ToFFi

Toolbox for Frequency-based Fingerprinting

Michał Konrad Komorowski
michu.kom@gmail.com

Faculty of Philosophy and Social Sciences, Institute of Information and Communication Research,
Department of Cognitive Science,

Faculty of Physics, Astronomy and Informatics,
Department of Informatics,

Nicolaus Copernicus University in Toruń, Poland

Supporting authors:

- Tomasz Piotrowski, Włodzisław Duch
  Department of Informatics, Faculty of Physics, Astronomy, and Informatics,
  Nicolaus Copernicus University, Toruń, Poland
- Krzysztof Rykaczewski
  Neurocognitive Laboratory, Centre for Modern Interdisciplinary Technologies, Nicolaus Copernicus University, Toruń, Poland
- Katarzyna Jurewicz
  Nencki Institute of Experimental Biology, Polish Academy of Sciences, Warsaw, Poland
- Jakub Wojciechowski
  Bioimaging Research Center, Institute of Physiology and Pathology of Hearing, Kajetany, Poland
- Joanna Dreszer
  Faculty of Philosophy and Social Sciences, Institute of Psychology, Nicolaus Copernicus University, Toruń, Poland
- Anne Keitel
  Division of Psychology, University of Dundee, Dundee, United Kingdom

Publication: https://doi.org/10.1016/j.neucom.2023.126236
URL: ToFFi repository
Last modification: May 4, 2023

Contents

Part I

Introduction

This part covers the motivation behind the toolbox development, instructions on installation and helps to get started with the toolbox, guiding the user through the illustrative example.

1 Motivation

Brain dynamics and brain oscillations are among the most critical topics in neuroscience. Different methods proved to help study robust whole-brain and regionally-specific patterns of activity, called brain fingerprints. They can serve as signatures for mental states during task execution or rest [1], [2], [3], [4]. Frequency of oscillations turned out to be one of the key features in many studies describing particular regions of interest (ROIs) [5], [6], [7] and large-scale brain networks [8], [9], [10], [11], [12].

Keitel & Gross in 2016 [7] conceived a method of combining magnetoencephalographic multichannel signals, representing human brain oscillatory activity, and anatomical structure of the brain into sets of regional spectra (Fig. 4.1, Fig. 4.5, Fig. 4.6) that were intended to serve as meaningful signatures, allowing a robust recognition of given brain areas through their characteristic neural patterns. **Spectral fingerprints** turned out to be stable biomarkers that are sufficiently specific to permit the successful identification of brain regions. Moreover, spectral profiles' peaks, which correspond to the natural frequencies of ROIs, are consistently modulated by specific tasks, neurological or mental disorders, and they can be generalized across groups of participants [13], [7].

Here, we introduce our novel implementation of the Spectral Fingerprinting technique as a highly-configurable MATLAB toolbox. There are many open software packages available to analyze neural data. The Fieldtrip Toolbox [14] was designed to perform analysis both on sensor and source level of EEG/MEG/iEEG/NIRS data. EEGLAB [15] helps with processing continuous and event-related electrophysiological data implementing many analytic methods (ICA, time/frequency analysis, artifact rejection, event-related statistics, microstates analysis) and several useful routines for visualization. To simulate brain dynamics, perform connectivity analyses, and solve forward/inverse problems, the supFunSim toolbox [16] and the Virtual Brain system [17] are among suitable choices.

However, there is no open software for analyzing spectral fingerprints, and our work attempts to fill this gap. We have designed the **Toolbox for Frequency-based Fingerprinting (ToFFi)** for the analysis of MEG, EEG, and other multichannel data. One can configure many parameters for each processing stage and decide which of them will run in parallel (cluster computations are also supported for Linux clusters equipped with SLURM workload manager). Results of the computations are reproducible thanks to the implemented control using pseudo-random number generators and visualization scripts.

2 Conventions

NOTE

Important notes are written inside blue boxes.

WARNING!

Red boxes contain essential notes, which should be taken with much care. They help to avoid many of the errors that could otherwise happen.

In this document paths, file names, file extensions, variables, and terminal/MATLAB's Command Window text output are written using `monospaced fonts`.

Important or new concepts are introduced using *slanted font*.

For notation regarding file paths:

- `./` means the current location (a.k.a. *working directory*),
- `../` means the parent directory of the current working directory.

In Windows, paths are delimited by backslashes, for example: `C:\johndoe\ToFFi_Toolbox`

In Linux, paths are delimited by forward slashes, for example: `/home/johndoe/ToFFi_Toolbox`

An *absolute path* points to the exact location in a file system, regardless of the current working directory. By contrast, a *relative path* starts from some given working directory, avoiding the need to provide the full absolute path. For example:

- `C:\johndoe\ToFFi_Toolbox` is an absolute path.
- `.\ToFFi_Toolbox` is a path relative to the current working directory, which is `C:\johndoe` in this example.

In Linux, absolute paths starts with a slash sign `/` which means the root directory, e.g.:

- `/usr/local/bin`
- `/home/johndoe/ToFFi_Toolbox`

WARNING! Avoid tilde (~) symbol

When configuring the toolbox, one should not use the tilde symbol (~), when referring to user's home directory, e.g. for user `johndoe`:

`someVariable = '~/ToFFi_Toolbox'` (**incorrect!**)
`someVariable = '/home/johndoe/ToFFi_Toolbox'` (correct!)

In Windows, absolute paths starts with a partition name like `C:` whichs mean the root directory, e.g.:

- `C:\Program_Files`
- `D:\johndoe\ToFFi_Toolbox`

Toolbox components names (see chapter 7) are written using CAPITAL LETTERS.

A path to the downloaded repository directory is called the **`$REPO_ROOT`**, e.g. `C:\johndoe\ToFFi_Toolbox-YYYYMMDD\`.
In practice, `YYYYMMDD` should be substituted with the proper toolbox revision date, e.g.
`C:\johndoe\ToFFi_Toolbox-20210826\`.
The `$REPO_ROOT` symbol is used extensively throughout this documentation.

3 Installation

NOTE: Necessary toolboxes and recommended versions of the software

Recommended versions of the software:
- MATLAB - R2021a or newer,
- Fieldtrip - revision 20210816 or newer.

The following toolboxes need to be installed alongside MATLAB:
- Signal Processing Toolbox,
- Statistics and Machine Learning Toolbox,
- Parallel Computing Toolbox.

1. Install MATLAB,
2. Download the Fieldtrip Toolbox
   (use `https://www.fieldtriptoolbox.org/download/` or download it directly using any FTP client to enter `ftp://ftp.fieldtriptoolbox.org/pub/fieldtrip/`),
3. Setup Fieldtrip,
4. Download the ToFFi repository,
5. Extract downloaded repository if it is compressed within a `.zip` file,
6. The ToFFi toolbox is ready to use.

WARNING! MATLAB Search paths

1. **Do not** manually add any MATLAB search paths related to the ToFFi Toolbox content using `addpath` command or GUI, as it may confuse the toolbox.
2. Before running the toolbox routines, remove any MATLAB search paths to old Fieldtrip versions you do not plan to use. Otherwise, Fieldtrip will behave confusingly and could produce unexpected errors.

Consult:
`https://www.mathworks.com/help/matlab/matlab_env/what-is-the-matlab-search-path.html`
to learn more about MATLAB search paths.

To get started check 4 ILLUSTRATIVE EXAMPLE.

4 ILLUSTRATIVE EXAMPLE

NOTE: Quick Start

To skip the theoretical introduction to this example and to start immediately, jump to 4.5 Follow-through guide.

We recommend this section to be the starting point for learning how to use the provided software. This is the most rewarding part of this document because one can get a grasp on many concepts regarding Spectral Fingerprinting relatively quickly. It starts with the definition of the spectral fingerprint and overviews how it is constructed. A brief description of the input data helps to prepare the main part of the example, which is a loosely stated problem of finding resting-state regional fingerprints inside prescribed the frequency band. One is guided precisely through the process of preparing the data, configuring its components, launching calculation, and analyzing the results.

4.1 What is a spectral fingerprint?

Mathematically, a spectral fingerprint of a particular brain region (Fig. 4.1) can be rendered as a set of pairs:

$$\mathrm{SF} = (\mathbf{m}_i, t_i)_{i=1}^{k}$$

where $\mathbf{m}_i \in \mathbb{R}^F$ is the i-th vector called a *spectral mode*, F is the number of frequencies of interest, $t_i \in (0; 1]$ is the duration of the i-th spectral mode, and k is the number of spectral modes.

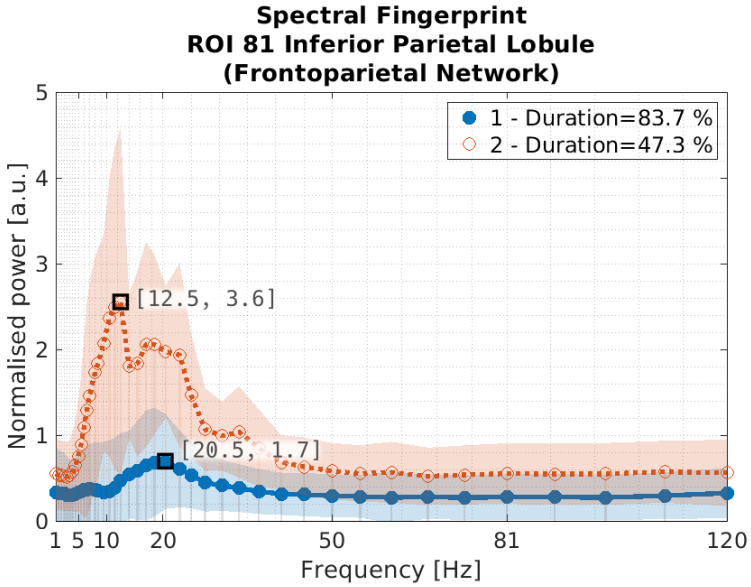


Figure 4.1: A spectral fingerprint of the inferior parietal lobule. For this particular region, it consists of two spectral modes ($k = 2$). It is formed by clustering power spectra segments (normalized, i.e. spectral power in comparison to the whole brain) first on the individual subjects level and then clustered again on the group level. Each mode is one of the centroids found by the clustering algorithm and shows normalized power for $F = 42$ frequencies of interest. Shaded regions depict the standard deviation (1σ) estimated from the covariance matrix of the Gaussian Mixture Model component corresponding to the given spectral mode. The first mode peaks at 12.5 Hz, and the second mode peaks at 20.5 Hz. The frequency axis resolution is set to logarithmic to optimize spectral analysis resolution of lower frequencies. Duration is shown as a percentage of time segments in which each spectral mode was present on average during recording.

4.2 Method overview

Figure 4.2 illustrates the idea behind the Spectral Fingerprinting method. Activity of the brain is captured using an array of sensors. Collected and cleaned from artifacts, continuous signal is splitted into separate non-overlapping

windows called *segments*. Some segments can be rejected if they still contain artifacts. Then, actual neural activity segments are estimated for given brain locations using a source reconstruction algorithm. Next, each segment's power spectra are obtained using Fourier Transform. Power values at frequencies of interest can be rendered as vectors' coordinates in a frequency space so that each segment can be viewed as a single point in this space. Similar brain activity will occupy some volume spanned across points in proximity, thus clustering is performed to find the centroids of those volumes. These centroids represent the common patterns in neural activity, called *spectral modes*. Each spectral mode has its *duration* which is proportional to the number of points belonging to that cluster. A set of spectral modes for a given brain region constitutes a regional *fingerprint*.

Note that this fingerprinting process is done at an individual level, i.e. a signal from a single subject is used to construct a fingerprint. Fingerprint obtained on this level is called an *individual fingerprint*. The process can go a step further. Spectral modes for a given brain region but from different subjects can be clustered again and thus form so-called *group-level fingerprint* or a *spectral fingerprint*. This fingerprint is not assigned to a particular subject but represents activity patterns common for a group of subjects.

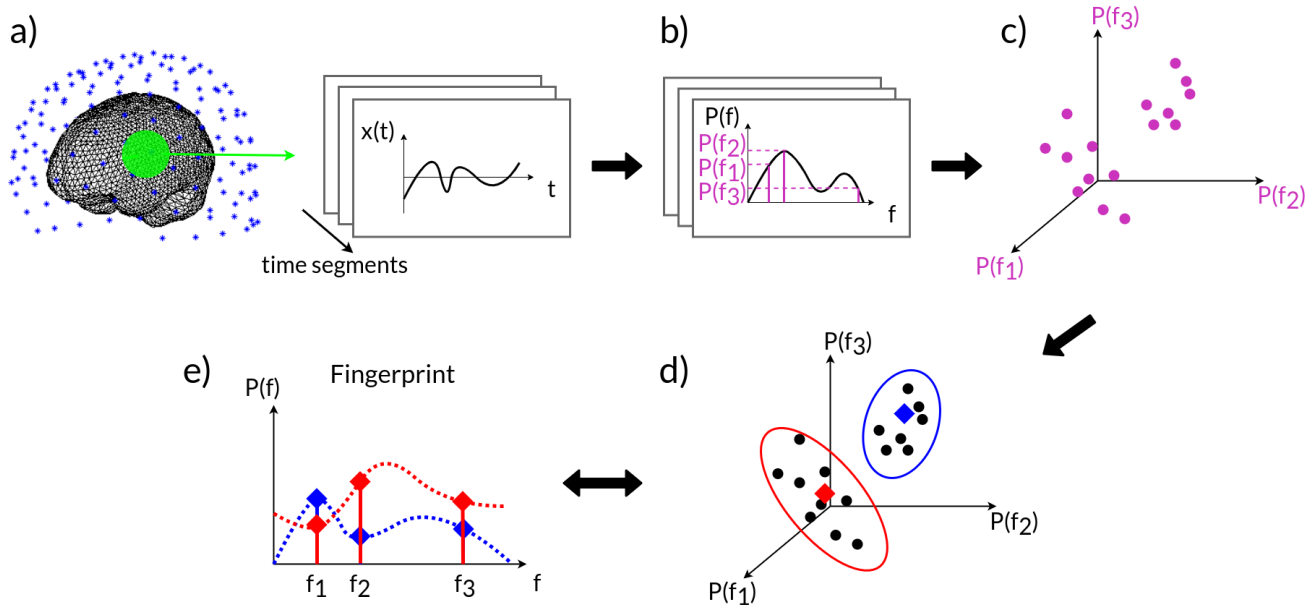


Figure 4.2: Illustration of the Spectral Fingerprinting algorithm [7]. a) Mean-field, electrical activity of the brain is recorded via an array of sensors (blue dots). A beamformer [18] solves so-called *inverse problem*, thus enabling reconstruction of the source activity in selected voxels constituting chosen brain regions of interest (green area). Source-level time series are cut into segments of equal length. b) Power within each segment is estimated for a given set of frequencies of interest (here three frequencies, f_1, f_2, f_3, are shown for readability) and then averaged across locations inside the region of interest. c) Power spectrum of each segment is represented as a point in a n-dimensional frequency space, where power of the selected frequencies provides coordinates of the point. d) Segments are clustered together. Each centroid is equivalent to one *spectral mode* of a given brain region and depicted as an interpolant curve spanned between the frequencies of interest in the resulting fingerprint (interpolation type is arbitrary, as it serves visualization purpose only).

After going through the overview of the method, let us define the illustrative problem that will demonstrate how to use the ToFFi toolbox.

4.3 Illustrative problem

This example aims to showcase the ToFFi toolbox capabilities by finding and studying fingerprints in the 1–40 Hz frequency interval, from different distant parts of the brain, both on the individual- and group-level. We want to find the characteristic frequencies of their activity, how similar or dissimilar these fingerprints are, and how well we can discriminate between them (identification).

4.4 Input data

In this example, the operation of the ToFFi toolbox will be demonstrated on the Human Connectome Project [19] magnetoencephalographic dataset (5.2 Brief description of the HCP dataset). We will select the first N=10 subjects with the MEG resting-state signal acquired via 248 channel array in first of the total of three subsequent runs, approx. 3 min. each. Normally, the larger the subject sample, the better the results, but here, for reasonable processing time, we will limit ourselves to ten subjects and a single run, as mentioned.

The entry point for the Spectral Fingerprinting method is source reconstruction which involves inverting a *forward model* (Fig. 4.3):

$$\mathbf{y} = \mathbf{Hq} + \mathbf{n} \tag{4.1}$$

where: $\mathbf{y}$ - sensors signal $\in \mathbb{R}^{m \times T}$, m - the number of sensors, T - the number of sampling points, $\mathbf{H}$ - scalar leadfield matrix (i.e., dipole orientations are fixed) $\in \mathbb{R}^{m \times L}$, L - the number of sources, $\mathbf{q}$ - sources signal $\in \mathbb{R}^{L \times T}$, $\mathbf{n}$ - noise $\in \mathbb{R}^{m \times T}$.

Source data can then be estimated as:

$$\hat{\mathbf{q}} = \mathbf{Wy} \tag{4.2}$$

where: $\mathbf{q}$ - estimated sources signal $\in \mathbb{R}^{L \times T}$, $\mathbf{W}$ - spatial filter matrix $\in \mathbb{R}^{L \times m}$.

As a consequence, one needs then to provide sensor signal $\mathbf{y}$ and spatial filter matrices $\mathbf{W}$ for all the subjects one wants to study.

From the fact that the inverse problem is ill-posed and therefore has no unique solution, there are different techniques to reconstruct the sources [20], [21], [22], [23], [24], [25], [26], [27], [28], [29], [30]. Here we will use *linearly constrained minimum variance* beamformers (LCMV) [20] to serve as spatial filters.

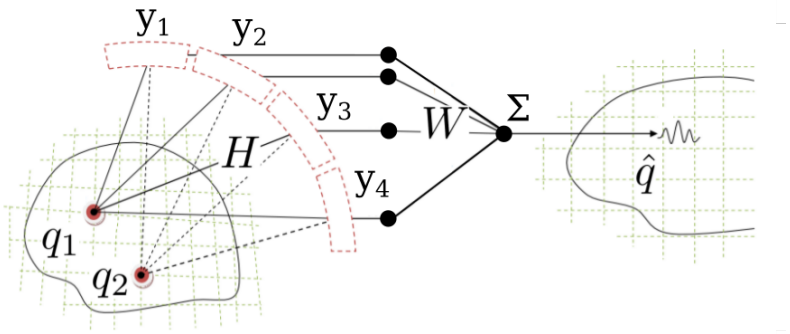


Figure 4.3: Source reconstruction schematic. Propagation of the activity of the sources of interest, q_1 and q_2, to the sensors is modeled via leadfield matrix $\mathbf{H}$. This process can be inverted using the spatial filter matrix $\mathbf{W}$. Each estimated source $\hat{q}$ is reconstructed as a linear combination of the sensors signals y_1, y_2, y_3, y_4, and spatial filter matrix row coefficients. Figure adapted from the Fieldtrip Toolbox Walktrough (`https://www.fieldtriptoolbox.org/walkthrough/`).

One needs also to define *regions of interest* (ROIs) by providing a proper brain atlas structure (see 5.5 Precomputed brain atlases and `brain atlas` structure in 6.2 Diagram data description) and a common-space grid template (described below) in order to enable group-level analyses.

For simplicity, we will choose 8 regions of interest (Fig. 4.4) out of 113 ROIs from the Desikan–Killiany segmentation [31] available in the dataset (5.5 Precomputed brain atlases):

- left precentral gyrus (`ctx-lh-precentral`, ROI nr 67),
- right precentral gyrus (`ctx-rh-precentral`, ROI nr 102),
- left lateral occipital gyrus (`ctx-lh-lateraloccipital`, ROI nr 54),

- right lateral occipital gyrus (`ctx-rh-lateraloccipital`, ROI nr 89),
- anterior part of left middle frontal gyrus (`ctx-lh-rostralmiddlefrontal`, ROI nr 70),
- anterior part of right middle frontal gyrus (`ctx-rh-rostralmiddlefrontal`, ROI nr 105),
- left thalamus (`Left-Thalamus-Proper`, ROI nr 6),
- right thalamus (`Right-Thalamus-Proper`, ROI nr 25).

Regions were chosen to cover different sites of the human brain and possibly explore the variety of neural activities.

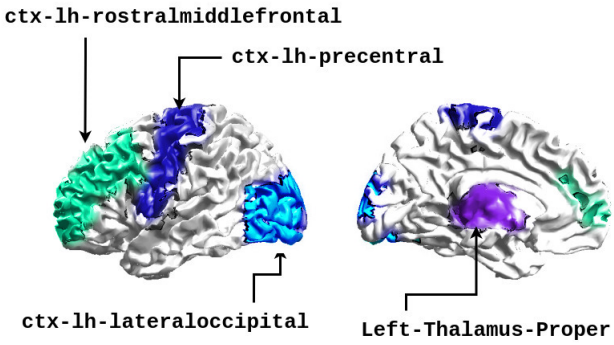


Figure 4.4: Brain regions chosen from the Desikan–Killiany atlas for the purpose of the illustrative example. Here only the left counterpart is depicted, whereas right hemisphere homologues were chosen as well.

To summarize, the data needed to run all the stages of the Spectral Fingerprinting are:

- multichannel signal arrays divided into segments of equal duration,
- spatial filter matrices,
- brain atlas (for brain regions definition),
- common source grid (to enable group-level clustering and comparisons),

which were implemented as the following MATLAB variables, respectively:

- `data`
  (read `STAGE_1/functions/checkDataAndFilter.m` for details),
- `spatialFilter`
  (read `STAGE_1/functions/checkDataAndFilter.m` for details),
- `sourceAtlas`
  (this is the output of the `globalFunctionsScripts/prepareAtlas.m` - read it for details),
- `sourcemodel`
  (we have provided the `commonData/templategrid_HCP_8mm.mat` for this purpose. It can be also generated using `DATA_PREPARATION/PARCELLATIONS_PREP/common_grid_prep.m` script.)

NOTE: Input variables

Suppose one wants to use particular data, either recorded in his laboratory or already available in one of the many online repositories. In that case, one needs to adjust his data to follow the design of the variables mentioned above. Check 5 INPUT DATA section to see the details of these variables.

The following section will describe step-by-step how to prepare HCP data, how to configure and run the Spectral Fingerprinting to solve posed illustrative problem.

4.5 Follow-through guide

Instructions one must follow are written inside green boxes. The rest of the text in this section supports the user by providing explanations and important notes.

WARNING: Running scripts

When running scripts, they should be run from their location directory, i.e. MATLAB's working directory should be set to the script's location to be run!

Hint 1: The simplest way to do this (providing one is running MATLAB with GUI and have default keyboard shortcut settings) is to run scripts by hitting `F5` and then click the `"Change folder"` button in the popup window.

Hint 2: In case of errors, one should check first if the correct working directory is set.

The diagram in Fig. 7.1 can help to imagine how different components of the software (denoted by the Roman numerals I, II, III, IV, V) are situated in the toolbox design. Having this in mind, in this section, we will describe step-by-step how to prepare HCP data (I) and how to run the Spectral Fingerprinting (II) in order to solve posed illustrative problem. Scripts from component III will also be launched in order to prepare the analysis of the obtained fingerprints, aided by visualizations generated by component IV.

NOTE: Preliminaries

At this point it is assumed that you have:

- downloaded the HCP data (5.4 How to download the HCP data?),
- downloaded (and extracted from a `.zip` package if necessary) the repository containing the ToFFi Toolbox,
- installed MATLAB (version R2021a is recommended),
- installed and configured the Fieldtrip toolbox (revision 20210816 or newer is recommended).

In order to run the illustrative example, the data for the following subjects are needed: 100307 (Sub_1), 102816 (Sub_2), 105923 (Sub_3), 106521 (Sub_4), 108323 (Sub_5), 109123 (Sub_6), 111514 (Sub_7), 112920 (Sub_8), 113922 (Sub_9), 116524 (Sub_10).

Please see section 5.3 to check summary list of HCP files needed.

NOTE: `$REPO_ROOT`

A path to the downloaded repository directory is called the **`$REPO_ROOT`**.
Make sure that there is no duplicated nested directory like:

`C:\johndoe\ToFFi_Toolbox-YYYYMMDD\ToFFi_Toolbox-YYYYMMDD\`.

Path containing the repository files like `CONFIGURE.m` should look like this:
`C:\johndoe\ToFFi_Toolbox-YYYYMMDD\`

In practice, `YYYYMMDD` should be substituted with the proper toolbox revision date, e.g. `C:\johndoe\ToFFi_Toolbox-20211013\`.

4.5.1 Configuring and launching: Data Preparation (I)

MEG data from the Human Connectome Dataset is not suited to be analyzed by the ToFFi toolbox out of the box. We need to slice the signal into pieces and compose an LCMV spatial filter, so the main routines of the ToFFi toolbox can estimate cerebral activity and cluster it into spectral modes - the main building block of each spectral fingerprint. This means that the first step will be to use DATA PREPARATION scripts, in the given order, to adjust the data.

These adjustments are:

- Reslicing signals into segments of a given duration,
  (script: `$REPO_ROOT/DATA_PREPARATION/HCP_DATA_PREP/01_PREPROCESSING/PREPROCESSING.m`)
- Detrending signals inside the segments,
  (script: `$REPO_ROOT/DATA_PREPARATION/HCP_DATA_PREP/01_PREPROCESSING/PREPROCESSING.m`)
- Generating covariance matrices,
  (script: `$REPO_ROOT/DATA_PREPARATION/HCP_DATA_PREP/02_COVARIANCE_GEN/COVARIANCE_GEN.m`)
- Generating leadfields,
  (script: `$REPO_ROOT/DATA_PREPARATION/HCP_DATA_PREP/03_LEADFIELD_GEN/LEADFIELD_GEN.m`)
- Combining covariance matrices and leadfields into LCMV spatial filters,
  (script: `$REPO_ROOT/DATA_PREPARATION/HCP_DATA_PREP/04_LCMV_FILTER_GEN/LCMV_FILTER_GEN.m`)
- Renaming the data files and variables.
  (script: `$REPO_ROOT/DATA_PREPARATION/HCP_DATA_PREP/05_RENAMING_TOOL/RENAMING_TOOL.m`

These steps will provide a necessary foundation for running STAGE 1 - Source Projection, which will transform the data from sensor-level time-series into the source-level spectra.

Let us assume that path to the Fieldtrip toolbox is `C:\johndoe\fieldtrip-20210816\` and the HCP data is downloaded somewhere, e.g. located inside `C:\johndoe\HCP_data\`. This directory should have the following structure and obligatory files shown below (example for two subjects '100307' and '102816'). Note that there might be additional directories like `provenance`, `figures`, etc. Their presence will not harm anything. It is crucial to have at least the files shown below:

```
C:\johndoe\HCP_data\
                ├── 100307\
                │   ├── MEG\
                │   │   ├── anatomy\
                │   │   │   ├── 100307_MEG_anatomy_headmodel.mat
                │   │   │   ├── 100307_MEG_anatomy_sourcemodel_3d8mm.mat
                │   │   └── Restin\
                │   │       └── rmegpreproc\
                │   │           ├── 100307_MEG_3-Restin_rmegpreproc.mat
                │   │           ├── 100307_MEG_4-Restin_rmegpreproc.mat         <--- optional
                │   │           ├── 100307_MEG_5-Restin_rmegpreproc.mat         <--- optional
                │   └── MNINonLinear\
                │       └── aparc+aseg.nii.gz
                ├── 102816\
                │   ├── MEG\
                │   │   ├── anatomy\
                │   │   │   ├── 102816_MEG_anatomy_headmodel.mat
                │   │   │   ├── 102816_MEG_anatomy_sourcemodel_3d8mm.mat
                │   │   └── Restin\
                │   │       └── rmegpreproc\
                │   │           ├── 102816_MEG_3-Restin_rmegpreproc.mat
                │   │           ├── 102816_MEG_4-Restin_rmegpreproc.mat         <--- optional
                │   │           ├── 102816_MEG_5-Restin_rmegpreproc.mat         <--- optional
                │   └── MNINonLinear\
                │       └── aparc+aseg.nii.gz
                │   ... ...
```

WARNING!

Inside the `C:\johndoe\HCP_data` directory, there should not be any other files but directories named with the subject codes!

INSTRUCTIONS (Data Preparation)

To configure and run DATA PREPARATION scripts properly:

1. Open MATLAB.
2. Edit script
   `$REPO_ROOT/DATA_PREPARATION/HCP_DATA_PREP/CommonInit.m`:
   (look for `%% USER SETTINGS` sections)
   - `inputDataPath = 'C:/johndoe/HCP_data/' % absolute path!`
   - `fieldTripPath = 'C:/johndoe/fieldtrip-20210816/' % absolute path!`
   - `nSub = 10`
3. Set working directory to `$REPO_ROOT/DATA_PREPARATION/HCP_DATA_PREP/01_PREPROCESSING/`,
4. Edit script `$REPO_ROOT/DATA_PREPARATION/HCP_DATA_PREP/01_PREPROCESSING/PREPROCESSING.m`:
   - `doDetrending = 1`
5. Run `PREPROCESSING.m`,
6. Set working directory to `$REPO_ROOT/DATA_PREPARATION/HCP_DATA_PREP/02_COVARIANCE_GEN/`,
7. Edit script `$REPO_ROOT/DATA_PREPARATION/HCP_DATA_PREP/02_COVARIANCE_GEN/COVARIANCE_GEN.m`:
   - `DO_DATA_COND = 0`
8. Run `COVARIANCE_GEN.m`,
9. Set working directory to `$REPO_ROOT/DATA_PREPARATION/HCP_DATA_PREP/03_LEADFIELD_GEN/`,
10. Edit script `$REPO_ROOT/DATA_PREPARATION/HCP_DATA_PREP/03_LEADFIELD_GEN/LEADFIELD_GEN.m`:
    - `dataType = 'MEG'`
11. Run `LEADFIELD_GEN.m`,
12. Set working directory to `$REPO_ROOT/DATA_PREPARATION/HCP_DATA_PREP/04_LCMV_FILTER_GEN/`,
13. For script `$REPO_ROOT/DATA_PREPARATION/HCP_DATA_PREP/04_LCMV_FILTER_GEN_GEN/LCMV_FILTER_GEN.m` no additional configuration is needed.
14. Run `LCMV_FILTER_GEN.m`,
15. There should be the following new files for each subject (here an example for subject `100307`):

```
$REPO_ROOT/DATA_PREPARATION/HCP_DATA_PREP/output/100307/100307_MEG_3-Restin_rmegpreproc_preproc_covmat.mat
$REPO_ROOT/DATA_PREPARATION/HCP_DATA_PREP/output/100307/100307_MEG_3-Restin_rmegpreproc_preprocessed.mat
$REPO_ROOT/DATA_PREPARATION/HCP_DATA_PREP/output/100307/100307_MEG_3-Restin_source_LCMV_3d8mm.mat
$REPO_ROOT/DATA_PREPARATION/HCP_DATA_PREP/output/100307/100307_MEG_3-Restin_spatialFilter_LCMV_3d8mm.mat
$REPO_ROOT/DATA_PREPARATION/HCP_DATA_PREP/output/100307/100307_MEG_anatomy_leadfield_SVD_3d8mm.mat
$REPO_ROOT/DATA_PREPARATION/HCP_DATA_PREP/output/100307/100307_MEG_anatomy_leadfield_VEC_3d8mm.mat
```

16. Set working directory to `$REPO_ROOT/DATA_PREPARATION/HCP_DATA_PREP/05_RENAMING_TOOL/`,
17. Edit `$REPO_ROOT/DATA_PREPARATION/HCP_DATA_PREP/05_RENAMING_TOOL/RENAMING_TOOL.m` script to make sure it is set as below:

```
%% ==== USER SETTINGS: ============================================================
inputCfg.path          = '../output/'; % source directory where data to be renamed are
outputCfg.path         = '../output/000000_RENAMED/';

inputCfg.SubCodes      = {'100307','102816','105923','106521', ...
'108323','109123','111514','112920', ...
'113922','116524'}; % input subject codes


(verte ...)
```

```
(... continued from the previous page)

outputCfg.SubCodes     = {'Sub_1', 'Sub_2', 'Sub_3', 'Sub_4', ...
'Sub_5', 'Sub_6', 'Sub_7', 'Sub_8', ...
'Sub_9', 'Sub_10'}; % new subject codes

% sizes of two lists below should match
inputCfg.dataNames     = {  '_MEG_3-Restin_rmegpreproc_preprocessed', ...
'_MEG_3-Restin_rmegpreproc_preproc_covmat', ...
'_MEG_anatomy_leadfield_SVD_3d8mm', ...
'_MEG_anatomy_leadfield_VEC_3d8mm', ...
'_MEG_3-Restin_source_LCMV_3d8mm', ...
'_MEG_3-Restin_spatialFilter_LCMV_3d8mm'};
outputCfg.dataNames    = {  'data_clean_HCP_att2', ...
'data_clean_covmat_HCP_att2', ...
'lfg_SVD_HCP_att2', ...
'lfg_VEC_HCP_att2', ...
'source_LCMV_HCP_att2', ...
'flt_LCMV_HCP_att2'};

outputCfg.namesFormat  = 'name_subNum';   % use combination with 'name' (obligatory),
```

It is used to make copies of the new files, with file names and variables renamed.

