# philistine Documentation

***Release 0.2.0***

**Phillip M. Alday**

**Jan 13, 2023**

# Contents

This is a complete reference for everything you get when you `import philistine`.

# CHAPTER 1

# MNE-Python related functionality

| | |
|---|---|
| *philistine.mne.savgol_iaf*(raw[, picks, ...]) | Estimate individual alpha frequency (IAF). |
| *philistine.mne.attenuation_iaf*(raws[, ...]) | Estimate individual alpha frequency (IAF). |
| *philistine.mne.abs_threshold*(epochs, threshold) | Compute mask for dropping epochs based on absolute voltage threshold. |
| *philistine.mne.retrieve*(epochs, windows[, ...]) | Retrieve summarized epoch data for further statistical analysis. |
| *philistine.mne.write_raw_brainvision*(raw, ...) | Write raw data to BrainVision format. |

## 1.1 philistine.mne.savgol_iaf

philistine.mne.**savgol_iaf**(*raw*, *picks=None*, *fmin=None*, *fmax=None*, *resolution=0.25*, *average=True*, *ax=None*, *window_length=11*, *polyorder=5*, *pink_max_r2=0.9*)

Estimate individual alpha frequency (IAF).

**Parameters**

- **raw** (*instance of Raw*) – The raw data to do these estimations on.
- **picks** (*array-like of int | None*) – List of channels to use.
- **fmin** (*int | None*) – Lower bound of alpha frequency band. If None, it will be empirically estimated using a polynomial fitting method to determine the edges of the central parabolic peak density, with assumed center of 10 Hz.
- **fmax** (*int | None*) – Upper bound of alpha frequency band. If None, it will be empirically estimated using a polynomial fitting method to determine the edges of the central parabolic peak density, with assumed center of 10 Hz.
- **resolution** (*float*) – The resolution in the frequency domain for calculating the PSD.

- **average** (*bool*) – Whether to average the PSD estimates across channels or provide a separate estimate for each channel. Currently, only True is supported.
- **ax** (*instance of matplotlib Axes | None | False*) – Axes to plot PSD analysis into. If None, axes will be created (and plot not shown by default). If False, no plotting will be done.
- **window_length** (*int*) – Window length in samples to use for Savitzky-Golay smoothing of PSD when estimating IAF.
- **polyorder** (*int*) – Polynomial order to use for Savitzky-Golay smoothing of PSD when estimating IAF.
- **pink_max_r2** (*float*) – Maximum R^2 allowed when comparing the PSD distribution to the pink noise 1/f distribution on the range 1 to 30 Hz. If this threshold is exceeded, then IAF is assumed unclear and None is returned for both PAF and CoG.

**Returns** **IafEst** – Named tuple with fields for the peak alpha frequency (PAF), alpha center of gravity (CoG), and the bounds of the alpha band (as a tuple).

**Return type** instance of `collections.namedtuple` called IAFEstimate

### Notes

Based on method developed by Andrew Corcoran. In addition to appropriate software citation (Zenodo DOI or git commit), please cite:

> Corcoran, A. W., Alday, P. M., Schlesewsky, M., & Bornkessel-Schlesewsky, I. (2018). Toward a reliable, automated method of individual alpha frequency (IAF) quantification. Psychophysiology, e13064. doi:10.1111/psyp.13064

## 1.2 philistine.mne.attenuation_iaf

`philistine.mne.`**attenuation_iaf**(*raws*, *picks=None*, *fmin=None*, *fmax=None*, *resolution=0.25*, *average=True*, *ax=None*, *savgol=False*, *window_length=11*, *polyorder=5*, *flat_max_r=0.98*)

Estimate individual alpha frequency (IAF).

**Parameters**

- **raws** (*list-like of Raw*) – Two Raws to calculate IAF from difference (attenuation) in PSD from.
- **picks** (*array-like of int | None*) – List of channels to use.
- **fmin** (*int | None*) – Lower bound of alpha frequency band. If None, it will be empirically estimated using a polynomial fitting method to determine the edges of the central parabolic peak density, with assumed center of 10 Hz.
- **fmax** (*int | None*) – Upper bound of alpha frequency band. If None, it will be empirically estimated using a polynomial fitting method to determine the edges of the central parabolic peak density, with assumed center of 10 Hz.
- **resolution** (*float*) – The resolution in the frequency domain for calculating the PSD.
- **average** (*bool*) – Whether to average the PSD estimates across channels or provide a separate estimate for each channel. Currently, only True is supported.

