eegutils Documentation

Release ('0.0.5',)

Samuele Carcagno

March 24, 2016

Contents

Contents:

CHAPTER 1

Introduction

Author Samuele Carcagno

eegutils is a python library for extracting and processing event related potentials (ERPs) from electroencephalographic (EEG) recordings.

CHAPTER 2

eegutils – Utilities for processing EEG recordings

This module contains functions to extract and process event related potentials (ERPs) from electroencephalographic (EEG) recordings.

eegutils.**averageAverages**(*aveList*, *nSegments*)

Perform a weighted average of a list of averages. The weight of each average in the list is determined by the number of segments from which it was obtained.

Parameters **aveList** : list of dicts of 2D numpy arrays

The list of averages for each experimental condition.

nSegments : list of dicts of ints

The number of epochs on which each average is based.

Returns **weightedAve** : dict of 2D numpy arrays

The weighted averages for each condition.

nSegsSum : dict of ints

The number of epochs on which each weighted average is based.

Examples

```
>>> #simulate averages
>>> import numpy as np
>>> ave1 = {'cnd1': np.random.rand(4, 2048), 'cnd2': np.random.rand(4, 2048)}
>>> ave2 = {'cnd1': np.random.rand(4, 2048), 'cnd2': np.random.rand(4, 2048)}
>>> nSegs1 = {'cnd1': 196, 'cnd2': 200}
>>> nSegs2 = {'cnd1': 198, 'cnd2': 189}
>>> aveList = [ave1, ave2]; nSegments = [nSegs1, nSegs2]
>>> weightedAve, nSegsSum = averageAverages(aveList=aveList, nSegments=nSegments)
```

eegutils.**averageEpochs**(*rec*)

Average the epochs of a segmented recording.

Parameters **rec** : dict of 3D numpy arrays with dimensions (n_channels x n_samples x n_epochs)

The segmented recording

Returns **ave** : dict of 2D numpy arrays with dimensions (n_channels x n_samples)

The average epochs for each condition.

nSegs : dict of ints

The number of epochs averaged for each condition.

Examples

```
>>> ave, nSegs = averageEpochs(rec=rec)
```

eegutils.**baselineCorrect**(*rec*, *baselineStart*, *preDur*, *sampRate*)

Perform baseline correction by subtracting the average pre-event voltage from each channel of a segmented recording.

Parameters **rec** : dict of 3D arrays

The segmented recording.

baselineStart : float

Start time of the baseline window relative to the event onset, in seconds. The absolute value of baselineStart cannot be greater than preDur. In practice baselineStart allows you to define a baseline window shorter than the time window before the experimental event (preDur).

preDur : float

Duration of recording epoch before the experimental event, in seconds.

sampRate : int

The samplig rate of the EEG recording.

Examples

```
>>> #baseline window has the same duration of preDur
>>> baseline_correct(rec=rec, baselineStart=-0.2, preDur=0.2, sampRate=512)
>>> #now with a baseline shorter than preDur
>>> baseline_correct(rec=rec, baselineStart=-0.15, preDur=0.2, sampRate=512)
```

eegutils.**chainSegments**(*rec*, *nChunks*, *sampRate*, *start*, *end*, *baselineDur=0*, *window=None*)

Take a dictionary containing in each key a list of segments, and chain these segments into chunks of length *nChunks*. *baselineDur* is for determining what is the zero point. *start* and *end* are given with reference to the zero point. This chaining technique is used to increase the spectral resolution of FFT analyses of auditory steady-state responses.

Parameters **rec** : dict of 3D arrays

The segmented recordings for each experimental condition.

nChunks : int

The number of segments to chain together for each chunk.

sampRate : int

The EEG recording sampling rate.

start : float

Start time of the epoch segments to be chained, in seconds.

end : float

End time of the epoch segments to be chained, in seconds.

baselineDur : float

Duration of the baseline, in seconds.

Returns **eegChained** : dict of 2D arrays

The chained recordings for each experimental condition.

