This is documentation for the *unstable development version* of MNE-Python, **available here (https://mne.tools/dev/install/advanced.html#using-the-development-version-of-mne-python-latest-master)**. Or, switch to documentation for the **current stable version (https://mne.tools/stable)**.

Basic analysis of an SSVEP/vSSR dataset

Example script to compute frequency spectrum and extract snr of a target frequency

We use a simple example dataset with frequency tagged visual stimulation (a.k.a. steady state visually evoked potentials, SSVEP, or visual steady-state responses, vSSR):

N=2 participants observed checkerboards patterns inverting with a constant frequency of either 12Hz of 15Hz. 10 trials of 30s length each. 32ch wet EEG was recorded.

Data format: BrainVision .eeg/.vhdr/.vmrk files organized according to BIDS standard.

Data can be downloaded at https://osf.io/7ne6y/ (https://osf.io/7ne6y/)

```
# Authors: Dominik Welke <dominik.welke@web.de>
#          Evgenii Kalenkovich <e.kalenkovich@gmail.com>
#
# License: BSD (3-clause)

import warnings
import matplotlib.pyplot as plt
import mne
import numpy as np
from mne_bids import read_raw_bids, BIDSPath
from scipy.stats import ttest_rel (https://scipy.github.io/devdocs/generated/scipy.sta
```

Load raw data

```
event_id (https://docs.python.org/3/library/stdtypes.html#dict) = {
    '12hz': 10001,
    '15hz': 10002
}

data_path (https://docs.python.org/3/library/stdtypes.html#str) = mne.datasets.ssvep.da
bids_path = BIDSPath(subject='02', session='01', task='ssvep', root=data_path (https:/

# read_raw_bids issues warnings about missing electrodes.tsv and coordsystem.json.
# These warning prevent successful building of the tutorial.
# As a quick workaround, we just suppress the warnings here.
with warnings.catch_warnings (https://docs.python.org/3/library/warnings.html#warnings
    warnings.simplefilter (https://docs.python.org/3/library/warnings.html#warnings.si
    raw = read_raw_bids(bids_path, verbose=False)
raw.load_data (../../generated/mne.io.BaseRaw.html#mne.io.BaseRaw.load_data)()
```

Out:

```
    Extracting parameters from C:\Users\evgenii\mne_data\ssvep-example-data\sub-02\ses-
    Setting channel info structure...
    Reading events from C:\Users\evgenii\mne_data\ssvep-example-data\sub-02\ses-01\eeg\
    Reading channel info from C:\Users\evgenii\mne_data\ssvep-example-data\sub-02\ses-0
    Reading 0 ... 467579  =      0.000 ...   467.579 secs...
```

Minimal preprocessing

Due to a generally high SNR in SSVEP/vSSR, typical preprocessing steps are considered optional. this doesnt mean, that a proper cleaning would not increase your signal quality!

Raw data comes with FCz recording reference, so we will apply common-average rereferencing.

Set montage

```
montage_style (https://docs.python.org/3/library/stdtypes.html#str) = 'easycap-M1'
montage (../../generated/mne.channels.DigMontage.html#mne.channels.DigMontage) = mne.ch
    montage_style (https://docs.python.org/3/library/stdtypes.html#str),
    head_size=0.095)  # head_size parameter default = 0.095
raw.set_montage (../../generated/mne.io.BaseRaw.html#mne.io.BaseRaw.set_montage)(montag
```

Set common average reference

```
raw.set_eeg_reference (../../generated/mne.io.BaseRaw.html#mne.io.BaseRaw.set_eeg_refe
```

Out:

```
EEG channel type selected for re-referencing
Applying average reference.
Applying a custom EEG reference.
```

Apply notch filtering

```
notch (https://numpy.org/devdocs/reference/generated/numpy.ndarray.html#numpy.ndarray) =
                  raw.info (../../generated/mne.Info.html#mne.Info)['line_freq'])
raw.notch_filter (../../generated/mne.io.BaseRaw.html#mne.io.BaseRaw.notch_filter)(not
```

