Software and resources for experiments and data analysis of MEG and EEG data

Data from magnetoencephalography (MEG) and electroencephalography (EEG) is extremely rich and multifaceted. For example, in a standard MEG recording with 306 sensors and a sampling rate of 1,000 Hz, 306,000 data points are sampled every second. To be able to answer the question, which was the ultimate reason for acquiring the data, thus necessitates efficient data handling. Luckily, several software packages have been developed for handling MEG and/or EEG data. To name some of the most popular: MNE-Python; FieldTrip; Brainstorm; EEGLAB and SPM. These are all available under a public domain licence, meaning that they can be run, shared and modified by anyone. Commercial software released under proprietary licences include BESA and CURRY. It is important to be aware of that for clinical diagnosis of for example epilepsy, certified software is required FieldTrip, MNE-Python, Brainstorm, EEGLAB and SPM for example cannot be used for that. In this chapter, the emphasis will be on MNE-Python and FieldTrip. This will allow users of both Python and MATLAB (or alternatively GNU Octave to code along as the chapter unfolds. As a general remark, all that MNE-Python can do, FieldTrip can do and vice versa – though with some small difference. A full analysis going from raw data to a source reconstruction will be presented, illustrated with both code and figures with the aim of providing newcomers to the field a stepping stone towards doing their own analyses of their own datasets.

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.27988v2 | CC BY 4.0 Open Access | rec: 21 Oct 2019, publ: 21 Oct 2019

Software and resources for experiments and data analysis of MEG and EEG data

Lau M. Andersen[1,2*]

[1] Center of Functionally Integrative Neuroscience, Aarhus University, Denmark

[2] NatMEG, Karolinska Institutet, Stockholm, Sweden

[*] Corresponding author: lmandersen@cfin.au.dk

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.27988v2 | CC BY 4.0 Open Access | rec: 21 Oct 2019, publ: 21 Oct 2019

Software and resources for experiments and data analysis

Lau Møller Andersen

1. Introduction

Data from magnetoencephalography (MEG) and electroencephalography (EEG) is extremely rich and multifaceted. For example, in a standard MEG recording with 306 sensors and a sampling rate of 1,000 Hz, 306,000 data points are sampled every second. To be able to answer the question, which was the ultimate reason for acquiring the data, thus necessitates efficient data handling. Luckily, several software packages have been developed for handling MEG and/or EEG data. To name some of the most popular: MNE-Python (Gramfort et al. 2013); FieldTrip (Oostenveld et al. 2011); Brainstorm (Tadel et al. 2011); EEGLAB (Delorme and Makeig 2004) and SPM (Friston et al. 2007). These are all available under a public domain licence, meaning that they can be run, shared and modified by anyone. Commercial software released under proprietary licences include BESA (besa.de) and CURRY (compumedics.com.au/products/curry/). It is important to be aware of that for clinical diagnosis of for example epilepsy, certified software is required FieldTrip, MNE-Python, Brainstorm, EEGLAB and SPM for example cannot be used for that.

In this chapter, the emphasis will be on MNE-Python and FieldTrip. This will allow users of both Python (python.org) and MATLAB (mathworks.com), (or alternatively GNU Octave (gnu.org/software/octave/) to code along as the chapter unfolds. As a general remark, all that MNE-Python can do, FieldTrip can do and vice versa – though with some small difference. A full analysis going from raw data to a source reconstruction will be presented, illustrated with both code and figures. Before going to the software, however, some general remarks about single subject analysis and group analysis together with a short recapitulation of MEG and EEG sensor space and source space are in place.

1.1. Single subject analysis and group analysis

MEG and EEG (henceforth when no distinction between the two is needed, MEEG will be used) studies often focus on questions or hypotheses about how different experimental factors influence patterns of MEEG activity, and by inference the underlying brain activity giving rise to these patterns. By designing contrasts between experimental factors, the unique contributions of each experimental factor can be singled out. In principle, studies on single subjects would be sufficient to discover these unique contributions, but there are two limiting factors. The first one is that MEEG signals of interest are very weak in terms of signal-to-noise ratio (SNR), and the second is that most scientific questions aspire to say something about the population as a whole. With a group of subjects, a statistical inference to the general population is possible, and the SNR increases significantly when more subjects are included. For group analyses, it becomes even more necessary to efficiently handle data processing and analysis, because now beside the space (sensors) and time (sampling rate) dimensions, furthermore the subject dimension has to be considered, increasing the complexity of the data even more. Single subject data has to be processed individually however

before group measures can be applied. Therefore, a full analysis will be run on data from a single subject, followed by a short discussion of relevant literature for analysing groups of subjects.

1.2. Sensor space and source space

An important distinction in MEEG is the one between the sensor space and the source space. Simplifying the matter, the sensor space is the space in which the data is actually recorded, in gradiometers and magnetometers for MEG and in electrodes for EEG. The source space is a model of the underlying source configuration in the brain that gives rise to the pattern of activity in sensor space, also called source localization. Thus, there is nothing in the source space, which is also not in the sensor space. The source space is rather another perspective on the data in the sensor space. Historically, most EEG studies have been analysed in only the sensor space, whereas MEG studies more often, but not always, include the source space. The reason that this distinction is emphasized is that the processing of MEG and EEG data respectively differs the most when bringing the data from sensor space to source space.

1.3. Unique issues to MEG and EEG respectively

Analysis of MEG and EEG data respectively differ mainly in two regards. First, how the sensors are fixed relative to the head of the subject, and second, how the electric potentials and the magnetic fields respectively spread through the brain, skull and skin compartments of the head. In MEG, the sensors are fixed in a helmet into which the subject inserts his head, whereas in EEG a cap with electrodes is fixed to the head of the subject. Thus, MEG sensors will be in different positions relative to the head shapes of subjects, whereas for EEG due to the standard, e.g. the 10-20 system, ways of arranging electrodes on the cap, the electrodes will share the same positions relatively speaking even if subject head shapes differ. The problem of the fixed array of MEG is amplified if the subject moves; in MEG that would mean that MEG sensors are moving relative to the subject's head, whereas in EEG the cap just follows the movement of the head, unless the cap itself is moved.

Now for the second issue: the spread of the magnetic field can be assumed to spread homogeneously throughout the brain, skull and skin compartments of the head, whereas the electric potential spreads through these compartments with different conductivities. Due to the low conductivity of the skull, the electric potential is smeared on the scalp. Thus to create precise source models based on EEG data, precise and careful anatomical modelling is needed. For MEG, due to the homogeneous spread of the magnetic field, creating precise source models is more simple. Furthermore, creating source models of the data also eliminates the issue of sensors being in different positions relative to the heads of subjects. This is because the source space of each subject can be expressed in a shared template space, which can be compared directly between subjects.

These issues will be emphasized in the analysis below, but otherwise MEG and EEG analyses can be carried out with very similar strategies.

2. Data analysis

The data analysis examples below will use open data from Simanova et al. (2010) (EEG) and Lam et al. (2016) (MEG). The scripts used are similar to earlier protocol publications of mine (Andersen 2018a, b). Scripts were tested with FieldTrip version 20190716 and MNE-Python version 0.18.2

respectively and can be found in my repository on GitHub:
https://github.com/ualsbombe/language_electrified.

2.1. EEG single subject analysis

The data from a single subject used in this analysis was retrieved from ftp://ftp.fieldtriptoolbox.org/pub/fieldtrip/tutorial/SubjectEEG.zip. The data is associated with a study on how representations of words might differ dependent on whether they were elicited by pictures, spoken words or written words. A short description of the dataset and the recording will follow. For further details, the reader is referred to the original article (Simanova et al. 2010). The relevant script is called *single_subject_analysis.m* and is found in the *language_EEG* folder of the GitHub repository, https://github.com/ualsbombe/language_electrified.

2.1.1. Short description of dataset

Three semantic categories of words were used: *animals* and *tools*, which were the target categories of analysis, and a third task category that differed between subjects, which was either *clothing* or *vegetables*. Each category contained four exemplars, and each exemplar in each target category was presented eighty times in each of three modalities: *auditory*, *visual* (picture) and *orthographical*. Task exemplars were repeated sixteen times and required a button press with the right index finger. Visual and orthographical stimuli were presented for 300 ms and were followed by a blank screen with a duration between 1,000-1,200 ms. A fixation cross was presented during auditory stimulation, and stimulation was followed by a blank screen with a similar duration to the visual and orthographic modalities. The subject whose dataset explored here was exposed to the *clothing* category. The task category will not be analysed, but event-related responses will be calculated for each of the remaining six possible combinations, that is: three modalities and two target semantic categories.

2.1.2. EEG recording

EEG was continuously registered using a 64-channel ActiCap system (Brain Products GmbH) online bandpass filtered at 0.2-200 Hz and sampled at 500 Hz. 60 electrodes were placed equidistantly on the scalp. The data was registered against a reference on the right mastoid. An additional electrode measure the voltage on the left mastoid. The electrooculogram (EOG) was measured using electrodes placed horizontally and vertically around the eyes.

2.1.3. Scripting

Here, I am going to follow the analysis steps applied by Simanova et al. (2010). First, a recommendation – make sure that you are producing easily reproducible scripts, so you and others can replicate the analysis or apply changes to the analysis without having to start all over. A general piece of advice is to always start off cleaning the workspace and restoring the path of MATLAB to your default (Code Snippet 1). This way you are making sure that the script is self-contained, and that your analysis will not be dependent on variables create elsewhere.

```
clear variables
restoredefaultpath

% change "home_dir", "data_dir", project_dir and "fieldtrip_dir" such that
% it applies to your setup.
% If you prefer having the "fieldtrip_dir" on your default path
% delete the lines about "fieldtrip_dir" and adding "fieldtrip_dir" to your
% path. Do remember to run "ft_defaults". This makes sure that the
% appropriate FieldTrip subdirectories are loaded
home_dir = '/home/lau/';
project_dir = fullfile(home_dir, 'analyses', 'fieldtrip_datasets', ...
                    'language_EEG');
data_dir = fullfile(project_dir, 'data');
figures_dir = fullfile(project_dir, 'figures');
fieldtrip_dir = fullfile(home_dir, 'matlab', 'fieldtrip');

addpath(fieldtrip_dir);
ft_defaults
```

Code Snippet 1 An example of how a script can be started, making sure that all variables are cleaned, and that MATLAB path used is the default path. Note that changes have to be made to fit the user's directory structure. Ideally, this should be the only change that should be made between operating systems. On macOS the "home_dir" is usually /Users/<your_name>" and on Windows it is C:\Users\<your_name>.

To explore the EEG data and the trigger codes in there, we can run the following Code Snippet (2) and look at its output (Output 1).

```
cfg = [];
cfg.dataset = fullfile(data_dir, 'subj2.vhdr');
cfg.trialdef.eventtype = '?';

temp = ft_definetrial(cfg);
```

Code Snippet 2 Reading in the information about the triggers contained in the data. These indicate among other things when responses were made and when stimulations appeared. Setting *eventtype* to *'?'* lists all the available triggers.

```
Warning: no trialfun was specified, using ft_trialfun_general
evaluating trialfunction 'ft_trialfun_general'
reading the header from '/home/lau/analyses/fieldtrip_datasets/language_EEG/data/subj2.vhdr'
reading the events from '/home/lau/analyses/fieldtrip_datasets/language_EEG/data/subj2.vhdr'
the following events were found in the datafile
event type: 'New Segment'
with event values:

event type: 'Response'
with event values: 'R  8'

event type: 'Stimulus'
with event values: 'S  1' 'S 12' 'S 13' 'S 21' 'S 27' 'S111' 'S112' 'S113' 'S121' 'S122' 'S123' 'S131' 'S132' 'S133' 'S141'
'S142' 'S143' 'S151' 'S152' 'S153' 'S161' 'S162' 'S163' 'S171' 'S172' 'S173' 'S181' 'S182' 'S183' 'S211' 'S212' 'S213' 'S221'
'S222' 'S223' 'S231' 'S232' 'S233' 'S241' 'S242' 'S243'

no trials have been defined yet, see FT_DEFINETRIAL for further help
found 1570 events
created 0 trials
the call to "ft_definetrial" took 1 seconds and required the additional allocation of an estimated 0 MB
```

Output 1 Output from Code Snippet 2. In total there 1,570 events, and of two types "Stimulus" and "Response".

The triggers that we will use here are listed in Table 1:

	Orthographic	Visual	Auditory
Animals	S111; S121; S131; S141	S112; S122; S132; S142	S113; S123; S133; S143
Tools	S151; S161; S171; S181	S152; S162; S172; S182	S153; S163; S173; S183

Table 1 The trigger codes for the four different kinds of stimuli for each of the two target categories for each of the three modalities.

Note also that each Code Snippet presented here is initiated with two percentage signs. This means that MATLAB arranges this as a *block* of code. All the lines in a code can be evaluated by placing the cursor inside the block and pressing *Ctrl-ENTER*. This is a good way of dividing your analysis into logical steps. Note also that most FieldTrip functions (starting with *ft*) take a *configuration* as its first argument. The configuration variable is as a standard named *cfg* and is a *structure* variable. FieldTrip functions, e.g. *ft_definetrial* (Code Snippet 3), expect and allow different *fields* to be present. In Code Snippet 3 *dataset* and *trialdef* are *fields* of *cfg*. Note that *fields* can have their own *fields* such as *eventtype* here being a *field* of *trialdef*, which in turn is a *field* of *cfg*.

2.1.4. Reading in and segmenting data

We can take the raw data and segment it into epochs around events of interest. We follow Simanova et al. (2010) in having segments of 1 s with 300 ms of data before the stimulation and 700 ms of data after the stimulation. First, we define the trial structure (Code Snippet 3) and subsequently we preprocess the data.

```
orthographic_animals = {'S111' 'S121' 'S131' 'S141'};
orthographic_tools   = {'S151' 'S161' 'S171' 'S181'};

visual_animals       = {'S112' 'S122' 'S132' 'S142'};
visual_tools         = {'S152' 'S162' 'S172' 'S182'};

auditory_animals     = {'S113' 'S123' 'S133' 'S143'};
auditory_tools       = {'S153' 'S163' 'S173' 'S183'};

cfg = [];
cfg.dataset = fullfile(data_dir, 'subj2.vhdr');
cfg.trialdef.eventtype  = 'Stimulus';
cfg.trialdef.eventvalue = [orthographic_animals orthographic_tools ...
                           visual_animals visual_tools ...
                           auditory_animals auditory_tools];
cfg.trialdef.prestim    = 0.300; % s
cfg.trialdef.poststim   = 0.700; % s

cfg_def = ft_definetrial(cfg);
```

Code Snippet 3 Defining a trial structure using the raw dataset (*cfg.dataset*) and indicating what event type and what event values (Output 1) should be used. *cfg_def* contains 1,570 events, but only 1,200 were defined as trials of interest. Explore *cfg_def* for further details.

The next step will be to preprocessing and read in the data based on the trial definition just given (Code Snippet 4; *cfg_def*).

```
% re-referencing
cfg_def.reref          = 'yes'; % create a new reference
cfg_def.channel        = 'all'; % apply reference to all EEG channels
cfg_def.implicitref    = 'M1'; % the non-recorded reference
cfg_def.refmethod      = 'avg'; % method of referencing
cfg_def.refchannel     = {'M1' '53'}; % the average (see "refmethod" above) of
                                      % these two channels will be the new
                                      % reference; 53 is M2, the recorded
                                      % reference channel

% filtering and demeaning (baselining)
cfg_def.bpfilter       = 'yes'; % apply a band pass filter
cfg_def.bpfreq         = [1 30]; % Hz; range of band pass filter
cfg_def.demean         = 'yes'; % divide the amplitude of each trial by the
                                %
                                % mean amplitude in the time window chosen in
                                % "baselinewindow" below
cfg_def.baselinewindow = [-Inf 0]; % from beginning to time zero

data = ft_preprocessing(cfg_def);
```

Code Snippet 4 Reading in and preprocessing the data, here re-referencing and band-pass filtering

Voltage is always expressed as relative to a reference point. In the recording this was M2 (53), but we are now choosing it to be the average of M1 and M2. Afterwards we are applying a band-pass filter from 1-30 Hz, aiming to attenuate any part of the signal outside this range. Event-related responses are seldom outside this range. The applications of *ft_preprocessing* on *cfg_def* results in a data *structure* of 1,200 trials (Output 2).

```
data =

          hdr: [1x1 struct]
        label: {1x65 cell}
         time: {1x1200 cell}
        trial: {1x1200 cell}
      fsample: 500
   sampleinfo: [1200x2 double]
    trialinfo: [1200x1 double]
          cfg: [1x1 struct]
```

Output 2 The preprocessed data. Among other *fields*, it contains *trial*, which has the data. Each cell has 65 rows and 500 columns, corresponding to 65 electrodes and 500 time points. *trialinfo* contains 1,200 numbers indicating the trigger value for each trial.

2.1.5. Extracting EOG channels

In this analysis, so-called bipolar EOG channels were created based on the EEG electrodes horizontally and vertically around the eyes (Code Snippet 5).

```
cfg             = [];
cfg.channel     = {'50' '64'};% electrodes placed vertically around the left eye
cfg.reref       = 'yes';
cfg.implicitref = []; % means no implicit reference
cfg.refchannel  = '50'; % reference '50' and '64' to '50
cfg.refmethod   = 'avg';

eog_vertical = ft_preprocessing(cfg, data);

% keep only one channel and rename it to EEGv
cfg         = [];
cfg.channel = '64'; % the channel to keep

eog_vertical = ft_selectdata(cfg, eog_vertical);
eog_vertical.label = {'EOGv'}; % renaming it

% create horizontal EOG (logic similar to above)
cfg             = [];
cfg.channel     = {'51' '60'};
cfg.reref       = 'yes';
cfg.implicitref = [];
cfg.refchannel  = '51';
cfg.refmethod   = 'avg';

eog_horizontal = ft_preprocessing(cfg, data);

% keep only channel and rename it to EEGh (logic similar to above)
cfg         = [];
cfg.channel = '60';

eog_horizontal = ft_selectdata(cfg, eog_horizontal);
eog_horizontal.label = {'EOGh'};

% rename '53' to M2 for consistency
channel_1_index = strcmp(data.label, '53');
data.label{channel_1_index} = 'M2';

% keep only the EEG channels from the original data
cfg         = [];
cfg.channel = setdiff(1:60, [50 51 60 64]); % removing channels 50, 51, 60 & 64
                                            % since they do not contain EEG
                                            % signal, but have been
                                            % converted to EOG
data = ft_selectdata(cfg, data);

% finally append EOGv and EOGh data to EEG data
cfg = [];
data = ft_appenddata(cfg, data, eog_vertical, eog_horizontal);
```

Code Snippet 5 Creating the bipolar EOG channels, removing the extraneous channels and finally appending the data, such that it consists of the 57 EEG channels and the 2 EOG channels.