Examples

```
>>> chainSegments(rec, nChunks=20, sampRate=2048, start=0, end=0.5, baselineDur=0.1)
```

eegutils.**detrendEEG**(*rec*)

Remove the mean value from each channel of an EEG recording.

Parameters **rec** : dict of 2D arrays

The EEG recording.

Examples

```
>>> detrend(rec)
```

eegutils.**detrendSegmented**(*rec*)

Remove the mean value from each channel of an EEG recording.

Parameters **rec** : dict of 3D arrays

The segmented EEG recording.

Examples

```
>>> detrendSegmented(rec)
```

eegutils.**extractEventTable**(*trigChan*, *sampRate*)

Extract the event table from the EEG channel containing the trigger codes.

Parameters **trigChan** : array

The trigger channel.

sampRate : int

The EEG recording sampling rate.

Returns **eventTable** : a dictionary with the following keys

- **code** [array of ints] The trigger codes.
- **idx** [array of ints] The indexes of the trigger codes.
- **dur** [array of floats] The duration of the triggers, in seconds.

Examples

```
>>> evtTab = extractEventTable(trigChan, 2048)
```

eegutils.**filterContinuous**(*rec*, *channels*, *sampRate*, *filterType*, *nTaps*, *cutoffs*, *transitionWidth*)

Filter a continuous recording.

Parameters **rec** : 2D array

The nChannelsXnSamples array with the EEG recording.

channels : array of ints

The list of channels that should be filtered.

sampRate : int

The EEG recording sampling rate.

filterType : str {'lowpass', 'highpass', 'bandpass'}

The filter type.

nTaps : int

The number of filter taps.

cutoffs : array of floats

The filter cutoffs. If 'filterType' is 'lowpass' or 'highpass' the 'cutoffs' array should contain a single value. If 'filterType' is bandpass the 'cutoffs' array should contain the lower and the upper cutoffs in increasing order.

transitionWidth : float

The width of the filter transition region, normalized between 0-1. For a lower cutoff the nominal transition region will go from *(1-transitionWidth)*cutoff* to *cutoff*. For a higher cutoff the nominal transition region will go from cutoff to *(1+transitionWidth)*cutoff*.

Examples

```
>>> filterContinuous(rec=rec, channels=[0,1,2,3], sampRate=2048, filterType='highpass', nTaps=51
```

eegutils.**filterSegmented**(*rec*, *channels*, *sampRate*, *filterType*, *nTaps*, *cutoffs*, *transitionWidth*)

Filter a segmented recording.

Parameters **rec** : dict of 3D arrays

The segmented EEG recording.

channels : array of ints

The list of channels that should be filtered.

sampRate : int

The EEG recording sampling rate.

filterType : str {'lowpass', 'highpass', 'bandpass'}

The filter type.

nTaps : int

The number of filter taps.

cutoffs : array of floats

The filter cutoffs. If 'filterType' is 'lowpass' or 'highpass' the 'cutoffs' array should contain a single value. If 'filterType' is bandpass the 'cutoffs' array should contain the lower and the upper cutoffs in increasing order.

transitionWidth : float

The width of the filter transition region, normalized between 0-1. For a lower cutoff the nominal transition region will go from *(1-transitionWidth)*cutoff* to *cutoff*. For a higher cutoff the nominal transition region will go from cutoff to *(1+transitionWidth)*cutoff*.

Examples

```
>>> filterSegmented(rec=rec, channels=[0,1,2,3], sampRate=2048, filterType='highpass', nTaps=512
```

eegutils.**findArtefactThresh**(*rec, thresh=[100], channels=[0]*)

Find epochs with voltage values exceeding a given threshold.

Parameters **rec** : dict of 3D arrays

The segmented recording.

thresh : array of floats

The threshold value for each channel listed in *channels*.

channels = array or list of ints

The indexes of the channels to check for artefacts.

Returns **segsToReject** : array of ints

The indexes of the epochs exceeding the threshold.