Out:

```
Setting up band-stop filter

FIR filter parameters
---------------------
Designing a one-pass, zero-phase, non-causal bandstop filter:
- Windowed time-domain design (firwin) method
- Hamming window with 0.0194 passband ripple and 53 dB stopband attenuation
- Lower transition bandwidth: 0.50 Hz
- Upper transition bandwidth: 0.50 Hz
- Filter length: 6601 samples (6.601 sec)
```

Apply linear filtering

```
hp (https://docs.python.org/3/library/functions.html#float) = .1
lp (https://docs.python.org/3/library/functions.html#float) = 250.
raw.filter (../../generated/mne.io.BaseRaw.html#mne.io.BaseRaw.filter)(hp (https://docs
```

Out:

```
Filtering raw data in 1 contiguous segment
Setting up band-pass filter from 0.1 - 2.5e+02 Hz

FIR filter parameters
---------------------
Designing a one-pass, zero-phase, non-causal bandpass filter:
- Windowed time-domain design (firwin) method
- Hamming window with 0.0194 passband ripple and 53 dB stopband attenuation
- Lower passband edge: 0.10
- Lower transition bandwidth: 0.10 Hz (-6 dB cutoff frequency: 0.05 Hz)
- Upper passband edge: 250.00 Hz
- Upper transition bandwidth: 62.50 Hz (-6 dB cutoff frequency: 281.25 Hz)
- Filter length: 33001 samples (33.001 sec)
```

Frequency analysis

We use Welch's method for frequency decomposition, since it is really fast. You can compare it with, e.g., multitaper to get an impression of the influence on SNR. All the other methods implemented in MNE can be used as well.

Construct epochs

```
events (https://numpy.org/devdocs/reference/generated/numpy.ndarray.html#numpy.ndarray
raw.info (../../generated/mne.Info.html#mne.Info)["events"] = events (https://numpy.org
tmin (https://docs.python.org/3/library/functions.html#float), tmax (https://docs.pyth
baseline = None
epochs (../../generated/mne.Epochs.html#mne.Epochs) = mne.Epochs (../../generated/mne.
                    tmax (https://docs.python.org/3/library/functions.html#float)=tmax
```

Out:

```
    Used Annotations descriptions: ['stim/12hz', 'stim/15hz']
    Not setting metadata
    Not setting metadata
    10 matching events found
    No baseline correction applied
    0 projection items activated
```

Calculate power spectral density

```
tmin (https://docs.python.org/3/library/functions.html#float) = 0.
tmax (https://docs.python.org/3/library/functions.html#float) = 30.
fmin (https://docs.python.org/3/library/functions.html#float) = 1.
fmax (https://docs.python.org/3/library/functions.html#float) = 90.
sf (https://docs.python.org/3/library/functions.html#float) = epochs.info (../../genera

psds (https://numpy.org/devdocs/reference/generated/numpy.ndarray.html#numpy.ndarray),
    epochs (../../generated/mne.Epochs.html#mne.Epochs),
    n_fft=int(sf (https://docs.python.org/3/library/functions.html#float) * (tmax (http
    tmin (https://docs.python.org/3/library/functions.html#float)=tmin (https://docs.py
    fmin (https://docs.python.org/3/library/functions.html#float)=fmin (https://docs.py
```

Out:

```
    Loading data for 10 events and 31001 original time points ...
    0 bad epochs dropped
    Effective window size : 30.000 (s)
```

Extract SSVEP/vSSR

The function below calculates the ratio of power in the target frequency bin to average power in a set of neighbor (noise) bins. The composition of noise bins can be tweaked by two parameters:

- how many noise bins do you want?
- do you want to skip n bins directly next to the target bin?

