frontiers

Group analysis in MNE-Python of evoked response from a tactile stimulation paradigm: a pipeline for reproducibility at every step of processing, going from individual sensor space representations to an across-group source space representation

Lau M. Andersen[1*]

[1]Clinical Neuroscience, Karolinska Institute (KI), Sweden

Submitted to Journal:
Frontiers in Neuroscience

Specialty Section:
Brain Imaging Methods

Article type:
Protocols Article

Manuscript ID:
314807

Received on:
27 Sep 2017

Frontiers website link:
www.frontiersin.org

Conflict of interest statement

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest

Author contribution statement

LMA is responsible for all content in the article, scripts and data

Keywords

MEG, analysis pipeline, MNE-Python, minimum norm estimate (MNE), Tactile Expectations, Group analysis, good practice

Abstract

Word count: 350

An important aim of an analysis pipeline for magnetoencephalographic data is that it allows for the researcher spending maximal effort on making the statistical comparisons that will answer the questions of the researcher, while in turn spending minimal effort on the intricacies and machinery of the pipeline.
I here present a set of functions and scripts that allow for setting up a clear, reproducible structure for separating raw and processed data into folders and files such that minimal effort can be spend on: 1) double-checking that the right input goes into the right functions; 2) making sure that output and intermediate steps can be accessed meaningfully; 3) applying operations efficiently across groups of subjects; 4) re-processing data if changes to any intermediate step are desirable.
Applying the scripts requires only general knowledge about the Python and the Bash languages.
The data analysed are neural responses to tactile stimulations of the right index finger in a group of twenty healthy participants acquired from an Elekta Neuromag System. Two analyses are presented: going from individual sensor space representations to respectively an across-group sensor space representation and an across-group source space representation.
The processing steps covered for the first analysis are filtering the raw data, finding events of interest in the data, epoching data based on events of interest, finding and removing independent components related to eye blinks, eye movements and heart beats, calculating participants' individual evoked responses by averaging over epoched data and calculating a grand average sensor space representation over participants. The second analysis starts from the participants' individual evoked responses and covers: estimating noise covariance, creating a forward model, creating an inverse operator, estimating distributed source activity on the cortical surface using a minimum norm procedure, morphing those estimates onto a common cortical template and calculating the patterns of activity that are statistically different from baseline. These cover importing magnetic resonance images, segmenting the brain, estimating boundaries between different tissue layers, making fine-resolution scalp surfaces for facilitating co-registration, creating source spaces and creating volume conductors for each subject.
Plotting functions that save relevant figures are present for all processing steps.

Ethics statements

(Authors are required to state the ethical considerations of their study in the manuscript, including for cases where the study was exempt from ethical approval procedures)

Does the study presented in the manuscript involve human or animal subjects: Yes

Please provide the complete ethics statement for your manuscript. Note that the statement will be directly added to the manuscript file for peer-review, and should include the following information:

- Full name of the ethics committee that approved the study
- Consent procedure used for human participants or for animal owners
- Any additional considerations of the study in cases where vulnerable populations were involved, for example minors, persons with disabilities or endangered animal species

As per the Frontiers authors guidelines, you are required to use the following format for statements involving human subjects:
This study was carried out in accordance with the recommendations of 'name of guidelines, name of committee' with written informed consent from all subjects. All subjects gave written informed consent in accordance with the Declaration of Helsinki. The protocol was approved by the 'name of committee'.
For statements involving animal subjects, please use:

This study was carried out in accordance with the recommendations of 'name of guidelines, name of committee'. The protocol was approved by the 'name of committee'.

If the study was exempt from one or more of the above requirements, please provide a statement with the reason for the exemption(s).
Ensure that your statement is phrased in a complete way, with clear and concise sentences.

The experiment was approved by the local ethics committee, Regionala etikprövningsnämnden i Stockholm. Both written and oral consent were obtained from all subjects.

Group analysis in MNE-Python of evoked response from a tactile stimulation paradigm: a pipeline for reproducibility at every step of processing, going from individual sensor space representations to an across-group source space representation

Lau M. Andersen[1*]

[1] *NatMEG, Department of Clinical Neuroscience, Karolinska Institutet, Nobels väg 9, 171 77 Stockholm, Sweden*

* Corresponding author:

Email: lau.moller.andersen@ki.se

Abstract

An important aim of an analysis pipeline for magnetoencephalographic data is that it allows for the researcher spending maximal effort on making the statistical comparisons that will answer the questions of the researcher, while in turn spending minimal effort on the intricacies and machinery of the pipeline.

I here present a set of functions and scripts that allow for setting up a clear, reproducible structure for separating raw and processed data into folders and files such that minimal effort can be spend on: 1) double-checking that the right input goes into the right functions; 2) making sure that output and intermediate steps can be accessed meaningfully; 3) applying operations efficiently across groups of subjects; 4) re-processing data if changes to any intermediate step are desirable.

Applying the scripts requires only general knowledge about the Python and the Bash languages.

The data analysed are neural responses to tactile stimulations of the right index finger in a group of twenty healthy participants acquired from an Elekta Neuromag System. Two analyses are presented: going from individual sensor space representations to respectively an across-group sensor space representation and an across-group source space representation.

The processing steps covered for the first analysis are filtering the raw data, finding events of interest in the data, epoching data based on events of interest, finding and removing independent components related to eye blinks, eye movements and heart beats, calculating participants' individual evoked responses by averaging over epoched data and calculating a grand average sensor space representation over participants. The second analysis starts from the participants' individual evoked responses and covers: estimating noise covariance, creating a forward model, creating an inverse operator, estimating distributed source activity on the cortical surface using a minimum norm procedure, morphing those estimates onto a common cortical template and calculating the patterns of activity that are statistically different from baseline. These cover importing magnetic resonance images, segmenting the brain, estimating boundaries between different tissue layers, making fine-resolution scalp surfaces for facilitating co-registration, creating source spaces and creating volume conductors for each subject.

Plotting functions that save relevant figures are present for all processing steps.

Introduction

When conducting a magnetoencephalography (MEG) study that includes questions about how different experimental factors compare, it will almost always be necessary to include a group level analysis. Single subject analyses are only meaningful for clinical patients. For the present moment only epilepsy investigation is carried on single subjects. Despite the fact that most studies rely on group level comparisons, almost all tutorials are based on single subject analyses. In the current paper, part of a special issue devoted to group analysis pipelines, I try to remedy this for anyone fancying using the MNE-python (Gramfort et al., 2013) analysis package.

The basic idea of the current group pipeline is to set up a structure that allows for:

1. Dividing output files into folders belonging to the respective subjects and recordings
2. Applying an operation across a group of subjects
3. (Re)starting the analysis at any intermediate point by saving output for each intermediate point
4. Plotting the results in a way that allows for changing the figures in a principled, but flexible manner

The neuroscientific experiment

Since the focus is on how to conduct a group analysis, the neuroscientific questions answered with the pipeline are neither novel nor interesting. The focus is rather on the pipeline, which can facilitate other experimenters' research, so that they efficiently can answer their own novel and interesting questions. The reserved digital object identifier for the data repository, where data for this experiment and scripts for the pipeline can be freely downloaded is: 10.5281/zenodo.998518.

Subjects

Twenty participants volunteered to take part in the experiment (eight males, twelve females, Mean Age: 28.7 y; Minimum Age: 21; Maximum Age: 47). The experiment was approved by the local ethics committee, Regionala etikprövningsnämnden i Stockholm. Both written and oral consent were obtained from all subjects.

Paradigm

The paradigm is based on building up tactile expectations by rhythmic tactile stimulations. These tactile expectations are every now and then violated by omitting otherwise expected stimulations (Figure 1). The inter-stimulus intervals was 3000 ms. Around every twenty-five trials, and always starting after an omission, periods of non-stimulation occurred that would last 15 s. The first six seconds worked as a wash-out period, and the remaining nine seconds were cut into three epochs of non-stimulation. There are thus nine trigger values in the data responding to nine different kinds of events (Table 1).

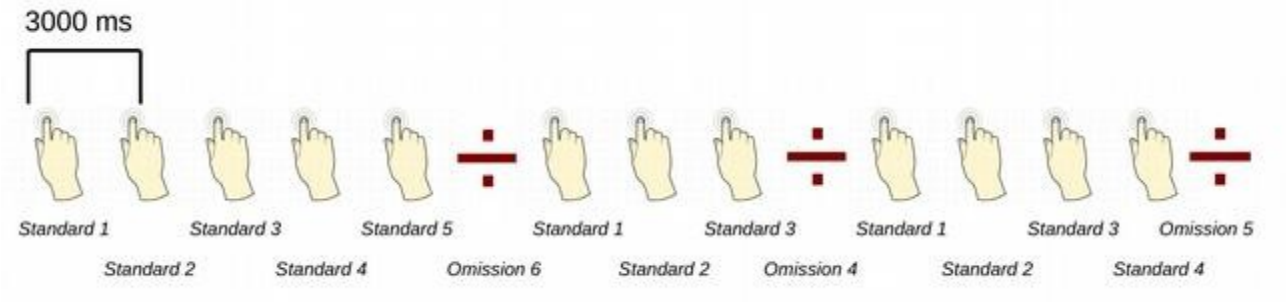

Figure 1: An example sequence of the experimental paradigm is shown. The annotations on the bottom show the coding used throughout for the different events of interest. Stimulations happened at a regular pace, every three seconds. When omissions occurred, there were thus six seconds between two consecutive stimulations.

During the stimulation procedure, participants were watching an unrelated nature programme with sound being fed through sound tubes into the ears of participants at approximately 65 dB, rendering the tactile stimulation completely inaudible. Participants were instructed to pay full attention to the movie and no attention to the stimulation of their finger. In this way, expectations should be mainly stimulus driven, and thus not cognitively driven or attention driven.

Table 1: Mapping of trigger values and annotated events

Trigger value	Annotation	Notes	Number of trials
1	*Standard 1*	First stimulation	~200
2	*Standard 2*	Second stimulation	~200
3	*Standard 3*	Third stimulation	~200
4	*Standard 4*	Fourth stimulation	~135
5	*Standard 5*	Fifth stimulation	~66
13	*Omission 4*	Omission following third stimulation	~66
14	*Omission 5*	Omission following fourth stimulation	~66
15	*Omission 6*	Omission following fifth stimulation	~66
21	*Non-Stimulation*	Absence of stimulation outside the rhythmic stimulation sequences	~130

An analysis of evoked responses will be carried out. It is well know that stimulation of the finger evokes (at least) two evoked responses, the first after ~60 ms and the second after ~135 ms. The first localizes to contralateral primary somatosensory cortex and the second to bilateral secondary somatosensory cortex. The specific parameters going into the analysis will become apparent in the analysis steps below.

Preparation of subjects

In preparation for the MEG-measurement each subject had their head shape digitized using a Polhemus Fastrak. Three fiducial points, the nasion and the left and right pre-auricular points, were digitized along with the positions of four head-position indicator coils (HPI-coils). Furthermore, about 200 extra points digitizing the head shape of each subject were acquired.

Acquisition of data

Data was sampled on an Elekta TRIUX system at a sampling frequency of 1000 Hz and on-line low-pass and high-pass filtered at 330 Hz and 0.1 Hz respectively. The data were first MaxFiltered (-v2.2) (Taulu and Simola, 2006), movement corrected and line-band filtered (50 Hz). MaxFiltering was done with setting the coordinate frame to the head coordinates, setting the origin of the head to (0, 0, 40 mm), setting the order of the inside expansion to 8, setting the order of the outside expansion to 3, enabling automatic detection of bad channels, doing a temporal Signal Space Separation (tSSS) with a buffer length of 10 s and a correlation limit of 0.980. Calibration adjustment and cross-talk corrections were based on the most recent calibration adjustment and cross-talk correction performed by the certified Elekta engineers maintaining the system.

Goal of analysis

The goal of the analysis is to make a statistical appraisal of what activity evoked from the stimulation of the finger is interesting to consider. To meet this goal, the following are necessary: (1) evoked responses from each subject's raw data are extracted. (2) volume conductors and forward models based on the individuals MRIs. (3) minimum norm estimates for each subject (4) statistics across the events based on the individual source reconstructions. The whole analysis pipeline for each subject is shown in Figure 2.

Conventions

<variable> will be used to refer to the variable called "variable".

function will be used to refer to the function called "function".

[parameter] will be used to the parameter called "parameter".

script will be used to refer to the script called "script".

Requirements

The packages in Table 1 are required to run the scripts, and the versions listed are the ones that have been used to test the scripts.

Table 2: packages, their purposes and origins, that are necessary for the pipeline.

PACKAGES	PURPOSES	ORIGIN	VERSION
mne	Analysing MEG data	Anaconda	0.14.1
numpy	Easing numerical operations	Anaconda	1.9.2
os	Interacting with operating system	Anaconda	from python 2.7.11
matplotlib	Enabling MATLAB-like plotting	Anaconda	1.4.3
scipy	Getting statistical functions and distributions	Anaconda	0.15.1
mayavi	Plotting 3D-plots	Anaconda	4.4.0

Code

General structure of the code

The idea behind this pipeline is that each processing step can be run independently of what is in the workspace of the python interpreter as long as the appropriate processing step has been applied once earlier. To ascertain this almost all the functions begins with loading the appropriate data and by saving the processed data.

MNE-Python functions are used to do the actual operations. The functions supplied in this pipeline mostly serve as convenience functions that loads the right data, processes it and finally saves it so it can be loaded for the next processing step.

Structure of pipeline.py

This is the main script, which is used to designate which operations should be run on the MEG data. The pipeline script is ordered into five blocks of code: Imports (Code Snippet 1), Paths (Code Snippet 2), Operations (Code Snippet 3), Parameters (Code Snippet 4), and the Processing Loop. It can be found one directory up from <script_path> (Code Snippet 1).

Imports

This sets the home folder <home_path>, which should to be changed to the user's home folder and imports necessary packages. Also make sure that the path to the scripts <script_path> points to the appropriate path where the below scripts can be found (Code Snippet 1). Finally also set the project name <project_name> to the folder where your analysis is stored.


```
#============================================================================
# SET HOME PATH
#%%==========================================================================

home_path = '/home/lau/' ## change this according to needs

#============================================================================
# IMPORTS
#%%==========================================================================

from os.path import join
from os import chdir
project_name = 'analyses/omission_frontiers_BIDS-MNE-Python/'
script_path = home_path + project_name + \
                    'scripts/python/analysis_functions_frontiers/'
chdir(script_path)
import operations_functions as operations
import io_functions as io
import plot_functions as plot
```

Code Snippet 1: Importing packages necessary for the pipeline.

Input/Output

The file *io_functions.py* is a set of functions that load and save operational steps with a consistent naming structure. These need not be called from *pipeline.py*, since everything is taken care of in the appropriate operations (Code Snippet 3).

Operations

The file *operations_functions.py* is a set of functions that uses MNE-Python functions to apply the actual operations that are set with the pipeline script. These are set by the operations dictionary ([operations_to_apply], (Code Snippet 3))

Plotting

The file *plot_functions.py* is a set of convenience functions used for making a subset of possible plots. If [save_plots] is set to <True>, whatever is plotted will be saved in the given subject's figure directory (see [figures_path]). Since there are many variations on what plots one might want to create, I have included only very general plot functions that users can modify according to their own needs.

