feature that will be visible in the df.

Returns

None

addFeatures (*features*)

Parameters

features: list(tuple(str,list, dict)) A list containing tuples that represent the parameters needed to add a single feature.

featuresNames ()

Returns the names of the specified features.

getAllFeatures ()

Iterates over all the windows in the helper and returns all the values.

Returns

pandas.DataFrame

getFeatures ()

Returns the features in the current window of the helper iterator.

Returns

pandas.Series

getStoredFeatures ()

Returns the stored features if store was set to True, else it returns None.

reset ()

Resets the

Returns

None.

DOCUMENTATION OF EEGLIB.FEATURES MODULE

—

This module define the functions of the features for EEG analysis

`eeglib.features.DFA`(*data*, *fit_degree=1*, *min_window_size=4*, *max_window_size=None*, *fskip=1*,
max_n_windows_sizes=None)

Applies Detrended Fluctuation Analysis algorithm to the given data.

Parameters

data: array_like The signal.

fit_degree: int, optional Degree of the polynomial used to model de local trends. Default: 1.

min_window_size: int, optional Size of the smallest window that will be used. Default: 4.

max_window_size: int, optional Size of the biggest window that will be used. Default: `signalSize//4`

fskip: float, optional Fraction of the window that will be skiped in each iteration for each window size. Default: 1

max_n_windows_sizes: int, optional Maximum number of window sizes that will be used. The final number can be smaller once the repeated values are removed Default: `log2(size)`

Returns

float The resulting value

`eeglib.features.HFD`(*data*, *kMax=None*)

Returns the Higuchi Fractal Dimension of the signal given data.

Parameters

data: array_like signal

kMax: int, optional By default it will be `windowSize//4`.

Returns

float The resulting value

`eeglib.features.LZC`(*data*, *threshold=None*)

Returns the Lempel-Ziv Complexity (LZ76) of the given data.

Parameters

data: array_like The signal.

threshold: numeric, optional A number use to binarize the signal. The values of the signal above threshold will be converted to 1 and the rest to 0. By default, the median of the data.

References

[1]

`eeglib.features.PFD(data)`

Returns the Petrosian Fractal Dimension of the signal given in data.

Parameters

data: `array_like` Signal

Returns

float The resulting value

`eeglib.features.bandPower(spectrum, bandsLimits, freqRes, normalize=False)`

Returns the power of each band at the given index.

Parameters

spectrum: `1D arraylike` An array containing the spectrum of a signal

bandsLimits: `dict` This parameter is used to indicate the bands that are going to be used. It is a dict with the name of each band as key and a tuple with the lower and upper bounds as value.

freqRes: `float` Minimum resolution for the frequency.

normalize: `bool, optional` If True the each band power is divided by the total power of the spectrum. Default False.

Returns

dict The keys are the name of each band and the values are their power.

`eeglib.features.countSignChanges(data)`

Returns the number of sign changes of a 1D array

Parameters

data: `array_like` The data from which the sign changes will be counted

Returns

int Number of sign changes in the data

`eeglib.features.hjorthActivity(data)`

Returns the Hjorth Activity of the given data

Parameters

data: `array_like`

Returns

float The resulting value

`eeglib.features.hjorthComplexity(data)`

Returns the Hjorth Complexity of the given data

Parameters

data: `array_like`

Returns

float The resulting value

`eeglib.features.hjorthMobility(data)`

Returns the Hjorth Mobility of the given data

Parameters

data: `array_like`

Returns

float The resulting value

`eeglib.features.sampEn(data, m=2, l=1, r=None, fr=0.2, eps=1e-10)`

Returns Sample Entropy of the given data.

Parameters

data: `array_like` The signal

m: `int, optional` Size of the embedded vectors. By default 2.

l: `int, optional` Lag between elements of embedded vectors. By default 1.

r: `float, optional` Tolerance. By default `fr*std(data)`

fr: `float, optional` Fraction of `std(data)` used as tolerance. If `r` is passed, this parameter is ignored. By default, 0.2.

eps: `float, optional` Small number added to avoid infinite results. If 0 infinite results can appear. Default: `1e-10`.