Examples

```
>>> toRemove = eeg.findArtefactThresh(rec=segs, thresh=[100,60,100], channels=[0,1,2])
```

eegutils.**getFRatios**(*ffts*, *freqs*, *nSideComp*, *nExcludedComp*, *otherExclude*)

Compute signal to noise ratio (SNR) of one or more signals from a fast fourier transform (FFT) and test the SNR significance using an F-test.

Parameters **ffts** : dict

The ffts for each experimental condition. The ffts should be in the same format as returned by the *getSpectrum()* function, i.e. a dictionary with *freq* and *mag* keys.

freqs : array of floats

The frequencies of the signals.

nSideComp : int

The number of components adjacent to each side of the signal components from which to estimate the noise power. *nSideComp* above and *nSideComp* below each signal will be used for each noise-power estimate. In other words, the noise power around each signal component will be estimated from *2*nSideComp* components.

nExcludedComp: int

To avoid that spectral leaks from the signal affect the noise-power estimate, the *nExcludedComp* components just above and the *nExcludeComp* components just below the signal will not be used for estimating noise power.

otherExclude : array of ints

The frequencies of other components to exclude from the computation of the noise power. This may be useful to exclude components corresponding to distortion products generated by the signal. The *nExcludedComp* components just above and the *nExcludeComp* components just below each component in *otherExclude* will also be excluded.

Returns **res** : dict with the following keys

- **fftVals** [dict] The signal and noise power for each component and experimental condition. Each key of *fftVals* corresponds to an experimental condition. For each experimental condition there is a dictionary with keys *noisePow* and *sigPow* that list the noise and signal power for each component given in *freqs*.
- **fRatio :** The F and corresponding p-value for each component and experimental condition. Each key of *fRatio* corresponds to an experimental condition. For each experimental condition there is a dictionary with keys *F* and *pval* that list the F and p value for each component given in *freqs*.
- **compIdx** [list] The indexes of the signal frequencies in the FFT array.
- **sideBandsIdx** [list] The indexes of the noise side bands in the FFT array. A separate sublist is returned for each component specified in *freqs*.
- **excludedIdx** [list] The indexes of the components excluded from the noise side bands.
- **minSideFreq** [list] For each signal, the lowest frequency of the noise bands.
- **maxSideFreq** [list] For each signal, the highest frequency of the noise bands.

Examples

```
>>> getFRatios(ffts=ffts, freqs=[30, 75], nSideComp=30, nExcludedComp=1, otherExclude=[25, 68])
```

eegutils.**getFilterCoefficients**(*sampRate*, *filterType*, *nTaps*, *cutoffs*, *transitionWidth*)

Get the coefficients of a FIR filter. This function is used internally by eegutils.

Parameters **sampRate** : int

The EEG recording sampling rate.

filterType : str {'lowpass', 'highpass', 'bandpass'}

The filter type.

nTaps : int

The number of filter taps.

cutoffs : array of floats

The filter cutoffs. If 'filterType' is 'lowpass' or 'highpass' the 'cutoffs' array should contain a single value. If 'filterType' is bandpass the 'cutoffs' array should contain the lower and the upper cutoffs in increasing order.

transitionWidth : float

The width of the filter transition region, normalized between 0-1. For a lower cutoff the nominal transition region will go from *(1-transitionWidth)*cutoff* to *cutoff*. For a higher cutoff the nominal transition region will go from cutoff to *(1+transitionWidth)*cutoff*.

Returns **filterCoeff** : array of floats

The filter coefficients.

Examples

```
>>> getFilterCoefficients(sampRate=2048, filterType='highpass', nTaps=512, cutoffs=[30], transit
```

eegutils.**getFilterFreqResp**(*sampRate*, *filterType*, *nTaps*, *cutoffs*, *transitionWidth*, *plotResp=False*)

Get the frequency response of a eegutils filter.

Parameters **sampRate** : int

The EEG recording sampling rate

filterType : string {lowpass, highpass, bandpass}

The filter type.

nTaps : int

The number of filter taps.

cutoffs : array of floats

The filter cutoffs. If 'filterType' is 'lowpass' or 'highpass' the 'cutoffs' array should contain a single value. If 'filterType' is bandpass the 'cutoffs' array should contain the lower and the upper cutoffs in increasing order.

transitionWidth : float

The width of the filter transition region, normalized between 0-1. For a lower cutoff the nominal transition region will go from *(1-transitionWidth)*cutoff* to *cutoff*. For a higher cutoff the nominal transition region will go from cutoff to *(1+transitionWidth)*cutoff*.

plotResp : bool

Whether to plot the frequency response.