Paths

This sets the paths according to the structure of the downloadable data. <subjects_to_run> can be set to only a subset of the subjects (Code Snippet 2).

```
#============================================================================
# PATHS
#%%==========================================================================

data_path = join(home_path, project_name, 'data/')
subjects_dir = join(home_path, project_name + '/data/FreeSurfer/')
name = 'oddball_absence'
save_dir_averages = data_path + 'grand_averages/'
figures_path = join(home_path, project_name, 'figures/')

subjects = [
                    'sub-01',
                    'sub-02',
                    'sub-03',
                    'sub-04',
                    'sub-05',
                    'sub-06',
                    'sub-07',
                    'sub-08',
                    'sub-09',
                    'sub-10',
                    'sub-11',
                    'sub-12',
                    'sub-13',
                    'sub-14',
                    'sub-15',
                    'sub-16',
                    'sub-17',
                    'sub-18',
                    'sub-19',
                    'sub-20'
                ]
subjects_to_run = (None, None) ## means all subjects
#subjects_to_run = (0, 1) # subject indices to run, if you don't want to run all
```

Code Snippet 2: Setting up the paths for structuring the data

Operations

The operations block contains a dictionary with all the operations you can apply to the downloadable data. All values should be Boolean, meaning that they should be set to either True (1) or False (0). The appropriate operations for each values set to True will be applied to all subjects. The keys, e.g. "filter_raw", correspond one-to-one in name with functions in *operations_functions* imported above (Code Snippet 1), except for keys that start with "plot_". They correspond one-to-one with functions in *plot_functions* (Code Snippet 1). Make sure to run "populate_data_directory" before all the others. This will create all the necessary paths for the current analysis (Code Snippet 3).

```
#============================================================================
# OPERATIONS
#%%============================================================================
operations_to_apply = dict(

        ## OS commands

        populate_data_directory=0,

        ## WITHIN SUBJECT

        ## sensor space operations
        filter_raw=0,
        find_events=0,
        epoch_raw=0,
        run_ica=0,
        apply_ica=0,
        get_evokeds=0,

        ## source space operations
        create_forward_solution=0,
        estimate_noise_covariance=0,
        create_inverse_operator=0,
        source_estimate=0,
        morph_to_fsaverage=0,

        ## BETWEEN SUBJECTS

        ## compute grand averages
        grand_averages_evokeds=0, # sensor space
        average_morphed_data=0, # source space

        ## PLOTTING

        ## plotting sensor space (within subject)
        plot_maxfiltered=0,
        plot_filtered=0,
        plot_power_spectra=0,
        plot_ica=0,
        plot_epochs_image=0,
        plot_evokeds=0,
        plot_butterfly_evokeds=0,

        ## plotting source space (within subject)
        plot_transformation=0,
        plot_source_space=0,
        plot_noise_covariance=0,
        plot_source_estimates=0,

        ## plotting sensor space (between subjects)
        plot_grand_averages_evokeds=0,
        plot_grand_averages_butterfly_evokeds=0,

        ## plotting source space (between subjects)
        plot_grand_averages_source_estimates=0,

        ## statistics in source space
        statistics_source_space=0,

        ## plot source space with statistics mask
        plot_grand_averages_source_estimates_cluster_masked=0
    )
```

Code Snippet 3: Dictionary of operations that can be applied to the data. The value associated with each key (e.g. “filter_raw”) is a boolean, i.e. either True (1) or False (0).

Parameters

The variables here (Code Snippet 3) go into the functions as parameters that the comments above them associate them with, e.g. <lowpass> goes as a parameter into ***filter_raw*** (Code Snippet 5). Preset is a number of bad channels. <overwrite>, allows for making sure that overwriting is only done when explicitly requested, while <save_plots> determines whether plots are saved.

```
#==============================================================================
# PARAMETERS
#%%============================================================================
## should files be overwritten
overwrite = False ## this counts for all operations below that save output
save_plots = True ## should plots be saved

## raw
lowpass = 70 ## Hz

bad_channels_dict = dict()
bad_channels_dict[subjects[0]] = []
bad_channels_dict[subjects[1]] = []
bad_channels_dict[subjects[2]] = []
bad_channels_dict[subjects[3]] = []
bad_channels_dict[subjects[4]] = []
bad_channels_dict[subjects[5]] = []
bad_channels_dict[subjects[6]] = ['MEG0111', 'MEG0121']
bad_channels_dict[subjects[7]] = ['MEG1411', 'MEG1421', 'MEG2121']
bad_channels_dict[subjects[8]] = ['MEG1531', 'MEG1541', 'MEG1711', 'MEG0141']
bad_channels_dict[subjects[9]] = []
bad_channels_dict[subjects[10]] = []
bad_channels_dict[subjects[11]] = []
bad_channels_dict[subjects[12]] = ['MEG0111', 'MEG0121']
bad_channels_dict[subjects[13]] = ['MEG0111', 'MEG0121', 'MEG0141']
bad_channels_dict[subjects[14]] = []
bad_channels_dict[subjects[15]] = []
bad_channels_dict[subjects[16]] = ['MEG0111', 'MEG0121']
bad_channels_dict[subjects[17]] = []
bad_channels_dict[subjects[18]] = []
bad_channels_dict[subjects[19]] = []

## events
adjust_timeline_by_msec = 41 ## delay to stimulus

## epochs
stim_channel = 'STI101'
min_duration = 0.002 # s
event_id = dict(standard_1=1, standard_2=2,
                standard_3=3, standard_4=4, standard_5=5,
                omission_4=13, omission_5=14, omission_6=15,
                non_stimulation=21)
tmin = -0.200 # s
tmax = 1.000 # s
baseline = (None, 0) # from tmin to 0
reject = dict(grad=400e-12, mag=4e-12) # T/cm and T
decim = 1 ## downsampling factor

## source reconstruction
method = 'dSPM'

## grand averages
   ## empty containers to the put the single subjects data in
evoked_data_all = dict(standard_1=[], standard_2=[], standard_3=[],
                       standard_4=[], standard_5=[], omission_4=[],
                       omission_5=[], omission_6=[], non_stimulation=[])
morphed_data_all = evoked_data_all.copy()

## plotting
mne_evoked_time = 0.056 ## s

## statistics
independent_variable_1 = 'standard_3'
independent_variable_2 = 'non_stimulation'
time_window = (0.050, 0.060)
n_permutations = 10000 ## specify as integer

## statistics plotting
p_threshold = 1e-15 ## 1e-15 is the smallest it can get for the way it is coded
```

Code Snippet 4: Parameters that need to be set for the operations applied.

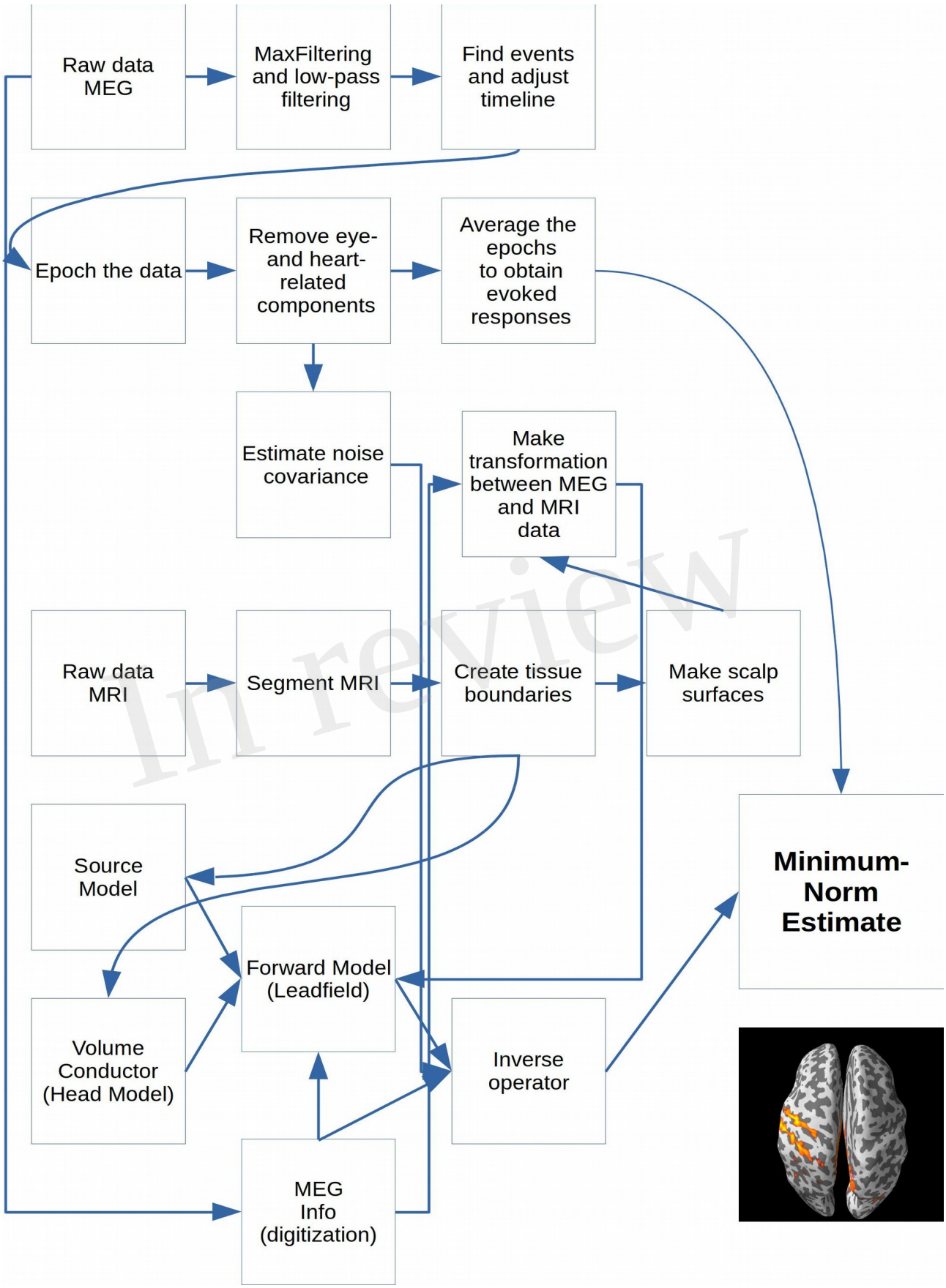

Figure 2: Cookbook for performing the Minimum-Norm Estimates for a single subject.

Applying the operations

To reiterate: all functions mentioned below come from the script: *operations_functions.py.* They are dependent on the input/output-functions from *io_functions.py* that are always from within the *operations_functions.py*. Which processing steps depends on what dictionary keys in <operations_to_apply> in *pipeline.py* are set to True. The user need only to change *pipeline.py* to apply the functions described herein. The functions *operations_functions.py* and *io_functions.py* should not be changed, but more functions can be added for needs not covered in this protocol.

Preprocessing the MEG data

Dependencies

This part is only dependent on MNE-Python. All data plotted for single subjects is from subject *sub-01*.

MaxFilter

Since the MaxFilter software is proprietary software we do not expect everyone to have it, and thus the MaxFiltered data will be the starting point of the analysis from the MEG side.

Read MaxFiltered data and low-pass filter

Use ***filter_raw*** (Code Snippet 5) to read in the data and low-pass filter it according to [lowpass]. Three parameters, [name, save_dir, overwrite] occur for the first time here and are set by the corresponding variables <name, save_dir, overwrite> in *pipeline.py.* They determine the prepending name (oddball_absence), the path to which it should be saved, and finally whether it should be overwritten or not (True/False). Both the MaxFiltered and the low-pass filtered data can be plotted. This is done, respectively, with ***plot_maxfiltered*** and ***plot_filtered***. The power spectra for the raw data can be plotted with ***plot_power_spectra***.


```
def filter_raw(name, save_dir, lowpass, overwrite):

    filter_name = name  + filter_string(lowpass) + '-raw.fif'
    filter_path = join(save_dir, filter_name)
    if overwrite or not isfile(filter_path):

        raw = io.read_maxfiltered(name, save_dir)
        raw.filter(None, lowpass)

        filter_name = name  + filter_string(lowpass) + '-raw.fif'
        filter_path = join(save_dir, filter_name)
        raw.save(filter_path, overwrite=True)

    else:
        print('raw file: ' + filter_path + ' already exists')
```

Code Snippet 5: The function for filtering the raw data

Find events of interest and adjust timeline

Use ***find_events*** (Code Snippet 6) to find the events in the low-pass filtered data file and to adjust the events by the delay between the trigger and the actual event (the blowing up of the membrane). [stim_channel] and [min_duration] are used to set the stimulus channel and the minimum duration of an event in seconds, which is also their normal behaviour in MNE-Python. Events shorter than that are regarded as spurious and not included. [adjust_timeline_by_msec] is adjusting the events by the measure delay (41 ms).

```
def find_events(name, save_dir, stim_channel, min_duration,
            adjust_timeline_by_msec, lowpass, overwrite):

    events_name = name + '-eve.fif'
    events_path = join(save_dir, events_name)
    if overwrite or not isfile(events_path):

        raw = io.read_filtered(name, save_dir, lowpass)
        events = mne.find_events(raw, stim_channel, min_duration=min_duration)
        events[:, 0] = [ts + np.round(adjust_timeline_by_msec * 10**-3 * \
                raw.info['sfreq']) for ts in events[:, 0]]

        mne.event.write_events(events_path, events)

    else:
        print('event file: '+ events_path + ' already exists')
```

Code Snippet 6: The function for finding the events in the raw files. This function also adjusts the timeline for the events for the delay between the trigger and the actual event.

Epoch the raw data files

The parameters [event_id, tmin, tmax, baseline, reject, bad_channels, decim] all serve their normal purpose in MNE-Python. A list of subject-specific bad channels is passed to ***epochs_raw*** (Code Snippet 7) by [bad_channels], which have been filled in in <bad_channels_dict>. For faster processing of the pipeline [decim] can be set to a higher value to downsample the data.

```
def epoch_raw(name, save_dir, lowpass, event_id, tmin, tmax,
          baseline, reject, bad_channels, decim, overwrite):

    epochs_name = name + filter_string(lowpass) + '-epo.fif'
    epochs_path = join(save_dir, epochs_name)
    if overwrite or not isfile(epochs_path):

        events = io.read_events(name, save_dir)
        raw = io.read_filtered(name, save_dir, lowpass,)
        raw.info['bads'] = bad_channels
        picks = mne.pick_types(raw.info, meg=True, eog=True, ecg=True,
                           exclude='bads')

        epochs = mne.Epochs(raw, events, event_id, tmin, tmax, baseline,
                        reject=reject, preload=True, picks=picks,
                        decim=decim)

        epochs.save(epochs_path)

    else:
        print('epochs file: '+ epochs_path + ' already exists')
```

Code Snippet 7: The function for epoching the raw data, defining events, time before trigger, time after trigger, what to use as the baselining period, rejection threshold, which channels are bad and by which factor to decimate (downsample) the data.

Run independent component analysis (ICA)

Use ***run_ica*** to estimate the independent components that explain the data the best, using the "fastica" algorithm (Hyvärinen, 1999). Epochs are then created that contain the electrooculographic- and electrocardiographic-related signals (eye blinks, eye movements and heart beats). Next step is finding the indices for the components that correlate with eye blinks, eye movements and heart beats. Finally, these components are removed from the ICA-solution, and the solution is saved. The removed components can be plotted with ***plot_ica*** (Figure 3).