SNR calculation function

```
def snr_spectrum(psd, noise_n_neighborfreqs=1, noise_skip_neighborfreqs=1):
    """
    Parameters
    ----------
    psd - np.array
        containing psd values as spit out by mne functions. must be 2d or 3d
        with frequencies in the last dimension
    noise_n_neighborfreqs - int
        number of neighboring frequencies used to compute noise level.
        increment by one to add one frequency bin ON BOTH SIDES
    noise_skip_neighborfreqs - int
        set this >=1 if you want to exclude the immediately neighboring
        frequency bins in noise level calculation

    Returns
    -------
    snr - np.array
        array containing snr for all epochs, channels, frequency bins.
        NaN for frequencies on the edge, that do not have enoug neighbors on
        one side to calculate snr
    """
    # Construct a kernel that calculates the mean of the neighboring frequencies
    averaging_kernel = np.concatenate (https://numpy.org/devdocs/reference/generated/nu
        np.ones (https://numpy.org/devdocs/reference/generated/numpy.ones.html#numpy.o
        np.zeros (https://numpy.org/devdocs/reference/generated/numpy.zeros.html#numpy
        np.ones (https://numpy.org/devdocs/reference/generated/numpy.ones.html#numpy.o
    averaging_kernel /= averaging_kernel.sum()

    # Calculate the mean of the neighboring frequencies by convolving with the averagi
    mean_noise = np.apply_along_axis (https://numpy.org/devdocs/reference/generated/nu
                                      axis=-1, arr=psd)

    # The mean is not defined on the edges so we will pad it with nas. The padding nee
    # dimension only so we set it to (0, 0) for the other ones.
    edge_width = noise_n_neighborfreqs + noise_skip_neighborfreqs
    pad_width = [(0, 0)] * (mean_noise.ndim - 1) + [(edge_width, edge_width)]
    mean_noise = np.pad (https://numpy.org/devdocs/reference/generated/numpy.pad.html#n

    return psd / mean_noise
```

Calculate SNR

Now we call the function to compute our snr spectrum. SNR is a relative measure: it's the ratio of power in a given frequency bin compared to a baseline - the average power in the surrounding frequency bins. Hence, we need to define some parameters for this 'baseline' - how many neighboring bins should be taken for this computation, and do we want to skip the direct neighbors (this can make sense if the stimulation frequency is not super constant, or frequency bands are very narrow).

```
snrs (https://numpy.org/devdocs/reference/generated/numpy.ndarray.html#numpy.ndarray) =
                    noise_skip_neighborfreqs=1)
```

Find frequency bin containing stimulation frequency

Ideally, this bin should should have the stimulation frequency exactly in the center.

```
stim_freq (https://docs.python.org/3/library/functions.html#float) = 12.
tmp_distlist (https://numpy.org/devdocs/reference/generated/numpy.ndarray.html#numpy.n
i_signal (https://numpy.org/devdocs/reference/arrays.scalars.html#numpy.int64) = np.wh
# could be updated to support multiple frequencies
```

Calculate SNR

Extract and average SNRs at this frequency

```
snrs_stim (https://numpy.org/devdocs/reference/generated/numpy.ndarray.html#numpy.ndar
print('average SNR at %iHz (all channels, all trials): %.3f '
      % (stim_freq (https://docs.python.org/3/library/functions.html#float), snrs_stim
```

Out:

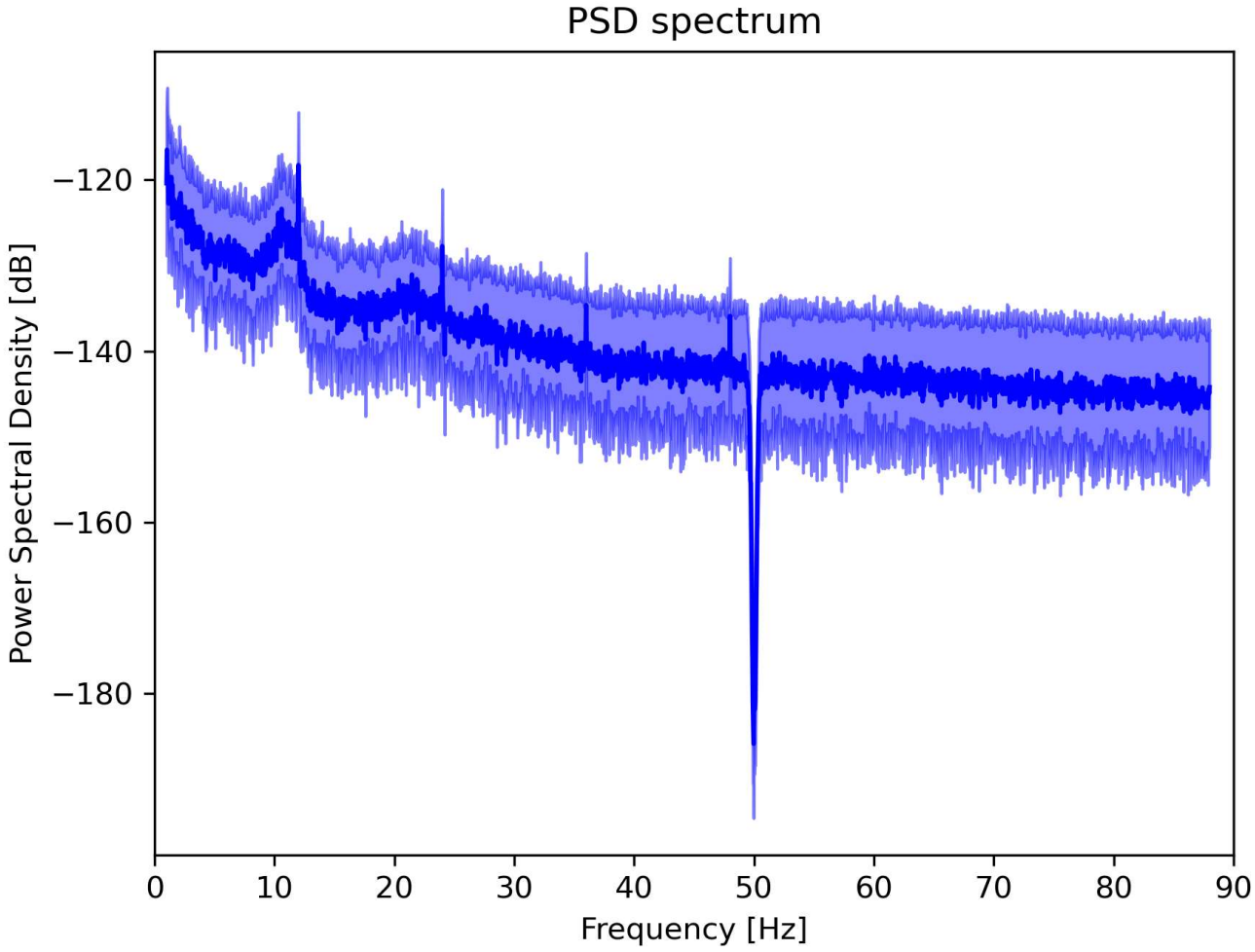
```
    average SNR at 12Hz (all channels, all trials): 9.098
```