- **ax** (*instance of matplotlib Axes | None | False*) – Axes to plot PSD analysis into. If None, axes will be created (and plot not shown by default). If False, no plotting will be done.
- **savgol** (*False | 'each' | 'diff'*) – Use Savitzky-Golay filtering to smooth PSD estimates – either applied to either each PSD estimate or to the difference (i.e. the attenuation estimate).
- **window_length** (*int*) – Window length in samples to use for Savitzky-Golay smoothing of PSD when estimating IAF.
- **polyorder** (*int*) – Polynomial order to use for Savitzky-Golay smoothing of PSD when estimating IAF.
- **flat_max_r** (*float*) – Maximum (Pearson) correlation allowed when comparing the raw PSD distributions to each other in the range 1 to 30 Hz. If this threshold is exceeded, then IAF is assumed unclear and None is returned for both PAF and CoG. Note that the sign of the coefficient is ignored.

**Returns** **IafEst** – Named tuple with fields for the peak alpha frequency (PAF), alpha center of gravity (CoG), and the bounds of the alpha band (as a tuple).

**Return type** instance of `collections.namedtuple` called IAFEstimate

### Notes

Based on method developed by Andrew Corcoran. In addition to appropriate software citation (Zenodo DOI or git commit), please cite:

> Corcoran, A. W., Alday, P. M., Schlesewsky, M., & Bornkessel-Schlesewsky, I. (2018). Toward a reliable, automated method of individual alpha frequency (IAF) quantification. Psychophysiology, e13064. doi:10.1111/psyp.13064

## 1.3 philistine.mne.abs_threshold

`philistine.mne.`**`abs_threshold`**(*epochs*, *threshold*, *eeg=True*, *eog=False*, *misc=False*, *stim=False*)

Compute mask for dropping epochs based on absolute voltage threshold.

**Parameters**

- **epochs** (*instance of Epochs*) – The epoched data to do threshold rejection on.
- **threshold** (*float*) – The absolute threshold (in *volts*) to reject at.
- **eeg** (*bool*) – If True include EEG channels in thresholding procedure.
- **eog** (*bool*) – If True include EOG channels in thresholding procedure.
- **misc** (*bool*) – If True include miscellaneous channels in thresholding procedure.
- **stim** (*bool*) – If True include stimulus channels in thresholding procedure.

**Returns** **rej** – Boolean mask for whether or not the epochs exceeded the rejection threshold at any time point for any channel.

**Return type** instance of ndarray

### Notes

More precise selection of channels can be performed by passing a 'reduced' Epochs instance from the various `picks` methods.

## 1.4 philistine.mne.retrieve

`philistine.mne.`**`retrieve`**(*epochs*, *windows*, *items=None*, *summary_fnc={'mean': <function mean>}*, ***kwargs*)

Retrieve summarized epoch data for further statistical analysis.

**Parameters**

- **`epochs`** (*`instance of Epochs`*) – The epoched data to extract windowed summary statistics from.
- **`windows`** (*`dict of tuples`*) – Named tuples defining time windows for extraction (relative to epoch-locking event). Units are dependent on the keyword argument scale_time. Default is milliseconds.
- **`summary_fnc`** (*`dict of functions`*) – Functions to apply to generate summary statistics in each time window. The keys serve as column names.
- **`items`** (*`ndarray | None`*) – Items corresponding to the individual epoch / trials (for e.g. repeated measure designs). Shape should be (n_epochs,). If None (default), then item numbers will not be included in the generated data frame.
- **`kwargs`** – Keyword arguments to pass to Epochs.to_data_frame. Particularly relevant are `scalings` and `scale_time`.

**Returns** **dat** – Long-format data frame of summarized data

**Return type** instance of pandas.DataFrame

## 1.5 philistine.mne.write_raw_brainvision

`philistine.mne.`**`write_raw_brainvision`**(*raw*, *vhdr_fname*, *events=True*)

Write raw data to BrainVision format.