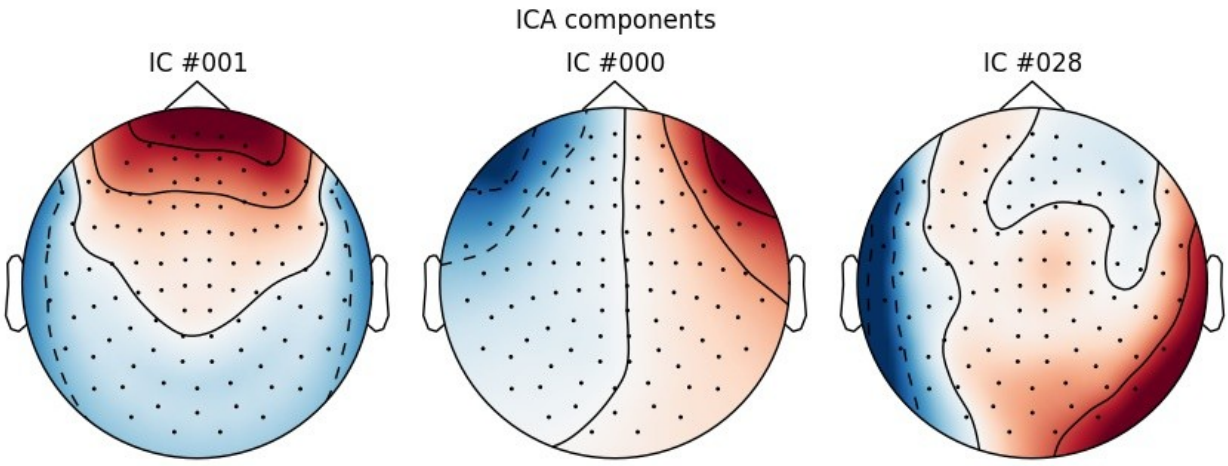

Figure 3: ICA components corresponding to eye blinks (ICA000), eye movements (ICA001) and heart beats(ICA028)

```
def run_ica(name, save_dir, lowpass, overwrite):

    ica_name = name + filter_string(lowpass) + '-ica.fif'
    ica_path = join(save_dir, ica_name)

    if overwrite or not isfile(ica_path):

        raw = io.read_filtered(name, save_dir, lowpass)
        epochs = io.read_epochs(name, save_dir, lowpass)

        ica = mne.preprocessing.ICA(n_components=0.95, method='fastica')
        ica.fit(epochs)

        eog_epochs = mne.preprocessing.create_eog_epochs(raw)
        ecg_epochs = mne.preprocessing.create_ecg_epochs(raw)

        eog_indices, eog_scores = ica.find_bads_eog(eog_epochs)
        ecg_indices, ecg_scores = ica.find_bads_ecg(ecg_epochs)

        ica.exclude += eog_indices
        ica.exclude += ecg_indices

        ica.save(ica_path)

    else:
        print('ica file: '+ ica_path + ' already exists')
```


Code Snippet 8: The function for finding the independent components that most likely correspond to eye blinks, eye movements and heart beats.

Zero out eye- and heart-related components in the epoched data

Use ***apply_ica*** to zero out the components identified above to clean the data of eye- and heart-related activity.

```
def apply_ica(name, save_dir, lowpass, overwrite):

    ica_epochs_name = name + filter_string(lowpass) + '-ica-epo.fif'
    ica_epochs_path = join(save_dir, ica_epochs_name)

    if overwrite or not isfile(ica_epochs_path):

        epochs = io.read_epochs(name, save_dir, lowpass)
        ica = io.read_ica(name, save_dir, lowpass)

        ica_epochs = ica.apply(epochs)

        ica_epochs.save(ica_epochs_path)

    else:
        print('ica epochs file: '+ ica_epochs_path + ' already exists')
```

Code Snippet 9: The function for removing the eye blink, eye movement and heart beat components from the epoched data.

Event-related fields after relevant components have been removed

Finally, the event-related fields are found for all the events of interest by looping through <epochs.event_id> and an averaged response is created for each event by calling ***get_evokeds***. The cleaned epochs can be plotted with ***plot_epochs_image*** (Figure 4). The event-related fields can be plotted with ***plot_evokeds*** and ***plot_butterfly_evokeds***.

```
def get_evokeds(name, save_dir, lowpass, overwrite):

    evokeds_name = name + filter_string(lowpass) + '-ave.fif'
    evokeds_path = join(save_dir, evokeds_name)
    if overwrite or not isfile(evokeds_path):

        epochs = io.read_ica_epochs(name, save_dir, lowpass)

        evokeds = []

        for trial_type in epochs.event_id:
            evokeds.append(epochs[trial_type].average())

        mne.evoked.write_evokeds(evokeds_path, evokeds)

    else:
        print('evokeds file: '+ evokeds_path + ' already exists')
```

Code Snippet 10: The function for calculating the evoked responses based on the ICA-cleaned epochs.

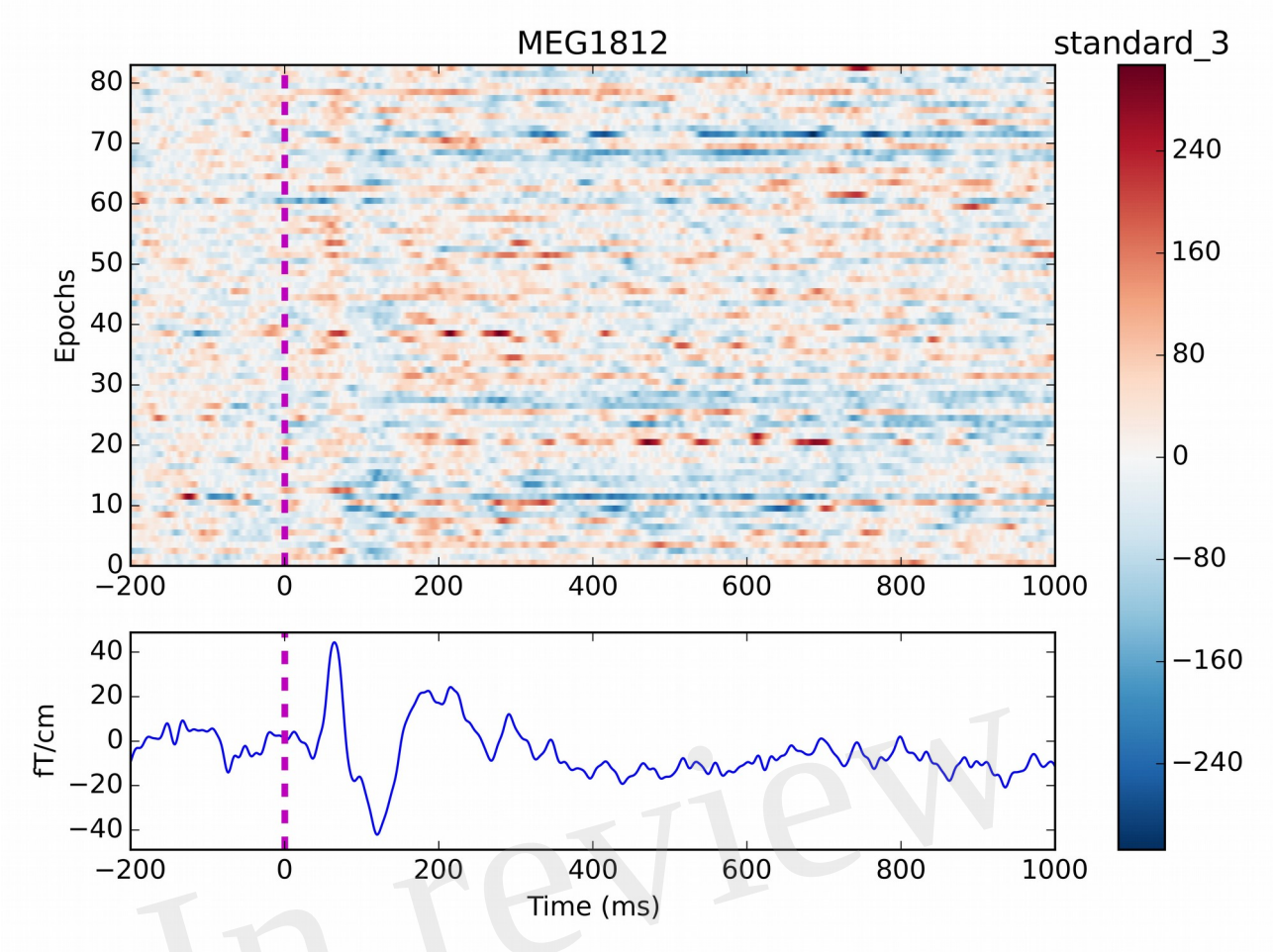

Figure 4: Epochs and event-related field for channel MEG 1812 for condition *standard 3.* The colouring indicates the field strength for each epoch. Two evoked responses can be seen after about 60 ms and 135 ms respectively.

Summary

This part of the code covered filtering of the data, finding events of interest, epoching of the data, estimating independent components, removing the eye- and heart-related components and finally averaging the cleaned data. Expected evoked response can be seen after about 60 and 135 ms (Figure 4). The averages will be used for the subsequent source reconstruction of the data. To this end we need to preprocess the MRI data as well.

Preprocessing the MRI data

Dependencies

The bash scripts require FreeSurfer http://freesurfer.net/ and MNE-C http://martinos.org/mne. Both run exclusively on Linux and Mac platforms. The plotting functions are based on MNE-Python. The function for creating high-resolution scalp surfaces also requires MATLAB. This is not strictly necessary for completing the source analysis, but is included since it aids in aligning the MEG and MRI coordinate systems. Due to concerns about subject anonymity, the original MRI data are not provided. The "bem" folder for each subject in $SUBJECTS_DIR (Code Snippet 12) is provided though, as this information is judged non-sensitive. The first three bash scripts (Code Snippets 12-14) can thus not be applied to the data, but they are included such that users can these in their own experiments. They cover reading in dicom files, segmenting the brain and delineating the surface between brain, skull and skin.

Subjects file

All the bash files below depend on reading in the subjects from subjects.txt (Code Snippet 11).

```
sub-01
sub-02
...
sub-20
```

Code Snippet 11: A text file with the subject numbers to run through for the Bash scripts.

Read in dicom files

Use Code Snippet 12 to read in the magnetic resonance images. This creates a subject folder in the SUBJECTS_DIR directory required by FreeSurfer.

```
#! /bin/bash

export SUBJECTS_DIR=/home/lau/analyses/omission_frontiers_BIDS-MNE-Python/data/FreeSurfer
data_path=/home/lau/analyses/omission_frontiers_BIDS-MNE-Python/data
subjects=( `cat '/home/lau/analyses/omission_frontiers_BIDS-MNE-Python/scripts/bash/subjects.txt' `)

for subject in "${subjects[@]}"

do

   dicom_path=${data_path}/${subject}/ses-mri/anat/
   cd $dicom_path
   first_dicom_file=$(ls | head -n 1)
   recon-all -subjid $subject -i $first_dicom_file -openmp 32

done
```

Code Snippet 12: Code for importing the dicom files into the FreeSurfer folder, which FreeSurfer requires.

Segment the MRI

Use Code Snippet 13 to do the full segmentation of the brain into its constituent parts using FreeSurfer. [openmp] sets the number of processors that FreeSurfer will use. This is a very lengthy process and takes between ~6-24 h for each subject depending on processing power.

```
export SUBJECTS_DIR=/home/lau/analyses/omission_frontiers_BIDS-MNE-Python/data/FreeSurfer
subjects=( `cat '/home/lau/analyses/omission_frontiers_BIDS-MNE-Python/scripts/bash/subjects.txt' `)

for subject in "${subjects[@]}"

do

   recon-all -subjid $subject -all -openmp 32

done
```

Code Snippet 13: Code for doing a full FreeSurfer segmentation (a very lengthy process).

Create boundaries with the Boundary Element Method (BEM) using the watershed algorithm

Use Code Snippet 14 to create surfaces for the inner skull, the outer skin, the outer skull and the brain surface with an MNE-C command, which uses FreeSurfer code. Symbolic links to the watershed files are created in the bem-folder for each subject since this this is where MNE-C expects to find them.

```
#! /bin/bash

export SUBJECTS_DIR=/home/lau/analyses/omission_frontiers_BIDS-MNE-Python/data/FreeSurfer
subjects=( `cat '/home/lau/analyses/omission_frontiers_BIDS-MNE-Python/scripts/bash/subjects.txt' `)

for subject in "${subjects[@]}"
```

```
do

    echo 'Making BEM solution for SUBJECT: '$subject
    cd $SUBJECTS_DIR/$subject/bem

    mne_watershed_bem --subject $subject --overwrite
    ln -sf $SUBJECTS_DIR/$subject/bem/watershed/${subject}_inner_skull_surface $SUBJECTS_DIR/$subject/bem/inner_skull.surf
    ln -sf $SUBJECTS_DIR/$subject/bem/watershed/${subject}_outer_skin_surface $SUBJECTS_DIR/$subject/bem/outer_skin.surf
    ln -sf $SUBJECTS_DIR/$subject/bem/watershed/${subject}_outer_skull_surface $SUBJECTS_DIR/$subject/bem/outer_skull.surf
    ln -sf $SUBJECTS_DIR/$subject/bem/watershed/${subject}_brain_surface $SUBJECTS_DIR/$subject/bem/brain_surface.surf

done
```

Code Snippet 14: Code for creating the boundary elements necessary for defining the volume conductor.

Make source spaces

Use Code Snippet 15 to to create a source space that is restricted to the cortex with ~10000 sources modelled as equivalent current dipoles normal to the cortical surface.

```
#! /bin/bash

export SUBJECTS_DIR=/home/lau/analyses/omission_frontiers_BIDS-MNE-Python/data/FreeSurfer
subjects=( `cat '/home/lau/analyses/omission_frontiers_BIDS-MNE-Python/scripts/bash/subjects.txt' `)

for subject in "${subjects[@]}"

do

    mne_make_scalp_surfaces --subject $subject --overwrite

done
```

Code Snippet 15: Code for making the source space. All sources are perpendicular to the cortical surface that was delineated in Code Snippet 13.