18. Run `RENAMING_TOOL.m` script.
19. In the Command Window, you will be asked whether you agree with the list of planned copies:

```
Here is the list of planned copies:
{'../output/100307/100307_MEG_3-Restin_rmegpreproc_preprocessed.mat ->
../output/000000_RENAMED/Sub_1/data_clean_HCP_att2_1.mat'          }
{'../output/100307/100307_MEG_3-Restin_rmegpreproc_preproc_covmat.mat ->
../output/000000_RENAMED/Sub_1/data_clean_covmat_HCP_att2_1.mat'  }
{'../output/100307/100307_MEG_anatomy_leadfield_SVD_3d8mm.mat ->
../output/000000_RENAMED/Sub_1/lfg_SVD_HCP_att2_1.mat'                    }
{'../output/100307/100307_MEG_anatomy_leadfield_VEC_3d8mm.mat ->
../output/000000_RENAMED/Sub_1/lfg_VEC_HCP_att2_1.mat'                    }
{'../output/100307/100307_MEG_3-Restin_source_LCMV_3d8mm.mat ->
../output/000000_RENAMED/Sub_1/source_LCMV_HCP_att2_1.mat'                 }
{'../output/100307/100307_MEG_3-Restin_spatialFilter_LCMV_3d8mm.mat ->
../output/000000_RENAMED/Sub_1/flt_LCMV_HCP_att2_1.mat'            }

... [followed by 50 lines for the remaining nine subjects]

"Do you agree with it? y/n :"
```

20. Write `y` in the Command Window and press ENTER to accept. After a few moments copying and renaming process will be finished.

The output directory, `$REPO_ROOT/HCP_DATA_PREP/output/000000_RENAMED/`, is essential, as later, when setting configuration variables, `CFG.Global.veryFirstInputDataDir` should contain the absolute path to this directory!

Now, signal and spatial filter preparation is finished. The following new output subject files should appear (example for subject `100307`who became re-coded onto `Sub_1`):

```
$REPO_ROOT/DATA_PREPARATION/HCP_DATA_PREP/output/000000_RENAMED/Sub_1/data_clean_covmat_HCP_att2_1.mat
$REPO_ROOT/DATA_PREPARATION/HCP_DATA_PREP/output/000000_RENAMED/Sub_1/data_clean_HCP_att2_1.mat
$REPO_ROOT/DATA_PREPARATION/HCP_DATA_PREP/output/000000_RENAMED/Sub_1/flt_LCMV_HCP_att2_1.mat
$REPO_ROOT/DATA_PREPARATION/HCP_DATA_PREP/output/000000_RENAMED/Sub_1/lfg_SVD_HCP_att2_1.mat
$REPO_ROOT/DATA_PREPARATION/HCP_DATA_PREP/output/000000_RENAMED/Sub_1/lfg_VEC_HCP_att2_1.mat
$REPO_ROOT/DATA_PREPARATION/HCP_DATA_PREP/output/000000_RENAMED/Sub_1/source_LCMV_HCP_att2_1.mat
$REPO_ROOT/DATA_PREPARATION/HCP_DATA_PREP/output/000000_RENAMED/YYYY_MM_DD_HH-MM-SS_files_change.txt
$REPO_ROOT/DATA_PREPARATION/HCP_DATA_PREP/output/000000_RENAMED/YYYY_MM_DD_HH-MM-SS_subjCodes_change.txt
```

, where `YYYY` denotes year, `MM` - month, `DD` - day, `HH` - hour, `MM` - minute, `SS` - second of file creation.

Text files: `files_change` and `subjCodes_change` report the information about all the renames made, which can be helpful.

In this illustrative example, we use Desikan–Killiany parcellations. It is recommended that one will take advantage of the already precomputed atlases that can be found here:
`DATA_PREPARATION/PARCELLATIONS_PREP/DK_individual_atlases/output_precomputed`.

If one wants to compute them himself, please read the instructions provided in

`DATA_PREPARATION/PARCELLATIONS_PREP/DK_individual_atlases/Desikan_Killiany_preparation.m`

4.5.2 Configuring and launching: Spectral Fingerprinting (II), Analysis (III)

Here we will configure and run components II and III presented in Fig. 7.1, where certain *STAGES* of processing (abbreviated "S1", "S2", etc.) are outlined. In the next step, we will run STAGE 1 and 2 to obtain individual fingerprints. This will transform sensor signal segments into source-level power-spectra segments, which will form individual fingerprints once clustered. To cluster on a group-level, where spectral fingerprints can be defined, we need to run STAGE 3 to pool individual clusters together and then run STAGE 4 to determine the optimal number of clusters for each brain region separately. Then during STAGE 5, the clusters will be formed and thus spectral fingerprints. After that, we will be able to run the Network Analysis (STAGE 8) in order to see which ROIs are similar in terms of their fingerprints. To test how well we can identify ROIs relying on their fingerprints, we should run STAGE 6 and 7. However, we will limit due to the long processing time ourselves to group-level identification and run STAGE 6 only. The decision on running STAGE 7 is left to the user.

To demonstrate the possibility of parallel computations, we will configure the algorithm to run on two CPU cores. Running the obligatory part of the example (i.e. all stages but STAGE 7) on a machine equipped with a 64-bit Intel© Core™ i5-6200U 2.3 GHz processor, 16 GB RAM should take about 40 minutes when running in parallel on 2 CPU cores. Optional individual-level identification will take another 40 minutes to complete.

It is most convenient to use the ToFFi toolbox on Linux, however here we show how to run this example operates on the Windows or MacOS platform because for both platforms the procedure is the same.

INSTRUCTIONS (Configuration)

Configuration script `CONFIGURE.m` is already set up for illustrative example purposes by default. One needs only to correct variables containing paths:

1. Set MATLAB's working directory to `$REPO_ROOT`.
2. Open `$REPO_ROOT/CONFIGURE.m`.
3. Set `CFG.Global.rootDir = 'C:/johndoe/ToFFi_Toolbox-YYYYMMDD/' % absolute path!`
4. Set `CFG.Global.veryFirstInputDataDir = 'C:/johndoe/ToFFi_Toolbox-YYYYMMDD/DATA_PREPARATION/HCP_DATA_PREP/output/000000_RENAMED/' % absolute path (mentioned in the previous instruction box)!`
5. Set `CFG.Global.fieldtripPath = C:\johndoe\fieldtrip-20210816\' % absolute path!`
6. Run `CONFIGURE.m` making sure that MATLAB's working directory is set to the `$REPO_ROOT` directory, i.e. `C:\johndoe\ToFFi_Toolbox-YYYYMMDD\`.
7. When the computations are done, confirm that in the `$REPO_ROOT/STAGE_X/output/` directories there are additional `CFG.mat` output files which store configuration parameters.

Configuration - the rationale behind the choice of the parameters

Below we will explain some of the choices regarding variable values. Parameters that were not mentioned in the list above follow values typical for the Spectral Fingerprinting, as for [7].

GLOBAL parameters choice:

- `CFG.Global.goodSubjects = [1 2 3 4 5 6 7 8 9 10];`
  as we would like to use the first 10 subjects whose data we have just prepared

- `CFG.Global.goodROI = [6 25 54 67 70 89 102 105];`
  as those are the indices of the regions of interest in the `DK_indAtlas_*.mat` files (values of the `tissue` field corresponding to the `tissuelabel` field values)
- `CFG.Global.atlasType = 'individual';`
  as Desikan–Killiany atlases were prepared based on individual-subject `aparc+aseg.nii.gz` files
- `CFG.Global.atlasPath = 'DATA_PREPARATION/PARCELLATIONS_PREP/DK_individual_atlases/output_precomputed/';`
  as this is the directory where the prepared atlas files are stored
- `CFG.Global.maxNumQueuedJobsPerUser = 1;`
  as we are using a single computer to run this example
- `CFG.Global.maxNumSpmdWorkers = 2;`
  as we plan to use 2 CPU cores

  NOTE: Mind memory resources

  One can increase the number of used CPU cores (`G_maxNumSpmdWorkers` variable; refer to tables in 7.4.2 Configuration), but it should be done with care, as it is very easy to run out of memory!
- `CFG.(STAGE_NAME).MODE = 'parallel';`
  for stages 1, 2, 4, 6, 7 as their computational time will be reduced the most by using more than one CPU core
- `CFG.(STAGE_NAME).MODE = 'serial';`
  for stages 5 and 8
- `CFG.(STAGE_NAME).rngSeed = 2021;`
  for all the stages as a fixed seed will guarantee reproducibility of results

STAGE 1 parameters choice:

- `CFG.(STAGE_NAME).dummySignalMode = 'no';`
  as we want to use empirical sensor signals instead of synthetic data
- `CFG.(STAGE_NAME).frequenciesOfInterest = logspace(0, log10(40), 20);`
  as logarithmic distribution will optimize resolution for lower frequencies

STAGE 2 parameters choice:

- `CFG.(STAGE_NAME).doZeroShift = 0;`
  as we want to express normalized spectral power as a non-negative ratio for easier interpretation
- `CFG.(STAGE_NAME).numClustersMode = 'optimal';`
  as we allow each ROI to have a different number of spectral modes for each subject and prevent forming spurious clusters
- `CFG.(STAGE_NAME).clustering.(CFG.(STAGE_NAME).clusteringMethod).distanceMetric = 'cosine';`
  as cosine distance puts more emphasis on the shape of the fingerprints (peaks) than on absolute amplitude value, which is desired
- `CFG.(STAGE_NAME).NumClustEval_kList = [1 2 3 4 5];`
  to speed up the computations (from our observations, two spectral modes is sufficient most of the time)

STAGE 3 is sufficiently configured by default.

STAGE 4 parameters choice:

- `CFG.(STAGE_NAME).distanceMetric = 'cosine';`
  as cosine distance puts more emphasis on the shape of the fingerprints (peaks) than on absolute amplitude value, which is desired
- `CFG.(STAGE_NAME).kList = [1 2 3 4 5];`
  to speed up the computations (from our observations, two spectral modes is sufficient most of the time)

STAGE 5 parameters choice:

- `CFG.(STAGE_NAME).majoritySubjectsNum = 5;`
  as we want to see which spectral modes are *stable*, i.e. which modes are common to at least five subjects
- `CFG.(STAGE_NAME).numberOfClusters = 'optimal';`
  as we allow each ROI to have a different number of spectral modes for each subject and prevent forming spurious clusters
- `CFG.(STAGE_NAME).clustering.(CFG.(STAGE_NAME).clusteringMethod).distanceMetric = 'cosine'`
  as cosine distance puts more emphasis on the shape of the fingerprints (peaks) than on absolute amplitude value, which is desired

STAGE 6 parameters choice:

- `CFG.(STAGE_NAME).CV.nFolds = 10;`
  as for we have ten subjects, this effectively turns cross-validation into a leave-one-out validation, which clearly shows how efficient is the ROI recognition for the new subjects, based on previously trained models
- `CFG.(STAGE_NAME).nClustersSetting = 'fixed';` and `CFG.(STAGE_NAME).fixed_nClusters = 1;`
  to show that this setting is possible, and even considering only a single spectral mode, which is faster than the 'optimal' setting, allows for high identification accuracies
- `CFG.(STAGE_NAME).clustering.(CFG.(STAGE_NAME).clusteringMethod).distanceMetric = 'cosine'`
  as cosine distance puts more emphasis on the shape of the fingerprints (peaks) than on absolute amplitude value, which is desired
- `CFG.(STAGE_NAME).majoritySubjectNum = 5;`
  as, during group-identification, we want to take into account spectral fingerprints common to our group of participants

STAGE 7 parameters choice:

- `CFG.(STAGE_NAME).CV.nFolds = 5;`
  as not only 10-fold CV is possible, but other data divisions are supported as well
- `CFG.(STAGE_NAME).nClustersSetting = 'optimal';`
  as here we would like to calculate optimally adjusted individual fingerprints
- `CFG.(STAGE_NAME).clustering.(CFG.(STAGE_NAME).clusteringMethod).distanceMetric = 'cosine';`
  as cosine distance puts more emphasis on the shape of the fingerprints (peaks) than on absolute amplitude value, which is desired
- CFG.(STAGE_NAME).NumClustEval_kList = [1 2 3 4 5];
  to speed up the computations (from our observations, two spectral modes is sufficient most of the time)

STAGE 8 parameters choice:

- `CFG.(STAGE_NAME).nSimilarityClusters = 4;`
  as we would like to see if our 8 ROIs will be matched into four homologous pairs
- `CFG.(STAGE_NAME).majoritySubjectsNum = 5;`
  as we want to take into account spectral fingerprints common to our group of participants

NOTE: Configuration - DIY

Optionally, if one wants to enter all parameters himself or change some of them, reading 7.4.2 Configuration and 6 METHODS sections is highly recommended. They cover:

- what parameters and significant structures are used,
- the proper syntax,
- how the data are transformed.

Launching configured Spectral Fingerprinting

INSTRUCTIONS (Launching Spectral Fingerprinting)

To launch just configured Spectral Fingerprinting:

1. Set the working directory to the `$REPO_ROOT`.
2. Open `$REPO_ROOT/RUN_SELECTED.m` file in MATLAB.
3. Set `stages = [1 2 3 4 5 6 8]`.
4. Optionally, add 7 to the list (note that this will approximately double the computation time).
5. Run that script, making sure that MATLAB's working directory is set to the `$REPO_ROOT` directory, i.e. `C:\johndoe\ToFFi_Toolbox-YYYYMMDD\`.

Stages 6–8 implement identification and network analysis routines which results are also discussed in the 4.6 Results analysis section. To learn how they work, see 6.3.4 Analysis (STAGES 6-8) section.

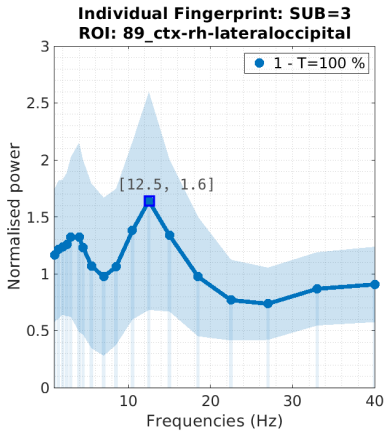
When the computations are done, one can confirm that in the `$REPO_ROOT/STAGE_X/output/` directories there are additional `.mat` output files with the results of a particular stage (there may be more files, but below we listed obligatory ones):

```
$REPO_ROOT/STAGE_1/output/CFG.mat
$REPO_ROOT/STAGE_1/output/Sub_X/normalizedSourcePower_X.mat    (X = 1, 2, ..., 10)

$REPO_ROOT/STAGE_2/output/CFG.mat
$REPO_ROOT/STAGE_2/output/Sub_X/individualFingerprint_X.mat    (X = 1, 2, ..., 10)
$REPO_ROOT/STAGE_2/output/Sub_X/singleSubjectPowerData_X.mat    (X = 1, 2, ..., 10)

$REPO_ROOT/STAGE_3/output/CFG.mat
$REPO_ROOT/STAGE_3/output/pooledClustersInAllROI.mat

$REPO_ROOT/STAGE_4/output/CFG.mat
$REPO_ROOT/STAGE_4/output/clusteringEvaluationAllROI.mat

$REPO_ROOT/STAGE_5/output/CFG.mat
$REPO_ROOT/STAGE_5/output/singleRoiSpectralFingerprint_iROI102.mat
$REPO_ROOT/STAGE_5/output/singleRoiSpectralFingerprint_iROI105.mat
$REPO_ROOT/STAGE_5/output/singleRoiSpectralFingerprint_iROI25.mat
$REPO_ROOT/STAGE_5/output/singleRoiSpectralFingerprint_iROI54.mat
$REPO_ROOT/STAGE_5/output/singleRoiSpectralFingerprint_iROI67.mat
$REPO_ROOT/STAGE_5/output/singleRoiSpectralFingerprint_iROI6.mat
$REPO_ROOT/STAGE_5/output/singleRoiSpectralFingerprint_iROI70.mat
$REPO_ROOT/STAGE_5/output/singleRoiSpectralFingerprint_iROI89.mat

$REPO_ROOT/STAGE_6/output/CFG.mat
$REPO_ROOT/STAGE_6/output/classificationResult.mat
$REPO_ROOT/STAGE_6/output/NLOGL.mat
$REPO_ROOT/STAGE_6/output/roiModels_Rep_1_Fold_X.mat    (X = 1, 2, ..., 10)

$REPO_ROOT/STAGE_8/output/CFG.mat
$REPO_ROOT/STAGE_8/output/networkAnalysisResult.mat

(optionally)
$REPO_ROOT/STAGE_7/output/CFG.mat
$REPO_ROOT/STAGE_7/output/indCV_iSubX.mat    (X = 1, 2, ..., 10)
```

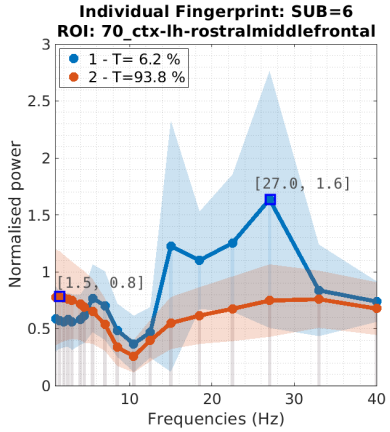
4.6 Results analysis

Now we will run visualizations to examine the results.

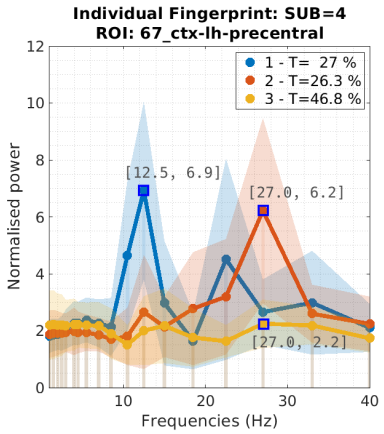
Individual Fingerprints



(a) Right hemisphere occipital fingerprint for subject no. 3. Single spectral mode with a clear dominance of the alpha peak visible.



(b) A fingerprint of the anterior middle frontal gyrus for subject no. 6. Two spectral modes, one of which is dominant (orange) and the other is supportive (blue). See the text for the explanation.



(c) A fingerprint of the left precentral gyrus for subject no. 4. Multiple spectral modes peaking at different frequencies with no clear dominance visible.

Figure 4.5: Resting-state Individual Fingerprints for Desikan–Killiany atlas in the 1–40 Hz frequency interval. Legends show the corresponding duration of each spectral mode (i.e., the percentage of trials in which each spectrum was present on average during recording). The frequency axis was configured to be logarithmic in order to optimize the lower frequencies resolution. Y-axis depicts the power normalized in relation to the average spectrum of the whole brain. Shaded regions depict the standard deviation (1σ) of the corresponding spectral mode. For i-th of total F frequencies of interest, standard deviation was estimated as $\sqrt{\Sigma_{i,i}}$, where $\Sigma_{i,i}$ is the i-th diagonal entry of the covariance matrix of the Gaussian Mixture Model component corresponding to the given spectral mode. Different types of fingerprints can occur: a) fingerprints with single spectral mode, b) fingerprints with dominant and supportive spectral modes, and c) fingerprints with more than two modes, switching between one another during the recording. Regions tend to differ in terms of peaking frequencies.

INSTRUCTIONS (Plotting Individual Fingerprints)

1. Set MATLAB's working directory to `$REPO_ROOT/PRESENTATION/fingerprints/`.
2. Edit script `$REPO_ROOT/PRESENTATION/fingerprints/plotIndividualFingerprint.m` entering the following settings:

```
cfg = [];
cfg.SRC_DIR      = '../../'
cfg.SUB_LIST     = 1:10
cfg.ROI_LIST     = [];       % this will select all the ROIs we have analyzed
cfg.FIG_VISIBLE  = 'off';    % to prevent slowing down by displaying 80 figures
cfg.FIG_SAVE     = 'y';
cfg.DOT_SIZE     = 40;
cfg.STD_OPACITY  = 0.2;      % adjusts opacity of the std shaded error bars
cfg.LINE_WIDTH   = 5;
cfg.LABEL_PEAKS  = 1;        % to label frequency peak of each spectral mode
```

3. Run `plotIndividualFingerprint.m`.

After running this whole script, there will be no pictures displayed, as `cfg.FIG_VISIBLE = 'off'`, however, one can see a new directory named e.g. `$REPO_ROOT/PRESENTATION/fingerprints/output_2021-07-28_21-37-50`, containing generated images of individual fingerprints for all the subjects and ROIs (Fig. 4.5).

It can be seen that different regions tend to peak at different frequencies, e.g. occipital areas mainly in the alpha band and frontal areas in the gamma band. Regions also differ in terms of the number of spectral modes. Mostly two modes describe each region, but one of the modes usually lasts much longer than the other one (have a much higher percentage of time), which can be thought of as *a dominant mode* and a *supportive mode* pair. There are some notable exceptions, e.g. for frontal areas, there is often a single spectral mode, or for subject no. 4 and ROI 67, there are three modes with no clear dominance among them. Patterns are usually consistent between subjects, however, some individual specificity can be clearly seen.

Spectral Fingerprints

Now it will be explored how well these patterns are preserved after clustering on the group level.

INSTRUCTIONS (Plotting Spectral Fingerprints)

1. Set MATLAB's working directory to `$REPO_ROOT/PRESENTATION/fingerprints/`.
2. Open `$REPO_ROOT/PRESENTATION/fingerprints/plotSpectralFingerprint.m` script.
3. Enter the following settings:

```
cfg = [];
cfg.SRC_DIR     = '../../'
cfg.ROI_LIST    = [];       % this will select all the ROIs we have analyzed
cfg.FIG_VISIBLE = 'on';     % 8 figures in total, so there is no problem displaying them
cfg.FIG_SAVE    = 'y';
cfg.DOT_SIZE    = 40;
cfg.STD_OPACITY = 0.2;      % adjusts opacity of the std shaded error bars
cfg.LINE_WIDTH  = 5;
cfg.LABEL_PEAKS = 1;        % to label frequency peak of each spectral mode
```

4. Run `plotSpectralFingerprint.m`.

After running that script, eight images will be displayed, one per ROI (Fig. 4.6). One can also see a new directory named e.g. `$REPO_ROOT/PRESENTATION/fingerprints/output_2021-07-28_21-41-00`, containing generated images.

Similarly to the individual fingerprints, we can say that particular spectral mode is:

- *stable* when at least five subjects represent it, or *unstable* otherwise;
- *dominant* when it has a high percentage (duration), or *supportive* when it has a low percentage.

NOTE: Spectral Fingerprints duration

For spectral fingerprints, spectral mode's time (Fig. 4.6) is computed as a sum of points' duration (i.e. spectral modes from individual fingerprints) divided by the number of unique subjects that these points belong to. This procedure was inspired by the original publication [7] and can yield cumulative percentages of above 100%, as it was stated there.

Observations:

- Fingerprints have a similar shape among homologue areas, as it was for individual fingerprints, suggesting that homologue areas may have similar functions.
- Precentral areas tend to peak in the beta band. Occipital areas peak in the alpha band. Frontal areas peak mainly in the theta and gamma band. Thalamus has many spectral modes which peak at different frequencies.
- Every ROI has a single *stable mode* (i.e. one that is represented by at least five subjects - filled dots in legends) which happens to be a *dominant mode* as well (having the largest duration percentage). Other modes represent individual variability that happens to be similar across few subjects, rendering them as less important, a.k.a. *unstable*.

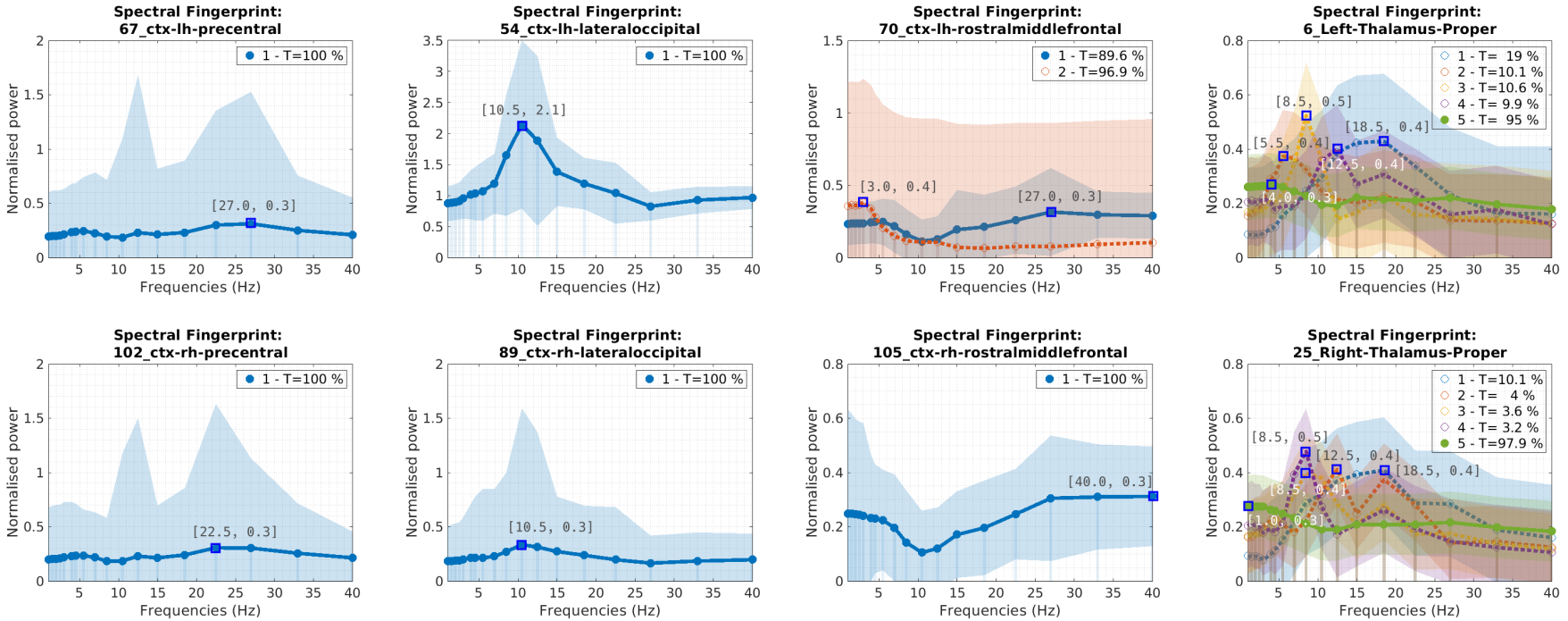


Figure 4.6: (in color) Resting-state spectral fingerprints for Desikan-Killiany atlas in the 1–40 Hz frequency interval. Each column shows two homologue brain areas. Legends show the corresponding duration of each spectral mode (i.e., the percentage of trials in which each spectrum was present on average during recording) and whether the mode was present for at least five subjects (filled dot) or not (empty dot). The frequency axis was configured to be logarithmic in order to optimize the lower frequencies resolution. Y-axis depicts the power normalized in relation to the average spectrum of the whole brain. Shaded regions depict the standard deviation (1σ) of the corresponding spectral mode. For i-th of total F frequencies of interest, standard deviation was estimated as $\sqrt{\Sigma_{i,i}}$, where $\Sigma_{i,i}$ is the i-th diagonal entry of the covariance matrix of the Gaussian Mixture Model component corresponding to the given spectral mode. Standard deviations have relatively large values due to the small number of subjects used in the illustrative example. Homologue areas have very similar fingerprints.

Similarity of the Spectral Fingerprints (Network Analysis)

The next step will be to match calculated spectral fingerprints into pairs using Network Analysis scripts.

INSTRUCTIONS (Plotting similarity tree and 3D-brain)

1. Set MATLAB's working directory to `$REPO_ROOT/PRESENTATION/network_analysis/`.
2. Open the following script `$REPO_ROOT/PRESENTATION/network_analysis/networksPlot.m`
3. Enter the following settings:

```
cfg = [];
cfg.srcDir      = '../../'
cfg.treeLevel   = 4;
cfg.labelMode   = 'name';
cfg.figSave     = 'y';
```

4. Run `networksPlot.m`.

After a while, there will be four figures displayed:

- binary tree,
- 3D-view of both brain hemispheres,
- 3D-view of the left brain hemisphere,
- 3D-view of the right brain hemisphere.