2.1.6. Rejecting based on objective threshold

We follow Simanova et al. (2010) in using an objective threshold of 150 µV meaning that they rejected each epoch containing a voltage above this value at any time, at any channel, be it EEG or EOG (Code Snippet 6). We make a common average reference of the EEG data as well, which is necessary to do for the way that FieldTrip has implemented source reconstruction.

```
threshold = 150; % µV
n_trials = length(data.trial);
bad_indices = []; % initialize empty - we don't know how many bad trials
                        % there will be

for trial_index = 1:n_trials

    trial = data.trial{trial_index};  % get trial n
    % check whether threshold is exceeed in either the negative or (|) the
    % positive direction for any of the time points on any of the
    % electrodes
    threshold_exceeded = any(any(trial > threshold | trial < -threshold));
    if threshold_exceeded
        % add the index of each trial that exceeded the threshold
        bad_indices = [bad_indices trial_index]; %#ok<AGROW>
    end

end

% keep only the good indices
good_indices = setdiff(1:n_trials, bad_indices);

cfg =        [];
cfg.trials = good_indices;

cleaned_data = ft_selectdata(cfg, data);

% and make common reference

cfg = [];
cfg.reref = 'yes';
cfg.refchannel = 'all';

cleaned_data = ft_preprocessing(cfg, cleaned_data);
```

Code Snippet 6 Code used for applying an objective threshold to the data and re-referencing to a common average.

2.1.7. Create the Event-Related Potentials

The final processing step will be to calculate the Event-Related Potentials (ERPs). An ERP is created simply by averaging the trials (Code Snippet 7) for each time point. The code below (Code Snippet 7) will create an ERP for each of the six categories.

```
categories = {
                [111 121 131 141] % orthographic_animals
                [151 161 171 181] % orthographic_tools

                [112 122 132 142] % visual_animals
                [152 162 172 182] % visual_tools

                [113 123 133 143] % auditory_animals
                [153 163 173 183] % auditory_tools
             };

n_categories = length(categories);
ERPs = cell(1, n_categories);

for category_index = 1:n_categories

    category = categories{category_index};

    cfg                 = [];
    cfg.trials          = ismember(cleaned_data.trialinfo, category);
    cfg.covariance      = 'yes';
    cfg.covariancewindow = 'prestim';

    ERPs{category_index} = ft_timelockanalysis(cfg, cleaned_data);

end
```

Code Snippet 7 Creating the ERPs for each of the two semantic categories across modalities

2.1.8. Plotting the ERPs

We can subsequently plot the ERPs. Here, three kinds are plotted (Code Snippet 8; Fig. 1). Some extra text and comments have been added to inspire readers, but rough plots for exploring can be invoked by a minimal *cfg* (Code Snippet 9).

```
mplot = figure('units', 'normalized', 'outerposition', [0 0 1 1]);
cfg             = [];
cfg.layout      = 'easycapM10.mat';
cfg.xlim        = [-0.300 0.698]; % s
cfg.ylim        = [-10 10]; % µV
cfg.limittext   = sprintf(['Auditory Response\nx: Time: ' ...
                        num2str(cfg.xlim(1), '%0.3f') '-' ...
                        num2str(cfg.xlim(2), '%0.3f') ...
                        ' s\n\t\ty: Electric Potential ' ...
                        num2str(cfg.ylim(1), '%0.1f') '-' ...
                        num2str(cfg.ylim(2), '%0.1f') ' µV']);
cfg.fontsize    = 25;
cfg.showscale   = 'no';

ft_multiplotER(cfg, ERPs{5});

%single channel plot
splot = figure('units', 'normalized', 'outerposition', [0 0 1 1]);
cfg           = [];
cfg.xlim      = [-0.300 0.698]; % s
cfg.ylim      = [-10 10]; % µV
cfg.layout    = 'easycapM10.mat';
cfg.channel   = '1';
cfg.title     = sprintf(['Auditory Response\nCentral Electrode (' ...
                         cfg.channel ')']);
cfg.fontsize  = 35;
cfg.linewidth = 3;
xlabel('Time (s)')
ylabel('Electric Potential (µV)')

ft_singleplotER(cfg, ERPs{5});
hold on
plot([cfg.xlim(1) cfg.xlim(2)], [0 0], 'k--'); % horizontal line
plot([0 0], [cfg.ylim(1) cfg.ylim(2)], 'k--'); % vertical line

% topoplot
tplot = figure('units', 'normalized', 'outerposition', [0 0 1 1]);
cfg          = [];
cfg.layout   = 'easycapM10.mat';
cfg.xlim     = [0.120 0.120]; % s
cfg.zlim     = [-4 4]; % µV
cfg.comment  = sprintf(['Auditory N1\nTime: ' ...
                        num2str(cfg.xlim(1), '%0.3f') ' s']);
cfg.fontsize = 25;
c = colorbar;
c.Label.String = 'Electric Potential (µV)';
c.Label.FontSize = 40;

ft_topoplotER(cfg, ERPs{5});
```

166

Code Snippet 8 Making (pretty) plots of ERPs

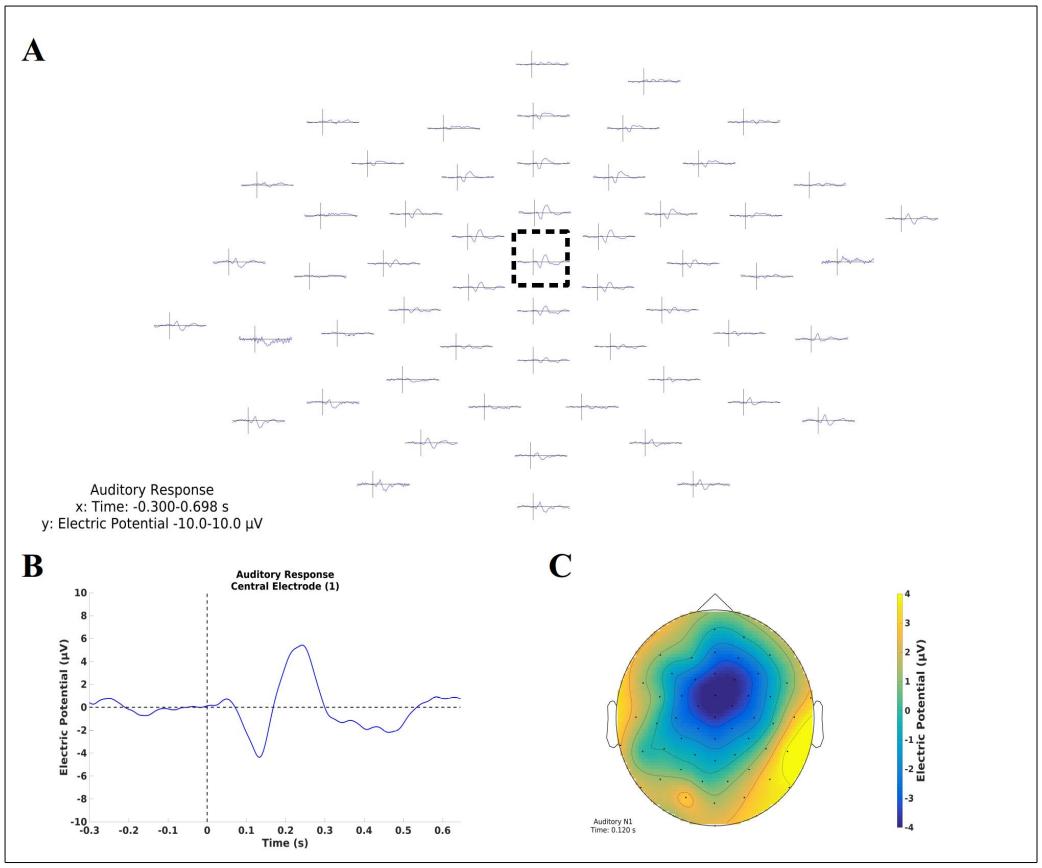


Fig. 1 Plots of the auditory response. **A:** Plots of all channels, where it is clear that the biggest responses are centred around the central sensors. **B:** A central electrode (1), highlighted in **A**. It shows the N1 auditory response plus a second response after ~250 ms. **C:** The N1 response (120 ms) as a topography clearly centred on central sensors

```
figure('units', 'normalized', 'outerposition', [0 0 1 1]);
cfg             = [];
cfg.layout      = 'easycapM10.mat';

ft_multiplotER(cfg, ERPs{5});
%single channel plot
figure('units', 'normalized', 'outerposition', [0 0 1 1]);
cfg           = [];

cfg.layout    = 'easycapM10.mat';
cfg.channel   = '1';

ft_singleplotER(cfg, ERPs{5});

% topoplot
figure('units', 'normalized', 'outerposition', [0 0 1 1]);
cfg          = [];
cfg.layout   = 'easycapM10.mat';
cfg.xlim     = [0.120 0.120]; % s

ft_topoplotER(cfg, ERPs{5});
```

167

Code Snippet 9 Simple code for invoking (rough) plots

2.1.9. Difference waves

We can also compute the differences between *animals* and *tools* for each of the three modalities (orthographic, visual and auditory) (Code Snippet 10).

```
categories = {
                [111 121 131 141] % orthographic_animals
                [151 161 171 181] % orthographic_tools

                [112 122 132 142] % visual_animals
                [152 162 172 182] % visual_tools

                [113 123 133 143] % auditory_animals
                [153 163 173 183] % auditory_tools
             };

n_categories = length(categories);
differences_semantic_categories = cell(1, n_categories/2);

for category_index = 2:2:n_categories

    cfg = [];
    cfg.operation = 'x1 - x2'; % subtract animals from tools
    cfg.parameter = 'avg';

    differences_semantic_categories{category_index/2} = ...
        ft_math(cfg, ERPs{category_index}, ERPs{category_index-1});

end
```

Code Snippet 10 Code for creating difference waves.

These can be plotted using the same tools as above, but will not be done here.

2.1.10. Source reconstruction

Below, I'll give an example of source reconstruction using templates. For optimal source reconstruction, it is preferable to have a magnetic resonance image of subjects' brains and to accurately know the position of electrodes on the subjects' heads. This can be known for examples by digitizing the positions of electrodes using a Polhemus FASTRAK or by a reconstruction procedure based on using photographs (Clausner et al. 2017). Here, I'll use the templates distributed with FieldTrip (Code Snippet 11).

```
headmodel = ft_read_headmodel('standard_bem.mat');
elec = ft_read_sens('easycap-M10.txt');
sourcemodel = ft_read_headshape('cortex_5124.surf.gii');

% realign electrodes
cfg           = [];
cfg.method    = 'project';
cfg.headshape = headmodel.bnd(1); % head surface

elec_realigned = ft_electroderealign(cfg, elec);

% get everything in SI units

headmodel = ft_convert_units(headmodel, 'm');
sourcemodel = ft_convert_units(sourcemodel, 'm');
elec = ft_convert_units(elec, 'm');
elec_realigned = ft_convert_units(elec_realigned, 'm');

% plot models

close all

quality_plot = figure('units', 'normalized', 'outerposition', [0 0 1 1]);
subplot(1, 2, 1)
hold on
ft_plot_headmodel(headmodel, 'facealpha', 0.3)
ft_plot_sens(elec, 'facecolor', 'red', 'label', 'label', ...
    'fontsize', 30, 'elecsize', 40)
ft_plot_mesh(sourcemodel)
view(-90, 0)
title('No projection');

subplot(1, 2, 2)
hold on
ft_plot_headmodel(headmodel, 'facealpha', 0.3)
ft_plot_sens(elec_realigned, 'facecolor', 'red', 'label', 'label', ...
    'fontsize', 30, 'elecsize', 40)
ft_plot_mesh(sourcemodel)
view(-90, 0)
title('Projected onto scalp')
```

```
headmodel = ft_read_headmodel('standard_bem.mat');
elec = ft_read_sens('easycap-M10.txt');
sourcemodel = ft_read_headshape('cortex_5124.surf.gii');

% realign electrodes
cfg           = [];
cfg.method    = 'project';
cfg.headshape = headmodel.bnd(1); % head surface

elec_realigned = ft_electroderealign(cfg, elec);

% get everything in SI units

headmodel = ft_convert_units(headmodel, 'm');
sourcemodel = ft_convert_units(sourcemodel, 'm');
elec = ft_convert_units(elec, 'm');
elec_realigned = ft_convert_units(elec_realigned, 'm');

% plot models

close all

quality_plot = figure('units', 'normalized', 'outerposition', [0 0 1 1]);
subplot(1, 2, 1)
hold on
ft_plot_headmodel(headmodel, 'facealpha', 0.3)
ft_plot_sens(elec, 'facecolor', 'red', 'label', 'label', ...
    'fontsize', 30, 'elecsize', 40)
ft_plot_mesh(sourcemodel)
view(-90, 0)
title('No projection');

subplot(1, 2, 2)
hold on
ft_plot_headmodel(headmodel, 'facealpha', 0.3)
ft_plot_sens(elec_realigned, 'facecolor', 'red', 'label', 'label', ...
    'fontsize', 30, 'elecsize', 40)
ft_plot_mesh(sourcemodel)
view(-90, 0)
title('Projected onto scalp')
```

Code Snippet 11 Loading a standard head model, the standard positions of the electrodes and a standard source model. The electrodes are subsequently projected onto the surface. It is extremely important that one always plots these three things together to assess whether things look reasonable. Otherwise, the source reconstruction will not be meaningful.

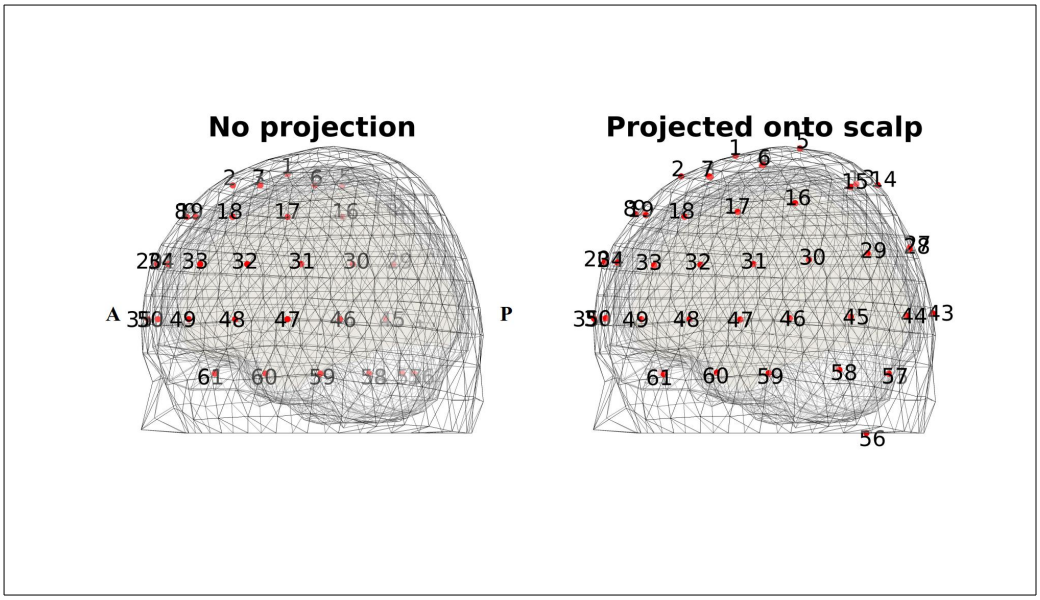


Fig. 2 The alignment of the source model (the brain inside the head), the head model (the skin, skull and brain compartments) and the electrodes. Note that without the projection, many of the electrodes are in wrong place, even inside the brain. Always plot to see if structures are properly aligned! A = Anterior, P = Posterior.