The source space can be plotted with ***plot_source_space*** (Figure 5)**.**

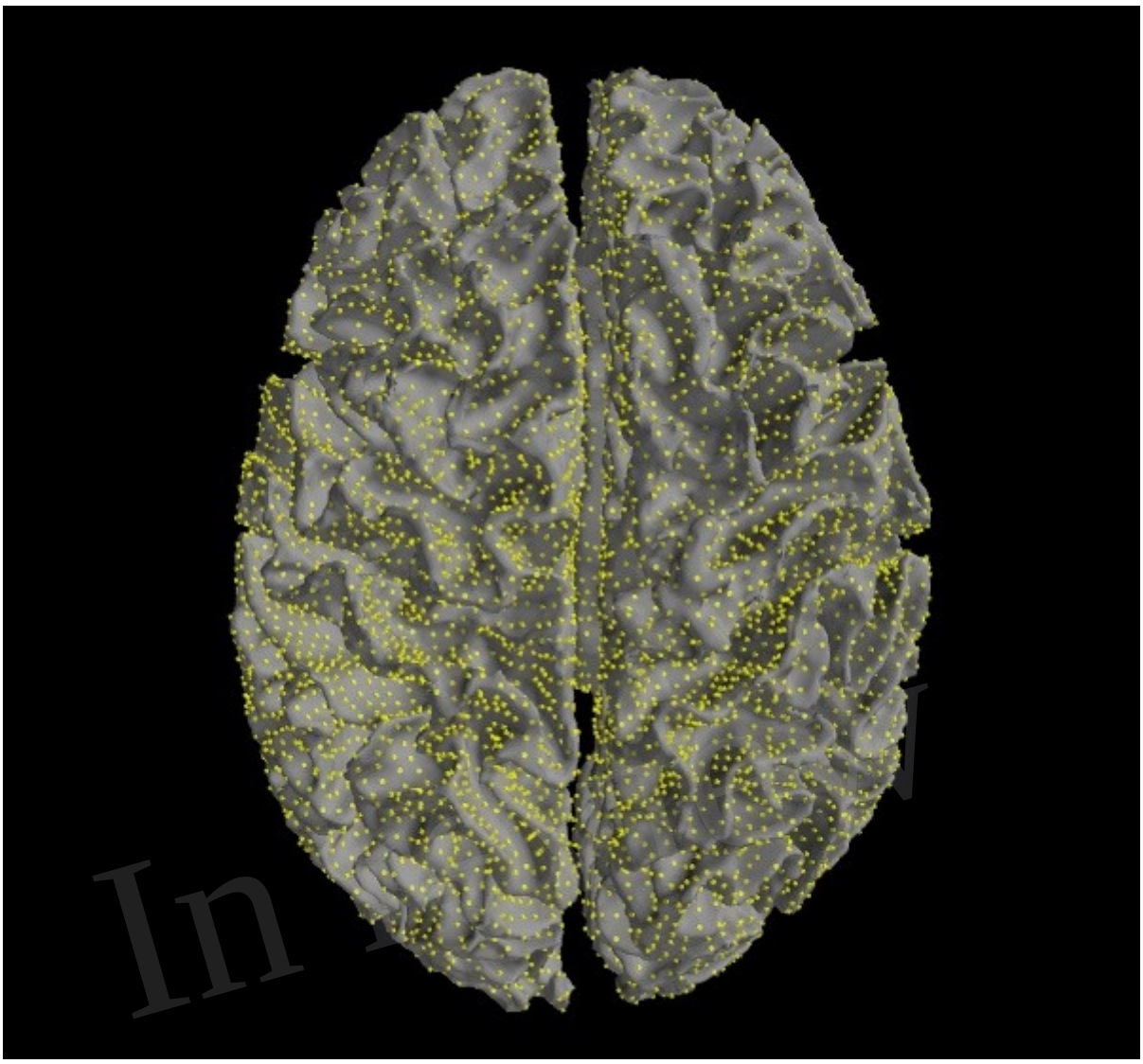

Figure 5: Source space. Sources are restricted to the cortex. Yellow dots mark equivalent current dipoles oriented normally to the cortical surface.

Make scalp surfaces

Use Code Snippet 16 to make high-resolution scalp surfaces for each subject. This eases the co-registration since it makes it easier to identify the fiducials, nasion and left and right pre-auricular points. The MNE-C code here is dependent on MATLAB, but the high-resolution scalp surfaces are not strictly necessary for the completing the analysis. Their purpose is to ease the co-registration of the MEG and MRI data.

```
#! /bin/bash

export SUBJECTS_DIR=/home/lau/analyses/omission_frontiers_BIDS-MNE-Python/data/FreeSurfer

subjects=( `cat '/home/lau/analyses/omission_frontiers_BIDS-MNE-Python/scripts/bash/subjects.txt' `)

for subject in "${subjects[@]}"

do

	mne_setup_source_space --subject $subject --ico 5 --overwrite

done
```

Code Snippet 16: Code for making high-resolution scalp surfaces.

Create solutions for the BEMs

Use Code Snippet 17 to create a volume conductor model describing how the magnetic fields spread throughout the conductor (the head). [homog] makes a single-compartment model (sensible for MEG). [surf] instructs MNE-C to use the surfaces created with the watershed algorithm. [ico] determines the downsampling of the surface. [ico=4] results in ~10000 sources for the two hemispheres.

```
#! /bin/bash

export SUBJECTS_DIR=/home/lau/analyses/omission_frontiers_BIDS-MNE-Python/data/FreeSurfer
subjects=( `cat '/home/lau/analyses/omission_frontiers_BIDS-MNE-Python/scripts/bash/subjects.txt' `)

for subject in "${subjects[@]}"

do

    mne_setup_forward_model --subject $subject --homog --surf --ico 4


done
```

Code Snippet 17: Code for making the BEM-solutions, that is the volume conductor.

Source reconstruction of time courses

Dependencies

This part is only dependent on MNE-Python.

Co-registration

Call the function ***mne.gui.coregistration*** directly from a Python environment to co-register the MEG data to the MRI data. Fiducials used are the nasion and the left and right pre-auricular points. The scalp surfaces made above should make it easier to identify these fiducials. When these have been set, load a file that has the extra head shape digitization points and lock the fiducials. Then fit the head shape, and if the fit looks good save the transformation file as ‘oddball_absence_dense-trans.fif’ in the same folder where all other MEG data files are saved. The resulting transformation can be plotted with ***plot_transformation*** (Figure 6). Note that a transformation with this name has already been supplied, such that the analysis can be replicated faithfully.

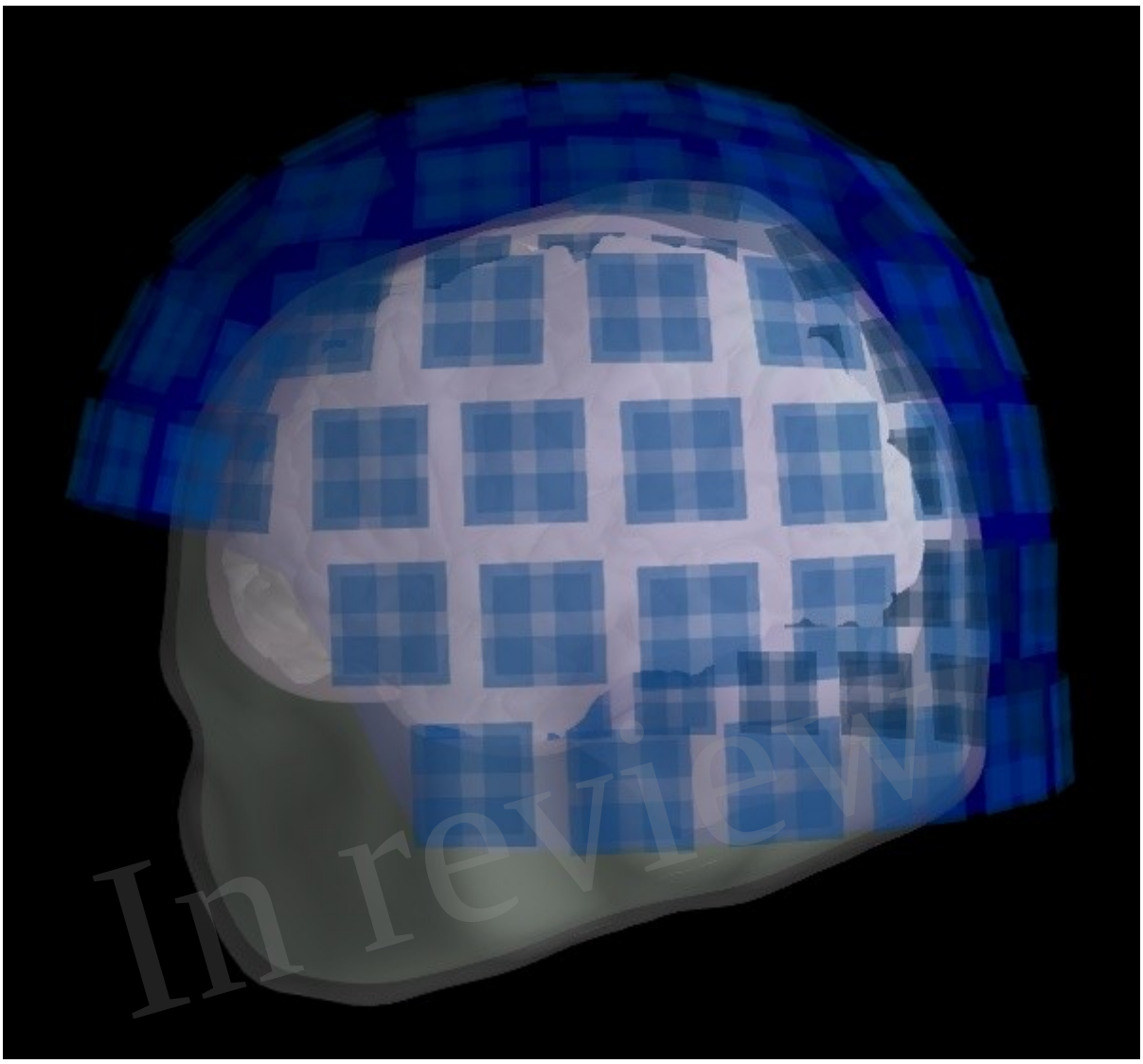

Figure 6: Transformation. The positions of the head, skull,brain and helmet sensors after the transformation.

Create forward model

Use ***create_forward_solution*** (Code Snippet 18) to create the forward model for the source reconstructions. This contains the source space, the volume conductor model, the transformation between the MEG and MRI coordinate systems and info about the channels in the data.

```
def create_forward_solution(name, save_dir, subject, subjects_dir,
                            overwrite):

    forward_name = name + '-fwd.fif'
    forward_path = join(save_dir, forward_name)

    if overwrite or not isfile(forward_path):

        info = io.read_info(name, save_dir)
        trans = io.read_transformation(name, save_dir)
        bem = io.read_bem_solution(subject, subjects_dir)
        source_space = io.read_source_space(subject, subjects_dir)

        forward = mne.make_forward_solution(info, trans, source_space, bem,
                                            n_jobs=1)

        mne.write_forward_solution(forward_path, forward, overwrite)

    else:
        print('forward solution: ' + forward_path + ' already exists')
```

Code Snippet 18: Function for creating the forward solution (also known as the lead field). This is created from the BEM-solution (the volume conductor), the channel info about the sensor positions, the coordinate transformation between the MEG and the MRI data and the source space defining where sources are.

Estimate noise covariance

Use ***estimate_noise_covariance*** (Code Snippet 19) to estimate the noise covariance and regularize it. The noise covariance serves as an estimate of the noise in the data, which is necessary for MNE-like solutions. The noise covariance matrix can be plotted with ***plot_noise_covariance*** (Figure 7).

```
def estimate_noise_covariance(name, save_dir, lowpass, overwrite):

    covariance_name = name + filter_string(lowpass) + '-cov.fif'
    covariance_path = join(save_dir, covariance_name)

    if overwrite or not isfile(covariance_path):

        epochs = io.read_epochs(name, save_dir, lowpass)

        noise_covariance = mne.compute_covariance(epochs, n_jobs=1)

        noise_covariance = mne.cov.regularize(noise_covariance,
                                              epochs.info)

        mne.cov.write_cov(covariance_path, noise_covariance)

    else:
        print('noise covariance file: '+ covariance_path + \
              ' already exists')
```

Code Snippet 19: Function for estimating the noise covariance in the MEG data.

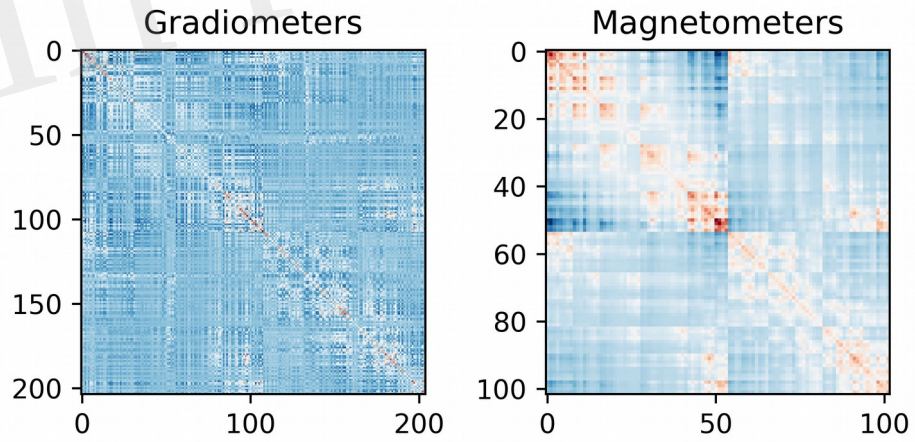

Figure 7: Noise covariance matrices. As can be seen the covariance between magnetometers is greater than between gradiometers. This can be explained by magnetometers being more sensitive to far sources than gradiometers are.

Create the inverse operator

The final step before estimating source activity is to create an inverse operator, which contains the info about the MEG-recordings, the estimated noise and the forward model. This is done with ***create_inverse_operator*** (Code Snippet 19).

```
def create_inverse_operator(name, save_dir, lowpass, overwrite):

    inverse_operator_name = name + filter_string(lowpass) +  '-inv.fif'
    inverse_operator_path = join(save_dir, inverse_operator_name)

    if overwrite or not isfile(inverse_operator_path):

        info = io.read_info(name, save_dir)
        noise_covariance = io.read_noise_covariance(name, save_dir, lowpass)
        forward = io.read_forward(name, save_dir)

        inverse_operator = mne.minimum_norm.make_inverse_operator(
                        info, forward, noise_covariance)

        mne.minimum_norm.write_inverse_operator(inverse_operator_path,
                                        inverse_operator)

    else:
        print('inverse operator file: '+ inverse_operator_path + \
            ' already exists')
```

Code Snippet 20: Function for creating the inverse operator that defines what inverse solution should be applied.

Estimating the source time courses

Finally, we estimate the source time courses. [method] is set in the parameter selection. *dSPM* is a depth-weighted minimum source estimate (Dale et al., 2000), *MNE* is the classical algorithm described by Hämäläinen and Ilmoniemi (1994) and sLORETA is described by Pascual-Marqui (2002) A source time course (stc-file) is created for each condition. This is done with ***source_estimate*** (Code Snippet 21).

```
def source_estimate(name, save_dir, lowpass, method,
                overwrite):

    inverse_operator = io.read_inverse_operator(name, save_dir, lowpass)
    to_reconstruct = io.read_evokeds(name, save_dir, lowpass)
    evokeds = io.read_evokeds(name, save_dir, lowpass)

    stcs = dict()
    for to_reconstruct_index, evoked in enumerate(evokeds):
        stc_name = name + filter_string(lowpass) + '_' + evoked.comment + '_' + method + '-lh.stc'
        stc_path = join(save_dir, stc_name)
        if overwrite or not isfile(stc_path):
            trial_type = evoked.comment

            stcs[trial_type] = mne.minimum_norm.apply_inverse(
                                    to_reconstruct[to_reconstruct_index],
                                    inverse_operator,
                                    method=method)
        else:
            print('source estimates for: '+  stc_path + \
            ' already exists')

    for stc in stcs:
        stc_name = name + filter_string(lowpass) + '_' + stc + '_' + method
        stc_path = join(save_dir, stc_name)
        if overwrite or not isfile(stc_path + '-lh.stc'):
            stcs[stc].save(stc_path)
```

Code Snippet 21: Function for doing the actual minimum norm estimate source reconstruction.