To see the inner parts of the brain (Fig. 4.7), one can rotate hemispheres models using a figure's GUI. One can also see a new directory named e.g. `$REPO_ROOT/PRESENTATION/network_analysis/output_2021-07-28_21-47-10`, containing generated images.

As expected, similar fingerprints were matched together.

Group-level identification of brain areas

As will be shown next, this regional similarity, however, might pose a problem with the region identification. Similar fingerprints can be easily confused together.

INSTRUCTIONS (Group-level identification results)

1. Set MATLAB's working directory to `$REPO_ROOT/PRESENTATION/identification/`.
2. Open `$REPO_ROOT/PRESENTATION/identification/groupIdentificationPlot.m` script.
3. Enter the following settings:

```
cfg = [];
cfg.srcDir      = '../../';
cfg.measure     = 'acc';
cfg.meanAcross  = 'folds';   % as there were no additional CV repetitions set
cfg.labelMode   = 'name';
cfg.barColor    = [0 0 1];
cfg.std         = 'y';
% writing ROIs explicitly arranges them in preferred order in plots
cfg.roi         = [6, 25, 54, 89, 67, 102, 70, 105];
cfg.figVisible  = 'on';
cfg.figSave     = 'y';
cfg.fullScreen  = 'n';
```

4. Run `groupIdentificationPlot.m`.

Running this script will display a bar chart showing the identification accuracy of the given ROI (Fig. 4.8a). As predicted, one region of the homologous pair has very high accuracy, whereas the other has very low accuracy.

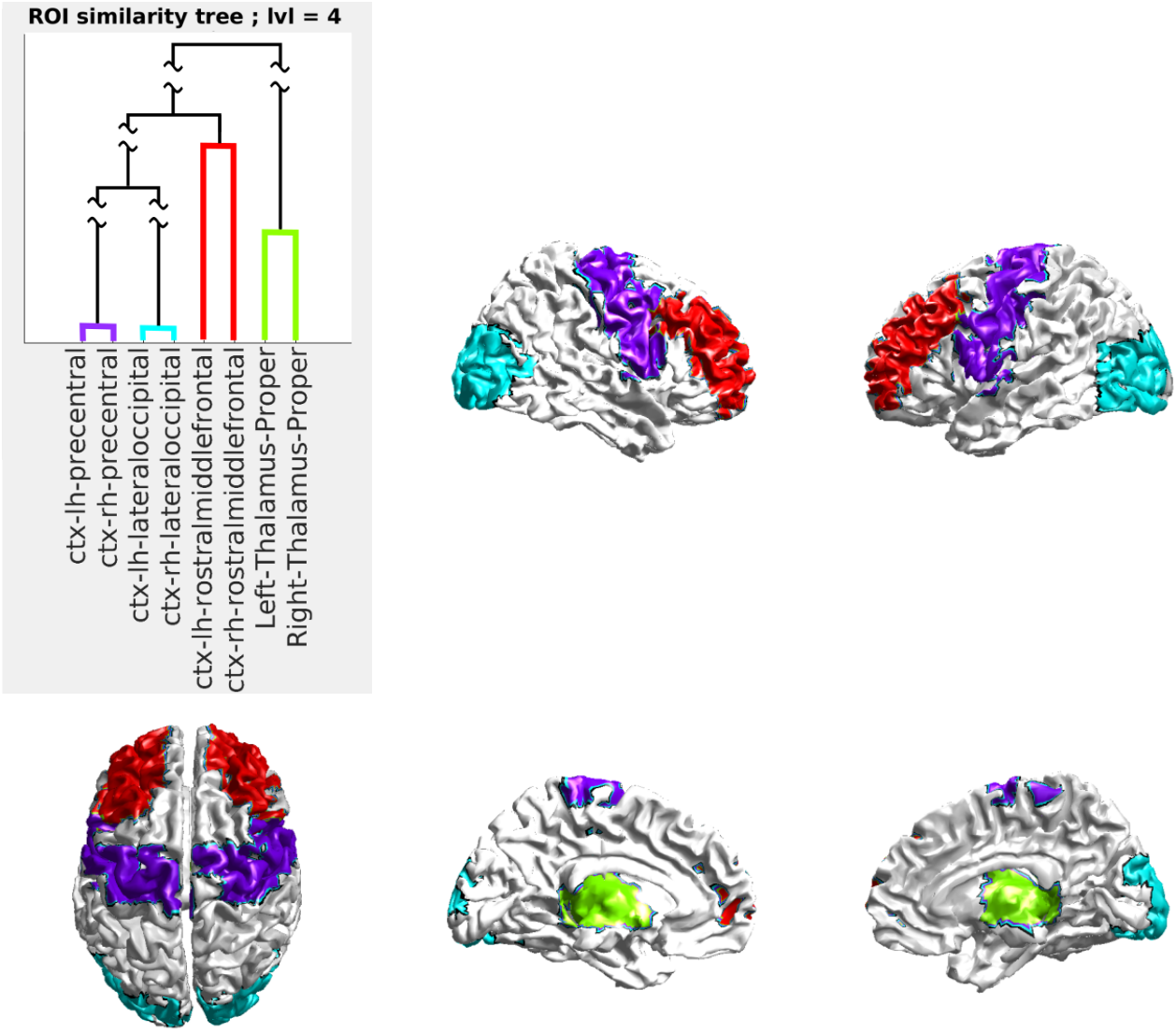


Figure 4.7: Result of the hierarchical agglomerative clustering of the spectral fingerprints presented in Fig. 4.6 (Network Analysis script). Homologue areas were automatically matched together according to the similarity of their fingerprints. The similarity tree has disproportionately long branches that were broken for clarity (waved lines).

(... continued from the page 21)

To examine the situation more closely:

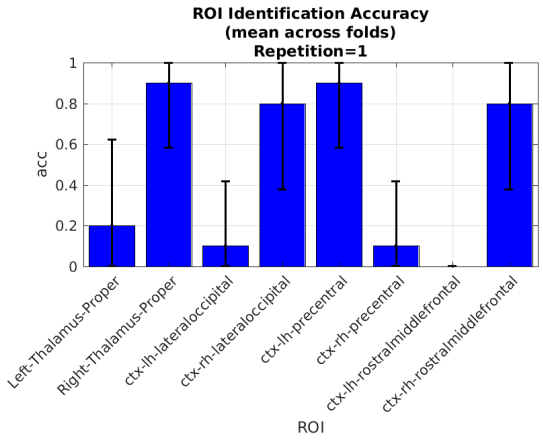
1. Set MATLAB's working directory to `$REPO_ROOT/PRESENTATION/identification/`.
2. Open `$REPO_ROOT/PRESENTATION/identification/groupHitMatrix.m` script.
3. Set the following parameters:

```
cfg = [];
cfg.srcDir      = '../../';
cfg.labelMode   = 'name';
% writing ROIs explicitly arranges them in preferred order in plots
cfg.roi         = [6, 25, 54, 89, 67, 102, 70, 105];
cfg.repRange    = 1:1;
cfg.foldRange   = 1:10;
cfg.percent     = 0;
cfg.figVisible  = 'on';
cfg.figSave     = 'y';
cfg.fontSize    = 14;
```

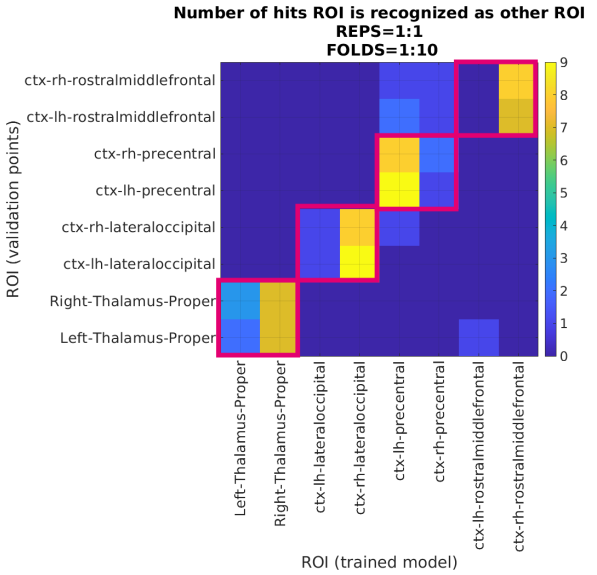
4. Run it `groupHitMatrix.m`.

If there were any figures opened in MATLAB, they now disappeared, and a figure containing a hit matrix is shown. One can also see two new directories named e.g. `$REPO_ROOT/PRESENTATION/identification/output_2021-07-28_21-56-33`, containing generated images.

Hit matrix allowed to confirm that homologue areas contributed mostly to the confusion, but also other ROIs with similar fingerprints, as few hits were outside the 2x2 diagonal submatrices (Fig. 4.8b).



(a)



(b)

Figure 4.8: Group-level identification accuracy: a) bar plot showing the average identification accuracy across cross-validation iterations (leave-one-out), b) confusion matrix showing in each row a distribution of "votes" for each ROI. Each ROI was tested ten times (model trained on nine subjects versus one validation subject). For ideal identification, this matrix would have a value of 10 for the diagonal elements and zeros elsewhere. Confusion happens mostly between homologue areas (2x2 red boxes). Left hemisphere ROIs are recognized as the right hemisphere homologue areas.

(Optional) Individual-level identification of brain areas

To check the results of the individual-level identification:

INSTRUCTIONS (Individual-level identification results)

1. Set MATLAB's working directory to `$REPO_ROOT/PRESENTATION/identification/`.
2. Open `$REPO_ROOT/PRESENTATION/identification/individualIdentificationPlot.m` script.
3. Enter the following settings:

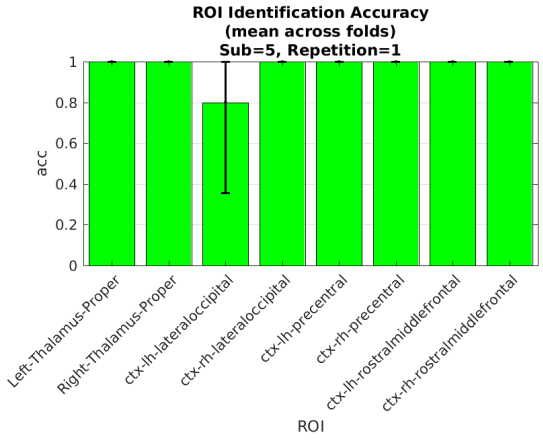
```
cfg = [];
cfg.srcDir      = '../../';
cfg.measure     = 'acc';
cfg.meanAcross  = 'folds'; % as there were no additional CV repetitions set
cfg.labelMode   = 'name';
cfg.barColor    = [0 1 0];
cfg.std         = 'y';
% writing ROIs explicitly arranges them in preferred order in plots
cfg.roi         = [6, 25, 54, 89, 67, 102, 70, 105];
cfg.sub         = 1:10;
cfg.figVisible  = 'on';
cfg.figSave     = 'y';
cfg.fullScreen  = 'n';
```

4. Run `individualIdentificationPlot.m`.
5. Couple of figures will appear (they are saved in a new output directory, e.g. `$REPO_ROOT/PRESENTATION/identification/output_2021-07-28_22-01-07`).
6. Set MATLAB's working directory to `$REPO_ROOT/PRESENTATION/identification/`.
7. Open `$REPO_ROOT/PRESENTATION/identification/indHitMatrix.m`
8. Enter the following settings:

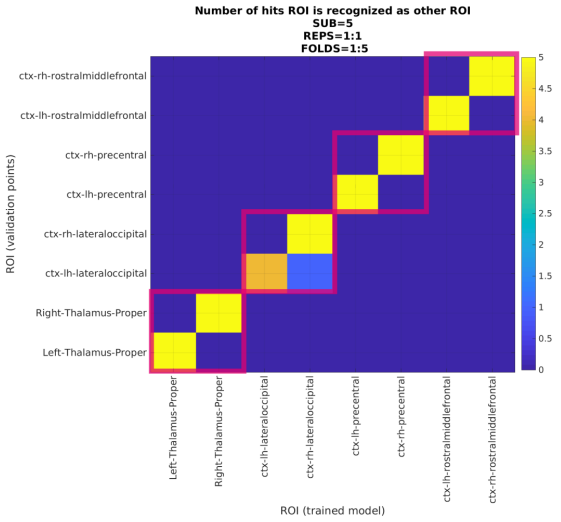
```
cfg = [];
cfg.srcDir      = '../../'
cfg.labelMode   = 'name';   % 'number', 'name'
cfg.sub         = 1:10;
% writing ROIs explicitly arranges them in preferred order in plots
cfg.roi         = [6, 25, 54, 89, 67, 102, 70, 105];
cfg.repRange    = 1:1;
cfg.foldRange   = 1:5;
cfg.percent     = 0;
cfg.figVisible  = 'on';
cfg.figSave     = 'y';
cfg.fontSize    = 14;
```

9. Run `indHitMatrix.m`.
10. Previous figures will disappear and a couple of new figures will appear (they are saved in a new output directory, e.g. `$REPO_ROOT/PRESENTATION/identification/output_2021-07-28_22-12-21`)

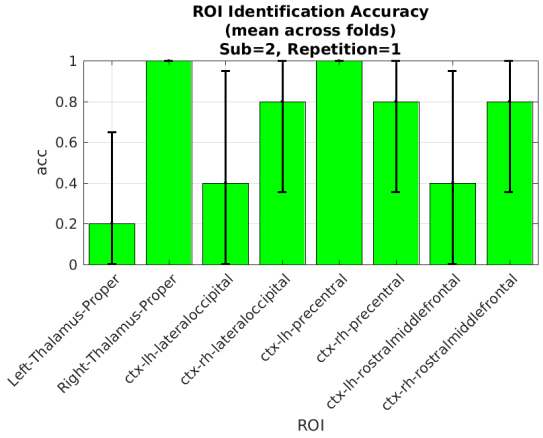
Similarly to group-level identification, one region of the homologous pair has very high accuracy, whereas the other has slightly lower accuracy (Fig. 4.9a, Fig. 4.9c). Again, hit matrices allowed to confirm that homologue areas solely contributed to the confusion, as there were no hits outside the 2x2 diagonal submatrices (Fig. 4.9b, Fig. 4.9d).



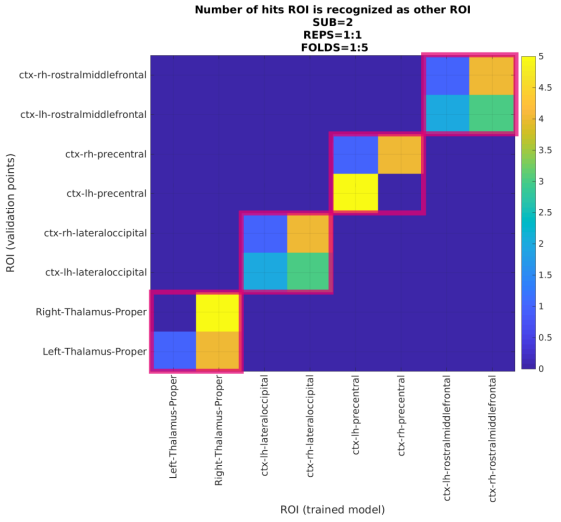
(a)



(b)



(c)



(d)

Figure 4.9: Individual-level identification accuracy. Best results were achieved for subject no. 5 (a,b), and worst results for subject no. 2 (c,d). Bar plots a), c) show the average identification accuracy across 5-fold cross-validation iterations; Confusion matrices b), d) show a distribution of "votes" for each ROI for a given subject. Each ROI was tested five times (as 5-fold cross-validation was set). For ideal identification, this matrix would have a value of 5 for the diagonal elements and zeros elsewhere. Notably, the confusion happens only between homologue areas (2x2 red boxes). Left hemisphere ROIs are recognized as the right hemisphere homologue areas.

4.7 Conclusions

To summarize, the Toolbox for Frequency-based Fingerprinting (ToFFi) allowed the discovery of common spectral patterns in the broadly used resting-state MEG multichannel recordings (The Human Connectome dataset) using a two-level clustering procedure with an optimally adjusted number of clusters. The proposed method discriminated between different modes of operation for a range of brain areas defined in the atlas of choice. Dominant and supportive oscillation profiles were recognized and separated, both for the individual- and group-level. Functional similarity between homologue areas was confirmed using hierarchical clustering analysis. Identification accuracy remained very high for the *individual fingerprints*. Recognition of the brain areas based on the *spectral fingerprints* turned out to be challenging among homologue areas due to the aforementioned functional similarity, yet remain still informative.

The toolbox's operation can be adjusted to solve different problems thanks to the versatile configuration. Parallel computations support using multiple machines helps to analyze bigger data sets in a fraction of time, compared to the serial calculations. The possibility for further development of the toolbox remains open, as this software is given to the community as an open-source package.

The next part of this document is meant to provide an in-depth understanding of the toolbox operation and help users design their own protocols for using this software.

We believe it is worthwhile to take advantage of our toolbox and lead to some exciting discoveries.

Good luck,

Michał Konrad Komorowski, et al.

Part II

Documentation

This part contains more information on the toolbox architecture, what data can be used with the toolbox, how to configure and operate the toolbox, and what methods are used to transform the data across the stages.

5 INPUT DATA

The main component of the toolbox is the Spectral Fingerprinting (II in Fig. 7.1), which demands specific data on its input. Here we describe the requirements of the data to be met. Should one wants to use particular data, either recorded in his laboratory or already available in one of the many online repositories, the data need to be adjusted to follow the design of variables shown below. Exemplary data adaptation pipeline was implemented as the DATA PREPARATION scripts (show in the figure as well), which do the preprocessing of The Human Connectome MEG Dataset [19] to meet all the requirements.

5.1 Obligatory data structures

The data needed to run all the stages of the Spectral Fingerprinting are:

- **multichannel signal** (EEG, MEG, iEEG, NIRS, etc.) matrices (one array per subject) divided into segments of equal duration,
- **spatial filter** matrices (one matrix per subject),
- **brain atlas** for brain regions definition (single group-level atlas or a set of individual-subject atlases),
- **common source grid** (a.k.a. template grid) to enable group-level clustering and comparisons (single grid).

These objects need to be implemented as the following MATLAB variables:

- `data` - a structure containing the following obligatory fields:
  - *trial* - cell array containing 2D-arrays (sensors x time frames). Each array stores signal values for a single epoch (a.k.a. *segment*) of the multichannel signal that is possibly free from artifacts. Segments should be of equal length. Each row of an array represents signal values for a different sensor. It is allowed that some of the segments are missing, e.g. were rejected during artifact removal routines. Ideally, segments shall concatenate up to a continuous, uninterrupted signal, but this is rarely achievable in practice.
  - *time* - cell array containing vectors of time axis values. Each vector stores a time axis for a single epoch (a.k.a. *segment*). The length of each vector should match the length of its corresponding signal array in the `trial` cell array.
  - *label* - cell containing names of the sensors (e.g. EEG electrodes, MEG squids).
  - *fsample* - positive double; sampling frequency in Hz.

  Example (294 segments, 509 samples per segment, 241 MEG channels):

```
>> data =
struct with fields:

fsample: 508.6275
trial: {1×294 cell}
time: {1×294 cell}
label: {241×1 cell}

>> data.trial

(verte ...)
```

```
(... continued from the previous page)

ans =

1×294 cell array

Columns 1 through 9

{241×509 double}    {241×509 double}    {241×509 double} ...

>> data.time

ans =

1×294 cell array

Columns 1 through 10

{1×509 double}    {1×509 double}    {1×509 double}

>> data.label

ans =

241×1 cell array

{'A22' }
{'A104'}
{'A241'}

...
```

- `spatialFilter` - 2D-array containing the spatial filter coefficients (e.g. LCMV) used during beamforming. The number of rows is equal to the number of sources to be estimated and, the number of columns is equal to the number of sensors (see Eq. 4.2).

  Example (5798 sources, 241 channels as above):

```
>> whos
  Name                  Size                Bytes  Class     Attributes

  spatialFilter      5798x241            11178544  double

>> spatialFilter(1:5, 1:5)

ans =

1.0e-08 *

0.0128    0.1253   -0.4061    0.1013   -0.1722
-0.0531    0.0167    0.1497   -0.0161    0.1230
0.2291    0.2997   -0.3320    0.1846   -0.2533
0.1917    0.2622   -0.4020    0.1926   -0.2697
0.0375    0.1647   -0.4799    0.1455   -0.2073
```

- `sourcemodel` - (a.k.a. template grid) structure that represents a 3D-grid of locations where the sources can be defined. Each subject can have a different mapping between possible locations defined in this template and

brain regions. These mappings can be expressed in the "tissue" field (thus implying the use of individual atlases) or one can assume the same mapping for all subjects (via single group-level atlas). It should contain the following fields:

- *xgrid* - double; an array of possible x-coordinates of brain voxels expressed in millimeters, centimeters, or meters.
- *ygrid* - double; an array of possible y-coordinates of brain voxels expressed in the same units as *xgrid*.
- *zgrid* - double; an array of possible z-coordinates of brain voxels expressed in the same units as *xgrid*.
- *dim* - double 1-D array of positive integers; the size of the 3D volume space expressed in voxels.
- *pos* - double; 3-vector of number voxels in each dimension. A total number of rows should equal the total number of voxels, i.e. *dim(1)*dim(2)*dim(3)*. It should count both voxels inside and outside the brain. Possible coordinates should be a combination of coordinate values from *xgrid*, *ygrid* and *zgrid* fields.
- *inside* - double 1-D array that lists indices of voxels that lie inside the brain. For these voxels, source reconstruction will be performed.
- *outside* - double; 1-D array that lists indices of voxels that lie outside the brain. These voxels will be omitted during the source reconstruction.

Example (5798 sources inside the brain):

```
>> sourcemodel =

struct with fields:

xgrid: [-7.6000 -6.8000 -6 -5.2000 ... 5.2000 6 6.8000 7.6000]
ygrid: [-11.2000 -10.4000 -9.6000 -8.8000 ... 5.6000 6.4000 7.2000 8]
zgrid: [-7.2000 -6.4000 -5.6000 -4.8000 ... 7.2000 8.0000 8.8000 9.6000]
dim: [20 25 22]
pos: [11000×3 double]
inside: [1×5798 double]
outside: [1×5202 double]

>> sourcemodel.pos(1:10,:)

ans =

-7.6000  -11.2000   -7.2000
-6.8000  -11.2000   -7.2000
-6.0000  -11.2000   -7.2000
-5.2000  -11.2000   -7.2000
-4.4000  -11.2000   -7.2000
-3.6000  -11.2000   -7.2000
-2.8000  -11.2000   -7.2000
-2.0000  -11.2000   -7.2000
-1.2000  -11.2000   -7.2000
-0.4000  -11.2000   -7.2000

>> sourcemodel.inside

ans =

Columns 1 through 15

114         127         131         132         133         134 ...

(verte ...)
```

```
(... continued from the previous page)

>> sourcemodel.outside

ans =

Columns 1 through 15

1           2           3           4           5           6 ...
```

- `sourceAtlas` - structure storing group atlas or individual subjects' brain atlases. It can be created as an interpolation of an atlas structure with different dimensions onto the source model defined above (check `$REPO_ROOT/globalFunctionsScripts/prepareAalAtlas.m` as an example). Its dimensions should match the template grid (`sourcemodel`). It should contain the following obligatory fields:
  - *dim* - double 1-D array of positive integers; the size of the 3D volume space expressed in voxels. It should match the dimensions of the `sourcemodel.dim`.
  - *transform* - double 2-D array containing affine transformation matrix for mapping the voxel coordinates to head coordinate system.
  - *unit* - string; the units in which the coordinate system is expressed. The coordinates stored in the `sourcemodel` should be expressed in the same units as the `sourceAtlas.unit`.
  - *tissue* - double 3-D array with integer values from 1 to N (the value 0 means unknown). Values are related to the ROI labels stored in *tissuelabel* fields. The size of each dimension is stored in the *dim* field. It stores a ROI label for each brain voxel. It should match the dimensions of the template grid (`sourcemodel`).
  - *tissuelabel* - cell of strings containing name of each ROI.

  Example (group-level atlas with 100 ROIs):

```
>> sourceAtlas

sourceAtlas =

struct with fields:

dim: [20 25 22]
transform: [4×4 double]
unit: 'cm'
tissue: [20×25×22 double]
tissuelabel: {1×100 cell}

>> sourceAtlas.transform

ans =

0.8000    0.0000   -0.0000   -8.4000
-0.0000    0.8000   -0.0000  -12.0000
-0.0000    0.0000    0.8000   -8.0000
0.0000    0.0000   -0.0000    1.0000

>> unique(sourceAtlas.tissue)'


(verte ...)
```

```
(... continued from the previous page)

ans =

Columns 1 through 31

0     1     2     3     4  ...

Columns 94 through 101

93    94    95    96    97    98    99   100

>> sourceAtlas.tissuelabel

ans =

1×100 cell array

Columns 1 through 11

{'VIS_LH_1'}    {'VIS_LH_2'}    {'VIS_LH_3'}    {'VIS_LH_4'} ...

Columns 99 through 100

{'DMN_RH_pCunPCC_1'}    {'DMN_RH_pCunPCC_2'}
```

5.2 Brief description of the HCP dataset

NOTE: HCP S1200 Release Reference Manual

For detailed description consult WU-Minn HCP 1200 Subjects Data Release Reference Manual and HCP Wiki.

The HCP aimed to study and freely share data from 1200 young adult (ages 22-35) subjects. A total of 95 subjects participated in the MEG resting-state acquisition, and 89 subjects have a complete set of files allowing to prepare for the Spectral Fingerprinting (see 5.3 Summary list of the needed HCP data files) - in total 48 males and 41 females, age range 22-35 years, mean 29 +/- 4 years.

MEG multichannel signal. Each subject was laying in the supine position and was scanned on a whole head MAGNES 3600 WH (4-D Neuroimaging) system housed in a magnetically shielded room. For the resting state acquisition, three runs (with short 1-2 min breaks between runs for the equipment check) were carried out, resulting in approximately 3 mins of clean signal per run after artifact rejection. The MEG system included 248-magnetometer channels, recording at 2034.5101 Hz sampling rate, further reduced to 508.6275 Hz for computational speedup. The raw data were segmented into 2-second pieces. Bad channels and segments were removed. ICA components that have been classified as artifacts were projected out of the data.

Structural MRI T1 scans. Structural MRI T1 scans were recorded on a customized Siemens 3T "Connectome Skyra" housed at Washington University in St. Louis, using a standard 32-channel Siemens receive head coil and a "body" transmission coil using 3D MPRAGE protocol with the following parameters: TR=2400 ms, TE=2.14 ms, TI=1000 ms, 8 deg flip angle, 224x224 FOV, 0.7 mm isotropic voxel size, 210 Hz/Px, iPAT=2, acquisition time: 7 min 40 seconds. Quality control checks and procedures are documented in [32]. Acquired scans were processed in Freesurfer [33] in order to obtain brain parcellation and segmentation files, for example, Desikan-Killiany atlas (`aparc+aseg.nii.gz`).

Coregistration. A Polhemus FASTRAK-III was used for spatial digitization of three anatomical landmarks (nasion and L/R periauricular points), five localizer coils, and subjects' head shape (about 2400 points). This resulted in precise coregistration with individual structural MRI scan, which served for generating the volume conduction model (headmodel).

Source models. The 3-dimensional grids (source models) with a regular spacing of the dipoles were defined in normalized MNI space. For each participant, the template grid has been warped to individual head space in order to express the dipoles' coordinates in the head coordinate system used for sources reconstruction. This warp was a non-linear one and was based on the inverse of the non-linear volumetric spatial normalization from individual head space to normalized MNI space.

5.3 Summary list of the needed HCP data files

If you want to use the HCP dataset, here is the summary list of the data files you need to have in order to run DATA PREPARATION:

- Subject-specific files (example for '100307' subject):
  - `MEG/Restin/100307_MEG_3-Restin_rmegpreproc.mat`
  - `MEG/anatomy/100307_MEG_anatomy_sourcemodel_3d8mm.mat`
  - `MEG/anatomy/100307_MEG_anatomy_headmodel.mat`
  - `MNINonLinear/aparc+aseg.nii.gz` (if you want to prepare and work with the Desikan-Killiany atlas)

5.4 How to download the HCP data?

1. Go to the page `https://db.humanconnectome.org`,
2. Create a new account and log in,
3. Find the "WU-Minn HCP Data - 1200 Subjects",
4. Click on the button "Data Available on Amazon S3",
5. Click "Create my AWS Credentials",
6. Store the credentials in a safe place,
7. Download and install an application that will allow you to access the HCP data container on AWS, e.g. Amazon S3 Browser or CyberDuck, and enter the credentials.
8. Browse the dataset and download necessary directories into a directory of choice (e.g. `/home/johndoe/HCP_data/`), so it contains necessary files and subdirectories shown below. Note that there might be additional directories like `provenance`, `figures`, etc. Their presence will not harm anything. It is crucial to have at least the files shown below:

```
/home/johndoe/HCP_data/
                    ├── 100307
                    │   ├── MEG
                    │   │   ├── anatomy
                    │   │   │   ├── 100307_MEG_anatomy_headmodel.mat
                    │   │   │   ├── 100307_MEG_anatomy_sourcemodel_3d8mm.mat
                    │   │   └── Restin
                    │   │       └── rmegpreproc
                    │   │           ├── 100307_MEG_3-Restin_rmegpreproc.mat
                    │   │           ├── 100307_MEG_4-Restin_rmegpreproc.mat         <--- optional
                    │   │           ├── 100307_MEG_5-Restin_rmegpreproc.mat         <--- optional
                    │   └── MNINonLinear
                    │       └── aparc+aseg.nii.gz
                    ├── 102816
                    │   ├── MEG
                    │   │   ├── anatomy
(verte ...)
```

(... continued from the previous page)

```
│   │   │   ├── 102816_MEG_anatomy_headmodel.mat
│   │   │   ├── 102816_MEG_anatomy_sourcemodel_3d8mm.mat
│   │   └── Restin
│   │       └── rmegpreproc
│   │           ├── 102816_MEG_3-Restin_rmegpreproc.mat
│   │           ├── 102816_MEG_4-Restin_rmegpreproc.mat        <--- optional
│   │           ├── 102816_MEG_5-Restin_rmegpreproc.mat        <--- optional
│   └── MNINonLinear
│       └── aparc+aseg.nii.gz
│
... ...
```

NOTE: Data directory content

Inside the `/home/johndoe/HCP_data/` directory, there should not be any other files but directories named with the subject codes, i.e. 100307, 102816, etc.!

