bidspm

Release v2.2.0

the bidspm pipeline dev team

Nov 04, 2022

CONTENT

.. only:: not latex

```
[![pre-commit.ci status](https://results.pre-commit.ci/badge/github/cpp-lln-lab/bidspm/
↪main.svg)](https://results.pre-commit.ci/latest/github/cpp-lln-lab/bidspm/main)
[![miss hit](https://img.shields.io/badge/code%20style-miss_hit-000000.svg)](https://
↪misshit.org/)
[![Documentation Status: main](https://readthedocs.org/projects/bidspm/badge/?
↪version=stable)](https://bidspm.readthedocs.io/en/stable/?badge=stable)
[![Binder](https://mybinder.org/badge_logo.svg)](https://mybinder.org/v2/gh/cpp-lln-lab/
↪bidspm/dev)
[![tests with matlab](https://github.com/cpp-lln-lab/bidspm/actions/workflows/run_tests_
↪matlab.yml/badge.svg)](https://github.com/cpp-lln-lab/bidspm/actions/workflows/run_
↪tests_matlab.yml)
[![tests with octave](https://github.com/cpp-lln-lab/bidspm/actions/workflows/run_tests_
↪octave.yml/badge.svg)](https://github.com/cpp-lln-lab/bidspm/actions/workflows/run_
↪tests_octave.yml)
[![system tests with matlab](https://github.com/cpp-lln-lab/bidspm/actions/workflows/run_
↪system_tests_matlab.yml/badge.svg)](https://github.com/cpp-lln-lab/bidspm/actions/
↪workflows/run_system_tests_matlab.yml)
[![system tests with octave](https://github.com/cpp-lln-lab/bidspm/actions/workflows/run_
↪system_tests_octave.yml/badge.svg)](https://github.com/cpp-lln-lab/bidspm/actions/
↪workflows/run_system_tests_octave.yml)
[![codecov](https://codecov.io/gh/cpp-lln-lab/bidspm/branch/main/graph/badge.svg?
↪token=PMQYH0DIPX)](https://codecov.io/gh/cpp-lln-lab/bidspm)
[![DOI](https://zenodo.org/badge/DOI/10.5281/zenodo.3554331.svg)](https://doi.org/10.
↪5281/zenodo.3554331)
[![All Contributors](https://img.shields.io/badge/all_contributors-14-orange.svg?
↪style=flat-square)](https://github.com/cpp-lln-lab/bidspm#contributors)
```

CHAPTER

ONE

BIDSPM

This is a Matlab / Octave toolbox to perform MRI data analysis on a BIDS data set using SPM12.

1.1 Installation and set up

```
git clone \
    --recurse-submodules \
    https://github.com/cpp-lln-lab/bidspm.git
```

To get the latest version that is on the `dev` branch.

```
git clone \
    --recurse-submodules \
    --branch dev \
    https://github.com/cpp-lln-lab/bidspm.git
```

To start using bidspm, you just need to initialize it for this MATLAB / Octave session with::

```
bidspm()
```

Please see our documentation for more info.

1.2 Usage

For some of its functionality bidspm has a BIDS app like API.

See this page for more information.

1.2.1 Preprocessing

```
bids_dir = path_to_raw_bids_dataset;
output_dir = path_to_where_the_output_should_go;

subject_label = '01';

bidspm(bids_dir, output_dir, 'subject', ...
        'participant_label', {subject_label}, ...
```

(continues on next page)

(continued from previous page)

```
        'action', 'preprocess', ...
        'task', {'yourTask'})
```

1.2.2 GLM

```
bids_dir = path_to_raw_bids_dataset;
preproc_dir = path_to_preprocessed_dataset;
output_dir = path_to_where_the_output_should_go;
model_file = path_to_bids_stats_model_json_file;

subject_label = '01';

bidspm(bids_dir, output_dir, 'subject', ...
        'participant_label', {subject_label}, ...
        'action', 'stats', ...
        'preproc_dir', preproc_dir, ...
        'model_file', model_file)
```

Please see our documentation for more info.

1.3 Features

1.3.1 Preprocessing

If your data is fairly "typical" (for example whole brain coverage functional data with one associated anatomical scan for each subject), you might be better off running fmriprep on your data.