The spatial source distribution for a given time point can be plotted with ***plot_source_estimates*** (Figure 8). <mne_evoked_time> in *pipeline.py* can be set to control which time point is plotted.

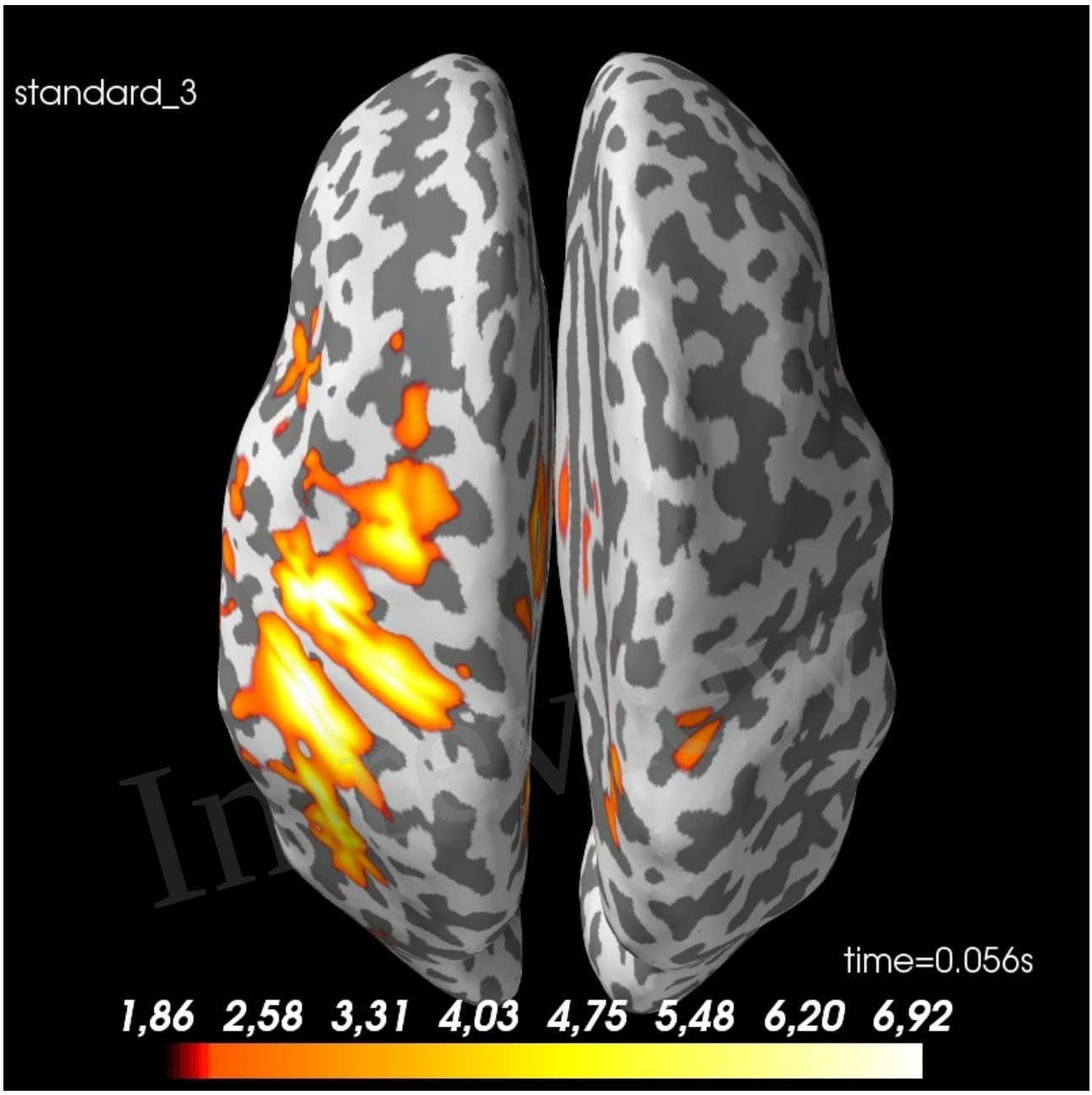

Figure 8: Spatial distribution for 56 ms for *standard 3*: There is some spread, but there is a clear activation of the contralateral sensory cortex. Values are dSPM-values. These are current estimates normalized with the noise-covariance. The cortex is shown inflated with gyri darker than sulci.

Morph to a common template

Use ***morph_data_to_fsaverage*** (Code Snippet 22) to make a meaningful estimate across subjects, by morphing the data for each individual subject to a common template brain. In this case, the fsaverage brain from FreeSurfer is used (This requires the fsaverage brain to be in $SUBJECTS_DIR). [method] can be ‘dSPM’, ‘MNE’ or ‘sLORETA’.

```
def morph_data_to_fsaverage(name, save_dir, subjects_dir, subject,
                            lowpass, method, overwrite):

    stcs = io.read_source_estimates(name, save_dir, lowpass, method)

    subject_to = 'fsaverage'
    stcs_morph = dict()

    for trial_type in stcs:
        stc_morph_name = name + filter_string(lowpass) + '_' + \
        trial_type + '_' + method + '_morph'
        stc_morph_path = join(save_dir, stc_morph_name)

        if overwrite or not isfile(stc_morph_path + '-lh.stc'):
            stc_from = stcs[trial_type]
            stcs_morph[trial_type] = mne.morph_data(subject, subject_to,
                                                    stc_from,
                                                    subjects_dir=subjects_dir,
                                                    n_jobs=-1)
        else:
            print('morphed source estimates for: '+  stc_morph_path + \
            ' already exists')

    for trial_type in stcs_morph:
        stc_morph_name = name + filter_string(lowpass) + '_' + \
        trial_type + '_' + method + '_morph'
        stc_morph_path = join(save_dir, stc_morph_name)
        if overwrite or not isfile(stc_morph_path + '-lh.stc'):
            stcs_morph[trial_type].save(stc_morph_path)
```

Code Snippet 22: Function for making morph maps that define how individual subject source reconstructions can be mapped onto a common template that allows for comparisons between subjects.

Summary

Now we have estimated source time courses for all the individual subjects. The next step is to meaningfully make a group estimate across subjects finding that the activity for our example subject (*sub-01*) can be localized to the somatosensory cortex (Figure 8).

Between subjects analyses

Dependencies

This part is only dependent on MNE-Python.

Sensor space

With the function ***grand_average_evokeds*** (Code Snippet 23), the grand average in sensor space for each condition is calculated and saved. Grand averages can be plotted with ***plot_grand_averages_evokeds*** and ***plot_grand_averages_butterfly_evokeds*** (Figure 9). Note that these may not be easy to interpret since the relative positions between a given subject's head and the MEG sensors will differ from the relative positions between any other subject's head and the MEG sensors. The early and the late responses are picked up however (Figure 9).

```
def grand_average_evokeds(evoked_data_all, save_dir_averages, lowpass):

    grand_averages = dict()
    for trial_type in evoked_data_all:
        grand_averages[trial_type] = \
            mne.evoked.grand_average(evoked_data_all[trial_type])

    for trial_type in grand_averages:
        grand_average_path = save_dir_averages + \
            trial_type +  filter_string(lowpass) + \
            '_grand_average-ave.fif'
        mne.evoked.write_evokeds(grand_average_path, grand_averages[trial_type])
```

Code Snippet 23: Function for calculating grand averages across the evokeds of individual subjects.

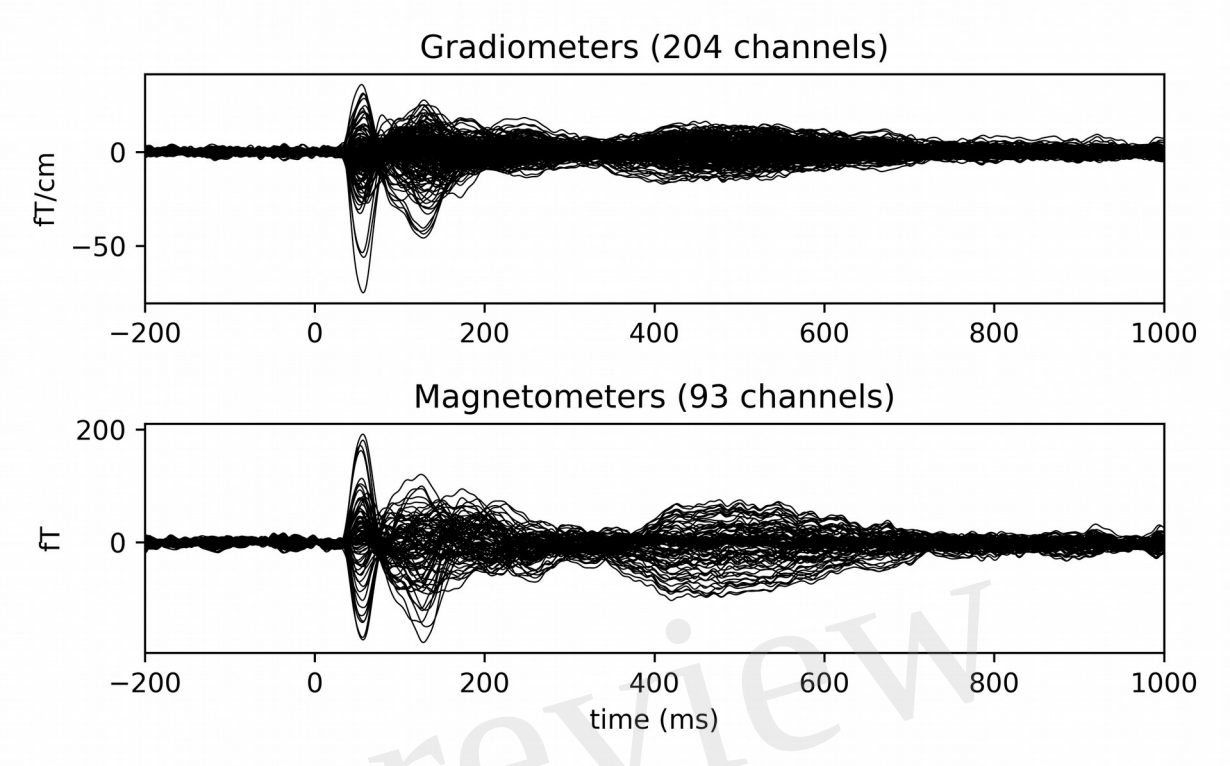

Figure 9: Grand average butterfly plot for *standard 3* showcasing the SI (56 ms) and SII (135 ms) components.

Source space

With the function ***average_morphed_data*** (Code Snippet 24), the grand average in source space over the morphed source time courses for each condition is calculated and saved. [method] can be ‘dSPM’, ‘MNE’ or ‘sLORETA’. The grand averages for the source space can be plotted with ***plot_grand_averages_source_estimates*** (Figure 10). <mne_evoked_time> needs to be set.

```
def average_morphed_data(morphed_data_all, method, save_dir_averages, lowpass):

    averaged_morphed_data = dict()

    n_subjects = len(morphed_data_all['standard_1']) ##
    for trial_type in morphed_data_all:
        trial_morphed_data = morphed_data_all[trial_type]
        trial_average = trial_morphed_data[0].copy() ## get copy of first instance

        for trial_index in range(1, n_subjects):
            trial_average._data += trial_morphed_data[trial_index].data

        trial_average._data /= n_subjects
        averaged_morphed_data[trial_type] = trial_average

    for trial_type in averaged_morphed_data:
        stc_path = save_dir_averages  + \
            trial_type + filter_string(lowpass) + '_morphed_data_' + method
        averaged_morphed_data[trial_type].save(stc_path)
```

Code Snippet 24: Function for calculating the grand average across all individual morphed subject source reconstructions.

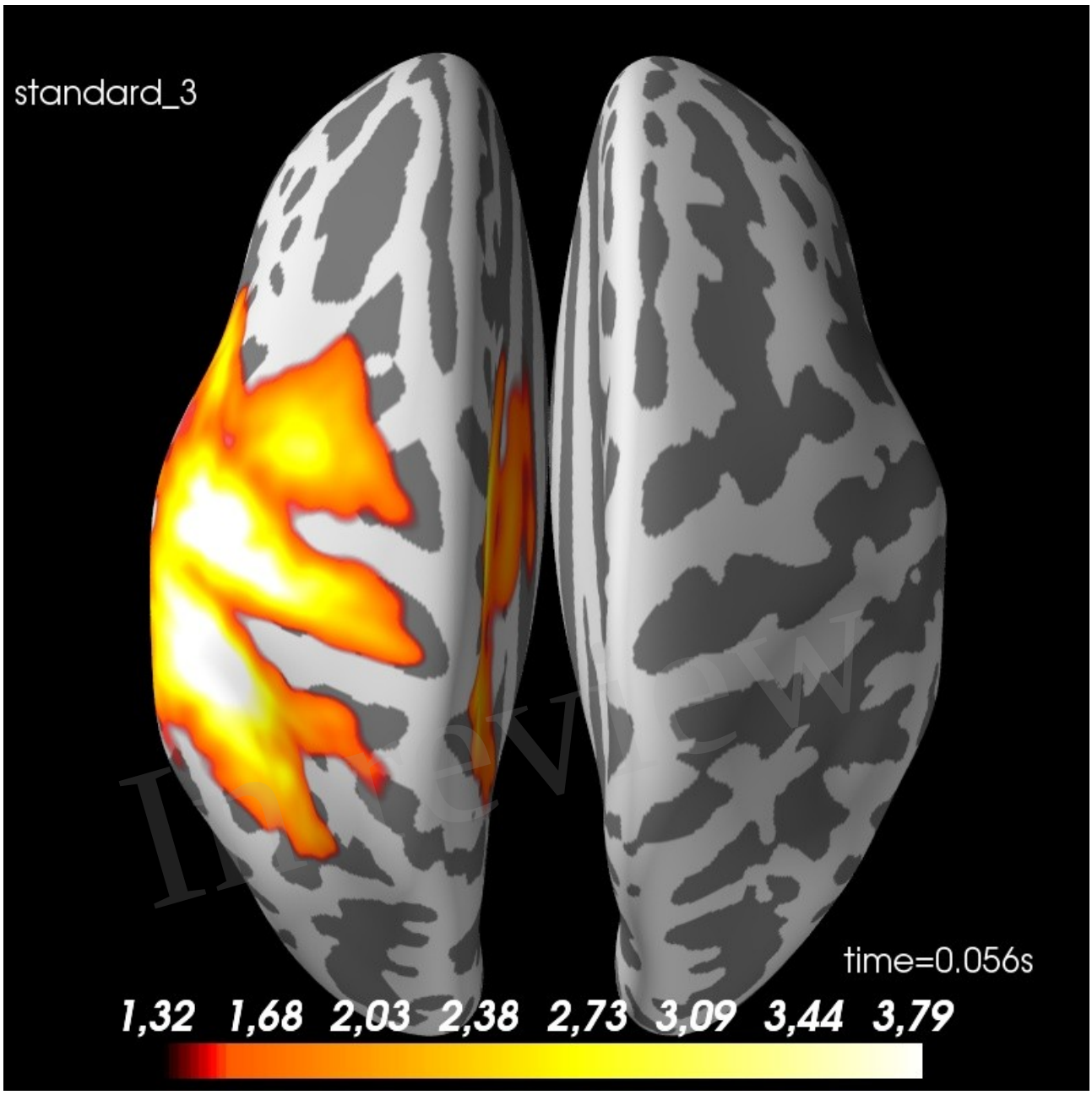

Figure 10: Spatial distribution for 56 ms for grand average of *standard 3*: There is some spread, but there is a clear activation of the contralateral sensory cortex. Values are dSPM-values. These are current estimates normalized with the noise-covariance. The cortex is shown inflated with gyri darker than sulci.