NOTE: Subjects for the illustrative example

In order to download the data needed for the ILLUSTRATIVE EXAMPLE, download the data for the following subjects: 100307 (Sub_1), 102816 (Sub_2), 105923 (Sub_3), 106521 (Sub_4), 108323 (Sub_5), 109123 (Sub_6), 111514 (Sub_7), 112920 (Sub_8), 113922 (Sub_9), 116524 (Sub_10).

5.5 Precomputed brain atlases

We have provided precomputed atlases to avoid problems with some Fieldtrip routines for preparing atlases under Windows:

Name	Reference	#ROIs	A / F*	I / G**	Voxel size in mm^3	Average ROI size in voxels (mean ± std)
Desikan-Killiany	[31]	113	A	I	8	30 ± 79
Schaefer	[34]	100	F	G	8	21 ± 9
AAL	[35]	116	A	G	8	25 ± 19

* - (A)natomical - each ROI consists of voxels with similar anatomical features; (F)unctional - each ROI consists of voxels with similar functional features.

** - (I)ndividually defined ROIs, i.e. each subject has a different map between voxel and ROI label; (G)roup-average ROIs, i.e. a map between voxels and ROI label is common for all subjects.

Paths to the precomputed atlas files:

- Desikan-Killiany:
  `$REPO_ROOT/DATA_PREPARATION/PARCELLATIONS_PREP/DK_individual_atlases/output_precomputed/DK_indAtlas_*.mat`
- Schaefer:
  `$REPO_ROOT/commonData/Schaefer2018_100Parcels_7Networks_ATLAS_interpolated_on_8mm_HCP_template.mat`
- AAL:
  `$REPO_ROOT/commonData/AAL_116Parcels_ATLAS_interpolated_on_8mm_HCP_template.mat`

6 METHODS

6.1 Data flow diagrams

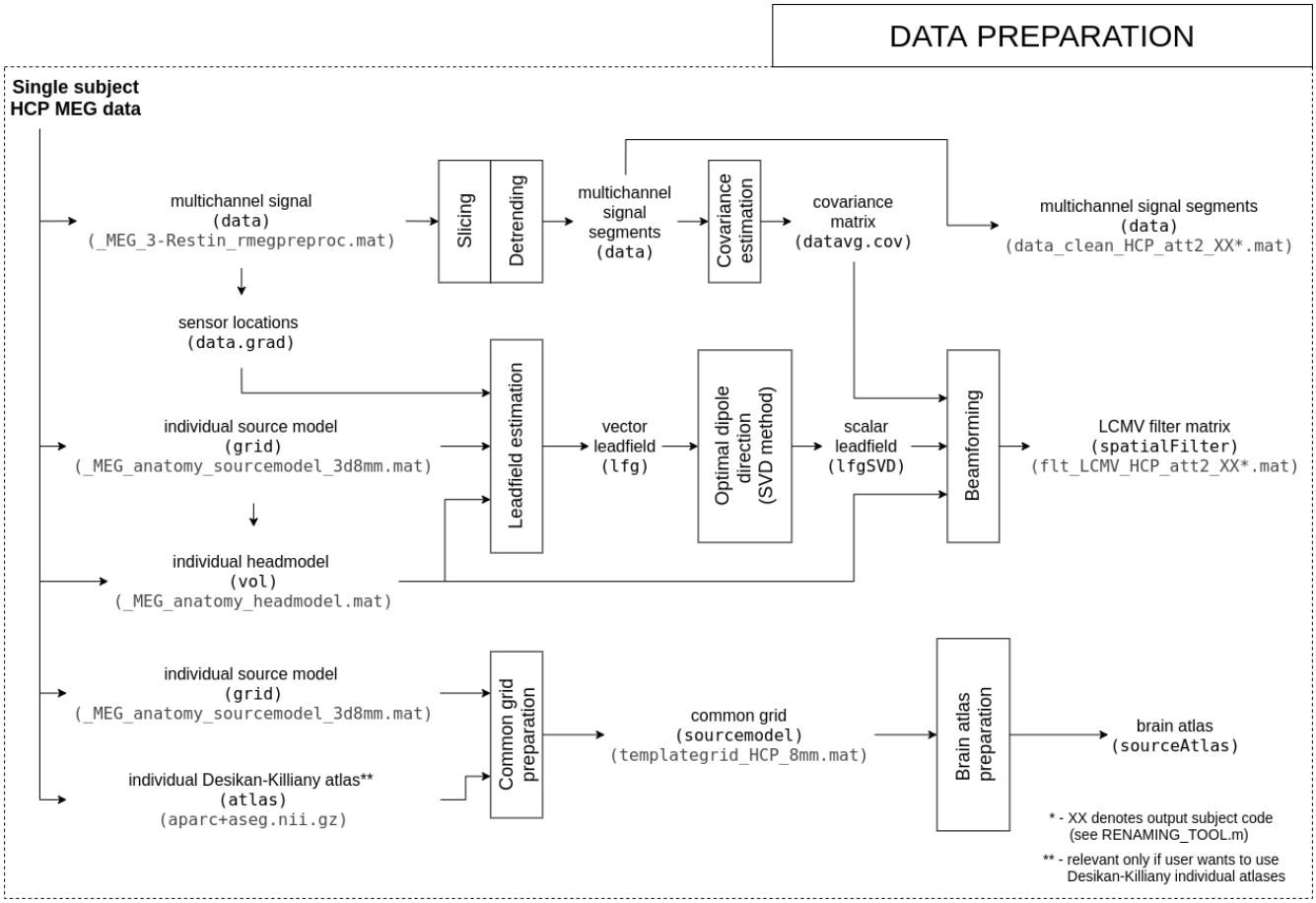


Figure 6.1: Diagram of how the data are processed by the DATA PREPARATION component (see 7 TOOLBOX ARCHITECTURE). Each depicted piece of data is endowed with its name, MATLAB workspace variable name (middle parentheses), and a file name (bottom parentheses) if it is read or written to the disk during processing. A more detailed description of each data piece is written in the text below.

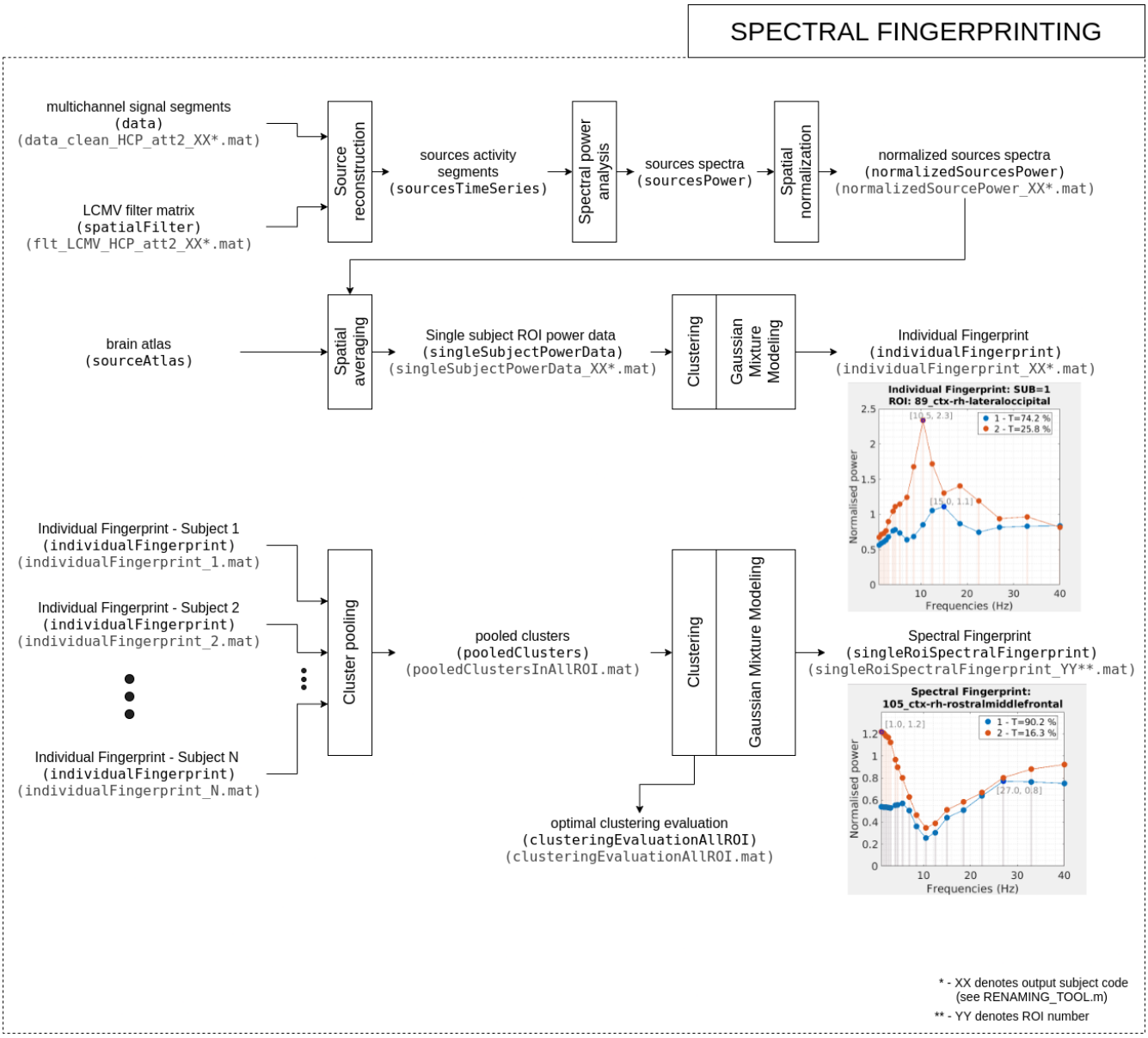


Figure 6.2: Diagram of how the data are processed by the SPECTRAL FINGERPRINTING component (see 7 TOOLBOX ARCHITECTURE). Each depicted piece of data is endowed with its name, MATLAB workspace variable name (middle parentheses), and a file name (bottom parentheses) if it is read or written to the disk during processing. A more detailed description of each data piece is written in the text below.

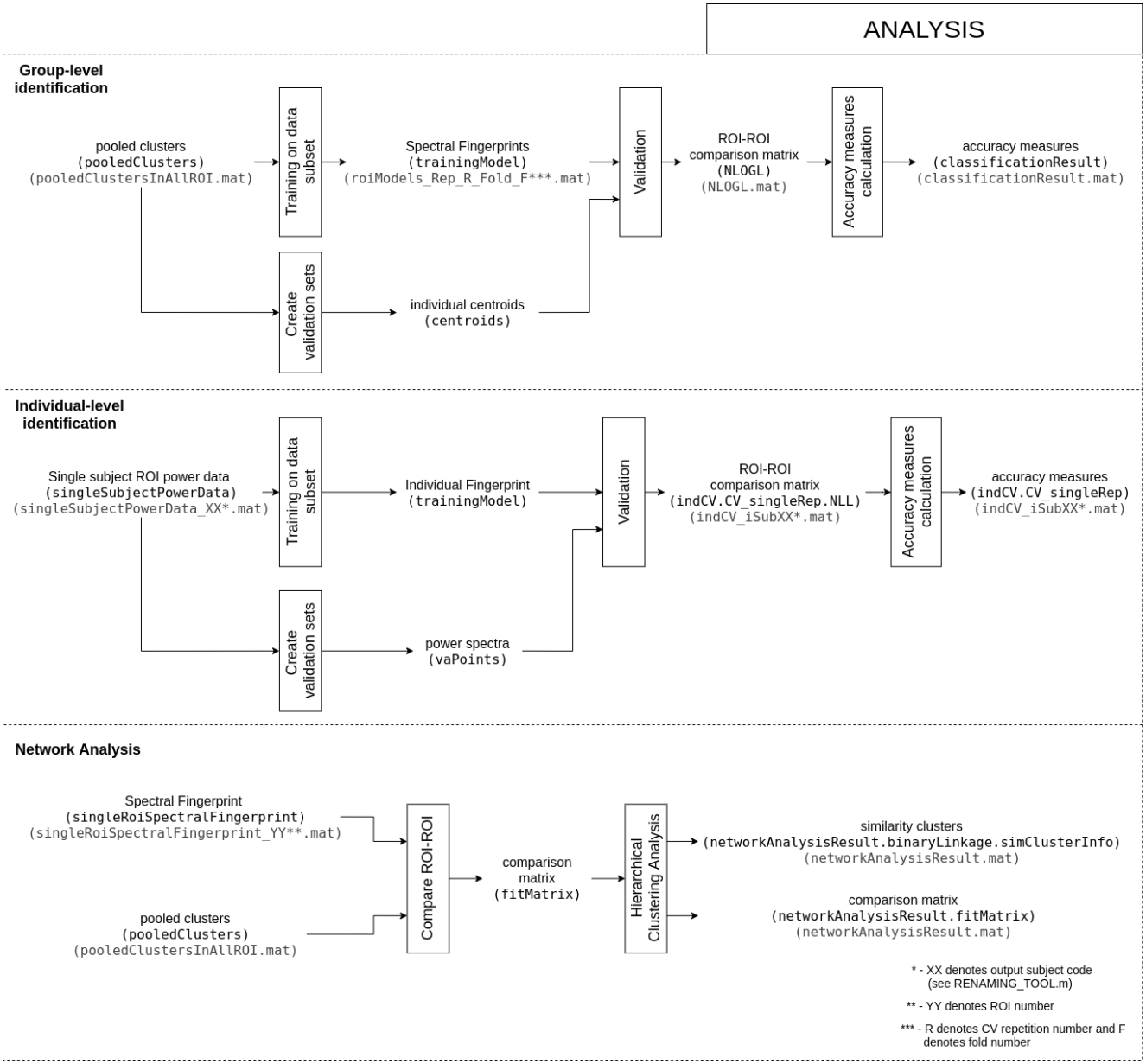


Figure 6.3: Diagram of how the data are processed by the ANALYSIS component (see 7 TOOLBOX ARCHITECTURE). Each depicted piece of data is endowed with its name, MATLAB workspace variable name (middle parentheses), and a file name (bottom parentheses) if it is read or written to the disk during processing. A more detailed description of each data piece is written in the text below.

6.2 Diagram data description

`multichannel signal`

- file name: **`_MEG_3-Restin_rmegpreproc.mat`**
- variable name: `data`
- stage: DATA PREPARATION
- description: Data structure storing multichannel signal before adjusting it for the Spectral Fingerprinting.
- fields:
  - `hdr` - structure with subfields related to the information related to the signal acquisition
  - `fsample` - sampling frequency
  - `trialinfo` - trigger info (not relevant for resting-state)
  - `grad` - sensor position data (`https://www.fieldtriptoolbox.org/faq/how_are_electrodes_magnetometers_or_gradiometers_described/`)
  - `trial` - signal values organized in segments for all sensors
  - `time` - time axis of each signal segment
  - `label` - sensor labels
  - `cfg` - additional processing information

`multichannel signal segments`

- file name: **`data_clean_HCP_att2_XX.mat`**
- variable name: `data`
- stage: DATA PREPARATION , STAGE 1
- description: Data structure storing multichannel signal after adjusting it for the Spectral Fingerprinting.
- fields:
  - `fsample` - sampling frequency
  - `hdr` - structure with subfields related to the information related to the signal acquisition
  - `grad` - sensor position data (`https://www.fieldtriptoolbox.org/faq/how_are_electrodes_magnetometers_or_gradiometers_described/`)
  - `sampleinfo` - indices of beginning and ending sample of a particular segment
  - `trial` - signal values organized in segments for all sensors (1 second segments for this particular data files)
  - `time` - time axis of each signal segment
  - `label` - sensor labels
  - `cfg` - additional processing information

`covariance matrix`

- file name: **`none`**
- variable name: `datavg.cov`
- stage: DATA PREPARATION
- description: Data covariance matrix calculated from averaged individual segments covariance matrices.

`sensor locations`

- file name: **`none`**
- variable name: `data.grad`
- stage: DATA PREPARATION
- description: Sensor position data. (`https://www.fieldtriptoolbox.org/faq/how_are_electrodes_magnetometers_or_gradiometers_described/`) necessary for preparing the leadfield, which is used in source activity reconstruction

---

`vector leadfield`

- file name: **`none`**
- variable name: `lfg`
- stage: DATA PREPARATION
- description: Forward solution for a set of dipoles in a a volume conductor model, expressed as the leadfield matrix. Each column corresponds with the potential or field distributions on all sensors for one of the x,y,z-orientations of the dipole. All single-dipole leadfields are store inside a cell array.
- fields:
  - `pos` - maps each dipole index onto set of x, y, z coordinates
  - `dim` - dimensions of the 3D space
  - `unit` - 'mm', 'cm' or 'm'
  - `inside` - logical vector containing "1" if given voxel is inside the brain or "0" otherwise
  - `cfg` - additional processing information
  - `leadfield` - cell array containing leadfield matrices for all the voxels
  - `label` - sensor labels
  - `leadfielddimord` - equals 'pos_chan_ori' which means that each cell is a position, which contains a matrix ordered by channels x orientations (xyz)

`scalar leadfield`

- file name: **`none`**
- variable name: `lfgSVD`
- stage: DATA PREPARATION
- description: Forward solution for a set of dipoles in a a volume conductor model, expressed as the leadfield vector. Each vector corresponds with the potential or field distributions on all sensors for the orientation of maximum power of the dipole. All single-dipole leadfields are store inside a cell array.
- fields:
  - `leadfield` - cell array containing leadfield vectors for all the voxels
  - `AllChanLabels` - sensor labels
  - `SelChan` - sensor labels of selected channels (not contain channels like ECG or EOG, etc.)
  - `SelChanIdx` - indices of channels selected from `lfgSVD.AllChanLabels` list
  - `inside` - logical vector containing "1" if given voxel is inside the brain or "0" otherwise

---

`LCMV filter matrix`

- file name: **`flt_LCMV_HCP_att2_XX.mat`**
- variable name: `spatialFilter`
- stage: DATA PREPARATION , STAGE 1
- description: 2D-array which represents spatial filter coefficients used for reconstruction on brain activity from sensors signal (see e.g. reference [20] to learn about LCMV spatial filter)

`individual source model`

- file name: **`_MEG_anatomy_sourcemodel_3d8mm.mat`**
- variable name: `grid`
- stage: DATA PREPARATION
- description: Grid of possible dipole locations.
- fields:
  - `pos` - maps each dipole index onto set of x, y, z coordinates
  - `dim` - dimensions of the 3D space
  - `unit` - 'mm', 'cm' or 'm'
  - `inside` - logical vector containing "1" if given voxel is inside the brain or "0" otherwise
  - `params` - warping parameters to relate individual structural MRI with the template MNI brain
  - `initial` - transform matrix connecting subject space with target (e.g. BTI) space
  - `cfg` - additional processing information
  - `coordsys` - coordinate system in which dimensions are expressed

---

`individual headmodel`

- file name: **`_MEG_anatomy_headmodel.mat`**
- variable name: `vol`
- stage: DATA PREPARATION
- description: A volume conduction model of the head (head model) based on an indivdual subject's MRI.
- fields:
  - `bnd` - structure containing geometry data of the brain tissue
  - `type` - head model type (e.g. 'singleshell')
  - `unit` - 'mm', 'cm' or 'm'
  - `cfg` - additional processing information
  - `coordsys` - coordinate system in which dimensions are expressed

---

`individual Desikan-Killiany atlas`

- file name: **`aparc+aseg.nii.gz`**
- variable name: `atlas`
- stage: DATA PREPARATION

- description: Represents individual subject atlas with ROIs from Desikan-Killiany volume segmentation and cortical parcelation.
- fields:
  - `dim` - dimensions of the 3D space
  - `hdr` - structure with subfields related to the information related to original `nii.gz` file
  - `transform` - transform matrix connecting subject space with target (e.g. BTI) space
  - `unit` - 'mm', 'cm' or 'm'
  - `aparc` - double 3-D array with integer values from 1 to N (the value 0 means unknown). Values are related to the ROI labels stored in aparclabel field. The size of each dimension is stored in the dim field. It stores a ROI label for each brain voxel.
  - `aparclabel` - cell of strings containing name of each ROI

`common grid`

- file name: **`templategrid_HCP_8mm.mat`**
- variable name: `sourcemodel`
- stage: DATA PREPARATION
- description: Represents grid of voxels where brain source activity will be reconstructed at. It is the space common for all subjects.
- fields:
  - `xgrid` - double; array of possible x-coordinates of brain voxels expressed in milimeters, centimeters or meters.
  - `ygrid` - double; array of possible y-coordinates of brain voxels expressed in milimeters, centimeters or meters.
  - `zgrid` - double; array of possible z-coordinates of brain voxels expressed in milimeters, centimeters or meters.
  - `dim` - double; 3-vector of number voxels in each dimension. Total number of voxels equals (dim(1)*dim(2)*dim(3)).
  - `pos` - double; 2-D array of size (dim(1)*dim(2)*dim(3)) x 3. Each row represent 3D-coordinates of single voxels in the grid.
  - `inside` - double; 1-D array that lists indices of voxels that lie inside brain
  - `outside` - double; 1-D array that lists indices of voxels that lie outside brain

---

`brain atlas`

- file name: **`none`**
- variable name: `sourceAtlas`
- stage: DATA PREPARATION , STAGE 2
- description: Struct containing common atlas for all subjects or array of structs if atlas type is 'individual' (one struct per subjects).
- fields:
  - `dim` - double 1-D array of positive integers; the size of the 3D volume in voxels
  - `transform` - double 2-D array containing affine transformation matrix for mapping the voxel coordinates to the head coordinate system

  - `unit` - string; the units in which the coordinate system is expressed
  - `tissue` - double 3-D array with integer values from 1 to N (the value 0 means unknown). Values are related to the ROI labels stored in tissuelabel fields. Size of each dimension is stored in the dim field. It stores ROI label for each brain voxel.
  - `tissuelabel` - cell of strings containing name of each ROI.

`sources activity segments`

- file name: **`none`**
- variable name: `sourcesTimeSeries`
- stage: STAGE 1
- description: Data structure storing reconstructed time series for each dipole inside the brain.
- fields:
  - `trial` - signal values organized in segments for all sources (1 second segments for this particular data files)
  - `time` - time axis of each signal segment
  - `fsample` - sampling frequency
  - `label` - source labels
  - `sampleinfo` - indices of beginning and ending sample of a particular segment

---

`sources spectra`

- file name: **`none`**
- variable name: `sourcesPower`
- stage: STAGE 1
- description: Double 3D array that contains real values of power spectrum for each time segment (trial), source and frequency. Size: trials x sources x frequencies.

`normalized sources spectra`

- file name: **`normalizedSourcePower_XX.mat`**
- variable name: `normalizedSourcesPower`
- stage: STAGE 1 , STAGE 2 , STAGE 7
- description: double 3D array that contains real values of power spectrum for each time segment (trial), source and frequency, but normalized in some way (see `normalizeActivity.m` to learn about normalization possibilities). Normalization by the whole brain spectrum is default. Size: trials x sources x frequencies.
- fields:
  - `powerSpectrum` - contains values of normalized power spectrum for each time segment (trial), source/sensor and frequency. Size: trials x sources x frequencies.
  - `freqAxis` - frequencies of interest for which spectral power was computed
  - `dimord` - equals 'rpt_chan_freq' which means that first dimension denotes trials, second denotes sources (not channels; 'chan' for Fieldtrip compatibility), and third denotes frequencies of interest.
  - `label` - source labels

`Single subject ROI power data`

- file name: **`singleSubjectPowerData_XX.mat`**
- variable name: `singleSubjectPowerData`
- stage: STAGE 2 , STAGE 7
- description: Normalized sources spectra averaged across voxels belonging to particular ROI. Each cell contains data related to single ROI.
- fields:
  - `subjectID` - subject code
  - `iSub` - number of the subject (same as in `CFG.Global.goodSubjects`)
  - `iROI` - region number (same as in `CFG.Global.goodROI`)
  - `tissuelabels` - cell of strings containing name of each ROI
  - `goodROI` - list of ROIs from the atlas to be processed by the Spectral Fingerprinting routines (same as configured, i.e. `CFG.Global.goodROI`)
  - `nRoiAtlas` - number of ROIs in the atlas (greater or equal to the number of elements in `CFG.Global.goodROI`)
  - `normalizedPowerSpectrumTrials` - contains values of normalized power spectrum for current ROI for each time segment (trial) and frequency. Size: trials x frequencies.
  - `freqAxis` - frequencies of interest for which spectral power was computed
  - `nTrials` - number of time segments after after rejecting segments marked as outliers
  - `rejectedTrialsIndices` - indices of trials marked as outliers and rejected during STAGE_2
  - `nTrialsBeforeRejection` - number of time segments before rejecting segments marked as outliers

---

`Individual Fingerprint`

- file name: **`individualFingerprint_XX.mat`**
- variable name: `sourcesTimeSeries`
- stage: STAGE 1 , STAGE 2 , STAGE 3
- description: Structure containing result of the Spectral Fingerprinting approach for single subject and ROI.
- fields:
  - `subjectID` - subject code
  - `iSub` - number of the subject (same as in `CFG.Global.goodSubjects`)
  - `tissuelabels` - cell of strings containing name of each ROI
  - `nRoiAtlas` - number of ROIs in the atlas
  - `goodROI` - list of ROIs from the atlas to be processed by the Spectral Fingerprinting routines (same as configured, i.e. `CFG.Global.goodROI`)
  - `spectralModes` - cell array; each cell represents single ROI and contains a double 2D-array of size clusters x frequencies. Each row contain single gaussian component center. Equals to `gmdistribution.mu`. In the Spectral Fingerprinting jargon each component is called *spectral mode* or *cluster*.
  - `trialsSpectralModeMembership` - cell array; each cell represents single ROI and contains a double 1D-array containing cluster index for each of the clustered points in the input data matrix. Contains values from 1 to `cfg.nClusters`.

- `gmClassInstance` - cell array; each cell represents single ROI and contains an object that stores a Gaussian mixture model (GMM), which is a multivariate distribution that consists of multivariate Gaussian distribution components. Further details: (`https://www.mathworks.com/help/stats/gmdistribution.html`)
- `didGmmConverge` - vector of doubles; 0 (EM algorithm did not converge) or 1 (EM algorithm has converged)
- `clusterDuration` - cell array; each cell represents single ROI and contains a 1D-array containing percentages of points belonging to each cluster.
- `emptyClusters` - cell array; each cell represents a single ROI and contains a 1D-array containing cluster indices that has zero points assigned to it. Such a situation can happen if the input data are degenerated or parameters force that some of the resulting spectral modes are identical. In that situation, one of the "twins" has no points assigned to it because they were "stolen" by its "sibling".
- `numberOfClustersInThisRoi` - vector of doubles; number of spectral modes representing this particular ROI
- `optimalNumClustEvaluation` - cell array; each cell represents single ROI and contains a struct storing results of optimal cluster number evaluation for each iteration (see `$REPO_ROOT/STAGE_4/functions/serialProcessing.m` to learn more).
- `nTrials` - number of time segments after after rejecting segments marked earlier as outliers
- `freqAxis` - frequencies of interest for which spectral power was computed

---

`pooled clusters`

- file name: **`pooledClustersInAllROI.mat`**
- variable name: `pooledClusters`
- stage: STAGE 2
- description: Array of structs (size 1 x nRoiAtlas) contains individual fingerprints data gathered from all subjects.
- fields:
  - `numCentroidsPerSubject` - double 1D-array with numbers that mean how many spectral modes (a.k.a. centroids) was computed in STAGE_2 for particular subject (one number relates to one subject).
  - `centroidsAllSubjects` - double 2D-array where each row represents single point in multidimensional frequency space (this point is called spectral mode). Each point has its "owner", i.e. the subject to whom it belongs. Points were appended row-by-row, first all points from the first subjects, then all points from the second subject, and so on. A total number of rows equals the sum of the numbers in numCentroidsPerSubject field. The number of columns is equal to the dimensionality of the frequency space.
  - `centroidsSubjectIndices` - double 1D-array that has as many rows as centroidsAllSubjects field. Contains indices of subject that "owns" particular point (row in `centroidsAllSubjects` field)
  - `centroidsDuration` - cell of 1D-arrays storing time duration of each spectral mode expressed in percent. Each array contains data for single subject.
  - `iROI` - double integer; ROI index;
  - `nRoiAtlas` - double integer positive number of ROIs defined in used brain atlas.
  - `goodROI` - list of ROIs from the atlas to be processed by the Spectral Fingerprinting routines (same as configured, i.e. `CFG.Global.goodROI`)
  - `freqAxis` - double 1D-array of frequencies in Hz.

`Spectral Fingerprint`

- file name: **`singleRoiSpectralFingerprint_YY.mat`**
- variable name: `singleRoiSpectralFingerprint`
- stage: STAGE 5 , STAGE 6 , STAGE 8
- description: Structure representing the end result of the Spectral Fingerprinting pipeline for particular ROI.
- fields:
  - `iROI` - double integer; ROI index
  - `nRoiAtlas` - double integer positive number of ROIs defined in used brain atlas.
  - `goodROI` - list of ROIs from the atlas to be processed by the Spectral Fingerprinting routines (same as configured, i.e. CFG.Global.goodROI)
  - `freqAxis` - double 1D-array of frequencies in Hz.
  - `lvl1_numCentroidsPerSubject` - number of centroids in individual fingerprint for each subject
  - `lvl1_centroidsFromAllSubjects` - centroids from all subjects gathered in the single matrix, row-by-row
  - `lvl1_centroidsSubjectIndices` - vector containing subject ID numbers assigned to each of the centroids in `lvl1_centroidsFromAllSubjects` field
  - `lvl1_centroidsDuration` - cell array; each cell contains vector of spectral modes duration, one vector per subject
  - `lvl2_kMeans_pointsClusterIndices` - vector containing group cluster index from the first part of the group-level clustering (e.g. kmeans; before EM algorithm and GMM kicks in) for each of the gathered lvl1 centroid
  - `lvl2_kMeans_centroids` - array of centroids from the first part of the group-level clustering (ditto). Centroids are stacked row-by-row.
  - `lvl2_kMeans_withinClusterDistance` - within-cluster distance from the first part of the group-level clustering (ditto)
  - `lvl2_gmm_centroids` - array of centroids from the second part of the group-level clustering (Gaussian Mixture Modelling using EM algorithm). Centroids are stacked row-by-row. These are the final clusters.
  - `lvl2_gmm_pointsMembership` - vector containing group cluster index from the second part of the group-level clustering (ditto) for each of the gathered lvl1 centroid
  - `lvl2_gmm_gmClassInstance` - gmdistribution object representing the result of the EM algorithm implemented in MATLAB (`fitgmdist`).
  - `lvl2_gmm_didGmmConverge` - flag. Equals 1 if the EM algorithm converged in the prespecified number of iteration or 0 if not.
  - `lvl2_gmm_centroidDuration` - array containing percentages of points belonging to each final cluster
  - `lvl2_gmm_subjectsContributed` - cell containing the arrays with indices of unique subjects contributed to particular final cluster
  - `lvl2_gmm_nSubjectsPerCentroid` - array containing the number of unique subjects contributed to particular final cluster
  - `isOnlyPopularClustersShown` - flag. Equals 1 if final clusters stored in this structure are those after rejecting clusters with less then required number of subjects (value of `CFG.STAGE_5.majoritySubjectsNum`) or 0 if not.