2.1.11. Lead field

The next step is to create the lead field (or the forward solution) (Code Snippet 12). This contains information how the sources (found in the source model) link to the electrodes. More precisely, it is calculated for each source, what electric potential each electrode would pick *given* that that source was active with a current of 1 Am. How the electric current propagates throughout the three compartments is dependent on the conductivities of the tissues that the current passes through, here skin, skull and brain are modelled. Since the conductivity of the skull is a lot lower than that of the skin and the brain, the electric potential is smeared out. It is therefore important to accurately model these different compartments for the EEG. In this case, I have used a template.

```
cfg = [];
cfg.elec = elec_realigned;
cfg.sourcemodel.pos = sourcemodel.pos;
cfg.sourcemodel.inside = 1:size(sourcemodel.pos, 1);
cfg.headmodel = headmodel;

leadfield = ft_prepare_leadfield(cfg);
```

Code Snippet 12 Code for creating the leadfield

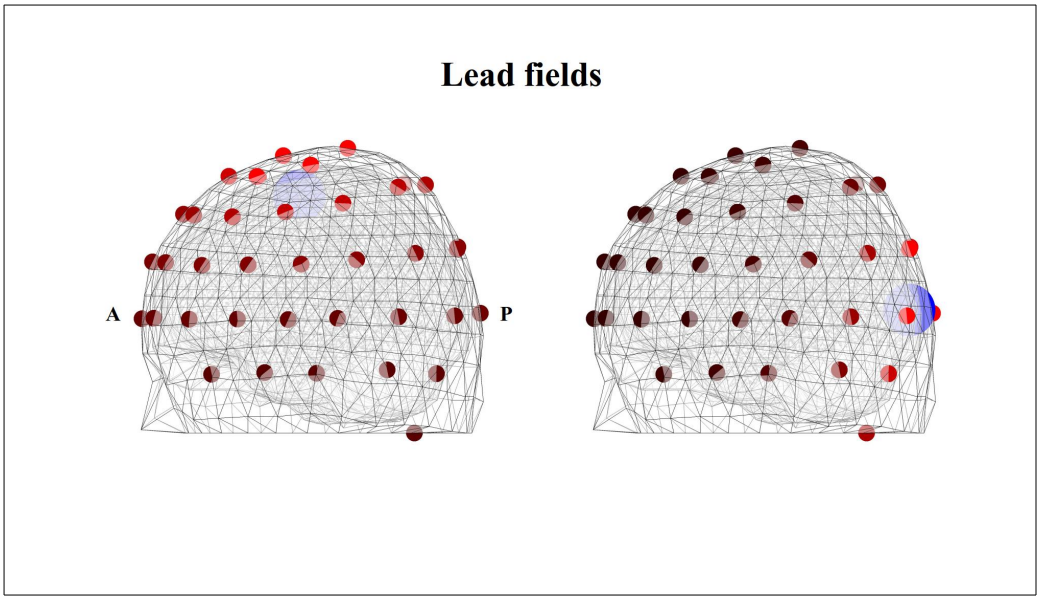


Fig. 3 Depiction of lead fields for two sources illustrated in light blue. Size is only for ease of identification. The lighter the red colour, the more electric potential is picked up at a given electrode. **Left**: a central source linking to a high degree to the electrodes just above it, but in general it links to all electrodes. **Right:** a posterior source linking to the highest degree to the electrodes just above it and less so to the others. On the frontal electrodes, it does not contribute much A = Anterior, P = Posterior

2.1.12. Compute the minimum norm estimate

In the next step (Code Snippet 13), we are going to compute the minimum norm estimate (MNE). The MNE computes the solution that explains the scalp pattern (e.g. Fig. 1C) for each time point. It needs information about where electrodes are (*cfg.elec)*, where sources are and how they are linked to the electrodes (the lead field) (*cfg.sourcemodel)*. It is necessary to regularise the data going into the MNE. If the data is not regularised, then even the noise will be fitted, since the MNE has to explain *all* the data. Fitting noise is called over-fitting. To avoid over-fitting, regularisation is thus applied, in effect smoothing the data (and the noise) over a larger region. Regularisation is controlled by *cfg.mne.lambda*. Higher lambda values mean more regularisation, thus more smoothing. Here, I have chosen only "little" regularisation, but do investigate what happens when lambda is increased. I have also scaled the source covariance with the noise covariance (*cfg.mne.scalesourcecov*). See Code Snippet 14 for how to plot the MNE on the cortical surface.

```
MNEs = cell(1, n_categories);

for category_index = 1:n_categories

    cfg                    = [];
    cfg.method             = 'mne';
    cfg.senstype           = 'eeg';
    cfg.elec               = elec_realigned;
    cfg.channel            = cleaned_data.label(1:57);
    cfg.sourcemodel        = leadfield;
    cfg.headmodel          = headmodel;
    cfg.mne.prewhiten      = 'no';
    cfg.mne.lambda         = 0.1;
    cfg.mne.scalesourcecov = 'yes';

    MNE = ft_sourceanalysis(cfg, ERPs{category_index});
    MNE.tri = sourcemodel.tri;

    MNEs{category_index} = MNE;

end
```

Code Snippet 13 Code for running minimum norm estimates on all six event related potentials.

```
close all

category_index = 5;

this_mne = MNEs{category_index};
this_mne.avg.pow_nAm = this_mne.avg.pow * 1e9;

cfg = [];
cfg.method = 'surface';
cfg.funparameter = 'pow_nAm';
cfg.funcolorlim = [0 50]; % nAm
cfg.funcolormap = 'parula';
cfg.latency = 0.100; % s
cfg.colorbar = 'no';

ft_sourceplot(cfg, this_mne);
view(90, 0)

c = colorbar;
c.Label.String = 'Current (nAm)';
mne_plot = gcf;
set(mne_plot, 'units', 'normalized', 'outerposition', [0 0 1 1]);
title(['N100 localization (' num2str(1e3 * cfg.latency) ' ms)']);
```

Code Snippet 14 Code for plotting the N100 on the cortical surface

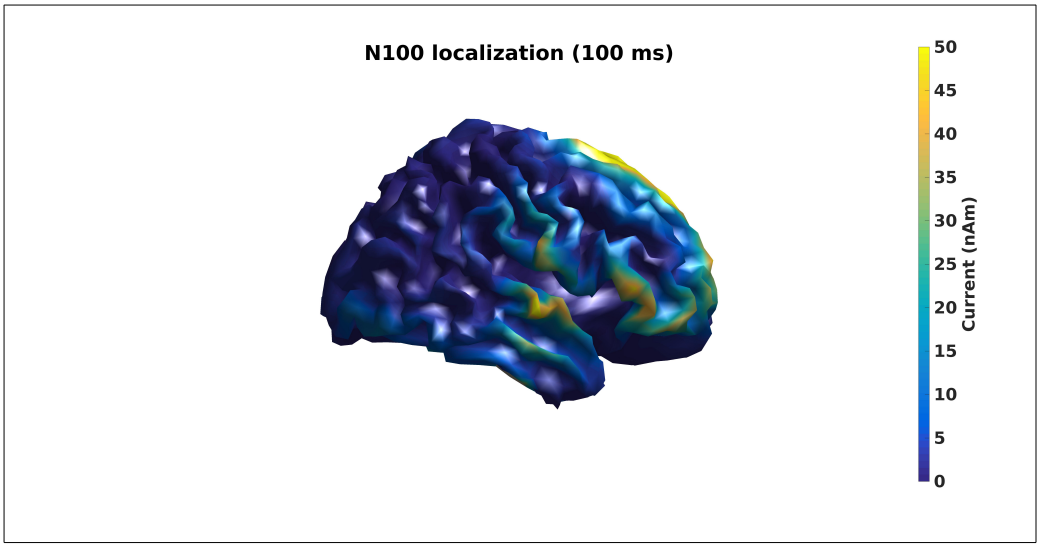


Fig. 4 Localization of the N100 response on the cortical surface using MNE

2.1.13. Alternative localization using dipole fits – lead field and source model

Another way to do the source localization is to assume a small number of sources and estimate their positions, orientations and amplitudes. In the MNE solution above, we on the contrary assumed a lot of sources and assumed that they were located on the cortical surface. The way we will be doing this is that we will use a so-called volumetric grid of sources (Code Snippet 15) overlain on the brain compartment of the headmodel. For all the sources in the grid that are inside the brain compartment, we will estimate a leadfield. Try plotting the source model yourself using *ft_plot_mesh* and *ft_plot_headmodel*.

```
cfg = [];
cfg.elec = elec_realigned;
cfg.headmodel = headmodel;
cfg.resolution = 0.01;
cfg.sourcemodel.unit = 'm';
cfg.channel = cleaned_data.label(1:57);

sourcemodel = ft_prepare_leadfield(cfg);
```

Code Snippet 15 Creating a lead field and a source model in preparation for dipole fitting. The channels are the ones defined in *cfg.elec*, the sources are 0.01 m (10 mm) apart (*cfg.resolution* and *cfg.sourcemodel.unit*), and the active electrodes are found in *cfg.channel*.

2.1.14. Alternative localization using dipole fits – dipole fit

We are going to find the two dipoles, symmetric over the *x*-axis, that optimally explain the scalp distributions for the electric potential in the time range 110-130 ms. We are going to assume fixed positions of the two dipoles during this time range. We are going to use a grid search followed by non-linear optimization to do the fit (Code Snippet 16). During the grid search, the optimal position to begin the non-linear optimization from is found. Thereafter, the optimal position, orientation and

amplitude of the dipoles are non-linearly optimized. This is done by reducing the difference by the actually measured scalp distribution and the scalp distribution that the fitted dipoles would produce.

```
cfg = [];
cfg.latency = [0.110 0.130];
cfg.sourcemodel = sourcemodel;
cfg.headmodel = headmodel;
cfg.model = 'regional';
cfg.nonlinear = 'yes';
cfg.gridsearch = 'yes';
cfg.numdipoles = 2;
cfg.symmetry = 'x';
cfg.elec = elec_realigned;

dipole_fit = ft_dipolefitting(cfg, ERPs{5});
```

Code Snippet 16 Doing the dipole fit on the time range from 110-130 ms (*cfg.latency*) using the newly created source model and earlier created head model. The dipoles are modelled as being in one position during the whole time range (*cfg.model*). Two dipoles (*cfg.numdipoles*) are fitted, and we require them to be symmetric across the *x*-axis (*cfg.symmetry*).

2.1.15. Alternative localization using dipole fits – plot dipole fit

We can now plot the dipoles (Code Snippet 17) on slices of the template brain using *ft_plot_dipole* and *ft_plot_slice* (Fig. 5).

```
load('standard_mri.mat')
mri = ft_convert_units(mri, 'm');
dipole_mri_plot = figure;
hold on
ft_plot_dipole(dipole_fit.dip.pos(1, :), ...
                mean(dipole_fit.dip.mom(1:3, :), 2), 'unit', 'm');
ft_plot_dipole(dipole_fit.dip.pos(2, :), ...
                mean(dipole_fit.dip.mom(4:6, :), 2), 'unit', 'm');

pos = mean(dipole_fit.dip.pos, 1);

orientations = {[1 0 0] [0 1 0] [0 0 1]};
n_orientations = length(orientations);

for orientation_index = 1:n_orientations

    orientation = orientations{orientation_index};

    ft_plot_slice(mri.anatomy, 'transform', mri.transform, 'location', pos, ...
                  'orientation', orientation);
end

axis tight
axis off
```

Code Snippet 17 Plotting the dipoles on three orthogonal slices of the template brain.

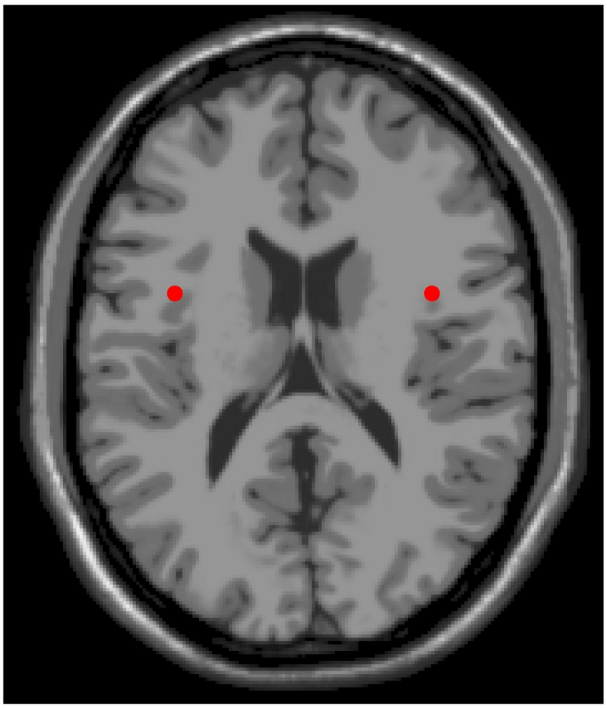


Fig. 5 Position of dipoles – use MATLAB tools to rotate the plot to see depth and orientation of the dipoles from different angles

2.1.16. Alternative localization using dipole fits – extract time courses

Now that we have the positions of our dipoles, we can estimate their time courses, i.e. how they develop over the whole epoch (-300-700 ms) (Code Snippet 18). We fit the dipoles again this time without grid search since we will use the already fitted positions as the positions of the dipoles (*cfg.dip.pos*). We will thus only fit the orientations and the amplitudes of the dipoles.

```
cfg = [];
cfg.latency = 'all';
cfg.sourcemodel = sourcemodel;
cfg.headmodel = headmodel;
cfg.model = 'regional';
cfg.nonlinear = 'yes';
cfg.gridsearch = 'no';
cfg.numdipoles = 2;
cfg.symmetry = 'x';
cfg.elec = elec_realigned;
cfg.dip.pos = dipole_fit.dip.pos;

dipole_time_courses = ft_dipolefitting(cfg, ERPs{5});
```

Code Snippet 18 We are estimating the time course for the whole epoch (*cfg.latency*) using known positions (*cfg.dip.pos*)

2.1.17. Alternative localization using dipole fits – plotting extracted time courses

Finally, we will plot the extracted time courses of the two dipoles (Code Snippet 19). Note that we divide the amplitudes with 1e6 to get everything in SI units. (The voltages were in µV). We see that the right hemisphere dipole shows a stronger activation at around 100 ms, whereas the left one shows a later peak around 200 ms (Fig. 6).

```
dipole_tc_plot = figure('units', 'normalized', 'outerposition', [0 0 1 1]);

n_times = length(dipole_time_courses.time);
amplitudes = zeros(2, n_times);

for time_index = 1:n_times
    amplitudes(1, time_index) = ...
                            norm(dipole_time_courses.dip.mom(1:3, time_index));
    amplitudes(2, time_index) = ...
                            norm(dipole_time_courses.dip.mom(4:6, time_index));
end

amplitudes = amplitudes / 1e6; %% Put it in SI units

plot(dipole_time_courses.time, amplitudes);
xlim([dipole_time_courses.time(1), dipole_time_courses.time(end)]);
xlabel('Time (s)')
ylabel('Amplitude (Am)')
title('Dipole time courses')
legend({'Right Hemisphere', 'Left Hemisphere'})
```

Code Snippet 19 Plotting the dipole time courses by getting the amplitudes (taking the norm of the moments for each time point)

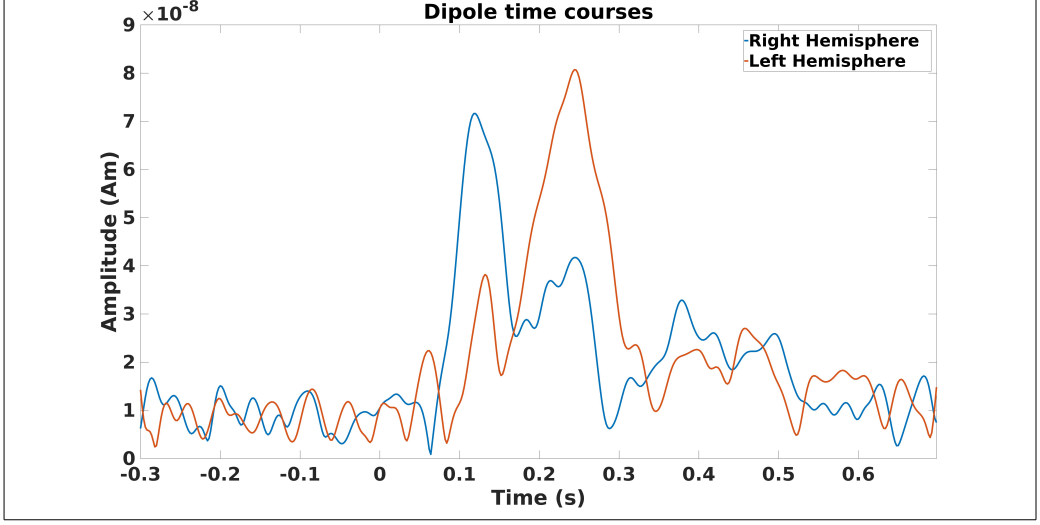


Fig. 6 Extracted time courses for the left and right hemisphere dipoles.

2.2. Summary of analysis

We have gone from raw EEG data through preprocessing and averaging to obtain ERPs. Then using template anatomical data, we source reconstructed the signal from the ERPs using a distributed solution (MNE) and a sparse solution (dipole fitting). Both methods localized an auditory response over the auditory cortex after around 100 ms.

2.3. Single subject MEG analysis – MNE-Python

In the following example, I will showcase a MEG analysis of responses related to reading words either in a well-formed sentence or in a scrambled sentence (Lam et al. 2016; Schoffelen et al. 2019). The dataset can be got from http://hdl.handle.net/11633/di.dccn.DSC_3011020.09_236. The subject *sub-V1002* is used.

MNE-Python will be used. It is recommended to follow the installation instructions on: https://martinos.org/mne/stable/install_mne_python.html. In the present analysis, I have used the Integrated Development Environment (IDE) *Spyder*. This can be gotten from the Anaconda distribution as advocated in the installation link. The relevant scripts are called *single_subject_analysis.py* and *single_subject_analysis.sh* and are found in the *language_MEG* folder of the GitHub repository, https://github.com/ualsbombe/language_electrified.

2.3.1. Short description of dataset

Two conditions were used. Either subjects were presented visually with a *sentence* or a *scrambled* version of the words in that sentence. All stimuli were presented at the centre of the screen within a visual angle of 4 ° inside a magnetically shielded room. Each trial began with a fixation cross presented for a jittered duration between 1,200 and 2,200 ms. Then each word, one at a time, were presented with a blank screen of 300 ms in between them. 10% of the 120 trials were followed by a yes/no question to make sure that subjects were paying attention

2.3.2. MEG recording

MEG data were collected with a 275 axial gradiometer system (CTF). It was sampled at 1,200 Hz and online lowpass-filtered at 300 Hz. Horizontal and vertical EOG and electrocardiogram (ECG) were measured.