Visualization

Plot power spectral density

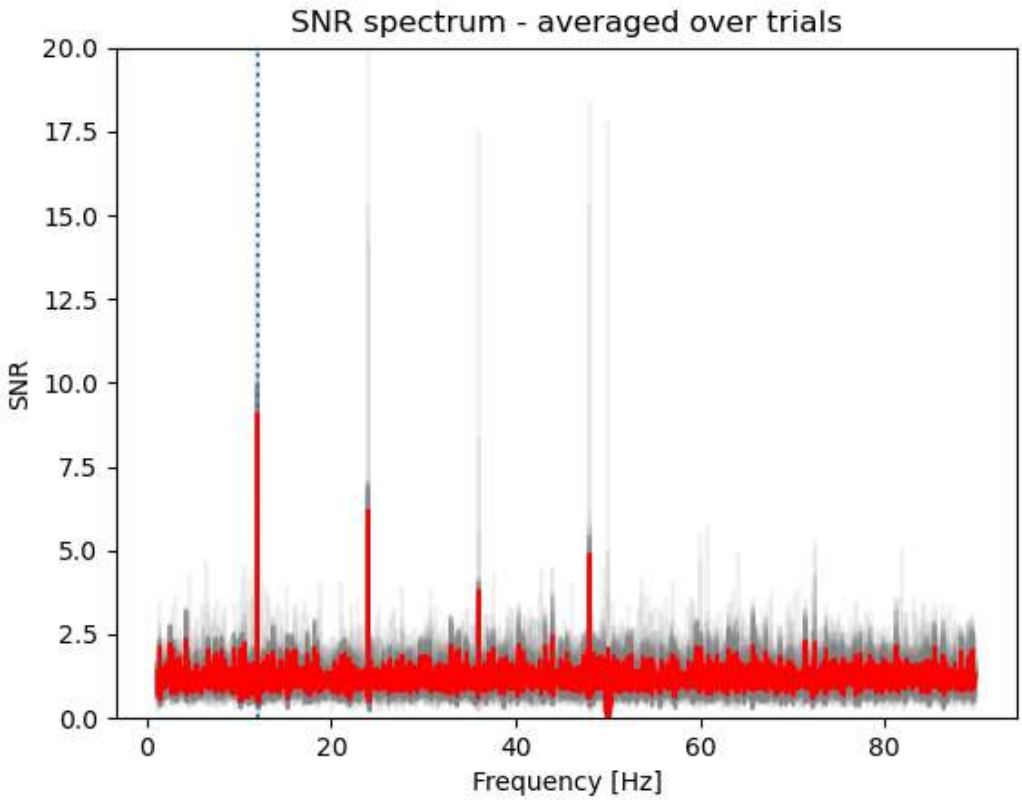
code snippet from https://martinos.org/mne/stable/auto_examples/time_frequency/plot_compute_raw_data_spectrum.html (https://martinos.org/mne/stable/auto_examples/time_frequency/plot_compute_raw_data_spectrum.html) # noqa E501

```
fig (https://matplotlib.org/api/_as_gen/matplotlib.figure.Figure.html#matplotlib.figur
rng (https://docs.python.org/3/library/stdtypes.html#range) = range(np.where (https://
            np.where (https://numpy.org/devdocs/reference/generated/numpy.where.html#nu

psds_plot (https://numpy.org/devdocs/reference/generated/numpy.ndarray.html#numpy.ndar
psds_mean (https://numpy.org/devdocs/reference/generated/numpy.ndarray.html#numpy.ndar
psds_std (https://numpy.org/devdocs/reference/generated/numpy.ndarray.html#numpy.ndarr
axes.plot (https://matplotlib.org/api/_as_gen/matplotlib.axes.Axes.plot.html#matplotli
axes.fill_between (https://matplotlib.org/api/_as_gen/matplotlib.axes.Axes.fill_betwee
                  color='b', alpha=.5)
axes.set (https://matplotlib.org/api/_as_gen/matplotlib.artist.Artist.set.html#matplot
         ylabel='Power Spectral Density [dB]')
plt.xlim (https://matplotlib.org/api/_as_gen/matplotlib.pyplot.xlim.html#matplotlib.py
fig.show (https://matplotlib.org/api/_as_gen/matplotlib.figure.Figure.html#matplotlib.
```