If you have more exotic data that cannot be handled well by fmriprep then bidspm has some automated workflows to perform amongst other things:

- remove dummies
- slice timing correction
- spatial preprocessing:
  - realignment OR realignm and unwarp
  - coregistration `func` to `anat`,
  - `anat` segmentation and skull stripping
  - (optional) normalization to SPM's MNI space
- smoothing
- fieldmaps processing and voxel displacement map creation (work in progress)

All (well almost all) preprocessed outputs are saved as BIDS derivatives with BIDS compliant filenames.

1.3.2 Statistics

The model specification are set up using the BIDS stats model and can be used to perform:

- whole GLM at the subject level
- whole brain GLM at the group level à la SPM (meaning using a summary statistics approach).
- ROI based GLM (using marsbar)
- model selection (with the MACS toolbox)

1.3.3 Quality control:

- anatomical data (work in progress)
- functional data (work in progress)
- GLM auto-correlation check

Please see our documentation for more info.

1.4 Citation

```
@software{bidspm,
  author = {Gau, Rémi and Barilari, Marco and Battal, Ceren and Rezk, Mohamed and␣
↪Collignon, Olivier and Gurtubay, Ane and Falagiarda, Federica and MacLean, Michèle and␣
↪Cerpelloni, Filippo and Shahzad, Iqra and Nunes, Márcia and Caron-Guyon, Jeanne and␣
↪Chouinard-Leclaire, Christine and Yang, Ying},
  license = {GPL-3.0},
  title   = {bidspm},
  url = {https://github.com/cpp-lln-lab/bidspm},
  version = {2.2.0}
  doi     = {10.5281/zenodo.3554331},
  publisher = {Zenodo},
  journal = {Software}
}
```

1.5 Contributors

Thanks goes to these wonderful people (emoji key):

This project follows the all-contributors specification. Contributions of any kind welcome!

CHAPTER

TWO

INSTALLATION

2.1 Dependencies

This SPM toolbox runs with Matlab and Octave.

Dependencies	Minimum required	Used for testing in CI
MATLAB	2014	2020 on Ubuntu 20.04
Octave	4.2.2	4.2.2 on Ubuntu 20.04
SPM12	7219	7771

Some functionalities require some extra SPM toolbox to work: for example the ALI toolbox for brain lesion segmentation.

2.1.1 Octave compatibility

The following features do not yet work with Octave:

- anatomicalQA
- functionalQA
- slice_display toolbox
- rsHRF workflow

Not (yet) tested with Octave:

- MACS toolbox workflow for model selection
- ALI toolbox workflow for model selection

2.2 Installation

If you are only going to use this toolbox for a new analysis and you are not planning to edit the code base of bidspm itself, we STRONGLY suggest you use this template repository to create a new project with a basic structure of folders and with the bidspm code already set up.

Otherwise you can clone the repo with all its dependencies with the following git command:

```
git clone \
    --recurse-submodules \
    https://github.com/cpp-lln-lab/bidspm.git
```

If you need the latest development, then you must clone from the `dev` branch:

```
git clone \
    --branch dev \
    --recurse-submodules \
    https://github.com/cpp-lln-lab/bidspm.git
```

If you just need the code without the commit history download and unzip, you can fin the latest version from HERE.

2.3 Initialization

> **Warning:** In general DO NOT ADD bidspm PERMANENTLY to your MATLAB / Octave path.

You just need to initialize for a given session with:

```
bidspm()
```

This will add all the required folders to the path.

You can also remove bidspm from the path with:

```
bidspm uninit
```

2.4 Installation on a computing cluster

> For stand alone download https://www.fil.ion.ucl.ac.uk/spm/download/restricted/utopia/
>
> To use SPM docker https://github.com/spm/spm-docker
>
> See FAQ: https://en.wikibooks.org/wiki/SPM/Standalone#Frequently_Asked_Questions

This relies on the fact that SPM and CPM SPM are Octave compatible, so you will be able to run most of bidspm on a high performance cluster (HPC) without having to worry about MATLAB licenses.

Of course this assumes that Octave is available on your HPC.

Note that it should also be possible to precompile with MATLAB all the things you want to run, but this is not shown here.

The pre-requisite steps are described in the example below that shows how to set up bidspm on one of the HPC of the université catholique de Louvain.