2.3.3. Set paths and import

First, we import the needed packages, *mne*, *matplotlib* and *mayavi*. The latter two allows for plotting what is processed using *mne*.. The function *join* is also imported from *os.path*, which allows for creating full paths easily (akin to MATLAB's *fullfile*) (Code Snippet 20).

```
#%% ========================================================================
# SET PATHS AND IMPORTS
# ========================================================================

from os.path import join
import mne
import matplotlib.pyplot as plt
import matplotlib as mpl
from mayavi import mlab


path = '/home/lau/analyses/fieldtrip_datasets/language_MEG/'
data_path = join(path, 'data')
figures_path = join(path, 'figures')
subjects_dir = join(data_path, 'freesurfer')

subject = 'sub-V1002'
subject_path = join(data_path, subject, 'meg', subject + '_task-visual_meg.ds')
bem_folder = join(subjects_dir, subject, 'bem')
label_folder = join(subjects_dir, subject, 'label')
save_path = join(data_path, subject, 'meg', 'derived')
```

Code Snippet 20 Setting up the prerequisites for the MNE-Python analysis

2.3.4. Find events in the raw data

First, we read in the metadata from the raw files (Code Snippet 21). If we wanted to load the data, *preload* should have been set to *True*. This way, data is only accessed when needed. Subsequently, I find the events based on the trigger channels in the data. Subsequently, I shift the event forward 36 ms, since I know that there is a constant delay between the trigger and the actual presentation of the word on screen. Finally, I define a dictionary that contains the four events that I will be interested in. Sentences with or without a relative clause and their scrambled counterparts.

```
#%% ========================================================================
# FIND EVENTS IN RAW DATA
# ========================================================================

raw = mne.io.read_raw_ctf(subject_path, preload=False)

events = mne.find_events(raw)
## there was a known delay between trigger and stimulation of 36 ms
shifted_events = mne.event.shift_time_events(events, [2, 4, 6, 8], 0.036,
                                             raw.info['sfreq'])
event_id = dict(rc_plus_sentence=2, rc_plus_scrambled=4,
                rc_minus_sentence=6, rc_minus_scrambled=8)

events = mne.write_events(join(save_path, 'reading-eve.fif'),
                          shifted_events)
```

Code Snippet 21 Finding the events, time-shifting them and defining the ones of interest

2.3.5. Epoch the data

Subsequent, I epoch the raw data by choosing the time (*tmin* and *tmax*) around the events and choosing the period (*baseline*) to demean the data with (Code Snippet 22). Subsequently, we discard the information about relative clauses and collapse the categories in to well-formed sentences and scrambled sentences.

```
#%% ========================================================================
# EPOCH THE RAW DATA
# ========================================================================

tmin = -0.150
tmax =  0.500
baseline = (None, 0)
picks = mne.pick_types(raw.info, meg=True)

epochs = mne.Epochs(raw, shifted_events, event_id, tmin, tmax, baseline, picks)

mne.epochs.combine_event_ids(epochs,
                             ['rc_minus_sentence', 'rc_plus_sentence'],
                             dict(sentence=16), copy=False)
mne.epochs.combine_event_ids(epochs,
                             ['rc_minus_scrambled', 'rc_plus_scrambled'],
                             dict(scrambled=32), copy=False)

epochs.save(join(save_path, 'reading-epo.fif'))
```

Code Snippet 22 Epoching the data with baselining for the data going into an evoked analysis

2.3.6. Creating evoked responses

Next, I will create the averages of the data epochs for each of the two categories (*sentence* and *scrambled*) (Code Snippet 23).

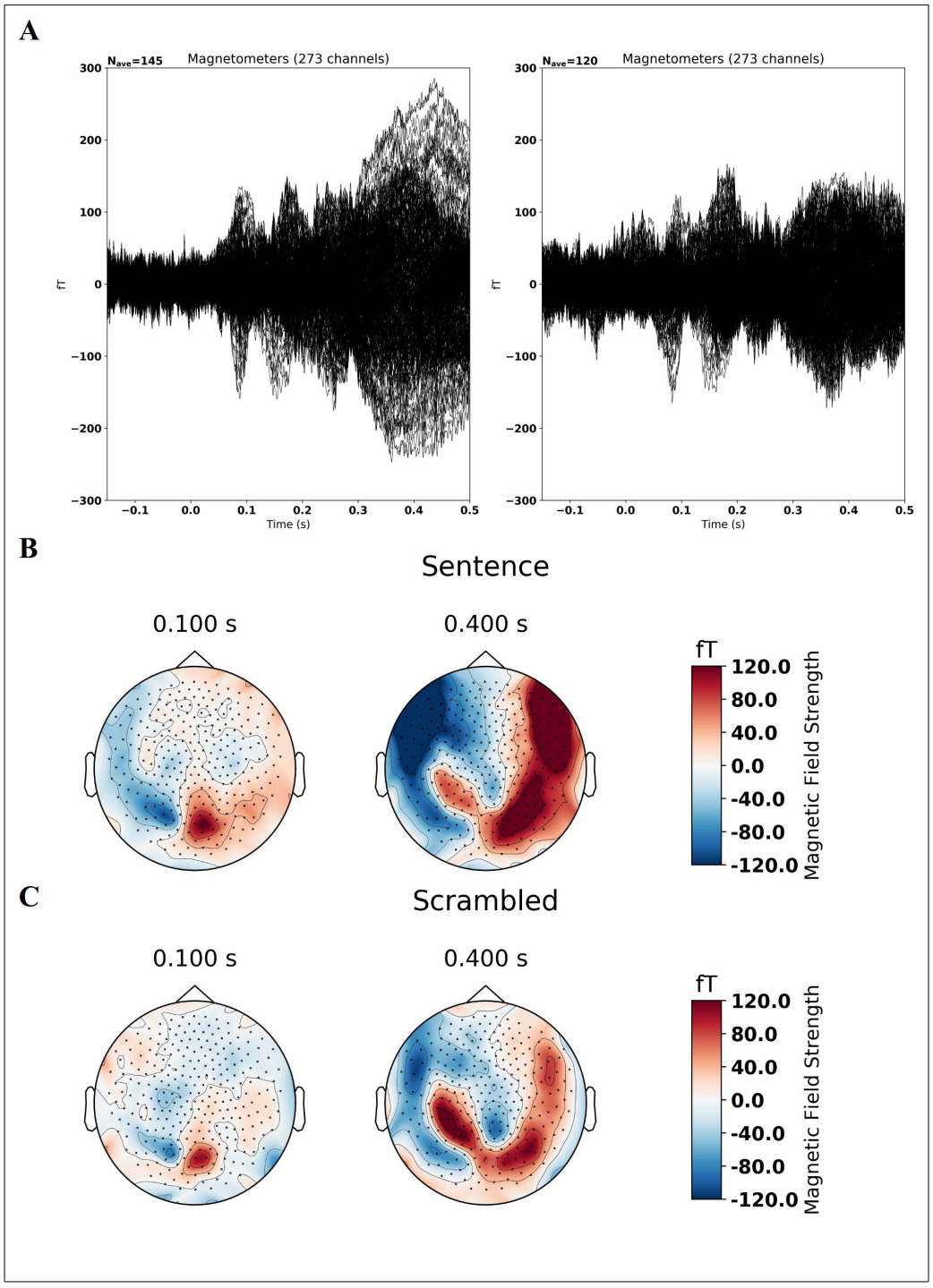
```
#%% ========================================================================
# EVOKEDS
# ========================================================================

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

mne.write_evokeds(join(save_path, 'reading-ave.fif'), evokeds)
```

Code Snippet 23 Looping through the events and averaging the epochs belonging to that event

```
In [16]: evokeds
Out[16]:
[<Evoked  |  'sentence' (mean, N=145), [-0.15, 0.5] sec, 301 ch, ~2.6 MB>,
 <Evoked  |  'scrambled' (mean, N=120), [-0.15, 0.5] sec, 301 ch, ~2.6 MB>]
```

Output 3 The two evokeds created with some information about them.

294 **2.3.7. Plotting evoked responses**



295

Fig. 7 Evoked responses for scrambled and well-formed sentences. Well-formed sentences show a stronger response at 400 ms **A:** Butterfly plots of axial gradiometers (note that the automatically generate plot title says magnetometers). **B&C:** Topographical plots of the responses.

In the next step, I am going to create a source model estimating the source distribution for each time point. This source reconstruction will be based on the individual subject's anatomy as imaged with magnetic resonance imaging (MRI). To do this, several steps of preprocessing the anatomical image is required. MNE-Python functions are tightly linked with those of FreeSurfer (http://www.freesurfer.net/). FreeSurfer functions are called from the Bash command language (https://www.gnu.org/software/bash/). This can easily be accessed on most Linux operating systems and on Apple's macOS. It is not part of the standard applications for Windows at the moment. We need FreeSurfer to run a segmentation of the brain.

2.4. MRI preprocessing

The two steps for FreeSurfer that are required are quite simple in terms of code. First, I am going to using the function *recon-all* to import the data and create the default folder structure of FreeSurfer. Secondly, I am going to run a full segmentation of the brain based on the image.

2.4.1. Importing the MRI

First, I set the (environment) variables, *SUBJECTS_DIR* and *SUBJECT*. Make sure that *SUBJECTS_DIR* is an empty folder. Indicate on which path the MRI file exists (*nii_path*). Indicate the file (*filename*) that should be imported. Finally, *recon-all* is called for importing the file. *openmp* indicates the number of processors you want to allocate to this process (Code Snippet 24).

```
#! /bin/bash

## export variables
export SUBJECTS_DIR=/home/lau/analyses/fieldtrip_datasets/language_MEG/data/freesurfer
export SUBJECT=sub-V1002

nii_path=/home/lau/analyses/fieldtrip_datasets/language_MEG/data/$SUBJECT/anat
cd $nii_path
filename=${SUBJECT}_T1w.nii
recon-all -subjid $SUBJECT -i $filename -openmp 32
```

Code Snippet 24 Bash code for importing the MRI into the FreeSurfer preferred format. Make sure that *SUBJECTS_DIR* and *nii_path* reflect your current paths.

This results in a folder named after *SUBJECT* being added to *SUBJECTS_DIR*. Inside this folder in *mri/orig*, you should now find the file *001.mgz*, which should be opened and inspected using tools such as *freeview* (comes with the FreeSurfer installation). Most of the folders created are empty for now, but will be filled during the next step.

2.4.2. Segmenting the brain

The next step is to segment the brain, again using the FreeSurfer *recon-all* function. This time I add the flag *-all* to indicate that I want to run the full algorithm. Be aware that this is a lengthy process, which can be sped up using as many processors you have available (Code Snippet 25).

```
#! /bin/bash

export SUBJECTS_DIR=/home/lau/analyses/fieldtrip_datasets/language_MEG/data/freesurfer
export SUBJECT=sub-V1002

recon-all -subjid $SUBJECT -all -openmp 32
```

Code Snippet 25 Bash code for doing the segmentation (or reconstruction) of the brain based on the imported filed (*001.mgz*)

2.4.3. Creating the Boundary Element Method (BEM) model

The next step is to create a BEM, which I will carry out in four steps (Code Snippet 26).

First, I apply the watershed algorithm, which creates four boundaries or surfaces on the segmented brain, the brain surface, the inner skull surface, the outer skull surface and the outer skin surface. This can be called from MNE-Python using *mne.bem.make_watershed_bem*. This calls a FreeSurfer function called *mri_watershed*. These surfaces which are put in the *bem* folder of *SUBJECT* can be inspected using *freeview*. (This command may not work for you in the Spyder IDE).

Second, I create a source space based on the surface of the white matter (*lh.white* and *rh.white*) in the *surf* folder of *SUBJECT*. Everything "above" this is assumed to be cortical areas where currents giving rise MEG fields can be generated. *spacing* indicates how one wants the sources distributed on the cortical surface. *oct6* results in ~4000 sources for each hemisphere (Fig. 8). Note that this restricts the solution to the cortical surface, e.g. we are not going source reconstruct, say, thalamic activity.

Third, I create the BEM model using *mne.bem.make_bem_model*. This uses the surfaces created with *mne.bem.make_watershed_bem*. For MEG, we rely only on *inner_skull.surf*, which MNE-Python expects to find in the *bem* folder of the *SUBJECT*. For EEG, we would have had to model three surfaces, the inner skull, the outer skull and the outer skin, since the different conductivities of these surfaces smear the electric potential. The magnetic field is not smeared, if we assume that the head is a sphere (or sphere-like) (Hämäläinen et al. 1993). Using just one surface is implicitly called by only assigning one element to the list *conductivity* among the arguments passed to *mne.bem.make_bem_model*.

Fourth, it is time to create the BEM solution using *mne.bem.make_bem_solution*. This calculates how currents generated in the source space spread throughout the volume conductor, i.e. the BEM model. The currents generated in the source space are what eventually gives rise to the measurable magnetic field outside the head.

```
#%% ========================================================================
# MR PREPROCESSING
# ========================================================================

## watershed

mne.bem.make_watershed_bem(subject, subjects_dir)

## source space

spacing = 'oct6'
filename = subject + '-' + spacing[:3] + '-' + spacing[-1] + '-src.fif'
fullpath = join(bem_folder, filename)

src = mne.source_space.setup_source_space(subject, spacing,
                                          subjects_dir=subjects_dir,
                                          n_jobs=-1, surface='white')
mne.source_space.write_source_spaces(fullpath, src)

## bem surfaces
ico = 4
if ico == 3:
    ico_string = '1280'
elif ico == 4:
    ico_string = '5120'
elif ico == 5:
    ico_string = '20484'
filename = subject + '-' + ico_string + '-bem.fif'
fullpath = join(bem_folder, filename)

surfaces = mne.bem.make_bem_model(subject, ico, conductivity=[0.3],
                                  subjects_dir=subjects_dir)
mne.bem.write_bem_surfaces(fullpath, surfaces)

## bem solution
filename = subject + '-' + ico_string + '-bem-sol.fif'
fullpath = join(bem_folder, filename)

bem = mne.bem.make_bem_solution(surfaces)
mne.bem.write_bem_solution(fullpath, bem)
```

Code Snippet 26 MR-preprocessing – running the *watershed* algorithm that separates the tissue boundaries from one another (brain, skull and skin), followed by creating a source space with equidistant sources on the cortical surface. Then a BEM model is made based on the brain surface. Finally, the BEM solution is made.

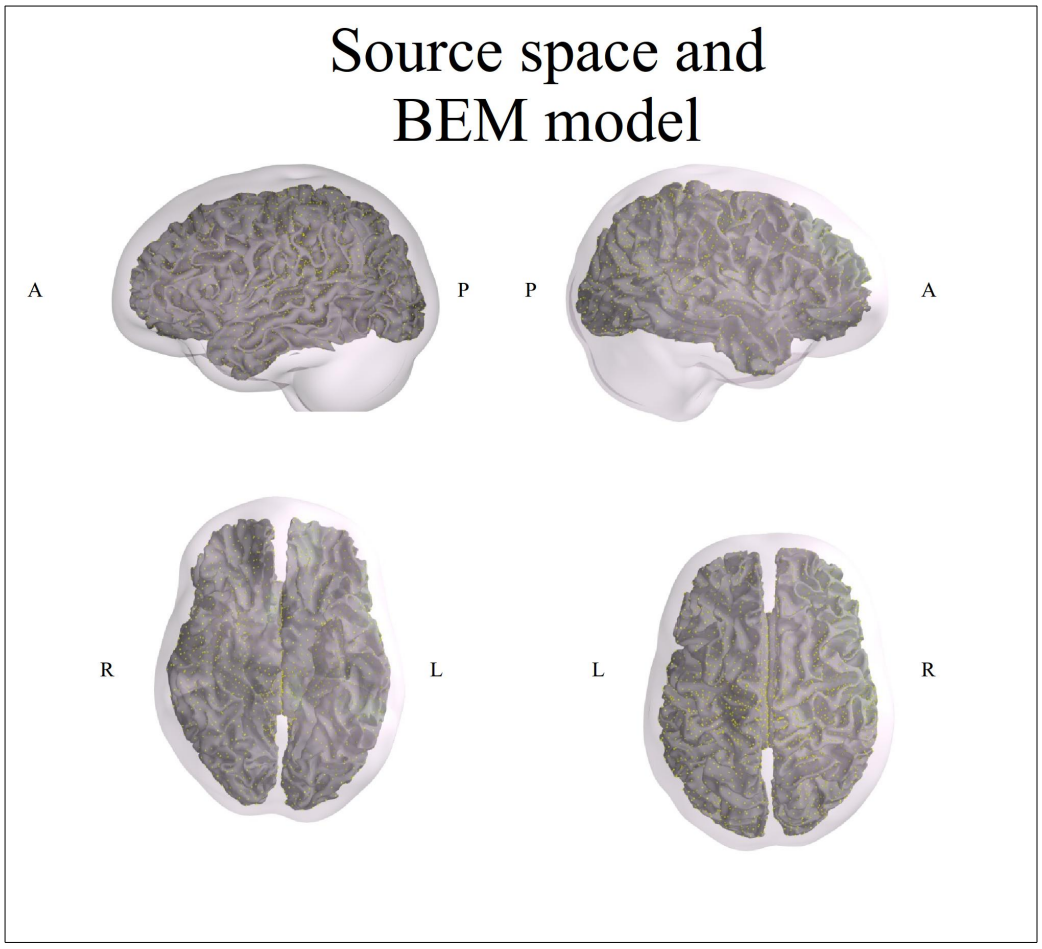


Fig. 8 The generated source space, the cortical surface, and BEM model (inner skull surface) plotted by calling *mne.SourceSpaces.plot*. The yellow dots indicate point like sources equidistantly spaced. A=anterior, P=posterior, R=right, L=left.