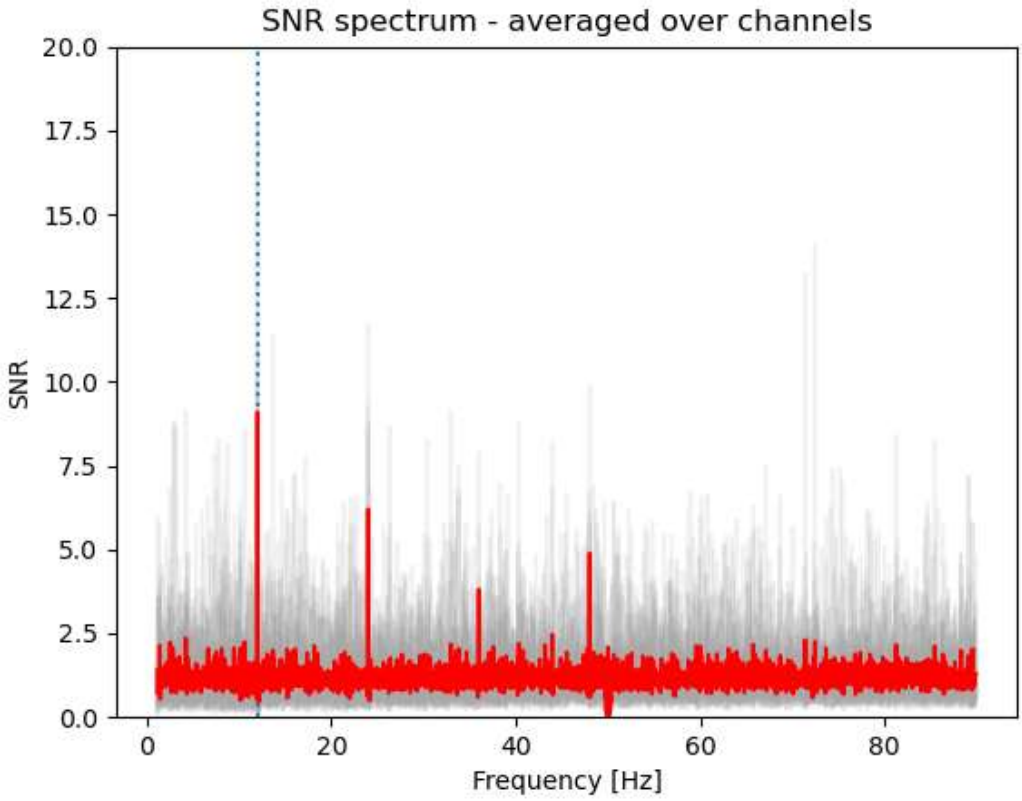
SNR spectrum - averaged over trials

```
plt.figure (https://matplotlib.org/api/_as_gen/matplotlib.pyplot.figure.html#matplotli
plt.plot (https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.py
plt.axvline (https://matplotlib.org/api/_as_gen/matplotlib.pyplot.axvline.html#matplot
plt.plot (https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.py
plt.gca (https://matplotlib.org/api/_as_gen/matplotlib.pyplot.gca.html#matplotlib.pypl
             ylabel='SNR', ylim=[0, 20])
plt.show (https://matplotlib.org/api/_as_gen/matplotlib.pyplot.show.html#matplotlib.py
```



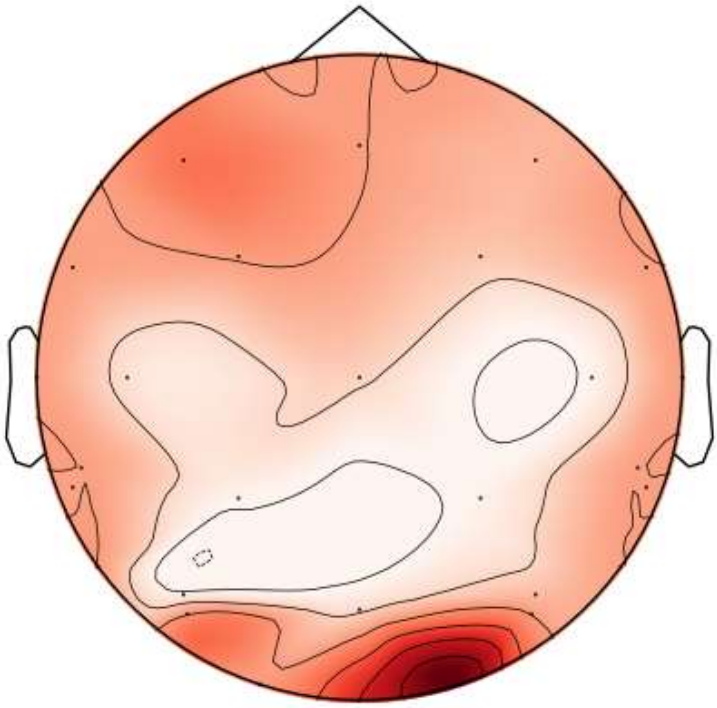
SNR spectrum - averaged over channels

```
plt.figure (https://matplotlib.org/api/_as_gen/matplotlib.pyplot.figure.html#matplotli
plt.plot (https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.py
plt.axvline (https://matplotlib.org/api/_as_gen/matplotlib.pyplot.axvline.html#matplot
plt.plot (https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.py
plt.gca (https://matplotlib.org/api/_as_gen/matplotlib.pyplot.gca.html#matplotlib.pypl
             ylabel='SNR', ylim=[0, 20])
plt.show (https://matplotlib.org/api/_as_gen/matplotlib.pyplot.show.html#matplotlib.py
```