1. SSH into the HPC

Assumes that you have set things up properly. For the UCLouvain see the documentation on this website (which has some good info about using HPC in general).

If you have everything set up it should be almost as easy as opening a terminal and typing:

```
ssh lemaitre3
```

2. Get SPM

You can simply clone the latest version of SPM from github with:

```
git clone https://github.com/spm/spm12.git --depth 1
```

3. Load the Octave modules

This first step might be different on your HPC, so you might have to figure out what the equivalent modules are called on your HPC (in the UCLouvain case you can find the relevant module by typing `module spider octave`)

Once you have found the modules load them:

```
module load releases/2018b
module load Octave/4.4.1-foss-2018b
```

4. Recompile SPM for Octave

You need to recompile SPM to make sure it works with Octave. This relies on running the following Make commands:

```
make -C spm12/src PLATFORM=octave distclean
make -C spm12/src PLATFORM=octave
make -C spm12/src PLATFORM=octave install
```

5. Add SPM to the path

In the example below `$` shows when you are in the bash terminal and `octave:1>` shows when you are in the Octave terminal.

Launch Octave:

```
$ octave

GNU Octave, version 4.4.1
Copyright (C) 2018 John W. Eaton and others.
This is free software; see the source code for copying conditions.
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE.  For details, type 'warranty'.

Octave was configured for "x86_64-pc-linux-gnu".

Additional information about Octave is available at https://www.octave.org.

Please contribute if you find this software useful.
For more information, visit https://www.octave.org/get-involved.html

Read https://www.octave.org/bugs.html to learn how to submit bug reports.
For information about changes from previous versions, type 'news'.
```

Add the SPM12 folder to the path and save the path:

```
octave:1> addpath(fullfile(pwd, 'spm12'))
octave:2> savepath
octave:3> exit
```

5. Install bidspm

As before install and run an initialization:

```
git clone \
  -b dev \
```

(continues on next page)

(continued from previous page)

```
 --recurse-submodules \
 https://github.com/cpp-lln-lab/bidspm.git
```

Warning: There are some warnings thrown during initialization:

```
octave:1> initCppSpm
warning: addpath: /home/users/r/g/rgau/bidspm/lib/spmup/utlilities/home/users/r/g/
↪rgau/bidspm/lib/spm_2_bids: No such file or directory
warning: called from initCppSpm at line 67 column 5
warning: function /home/users/r/g/rgau/bidspm/lib/spmup/external/cubehelix.m shadows␣
↪a core library function
warning: called from initCppSpm at line 67 column 5
warning: addpath: /home/users/r/g/rgau/bidspm/src/workflows/stats/home/users/r/g/rgau/
↪bidspm/lib/spmup: No such file or directory
```

As well as many warnings of the type:

```
sh: makeinfo: command not found
warning: doc_cache_create: unusable help text found in file 'analyze75info'
```

CHAPTER

THREE

USAGE NOTES

3.1 MATLAB API

src.messages.**bidspmHelp**()

General intro function for bidspm

Note:

- all parameters use snake_case
- most "invalid" calls simply initialize bidspm

BIDS APP CALLS

generic call:

```
bidspm(bids_dir, output_dir, analysis_level, ...
        'action', 'some_action', ...
        'participant_label', {}, ...
        'dry_run', false, ...
        'bids_filter_file', struct([]), ...
        'verbosity', 2, ...
        'space', {'individual', 'IXI549Space'}, ...
        'options', struct([]))
```

Obligatory parameters

Parameters

- **bids_dir** (path) – path to a raw BIDS dataset
- **output_dir** (path) – path where to output data
- **analysis_level** (string) – can either be 'subject' or 'dataset'
- **action** (char) – defines the pipeline to run; can be any of:
  - 'preprocess'
  - 'stats'
  - 'contrasts'
  - 'results'

Note:

- 'stats' runs model specification / estimation, contrast computation, display results

- `'contrasts'` runs contrast computation, display results
- `'results'` displays results

Optional parameters common to all actions

Parameters

- **participant_label** (`cellstr`) – cell of participants labels. For example: `{'01', '03', '08'}`. Can be a regular expression.
- **dry_run** (`logical`) – Defaults to `false`
- **bids_filter_file** (`path`) – path to JSON file or structure
- **verbosity** (`positive integer`) – can be `0`, `1` or `2`. Defaults to `2`
- **space** (`cell string`) – Defaults to `{'individual', 'IXI549Space'}`
- **options** (`path to JSON file or structure`) – See the `checkOptions` help to see the available options.