Returns **freq** : array of floats

The frequency axis.

mag : array of floats

The frequency response of the filter. This is an array of complex numbers, to get the real part use *abs(mag)*.

Examples

```
>>>  f, m = getFilterFreqResp(2048, 'highpass', 512, [30], 0.2)
```

eegutils.**getNoiseSidebands**(*componentsFreq*, *nCompSide*, *nExcludedComp*, *fftDict*, *otherExclude=None*)

Given one or more signal frequencies, get, for each signal frequency, the power in frequency bins adjacent to the signal frequency. The results can be used to estimate *local* noise in signal-to-noise-ratio computations.

Parameters **componentsFreq** : list of floats

The frequencies of the signal components.

nCompSide : int

The number of components adjacent to each side of the signal components from which to estimate the noise power. *nSideComp* above and *nSideComp* below each signal will be used for each noise-power estimate. In other words, the noise power around each signal component will be estimated from *2*nSideComp* components.

nExcludedComp : int

To avoid that spectral leaks from the signal affect the noise-power estimate, the *nExcludedComp* components just above and the *nExcludedComp* components just below the signal will not be used for estimating noise power.

FFTDict: dict with the following keys

- **mag** [array of floats] The array containing the FFT magnitude values.
- **freq** [array of floats] The array containing the FFT frequencies.

otherExclude : array of ints

The frequencies of other components to exclude from the computation of the noise power. This may be useful to exclude components corresponding to distortion products generated by the signal. The *nExcludedComp* components just above and the *nExcludedComp* components just below each component in *otherExclude* will also be excluded.

Returns **noiseBands** : list

The spectral magnitude of the noise bands. A separate sub-list is returned for each component specified in *freqs*.

noiseBandsIdx : list

The indexes of the frequency bins in *fftDict* corresponding to the noise bands. A separate sub-list is returned for each component specified in *freqs*.

idxProtect : list

The indexes of the frequency bins in *fftDict* that were excluded from the noise power computation.

Examples

```
>>> getNoiseSidebands(compIdx=[40, 44], nSideComp=30, nExcludedComp=2, FFTDict=ffts, otherExclud
```

eegutils.**getSpectrogram**(*sig*, *sampRate*, *winLength*, *overlap*, *winType*, *powerOfTwo*)

Compute the spectrogram of a 1-dimensional array.

Parameters **sig** : array of floats

The signal of which the spectrum should be computed.

sampRate : int

The sampling rate of the signal.

winLength : float

The length of the window over which to take the FFTs.

overlap : float

The percent of overlap between successive windows (useful for smoothing the spectrogram).

winType : str {'hamming', 'hanning', blackman', 'bartlett', 'none'}

The type of window to apply to the signal before computing its FFT. Choose 'none' if you don't want to apply any window.

powerOfTwo : bool

If *True sig* will be padded with zeros (if necessary) so that its length is a power of two.

Returns **spectrogram** : dict with the following keys

- **freq** [array of floats] The frequency axis.
- **time** [array of floats] The time axis.
- mag : the power spectrum.

Examples

```
>>> sig = np.random.random(512)
>>> getSpectogram(sig, 256, 'hamming')
```

eegutils.**getSpectrum**(*sig*, *sampRate*, *window*, *powerOfTwo*)

Compute the power spectrum of a 1-dimensional array.

Parameters **sig** : array of floats

The signal of which the spectrum should be computed.

sampRate : int

The sampling rate of the signal.

window : str {'hamming', 'hanning', blackman', 'bartlett', 'none'}

The type of window to apply to the signal before computing its FFT. Choose 'none' if you don't want to apply any window.

powerOfTwo : bool

If *True sig* will be padded with zeros (if necessary) so that its length is a power of two.