SNR topography - grand average per channel

```
# create montage (here the default)
montage (../../generated/mne.channels.DigMontage.html#mne.channels.DigMontage) = mne.cl

# convert digitization to xyz coordinates
montage.positions (https://docs.python.org/3/library/collections.html#collections.Orde

# plot montage, if wanted
# montage.plot(show=True)

# snr topography-plot grand average (all subs, all trials)

# get grand average SNR per channel (all subs, all trials) and electrode labels
snr_grave (https://numpy.org/devdocs/reference/generated/numpy.ndarray.html#numpy.ndar

# select only present channels from the standard montage
topo_pos_grave (https://numpy.org/devdocs/reference/generated/numpy.ndarray.html#numpy
[topo_pos_grave.append (https://numpy.org/devdocs/reference/generated/numpy.ndarray.htm
topo_pos_grave (https://numpy.org/devdocs/reference/generated/numpy.ndarray.html#numpy

# plot SNR topography
f (https://matplotlib.org/api/_as_gen/matplotlib.figure.Figure.html#matplotlib.figure.F
mne.viz.plot_topomap (../../generated/mne.viz.plot_topomap.html#mne.viz.plot_topomap)(s
print("sub 2, all trials")
print("average SNR: %f" % snr_grave (https://numpy.org/devdocs/reference/generated/numpy
```



Out:

```
sub 2, all trials
average SNR: 9.097707
```

Subsetting data

For statistical comparison you probably want specific subsets of the SNR array. Either some channels, or - obviously - different trials depending on the stimuli.

- So far, one needs to define the indices of the channels / trials by hand - not nice.
- Alternatively, one can subset trials already at the epoch level using MNEs event information, and create individual PSD and SNR objects.

Here we have already subsetted trials before snr calculation (only 12Hz stimulation) and will now compare SNR in different channel subsets.

For illustration purposes, we will still subset the first 5 and last 5 of the 10 trials with 12hz stimulation.

Define ROIs

```
roi_temporal (https://docs.python.org/3/library/stdtypes.html#list) = ['T7', 'F7', 'T8
roi_aud (https://docs.python.org/3/library/stdtypes.html#list) = ['AFz', 'Fz', 'FCz',
          'C1', 'CP1', 'F2', 'FC2', 'C2', 'CP2']  # auditory roi
roi_vis (https://docs.python.org/3/library/stdtypes.html#list) = ['POz', 'Oz', 'O1', '
          'PO8', 'PO9', 'PO10', 'O9', 'O10']  # visual roi
```

Create corresponding picks

```
picks_roi_temp (https://numpy.org/devdocs/reference/generated/numpy.ndarray.html#numpy
                               exclude='bads', selection=roi_temporal (https://docs.py
picks_roi_aud (https://numpy.org/devdocs/reference/generated/numpy.ndarray.html#numpy.
                              exclude='bads', selection=roi_aud (https://docs.python.
picks_roi_vis (https://numpy.org/devdocs/reference/generated/numpy.ndarray.html#numpy.
                              exclude='bads', selection=roi_vis (https://docs.python.
```

Subset data based on ROIs

```
snrs_trialwise_roi_aud (https://numpy.org/devdocs/reference/generated/numpy.ndarray.htm
snrs_trialwise_roi_vis (https://numpy.org/devdocs/reference/generated/numpy.ndarray.htm
snrs_trialwise_temp (https://numpy.org/devdocs/reference/generated/numpy.ndarray.html#n
```