Note: Arguments passed to bidspm have priorities over the options defined in `opt`. For example passing the argument `'dry_run', true` will override the option `opt.dryRun = false`.

PREPROCESSING:

```
bidspm(bids_dir, output_dir, 'subject', ...
        'action', 'preprocess', ...
        'participant_label', {}, ...
        'dry_run', false, ...
        'bids_filter_file', struct([]), ...
        'verbosity', 2, ...
        'space', {'individual', 'IXI549Space'}, ...
        'options', struct([]), ...
        'task', {}, ...
        'dummy_scans', 0, ...       % specific to preprocessing
        'anat_only', false, ...     % specific to preprocessing
        'ignore', {}, ...
        'fwhm', 6)
```

Obligatory parameters

Parameters

- **task** (`cell string`) – only one task
- **dummy_scans** (`positive scalar`) – Number of dummy scans to remove. Defaults to `0`

Optional parameters

Parameters

- **anat_only** (`logical`) –
- **ignore** (`cell string`) – can be any of `{'fieldmaps', 'slicetiming', 'unwarp', 'qa'}`
- **fwhm** (`positive scalar`) – smoothing to apply to the preprocessed data

STATS:

```
bidspm(bids_dir, output_dir, 'subject', ...
        'action', 'stats', ...
        'preproc_dir', preproc_dir, ...        % specific to stats
        'model_file', model_file, ...          % specific to stats
        'participant_label', {}, ...
        'dry_run', false, ...
        'bids_filter_file', struct([]), ...
        'verbosity', 2, ...
        'space', {'individual', 'IXI549Space'}, ...
        'options', struct([]), ...,
        'roi_based', false, ...
        'design_only', false, ...
        'ignore', {}, ...
        'task', {}, ...
        'fwhm', 6)
```

Obligatory parameters

Parameters

- **preproc_dir** (path) – path to preprocessed data
- **model_file** (path to JSON file or structure) –

Optional parameters

Parameters

- **roi_based** (logical) –
- **task** (cell string) –
- **fwhm** (positive scalar) – smoothing lelvel of the preprocessed data
- **design_only** (logical) – to only run the model specification when at the group level
- **ignore** (cell string) – can be any of {'qa'}

low level calls

USAGE:

```
% initialise (add relevant folders to path)
bidspm

% equivalent to
bidspm init
bidspm('action', 'init')

% help
bidspm help
bidspm('action', 'help')

% uninitialise (remove relevant folders from path)
bidspm uninit
bidspm('action', 'uninit')

% also adds folder for testing to the path
bidspm dev
```

(continues on next page)

(continued from previous page)

```
bidspm('action', 'dev')

% tried to update the current branch from the upstream repository
bidspm update
bidspm('action', 'update')

% misc
bidspm version
bidspm('action', 'version')

bidspm run_tests
bidspm('action', 'run_tests')


For a more readable version of this help section,
see the online <a
href="https://bidspm.readthedocs.io/en/stable/bids_app_api.html">documentation</a>.
```

3.2 Command line API

3.2.1 Preprocessing

bidspm is a SPM base BIDS app

```
usage: bidspm [-h] [-v] [--participant_label PARTICIPANT_LABEL [PARTICIPANT_LABEL ...]]␣
↪[--dry_run {True,False}]
               [--bids_filter_file BIDS_FILTER_FILE] [--verbosity {0,1,2}] [--options␣
↪OPTIONS] --action {preprocess}
               [--space SPACE [SPACE ...]] [--task TASK] [--dummy_scans DUMMY_SCANS] [--
↪anat_only {True,False}]
               [--ignore {fieldmaps,slicetiming,unwarp} [{fieldmaps,slicetiming,unwarp} ..
↪.]] [--fwhm FWHM]
               bids_dir output_dir {subject,dataset}
```

Positional Arguments

bids_dir The directory with the input dataset formatted according to the BIDS standard.

output_dir The directory where the output files will be stored. If you are running group level analysis this folder should be prepopulated with the results of the participant level analysis.

analysis_level Possible choices: subject, dataset

Level of the analysis that will be performed. Multiple participant level analyses can be run independently (in parallel) using the same output_dir.