Statistical analyses

Dependencies

This part is only dependent on MNE-Python.

With the function ***statistics_source_space*** (Code Snippet 25), different statistical null hypotheses can be tested.

```
def statistics_source_space(morphed_data_all, save_dir_averages,
                            independent_variable_1,
                            independent_variable_2,
                            time_window, n_permutations, lowpass, overwrite):

    cluster_name = independent_variable_1 + '_vs_' + independent_variable_2 + \
                   filter_string(lowpass) + '_time_' + \
                   str(int(time_window[0] * 1e3)) + '-' + \
                   str(int(time_window[1] * 1e3)) + '_msec.cluster'
    cluster_path = join(save_dir_averages, 'statistics', cluster_name)

    if overwrite or not isfile(cluster_path):

        input_data = dict(iv_1=morphed_data_all[independent_variable_1],
                          iv_2=morphed_data_all[independent_variable_2])
        info_data = morphed_data_all[independent_variable_1]
        n_subjects = len(info_data)
        n_sources, n_samples = info_data[0].data.shape

        ## get data in the right format
        statistics_data_1 = np.zeros((n_subjects, n_sources, n_samples))
        statistics_data_2 = np.zeros((n_subjects, n_sources, n_samples))

        for subject_index in range(n_subjects):
            statistics_data_1[subject_index, :, :] = input_data['iv_1'][subject_index].data
            statistics_data_2[subject_index, :, :] = input_data['iv_2'][subject_index].data
            print 'processing data from subject: ' + str(subject_index)

        ## crop data on the time dimension
        times = info_data[0].times
        time_indices = np.logical_and(times >= time_window[0],
                                      times <= time_window[1])

        statistics_data_1 = statistics_data_1[:, :, time_indices]
        statistics_data_2 = statistics_data_2[:, :, time_indices]

        ## set up cluster analysis
        p_threshold = 0.05
        t_threshold = stats.distributions.t.ppf(1 - p_threshold / 2, n_subjects - 1)
        seed = 7 ## my lucky number

        statistics_list = [statistics_data_1, statistics_data_2]

        T_obs, clusters, cluster_p_values, H0 = \
            mne.stats.permutation_cluster_test(statistics_list,
                                               n_permutations=n_permutations,
                                               threshold=t_threshold,
                                               seed=seed,
                                               n_jobs=-1)

        cluster_dict = dict(T_obs=T_obs, clusters=clusters,
                            cluster_p_values=cluster_p_values, H0=H0)

        with open(cluster_path, 'wb') as filename:
            pickle.dump(cluster_dict, filename)

        print 'finished saving cluster at path: ' + cluster_path

    else:
        print('cluster permutation: '+ cluster_path + \
            ' already exists')
```

Code Snippet 25: Function for doing cluster statistics in source space.

<independent_variable_1>, <independent_variable_2>, <time_window> and <n_permutations> should all be set. With ***plot_grand_averages_source_estimates_cluster_masked*** (Figure 11) the t-masked grand average source estimates can be plotted. <p_threshold> should be set. This function can be changed such that any other function in the ***mne.stats*** module is used.

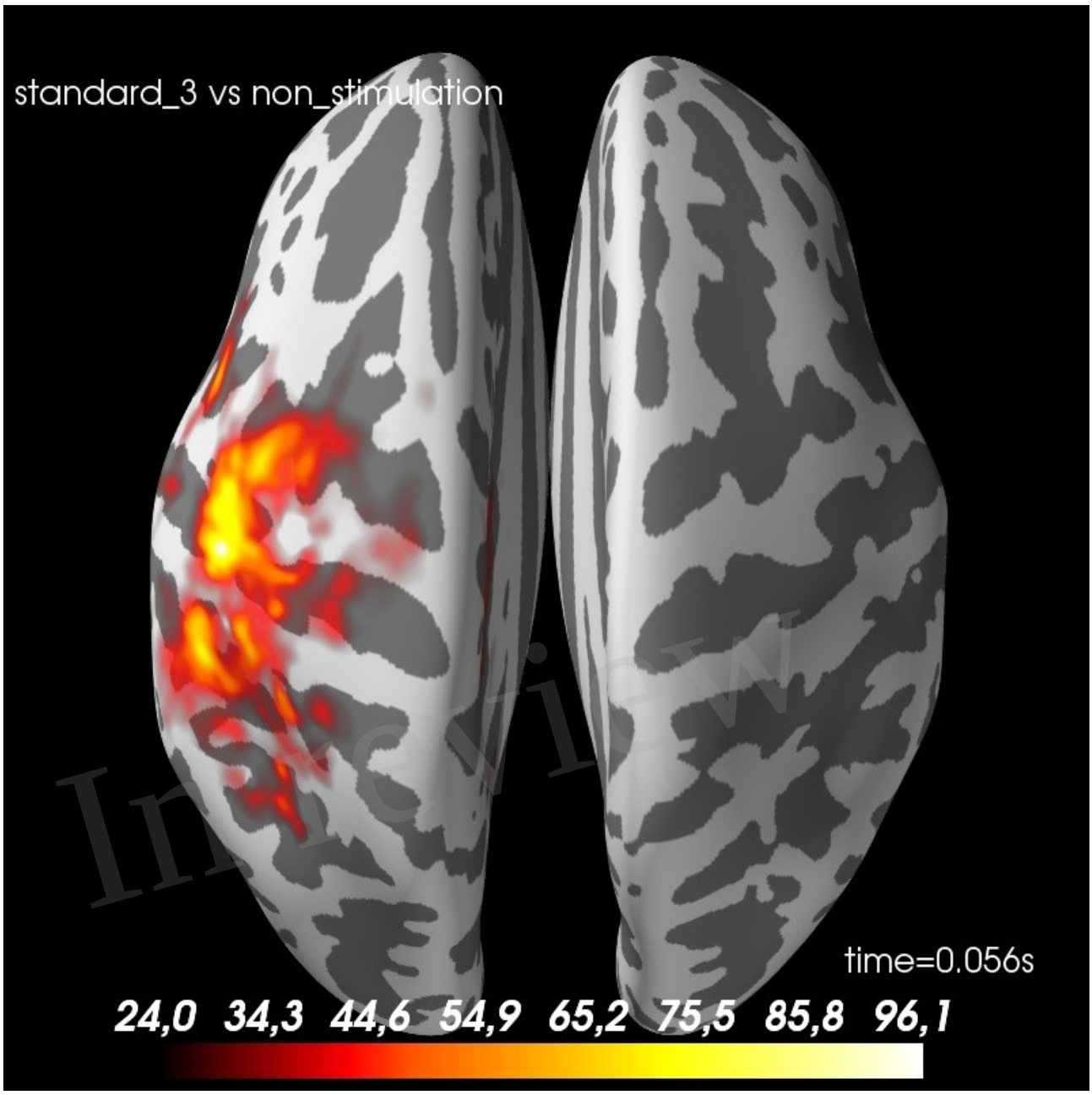

Figure 11: A *t*-value map for *standard 3* versus *non-stimulation* at 56 ms. The cortex is shown inflated with gyri darker than sulci.

Summary

This protocol allows for all steps of conducting a MEG group study aiming to provide evidence for a significant effect of one experimental condition compared to another experimental condition using Minimum Norm Estimates of MEG data. We found as expected that stimulation of the finger elicited more activity in the contralateral somatosensory cortex than when no such stimulation occurred.

Discussion

The presented pipeline allows for covering all involved in an MNE-Python pipeline focussing on evoked responses and the localization of their neural origin. Furthermore, it also supplies a very flexible framework that users should be able to extend to meet any further needs that the user may

have. Facilitating other MNE-Python functions not showcased here across groups of subjects can be attained by emulating the style of defining functions presented here. If one is interested in estimating induced responses, one can use the functions in the ***mne.time_frequency*** module. The neural origin of induced responses are often localized with beamformer solutions (Gross et al., 2013), which can also be performed with MNE-Python using the ***mne.beamformer*** module. Both these can be extended by a user with some programming experience.

Funding

Data for this study was collected at NatMEG (www.natmeg.se), the National infrastructure for Magnetoencephalography, Karolinska Institutet, Sweden. The NatMEG facility is supported by Knut & Alice Wallenberg (KAW2011.0207). The study, and Lau Møller Andersen, was funded by Knut & Alice Wallenberg Foundation (KAW2014.0102).

References

Dale, A. M., Liu, A. K., Fischl, B. R., Buckner, R. L., Belliveau, J. W., Lewine, J. D., et al. (2000). Dynamic Statistical Parametric Mapping: Combining fMRI and MEG for High-Resolution Imaging of Cortical Activity. *Neuron* 26, 55–67. doi:10.1016/S0896-6273(00)81138-1.

Gramfort, A., Luessi, M., Larson, E., Engemann, D. A., Strohmeier, D., Brodbeck, C., et al. (2013). MEG and EEG data analysis with MNE-Python. *Brain Imaging Methods* 7, 267. doi:10.3389/fnins.2013.00267.

Gross, J., Baillet, S., Barnes, G. R., Henson, R. N., Hillebrand, A., Jensen, O., et al. (2013). Good practice for conducting and reporting MEG research. *NeuroImage* 65, 349–363. doi:10.1016/j.neuroimage.2012.10.001.

Hämäläinen, M. S., and Ilmoniemi, R. J. (1994). Interpreting magnetic fields of the brain: minimum norm estimates. *Med. Biol. Eng. Comput.* 32, 35–42. doi:10.1007/BF02512476.

Hyvärinen, A. (1999). Fast and robust fixed-point algorithms for independent component analysis. *IEEE Trans. Neural Netw.* 10, 626–634. doi:10.1109/72.761722.

Pascual-Marqui, R. D. (2002). Standardized low-resolution brain electromagnetic tomography (sLORETA): technical details. *Methods Find. Exp. Clin. Pharmacol.* 24 Suppl D, 5–12.

Taulu, S., and Simola, J. (2006). Spatiotemporal signal space separation method for rejecting nearby interference in MEG measurements. *Phys. Med. Biol.* 51, 1759–1768. doi:10.1088/0031-9155/51/7/008.