`optimal cluster evaluation`

- file name: **`clusteringEvaluationAllROI.mat`**
- variable name: `pooledClusters`
- stage: STAGE 4 , STAGE 6 , STAGE 8
- description: Structure array containing results of the clustering evaluation for pooled clusters. Each structure is assigned to particular region of interest.
- fields:
  - `optimalNumClusters` - final optimal number of clusters determined for current ROI
  - `optNumClustersAllIter` - optimal number of clusters obtained for each single iteration of the clustering evaluation
  - `criterionValues` - values of the criterion used in the determination of optimal number of clusters (e.g. Silhouette criterion). Values for each single iteration.
  - `iROI` - double integer; ROI index
  - `goodRoi` - list of ROIs from the atlas to be processed by the Spectral Fingerprinting routines (same as configured, i.e. `CFG.Global.goodROI`)

`ROI-ROI comparison matrix`

- file name: **`NLOGL.mat`**
- variable name: `NLOGL`
- stage: STAGE 6
- description: Array of structures containing fitness 3D-array for each Cross-Validation repetition and each fold. It is a result of comparing each ROI trained model (from training subjects) with each ROI validation points (from validation subjects).
- fields:
  - `NLOGL.Reps` - structure array containing data for every CV repetition
  - `NLOGL.Reps.iRep` - index of CV repetition
  - `NLOGL.Reps.Folds` - structure array containing fitness 3D-arrays for each fold
  - `NLOGL.Reps.Folds.nLogL` - 3D-array containing negative-log likelihood values between ROI A (first dimension) and ROI B (second dimension) for given validation subject (third dimension). The lower the value the better the fit.

`accuracy measures`

- file name: **`classificationResult.mat`**
- variable name: `classificationResult`
- stage: STAGE 6
- description: Structure containing ROI identification accuracy measures. For more detailed explanation how accuracy and mean ranks are defined consult 6.3.4 Analysis (STAGES 6-8) section.
- fields:
  - `nReps` - number of CV repetitions
  - `nFolds` - number of data folds used in CV
  - `CV_reps` - structure containing accuracy measures for single CV repeitions

  - `accPerRoiMeanReps` - accuracy of identification for each ROI; averaged across repetitions
  - `acc_std_PerRoiStdReps` - standard deviation of the mean accuracy across repetitions (for each ROI)
  - `meanRankPerRoiMeanReps` - accuracy of identification for each ROI expressed as a mean rank; averaged across repetitions
  - `meanRank_std_PerRoiStdReps` - standard deviation of the averaged mean rank across repetitions (for each ROI)

`individual centroids`

- file name: **`none`**
- variable name: `centroids`
- stage: STAGE 6
- description: 2D array containing single ROI clusters from individual fingerprints of validation subjects to be tested against regional spectral fingerprints trained with training subjects data. Points are stacked row-wise.

`power spectra`

- file name: **`none`**
- variable name: `vaPoints`
- stage: STAGE 7
- description: 2D array containing single ROI normalized power spectra from validation data fold to be tested against regional individual fingerprints trained with training data fold. Points are stacked row-wise. Both training and validation data were extracted from the same single subject.

`ROI-ROI comparison matrix`

- file name: **`indCV_iSubXX*.mat`**
- variable name: `indCV.CV_singleRep.NLL`
- stage: STAGE 7
- description: Structure containing a result of testing validation data (regions' normalized power spectra) against training data (regional individual fingerprints). Values are negative log-likelihoods. The lower the value the better the fit.
- fields:
  - `Folds` - structure array; each structure containing validation result for given CV fold.
  - `Folds.nLogL` - 2D-array containing negative log-likelihoods values between ROI A (first dimension) and ROI B (second dimension). The lower the value the better the fit.

`accuracy measures`

- file name: **`indCV_iSubXX*.mat`**
- variable name: `indCV.CV_singleRep`
- stage: STAGE 7
- description: Structure containing ROI identification accuracy measures. For more detailed explanation how accuracy and mean ranks are defined consult 6.3.4 Analysis (STAGES 6-8) section.
- fields:
  - `iRep` - index of CV repetition

- `cvPart` - paritioning scheme for the data to be divided into training and validation fold (`https://www.mathworks.com/help/stats/cvpartition.html`)
- `cvFold` - contains data and results of single fold of CV
- `accRoiMeanFolds` - accuracy of identification for each ROI; averaged across folds
- `acc_std_RoiStdFolds` - standard deviation of the mean accuracy across folds (for each ROI)
- `meanRankRoiMeanFolds` - accuracy of identification for each ROI expressed as a mean rank; averaged across folds
- `meanRank_std_RoiStdFolds` - standard deviation of the averaged mean rank across folds (for each ROI)
- `NLL` - Structure containing a result of testing validation data (regions' normalized power spectra) against training data (regional individual fingerprints). Values are negative log-likelihoods. The lower the value the better the fit. Described above.

`comparison matrix`

- file name: **`none`**
- variable name: `fitMatrix`
- stage: STAGE 8
- description: 2D matrix of negative log-likelihoods representing fitness of the spectral fingerprint of particular ROI to pooled clusters of another ROI.

`similarity clusters`

- file name: **`networkAnalysisResult.mat`**
- variable name: `networkAnalysisResult.binaryLinkage.simClusterInfo`
- stage: STAGE 8
- description: Clusters data formed during Hierarchical Clustering Analysis done on spectral fingerprints.
- fields:
  - `indices` - cell array with ROI indices grouped in vectors. Each vector forms single clusters.
  - `labels` - labels of the ROIs stored in the indices field

`comparison matrix`

- file name: **`networkAnalysisResult.mat`**
- variable name: `networkAnalysisResult.fitMatrix`
- stage: STAGE 8
- description: 2D matrix of negative log-likelihoods representing fitness of the spectral fingerprint of particular ROI to pooled clusters of another ROI. Copy of the variable described above.

6.3 How data are processed?

This section describes the methods used throughout every part of the software.

6.3.1 Functions documentation

The code is rich in comments and explanations. Most `.m` files that contains a function is documented inside. For the list of those functions consult `$REPO_ROOT/docs/FUNCTIONS_REFERENCE.html` file.

6.3.2 Data preparation routines

Atlas preparation routines are described in the following files:

- To prepare Schaefer parcellation consult:
  `$REPO_ROOT/DATA_PREPARATION/PARCELLATIONS_PREP/Schaefer100_preparation.org`
- To prepare AAL parcellation consult:
  `$REPO_ROOT/DATA_PREPARATION/PARCELLATIONS_PREP/AAL116_preparation.org`
- To prepare individual-subjects Desikan-Killiany parcellation consult:
  `$REPO_ROOT/DATA_PREPARATION/PARCELLATIONS_PREP/DK_individual_atlases/Desikan_Killiany_preparation.m`

HCP data preparation routines are divided into four scripts that are located inside `$REPO_ROOT/DATA_PREPARATION/HCP_DATA_PREP/` directory:

1. `PREPROCESSING.m` - reslices data into segments of given duration (e.g. 1000 ms per trial) and detrends each segment's signal
2. `COVARIANCE_GEN.m` - generates single covariance matrix (the average of the single-segment covariance matrices);
3. `LEAFIELD_GEN.m` - generates two leadfields: first is a vector leadfield and second is a scalar leadfield; obtained from the vector leadfield by the SVD method (see chapter 4.3. "Scalar adaptive spatial filter: deriving the optimum source orientation" in [18]);
4. `LCMV_FILTER_GEN.m` - combines generated covariance matrix and scalar leadfield into LCMV filter matrix.

There is also a fifth script called `RENAMING_TOOL.m` which serves for creating a copy of selected data with converted names (changing subject codes into numbers and shorten file names if necessary).

6.3.3 Spectral Fingerprinting (STAGES 1-5)

The general idea behind the Spectral Fingerprinting is presented in Fig. 4.2.

Signal preprocessing. Computing spectral fingerprints starts with multivariate signal (cleaned from artifacts [36], [37], and individual structural scans or head models [38], [39] for a group of subjects. During data preparation routines selection of the signal data from the time interval of interest and division to subsequent short-time windows (e.g. 1 second segments) is performed, and computation of the forward model of the human brain to describe how source activity is propagated to sensors based on the conductivity of different types of brain tissue carried out [39]. The output consists of multiple spectral profiles assigned to each of the studied brain regions (Fig. 4.1). Each power spectrum (called *spectral profile* or *cluster*) represents a different mode of brain activity that can be generalized across participants. To calculate it, following steps need to be performed.

Spectral power. Estimation of frequency spectrum and its power for each time segment [40], [41].

Source reconstruction. Solve the inverse problem to estimate the frequency spectrum of the particular brain region (Region Of Interest, ROI) by utilizing source analysis methods [39], [29], [20], [26], [22], [42]. For the ROI definition one can select one of the possible brain atlases:

- Schaefer with 100 ROIs [34], [43],
- AAL with 116 ROIs [35],

AAL atlas includes cerebellar parcellation. Schaefer parcellation does not include cerebellar parcellation. AAL and Schaefer atlases consist of multiple voxels per ROI.

Individual Fingerprinting is the calculation of spectral profiles (a.k.a. *modes*) for each individual subject for each ROI [7]. Each subject's single-trial power spectra are partitioned into a fixed or optimal number of clusters per anatomical area using k-means as default. Then, Gaussian Mixture Models (GMM) were fitted separately to the clusters of each ROI for each of the participants. The number of points in a cluster is proportional to the time given activity was present. As a result, each profile potentially represents the involvement of that ROI in different functional brain network state.

Group-level fingerprinting. For each ROI: gather centers of profiles from all participants (pooled clusters), and cluster these one more time (kmeans + GMM) to obtain spectral profiles representing brain dynamics states generalizable across participants. Again, the number of clusters can be set to be fixed or optimal. Each profile could be viewed as a particular brain activity mode and plotted as the spectrum with a dominant peak/valley. A set of profiles belonging to each ROI is called *spectral fingerprint* (SF) of that ROI (see Fig. 4.1). The average duration of each cluster was computed by counting the number of individual trials that contributed to a specific cluster, expressing this in percent, and averaging across unique participants. Note that this procedure can yield cumulative percentages of above 100%, according to [7].

NOTE: Why do output frequencies can differ from those in the configuration?

Frequency analysis is done using `ft_freqanalysis`. This function takes as an input a set of frequencies of interest (`cfg.foi`). For example: `cfg.foi = logspace(0, log10(120), 50)`, which gives a following vector:

```
[ 1.0000, 1.1026, 1.2158, 1.3406, ..., 98.6999, 108.8301, 120.0000 ] Hz
```

However, vector of output frequencies is different:

```
[ 1.0000, 1.5000, 2.0000, 2.5000, ..., 98.5000, 109.0000, 120.0000 ] Hz
```

The aforementioned difference is a result of the rounding `cfg.foi` to match the frequency resolution and usage of MATLAB's `unique` function (refer to `ft_specest_mtmfft` lines starting from 104: `"% Set freqboi and freqoi"`). If initial frequencies are close to each other, then after quantization to match frequency resolution, some of them will overlap, causing repeats in the vector of frequencies of interest. After line 108: `freqboi = unique(freqboi);` repeats are removed and the "corrected" vector is shorter. To control this behaviour one can increase the padding length (`cfg.pad` for FieldTrip function `ft_freqanalysis`) so the frequency axis will be more granular and thus rounding will not produce repeats in the frequency vector and unique function will not remove any element from `cfg.foi`.

6.3.4 Analysis (STAGES 6-8)

Group-level Identification (STAGE 6).

This stage aims to measure how unique are the regional spectral fingerprints (obtained from STAGES 1-5), how well we can identify them. This can tell whether measured brain activity is characteristic enough to guess which fingerprint belongs to which brain region.

Each ROI is eventually characterized with two measures of identification accuracy:

- mean rank,
- accuracy.

Mean rank is based on the idea from [7]. It is a non-negative real value, constrained between 1 and the number of ROIs taken into consideration (for example, if one wants to perform identification among 10 ROIs, mean rank will be constrained between value 1 and 10). It tells how hard it is to identify a particular ROI. A mean rank of 1 indicates that the correct model area was the most likely to fit the test area; a rank of 2 indicates it was the second most likely, and so on.

Accuracy is more intuitive and also a more conservative measure than mean rank. It is a non-negative real value between 0 and 1 (which can be translated into percent easily), independent of the number of ROIs taken into consideration. It tells how often a particular ROI is correctly recognized among the others. An accuracy value of 1

means it is correctly guessed 100% times. A value of 0 means there were no correct guesses at all, and the spectral fingerprint of that ROI is totally confused with all other regions' fingerprints.

Group-level Identification is based on the *Cross-Validation* procedure. Cross-Validation is a statistical method of evaluating and comparing learning algorithms by dividing data into two segments:

1. **Training**. One used to learn or *train* a model
2. **Validation**. The other used to *validate* the model

In typical cross-validation, the training and validation sets must cross-over in successive rounds such that each data point has a chance of being validated against. The basic form of cross-validation is *k-fold cross-validation* [44].

In k-fold cross-validation, the data is:

1. First partitioned into k equally (or nearly equally) sized *folds*,
2. Subsequently, k *iterations* of *training* and *validation* are performed:
   2.1 Within each iteration, a different fold of the data is held out for *validation*,
   2.2 The remaining $k-1$ folds are used for *learning*.

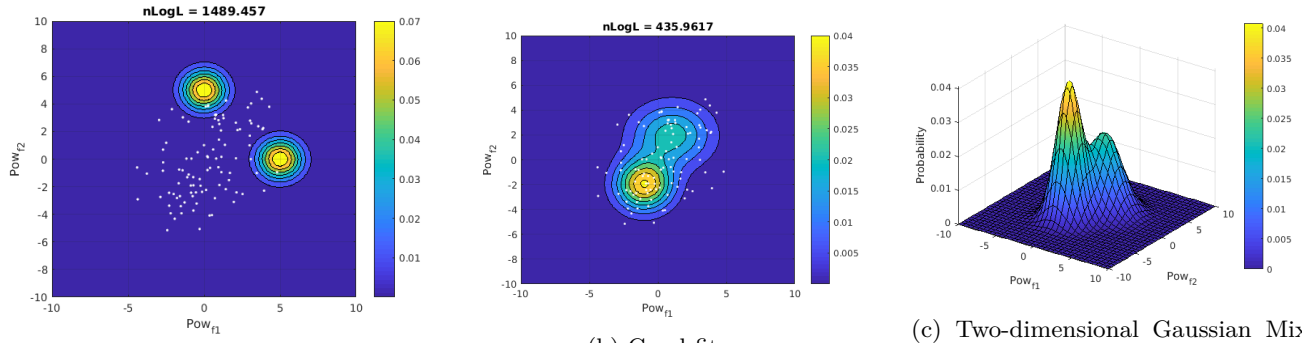
This framework was adapted to the data produced by STAGES 1-5 (Fig. 6.2). Subjects are **randomly** assigned to folds. Each data fold contains individual fingerprints' clusters (a.k.a. *spectral modes*, see `Individual Fingerprint` variable in 6.2 Diagram data description) from the subset of subjects assigned to that fold. Then spectral fingerprints are constructed from the clusters pooled in the training folds - for each ROI separately. These spectral fingerprints will serve as regional training models. Spectral modes from validation folds are tested against trained models using the following formula:

$$nLogL = -\sum_{i=1}^{n} log\left[\sum_{j=1}^{m} \alpha_j \phi(x_i|\mu_j, \Sigma_j)\right]$$

$$\phi(x \mid \mu, \Sigma) := \frac{1}{\sqrt{(2\pi)^F \det(\Sigma)}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right)$$

where:

F - number of dimensions (number of frequencies of interest)
n - number of validation points (centroids) from current validation subject's individual fingerprint
m - number of Gaussian components of trained model
α_j - j-th Gaussian component mixing proportion, $\sum_{j=1}^{k} \alpha_j = 1$
x_i - i-th point (centroid) from validation set of current validation subject, $x_i \in \mathbb{R}^F$
μ_j - j-th Gaussian component expected value, $\mu_j \in \mathbb{R}^F$
Σ_j - j-th Gaussian component covariance matrix, $\Sigma \in \mathbb{R}^{F \times F}$



(a) Poor fitness.

(b) Good fitness.

(c) Two-dimensional Gaussian Mixture Model.

Figure 6.4: Validation example for two-dimensional data. Validation data (white points) and trained regional model (contour lines) should match. Better fitness is reflected in a lower value of the nLogL measure.

The lower the nLogL value, the better fit between trained model and validation data (Fig. 6.4).

Average regional accuracy/rank is then computed (Fig. 6.6). These values represent a single fold of the cross-validation, and the iteration is finished.

For the next iteration, another fold of data is selected as a validation fold, and others serve as training folds. The training and validation phase repeats, then the iteration is concluded with the average accuracy/rank computation. After k iterations single repetition of cross-validation is completed.

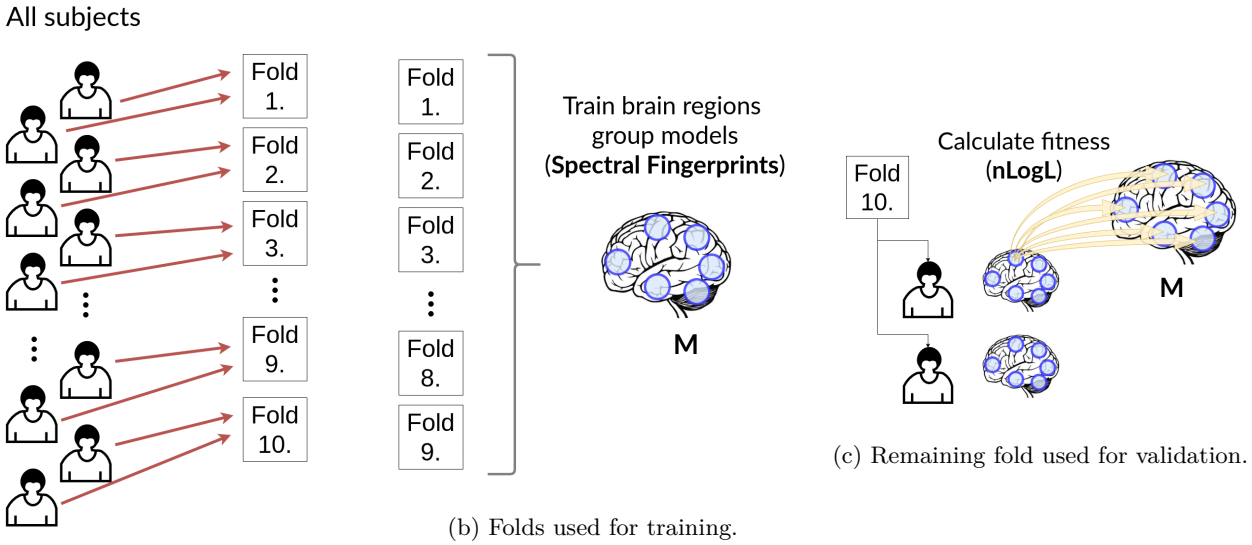
One can set more than one repetition of cross-validation to be launched, and the averaged results across repetitions will be obtained as well. Each repetition is different because subjects are assigned to the folds **randomly**.

To conclude, below is the exemplary summary of calculations for STAGE 6.

EXAMPLE: Summary of 10-fold CV repeated 100 times with the data from 89 subjects

1. For r = 1 ... 100 repetitions ($R > 1$ is optional*):
   (a) Assign randomly $N = 89$ subjects to 10 different sets (folds)
   (b) For each k-th fold:
      i. **Training**: Train model (spectral fingerprint) of each brain region using 79 subjects from folds $\neq k$
      ii. **Validation**: For each of 10 subjects in k-th fold:
         - Calculate fit (**nLogL**) between each brain region validation data and each brain region from trained model
      iii. Calculate **mean accuracy** and **mean rank** across validation subjects + their standard deviations
   (c) **Average** accuracy and rank **across folds** + compute standard deviation across folds
2. (Optional*) **Average** accuracy and rank **across repetitions** + compute standard deviation across repetitions

* - for justification see [45].



(a) Subjects assigned to folds.

(b) Folds used for training.

(c) Remaining fold used for validation.

Figure 6.5: Data division in 10-fold Cross-Validation in STAGE 6.

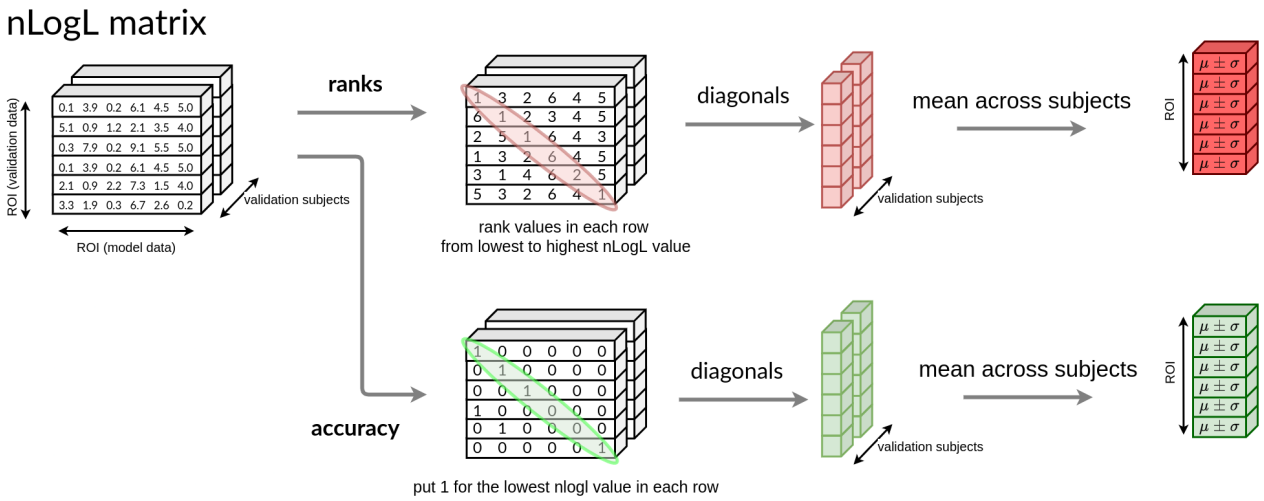


Figure 6.6: Fitness values (nLogL) between region pairs are obtained for all subjects from the validation fold and thus organized in a 3D-array form. Each row's values are ranked from the lowest to the highest. Then, two measures - ranks and accuracies - are obtained. To calculate mean regional ranks and their standard deviations, diagonals are averaged across validation subjects. The lower the rank value, the better identification of particular ROI. To calculate mean regional accuracies and their standard deviations, ranks > 1 are zeroed, then diagonals are averaged across validation subjects. The higher the accuracy value, the better identification of a particular ROI.

Individual-level Identification (STAGE 7). The goal of this stage is to measure how unique are the individual redional fingerprints (obtained from STAGES 1-2), how well we can identify them. This can tell whether measured brain activity is characteristic enough so it can be guessed which fingerprint belongs to which brain region. The difference between STAGE 6 and this stage is that here we conduct identification at the level of single subject.

Individual-level Identification is based on the *Cross-Validation* procedure. Identification accuracy is expressed via two measures:

- mean rank,
- accuracy.

See the beginning paragraphs of the 6.3.4 Analysis (STAGES 6-8) section to learn about these measures and the cross-validation procedure.

This framework was adapted to the data produced by STAGES 1-2 (Fig 6.2). Normalized power spectra segments from a single subject (see `normalizedSourcesPower` variable in 6.2 Diagram data description section) from multiple regions ale pooled together and then distributed **randomly** across the folds. Then individual fingerprints are constructed from the segments pooled in the training folds - for each ROI separately. These individual fingerprints will serve as regional training models. Segments from validation folds are tested against trained models using the following formula:

$$nLogL = -\sum_{i=1}^{n} log\left[\sum_{j=1}^{m} \alpha_j \phi(x_i|\mu_j, \Sigma_j)\right]$$

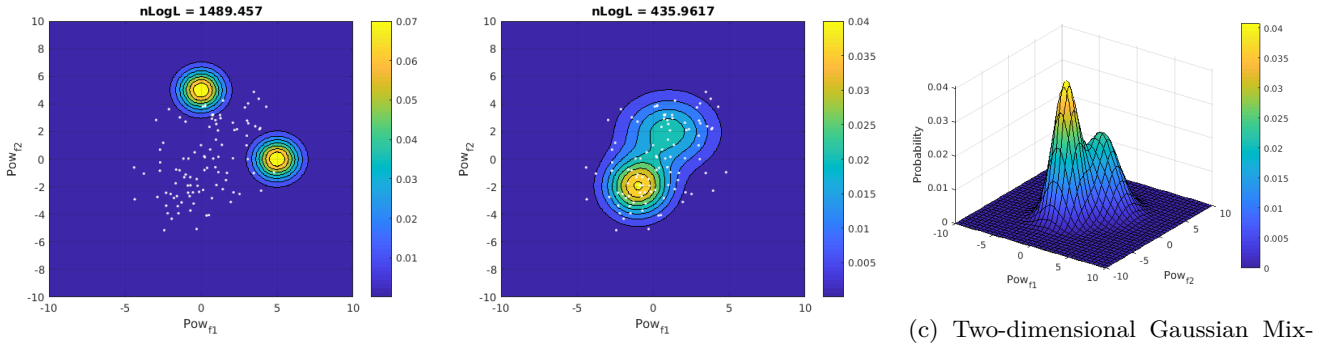
$$\phi(x \mid \mu, \Sigma) := \frac{1}{\sqrt{(2\pi)^F \det(\Sigma)}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right)$$

where:

F - number of dimensions (number of frequencies of interest)
n - number of validation points (power spectra) of particular ROI

m - number of Gaussian components of a trained model
α_j - j-th Gaussian component mixing proportion, $\sum_{j=1}^{k} \alpha_j = 1$
x_i - i-th point (power spectrum) from validation set of particular ROI, $x_i \in \mathbb{R}^F$
μ_j - j-th Gaussian component expected value, $\mu_j \in \mathbb{R}^F$
Σ_j - j-th Gaussian component covariance matrix, $\Sigma \in \mathbb{R}^{F \times F}$

The lower the nLogL value, the better fit between trained model and validation data (Fig. 6.7).

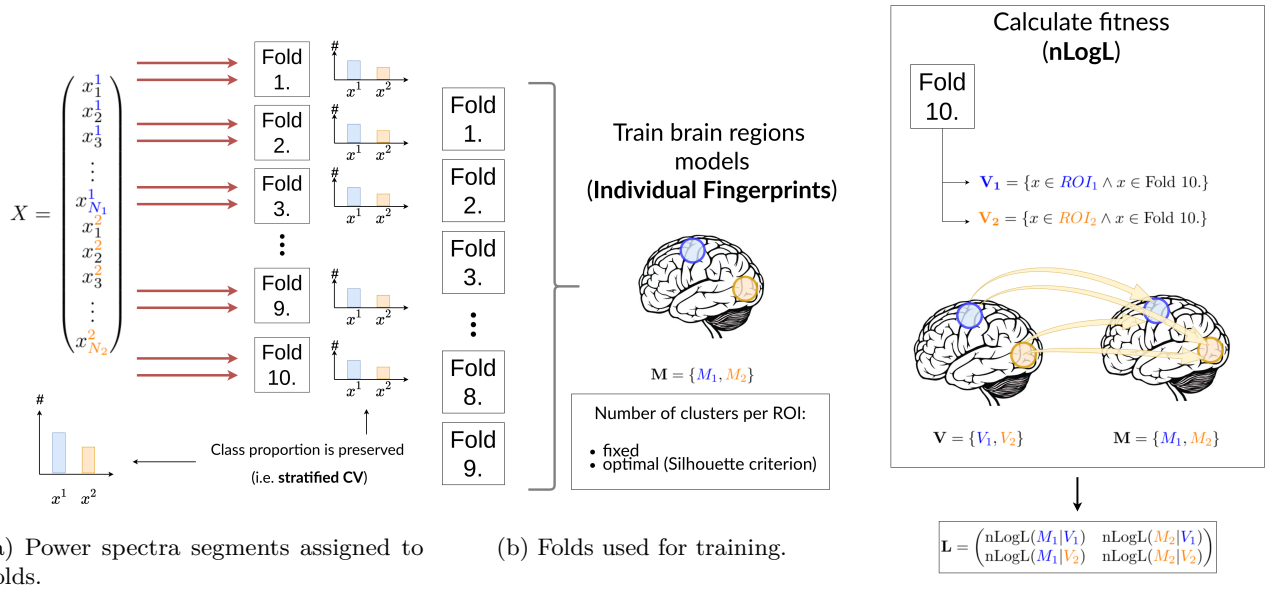


(a) Poor fitness.

(b) Good fitness.

(c) Two-dimensional Gaussian Mixture Model.

Figure 6.7: Validation example for two-dimensional data. Validation data (white points) and trained regional model (contour lines) should match. Better fitness is reflected in a lower value of the nLogL measure.



(a) Power spectra segments assigned to folds.

(b) Folds used for training.

(c) Remaining fold used for validation.

Figure 6.8: Data division in 10-fold Cross-Validation in STAGE 7.