2.4.4. The forward solution

Finally, it is time to put all these things together to create the forward solution (Code Snippet 27), which is a solution to the questions: "how do sources in the source space connect to the sensors in the MEG sensor array?". We can get the sensor positions from the metadata of the recording file, and we already have the position of sources in the source space (Fig. 8). These are expressed in different coordinate systems, so the next step is to align these two coordinate systems. This can be done using the GUI accessed from *mne.gui.coregistration* (Fig. 9). Call *mne.gui.coregistration* with the argument *subjects_dir* set to the library where your FreeSurfer reconstruction is found (Code Snippets 24-25). The importance of doing co-registration is shown in Fig. 10. With a wrong co-registration, the forward model will be wrong, resulting in the source reconstruction ending up wrong as well.

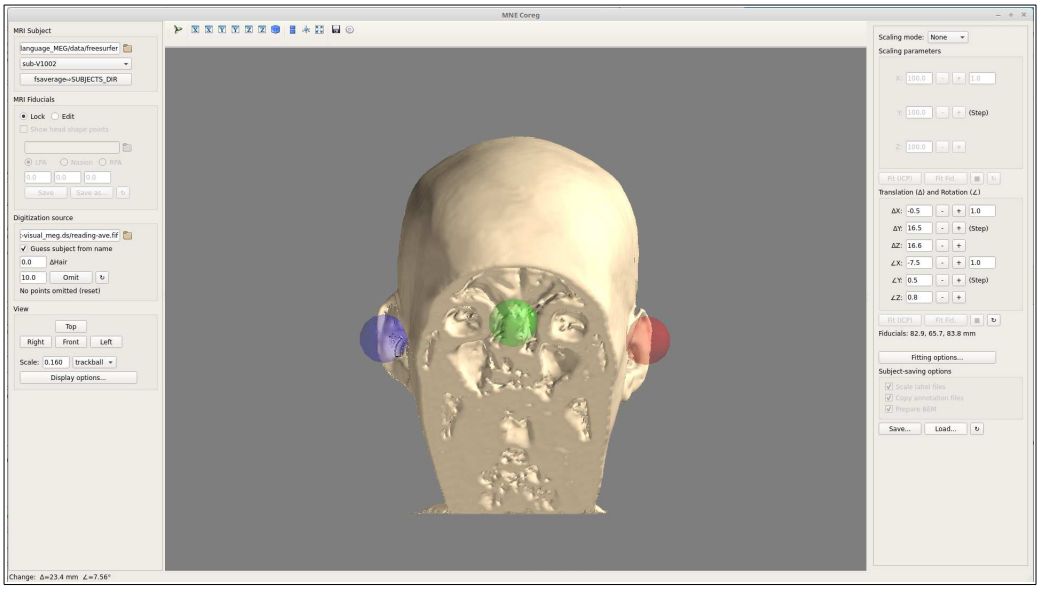


Fig. 9 The GUI for co-registration. The blue, red and green dots are the right pre-auricular point, the left pre-auricular point and the nasion respectively. Normally, the head would not be defaced allowing for a more precise placement of the nasion. In the case of having acquired extra digitization points with a Polhemus Fastrak, it would be possible to use these to optimise the co-registration further by for example using the Iterative Closest Point algorithm (right hand side)

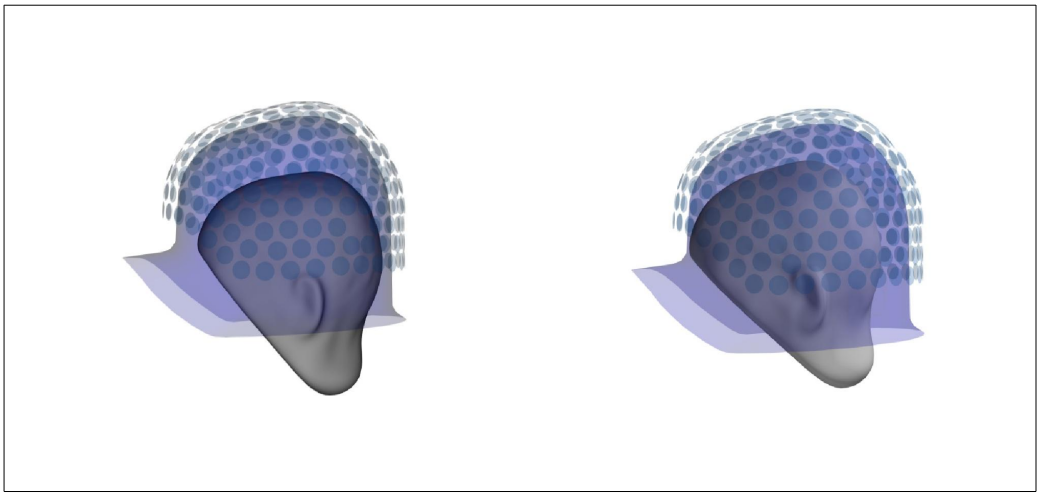


Fig. 10 The importance of doing correct alignment. **Left:** head in CTF-coordinates and helmet in Neuromag-coordinates. **Right:** Both in Neuromag-coordinates .

```
#%% ========================================================================
# FORWARD SOLUTION
# ========================================================================

info = raw.info
trans = join(bem_folder, subject + '-trans.fif')
src = join(bem_folder, subject + '-' + spacing[:3] + '-' + spacing[-1] + \
           '-src.fif')
bem = join(bem_folder, subject + '-' + ico_string + '-bem-sol.fif')
meg = True
eeg = False

filename = 'reading-fwd.fif'
fullpath = join(bem_folder, filename)

fwd = mne.make_forward_solution(info, trans, src, bem, meg, eeg,
                                n_jobs=-1)
mne.write_forward_solution(fullpath, fwd)
```

Code Snippet 27 Creating the forward solution. This requires information about the positions of the MEG sensors, gotten from *raw.info*, the positions of sources, gotten from *src*, and how the currents generated at those sources spread, gotten from *bem*. Importantly, since the positions of the MEG sensors and the source and BEM models are defined in two different coordinate systems, an MEG and an MR coordinate system, these two need to be co-registered. The transformation matrix aligning the two coordinate systems is in *trans*.

```
In [20]: fwd['sol']['data']
Out[20]:
array([[ 5.65681191e-07,  5.84685521e-08,  8.63432502e-08, ...,
        -5.16093991e-07,  1.87029982e-07,  4.52607658e-08],
       [ 6.46068781e-07,  7.96112630e-08,  1.78202741e-07, ...,
        -4.05042243e-07,  7.48822284e-08, -8.38535161e-09],
       [ 7.34370169e-07,  1.03006195e-07,  2.81791487e-07, ...,
        -3.20977364e-07, -5.78000981e-09, -4.68751046e-08],
       ...,
       [-1.35731593e-06, -4.67086085e-07, -1.69435707e-06, ...,
         4.68417713e-07,  2.90284221e-08,  5.15815494e-08],
       [-5.78553384e-06, -9.80045009e-07, -1.19382066e-06, ...,
         7.40245121e-07, -1.14981997e-07, -2.43388296e-08],
       [ 4.15148653e-06,  3.81912454e-07, -4.94396365e-07, ...,
         9.39094575e-08,  1.22600336e-07,  7.99115509e-08]])
```

Output 4 *fwd* contains the solution for each set of channels (273) and each source (*nsource* x 3 directions = 24588). The output of *fwd['sol']['data']* indicate the magnetic field (T) that a current of 1 A in a given source would induct in a sensor.

2.4.5. Inverse solution

The final step before doing the actual source reconstruction is to specify the inverse operator applied (Code Snippet 28).

```
#%% ========================================================================
# INVERSE OPERATOR
# ========================================================================

epochs = mne.read_epochs(join(save_path, 'reading-epo.fif'))

cov = mne.compute_covariance(epochs, tmax=0)
cov.save(join(save_path, 'reading-cov.fif'))

inv = mne.minimum_norm.make_inverse_operator(raw.info, fwd, cov)

mne.minimum_norm.write_inverse_operator(join(save_path, 'reading-inv.fif'),
                                        inv)
```

Code Snippet 28 Creating the inverse operator. The noise covariance is needed to supply an estimate of the noise in the recording

2.4.6. Estimate source time courses

Finally, we can estimate the time courses for each of the sources in our source space (n sources = 8196) (Code Snippet 29). That means that the data has the shape n sources x n samples (8196 x 781). For each sample, we can thus plot the source pattern (Fig. 11) using the code (Code Snippet 30).

```
#%% ========================================================================
# MNE
# ========================================================================

evokeds = mne.read_evokeds(join(save_path, 'reading-ave.fif'))
inv = mne.minimum_norm.read_inverse_operator(join(save_path,
                                                  'reading-inv.fif'))
stcs = dict()

for evoked in evokeds:
    stcs[evoked.comment] = mne.minimum_norm.apply_inverse(evoked, inv,
                                                          method='MNE')
```

Code Snippet 29 Estimating source time courses (*stcs*) using the *MNE* method. *dSPM* can also be used, which controls for the depth bias of *MNE (see script on GitHub)*

```
#%% ========================================================================
# PLOT MNE
# ========================================================================

mlab.close(None, True)

brain = mne.viz.plot_source_estimates(stcs['sentence'], subject, hemi='both',
                                      subjects_dir=subjects_dir,
                                      initial_time=0.100, background='w',
                                      foreground='k',
                                      clim=dict(kind='value',
                                                lims=[5e-11, 7e-11, 10e-11]),
                                      colorbar=True)
brain.show_view('caudal')
brain.save_image(join(figures_path, 'MNE_V1_sentence.jpeg'))

mlab.view(90, 135)
brain.set_time(0.400)
brain.save_image(join(figures_path, 'MNE_LOT_sentence.jpeg'))
```

Code Snippet 30 Plotting the MNE solution and saving some images.

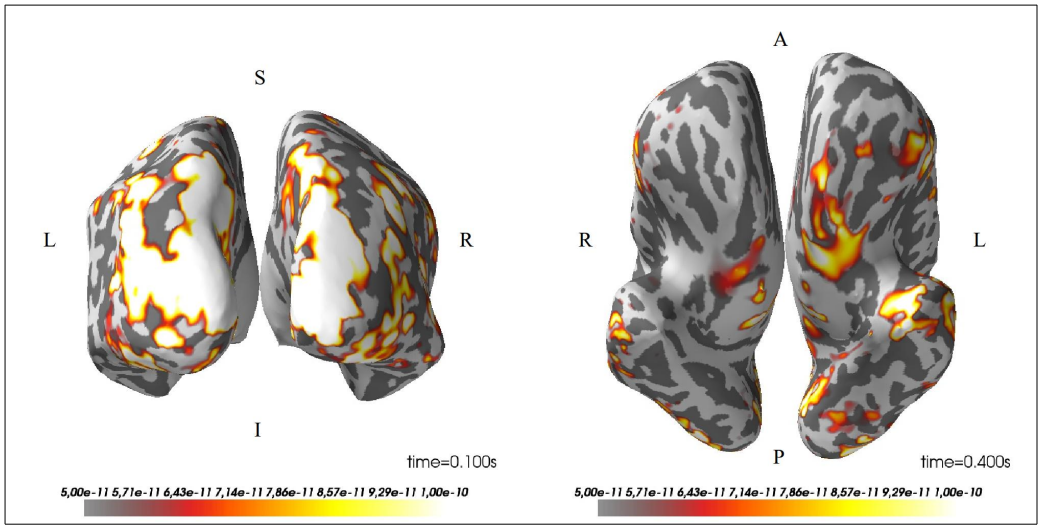


Fig. 11 Minimum Norm Estimate of the sentence activity. **Left:** activity in the visual cortex at 100 ms. **Right:** activity in frontal areas at 400 ms. The values on the colour bar are in Am.

We can also plot the time courses of all sources within an anatomically defined area. Here, we will extract time courses from V1 and the lateral orbitofrontal cortex (Code Snippet 31) and subsequently plot them (Code Snippet 32; Fig. 12).

```
#%% ========================================================================
# LABELS
# ========================================================================

hemis = ['lh', 'rh']
labels = dict()
ltcs = dict()

for hemi in hemis:
    labels[hemi] = dict()
    labels[hemi]['V1'] = mne.read_label(join(label_folder,
          hemi + '.V1.label'))
    labels[hemi]['LOF'] = mne.read_label(join(label_folder,
          hemi + '.lateralorbitofrontal.label'))

ltcs = dict()
for hemi in hemis:
    ltcs[hemi] = dict()
    for label_index, label in enumerate(labels[hemi]):
        ltcs[hemi][label] = dict()
        for event_index, event in enumerate(stcs):
            ltcs[hemi][label][event] = \
                              stcs[event].in_label(labels[hemi][label])
```

The relevant script is called *single_subject_analysis.m* and is found in the *language_MEG* folder of the GitHub repository.

Code Snippet 31 Extracting labels from the *stcs* and making label time courses (*ltcs*).

```
#%% ========================================================================
# PLOT LTCS
# ========================================================================

mpl.rcParams['axes.linewidth'] = 3
mpl.rcParams['lines.linewidth'] = 3
mpl.rcParams['figure.titlesize'] = 30
mpl.rcParams['axes.labelsize'] = 20
mpl.rcParams['axes.titlesize'] = 20

fig = plt.figure(figsize=(16, 12))
for hemi_index, hemi in enumerate(ltcs):
    for label_index, label in enumerate(ltcs[hemi]):
        plot_number = 2*label_index + hemi_index + 1
        plt.subplot(2, 2, plot_number)
        times = stcs['sentence'].times * 1e3
        data = ltcs[hemi][label]['sentence'].data
        data = mne.filter.filter_data(data, 1200.0, None, 45) * 1e9
        plt.plot(times, data.mean(axis=0))
        data = ltcs[hemi][label]['scrambled'].data
        data = mne.filter.filter_data(data, 1200.0, None, 45) * 1e9
        plt.plot(times, data.mean(axis=0))
        plt.xlim()
        plt.ylim((0, 0.1))
        if plot_number > 2:
            plt.xlabel('Time (ms)')
        plt.ylabel('Current (Am)')
        plt.vlines(0, 0, 0.1, linestyles='dashed')
        plt.title(hemi.upper() + ' ' + label)
        if plot_number == 1:
            plt.legend(['Sentence', 'Scrambled'])
plt.suptitle('Average activity over ROIs')
plt.savefig(join(figures_path, 'label_time_courses.jpeg'), dpi=300)
```

Code Snippet 32 Plotting the label time courses using the standard *matplotlib* functions.

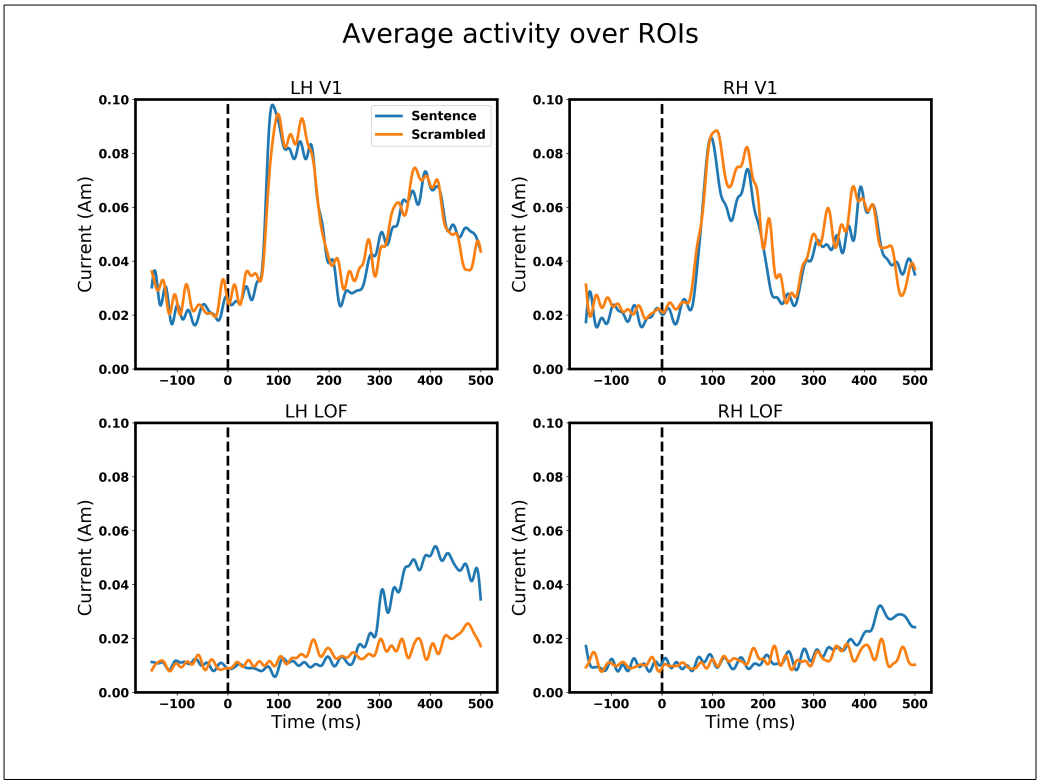


Fig. 12 Time courses for *Sentence* and *Scrambled* based on the mean of source activity of all sources within the two ROIs (V1 and lateral orbitofrontal cortex (LOF)), for each of the hemispheres (LH=left hemisphere, RH=right hemisphere)

2.5. Summary of MNE analysis

We found that the visual cortex is active for both sentences and scrambled word lists, but that the Lateral Orbitofrontal Cortex is responding more to sentences than to scrambled word lists.