Figure 1.JPEG

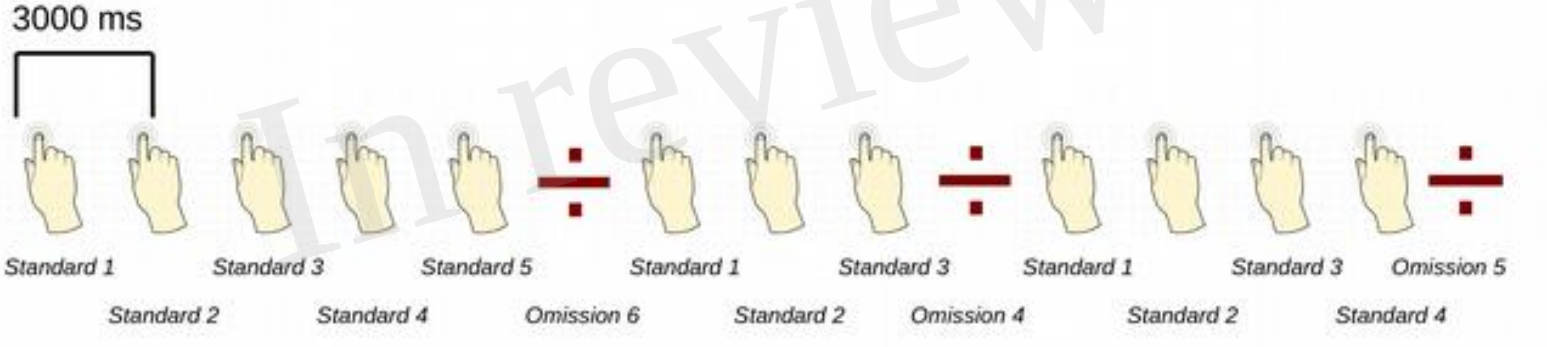

Figure 2.JPEG

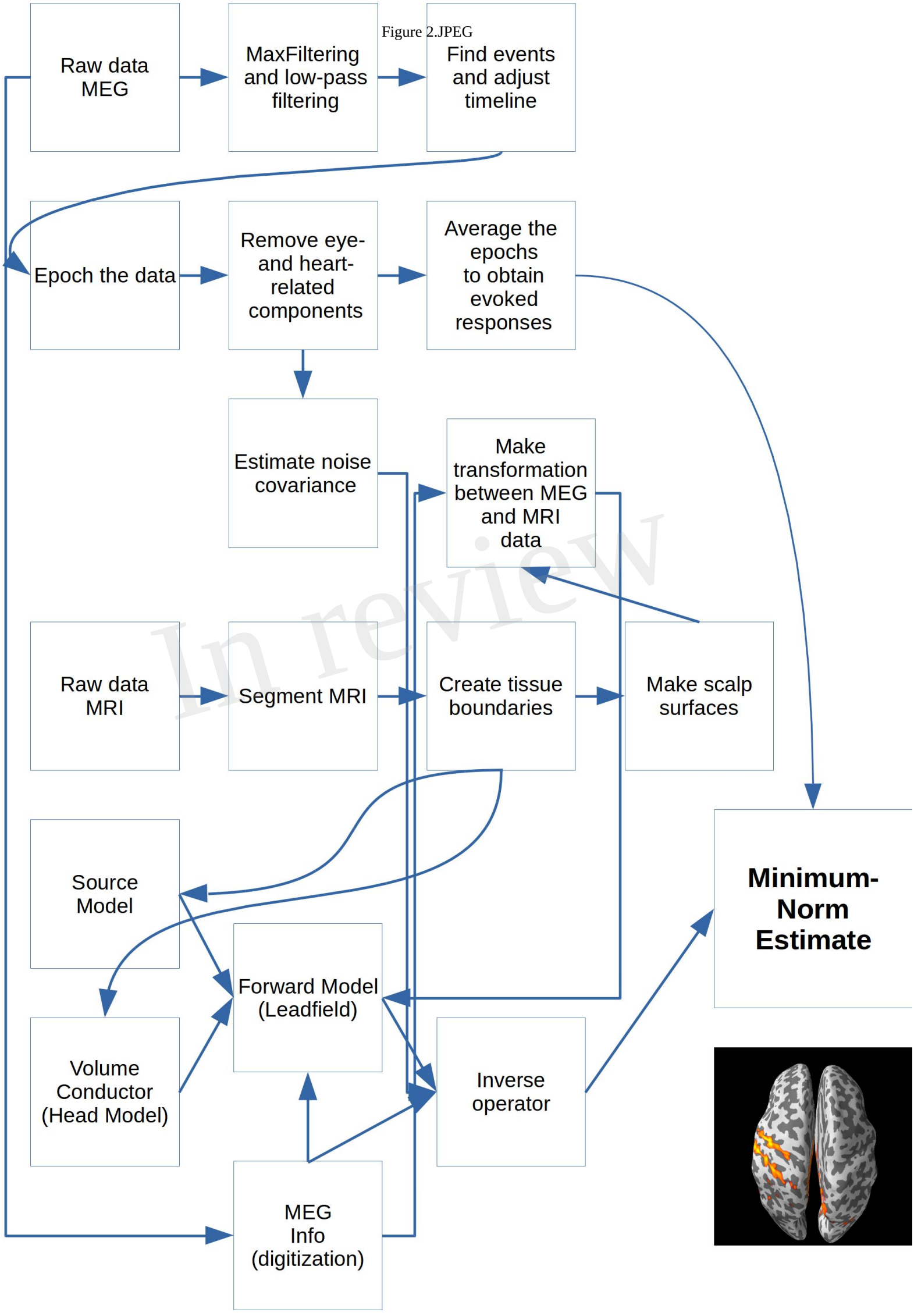

Figure 3.JPEG

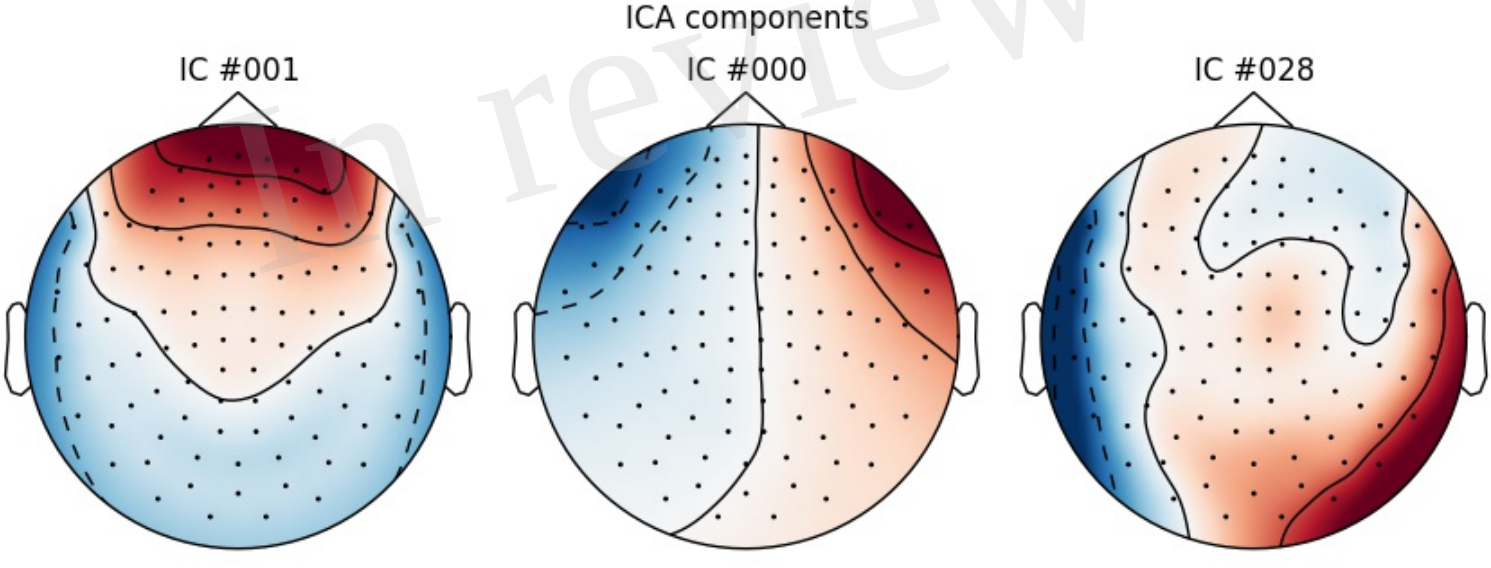

Figure 4.JPEG

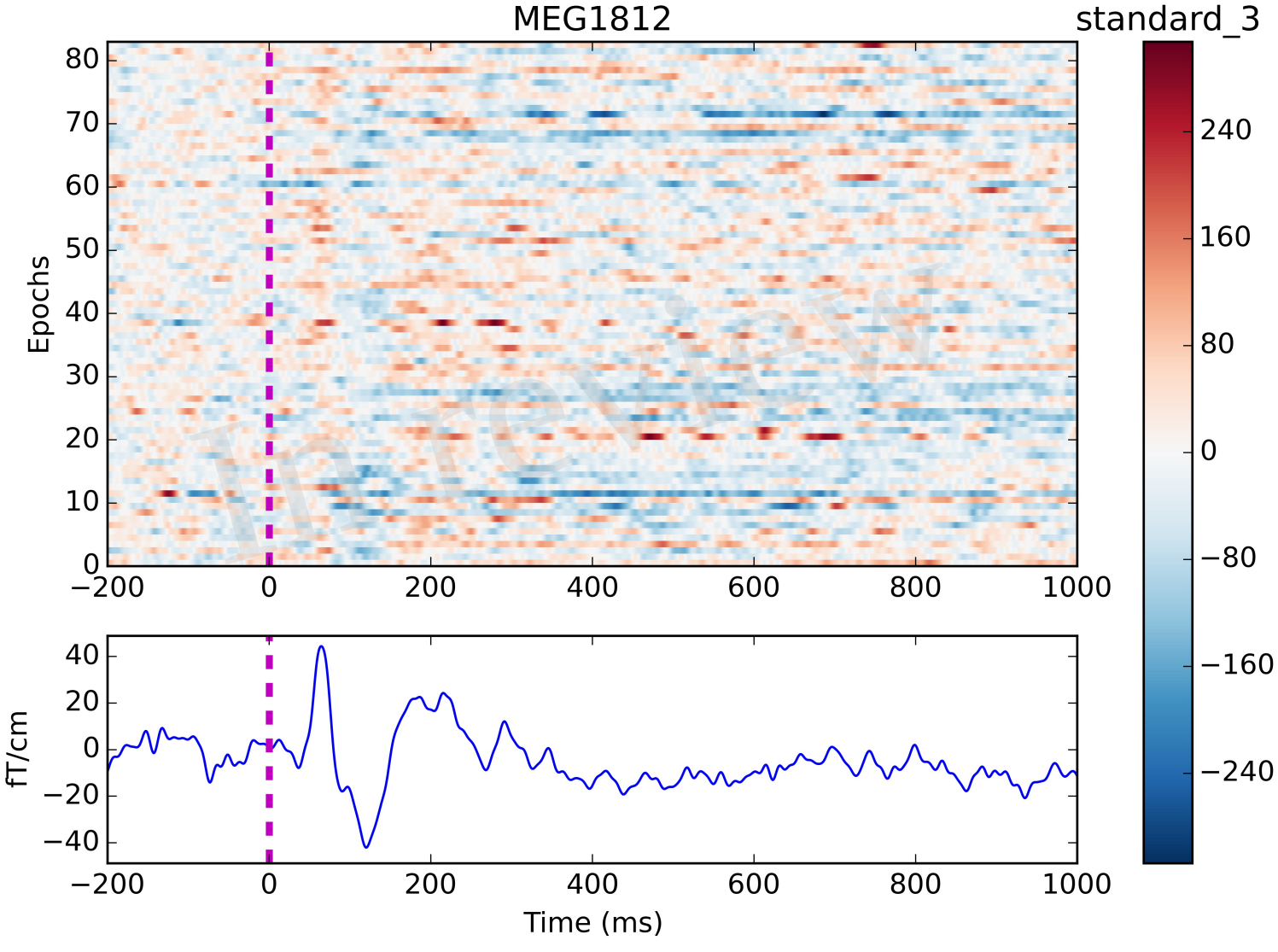

Figure 5.JPEG

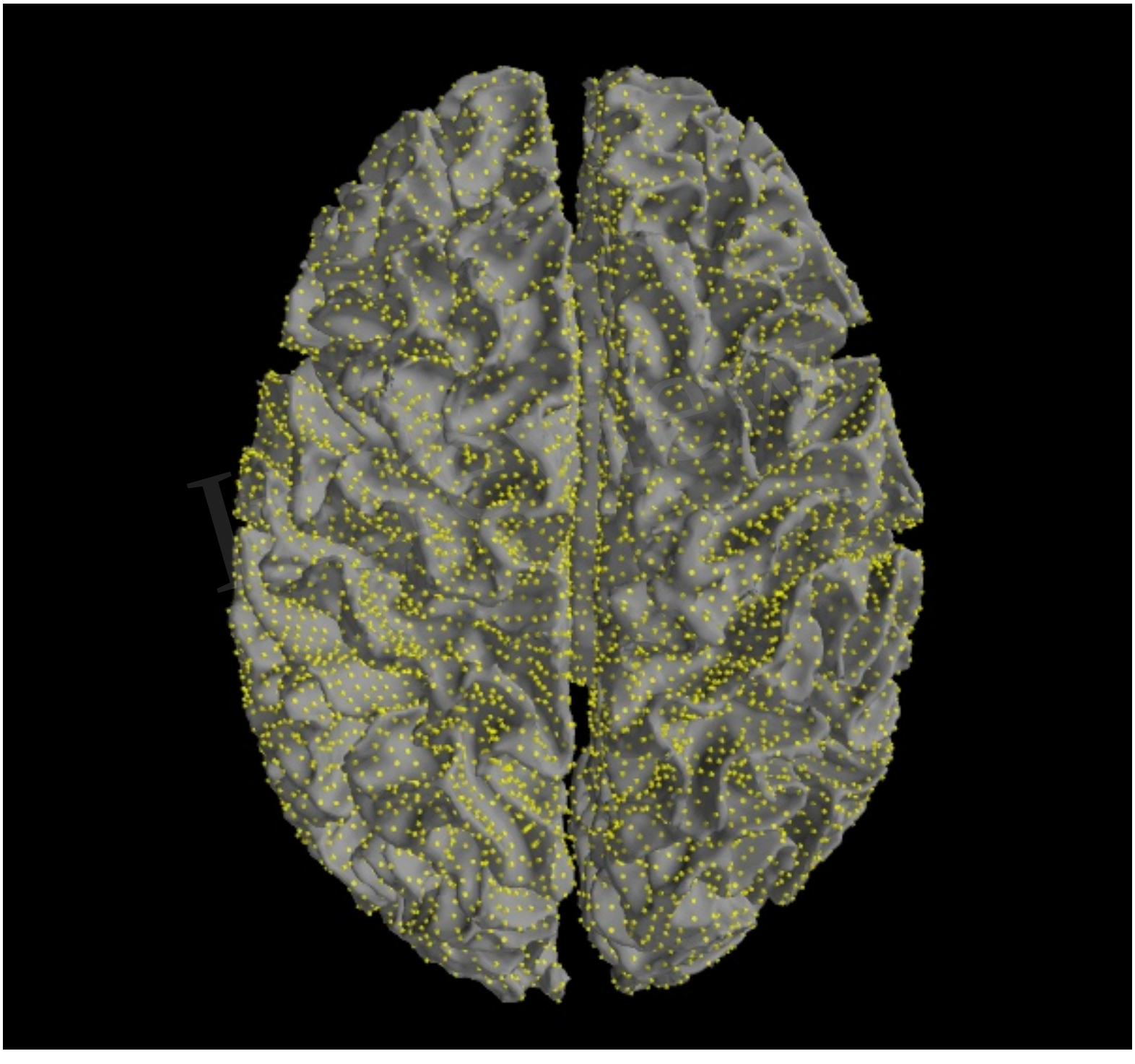