Returns **spectrum** : dict with the following keys

- **freq** [array of floats] The FFT frequencies.
- mag : the power spectrum.

Examples

```
>>> sig = np.random.random(512)
>>> getSpectrum(sig, 256, 'hamming')
```

eegutils.**mergeTriggersCnt**(*trigArray*, *trigList*, *newTrig*)

Take one or more triggers in trigList, and substitute them with newTrig

Parameters **trigArray** : array

The trigger channel.

trigList : array

The list of triggers that should be substituted with *newTrig*

newTrig :

The new trigger value.

Examples

```
>>> mergeTriggersCnt(trigArray, [1,2], 100)
```

eegutils.**mergeTriggersEventTable**(*eventTable*, *trigList*, *newTrig*)

Substitute the event table triggers listed in trigList with newTrig

Parameters **eventTable** : dict of int arrays

The event table

trigList : array of ints

The list of triggers to substitute

newTrig : int

The new trigger used to substitute the triggers in trigList

eegutils.**nextPowTwo**(*x*)

Compute the exponent of the closest power of two that is either equal to *x* of bigger than *x*.

Parameters **x** : numeric

Returns **y** : numeric

Examples

```
>>> nextPowTwo(7)
>>> nextPowTwo(8)
```

eegutils.**read_biosig**(*fileName*)

Wrapper of biosig4python functions for reading Biosemi BDF files.

Parameters **fileName** : string

Path of the BDF file to read

eegutils.**removeEpochs**(*rec*, *toRemove*)

Remove epochs from a segmented recording.

Parameters **rec** : dict of 3D arrays

The segmented recording

to_remove : dict of 1D arrays

List of epochs to remove for each condition

Examples

```
>>> removeEpochs(rec, toRemove)
```

eegutils.**removeSpuriousTriggers**(*eventTable*, *sentTrigs*, *minTrigDur*)
Remove from the eventTable triggers that were not actually sent.

eegutils.**rerefCnt**(*rec*, *refChannel*, *channels=None*)
Rereference channels in a continuous recording.

Parameters **rec** : array of floats

The nChannelsXnSamples array with the EEG data.

refChannel: int

The reference channel (indexing starts from zero).

channels : list of ints

List of channels to be rereferenced (indexing starts from zero).

Examples

```
>>> rerefCnt(rec=dats, refChannel=4, channels=[1, 2, 3])
```

eegutils.**rerefSegmented**(*rec*, *refChannel*, *channels=None*)
Rereference channels in a segmented recording.

Parameters **rec** : dict of 3D arrays

The segmented recording

refChannel: int

The reference channel (indexing starts from zero).

channels : list of ints

List of channels to be rereferenced (indexing starts from zero).

Examples

```
>>> rerefSegmented(rec=segs, refChannel=4, channels=[0,1])
```

eegutils.**segmentCnt**(*rec*, *eventTable*, *epochStart*, *epochEnd*, *sampRate*, *eventList=None*)
Segment a continuous EEG recording into discrete event-related epochs.

Parameters **rec: array of floats**

The nChannelsXnSamples array with the EEG data.

eventTable : dict with the following keys

- **trigs** [array of ints] The list of triggers in the EEG recording.
- **trigs_pos** [array of ints] The indexes of trigs in the EEG recording.

epochStart : float

The time at which the epoch starts relative to the trigger code, in seconds.

epochEnd : float

The time at which the epoch ends relative to the trigger code, in seconds.

sampRate : int

The sampling rate of the EEG recording.

eventList : list of ints

The list of events for which epochs should be extracted. If no list is given epochs will be extracted for all the trigger codes present in the event table.

Returns **segs** : dict of 3D arrays

The segmented recording. The dictionary has a key for each condition. The corresponding key value is a 3D array with dimensions nChannels x nSamples x nSegments

n_segs : dict of ints

The number of segments for each condition.

Examples

```
>>> segs, n_segs = eeg.segment_cnt(rec=dats, eventTable=evt_tab, epochStart=-0.2, epochEnd=0.8,
```

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

e