**Parameters**

- **`raw`** (*`instance of Raw`*) – The raw data to do these estimations on.
- **`vhdr_fname`** (*`str`*) – Path to the EEG header file.
- **`events`** (*`boolean or ndarray`*) – If ndarry, events to write in marker file. Otherwise, boolean indicator to extract and write events from raw.

### Notes

The BrainVision format supports by-channel filter and measurement information and moreover distinguishes between hardware and software filters. MNE does neither. Currently, filter information is not exported.

Moreover BrainVision also allows for more complex trigger codes than MNE's simple integers, e.g. distinguishing on supported hardware between stimulus codes (prefixed by an S) and responses codes (prefixed by an R). MNE's numeric events are all treated as 'stimulus markers' and prefixed by an S on output.

Note however that only channels of type 'eeg','eog', 'meg' and 'misc' are exported. This follows from the observation that BrainVision recordings produced by BrainProducts devices generally only contain EEG and a few auxiliary channels. The stimulus channel is not exported as channel data, but, in line with BrainVision convention, the events array can be exported to the vmrk file. Channels marked as bad are also not exported, in line with MNE's default behavior of generally ignoring bad channels. As the current MNE readers do not do much with the channel-level annotations in the vhdr file, it is not really desireable to depend on encoding channel-type or "goodness" there. As such any information related to channel-type or badness is lost upon export.

If you really want to export unsupported datatypes or bad channels, then create a copy, mark everything as good and of type 'eeg', and export. Be aware that the metadata will have to be corrected the next time the data is read. Other options are to use the private member functions directly that write each of the constituent files (understanding that their API is not guaranteed to be stable) or use the pybv library.

In other words, a round trip import-export is a lossy operation in terms of metadata. The actual EEG recording should be losslessly preserved within the realm of floating point precision and the constraints above.

# CHAPTER 2

# General purpose utilities

| | |
|---|---|
| *philistine.invert_dict*(d) | Return an 'inverted' dictionary, swapping keys against values. |

## 2.1 philistine.invert_dict

philistine.**invert_dict**(*d*)

Return an 'inverted' dictionary, swapping keys against values.

**Parameters** **d** (*dict-like*) – The dictionary to invert

**Returns** **inv_d** – The inverted dictionary.

**Return type** dict()

### Notes

If the key-mapping is not one-to-one, then the dictionary is not invertible and a ValueError is thrown.

CHAPTER 3

# Philistine

A Python package for Phillip's helper and utility functions, especially for EEG and statistics.

## 3.1 Status

## 3.2 Overview

Philistine is a collection of hopefully useful functions in Python for statistics and analysis of EEG data using existing packages in the Python ecosystem. It is not intended to be a standalone package, but rather a convenient way to distribute manipulations that I (Phillip) find useful in my own work.

This is very much aa hobby project developed in my free time (in a language I don't use much anymore) and the API is subject to change in a rather volatile fashion as improvements, corrections, etc. are made. The idea is provide a convenient way to redistribute functions that I (Phillip) find useful. The hope is that many of these functions are eventually integrated into packages such as MNE, bambi, etc. At that point, the functions will be changed into thin wrappers for those other packages, deprecated and eventually removed.

The BV-writer functionality will likely be removed in a future release. MNE now has an *export* module, which takes advantage of [*pybv*](https://pypi.org/project/pybv/), which in turn took the good parts of the writer here and added some active maintenance.

## 3.3 Installation

Philistine requires a working Python interpreter. As of version 0.2.0, this must be at least Python 3.7 for compatibility with MNE 1.3.

Assuming a standard Python environment is installed on your machine (including pip), Philistine itself can be installed in one line using pip:

```
python -m pip install --user --upgrade philistine
```

Alternatively, if you want the bleeding edge version of the package, you can install from GitLab:

```
python -m pip install --user --upgrade  git+https://gitlab.com/palday/philistine.git
```

Dependencies should be handled automatically by pip.

## 3.4 Development

The primary hosting for this project is on GitLab, and issues should be raised there. A GitHub mirror is provided for convenience and redundancy. Pull requests can be made on either site.

## 3.5 Site Navigation

- API Reference
- genindex
- search

# Index