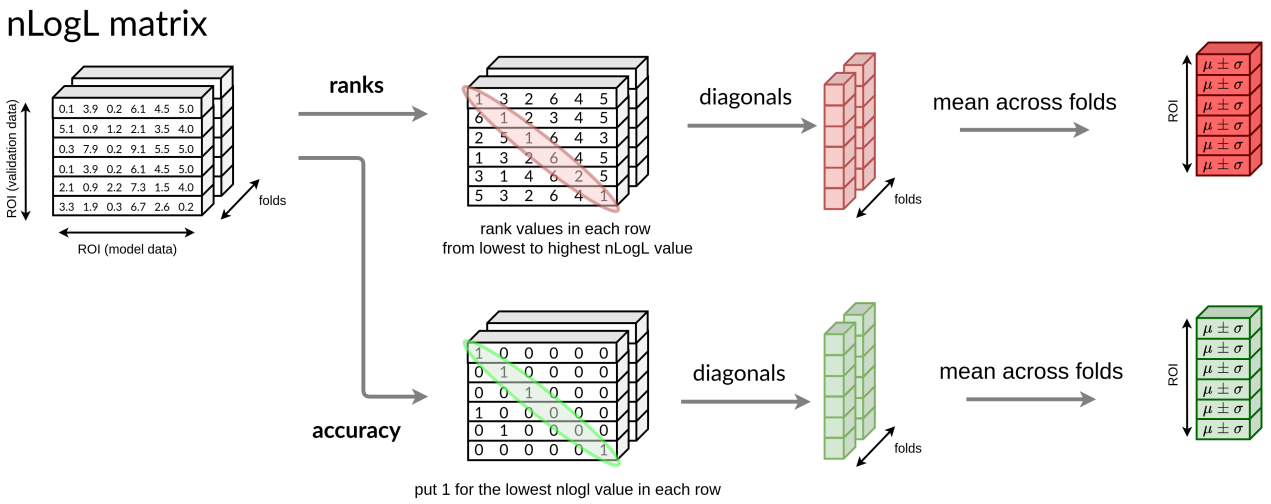


Figure 6.9: Fitness values (nLogL) between region pairs are obtained. Each row's values are ranked from the lowest to the highest. Each slice (array's 3rd dimension) gathered results when a particular fold was chosen to be a validation fold. Then, two measures - ranks and accuracies - are obtained. To calculate mean regional ranks and their standard deviations, diagonals are averaged across validation folds. The lower the rank value, the better identification of particular ROI. To calculate mean regional accuracies and their standard deviations, ranks > 1 are zeroed, then diagonals are averaged across validation folds. The higher the accuracy value, the better identification of particular ROI.

For the next iteration, another fold of data is selected as a validation fold and other serve as training folds. After k iterations single repetition of cross-validation is completed. Average regional accuracy/rank is then computed (Fig 6.9). One can set more than one repetition of cross-validation to be launched and the averaged results across repetitions will be obtained as well. Each repetition is different because power segments are assigned to the folds **randomly**. To conclude, below is the exemplary summary of calculations for STAGE 7.

Summary of the Individual-Level Identification

1. For r = 1 ... R repetitions ($R > 1$ is optional*):
   (a) Assign power spectra to 10 different sets (folds)
   (b) For each k-th fold:
      i. **Training**: Train model (individual fingerprint) of each brain region using power spectra from folds $\neq k$
      ii. **Validation**: For power spectra in k-th fold:
         - Calculate fit (**nLogL**) between each brain region validation data and each brain region's trained model
   (c) **Average** accuracy and rank **across folds** + compute standard deviation across folds
2. (Optional*) **Average** accuracy and rank **across repetitions** + compute standard deviation across repetitions

* - for justification see [45].

Network Analysis (STAGE 8).

This stage aims to compare spectral fingerprints obtained as a result of STAGES 1-5. Regional fingerprints are compared pairwise and successively joined to form a binary tree. This procedure utilizes the hierarchical clustering algorithm for this purpose. Details of this stage are presented below.

Network Analysis algorithm

1. **Input**: Set of subjects $\mathcal{S}$, individual fingerprints $\{\mathsf{IF}_u^s\}_{s\in\mathcal{S},u=1,\ldots,p}$, spectral fingerprints $\{\mathsf{SF}_u\}_{u=1,\ldots,p}$, u indexing brain regions (ROIs).
2. Define pooled clusters set:

$$\mathsf{PC}_u^s := \{\mu_h^{s,u}\}_{h=1}^k, \qquad u = 1,\ldots,p, \quad s \in \mathcal{S},$$
$$\mathsf{PC}_u := \bigcup_{s\in\mathcal{S}} \mathsf{PC}_u^s, \qquad u = 1,\ldots,p.$$

   where $\mu_h^{s,u} \in \mathbb{R}^F$ denotes spectral mode from subject s and region u, for F-frequencies of interest, $k \in \mathbb{N}$ Define negative log-likelihood matrix:

$$\mathsf{NL}_{u,v} := -\sum_{\mu\in\mathsf{PC}_u} \log\big(\mathrm{PDF}^v(\mu)\big), \quad u,v = 1,\ldots,p, \tag{6.1}$$

   where $\mathrm{PDF}^v(x) := \sum_{j=1}^k \alpha_j^v \phi\big(x \mid \mu_j^v, \Sigma_j^v\big)$ is the probability density function of the k-Gaussian components given by the spectral fingerprint of the v-th brain region $\mathsf{SF}_v = \big(\mu_i^v, \Sigma_i^v, \alpha_i^v\big)_{i=1}^{k_v}$, where μ_i^v - Gaussian Mixture centroid, Σ_i^v - covariance matrix, α_i^v - Gaussian Mixture proportion coefficient, $\sum_{j=1}^k \alpha_j^v = 1$, k_v - number of spectra in v-th spectral fingerprint
3. Let NL_u be uth row of matrix $[\mathsf{NL}_{u,v}]_{u,v=1,\ldots,p}$. Compute cosine distance of rows as follows

$$D_{i,j} := d\big(\mathsf{NL}_i, \mathsf{NL}_j\big) = 1 - \frac{\mathsf{NL}_i \cdot \mathsf{NL}_j}{\|\mathsf{NL}_i\|\|\mathsf{NL}_j\|}, \quad i,j = 1,\ldots,p. \tag{6.2}$$

4. Obtain clusters $\{\mathcal{C}_h\}_{h\in H}$ by performing hierarchical clustering using distance matrix $[D_{i,j}]_{i,j=1,\ldots,p}$ with UPGMA (unweighted pair group method with arithmetic mean) as an algorithm for computing the distance between clusters, where H is a set of indices of clusters for given tree truncation height γ.
5. **Output**: Clusters $\{\mathcal{C}_h\}_{h\in H}$.

7 TOOLBOX ARCHITECTURE

7.1 Parts of the software

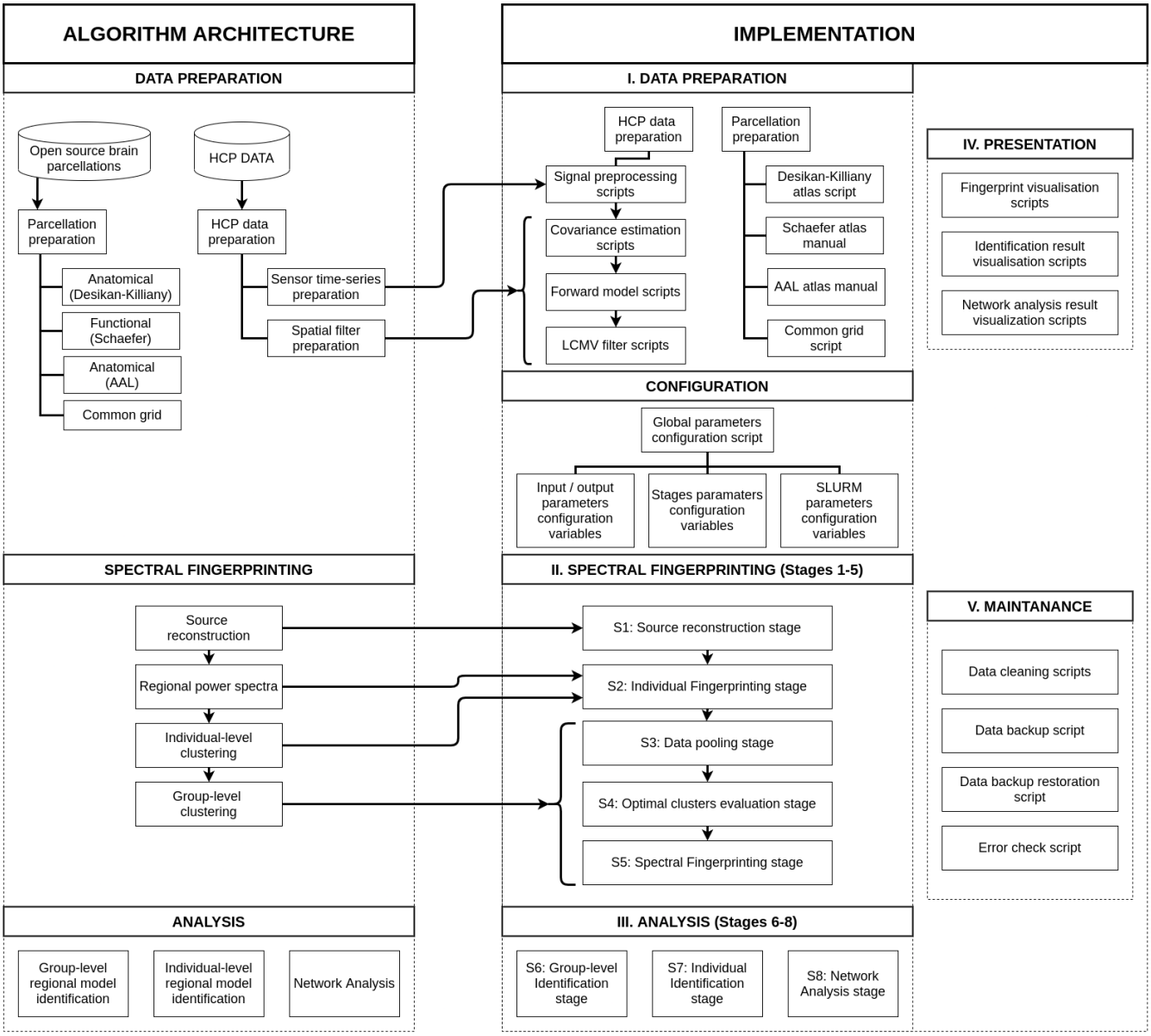


Figure 7.1: ToFFi toolbox architecture.

ToFFi is a modular piece of software that consists of the following components:

I. Data Preparation,

II. Spectral Fingerprinting,

III. Analysis,

IV. Presentation,

V. Maintenance.

The Data Preparation module (I) is responsible for arranging sensor time-series, spatial filters, and brain parcellation data for processing by the input of the part II routines.

The Spectral Fingerprinting (II) transforms MEG/EEG multichannel array of signals through five stages into

spatially localized power spectrum-driven representations called spectral fingerprints. Fourier Transform, source reconstruction (beamforming) and Gaussian Mixture Modeling algorithms are used to compute spectral fingerprints both at the individual and the group level (Fig. 4.5, Fig. 4.6).

The third component (III) consists of additional routines that can perform analysis on particular output files from component II. Currently, we have implemented group-level brain regions identification, individual-level brain regions identification, and regional clustering (network analysis) - all based on the concept of modeling brain activity as spectral fingerprints.

The presentation component (IV) is a collection of auxiliary scripts used to visualize particular results of performed computations for easier interpretation.

Maintenance routines (V) are used to automate some parts of the workflow, e.g. manage configuration files, manage output data files, etc.

7.2 Repository directories structure

The most important directories in the repository are shown below.

NOTE: `$REPO_ROOT`

A path to the directory containing directories listed below is called the **`$REPO_ROOT`**.

```
.
├── backup_output.sh                          <--- (V - Maintenance module)
├── CFG_illustrative.sh                       <--- configuration scripts to be run under
│                                                  Linux in order to launch
│                                                  the illustrative example
├── CFG_illustrative_serial.sh                <--- ditto, but w/o using multiple CPUs
├── CFG_illustrative_tryton.sh                <--- ditto, but for SLURM-managed clusters
├── check_errors.sh                           <--- (V - Maintenance module)
├── commonData                                <--- contains atlases, source models and
│                                                  other data relevant for all the stages
├── CONFIGURE.m                               <--- configuration script
├── DATA_PREPARATION                          <--- contains routines to prepare downloaded
│   │                                              HCP data for the Spectral Fingerprinting
│   │                                              (I - Data Preparation module)
│   ├── HCP_DATA_PREP
│   │   ├── 01_PREPROCESSING
│   │   ├── 02_COVARIANCE_GEN
│   │   ├── 03_LEADFIELD_GEN
│   │   ├── 04_LCMV_FILTER_GEN
│   │   ├── 05_RENAMING_TOOL
│   │   ├── CommonInit.m
│   │   ├── instruction.org
│   │   └── output
│   └── PARCELLATIONS_PREP                    <--- contains instuctions/scripts to prepare
│       │                                          brain atlases in order to segment source
│       │                                          volume into ROI
│       │                                          (I - Data Preparation module)
│       ├── AAL116_preparation.org
│       ├── common_grid_prep.m
│       ├── Common_grids_preparation.org
│       ├── DK_individual_atlases
│       └── Schaefer100_preparation.org
├── docs
```

(verte ...)

(... continued from the previous page)

```
│   ├── FUNCTIONS_REFERENCE.html
    └── ToFFi_Manual.pdf                          <--- this document!
├── ext_tools                                     <--- contains external tools and scripts
├── globalFunctionsScripts                        <--- contains scripts and functions used in
│                                                      more than one stage
├── PRESENTATION                                  <--- contains scripts and function used to
│   │                                                  visualize computed results
│   │                                                  (IV - Presentation module)
│   ├── common_functions
│   ├── fingerprints
│   ├── identification
│   ├── network_analysis
├── restore_from_backup.sh                        <--- (V - Maintenance module)
├── rm_logs_autosaves.sh                          <--- (V - Maintenance module)
├── rm_output_data.sh                             <--- (V - Maintenance module)
├── RUN_SELECTED.m                                <--- script used to launch STAGES 1-8 under
│                                                      single computer equipped with one
│                                                      or more CPU cores
│                                                      (V - Maintenance module)
├── RUN_SELECTED_SLURM.sh                         <--- script used to launch STAGES 1-8 under
│                                                      multiple computers equipped with one
│                                                      or more CPU cores (Linux only; SLURM only)
│                                                      (V - Maintenance module)
├── STAGE_1                                       <--- scripts, functions and output of the
│   │                                                  source analysis stage
│   │                                                  (II - Spectral Fingerprinting module)
│   ├── functions
│   ├── output
│   └── scripts
├── STAGE_1.sl
├── STAGE_1_Source_Analysis.info
├── STAGE_2                                       <--- scripts, functions and output of the
│   │                                                  Individual Fingerprinting stage
│   │                                                  (II - Spectral Fingerprinting module)
│   ├── functions
│   ├── output
│   └── scripts
├── STAGE_2_Individual_SF.info
├── STAGE_2INT.sl
├── STAGE_2.sl
├── STAGE_3                                       <--- scripts, functions and output of the
│   │                                                  cluster pooling stage
│   │                                                  (II - Spectral Fingerprinting module)
│   ├── functions
│   ├── output
│   └── scripts
├── STAGE_3_Pool.info
├── STAGE_3.sl
├── STAGE_4                                       <--- scripts, functions and output of the
│   │                                                  clustering evaluation stage
│   │                                                  (II - Spectral Fingerprinting module)
│   ├── functions
```

(verte ...)

(... continued from the previous page)

```
│   ├── output
│   └── scripts
├── STAGE_4_Eval_Clusters.info
├── STAGE_4INT.sl
├── STAGE_4.sl
├── STAGE_5                                  <--- scripts, functions and output of the
│   │                                             group fingerprinting stage
│   │                                             (II - Spectral Fingerprinting module)
│   ├── functions
│   ├── output
│   └── scripts
├── STAGE_5_Group_SF.info
├── STAGE_5.sl
├── STAGE_6                                  <--- scripts, functions and output of the
│   │                                             group-level brain regions identification
│   │                                             stage
│   │                                             (III - Analysis module)
│   ├── functions
│   ├── output
│   └── scripts
├── STAGE_6_Classification.info
├── STAGE_6INT.sl
├── STAGE_6.sl
├── STAGE_7                                  <--- scripts, functions and output of the
│   │                                             individual-level brain regions
│   │                                             identification stage
│   │                                             (III - Analysis module)
│   ├── functions
│   ├── output
│   └── scripts
├── STAGE_7_Individual_Classification.info
├── STAGE_7.sl
├── STAGE_8                                  <--- scripts, functions and output of the
│   │                                             network analysis stage
│   │                                             (III - Analysis module)
│   ├── functions
│   ├── output
│   └── scripts
├── STAGE_8_Network_Analysis.info
├── STAGE_8.sl
└── templates
```

7.3 Input data directories structure

Prepared data should be organized according to the exemplary directory structure shown below. Target directories and files can have different names as one can point to them during configuration (see 7.4.2 Configuration).

- multichannel signal files, spatial filter files (containing `data` and `spatialFilter` variables, respectively)

```
my_prepared_data_dir
├── Sub_1
│      ├── data_clean_1.mat
│      └── flt_1.mat
├── Sub_2
│      ├── data_clean_2.mat
│      └── flt_2.mat
│
...
│
└── Sub_N
       ├── data_clean_N.mat
       └── flt_N.mat
```

- group-level brain atlas files (containing `sourceAtlas` variable)

```
my_atlas_dir
└── atlas.mat
```

- individual* brain atlas files (containing `sourceAtlas` variable)

```
my_ind_atlas_dir
├── ind_atlas_1.mat
├── ind_atlas_1.mat
│
...
│
└── ind_atlas_1.mat
```

- common source grid (containing `sourcemodel` variable)

```
my_grid_dir
└── template_grid.mat
```

* - if the use of individual atlases instead of a group atlas is planned

7.4 Stages design

ToFFi Toolbox was designed as a modular piece of software. Each stage is a separate processing pipeline, with its own script and output, interwoven with common scripts placed in the `$REPO_ROOT/globalFunctionsScripts` directory. As a consequence, the structure of each stage is more or less common. It is reflected in the similarities between files structure, which is the topic of this section.

7.4.1 Stage-related directories and files

The Spectral Fingerprinting component consists of eight stages, each with a separate directory. Each `STAGE_X` directory consists of three subdirectories: `functions`, `output`, `scripts`, which names are self-explanatory. Inside a `scripts` directory there are some important files present that play a specific role:

- `RUN.m` - entry point for every job when starting given stage. It calls `CORE.m` script.
- `CORE.m` - contains a closed set of instructions, common for subjects / ROIs. It is a potential place for parallelization, so it is usually called on separate CPU cores at once.
- `INTEGRATE.m`* - this script is called by `RUN_SELECTED.m` or `RUN_SELECTED_SLURM.m` scripts at the end of processing STAGES 2, 4 or 6.

* - exists only for STAGES 2,4 and 6.

One of the most important files in the `output` directory is `CFG.mat` which is created during the configuration.

Output data should be organized according to the exemplary directory structure shown below. Target directories and files can have different names as one can point to them during configuration (see 7.4.2 Configuration). It is also possible to rename STAGES 1-8 output, however, it can be left as it is in default.

- output of the STAGES 1-2:

```
STAGE_1/2
└── output
    ├── Sub_1
    │   ├── dataX_1.mat
    │   └── dataY_1.mat
    ├── Sub_2
    │   ├── dataX_2.mat
    │   └── dataY_2.mat
    ...
    └── Sub_N
        ├── dataX_N.mat
        └── dataY_N.mat
```

- output of the STAGES 3-8:

```
STAGE_3/4/5/6/7/8
└── output
    ├── dataX.mat
    ├── dataY.mat
    └── dataZ.mat
```

7.4.2 Configuration

7.4.2.1 Proper syntax

This section describes how to fill the configuration file correctly in order to run the Spectral Fingerprinting. It will focus on specifically what one have to type after the equal signs in the configuration files, what brackets to use, where to put additional apostrophes, etc.

There are two ways of configuring the algorithm and thus two syntaxes possible:

- **Using .m file** (Fig. 7.2a). The first way is to write directly inside `CONFIGURE.m` file using MATLAB syntax. Then you need to run this script inside MATLAB. For computations managed by SLURM, one needs also to set all the parallel computing parameters inside each `.sl` file separately. This is a less convenient way of configuration, but suitable for all platforms (Windows, Linux, MacOS).
- **Using .sh file** (Fig. 7.2b). The second way is to copy any `CFG_*.sh` file and change variables using proper shell syntax. Then one needs to run this configuration under the shell. This way is suitable for Linux systems and convenient because one can also set SLURM configuration in the very same `.sh` file as well, instead of going through individual `.sl` files.

NOTE: Spaces

One should pay special attention to the spaces when reading examples for `.sh` files. Shell syntax is ruthless, while MATLAB is more liberal in that matter.

Below there are some notes regarding syntax required for different types of variables (strings, vectors, sequences, paths, etc.).

Path There are two types of paths: *absolute* and *relative*. Some configuration variables need to be written strictly as absolute paths and for other variables there is no difference whether one writes it in an *absolute* or *relative* fashion.

- Writing absolute paths:
  - `.m`: `CFG.Global.fieldtripPath = '/home/johndoe/fieldtrip-20210816/'`

(Windows: `'D:/johndoe/fieldtrip-20210816'`)

- `.sh`: `G_fieldtripPath='/home/johndoe/fieldtrip-20210816/'`
  (NO SPACES! Windows: `'D:/johndoe/fieldtrip-20210816'`)

- Writing relative (to the `$REPO_ROOT` directory) paths:
  - `.m`: `CFG.Global.fieldtripPath = '../fieldtrip-20210816/'`
    (same for Windows)
  - `.sh`: `G_fieldtripPath='../fieldtrip-20210816/'`
    (NO SPACES! Same for Windows)

- Writing paths combining other path variables:
  - `.m`: `CFG.Global.atlasPath = [CFG.Global.fieldtripPath, 'template/atlas/aal/ROI_MNI_V4.nii']` (square brackets to concatenate)
  - `.sh`: `G_atlasPath=$G_fieldtripPath'template/atlas/aal/ROI_MNI_V4.nii'`
    (NO SPACES! '$' sign is obligatory)

NOTE: Obligatory absolute paths

NOTE! The following variables need to be written using the *absolute* syntax (`.sh` equivalent in brackets):
- `CFG.Global.rootDir` (`G_rootDir`),
- `CFG.Global.veryFirstInputDataDir` (`G_veryFirstInputDataDir`),
- `CFG.Global.fieldtripPath` (`G_fieldtripPath`).

Single number This syntax is used for example for the following variables: `CFG.(STAGE_NAME).doZeroShift` (`S2_doZeroShift`), `CFG.(STAGE_NAME).rngSeed` (`S5_rngSeed`), `CFG.(STAGE_NAME).gaussianMixtureRegularization` (`S6_gaussianMixtureRegularization`).

- `.m`: `CFG.(STAGE_NAME).rngSeed = 2021`
- `.sh`: `S5_rngSeed=2021`
  (NO SPACES!)

Single string This syntax is used for example for the following variables: `CFG.Global.atlasType` (`G_atlasType`), `CFG.(STAGE_NAME).rngSeed` (`S5_rngSeed`), `CFG.(STAGE_NAME).linkageMethod` (`S8_linkageMethod`).

- `.m`: `CFG.(STAGE_NAME).rngSeed = 'time'`
  (string encapsuled in the apostrophes)
- `.sh`: `S5_rngSeed='time'`
  (NO SPACES! string encapsuled in the apostrophes)

Vector of numbers This syntax is used for example for the following variables: `CFG.Global.goodSubjects`, `CFG.(STAGE_NAME).frequenciesOfInterest` (`S1_frequenciesOfInterest`), `CFG.(STAGE_NAME).NumClustEval_kList` (`S2_NumClustEval_kList`).

- `.m`: `CFG.(STAGE_NAME).NumClustEval_kList = [2, 5, 18]`
  (square brackets; commas optional)
- `.sh`: `S2_NumClustEval_kList=(2 5 18)`
  (round brackets; no commas ! NO SPACES around equal sign!)

NOTE: Declaring sequences of numbers

You can use additional syntaxes to avoid entering long sequences of numbers by hand:

- sequence in `.m`: `CFG.Global.goodSubjects = 1:89`
  (':' operator)
- sequence in `.sh`: ``G_goodSubjects=(`seq 1 89`)``
  (`seq` function wrapped in backticks; NO SPACES around equal sign!)
- linspace in `.m`: `CFG.(STAGE_NAME).frequenciesOfInterest = linspace(1,120,50)`
- linspace in `.sh`: `S1_frequenciesOfInterest='linspace(1,120,50)'`
  (just enclose MATLAB command in apostrophes; NO SPACES!)
- logspace in `.m`: `CFG.(STAGE_NAME).frequenciesOfInterest = logspace(0,log10(120),50)`
- logspace in `.sh`: `S1_frequenciesOfInterest='logspace(0,log10(120),50)'`
  (just enclose MATLAB command in apostrophes; NO SPACES!)

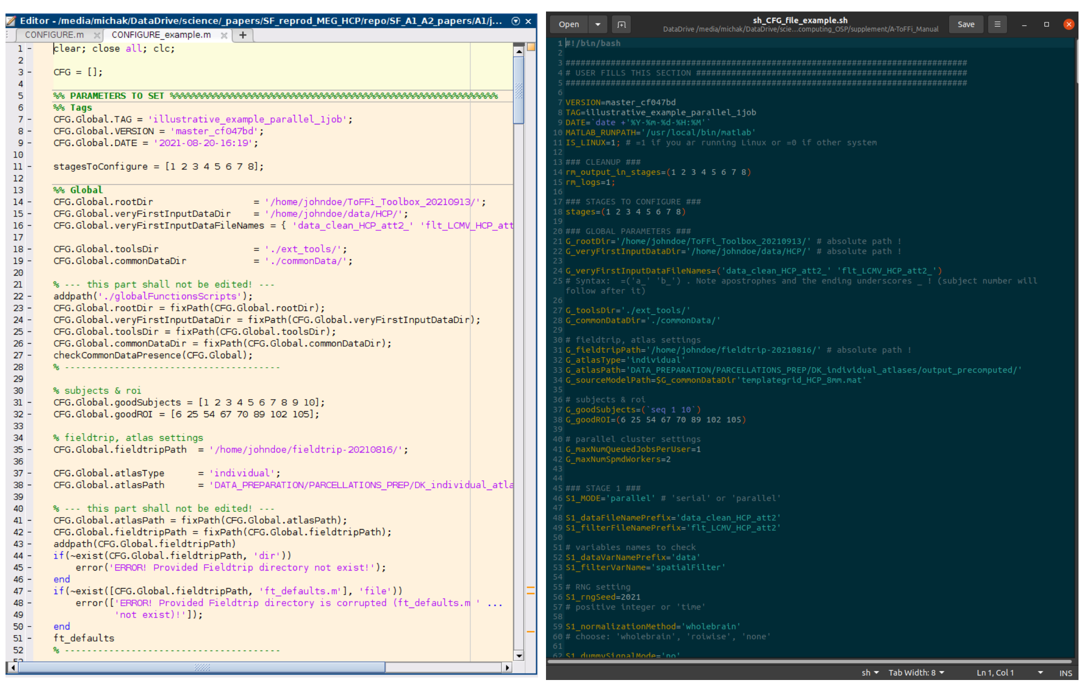
Vector of strings This syntax is used for example for the following variables: `CFG.Global.veryFirstInputDataFileNames` (`G_veryFirstInputDataFileNames`).

- `.m`: `CFG.Global.veryFirstInputDataFileNames = {'data_clean_HCP_att2_','flt_LCMV_HCP_att2_'}`
  (curly brackets; commas optional)
- `.sh`: `G_veryFirstInputDataFileNames=('data_clean_HCP_att2_' 'flt_LCMV_HCP_att2_')`
  (round brackets; no commas ! NO SPACES around equal sign!)

7.4.2.2 Global configuration file

The global configuration file contains the values of variables that can be set by the user. There are some global parameters, i.e. their value are used in all of the Stages 1-8, such as the list of brain regions of interest (`CFG.Global.goodROI`) or maximum number of CPU cores that can be used (`CFG.Global.maxNumSpmdWorkers`), as well as stage-specific variables.

On the next pages, there are tables with short descriptions of each variable that the user can set.



(a) Configuration using `CONFIGURE.m` file.

(b) Configuration using `.sh` file.

Figure 7.2: Comparison between two types of configuration files (see 7.4.2.1 Proper syntax).

Global

Variable name in CONFIGURE.m	Variable name in CFG_*.sh	Syntax	Role	Comment
CFG.Global. TAG	TAG	Single string	Configuration identifier for bookkeeping.	A short name that helps to recognize given configuration file among others.
CFG.Global. VERSION	VERSION	Single string	Another tag that helps to identify which code version was used.	For example, a commit hash.
CFG.Global. DATE	DATE	Single string	Date of running the configuration.	For Linux systems, it can be generated automatically when its value equals ‘date +’%Y-%m-%d-%H:%M‘‘.
n/a	MATLAB_RUNPATH	Path	Relative or absolute path to the matlab executable file.	It is usually in MATLAB/bin/ directory, e.g. ’/home/johndoe/MATLAB/bin/matlab’.
n/a	IS_LINUX	Single number	Flag for allowing automatic launch of the configuration within the shell.	1 - set for Linux when one wants to run CONFIGURE.m script automatically within the shell; 0 - otherwise.
n/a	rm_outputinstages	Vector of numbers	List of stages whose output directory is going to be cleaned during configuration.	Alternatively, one can run rm_outputdata.sh script.
n/a	rm_logs	Single number	Flag for allowing of log files and autosaves removal from the `$REPO_ROOT` directory.	1 - remove all .log, .out files from the `$REPO_ROOT` directory. 0 - do not remove. Alternatively, you can run rm_logsautosaves.sh script.
stagesToConfigure	stages	Vector of numbers	List of stages to be configured.	Possible values: 1, 2, . . . 8.
CFG.Global. rootDir	G_rootDir	Path	Absolute path to the `$REPO_ROOT` directory (directory containing the ToFFi Toolbox).	ABSOLUTE PATH!
CFG.Global. veryFirstInputDataDir	G_veryFirstInputDataDir	Path	Absolute path to the data files with sensor signals and spatial filters.	ABSOLUTE PATH! See 7.3 Input data directories structure section for details.
CFG.Global. veryFirstInputDataFileNames	G_veryFirstInputDataFileNames	Vector of strings	Name of the files containing sensor signals and spatial filters.	Vector of two strings. Strings should end with the underscore _ , e.g. {’data_’, ’filter_’} when using .m file or (’data_’ ’filter_’) when using .sh file. The subject index will be added after the underscore to identify subject-specific files. The first string is for data sensor signal files, and the second is for spatial filter files. See 7.3 Input data directories structure section and adjusted `data` / `spatialFilter` variables in 6.2 Diagram data description section for details.
CFG.Global. toolsDir	G_toolsDir	Path	Path to the directory containing external plugins and scripts.	
CFG.Global. commonDataDir	G_commonDataDir	Path	Path to the directory containing atlases and source models.	
CFG.Global. goodSubjects	G_goodSubjects	Vector of numbers	List of subjects to be processed.	These numbers should match the numbers in the directories.
CFG.Global. goodROI	G_goodROI	Vector of numbers	List of the ROIs to be processed.	These numbers should match the numbers in the selected atlas.
CFG.Global. fieldtripPath	G_fieldtripPath	Path	Absolute path to the directory containing The Fieldtrip Toolbox.	ABSOLUTE PATH!
CFG.Global. atlasType	G_atlasType	Single string	Type of the atlas to be used.	Consult globalFunctionsScripts/prepareAtlas.m file for details.
CFG.Global. atlasPath	G_atlasPath	Path	Path to the selected atlas file.	Consult globalFunctionsScripts/prepareAtlas.m file for details.
CFG.Global. sourceModelPath	G_sourceModelPath	Path	Path to the selected source model file.	Consult globalFunctionsScripts/prepareAtlas.m file for details.