2.6. Single subject analysis – Beamformer

I am now going to report a single subject analysis on the same dataset using FieldTrip and aiming to do a beamformer analysis of an oscillatory response. First step is setting up the paths again (Code Snippet 33). Some steps will be done in less detail compared to the single subject analysis for the EEG data since there is some repetition involved. We will use Dynamic Imaging of Coherent Sources (DICS) (Gross et al. 2001), since this allows for reconstruction of oscillatory signals. The relevant script is called *single_subject_analysis.m* and is found in the *language_MEG* folder of the GitHub repository, https://github.com/ualsbombe/language_electrified.

```
clear variables
restoredefaultpath

% change "home_dir", "data_dir", project_dir and "fieldtrip_dir" such that
% it applies to your setup.
% If you prefer having the "fieldtrip_dir" on your default path
% delete the lines about "fieldtrip_dir" and adding "fieldtrip_dir" to your
% path. Do remember to run "ft_defaults". This makes sure that the
% appropriate FieldTrip subdirectories are loaded
home_dir = '/home/lau/';
project_dir = fullfile(home_dir, 'analyses', 'fieldtrip_datasets', ...
                    'language_MEG');
data_dir = fullfile(project_dir, 'data');
figures_dir = fullfile(project_dir, 'figures');
fieldtrip_dir = fullfile(home_dir, 'matlab', 'fieldtrip');
derived_meg_dir = fullfile(data_dir, 'sub-V1002', 'meg', 'derived');
derived_mr_dir = fullfile(data_dir, 'sub-V1002', 'anat', 'derived');

addpath(fieldtrip_dir);
ft_defaults
```

Code Snippet 33 An example of how a script can be started, making sure that all variables are cleaned, and that MATLAB path used is the default path. Note that changes have to be made to fit the user's directory structure. Ideally, this should be the only change that should be made between operating systems. On macOS the "home_dir" is usually /Users/<your_name>" and on Windows it is C:\Users\<your_name>

2.6.1. Reading in and segmenting data

We read in the raw data and segment it into epochs of interest (Code Snippet 34). Following Lam et al. (2016), we are going to be interested in the oscillatory responses in the time window between -150 ms to 500 ms. To analyse oscillatory responses in this time window, it is necessary to include extra time to also capture oscillations that have slower cycles than the duration of the time window (650 ms). For example, it would not be possible to estimate the amplitude or power of a 1 Hz oscillation in this time window since a cycle lasts for 1,000 ms. In this example, I am using a time window from -550 ms to 1,000 ms (Code Snippet 35).

```
sentence  = {2 4};
scrambled = {6 8};

cfg = [];
cfg.dataset = fullfile(data_dir, 'sub-V1002', 'meg', ...
    'sub-V1002_task-visual_meg.ds');
cfg.trialdef.eventtype  = 'frontpanel trigger';
cfg.trialdef.eventvalue = [sentence scrambled];
cfg.trialdef.prestim    = 0.514; % s
cfg.trialdef.poststim   = 1.036; % s % allowing for adjsutment

cfg_def = ft_definetrial(cfg);
```

Code Snippet 34 Defining a trial structure using the raw dataset (*cfg.dataset*) and indicating what event type and what event values should be used. Note that I have allowed for the adjustment of 36 ms of the events due to the delay between trigger and actual stimulation.

```
cfg_def.continuous = 'yes'; %% the data was recorded continuously

data = ft_preprocessing(cfg_def);

cfg = [];
cfg.offset = -0.036 * data.fsample; %% push forward 36 ms in time

data = ft_redefinetrial(cfg, data);

save(fullfile(derived_meg_dir, 'preprocessed_data'),  '-struct', 'data');
```

Code Snippet 35 Reading in data using *ft_preprocessing* and subsequently adjusting the data by the number of samples that correspond to 36 ms (*cfg.offset*).

Note that we are not doing any preprocessing such as baseline correcting and filtering. These will come at later stages to make sure that this data can both be used for analysing event-related and oscillatory responses.

2.6.2. Creating boolean indices

In this analysis, I am defining the trial indices that belong to each type (*sentence* or *scrambled*) and put them in a cell array (Code Snippet 36). These can then subsequently be used to pick out trials of that particular type.

```
n_events = 2;
trials = cell(1, n_events);

for event_index = 1:n_events

    if event_index == 1
        trial_values = sentence;
    else
        trial_values = scrambled;
    end

    n_trials = length(data.trialinfo);
    trial_indices = zeros(1, n_trials);

    for trial_index = 1:n_trials
        if any(data.trialinfo(trial_index) == cell2mat(trial_values))
            trial_indices(trial_index) = 1;
        end
    end

    trials{event_index} = logical(trial_indices); %% make boolean and add
                                                  % to cell array

end
```

Code Snippet 36 This creates two boolean vectors that indicate whether or not a trial belongs to respectively *sentence* or *scrambled*.

2.6.3. Getting the event-related responses

Even though the main goal is to analysis oscillatory responses, it is always prudent to inspect the event-related responses to affirm that the expected responses are there. In this case, we should expect to see a visual response since subjects are watching visually presented words. In this step we are building a (sub-)structure *cfg.preproc*, which indicates which preprocessing options should be

applied before averaging using *ft_timelockanalysis* (Code Snippet 37). We are going to demean the data using a baseline window from -150 ms to 0 ms and lowpass filter the data at 40 Hz. We are also only going to consider the data in the time window from -150 ms to 500 ms as in the MNE-Python analysis above.

```
n_events = 2;
ERFs = cell(1, n_events);

for event_index = 1:n_events

    cfg = [];
    cfg.preproc.demean = 'yes'; %% demean the data
    cfg.preproc.baselinewindow = [-0.150 0]; %% use this time window to demean
    cfg.preproc.lpfilter = 'yes'; %% apply low-pass filter ...
    cfg.preproc.lpfreq = 40; %% at this frequency
    cfg.trials = trials{event_index}; %% use only these trials
    cfg.latency = [-0.150 0.500]; %% in only this time window
    cfg.covariance = 'yes';
    cfg.covariancewindow = cfg.preproc.baselinewindow;
    cfg.channel = 'MEG';

    ERFs{event_index} = ft_timelockanalysis(cfg, data);
end

save(fullfile(derived_meg_dir, 'ERFs'), 'ERFs');
```

Code Snippet 37 Computing the ERFs

2.6.4. Plotting the ERFs for a quick inspection

We are going to plot the ERFs at 100 ms (Code Snippet 38; Fig. 13).

```
cfg = [];
cfg.layout = 'CTF275.lay';
cfg.xlim = [0.100 0.100]; %% plot the visual response at 100 ms
cfg.zlim = [-120e-15 120e-15];
cfg.comment = ' ';

mplot = figure('units', 'normalized', 'outerposition', [0 0 1 1]);
subplot(1, 2, 1)
ft_topoplotER(cfg, ERFs{1});
title('Sentence (100 ms)')

subplot(1, 2, 2)
ft_topoplotER(cfg, ERFs{2});
title('Scrambled (100 ms)')
c = colorbar;
c.Label.String = 'Magnetic Field (T)';

print(mplot, fullfile(figures_dir, 'topoplot_fieldtrip.jpeg'), '-djpeg', ...
    '-r300')
```

Code Snippet 38 Plotting topographical plots of the response pattern at 100 ms.

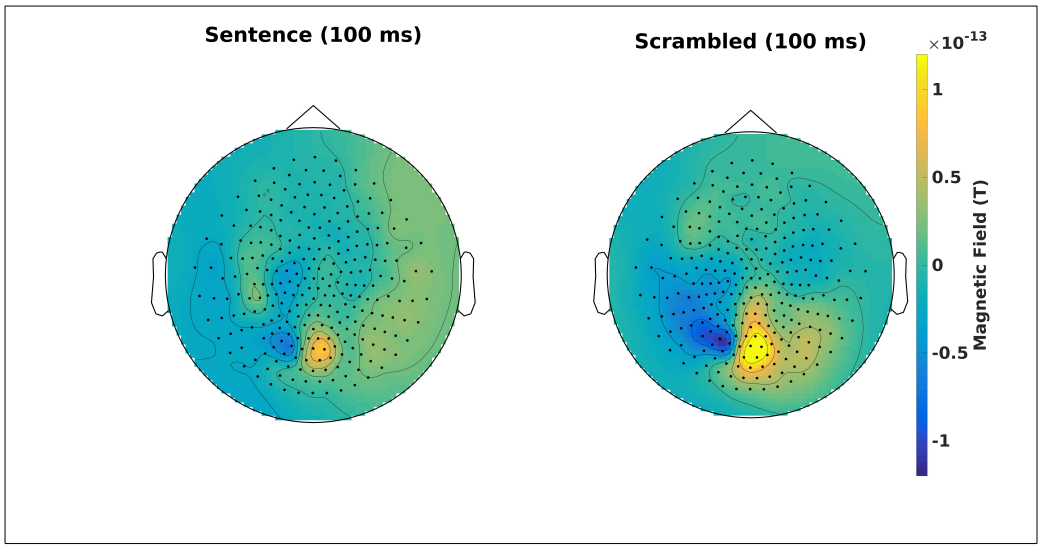


Fig. 13 Topographical plots showing the visual responses for *sentence* and *scrambled*.

These indicate that we get a visual response after 100 ms. Next step is to look at the time-frequency representations (TFR). First, we are going to look at a detailed representation in terms of time. This will be mostly for plotting. The TFRs that the beamformer source reconstructions will be based on will be more coarse. More will follow on that below.

2.6.5. Detailed time-frequency representation

The first step is to demean the data using the whole time period as the baseline. This is necessary to make sure that all channels have the same offset, the so-called Direct Current (DC) offset (Code Snippet 39).

```
cfg = [];
cfg.demean = 'yes';
cfg.baselinewindow = [-Inf Inf];

baselined_data = ft_preprocessing(cfg, data);
```

Code Snippet 39 Baselining the data is necessary to make sure that each channel has the same offset

For the next step we use a so-called Hanning taper to estimate the power at each time-frequency pair from 2-40 Hz (*cfg.foi*) and from -100 ms to 500 ms (*cfg.toi*). (Code Snippet 40) The steps are 2 Hz and 5 ms respectively. A sliding window with a width of 500 ms (*cfg.t_ftimwin*) was used. Note that this implies that the data are smoothed over both time and frequency meaning that time-frequency pairs nearby one another are highly correlated. An integer number of cycles should fit inside the sliding time window (500 ms). Given that

$$\frac{1}{500\,ms} = 2\,Hz$$

this means that we can estimate frequencies of multiples of 2 Hz, i.e. 2, 4, 6, 8, ... Hz.

```
n_events = 2;
detailed_tfrs = cell(1, n_events);

for event_index = 1:n_events

    cfg = [];
    cfg.method = 'mtmconvol'; %% run tfr
    cfg.taper = 'hanning'; %% use a hanning taper
    cfg.output = 'pow'; %% return power
    cfg.foi = 2:2:40; %% frequencies to centre on
    cfg.t_ftimwin = ones(1, length(cfg.foi)) * 0.500; %% use a sliding window
                                                      % of 400 ms
    cfg.toi = -0.100:(1/baselined_data.fsample):0.500; %% time points to
                                                       % centre on
    cfg.pad = 'nextpow2';
    cfg.channel = 'meg'; %% which channels to use
    cfg.trials = trials{event_index}; %% which condition, sentence or scrambled

    detailed_tfrs{event_index} = ft_freqanalysis(cfg, baselined_data);
end

save(fullfile(derived_meg_dir, 'detailed_tfrs'), 'detailed_tfrs');
```

Code Snippet 40 Creating a detailed TFR. It is detailed in the sense that a power estimate is made for each combination of frequency and time in steps of 2 Hz between 2 Hz and 40 Hz (*cfg.foi*) and in steps of 5 ms between -100 ms and 500 ms respectively using a sliding time window of 500 ms.

2.6.6. Plotting the time-frequency representation

Here, we plot a left temporal channel (*cfg.channel*) with or without baselining the images (Code Snippet 41 & Fig. 14).

```
n_events = 2;

cfg = [];
cfg.baseline = [-Inf 0]; %% baseline window
cfg.baselinetype = 'relative'; %% for each frequency demean based on the window
                               % above
cfg.layout = 'CTF275.lay';
cfg.zlim = [0.7 1.6];
cfg.channel = 'MLT15';
cfg.colorbartext = 'Power relative to baseline';
cfg.fontsize = 35; %% of title

names = {'Sentence' 'Scrambled'};
mplot = figure('units', 'normalized', 'outerposition', [0 0 1 1]);

for event_index = 1:n_events

    if event_index == 1
        cfg.colorbar = 'no';
    elseif event_index == 2
        cfg.colorbar = 'yes';
    end
    subplot(1, 2, event_index);
    cfg.title = names{event_index};
    ft_singleplotTFR(cfg, detailed_tfrs{event_index});
    xlabel('Time (s)')
    ylabel('Frequency (Hz)')

end

print(mplot, fullfile(figures_dir, 'detailed_tfr.jpeg'), '-djpeg', '-r300')
```

Code Snippet 41 Plotting the TFR using a *relative* baseline on the image. For producing Fig. 14A, just comment out *cfg.baseline*, *cfg.baselinetype* and *cfg.zlim* (*cfg.colorbartext* can also be changed).

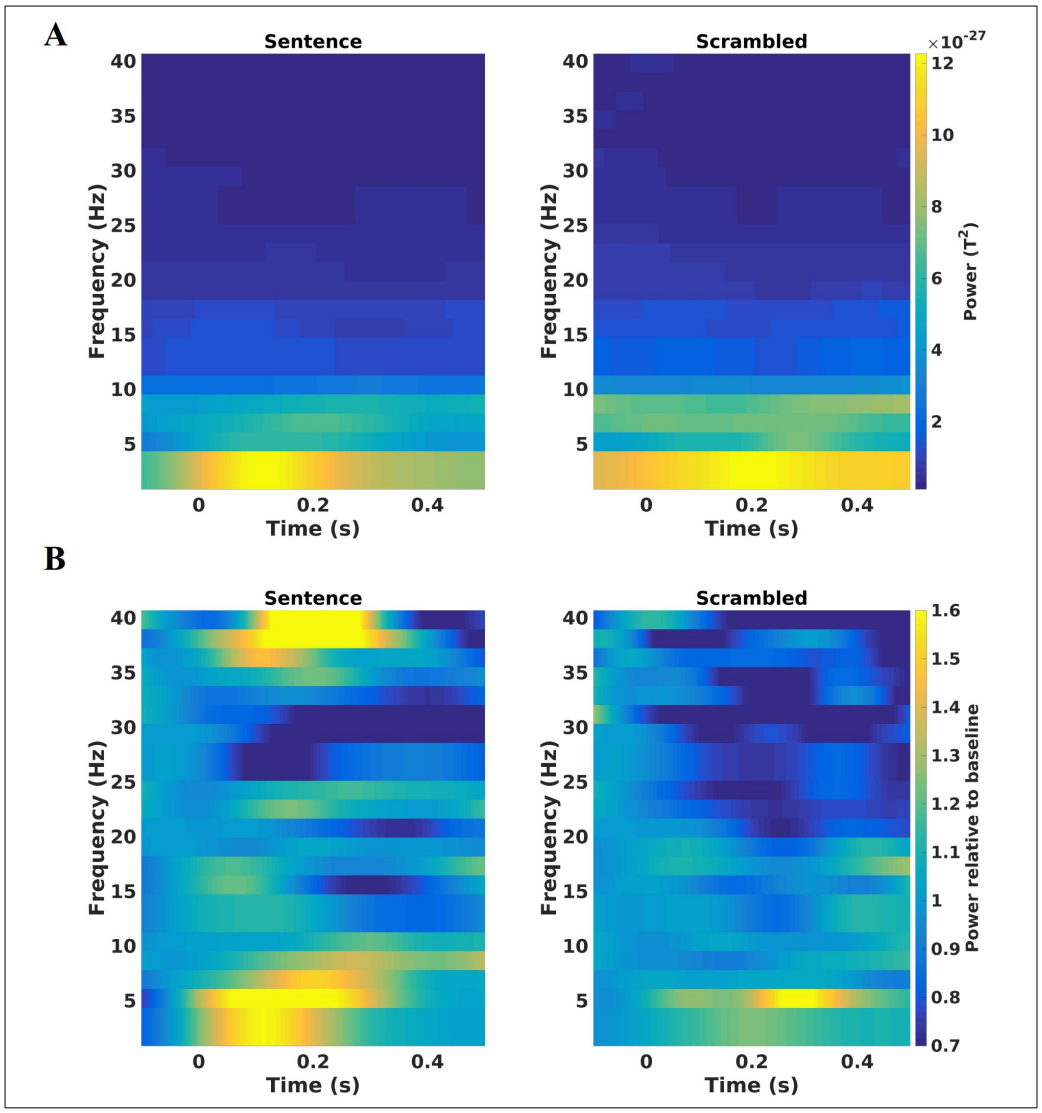


Fig. 14 TFR plots. **A:** Raw power for the different time-frequency pairs. **B:** Power relative to baseline (-100 ms to 0 ms). Note that raw power is always greater in the lower frequency bands as seen in **A**, so baselining can be done to make the changes in the different frequencies more comparable to the eye.

On this channel (Fig. 14B), we see a relative increase in the theta band (~3-7 Hz) and gamma band (>35 Hz) for the sentences that appears bigger than the corresponding ones in the scrambled condition. Use *ft_multiplotTFR* to get an overview of all channels at once. Having noticed this, we carry on to the MR preprocessing, which is needed before source reconstruction using a beamformer can be done.

2.7. MR preprocessing

In contrast to the EEG analysis before, we now have access to anatomical data, which has also been registered to the CTF coordinate system using *ft_volumerealign*. We reslice the data and segment the image into the brain, the skull and the scalp using *ft_volumesegment* (Code Snippet 42).

```
% read in
mri = ft_read_mri(fullfile(data_dir, 'sub-V1002', 'anat', ...
                           'sub-V1002_space-CTF_T1w.nii'));

mri.coordsys = 'ctf';

cfg = [];
cfg.resolution = 1;

mri_resliced = ft_volumereslice(cfg, mri);

cfg = [];
cfg.output = {'brain' 'skull' 'scalp'};

segmented_mri = ft_volumesegment(cfg, mri);

save(fullfile(derived_mr_dir, 'mri_resliced'), 'mri_resliced');
save(fullfile(derived_mr_dir, 'segmented_mri'), 'segmented_mri')
```

Code Snippet 42 Reading in the co-registered anatomical data. Reslicing it using *ft_volumereslice* is just to make sure that plots using *ft_sourceplot* look nice. Finally, the image is segmented into brain, skull and scalp.