Returns

float The resulting value

`eeglib.features.synchronizationLikelihood(c1, c2, m, l, w1, w2, pRef=0.05, epsilonIterations=20)`

Returns the Synchronization Likelihood between `c1` and `c2`. This is a modified version of the original algorithm.

Parameters

c1: `array_like` First signal

c2: `array_like` second signal

m: `int` Numbers of elements of the embedded vectors.

l: `int` Separation between elements of the embedded vectors.

w1: `int` Theiler correction for autocorrelation effects

w2: `int` A window that sharpens the time resolution of the Synchronization measure

pRef: `float, optional` The `pRef` param of the `synchronizationLikelihood`. Default 0.05

epsilonIterations: `int, optional` Number of iterations used to determine the value of `epsilon`. Default:20

Returns

float A value between 0 and 1. 0 means that the signal are not synchronized at all and 1 means that they are totally synchronized.

DOCUMENTATION OF EEG.LIB.PREPROCESSING MODULE

This module define the functions for preprocessing the signal data

`eeglib.preprocessing.bandPassFilter` (*data*, *sampleRate=None*, *highpass=None*, *low-*
pass=None, *order=2*)

Return the signal filtered between highpass and lowpass. Note that neither highpass or lowpass should be above `sampleRate/2`.

Parameters

data: `numpy.ndarray` The signal

sampleRate: `numeric, optional` The frequency at which the signal was recorded. By default it is the same as the number of samples of the signal.

highpass: `numeric, optional` The signal will be filtered above this value.

lowpass: `numeric, optional` The signal will be filtered bellow this value.

order: `int, optional` Butterworth

Returns

`numpy.ndarray` The signal filtered between the highpass and the lowpass

INDICES AND TABLES

Here there are diferents indices of the documentation and a search utility.

- [genindex](#)
- [modindex](#)
- [search](#)

BIBLIOGRAPHY

- [1] M. Aboy, R. Hornero, D. Abasolo and D. Alvarez, "Interpretation of the Lempel-Ziv Complexity Measure in the Context of Biomedical Signal Analysis," in IEEE Transactions on Biomedical Engineering, vol. 53, no.11, pp. 2282-2288, Nov. 2006.

PYTHON MODULE INDEX

e

- `eeglib`, [??](#)
- `eeglib.eeg`, [3](#)
- `eeglib.features`, [23](#)
- `eeglib.helpers`, [15](#)
- `eeglib.preprocessing`, [27](#)
- `eeglib.wrapper`, [21](#)

Symbols

`__init__()` (*eeglib.eeg.EEG method*), 7
`__init__()` (*eeglib.eeg.SampleWindow method*), 12
`__init__()` (*eeglib.helpers.CSVHelper method*), 15
`__init__()` (*eeglib.helpers.EDFHelper method*), 17
`__init__()` (*eeglib.helpers.Helper method*), 18
`__init__()` (*eeglib.helpers.Iterator method*), 20
`__init__()` (*eeglib.wrapper.Wrapper method*), 21

A

`addCustomFeature()` (*eeglib.wrapper.Wrapper method*), 22
`addFeatures()` (*eeglib.wrapper.Wrapper method*), 22

B

`bandPassFilter()` (*in module eeglib.preprocessing*), 27
`bandPower()` (*eeglib.eeg.EEG method*), 7
`bandPower()` (*in module eeglib.features*), 24

C

`CCC()` (*eeglib.eeg.EEG method*), 4
`countSignChanges()` (*in module eeglib.features*), 24
`CSVHelper` (*class in eeglib.helpers*), 15

D

`DFA()` (*eeglib.eeg.EEG method*), 4
`DFA()` (*in module eeglib.features*), 23
`DFT()` (*eeglib.eeg.EEG method*), 5
`DTW()` (*eeglib.eeg.EEG method*), 5

E

`EDFHelper` (*class in eeglib.helpers*), 16
`EEG` (*class in eeglib.eeg*), 3
`eeglib`
 module, 1
`eeglib.eeg`
 module, 3
`eeglib.features`
 module, 23