Variable name in CONFIGURE.m	Variable name in CFG_*.sh	Syntax	Role	Comment
CFG.Global. maxNumQueuedJobsPerUser	G_maxNumQueuedJobsPerUser	Single number	A total number of jobs that one plans to run.	Used to calculate the maximum number of jobs per each stage. See globalFunctionsScripts/calculateMaxJobArraySize.m for details.
CFG.Global. maxNumSpmdWorkers	G_maxNumSpmdWorkers	Single number	Total number of CPU cores (MATLAB's spmds) that one plans to run.	Limits the number of spmds used by the MATLAB routines during computations. Used to calculate the maximum number of jobs per each stage. See globalFunctionsScripts/calculateMaxJobArraySize.m for details.

STAGE 1 (Source Analysis)

Variable name in CONFIGURE.m	Variable name in CFG_*.sh	Syntax	Role	Comment
CFG.STAGE_1. MODE	S1_MODE	Single string	Tells MATLAB if iterations / ROIs / subjects are spread among different computers and CPU cores.	'serial' - perform all computations sequentially, 'parallel' - allow to use multiple MATLAB sessions and spmds to perform parallel computations where possible.
CFG.STAGE_1. dataFileNamePrefix	S1_dataFileNamePrefix	Single string	Prefix of the file names containing sensor signal data.	Subject-related files are named following convention PREFIX_X.mat, where X is the ID of the subject, same as those in CFG.Global.goodSubjects variable.
CFG.STAGE_1. filterFileNamePrefix	S1_filterFileNamePrefix	Single string	A prefix of the file names containing spatial filter data.	Subject-related files are named following convention PREFIX_X.mat, where X is the ID of the subject, same as those in CFG.Global.goodSubjects variable.
CFG.STAGE_1. dataVarNamePrefix	S1_dataVarNamePrefix	Single string	Name of the variable loaded to the workspace from sensor signal data file.	
CFG.STAGE_1. filterVarName	S1_filterVarName	Single string	Name of the variable loaded to the workspace from a spatial filter data file.	
CFG.STAGE_1. rngSeed	S1_rngSeed	Single number / Single string	The seed value of the pseudo-random numbers generator for STAGE 1 routines.	Possible values: positive integer - precisely defined seed value, 'time' - seed value based on launch time of the current job. See 7.4.4 Pseudo-random numbers generator and reproducibility section for details.
CFG.STAGE_1. normalizationMethod	S1_normalizationMethod	Single string	Method of normalizing ROI power spectra.	Possible values: 'wholebrain', 'roiwise', 'none'. Consult STAGE_1/functions/normalizeActivity.m file.
CFG.STAGE_1. dummySignalMode	S1_dummySignalMode	Single string	Replaces real signal data with a synthetic signal, e.g. noise.	Possible values: 'no' - do not replace real signal data, 'wgn' - substitutes real signal with White Gaussian Noise, 'wgn-keepTrials' - substitutes real signal with White Gaussian Noise, but keeps trial structure and length intact.
CFG.STAGE_1. frequenciesOfInterest	S1_frequenciesOfInterest	Vector of numbers	List of frequencies in Hz.	The list of final frequencies could be changed due to the frequency resolution constraints. See note in section 6.3.3 for details.

STAGE 2 (Individual Fingerprint)

Variable name in CONFIGURE.m	Variable name in CFG__*.sh	Syntax	Role	Comment
CFG.STAGE__2. MODE	S2__MODE	Single string	Tells MATLAB if iterations / ROIs / subjects are spread among different computers and CPU cores.	'serial' - perform all computations sequentially, 'parallel' - allow to use multiple MATLAB sessions and spmds to perform parallel computations where possible.
CFG.STAGE__2. sourcesPowerFileNamePrefix	S2__sourcesPowerFileNamePrefix	Single string	Name of the subject-related output files from STAGE 1 (normalized sources power).	Subject-related files are named following convention PREFIX__X.mat, where X is the ID of the subject, same as those in CFG.Global.goodSubjects variable.
CFG.STAGE__2. dataVarNamePrefix	S2__dataVarNamePrefix	Single string	Name of the variable loaded to the workspace from normalized sources power data files (output of STAGE 1).	
CFG.STAGE__2. rngSeed	S2__rngSeed	Single number / Single string	The seed value of the pseudo-random numbers generator for STAGE 2 routines.	Possible values: positive integer - precisely defined seed value, 'time' - seed value based on launch time of the current job. See 7.4.4 Pseudo-random numbers generator and reproducibility section for details.
CFG.STAGE__2. trialRejectZ	S2__trialRejectZ	Single number	The threshold value for trial rejection based on z-score of averaged power. An amount of standard deviation to be exceeded in order to reject a given trial.	Default value: 2.5; Set to inf to omit rejection.
CFG.STAGE__2. doZeroShift	S2__doZeroShift	Single number	Performs the substraction of one (-1) from the spectral power values.	1 - substract one from spectral power of all frequencies, 0 - omit substraction. See Keitel & Gross 2016 "Individual Human Brain Areas Can Be Identified from Their Characteristic Spectral Activation Fingerprints" for rationale.
CFG.STAGE__2. saveMeanRoiPower	S2__saveMeanRoiPower	Single number	Decides whether to save intermediate output data structure containing mean ROI spectral power before starting clustering phase.	1 - save, 0 - do not save. NOTE! These additional files are used in STAGE 7!
CFG.STAGE__2. clusteringMethod	S2__clusteringMethod	Single string	Name of the hard clustering method to be used.	Currently only 'kmeans' value is supported. Possibility for other choices remains open for development.
CFG.STAGE__2.clustering. (CFG.STAGE__2.clusteringMethod). distanceMetric	S2__distanceMetric	Single string	Name of the method used to assess dissimilarity of the clustered points.	Possible choices: `https://www.mathworks.com/help/stats/kmeans.html?searchHighlight=kmeans&s_tid=srchtitle#buefs04-Distance`
CFG.STAGE__2.clustering. (CFG.STAGE__2.clusteringMethod). nReplicates	S2__nReplicates	Single number	The number of times to repeat clustering using new initial cluster centroid positions.	
CFG.STAGE__2. gaussianMixtureRegularization	S2__gaussianMixtureRegularization	Single number	The amount of regularization to be applied to covariance matrices during Gaussian Mixture Modelling (EM algorithm).	Real number, λ, between 0 and 1. At the end of each iteration of the EM algorithm, λ is added to every diagonal element of the covariance matrix (track 'regVal' variable inside 'gmcluster_learn.m' function inside 'gmcluster.m' routine called by 'gmdistribution.fit' inside 'fitgmdist.m' MATLAB function).
CFG.STAGE__2. numClustersMode	S2__numClustersMode	Single string	Approach to be used when presetting the number of clusters for each ROI to be extracted from normalized ROI power.	'fixed' - number of clusters to be defined in CFG.STAGE__2.nSpectralModesPerROI variable; 'optimal' - number of clusters will be optimally adjusted for each ROI separately.

Variable name in CONFIGURE.m	Variable name in CFG_*.sh	Syntax	Role	Comment
CFG.STAGE_2. nSpectralModesPerROI	S2_nSpectralModesPerROI	Single number	Number of clusters to be extracted from normalized ROI power.	All ROIs will have the same number of clusters extracted. Ignored if CFG.STAGE_2.numClustersMode mode is set to 'optimal'.
CFG.STAGE_2. NumClustEval_kList	S2_NumClustEvalkList	Vector of numbers	List of number of clusters to evaluate. A vector of positive integer values.	Used when CFG.STAGE_2.numClustersMode mode is set to 'optimal'.
CFG.STAGE_2. NumClustEval_nIterations	S2_NumClustEvalnIterations	Single number	How many times the clustering evaluation algorithm should be repeated.	Used when CFG.STAGE_2.numClustersMode mode is set to 'optimal'.
CFG.STAGE_2. NumClustEval_criterionType	S2_NumClustEvalcriterionType	Single string	Name of the criterion for clustering evaluation to be used.	Used when CFG.STAGE_2.numClustersMode mode is set to 'optimal'. Possible options: `https://www.mathworks.com/help/stats/evalclusters.html#shared-criterion`

STAGE 3 (Pooling)

Variable name in CONFIGURE.m	Variable name in CFG_*.sh	Syntax	Role	Comment
CFG.STAGE_3. dataFileNamePrefix	S3_dataFileNamePrefix	Single string	Name of the subject-related output file from STAGE 2 (individual fingerprint file).	Subject-related files are named following convention PREFIX_X.mat, where X is the ID of the subject, same as those in CFG.Global.goodSubjects variable.
CFG.STAGE_3. dataVarNamePrefix	S3_dataVarNamePrefix	Single string	Name of the variable loaded to the workspace from individual fingerprint data files (output of STAGE 2).	

STAGE 4 (Optimal Clustering Evaluation)

Variable name in CONFIGURE.m	Variable name in CFG_*.sh	Syntax	Role	Comment
CFG.STAGE_4. MODE	S4_MODE	Single string	Tells MATLAB if iterations / ROIs / subjects are spread among different computers and CPU cores.	'serial' - perform all computations sequentially, 'parallel' - allow using multiple MATLAB sessions and spmds to perform parallel computations where possible.
CFG.STAGE_4. pooledClustersFileNamePrefix	S4_pooledClustersFileNamePrefix	Single string	Name of the output file from STAGE 3 containing pooled clusters from all individual fingerprints.	
CFG.STAGE_4. pooledClustersVarNamePrefix	S4_pooledClustersVarNamePrefix	Single string	Name of the variable loaded to the workspace from pooled clusters (output of STAGE 3).	
CFG.STAGE_4. rngSeed	S4_rngSeed	Single number / Single string	The seed value of the pseudo-random numbers generator for STAGE 4 routines.	Possible values: positive integer - precisely defined seed value, 'time' - seed value based on launch time of the current job. See 7.4.4 Pseudo-random numbers generator and reproducibility section for details.
CFG.STAGE_4. methodName	S4_methodName	Single string	Name of the clustering method used in the evaluation of optimal number of clusters for spectral fingerprints to be computed in STAGE 5.	Possible options: `https://www.mathworks.com/help/stats/evalclusters.html#bt0oocm_sep_shared-clust`
CFG.STAGE_4. criterionType	S4_criterionType	Single string	Name of the criterion for clustering evaluation to be used.	Possible options: `https://www.mathworks.com/help/stats/evalclusters.html#shared-criterion`

Variable name in CONFIGURE.m	Variable name in CFG__*.sh	Syntax	Role	Comment
CFG.STAGE__4. distanceMetric	S4__distanceMetric	Single string	Name of the method used to assess dissimilarity of the clustered points.	Possible choices: `https://www.mathworks.com/help/stats/evalclusters.html#bt0oocm_sep_shared-Distance`
CFG.STAGE__4. kList	S4__kList	Vector of numbers	List of number of clusters to evaluate. A vector of positive integer values.	
CFG.STAGE__4. nIterations	S4__nIterations	Single number	How many times the clustering evaluation algorithm should be repeated.	

STAGE 5 (Spectral Fingerprints)

Variable name in CONFIGURE.m	Variable name in CFG__*.sh	Syntax	Role	Comment
CFG.STAGE__5. MODE	S5__MODE	Single string	Tells MATLAB if iterations / ROIs / subjects are spread among different computers and CPU cores.	'serial' - perform all computations sequentially, 'parallel' - allow to use multiple MATLAB sessions and spmds to perform parallel computations where possible.
CFG.STAGE__5. pooledClustersFileNamePrefix	S5__pooledClustersFileNamePrefix	Single string	Name of the output file from STAGE 3 containing pooled clusters from all individual fingerprints.	
CFG.STAGE__5. clusteringEvaluationFileNamePrefix	S5__clusteringEvaluationFileNamePrefix	Single string	Name of the output file from STAGE 4 containing clustering evaluation result.	
CFG.STAGE__5. pooledClustersVarNamePrefix	S5__pooledClustersVarNamePrefix	Single string	Name of the variable loaded to the workspace from pooled clusters (output of STAGE 3).	
CFG.STAGE__5. clusteringEvaluationVarNamePrefix	S5__clusteringEvaluationVarNamePrefix	Single string	Name of the variable loaded to the workspace from clustering evaluation result (output of STAGE 4).	
CFG.STAGE__5. rngSeed	S5__rngSeed	Single number / Single string	The seed value of the pseudo-random numbers generator for STAGE 5 routines.	Possible values: positive integer - precisely defined seed value, 'time' - seed value based on launch time of the current job. See 7.4.4 Pseudo-random numbers generator and reproducibility section for details.
CFG.STAGE__5. majoritySubjectsNum	S5__majoritySubjectsNum	Single number	Clusters whose points belong to at least that number of subjects will be kept. The rest will be marked for rejection.	Set =1 for accepting all clusters.
CFG.STAGE__5. clusteringMethod	S5__clusteringMethod	Single string	Name of the hard clustering method to be used.	Currently only 'kmeans' value is supported. Possibility for other choices remains open for development.
CFG.STAGE__5. numberOfClusters	S5__numberOfClusters	Single number / Single string	Approach to be used when presetting the number of clusters for each ROI to be extracted from pooled clusters.	Possible values: positive integer - precisely defined number of clusters (same for all ROIs); 'optimal' - number of clusters will be optimally adjusted for each ROI separately.
CFG.STAGE__5. distanceMetric	S5__distanceMetric	Single string	Name of the method used to assess dissimilarity of the clustered points.	Possible choices: `https://www.mathworks.com/help/stats/kmeans.html?searchHighlight=kmeans&s_tid=srchtitle#buefs04-Distance`
CFG.STAGE__5.clustering. (CFG.STAGE__5.clusteringMethod). nReplicates	S5__nReplicates	Single number	The number of times to repeat clustering using new initial cluster centroid positions.	

Variable name in CONFIGURE.m	Variable name in CFG__*.sh	Syntax	Role	Comment
CFG.STAGE__5. gaussianMixtureRegularization	S5__gaussianMixtureRegularization	Single number	The amount of regularization to be applied to covariance matrices during Gaussian Mixture Modelling (EM algorithm).	Real number, λ, between 0 and 1. At the end of each iteration of the EM algorithm, λ is added to every diagonal element of the covariance matrix (track 'regVal' variable inside 'gmcluster__learn.m' function inside 'gmcluster.m' routine called by 'gmdistribution.fit' inside 'fitgmdist.m' MATLAB function).

STAGE 6 (Group Identification)

Variable name in CONFIGURE.m	Variable name in CFG__*.sh	Syntax	Role	Comment
CFG.STAGE__6. MODE	S6__MODE	Single string	Tells MATLAB if iterations / ROIs / subjects are spread among different computers and CPU cores.	'serial' - perform all computations sequentially, 'parallel' - allow to use multiple MATLAB sessions and spmds to perform parallel computations where possible.
CFG.STAGE__6. dataFileNamePrefix	S6__dataFileNamePrefix	Single string	Name of the output file from STAGE 3 containing pooled clusters from all individual fingerprints.	
CFG.STAGE__6. dataVarNamePrefix	S6__dataVarNamePrefix	Single string	Name of the variable loaded to the workspace from pooled clusters (output of STAGE 3).	
CFG.STAGE__6. rngSeed	S6__rngSeed	Single number / Single string	The seed value of the pseudo-random numbers generator for STAGE 6 routines.	Possible values: positive integer - precisely defined seed value, 'time' - seed value based on launch time of the current job. See 7.4.4 Pseudo-random numbers generator and reproducibility section for details.
CFG.STAGE__6. nRepetitions	S6__nRepetitions	Single number	Number of repetitions of the cross-validation.	
CFG.STAGE__6. nFolds	S6__nFolds	Single number	Number of folds of the cross-validation.	
CFG.STAGE__6. save__NLLMatrices	S6__saveNLLMatrices	Single number	Decides whether to save intermediate output data structure containing negative log-likelihood matrices.	1 - save, 0 - do not save.
CFG.STAGE__6. nClustersSetting	S6__nClustersSetting	Single string	Decides what number of clusters should training ROI models have.	Possible values: 'mode' - number of clusters equal to the most frequent number of clusters across training subjects (for each ROI separately); 'optimal' - number of clusters for each ROI training model will be the same as it was calculated in STAGE__4; 'fixed' - all ROI training models will get the same number of clusters precisely set in CFG.STAGE__6.fixed__nClusters variable.
CFG.STAGE__6. fixed__nClusters	S6__fixed__nClusters	Single number	The number of clusters for training models. Each ROI training model will have the same number of clusters.	Ignored if CFG.STAGE__6.nClustersSetting mode is set to 'mode' or 'optimal'.
CFG.STAGE__6. clusteringMethod	S6__clusteringMethod	Single string	Name of the hard clustering method to be used.	Should have the same value as CFG.STAGE__2.clusteringMethod.
CFG.STAGE__6.clustering. (CFG.STAGE__6.clusteringMethod). distanceMetric	S6__distanceMetric	Single string	Name of the method used to assess dissimilarity of the clustered points.	Should have the same value as CFG.STAGE__2.clustering. (CFG.STAGE__2.clusteringMethod). distanceMetric.

Variable name in CONFIGURE.m	Variable name in CFG_*.sh	Syntax	Role	Comment
CFG.STAGE_6.clustering. (CFG.STAGE_6.clusteringMethod). nReplicates	S6_nReplicates	Single number	The number of times to repeat clustering using new initial cluster centroid positions.	Should have the same value as CFG.(STAGE_2).clustering. (CFG.(STAGE_2).clusteringMethod). nReplicates.
CFG.STAGE_6. gaussianMixtureRegularization	S6_gaussianMixtureRegularization	Single number	The amount of regularization to be applied to covariance matrices during Gaussian Mixture Modelling (EM algorithm).	Should have the same value as CFG.STAGE_2.gaussianMixtureRegularization.
CFG.STAGE_6. majoritySubjectNum	S6_majoritySubjectNum	Single number	Clusters whose points belong to at least that number of subjects will be kept. The rest will be marked for rejection.	Set =1 for accepting all clusters
CFG.STAGE_6. optimalClustersDataFile	S6_optimalClustersDataFile	Single string	Name of the output file from STAGE 4 containing clustering evaluation result.	Ignored if CFG.STAGE_6.nClustersSetting mode is set to 'fixed' or 'mode'.
CFG.STAGE_6. optimalClustersVarName	S6_optimalClustersVarName	Single string	The name of the variable loaded to the workspace from clustering evaluation result (output of STAGE 4).	Ignored if CFG.STAGE_6.nClustersSetting mode is set to 'fixed' or 'mode'.

STAGE 7 (Individual Identification)

Variable name in CONFIGURE.m	Variable name in CFG_*.sh	Syntax	Role	Comment
CFG.STAGE_7. MODE	S7_MODE	Single string	Tells MATLAB if iterations / ROIs / subjects are spread among different computers and CPU cores.	'serial' - perform all computations sequentially, 'parallel' - allow to use multiple MATLAB sessions and spmds to perform parallel computations where possible.
CFG.STAGE_7. dataFileNamePrefix	S7_dataFileNamePrefix	Single string	Name of the subject-related output files from STAGE 2 (singleSubjectPowerData files).	NOTE! Files will be present only if CFG.STAGE_2.saveMeanRoiPower was set to 1 ! Subject-related files are named following convention PREFIX_X.mat, where X is the ID of the subject, same as those in CFG.Global.goodSubjects variable.
CFG.STAGE_7. dataVarNamePrefix	S7_dataVarNamePrefix	Single string	Name of the variable loaded to the workspace from subject-related output files from STAGE 2 (singleSubjectPowerData files).	NOTE! Files will be present only if CFG.STAGE_2.saveMeanRoiPower was set to 1 ! Subject-related files are named following convention PREFIX_X.mat, where X is the ID of the subject, same as those in CFG.Global.goodSubjects variable.
CFG.STAGE_7. rngSeed	S7_rngSeed	Single number / Single string	The seed value of the pseudo-random numbers generator for STAGE 7 routines.	Possible values: positive integer - precisely defined seed value, 'time' - seed value based on launch time of the current job. See 7.4.4 Pseudo-random numbers generator and reproducibility section for details.
CFG.STAGE_7. nRepetitions	S7_nRepetitions	Single number	The number of repetitions of the cross-validation.	
CFG.STAGE_7. nFolds	S7_nFolds	Single number	The number of folds of the cross-validation.	
CFG.STAGE_7. nClustersSetting	S7_nClustersSetting	Single string	Decides what number of clusters should training ROI models have.	Possible values: 'optimal' - number of clusters will be optimally adjusted for each ROI separately; 'fixed' - all ROI training models will get the same number of clusters precisely set in CFG.STAGE_7.fixed_nClusters variable.

Variable name in CONFIGURE.m	Variable name in CFG__*.sh	Syntax	Role	Comment
CFG.STAGE__7. fixed__nClusters	S7__fixed__nClusters	Single number	Number of clusters for training models. Each ROI training model will have the same number of clusters.	Ignored if CFG.STAGE__7.nClustersSetting mode is set to 'optimal'.
CFG.STAGE__7. clusteringMethod	S7__clusteringMethod	Single string	Name of the hard clustering method to be used.	Should have the same value as CFG.STAGE__2.clusteringMethod.
CFG.STAGE__7.clustering. (CFG.STAGE__7.clusteringMethod). distanceMetric	S7__distanceMetric	Single string	Name of the method used to assess dissimilarity of the clustered points.	Should have the same value as CFG.STAGE__2.clustering. (CFG.STAGE__2.clusteringMethod). distanceMetric.
CFG.STAGE__7.clustering. (CFG.STAGE__7.clusteringMethod). nReplicates	S7__nReplicates	Single number	The number of times to repeat clustering using new initial cluster centroid positions.	Should have the same value as CFG.(STAGE__2).clustering. (CFG.(STAGE__2).clusteringMethod). nReplicates.
CFG.STAGE__7. gaussianMixtureRegularization	S7__gaussianMixtureRegularization	Single number	The amount of regularization to be applied to covariance matrices during Gaussian Mixture Modelling (EM algorithm).	Should have the same value as CFG.STAGE__2.gaussianMixtureRegularization.
CFG.STAGE__7. NumClustEval__kList	S7__NumClustEvalkList	Vector of numbers	List of number of clusters to evaluate. A vector of positive integer values.	Used when CFG.STAGE__7.nClustersSetting mode is set to 'optimal'.
CFG.STAGE__7. NumClustEval__nIterations	S7__NumClustEvalnIterations	Single number	How many times the clustering evaluation algorithm should be repeated.	Used when CFG.STAGE__7.nClustersSetting mode is set to 'optimal'.
CFG.STAGE__7. NumClustEval__criterionType	S7__NumClustEvalcriterionType	Single string	Name of the criterion for clustering evaluation to be used.	Used when CFG.STAGE__7.nClustersSetting mode is set to 'optimal'.

STAGE 8 (Network Analysis)

Variable name in CONFIGURE.m	Variable name in CFG__*.sh	Syntax	Role	Comment
CFG.STAGE__8. MODE	S8__MODE	Single string	Tells MATLAB if iterations / ROIs / subjects are spread among different computers and CPU cores.	'serial' - perform all computations sequentially, 'parallel' - allow to use multiple MATLAB sessions and spmds to perform parallel computations where possible.
CFG.STAGE__8. pooledClustersFileNamePrefix	S8__pooledClustersFileNamePrefix	Single string	Name of the output file from STAGE 3 containing pooled clusters from all individual fingerprints.	
CFG.STAGE__8. spectralFingerprintsFileNamePrefix	S8__spectralFingerprintsFileNamePrefix	Single string	Name of the ROI-related output files from STAGE 5 (spectral fingerprints).	ROI-related files are named following convention PREFIX__Y.mat, where Y is the ID of the ROI, same as those in CFG.Global.goodROI variable.
CFG.STAGE__8. pooledClustersVarNamePrefix	S8__pooledClustersVarNamePrefix	Single string	Name of the variable loaded to the workspace from pooled clusters (output of STAGE 3).	
CFG.STAGE__8. spectralFingerprintsVarNamePrefix	S8__spectralFingerprintsVarNamePrefix	Single string	Name of the variable loaded to the workspace from spectral fingerprinting computation result (output of STAGE 5).	
CFG.STAGE__8. rngSeed	S8__rngSeed	Single number / Single string	The seed value of the pseudo-random numbers generator for STAGE 1 routines.	Possible values: positive integer - precisely defined seed value, 'time' - seed value based on launch time of the current job. See 7.4.4 Pseudo-random numbers generator and reproducibility section for details.
CFG.STAGE__8. linkageMethod	S8__linkageMethod	Single string	Algorithm for computing distance between clusters.	Possible choices: `https://www.mathworks.com/help/stats/linkage.html` ('method' argument)

Variable name in CONFIGURE.m	Variable name in CFG__*.sh	Syntax	Role	Comment
CFG.STAGE__8. linkageDistanceMetric	S8__linkageDistanceMetric	Single string	Name of the method used to assess dissimilarity of the clustered points.	Possible choices: `https://www.mathworks.com/help/stats/linkage.html` (‘metric‘ argument)
CFG.STAGE__8. nSimilarityClusters	S8__nSimilarityClusters	Single number	The number of clusters to be formed / maximal hierarchical tree deepness.	It is precisely the P parameter from the ‘dendrogram.m‘ function: `https://www.mathworks.com/help/stats/dendrogram.html`
CFG.STAGE__8. majoritySubjectsNum	S8__majoritySubjectsNum	Single number	Clusters whose points belong to at least that number of subjects will be kept. The rest will not take part in the clustering process.	Set =1 for accepting all clusters.

SLURM-specific variables

Variable name in CFG_*.sh file	Equivalent line in .sl file	Syntax	Role	Comment
SX_SLURMjobName	#SBATCH -J	Single string	Job name.	Helps to identify job when checking status with ‘squeue‘ command.
SX_SLURMqueueName	#SBATCH -p	Single string	Partition name (a.k.a queue name) that job will run on.	
SX_SLURMnTasksPerNode	#SBATCH –ntasks-per-node	Single number	Number of CPU cores to be used by jobs.	Should be maximum number possible on the machine, because it will be eventually limited by the value of ‘CFG.Global.maxNumQueuedJobsPerUser‘ variable
SX_SLURMmem	#SBATCH –mem	Single number	Amount of memory in MB to be reserved by each job.	
SX_SLURMtime	#SBATCH –time	Single string	Time limit for each job.	Format: ’HH:MM:SS’
SX_SLURMmailType	#SBATCH –mail-type	Single string	Notify by email when certain event types occur.	See `https://slurm.schedmd.com/sbatch.html`
SX_SLURMarray	#SBATCH –array	Single string	Range of jobs to be run.	Format: ’1-N’, where N is the number of jobs to be run.

X - denotes stage number (1-8).

7.4.3 Data distribution across the stages

When one wants to think about how computation time changes in relation to data size, it is helpful to know the loop structure used in a particular stage. These structures are depicted below.

For example, look a the loop structure of STAGE 6: it performs cross-validation R-times (outer loop). For each repetition, it needs to process a set of L-regions (ROIs; inner loop), and for each ROI, it needs to validate it against another L-regions (second inner loop). It means that the computational complexity of this particular stage is $\mathcal{O}(R * L^2)$. Increasing the number of CV-repetitions two times will double the computation time, however, if one wants to double the number of ROIs taken to the analysis, one needs to have in mind that this will *quadruple* the computation time, as L has quadratic influence on the complexity.

STAGE 1: Source reconstruction

```
├── Subject 1
├── Subject 2
├── Subject 3
...
└── Subject N
```

STAGE 2: Individual Fingerprints

```
├── Subject 1
│         ├── ROI 1
│         │       ├── Clusters evaluation iteration 1
│         │       ├── Clusters evaluation iteration 2
│         │       ├── Clusters evaluation iteration 3
│         │       ...
│         │       └── Clusters evaluation iteration K
│         ├── ROI 2
│         ├── ROI 3
│         ...
│         └── ROI L
├── Subject 2
├── Subject 3
...
└── Subject N
```

STAGE 3: Data pooling

```
├── ROI 1
├── ROI 2
├── ROI 3
...
└── ROI L
```

STAGE 4: Optimal clusters evaluation

```
├── ROI 1
│       ├── Clusters evaluation iteration 1
│       ├── Clusters evaluation iteration 2
│       ├── Clusters evaluation iteration 3
│       ...
│       └── Clusters evaluation iteration K
├── ROI 2
├── ROI 3
...
└── ROI L
```

STAGE 5: Spectral Fingerprints

```
├── ROI 1
├── ROI 2
├── ROI 3
...
└── ROI L
```

STAGE 6: Group-level identification

```
├── CV-repetition 1
│              ├── ROI 1 (TRAINING)
│              │       ├── ROI 1 (VALIDATION)
│              │       ├── ROI 2 (VALIDATION)
│              │       ├── ROI 3 (VALIDATION)
│              │           ...
│              │       └── ROI L (VALIDATION)
│              ├── ROI 2 (TRAINING)
│              ├── ROI 3 (TRAINING)
│                  ...
│              └── ROI L (TRAINING)
├── CV-repetition 2
├── CV-repetition 3
...
└── CV-repetition R
```

STAGE 7: Individual-level identification

```
├── Subject 1
│       ├── TRAINING:
│       │       ├── ROI 1
│       │       │       ├── Clusters evaluation iteration 1
│       │       │       ├── Clusters evaluation iteration 2
│       │       │       ├── Clusters evaluation iteration 3
│       │       │           ...
│       │       │       └── Clusters evaluation iteration K
│       │       ├── ROI 2
│       │       ├── ROI 3
│       │           ...
│       │       └── ROI L
│       └── VALIDATION:
│               ├── ROI 1
│               │       ├── Clusters evaluation iteration 1
│               │       ├── Clusters evaluation iteration 2
│               │       ├── Clusters evaluation iteration 3
│               │           ...
│               │       └── Clusters evaluation iteration K
│               ├── ROI 2
│               ├── ROI 3
│                   ...
│               └── ROI L
│
├── Subject 2
├── Subject 3
...
└── Subject N
```

STAGE 8: Network Analysis

```
├── ROI 1
│       ├── ROI 1 (comparison)
│       ├── ROI 2 (comparison)
│           ...
│       └── ROI L (comparison)
├── ROI 2
├── ROI 3
...
└── ROI L
```

Another vital notion to keep in mind when planning computations is to know how the data are distributed among jobs and CPU cores. For each stage of the algorithm, this is different, and one can consult the table below. Together with the notion of stage complexity (tree structures above), one can adjust the number of jobs and CPU cores to be used, so a decision on what numbers to put inside the configuration file can be properly assisted.