Named Arguments

-v, --version show program's version number and exit

--participant_label
The label(s) of the participant(s) that should be analyzed. The label corresponds to sub-<participant_label> from the BIDS spec (so it does not include "sub-"). If this parameter is not provided all subjects should be analyzed. Multiple participants can be specified with a space separated list. Can be a regular expression. Example: `'01', '03', '08'`.

--dry_run Possible choices: True, False

When set to `true` this will generate and save the SPM batches, but not actually run them.

Default: False

--bids_filter_file
A JSON file describing custom BIDS input filters.

--verbosity Possible choices: 0, 1, 2

Verbosity level.

Default: 2

--options
Path to JSON file containing bidspm options.

--action Possible choices: preprocess

Level of the analysis that will be performed. Multiple participant level analyses can be run independently (in parallel) using the same output_dir.

--space
Space to normalize to generate output in for `preprocess`.

Default: ['individual', 'IXI549Space']

--task
Tasks to `preprocess`. Only one value allowed.

--dummy_scans
Number of dummy scans to remove.

Default: 0

--anat_only Possible choices: True, False

If preprocessing should be done only on anatomical data.

Default: False

--ignore Possible choices: fieldmaps, slicetiming, unwarp

If preprocessing should be done only on anatomical data.

--fwhm
The full width at half maximum of the gaussian kernel to apply to the preprocessed data.

Default: 6.0

- all parameters use `snake_case`,
- most "invalid" calls simply initialize bidspm.

For a more readable version of this help section, see the online https://bidspm.readthedocs.io/en/stable/bids_app_api.html.

3.2.2 Stats

bidspm is a SPM base BIDS app

```
usage: bidspm [-h] [-v] [--participant_label PARTICIPANT_LABEL [PARTICIPANT_LABEL ...]]␣
↪[--dry_run {True,False}]
              [--bids_filter_file BIDS_FILTER_FILE] [--verbosity {0,1,2}] [--options␣
↪OPTIONS] --action {stats,contrasts,results}
              [--preproc_dir PREPROC_DIR] [--task TASK [TASK ...]] [--space SPACE] [--
↪model_file MODEL_FILE] [--fwhm FWHM]
              [--roi_based {True,False}]
              bids_dir output_dir {subject,dataset}
```

Positional Arguments

bids_dir The directory with the input dataset formatted according to the BIDS standard.

output_dir The directory where the output files will be stored. If you are running group level analysis this folder should be prepopulated with the results of the participant level analysis.

analysis_level Possible choices: subject, dataset

Level of the analysis that will be performed. Multiple participant level analyses can be run independently (in parallel) using the same `output_dir`.

Named Arguments

-v, --version show program's version number and exit

--participant_label The label(s) of the participant(s) that should be analyzed. The label corresponds to sub-<participant_label> from the BIDS spec (so it does not include "sub-"). If this parameter is not provided all subjects should be analyzed. Multiple participants can be specified with a space separated list. Can be a regular expression. Example: `'01', '03', '08'`.

--dry_run Possible choices: True, False

When set to `true` this will generate and save the SPM batches, but not actually run them.

Default: False

--bids_filter_file A JSON file describing custom BIDS input filters.

--verbosity Possible choices: 0, 1, 2

Verbosity level.

Default: 2

--options Path to JSON file containing bidspm options.

--action Possible choices: stats, contrasts, results

Level of the analysis that will be performed. Multiple participant level analyses can be run independently (in parallel) using the same output_dir.

- `stats`: runs model specification / estimation, contrast computation, display results
- `contrasts`: contrast computation, display results
- `results`: display results

--preproc_dir Path to preprocessed data.

--task Tasks of the input data.

--space Space of the input data.

Default: ['individual', 'IXI549Space']

--model_file Path to BIDS stats model.

--fwhm Full width at half maximum of the gaussian kernel of the input data.

Default: 6.0

--roi_based Possible choices: True, False

To run stats only in regions of interests.

Default: False

- all parameters use `snake_case`,
- most "invalid" calls simply initialize bidspm.