Figure 6.JPEG

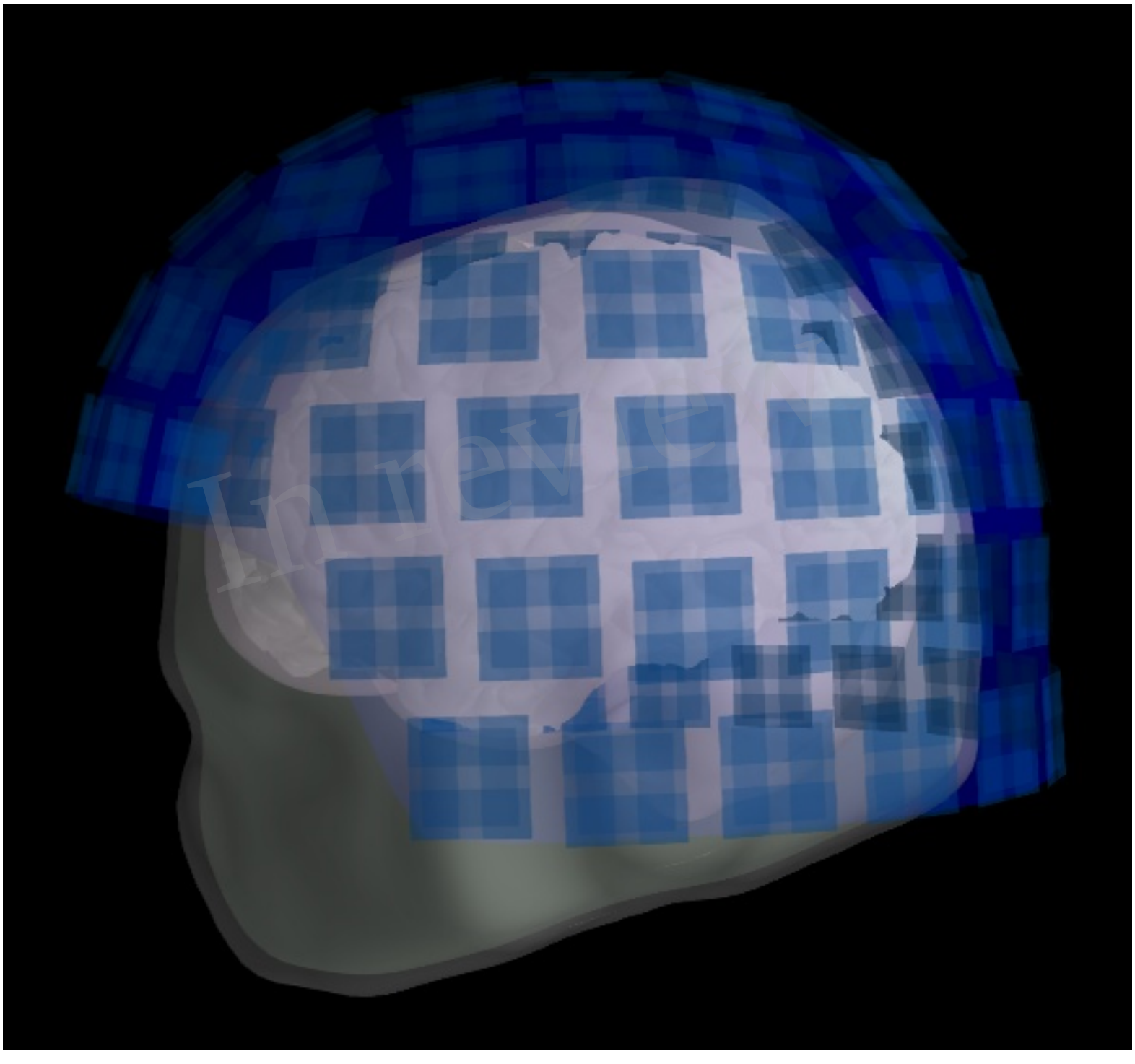

Figure 7.JPEG

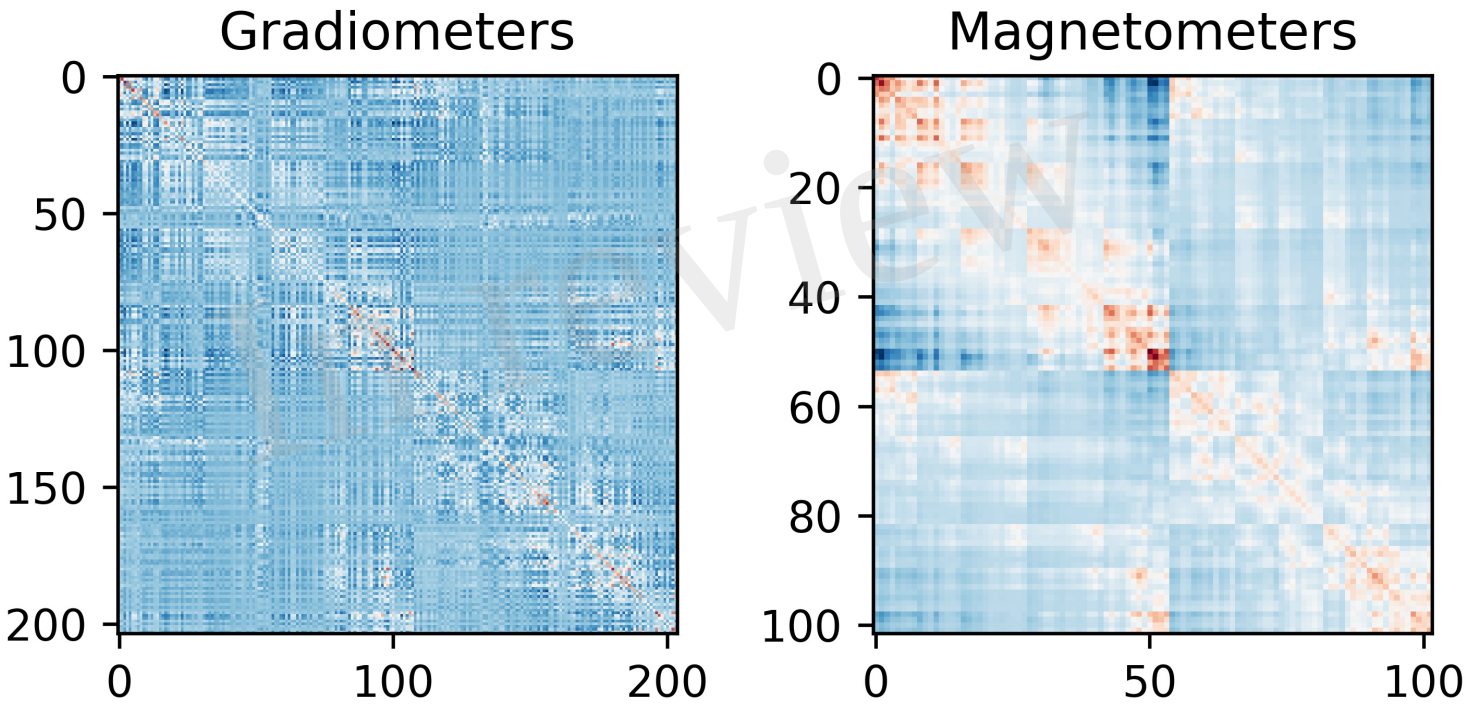

Figure 8.JPEG

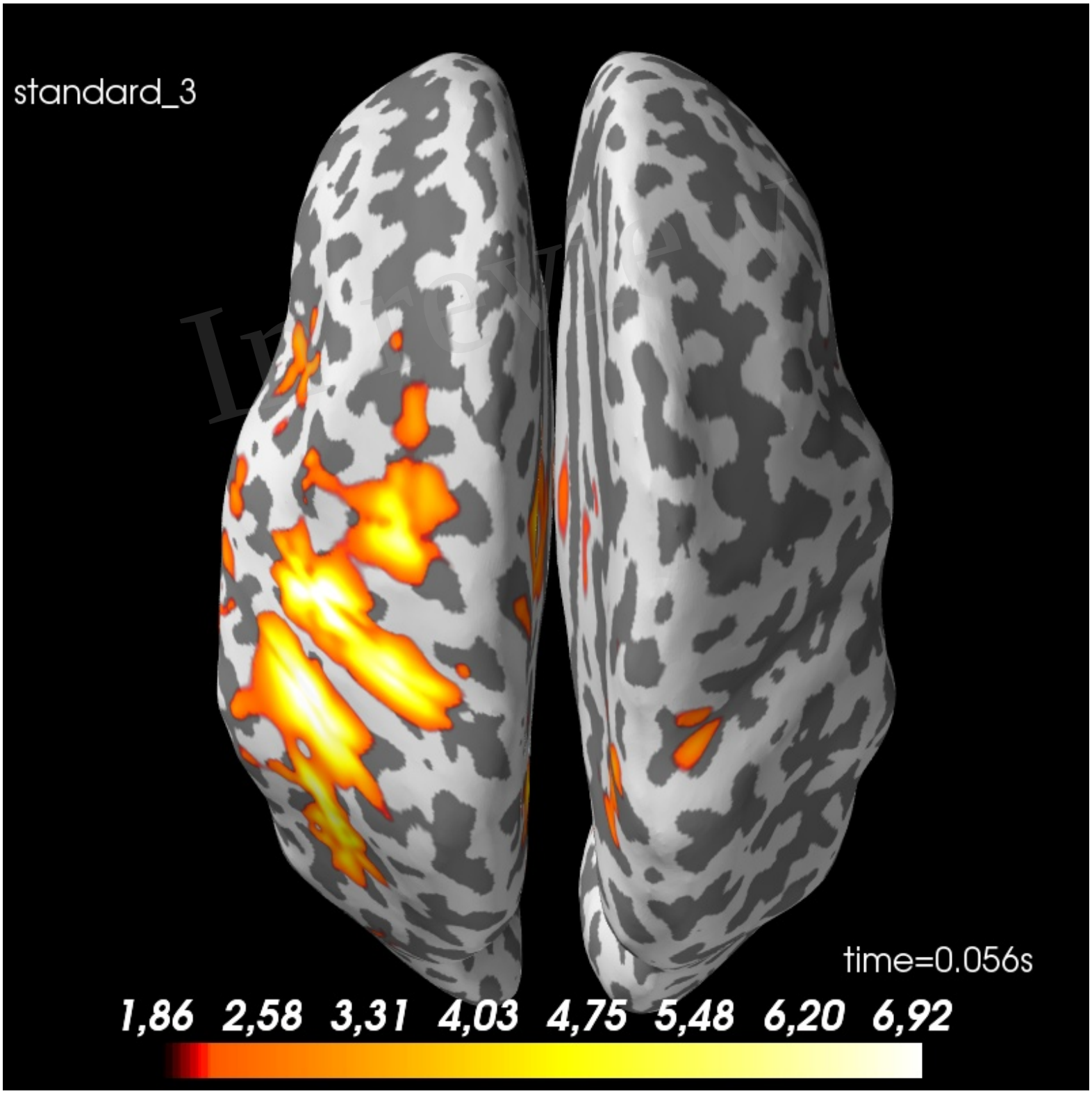

Figure 9.JPEG

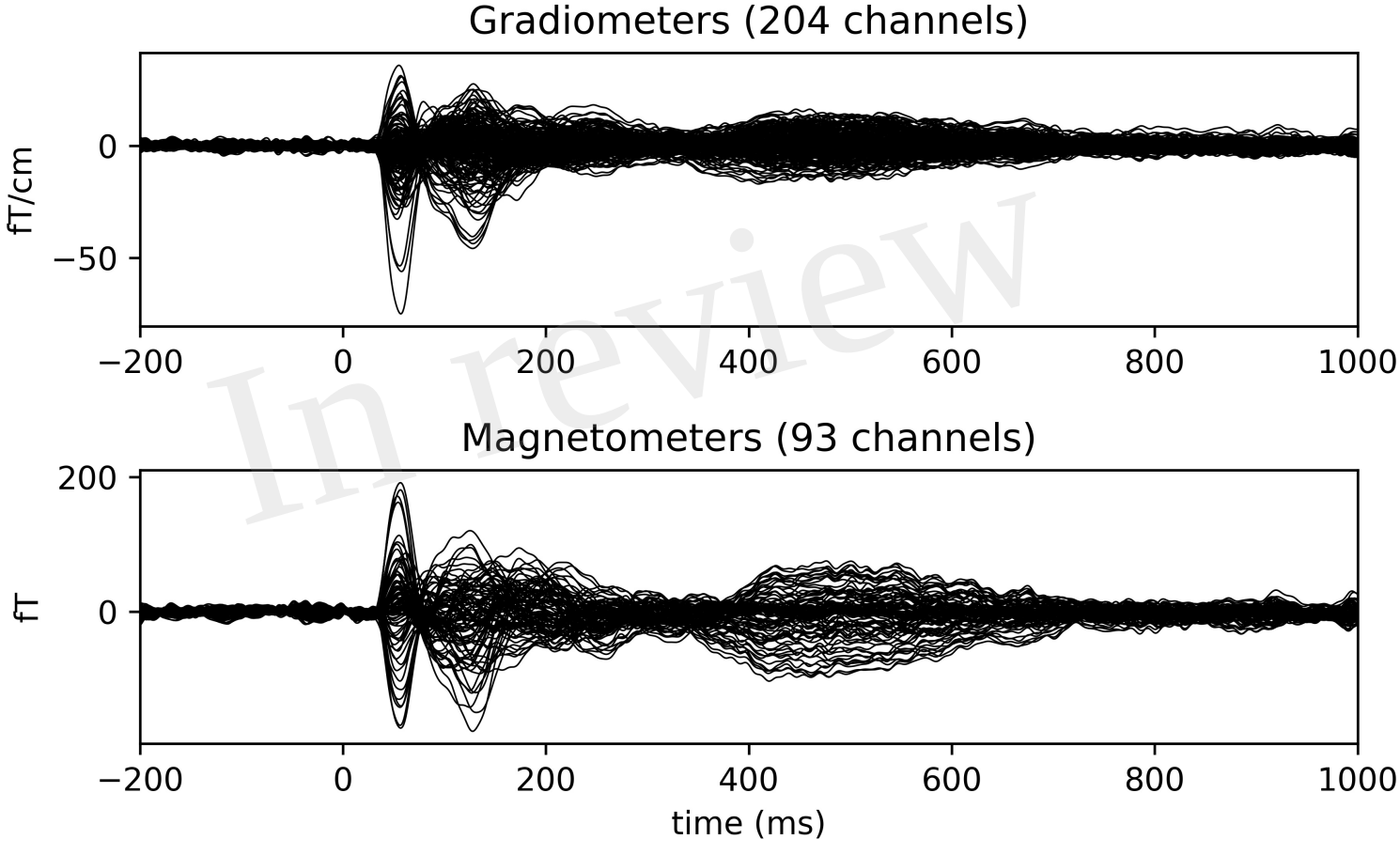

Figure 10.JPEG

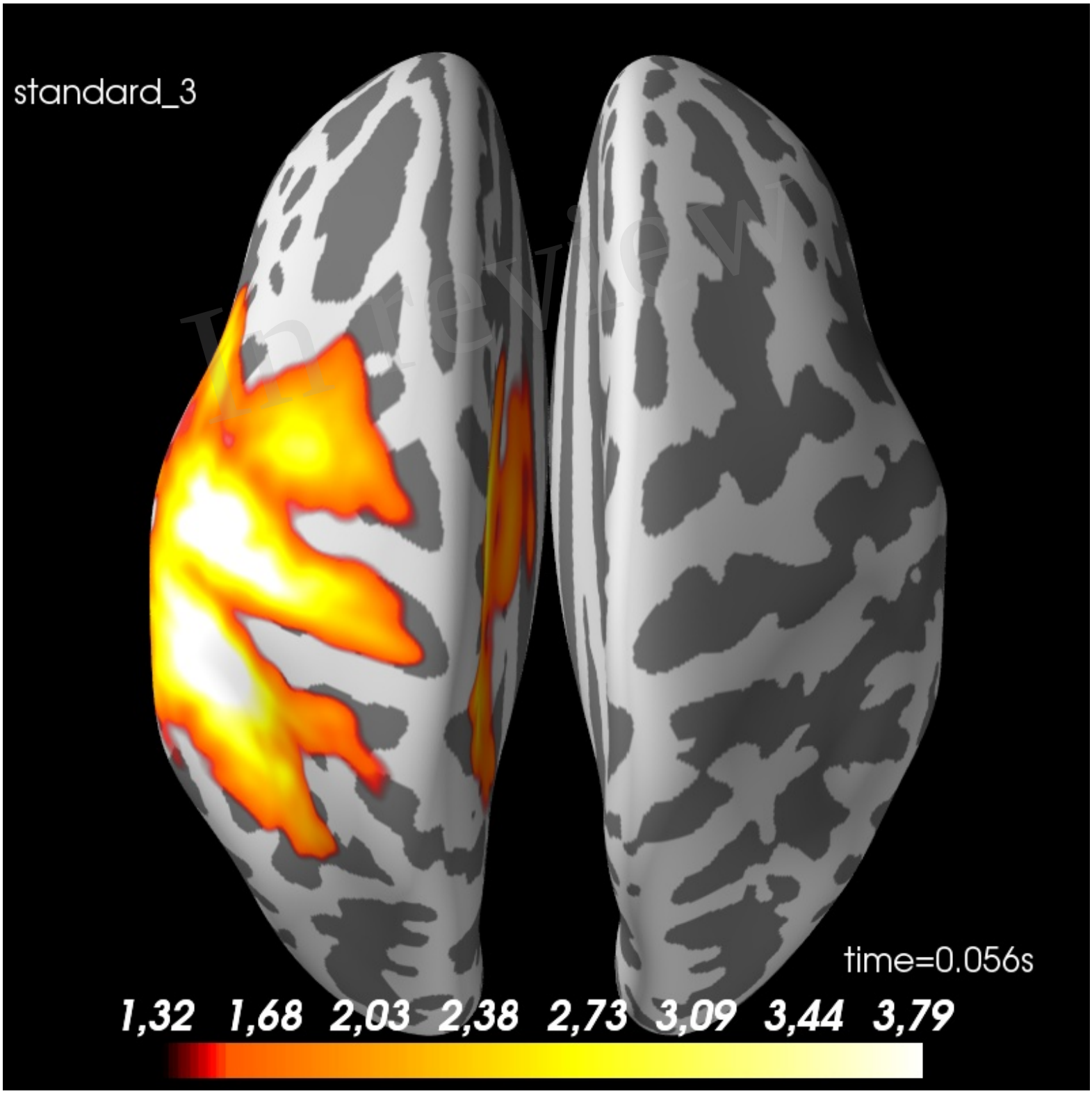

Figure 11.JPEG