2.7.1. Constructing a head model

Next step is to create a head model, which indicates how the currents that ultimately give rise to the measured magnetic field flow through the head (Code Snippet 43). We only use the brain mesh for the head model (see Section 2.4.3. during the MNE-Python analysis above)

```
cfg = [];
cfg.method = 'projectmesh';
cfg.tissue = 'brain';
cfg.numvertices = 3000;

mesh_brain = ft_prepare_mesh(cfg, segmented_mri);

cfg.tissue = 'scalp';
cfg.numvertices = 1000;

mesh_scalp = ft_prepare_mesh(cfg, segmented_mri);

% construct headmodel

cfg = [];
cfg.method = 'singleshell';

headmodel = ft_prepare_headmodel(cfg, mesh_brain);
headmodel = ft_convert_units(headmodel, 'm');

save(fullfile(derived_mr_dir, 'headmodel'), 'headmodel')
```

Code Snippet 43 First, we create a mesh of the brain and scalp segmentations. For the head model, only the brain compartment is used. (The scalp mesh will be used in the next step).

2.7.2. Inspect the match between MR and MEG sensors

One should always make a quality check about whether the co-registration process has been done properly (Code Snippet 44; Fig. 15).

```
sens = ft_read_sens(fullfile(data_dir, 'sub-V1002', 'meg', ...
    'sub-V1002_task-visual_meg.ds'), 'senstype', 'meg');
headshape = ft_read_headshape(fullfile(data_dir, 'sub-V1002', 'meg', ...
    'sub-V1002_headshape.pos'));
mplot = figure('units', 'normalized', 'outerposition', [0 0 1 1]);
hold on
ft_plot_sens(sens, 'unit', 'm');
ft_plot_headshape(headshape, 'unit', 'm', 'vertexcolor', 'red', ...
    'vertexsize', 50);
ft_plot_mesh(mesh_scalp, 'unit', 'm', 'facealpha', 0.8);
ft_plot_axes([], 'unit', 'm')

view(180, 0)
zoom(1.5)

print(mplot, fullfile(figures_dir, 'quality_check.jpeg'), '-djpeg', ...
    '-r300')
```

Code Snippet 44 Plotting the head shape, the scalp surface, the sensors and the axes all together

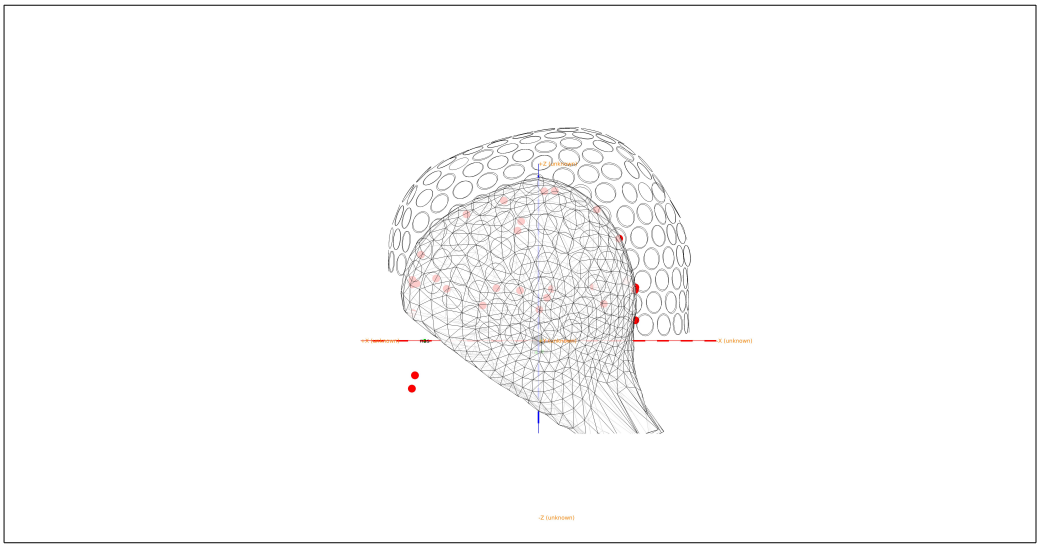


Fig 15 Alignment of scalp, sensors, head shape points (red dots) and axes.

2.7.3. Creating a source space and a lead field

Creating a source space for a beamformer analysis can be done very differently from when one does an MNE type of analysis (Code Snippet 45; Output 5). In MNE analyses, the sources need to be constrained to the cortical surface. For the beamformer analysis, this is not needed. Our strategy is instead to create a grid full of regularly spaced sources, which we can also plot (Code Snippet 46; Fig. 16). The reason for this difference is that beamformer source reconstructions are an example of a scanning technique where activity is estimated for each source in isolation from all other sources. An MNE solution is on the other hand a distributed solution where the source reconstruction is done for all sources at the same time, meaning that an infinite number of solutions would be possible if

the sources were not constrained. In FieldTrip the source model and the lead field can be built at the same time by using *ft_prepare_leadfield*.

```
sens = ft_read_sens(fullfile(data_dir, 'sub-V1002', 'meg', ...
    'sub-V1002_task-visual_meg.ds'), 'senstype', 'meg');
sens = ft_convert_units(sens, 'm');

cfg = [];
cfg.grad = sens;
cfg.headmodel = headmodel;
cfg.resolution = 0.01;
cfg.sourcemodel.unit = 'm';
cfg.channel = 'MEG';

sourcemodel = ft_prepare_leadfield(cfg);

save(fullfile(derived_mr_dir, 'sourcemodel'), 'sourcemodel')
```

Code Snippet 45 Creating a source model containing the leadfield. A regular grid is created with a source for every 10 mm (*cfg.resolution* and *cfg.sourcemodel.unit*).

```
sourcemodel =

               dim: [16 13 12]
               pos: [2496x3 double]
              unit: 'm'
            inside: [2496x1 logical]
               cfg: [1x1 struct]
         leadfield: {1x2496 cell}
             label: {273x1 cell}
   leadfielddimord: '{pos}_chan_ori'
```

Output 5 sourcemodel output: *dim* indicates that the grid contains 16x13x12 sources (2496) (*xyz*-directions); *pos* contains the *xyz*-coordinates for these 2496 sources; *unit* indicates the unit of *pos*; *inside* is a boolean vector indicates whether or not the source is inside the brain; *cfg* has information about the call made for the *sourcemodel* structure; *leadfield* is a cell array containing information about how each of the 2496 sources connect to the MEG sensors – it is empty however for sources that are not inside the brain according to *inside*; *label* contains the name of each (273) of the MEG sensors; *leadfielddimord* indicates how the dimensions of leadfield are to be interpreted: *{pos}_chan_ori* thus means that each cell in the cell array *leadfield* contains a matrix with 273 (each label) x 3 (each orientation (*xyz*)) dimensions.

```
mplot = figure('units', 'normalized', 'outerposition', [0 0 1 1]);
hold on
inside_pos = sourcemodel.pos(sourcemodel.inside, :);
outside_pos = sourcemodel.pos(~sourcemodel.inside, :);
ft_plot_headmodel(headmodel, 'facealpha', 0.6)
ft_plot_mesh(inside_pos, 'vertexcolor', 'blue', 'vertexsize', 30)
ft_plot_mesh(outside_pos, 'vertexcolor', 'red', 'vertexsize', 30);

view(33, 55)

print(mplot, fullfile(figures_dir, 'sourcemodel.jpeg'), '-djpeg', '-r300');
```

Code Snippet 46 Plotting and saving the source model indicating with colours, which source are inside (blue) and which are outside (red) the brain.

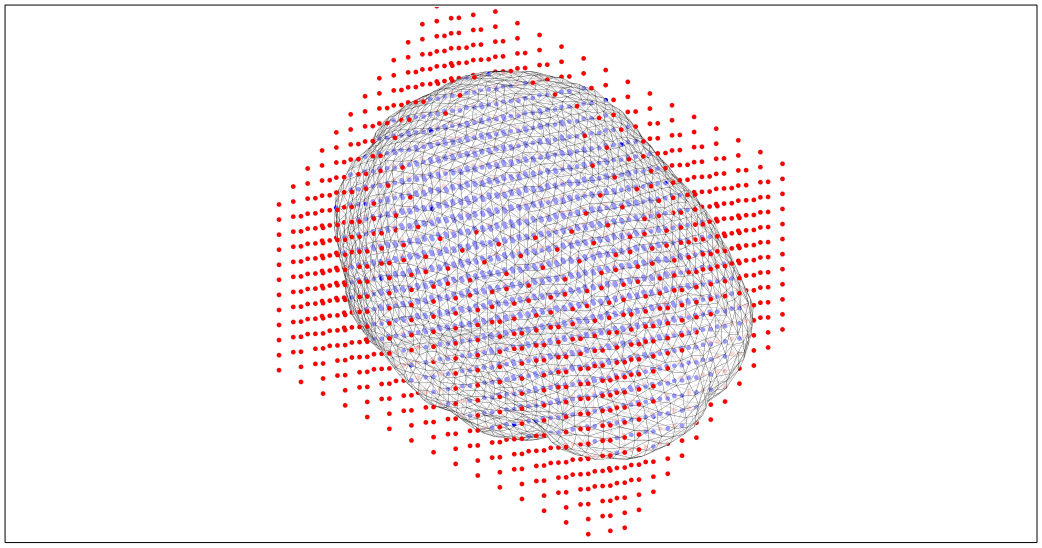


Fig. 17 The blue sources are the ones inside the brain. These have a lead field associated with them, whereas the red ones do not.

2.8. Do beamforming

As I mentioned earlier, when we created the detailed TFR, the data points from the resulting data are going to be heavily correlated with neighbouring data points. In some sense, the smoothness of Fig. 14B is for your eyes only. This means that what we can do with a courser TFR for the subsequent beamforming. In reality, we do not even need the power, but rather the cross-spectral densities to do the DICS beamforming (Gross et al. 2001).

2.8.1. Create a time-frequency representation for beamforming

We are here following Lam et al. (2016). We use a sliding window of 400 ms and a temporal resolution of 50 ms, but focus on the theta and alpha band, namely the two lower frequencies that they report (5 Hz and 10 Hz) (Code Snippet 47). Note also that we run a combined TFR, where we combine all the trials. It will become apparent below why that is so. This means that we do not have an integer number of cycles (see section 2.6.5), but we choose to follow the original paper.

```
n_events = 2;
tfrs = cell(1, n_events);

for event_index = 1:n_events

    cfg = [];
    cfg.method = 'mtmconvol'; %% run tfr
    cfg.taper = 'hanning'; %% use a hanning taper
    cfg.output = 'powandcsd'; %% return both power and cross-spectral density
    cfg.t_ftimwin = [0.400 0.400]; %% use a sliding window of 200 ms
    cfg.toi = -0.100:0.050:0.500; %% time points to centre on
    cfg.pad = 'nextpow2'; %% which padding to use
    cfg.channel = 'meg'; %% which channels to use
    cfg.trials = trials{event_index}; %% which condition, sentence or scrambled
    cfg.foi = [5 10]; %% the two frequencies focused on

    tfrs{event_index} = ft_freqanalysis(cfg, baselined_data);
end

cfg = rmfield(cfg, 'trials');
tfr_combined = ft_freqanalysis(cfg, baselined_data);

tfrs{3} = tfr_combined;

for tfr_index = 1:length(tfrs)
    tfrs{tfr_index}.grad = ft_convert_units(tfrs{tfr_index}.grad, 'm');
end

save(fullfile(derived_meg_dir, 'tfrs'), 'tfrs');
```

Code Snippet 47 Running a coarse TFR in preparation for the beamformer

2.8.2. Plot the coarse the time-frequency representation

Here we just plot using the *relative* baselining. The colouring indicates power relative to the baseline (Fig. 17).

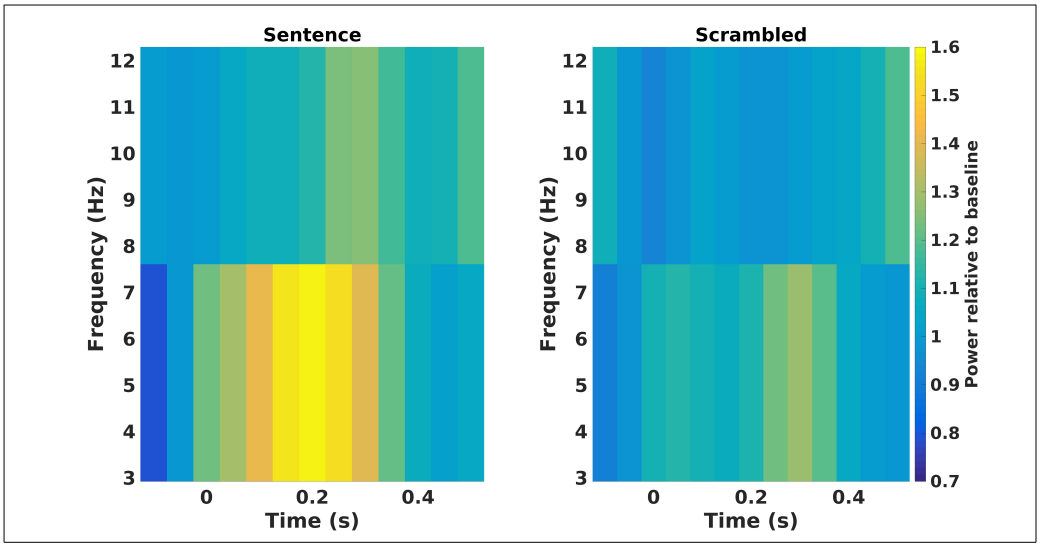


Fig. 17 The response for the two conditions from a left temporal sensor (MLT15). Compare with Fig. 14.

We will focus on the differences in the theta band (~3-7 Hz).

2.8.3. Beamforming the time-frequency pairs

The next step is to apply the DICS beamformer to reconstruct the sources giving rise to the oscillatory power changes in Fig. 17 (Code Snippet 48). For each of the thirteen time steps, we are going to estimate the power of each of the sources in the source model (Fig. 16). The way this is done is to construct a spatial filter for each source that maximally suppresses all activity related to other sources in the source model. In short, each source is estimated in isolation from all other sources. It is important to note that this filter is constructed based on the cross-spectral density of the actual time-frequency pairs. This means that the constructed filters would differ between the two conditions, *sentence* and *scrambled*. To prevent any biases arising from any such differences, we pre-construct the filter based on the *tfr_combined*, making sure that the beamformers for the two conditions use the same filters.

```
n_events = 2;
n_times = length(tfrs{1}.time);
beamformers = cell(1, n_events);

for time_index = 1:n_times

    time = tfrs{1}.time(time_index);

    cfg = [];
    cfg.method = 'dics'; % use Dynamic Imaging of Coherent Sources
    cfg.frequency = tfrs{1}.freq(1); % Choose the theta band
    cfg.sourcemodel = sourcemodel;
    cfg.headmodel = headmodel;
    cfg.dics.projectnoise = 'yes'; % project out noise
    cfg.dics.keepfilter = 'yes'; % keep the created spatial filter
    cfg.dics.realfilter = 'yes'; % use the real part of the filter
    cfg.latency = time; % the time

    beamformer_combined = ft_sourceanalysis(cfg, tfrs{3}); % do the combined

    cfg.sourcemodel.filter = beamformer_combined.avg.filter; % add filter to
                                                             % sourcemodel

    % do for sentence and scrambled
    for event_index = 1:n_events

        if time_index == 1
            beamformers{event_index} = cell(1, n_times);
        end

        beamformers{event_index}{time_index} = ft_sourceanalysis(cfg, ...
                                                    tfrs{event_index});

    end

end

save(fullfile(derived_meg_dir, 'beamformers'), 'beamformers');
```

Code Snippet 48 Two loops looping through each time step and each event. This results in a cell array with two cells (each event, *sentence* and *scrambled*) that each contain thirteen cells (for each of the thirteen time points).

2.8.4. Plotting the beamformer

First, let us have a look at the output of *ft_sourceanalysis* (Output 6).

```
>> beamformers{1}{8}

ans =

      freq: 5.2734
      time: 0.2500
       dim: [16 13 12]
    inside: [2496x1 logical]
       pos: [2496x3 double]
    method: 'average'
       avg: [1x1 struct]
       cfg: [1x1 struct]

>> beamformers{1}{8}.avg

ans =

             pow: [2496x1 double]
           noise: [2496x1 double]
          filter: {2496x1 cell}
           label: {273x1 cell}
    filterdimord: '{pos}_ori_chan'
```

Output 6 Beamformer output: *freq* and *time* indicate the time-frequency pair beamed. *dim*, *inside* and *pos* are inherited from the source model. *avg* contains the interesting parameters. *pow* has the estimated power at each of the 2496 sources of the source model, *noise* is the estimated noise projected out (Code Snippet 48), *filter* is the estimated spatial filter for each of the sources, *label* has the names of the channels and *filterdimord* indicates the meaning of the dimensions of *filter*.

To plot the output, we could simply plot the power (*pow)* on the source positions (*pos*). We are going to interpolate it onto our MR-images though since these are more easily interpreted (Code Snippet 49; Fig. 18).

```
cfg = [];
cfg.parameter = 'pow';
cfg.downsample = 2;

% {1} is sentence, {8} is 250 ms
beamformer_interpolated = ft_sourceinterpolate(cfg, beamformers{1}{8}, ...
                                                mri_resliced);
% SOURCE PLOT

cfg = [];
cfg.method = 'ortho';
cfg.funparameter = 'pow';
cfg.maskparameter = cfg.funparameter;
cfg.crosshair = 'no';
cfg.comment = ' ';
cfg.colorbartext = 'Source Power (Am)^2';

ft_sourceplot(cfg, beamformer_interpolated);

mplot = gcf;
set(mplot, 'units', 'normalized', 'outerposition', [0 0 1 1])

print(fullfile(figures_dir, 'beamformer.jpg'), '-djpeg', '-r300');
```

Code Snippet 49 Interpolate the power estimates from the beamformer onto the MR-image and subsequently plot it using an orthogonal view for the *sentence* condition at 250 ms.