`eeglib.helpers`
 module, 15
`eeglib.preprocessing`
 module, 27
`eeglib.wrapper`
 module, 21
`engagementLevel()` (*eeglib.eeg.EEG method*), 8

F

`featuresNames()` (*eeglib.wrapper.Wrapper method*), 22

G

`getAllFeatures()` (*eeglib.wrapper.Wrapper method*), 22
`getBoundsForBand()` (*eeglib.eeg.EEG method*), 8
`getChannel()` (*eeglib.eeg.EEG method*), 8
`getChannel()` (*eeglib.eeg.SampleWindow method*), 12
`getFeatures()` (*eeglib.wrapper.Wrapper method*), 22
`getIndicesList()` (*eeglib.eeg.SampleWindow method*), 12
`getNames()` (*eeglib.helpers.CSVHelper method*), 15
`getNames()` (*eeglib.helpers.EDFHelper method*), 17
`getNames()` (*eeglib.helpers.Helper method*), 19
`getPairOfChannels()` (*eeglib.eeg.SampleWindow method*), 12
`getPairsIndicesList()` (*eeglib.eeg.SampleWindow method*), 13
`getSignalAtBands()` (*eeglib.eeg.EEG method*), 9
`getStoredFeatures()` (*eeglib.wrapper.Wrapper method*), 22

H

`Helper` (*class in eeglib.helpers*), 18
`HFD()` (*eeglib.eeg.EEG method*), 6
`HFD()` (*in module eeglib.features*), 23
`hjorthActivity()` (*eeglib.eeg.EEG method*), 9
`hjorthActivity()` (*in module eeglib.features*), 24
`hjorthComplexity()` (*eeglib.eeg.EEG method*), 9
`hjorthComplexity()` (*in module eeglib.features*), 24

`hjorthMobility()` (*eeglib.eeg.EEG method*), 10
`hjorthMobility()` (*in module eeglib.features*), 24

I

`Iterator` (*class in eeglib.helpers*), 19

L

`LZC()` (*eeglib.eeg.EEG method*), 6
`LZC()` (*in module eeglib.features*), 23

M

module

- `eeglib`, 1
- `eeglib.eeg`, 3
- `eeglib.features`, 23
- `eeglib.helpers`, 15
- `eeglib.preprocessing`, 27
- `eeglib.wrapper`, 21

`moveEEGWindow()` (*eeglib.helpers.CSVHelper method*), 16
`moveEEGWindow()` (*eeglib.helpers.EDFHelper method*), 17
`moveEEGWindow()` (*eeglib.helpers.Helper method*), 19

P

`PFD()` (*eeglib.eeg.EEG method*), 6
`PFD()` (*in module eeglib.features*), 24
`prepareEEG()` (*eeglib.helpers.CSVHelper method*), 16
`prepareEEG()` (*eeglib.helpers.EDFHelper method*), 17
`prepareEEG()` (*eeglib.helpers.Helper method*), 19
`prepareIterator()` (*eeglib.helpers.CSVHelper method*), 16
`prepareIterator()` (*eeglib.helpers.EDFHelper method*), 17
`prepareIterator()` (*eeglib.helpers.Helper method*), 19
`PSD()` (*eeglib.eeg.EEG method*), 7

R

`reset()` (*eeglib.wrapper.Wrapper method*), 22

S

`sampEn()` (*eeglib.eeg.EEG method*), 10
`sampEn()` (*in module eeglib.features*), 25
`SampleWindow` (*class in eeglib.eeg*), 11
`selectSignals()` (*eeglib.helpers.CSVHelper method*), 16
`selectSignals()` (*eeglib.helpers.EDFHelper method*), 18
`selectSignals()` (*eeglib.helpers.Helper method*), 19

`set()` (*eeglib.eeg.EEG method*), 10
`set()` (*eeglib.eeg.SampleWindow method*), 13
`synchronizationLikelihood()` (*eeglib.eeg.EEG method*), 11
`synchronizationLikelihood()` (*in module eeglib.features*), 25

W

`Wrapper` (*class in eeglib.wrapper*), 21