SNR for different ROIs

```
print('mean SNR (all channels, all trials) at %iHz = %.3f '
      % (stim_freq (https://docs.python.org/3/library/functions.html#float), snrs_stim
print('mean SNR (auditory ROI) at %iHz = %.3f '
      % (stim_freq (https://docs.python.org/3/library/functions.html#float), snrs_tria
print('mean SNR (visual ROI) at %iHz = %.3f '
      % (stim_freq (https://docs.python.org/3/library/functions.html#float), snrs_tria
print('mean SNR (temporal chans) at %iHz = %.3f '
      % (stim_freq (https://docs.python.org/3/library/functions.html#float), snrs_tria
```

Out:

```
    mean SNR (all channels, all trials) at 12Hz = 9.098
    mean SNR (auditory ROI) at 12Hz = 5.852
    mean SNR (visual ROI) at 12Hz = 15.676
    mean SNR (temporal chans) at 12Hz = 9.854
```

Define trial subsets

```
i_cat1_1 (https://docs.python.org/3/library/stdtypes.html#list) = [i for i in range(5)]
i_cat1_2 (https://docs.python.org/3/library/stdtypes.html#list) = [i for i in range(5,
```

Subset data trial-wise

```
snrs_trialwise_cat1_1 (https://numpy.org/devdocs/reference/generated/numpy.ndarray.htm
snrs_trialwise_cat1_2 (https://numpy.org/devdocs/reference/generated/numpy.ndarray.htm
```

SNR for different subsets of trials

```
print('mean SNR (trial subset 1) at %iHz = %.3f '
      % (stim_freq (https://docs.python.org/3/library/functions.html#float), snrs_tria
print('mean SNR (trial subset 2) at %iHz = %.3f '
      % (stim_freq (https://docs.python.org/3/library/functions.html#float), snrs_tria
```

Out:

```
    mean SNR (trial subset 1) at 12Hz = 9.569
    mean SNR (trial subset 2) at 12Hz = 8.626
```

Statistics

Just a toy t-test example to test whether:

- SNR in visual ROI is significantly different from full scalp montage at $p < 0.05$
- SNR in first 5 trials does not significantly differ from 5 last trials

Compare SNR in ROIs after averaging over channels

```
tstat_roi = ttest_rel (https://scipy.github.io/devdocs/generated/scipy.stats.ttest_rel
                      snrs_stim (https://numpy.org/devdocs/reference/generated/numpy.n
print("trial-wise SNR in visual ROI is significantly different from full scalp"
      " montage: t = %.3f, p = %f" % tstat_roi)
```

Out:

```
trial-wise SNR in visual ROI is significantly different from full scalp montage: t
```

Compare SNR in subsets of trials after averaging over channels

```
tstat_trials = ttest_ind (https://scipy.github.io/devdocs/generated/scipy.stats.ttest_
                         snrs_trialwise_cat1_2 (https://numpy.org/devdocs/reference/ge
print("trial-wise SNR in trial subset 1 is NOT significantly different from"
      " trial subset 2: t = %.3f, p = %f" % tstat_trials)
```

Out:

```
trial-wise SNR in trial subset 1 is NOT significantly different from trial subset 2
```

Total running time of the script: (0 minutes 3.465 seconds)

Download Python source code: ssvep.py
(../../_downloads/2918e3228a4ec8264ae558bda52e31f5/ssvep.py)

Download Jupyter notebook: ssvep.ipynb
(../../_downloads/449b6fa2c5caaae95b5c9fd8fad687a5/ssvep.ipynb)

Gallery generated by Sphinx-Gallery (https://sphinx-gallery.github.io)

(https://www.massgeneral.org/) ... paris.fr/) institute.org/) juelich.de/)

(https://www.tu-ilmenau.de/) (https://bids.berkeley.edu/) (https://www.au.dk/) (https://www.uni-

ilmenau.de/) graz.at/)

© Copyright 2012-2020, MNE Developers. Last updated on 2020-03-27.