NOTE: Parallelism vs time/memory/license resources

It is advised to use as many CPU cores as possible instead of multiplying the number of jobs, as the job is a separate MATLAB session that needs separate licenses to be used - their quantity may be limited. However, since there might be a time and memory limit per job set in the workload manager, sometimes increasing the number of jobs is necessary to meet these resource constraints.

	Single JOB handles ...	Num. of JOBS possible	Single CPU CORE handles ...
STAGE 1	set of subjects	≥ 1	1 subject
STAGE 2	set of subjects	≥ 1	set of iterations
STAGE 3	n/a*	1	n/a*
STAGE 4	set of regions	≥ 1	set of iterations
STAGE 5	set of regions	≥ 1	subset of the set of regions
STAGE 6	set of CV-repetitions	1	subset of the set of CV-repetitions
STAGE 7	set of subjects	≥ 1	set of iterations
STAGE 8	set of regions	1	subset of the set of regions

*- STAGE 3 integrates data from all subjects into a single structure, and it is so simple and quick that it needs no parallel computations

7.4.4 Pseudo-random numbers generator and reproducibility

Two possible settings

For each stage individually, one can set *pseudo-random numbers generator* (RNG) in two possible ways (see tables in 7.4.2 Configuration):

1) Fixed seed value,

2) Time-based seed value.

Using method 1), one can get full reproducibility of results as long as the other configuration parameters or input data remain the same between subsequent runs. Using method 2), one will obtain different results from subsequent runs of the algorithm.

Seed values are passed to jobs and CPU cores according to the hierarchy described below.

Seed value hierarchy

Seed values from the configuration file are called *Global Seeds*. Each stage has its own Global Seed. Global RNG is seeded with Global Seed and generates a sequence of "child" random seeds - one for each job (*Job Seed*). Jobs are separate MATLAB sessions that compute **simultaneously**. Each job has its own RNG. Inside a job, the data are

divided into separate batches. Batches inside a single job are handled **sequentially** - one after another. A new set of random numbers are generated inside the job. These numbers then serve as *Batch Seeds*. Each batch obtains unique Batch Seed. This Batch Seed will serve as a seed to initialize a random stream generator (see `RandStream` in the MATLAB Documentation) which will create a separate stream of random numbers for each CPU core. Inside each batch, there is a set of CPU cores processing a given chunk of the data. CPU cores work **in parallel**. An individual stream of pseudo-random numbers is assigned to each CPU core, as was said before. In consequence, each CPU core has its own set of pseudo-random numbers for its disposal. This guarantees that there is no overlap between sets of random numbers of CPU cores, batches, and jobs.

EXAMPLE: RNG seeds across Jobs, Batches and CPUs

Suppose that data is divided between 2 jobs, where each data subset is divided into 3 batches. Assume that, by a proper configuration, (`CFG.Global.maxNumSpmdWorkers` variable - see configuration tables in section 7.4.2), it is allowed to use 4 CPUs. Each batch is then handled in parallel by 4 CPUs.

```
Global Seed (G; generates random numbers J1 and J2)
├── Job 1 Seed (J1; generates random numbers B11, B12, B13)
│      ├── Batch 1 Seed (B11)
│      │      └── set of unique random streams (set initialized with B11)
│      │            ├── stream for CPU 1
│      │            ├── stream for CPU 2
│      │            ├── stream for CPU 3
│      │            └── stream for CPU 4
│      ├── Batch 2 Seed (B12)
│      │      └── set of unique random streams (set initialized with B12)
│      │            ├── stream for CPU 1
│      │            ├── stream for CPU 2
│      │            ├── stream for CPU 3
│      │            └── stream for CPU 4
│      └── Batch 3 Seed (B13)
│             └── set of unique random streams (set initialized with B13)
│                   ├── stream for CPU 1
│                   ├── stream for CPU 2
│                   ├── stream for CPU 3
│                   └── stream for CPU 4
└── Job 2 Seed (J2; generates random numbers B21, B22)
       ├── Batch 1 Seed (B21)
       │      └── set of unique random streams (set initialized with B21)
       │            ├── stream for CPU 1
       │            ├── stream for CPU 2
       │            ├── stream for CPU 3
       │            └── stream for CPU 4
       └── Batch 2 Seed (B22)
              └── set of unique random streams (set initialized with B22)
                    ├── stream for CPU 1
                    ├── stream for CPU 2
                    ├── stream for CPU 3
                    └── stream for CPU 4
```

8 WORKING WITH THE TOOLBOX

8.1 Typical workflow

1. Download the toolbox and the data (3 Installation), 5.4 How to download the HCP data?),
2. Prepare the data using `DATA_PREPARATION` scripts (8.3 Data preparation, 8.4 Brain atlas and common grids preparation),
3. Configure the toolbox (8.5 Configuration),
4. Remove old data and logs if needed (8.7 Visualizations and Maintenance),
5. Run selected stages (8.6 Launching stages),
6. When finished, consult the logs for errors (use `check_errors.sh` script; 8.7 Visualizations and Maintenance),
7. Refine configuration in selected stages,
8. Delete old data from the selected stages (8.7 Visualizations and Maintenance),
9. Run the selected stages again,
10. Consult the logs for errors again,
11. Make a backup of the output data (8.7 Visualizations and Maintenance),
12. Visualize and analyze the results (8.7 Visualizations and Maintenance).

8.2 Preliminaries

Before you launch the toolbox, make sure that you have:

- prepared your own data or downloaded the HCP data,
- downloaded (and extracted from a `.zip` package if needed) the repository containing the ToFFi Toolbox,
- installed MATLAB,
- installed and configured the Fieldtrip toolbox.

NOTE: Necessary toolboxes and recommended versions of the software

Recommended versions of the software:
- MATLAB - R2021a or newer,
- Fieldtrip - revision 20210816 or newer.

The following toolboxes need to be installed alongside MATLAB:
- Signal Processing Toolbox,
- Statistics and Machine Learning Toolbox,
- Parallel Computing Toolbox.

8.3 Data preparation

Prepare your data according to the guidelines written in 5 INPUT DATA section or use the HCP data. For the latter, read the instructions provided in `$REPO_ROOT/DATA_PREPARATION/HCP_DATA_PREP/instruction.org` file.

8.4 Brain atlas and common grids preparation

NOTE: Precomputed atlases and common grids

This step is optional as precomputed brain atlases and common grids are provided.

- To prepare Schaefer parcellation consult:
  `$REPO_ROOT/DATA_PREPARATION/PARCELLATIONS_PREP/Schaefer100_preparation.org`
- To prepare AAL parcellation consult:
  `$REPO_ROOT/DATA_PREPARATION/PARCELLATIONS_PREP/AAL116_preparation.org`
- To prepare individual-subjects Desikan-Killiany parcellation consult:
  `$REPO_ROOT/DATA_PREPARATION/PARCELLATIONS_PREP/DK_individual_atlases/Desikan_Killiany_preparation.m`

8.5 Configuration

8.5.1 Using terminal (Linux)

1. Open a terminal and go to the `$REPO_ROOT` directory (see the note from 7.2 Repository directories structure section).
2. Make a copy of the `$REPO_ROOT/CFG_illustrative.sh` file and rename it as you wish, e.g. `$REPO_ROOT/CFG_myexperiment1.sh`.
3. Edit the copied configuration file accordingly - see 7.4.2 Configuration section for help.
4. In the terminal, run your configuration:

   ```
   . CFG_myexperiment1.sh
   ```

   This script will ask you few questions (answer `y` or `n`), then it will remove old data and logs (if you asked to), create/overwrite the `$REPO_ROOT/CONFIGURE.m` file, open MATLAB inside the terminal, and finally, it will run `$REPO_ROOT/CONFIGURE.m` inside the terminal to finish the configuration. If there are no errors, then the Spectral Fingerprinting is ready to be run.

8.5.2 Without using terminal (Windows/MacOS)

1. Open your file explorer and go to the `$REPO_ROOT` directory (see the note from 7.2 Repository directories structure section).
2. Delete all `.log` manually files if they are no longer needed.
3. Delete all the contents of `$REPO_ROOT/STAGE_X/output` directories manually without the directory itself (only for the stages of choice) to get rid of the old data, if needed.
4. Copy `$REPO_ROOT/templates/CONFIGURE_TEMPLATE.m` to `$REPO_ROOT/CONFIGURE.m`.
5. Edit copied configuration file accordingly - see 7.4.2 Configuration section for help.
6. Open MATLAB and set the working directory to the `$REPO_ROOT` directory.
7. Run `$REPO_ROOT/CONFIGURE.m` script to finish the configuration. If there are no errors, then the Spectral Fingerprinting is ready to be run.

8.6 Launching stages

Below we provide instructions for different scenarios of running the algorithm. We refer to the variables in the configuration (7.4.2 Configuration section).

NOTE

In the following descriptions `X` in the variable/file name stands for the stage index ($X = 1, 2, 3, \ldots, 8$).

NOTE: Editing the right configuration file

Depending on whether `.sh` files for configuration are used or not, one needs to change variables in this `.sh` file or in `CONFIGURE.m` file, respectively (refer to tables in 7.4.2 Configuration to see which variable from `CONFIGURE.m` file corresponds with which variable from `.sh` file).

NOTE: Run stages in a proper order

Stages order is important! You can run certain stages only if output data from specific stages are already computed. Diagram in Fig. 8.1 will help to illustrate which stages depend on the output of which other stages.

NOTE: Stages with `INTEGRATION.m` script

Stages 2, 4, and 6 have additional `INTEGRATION.m` scripts. When configuring using `.sh` file, note that there are additional `SXINT_*` variables to be configured, and they are launched automatically whenever stage 2, 4, or 6 is run.

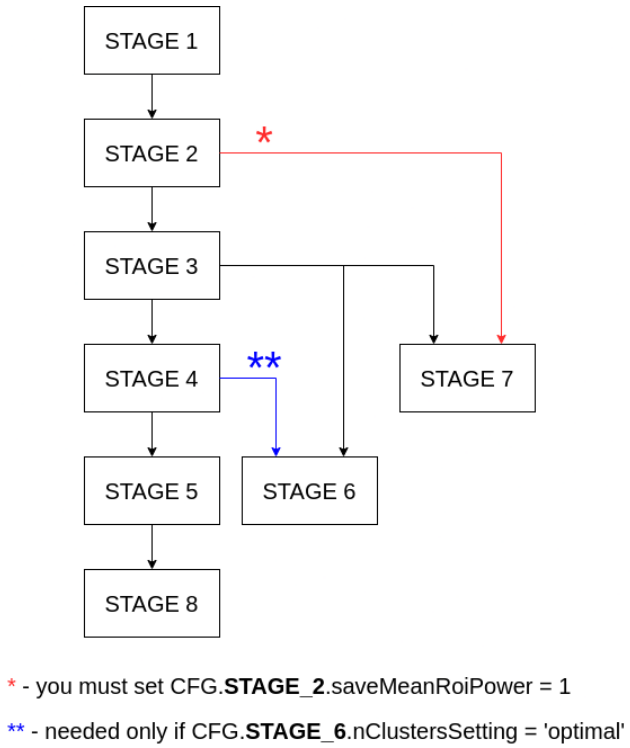


Figure 8.1: Stages interdependence diagram. An arrow means that output from stage written in a parent node is needed to run stage written in a child node.

8.6.1 Serial computations

1. Open MATLAB and set the working directory to the `$REPO_ROOT` directory (see the note from 7.2 Repository directories structure section)
2. Edit `$REPO_ROOT/RUN_SELECTED.m` file: put selected stages numbers in `stages` variable.
3. Run that script.

8.6.2 Parallel computations

8.6.2.1 Using single job

Adjusting configuration (using `CFG_*.sh` file - Linux)

1. Set `SX_SLURM_array` variable to `'1-1'` for all the stages.
2. Set `SXINT_SLURM_array` variable to `'1-1'` for stages 2, 4, and 6.
3. Set `SX_MODE` entry in `.sh` configuration file variable to `'parallel'`, ONLY for stages you want to be computed by many CPU cores at once.
4. Make sure `G_maxNumSpmdWorkers` variable is set to the desired value. This value needs to be less or equal to the number of CPU cores.

Adjusting configuration (using `CONFIGURE.m` file - Windows/MacOS)

1. Set `#SBATCH -array=1-1` value inside all `$REPO_ROOT/STAGE_X.sl` files and all `$REPO_ROOT/STAGE_XINT.sl`.
2. Set `CFG.(STAGE_NAME).MODE = 'parallel'` ONLY for stages you want to be computed by many CPU cores at once.
3. Set `CFG.(STAGE_NAME).MODE = 'serial'` ONLY for stages you want to be computed sequentially.
4. Make sure `G_maxNumSpmdWorkers` variable is set to the desired value. This value needx to be less or equal to the number of CPU cores.

NOTE: Mind memory resources

Setting the number of cores to be used (`G_maxNumSpmdWorkers` variable; refer to tables in 7.4.2 Configuration) should be done with care as it is very easy to run out of memory!

Running

1. Open MATLAB and set the working directory to the `$REPO_ROOT` directory (see the note from 7.2 Repository directories structure section).
2. Edit `$REPO_ROOT/RUN_SELECTED.m` file: put selected stages numbers in `stages` variable.
3. Run that script.

8.6.2.2 Using multiple jobs managed with SLURM (Linux only!)

Adjusting configuration

1. Configure the algorithm via `.sh` file (7.4.2 Configuration).
2. Set `SX_SLURM_array` variable for ALL the stages.

   NOTE

   Stages that should be run in serial need to have the variable set to `'1-1'`.
3. Set `SX_MODE` variable in `.sh` configuration file to `'parallel'`, ONLY for stages you want to be computed by many CPU cores at once.
4. Make sure `G_maxNumQueuedJobsPerUser` variable is set to the total number of jobs you want to use (for all stages in total).
5. Make sure `G_maxNumSpmdWorkers` variable is set to the desired value. This value needs to be less or equal to the number of CPU cores.

   NOTE: Mind memory resources

   Setting the number of cores to be used (`G_maxNumSpmdWorkers` variable; refer to tables in 7.4.2 Configuration) should be done with care as it is very easy to run out of memory!

Running

After configuration, run the script `$REPO_ROOT/RUN_SELECTED_SLURM.sh` (read instructions inside it).

There will be no questions asked and no other interactions, so one can wait until computations are done. Output files can be found inside `$REPO_ROOT/STAGE_X/output` directory. They are necessary to run visualizations.

8.7 Visualizations and Maintenance

Visualizations

1. Go to the `$REPO_ROOT/PRESENTATIONS/` directory.
2. Go to one of the directories related to results you want to show:
   - `fingerprints/` (to display results of STAGE 2 and STAGE 5)
   - `identification/` (to display results of STAGE 6 and STAGE 7)
   - `network_analysis/` (to display results of STAGE 8)
3. Edit one of the scripts:
   - `plotIndividualFingerprint.m` (plots individual subject single ROI frequency-power plot)
   - `plotSpectralFingerprint.m` (plots group-level single ROI frequency-power plot)
   - `groupHitMatrix.m` (plots matrix showing which ROI was matched with which ROI during group-level identification)
   - `groupIdentificationPlot.m` (shows bar plot with ROI identification accuracy during group-level identification)
   - `indHitMatrix.m` (plots matrix showing which ROI was matched with which ROI during individual-level identification)

- `individualIdentificationPlot.m` (shows bar plot with ROI identification accuracy for a single subject)
- `networksPlot.m` (plots binary tree showing which regions were connected based on their spectral fingerprints and displays these clusters on a cortex)

4. Change values of the variables in the `USER SETTINGS` sections accordingly - comments will help.
5. Run the script to show and/or save the visualizations.

Maintenance

Linux. On Linux operations like creating a backup, restoring from backup, removing spare data, removing spare logs, or automatic search for errors in logs can be performed by the following scripts:

- `$REPO_ROOT/backup_output.sh`
- `$REPO_ROOT/restore_from_backup.sh`
- `$REPO_ROOT/rm_output_data.sh`
- `$REPO_ROOT/rm_logs_autosaves.sh`
- `$REPO_ROOT/check_errors.sh`

Read comments inside these files to learn how to use them.

Windows/MacOS. On Windows/MacOS, operations like creating a backup, restoring from backup, removing spare data, removing spare logs, or automatic search for errors in logs need to be done manually. One can employ shell emulators like `cygwin`, `cmder`, etc., but their successful operation with provided scripts is not guaranteed.

Part III

References

[1] M. Bola and B. A. Sabel, "Dynamic reorganization of brain functional networks during cognition," *NeuroImage*, vol. 114, pp. 398–413, Jul. 2015. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S1053811915002530

[2] F. M. Krienen, B. T. T. Yeo, and R. L. Buckner, "Reconfigurable task-dependent functional coupling modes cluster around a core functional architecture," *Philosophical Transactions of the Royal Society B: Biological Sciences*, vol. 369, no. 1653, pp. 20 130 526–20 130 526, Sep. 2014. [Online]. Available: http://rstb.royalsocietypublishing.org/cgi/doi/10.1098/rstb.2013.0526

[3] R. Ciric, J. S. Nomi, L. Q. Uddin, and A. B. Satpute, "Contextual connectivity: A framework for understanding the intrinsic dynamic architecture of large-scale functional brain networks," *Scientific Reports*, vol. 7, no. 1, p. 6537, Jul. 2017, number: 1 Publisher: Nature Publishing Group. [Online]. Available: https://www.nature.com/articles/s41598-017-06866-w

[4] J. N. Keynan, Y. Meir-Hasson, G. Gilam, A. Cohen, G. Jackont, S. Kinreich, L. Ikar, A. Or-Borichev, A. Etkin, A. Gyurak, I. Klovatch, N. Intrator, and T. Hendler, "Limbic Activity Modulation Guided by Functional Magnetic Resonance Imaging–Inspired Electroencephalography Improves Implicit Emotion Regulation," *Biological Psychiatry*, vol. 80, no. 6, pp. 490–496, Sep. 2016. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0006322316000032

[5] M. Siegel, T. H. Donner, and A. K. Engel, "Spectral fingerprints of large-scale neuronal interactions," *Nature Reviews Neuroscience*, Jan. 2012. [Online]. Available: http://www.nature.com/doifinder/10.1038/nrn3137

[6] M. S. Mellem, S. Wohltjen, S. J. Gotts, A. S. Ghuman, and A. Martin, "Intrinsic frequency biases and profiles across human cortex," *Journal of Neurophysiology*, vol. 118, no. 5, pp. 2853–2864, Nov. 2017.

[7] A. Keitel and J. Gross, "Individual human brain areas can be identified from their characteristic spectral activation fingerprints," *PLoS Biol*, vol. 14, no. 6, p. e1002498, 2016. [Online]. Available: http://journals.plos.org/plosbiology/article?id=10.1371/journal.pbio.1002498

[8] K. Mahjoory, J.-M. Schoffelen, A. Keitel, and J. Gross, "The frequency gradient of human resting-state brain oscillations follows cortical hierarchies," *eLife*, vol. 9, p. e53715, Aug. 2020, publisher: eLife Sciences Publications, Ltd. [Online]. Available: https://doi.org/10.7554/eLife.53715

[9] J. Samogin, Q. Liu, M. Marino, N. Wenderoth, and D. Mantini, "Shared and connection-specific intrinsic interactions in the default mode network," *NeuroImage*, vol. 200, pp. 474–481, Oct. 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1053811919305750

[10] M. Marino, Q. Liu, J. Samogin, F. Tecchio, C. Cottone, D. Mantini, and C. Porcaro, "Neuronal dynamics enable the functional differentiation of resting state networks in the human brain," *Human Brain Mapping*, vol. 40, no. 5, pp. 1445–1457, 2019, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/hbm.24458. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/hbm.24458

[11] C. D. Hacker, A. Z. Snyder, M. Pahwa, M. Corbetta, and E. C. Leuthardt, "Frequency-specific electrophysiologic correlates of resting state fMRI networks," *NeuroImage*, vol. 149, pp. 446–457, Apr. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1053811917300629

[12] M. Rosanova, A. Casali, V. Bellina, F. Resta, M. Mariotti, and M. Massimini, "Natural Frequencies of Human Corticothalamic Circuits," *Journal of Neuroscience*, vol. 29, no. 24, pp. 7679–7685, Jun. 2009. [Online]. Available: http://www.jneurosci.org/cgi/doi/10.1523/JNEUROSCI.0445-09.2009

[13] C. Lubinus, J. Orpella, A. Keitel, H. Gudi-Mindermann, A. K. Engel, B. Roeder, and J. M. Rimmele, "Data-Driven Classification of Spectral Profiles Reveals Brain Region-Specific Plasticity in Blindness," *Cerebral Cortex*, vol. 31, no. 5, pp. 2505–2522, May 2021. [Online]. Available: https://doi.org/10.1093/cercor/bhaa370

[14] R. Oostenveld, P. Fries, E. Maris, and J.-M. Schoffelen, "FieldTrip: Open Source Software for Advanced Analysis of MEG, EEG, and Invasive Electrophysiological Data," *Computational Intelligence and Neuroscience*, vol. 2011, p. e156869, Dec. 2010. [Online]. Available: http://www.hindawi.com/journals/cin/2011/156869/abs/

[15] A. Delorme and S. Makeig, "EEGLAB: an open source toolbox for analysis of single-trial EEG dynamics including independent component analysis," *Journal of neuroscience methods*, vol. 134, no. 1, pp. 9–21, 2004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0165027003003479

[16] K. Rykaczewski, J. Nikadon, W. Duch, and T. Piotrowski, "supFunSim: Spatial Filtering Toolbox for EEG," *Neuroinformatics*, vol. 19, no. 1, pp. 107–125, Jan. 2021. [Online]. Available: https://doi.org/10.1007/s12021-020-09464-w

[17] P. Sanz Leon, S. A. Knock, M. M. Woodman, L. Domide, J. Mersmann, A. R. McIntosh, and V. Jirsa, "The Virtual Brain: a simulator of primate brain network dynamics," *Frontiers in Neuroinformatics*, vol. 7, 2013. [Online]. Available: http://journal.frontiersin.org/article/10.3389/fninf.2013.00010/abstract

[18] K. Sekihara and S. S. Nagarajan, *Adaptive Spatial Filters for Electromagnetic Brain Imaging*, ser. Series in Biomedical Engineering. Berlin Heidelberg: Springer-Verlag, 2008. [Online]. Available: https://www.springer.com/gp/book/9783540793694

[19] D. C. Van Essen, S. M. Smith, D. M. Barch, T. E. Behrens, E. Yacoub, and K. Ugurbil, "The WU-Minn Human Connectome Project: An Overview," *NeuroImage*, vol. 80, pp. 62–79, Oct. 2013. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3724347/

[20] B. D. Van Veen, W. Van Drongelen, M. Yuchtman, and A. Suzuki, "Localization of brain electrical activity via linearly constrained minimum variance spatial filtering," *IEEE Transactions on biomedical engineering*, vol. 44, no. 9, pp. 867–880, 1997. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=623056

[21] T. Piotrowski and I. Yamada, "MV-PURE Estimator: Minimum-Variance Pseudo-Unbiased Reduced-Rank Estimator for Linearly Constrained Ill-Conditioned Inverse Problems," *IEEE Transactions on Signal Processing*, vol. 56, no. 8, pp. 3408–3423, Aug. 2008.

[22] M. Hämäläinen, "Interpreting measured magnetic fields of the brain : Estimates of current distributions," *Univ. Helsinki, Finland Tech. Rep. TKK-F-A559*, 1984. [Online]. Available: https://ci.nii.ac.jp/naid/10011632127/

[23] R. D. Pascual-Marqui, C. M. Michel, and D. Lehmann, "Low resolution electromagnetic tomography: a new method for localizing electrical activity in the brain," *International Journal of Psychophysiology: Official Journal of the International Organization of Psychophysiology*, vol. 18, no. 1, pp. 49–65, Oct. 1994.

[24] M. Huang, C. J. Aine, S. Supek, E. Best, D. Ranken, and E. R. Flynn, "Multi-start downhill simplex method for spatio-temporal source localization in magnetoencephalography," *Electroencephalography and Clinical Neurophysiology/Evoked Potentials Section*, vol. 108, no. 1, pp. 32–44, Jan. 1998. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168559797000919

[25] A. M. Dale, A. K. Liu, B. R. Fischl, R. L. Buckner, J. W. Belliveau, J. D. Lewine, and E. Halgren, "Dynamic statistical parametric mapping: combining fMRI and MEG for high-resolution imaging of cortical activity," *Neuron*, vol. 26, no. 1, pp. 55–67, 2000. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0896627300811381

[26] J. Gross, J. Kujala, M. Hamalainen, L. Timmermann, A. Schnitzler, and R. Salmelin, "Dynamic imaging of coherent sources: Studying neural interactions in the human brain," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 98, no. 2, pp. 694–699, Jan. 2001.

[27] R. D. Pascual-Marqui, "Standardized low-resolution brain electromagnetic tomography (sLORETA): technical details," *Methods and Findings in Experimental and Clinical Pharmacology*, vol. 24 Suppl D, pp. 5–12, 2002.

[28] K. Friston, L. Harrison, J. Daunizeau, S. Kiebel, C. Phillips, N. Trujillo-Barreto, R. Henson, G. Flandin, and J. Mattout, "Multiple sparse priors for the M/EEG inverse problem," *NeuroImage*, vol. 39, no. 3, pp. 1104–1120, Feb. 2008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1053811907008786

[29] J. C. Mosher, R. M. Leahy, and P. S. Lewis, "EEG and MEG: forward solutions for inverse methods," *IEEE Transactions on Biomedical Engineering*, vol. 46, no. 3, pp. 245–259, 1999. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=748978

[30] L. Tait, A. Özkan, M. J. Szul, and J. Zhang, "A systematic evaluation of source reconstruction of resting MEG of the human brain with a new high-resolution atlas: Performance, precision, and parcellation," *Human Brain Mapping*, vol. n/a, no. n/a, 2021, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/hbm.25578. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/hbm.25578

[31] R. S. Desikan, F. Ségonne, B. Fischl, B. T. Quinn, B. C. Dickerson, D. Blacker, R. L. Buckner, A. M. Dale, R. P. Maguire, B. T. Hyman, M. S. Albert, and R. J. Killiany, "An automated labeling system for subdividing the human cerebral cortex on MRI scans into gyral based regions of interest," *NeuroImage*, vol. 31, no. 3, pp. 968–980, Jul. 2006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1053811906000437

[32] D. Marcus, J. Harwell, T. Olsen, M. Hodge, M. Glasser, F. Prior, M. Jenkinson, T. Laumann, S. Curtiss, and D. Van Essen, "Informatics and Data Mining Tools and Strategies for the Human Connectome Project," *Frontiers in Neuroinformatics*, vol. 5, 2011. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fninf.2011.00004/full

[33] B. Fischl, "FreeSurfer," *NeuroImage*, vol. 62, no. 2, pp. 774–781, Aug. 2012.

[34] A. Schaefer, R. Kong, E. M. Gordon, T. O. Laumann, X.-N. Zuo, A. J. Holmes, S. B. Eickhoff, and B. T. T. Yeo, "Local-Global Parcellation of the Human Cerebral Cortex from Intrinsic Functional Connectivity MRI," *Cerebral Cortex (New York, N.Y.: 1991)*, vol. 28, no. 9, pp. 3095–3114, 2018.

[35] N. Tzourio-Mazoyer, B. Landeau, D. Papathanassiou, F. Crivello, O. Etard, N. Delcroix, B. Mazoyer, and M. Joliot, "Automated anatomical labeling of activations in SPM using a macroscopic anatomical parcellation of the MNI MRI single-subject brain," *NeuroImage*, vol. 15, no. 1, pp. 273–289, Jan. 2002.

[36] A. Puce and M. S. Hämäläinen, "A Review of Issues Related to Data Acquisition and Analysis in EEG/MEG Studies," *Brain Sciences*, vol. 7, no. 6, May 2017.

[37] W. O. Tatum, *Handbook of EEG interpretation.* New York [N.Y.: Demos Medical Pub., 2013, oCLC: 841495449. [Online]. Available: http://www.credoreference.com/book/spheegi

[38] R. A. Poldrack, J. A. Mumford, and T. E. Nichols, *Handbook of functional MRI data analysis.* Cambridge New York Melbourne Madrid: Cambridge University Press, 2011, oCLC: 753167009.

[39] S. Baillet, J. C. Mosher, and R. M. Leahy, "Electromagnetic brain mapping," *IEEE Signal Processing Magazine*, vol. 18, no. 6, pp. 14–30, Nov. 2001.

[40] M. X. Cohen, *Analyzing neural time series data: theory and practice*, ser. Issues in clinical and cognitive neuropsychology. Cambridge, Massachusetts: The MIT Press, 2014.

[41] J. G. Proakis and D. K. Manolakis, *Digital Signal Processing*, 4th ed. Upper Saddle River, N.J: Pearson, Apr. 2006.

[42] Q. Liu, M. Ganzetti, N. Wenderoth, and D. Mantini, "Detecting Large-Scale Brain Networks Using EEG: Impact of Electrode Density, Head Modeling and Source Localization," *Frontiers in Neuroinformatics*, vol. 12, 2018, publisher: Frontiers. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fninf.2018.00004/full

[43] B. T. T. Yeo, F. M. Krienen, J. Sepulcre, M. R. Sabuncu, D. Lashkari, M. Hollinshead, J. L. Roffman, J. W. Smoller, L. Zöllei, J. R. Polimeni, B. Fischl, H. Liu, and R. L. Buckner, "The organization of the human cerebral cortex estimated by intrinsic functional connectivity," *Journal of Neurophysiology*, vol. 106, no. 3, pp. 1125–1165, Sep. 2011.

[44] P. Refaeilzadeh, L. Tang, and H. Liu, "Cross-Validation," in *Encyclopedia of Database Systems*, L. LIU and M. T. ÖZSU, Eds. Boston, MA: Springer US, 2009, pp. 532–538. [Online]. Available: https://doi.org/10.1007/978-0-387-39940-9_565

[45] G. Vanwinckelen and H. Blockeel, "On estimating model accuracy with repeated cross-validation," in *BeneLearn 2012: Proceedings of the 21st Belgian-Dutch Conference on Machine Learning*, Jan. 2012, pp. 39–44. [Online]. Available: https://lirias.kuleuven.be/1655861