519

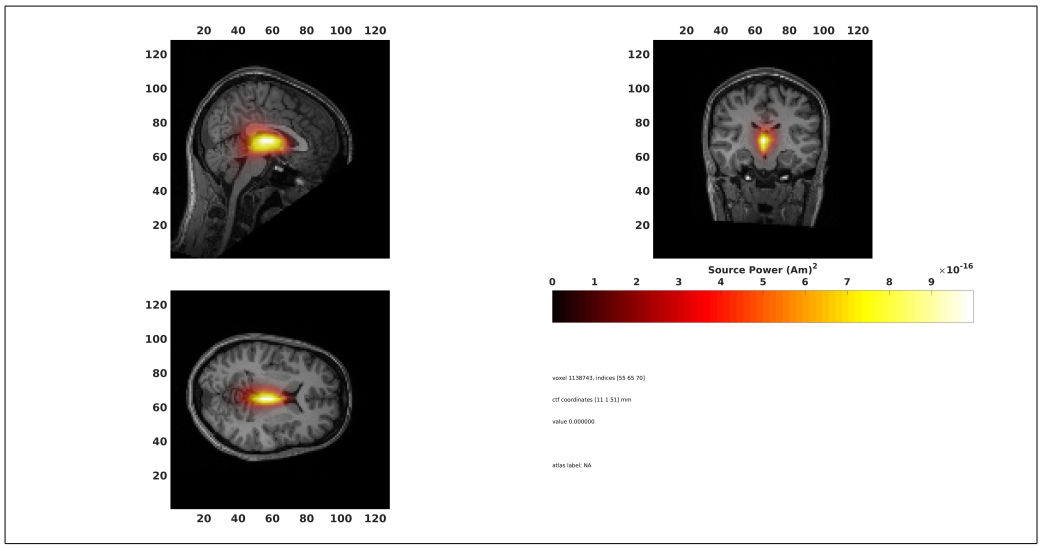


Fig. 18: Beamformer source reconstruction. It can be seen that the reconstructed power is greatest in the centre of the brain.

520 As can be attested from Fig. 18, the greatest power is found in the centre of the brain. Although, this
521 may seem surprising, this is perfectly expected when you consider the fact that sources in the centre
522 of the brain must be very strong to be detectable at all, since they are more or less perfectly radial
523 (Hari and Puce 2017), meaning that their lead fields are very weak. Because of the so-called *unit-*
524 *gain* constraint of the beamformer algorithm, the spatial filter will pass most of the activity to the
525 centre of the brain (and thus also the noise) This means though that it is hard to interpret the
526 meaning of the power of sources throughout the brain for a given beamformer reconstruction. What
527 is more interpretable though is the contrast between two conditions, in our case *sentence* and
528 *scrambled*, since the noise will be equally represented in both. Note that you can use *cfg.normalize*
529 in *ft_prepare_leadfield* if you want to obtain more meaningful non-contrasted images (Code Snippet
530 45).

531 2.8.5. Plotting the contrast between two conditions – beamformer

532 We will now plot the power ratios for each source between *sentence* and *scrambled* (Code Snippet
533 50; Fig. 19).

```
% {1} is sentence, {2} is scrambled, {8} is 250 ms

% how much greater/smaller is the power difference between the two
% conditions to the power in the scrambled condition
cfg = [];
cfg.operation = '(x1 - x2) / x2';
cfg.parameter = 'pow';

beamformer_contrast = ft_math(cfg, beamformers{1}{8}, beamformers{2}{8});

cfg = [];
cfg.parameter = 'pow';
cfg.downsample = 2;

beamformer_contrast_interpolated = ft_sourceinterpolate(cfg, ...
                                                        beamformer_contrast, ...
                                                        mri_resliced);

cfg = [];
cfg.method = 'ortho';
cfg.funparameter = 'pow';
cfg.maskparameter = cfg.funparameter;
cfg.crosshair = 'yes';
cfg.comment = ' ';
cfg.colorbartext = 'Source Power Ratio';
cfg.funcolorlim = [0.10 0.3];
cfg.location = [28 53 63];

ft_sourceplot(cfg, beamformer_contrast_interpolated);

mplot = gcf;
set(mplot, 'units', 'normalized', 'outerposition', [0 0 1 1])

print(fullfile(figures_dir, 'beamformer_contrast.jpg'), '-djpeg', '-r300');
```

Code Snippet 50 making a contrast between the two conditions and plotting it.

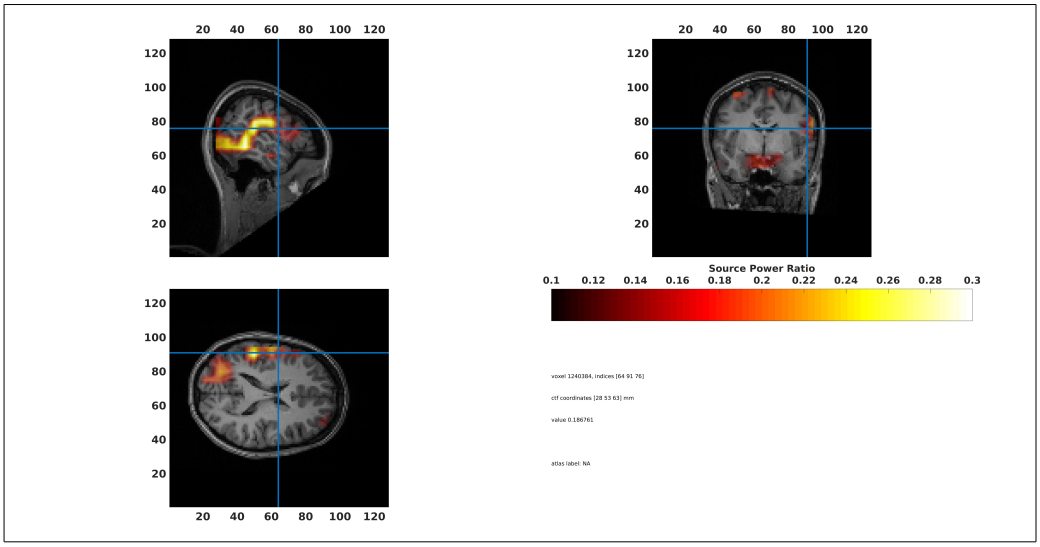


Fig. 19 About 30% more power in the left temporal areas in *sentence* compared to *scrambled*.

This difference between *sentence* and *scrambled* is quite similar to the one found by Lam et al. (2016) in the left temporal areas.

2.8.6. Regularisation

Using beamformer, it may be necessary to regularise the data. This is always the case when your data is not full-rank, such as in EEG when you re-reference the data to a common average or as in MEG if you work with so-called MaxFiltered data as many users of Elekta systems do. This will also be the case if you use Independent Component Analysis (ICA) and do not restore all the components. In our case we did not have to, since our data was full rank. You can add *cfg.dics.lambda* in Code Snippet 48 to see the effects of regularisation. Higher lambdas mean more regularisation. Here is Fig. 19 reproduced using a lambda of 10% (Fig. 20), notice how the solution is more smooth. Do try to play around with different lambdas and see its effects.

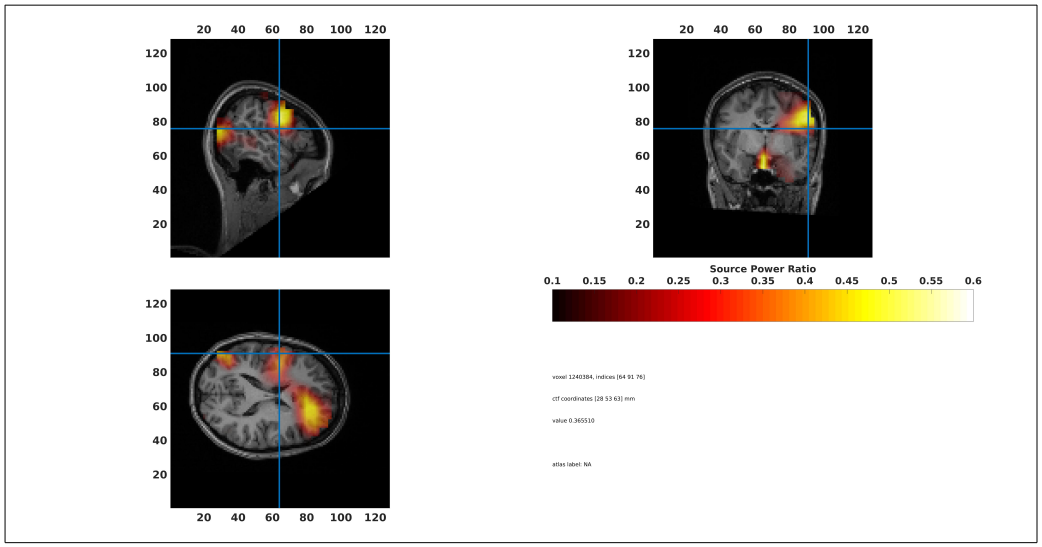


Fig. 20 About 60% more power in the left temporal areas in *sentence* compared to *scrambled*.

2.9. Summary of beamformer analysis

We saw that we can reconstruct oscillatory activity using a beamformer and that we can do contrasts between conditions to see differential effects of these conditions upon the oscillatory activity.

3. Group analysis

Group analysis is essentially single subject analyses applied over and over. A very important feature of robust group analyses is that prerequisite steps can easily be reconstructed if changes need to be made along the way, e.g. in the case of mistakes or a change of strategy. I have earlier published two tested pipelines (Andersen 2018a, b) that saves output along the way for you such that all steps can be reconstructed. I invite the reader to have a look at these Open Access publications that come with a full set of code and data. The issue that they are part of – *From raw MEG/EEG to publication: how to perform MEG/EEG group analysis with free academic software* – can be found at https://www.frontiersin.org/research-topics/5158/from-raw-megeeg-to-publication-how-to-perform-

megeeg-group-analysis-with-free-academic-software#overview where helpful material besides my two papers for group analysis can be found.

There are a few things that differ between single subject and group analyses though. In the MEG analyses above, we used the brain of the individual subject to do the source reconstruction. This is the recommended way to do if you have access to individual MR images. The challenge, then, is how to represent an average in the source space across individuals, because individuals' brains differ from one another. Two strategies can be employed. One is to *morph* each individual's brain onto a common template and then do the averages across the morphed brains. This strategy is followed in Andersen (2018a). Another is to warp the sources of the source model of each individual, such that his or her sources are aligned to a source model based on a template brain. This means that an average can be made across these warped source models, since each source in each warped brain would be aligned with a corresponding source in the template brain. This strategy is followed in Andersen (2018b). Examples of both will be shown below.

3.1. Morphing and warping

Morphing will be highlighted using MNE-Python code and warping will be highlighted using FieldTrip.

3.1.1. Morphing in MNE-Python

Use the function *mne.compute_source_morph* to do the morphing to the common template *fsaverage*, which is distributed with FreeSurfer (Code Snippet 51). Make sure that *fsaverage* is in your *subjects_dir*.

```
#%% ========================================================================
# MORPH TO FSAVERAGE
# ========================================================================

spacing = 'oct6'
src_name = join(bem_folder, subject + '-' + spacing[:3] + '-' + spacing[-1] + \
                '-src.fif')

src = mne.read_source_spaces(src_name)
stcs_morphs = dict()

for stc in stcs:
    morph = mne.compute_source_morph(src, subject, 'fsaverage', subjects_dir)
    stcs_morphs[stc] = morph.apply(stcs[stc])
```

Code Snippet 51 Morphing to the template brain *fsaverage* distributed with FreeSurfer. Make sure that *fsaverage* is in your *subjects_dir*.

Now *stcs_morphs* can be plotted using Code Snippet 30, changing *subject* to *'fsaverage'*. This brings subjects into a common space in which they can be averaged such that grand averages can be obtained for the source space.

3.1.2. Warping in FieldTrip

Using *cfg.sourcemodel.warpmni* with *ft_prepare_leadfield* warps the individual's MRI to the an MNI template meaning that the sources of individuals will be aligned across source spaces even though all calculations are done in the individual's source space (Code Snippet 52; Fig. 21). This

allows for grand averages across source spaces if the *pos* field of each individual's source reconstruction is switched for the *pos* field of the template MNI before averaging.

```
load(fullfile(derived_mr_dir, 'headmodel.mat'))

sens = ft_read_sens(fullfile(data_dir, 'sub-V1002', 'meg', ...
    'sub-V1002_task-visual_meg.ds'), 'senstype', 'meg');
sens = ft_convert_units(sens, 'm');

mri = ft_read_mri(fullfile(data_dir, 'sub-V1002', 'anat', ...
                           'sub-V1002_space-CTF_T1w.nii'));

mri.coordsys = 'ctf';

MNI_template = fullfile(fieldtrip_dir, 'template', 'sourcemodel', ...
                        'standard_sourcemodel3d10mm.mat');

cfg = [];
cfg.grad = sens;
cfg.headmodel = headmodel;
cfg.sourcemodel.unit = 'm';
cfg.sourcemodel.warpmni = 'yes';
cfg.sourcemodel.template = MNI_template;
cfg.sourcemodel.nonlinear = 'yes';
cfg.mri = mri;
cfg.channel = 'MEG';

warped_sourcemodel = ft_prepare_leadfield(cfg);

mplot = figure('units', 'normalized', 'outerposition', [0 0 1 1]);
hold on
ft_plot_headmodel(headmodel)
ft_plot_mesh(warped_sourcemodel)
view(180, 0)

print(mplot, fullfile(figures_dir, 'warped_sourcemodel.jpeg'), '-djpeg', ...
    '-r300')
```

Code Snippet 51 Creating the MNI warped sourcemodel and plotting it.

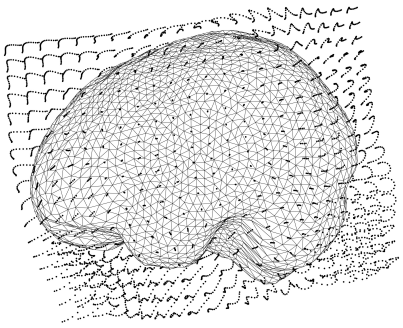


Fig. 21 An MNI warped source model that can be used to facilitate grand averages across a group of subjects.

3.2. Final words

The greatest challenge when doing group analysis is not technical, since only a few steps differ between single subject and group analyses. The greatest challenge is practical – keeping track of your analysis and keeping it structured. van Vliet has seven guiding principles that he proposes that I encourage an aspiring EEG or MEG analyser to consult. My group analysis (Andersen 2018a, b) articles to a high degree overlap with guiding principles.

Best of luck with your analysis endeavours!

599 # 4. References

Andersen LM (2018a) Group Analysis in MNE-Python of Evoked Responses 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. Front Neurosci 12:. doi: 10.3389/fnins.2018.00006

Andersen LM (2018b) Group Analysis in FieldTrip of Time-Frequency Responses: A Pipeline for Reproducibility at Every Step of Processing, Going From Individual Sensor Space Representations to an Across-Group Source Space Representation. Front Neurosci 12:. doi: 10.3389/fnins.2018.00261

Clausner T, Dalal SS, Crespo-García M (2017) Photogrammetry-Based Head Digitization for Rapid and Accurate Localization of EEG Electrodes and MEG Fiducial Markers Using a Single Digital SLR Camera. Front Neurosci 11:. doi: 10.3389/fnins.2017.00264

Delorme A, Makeig S (2004) EEGLAB: an open source toolbox for analysis of single-trial EEG dynamics including independent component analysis. J Neurosci Methods 134:9–21. doi: 10.1016/j.jneumeth.2003.10.009

Friston K, Ashburner J, Kiebel S, et al (2007) Statistical Parametric Mapping: The Analysis of Functional Brain Images, 1st Edition | William Penny, Karl Friston, John Ashburner, Stefan Kiebel, Thomas Nichols | ISBN 9780123725608

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

Gross J, Kujala J, Hämäläinen M, et al (2001) Dynamic imaging of coherent sources: Studying neural interactions in the human brain. Proc Natl Acad Sci 98:694–699. doi: 10.1073/pnas.98.2.694

Hämäläinen MS, Hari R, Ilmoniemi RJ, et al (1993) Magnetoencephalography—theory, instrumentation, and applications to noninvasive studies of the working human brain. Rev Mod Phys 65:413–497. doi: 10.1103/RevModPhys.65.413

Hari R, Puce A (2017) MEG-EEG Primer. Oxford University Press, New York, NY, US

Lam NHL, Schoffelen J-M, Uddén J, et al (2016) Neural activity during sentence processing as reflected in theta, alpha, beta, and gamma oscillations. NeuroImage 142:43–54. doi: 10.1016/j.neuroimage.2016.03.007

Oostenveld R, Fries P, Maris E, Schoffelen J-M (2011) FieldTrip: Open source software for advanced analysis of MEG, EEG, and invasive electrophysiological data. Comput Intell Neurosci 2011:156869. doi: 10.1155/2011/156869

Schoffelen J-M, Oostenveld R, Lam NHL, et al (2019) A 204-subject multimodal neuroimaging dataset to study language processing. Sci Data 6:17. doi: 10.1038/s41597-019-0020-y

Simanova I, Gerven M van, Oostenveld R, Hagoort P (2010) Identifying Object Categories from Event-Related EEG: Toward Decoding of Conceptual Representations. PLOS ONE 5:e14465. doi: 10.1371/journal.pone.0014465

Tadel F, Baillet S, Mosher JC, et al (2011) Brainstorm: A User-friendly Application for MEG/EEG Analysis. Intell Neurosci 2011:8:1–8:13. doi: 10.1155/2011/879716

600