For a more readable version of this help section, see the online https://bidspm.readthedocs.io/en/stable/bids_app_api.html.

3.2.3 Low level actions

bidspm is a SPM base BIDS app

```
usage: bidspm [-h] [-v] --action {init,dev,uninit,update,run_tests}
```

Named Arguments

-v, --version show program's version number and exit

--action Possible choices: init, dev, uninit, update, run_tests

Low level action to perform.

- `init`: initialise (add relevant folders to MATLAB path).
- `dev`: initialise and also adds folder for testing to the path.
- `uninit`: uninitialise (remove relevant folders from MATLAB path)

- `update`: tries to update the current branch from the upstream repository
- `run_tests`: tries to update the current branch from the upstream repository

- all parameters use `snake_case`,
- most "invalid" calls simply initialize bidspm.

  For a more readable version of this help section, see the online https://bidspm.readthedocs.io/en/stable/bids_app_api.html.

CHAPTER

FOUR

SET UP

4.1 Configuration of the pipeline

4.1.1 Options

Most of the options you have chosen for your analysis will be set in a variable `opt` an Octave/Matlab structure.

The content of that structure can be defined:

- "at run" time in a script or a function (that often called `getOption`)
- in a separate json file that can be loaded with `src/utils/loadAndCheckOptions.m()`.

You can find examples of both in the `demos` folder. You can also find a template function for `getOption` in the `templates` folder.

Check the *Pipeline defaults* page to see the available options and their defaults.

Selecting groups and subjects

The way to select certain subjects is summarised in the documentation of the `src/utils/getSubjectList()` function.

`src.bids.`**`getSubjectList`**`(`*BIDS*, *opt*`)`

Returns the subjects to analyze in `opt.subjects`

USAGE:

```
opt = getSubjectList(BIDS, opt)
```

Parameters

- **`BIDS`** (`structure`) – dataset layout. See also: bids.layout, getData.
- **`opt`** (`structure`) – Options chosen for the analysis. See also: `checkOptions()` and `loadAndCheckOptions()`.

Returns

- **opt**
  (structure)

If no group or subject is speficied in `opt` then all subjects are included. This is equivalent to the default:

```
opt.groups = {''};
opt.subjects = {[]};
```

If you want to run the analysis of some subjects only based on the group they belong to **as defined in the ``participants.tsv``** file, you can do it like this:

```
opt.groups = {'control'};
```

This will run the pipeline on all the `control` subjects.

If your subject label is `blnd02` (as in `sub-blnd02`) but its group affiliation in the `participants.tsv` says `control`, then this subject will NOT be included if you run the pipeline with `opt.groups = {'blnd'}`.

If you have more than 2 groups you can specify them like this:

```
opt.groups = {'cont', 'cat'};
```

You can also directly specify the subject label for the participants you want to run:

```
opt.subjects = {'01', 'cont01', 'cat02', 'ctrl02', 'blind01'};
```

And you can combine both methods:

```
opt.groups = {'blind'};
opt.subjects = {'ctrl01'};
```

This will include all `blind` subjects and `sub-ctrl01`.

Setting directories

Below are some example on how to specify input and output directories.

Note: It will be easier and make your code more portable, if you use relative path for your directory setting.

For preprocessing

For a given folder structure:

```
my_fmri_project
   ├── code
   │   └── getOptionPreproc.m
   ├── outputs/derivatives
   └── inputs
       └── raw
```

Example content of `getOptionPreproc` file:

```
opt.pipeline.type = 'preproc';

this_dir = fileparts(mfilename('fullpath'));

opt.dir.raw = fullfile(this_dir, '..' 'inputs', 'raw');
opt.dir.derivatives = fullfile(this_dir, '..', 'outputs', 'derivatives');
```

For statistics

To run a GLM, bidspm gets the images and confound time series from a preprocessed derivatives BIDS dataset (from fMRIprep or bidspm) and the `events.tsv` files from a raw BIDS dataset.

For a given folder structure:

```
my_fmri_project
  ├── code
  │   └── getOptionStats.m
  ├── outputs/derivatives
  └── inputs
      ├── fmriprep
      └── raw
```

Example content of `getOptionStats` file:

```
opt.pipeline.type = 'stats';

this_dir = fileparts(mfilename('fullpath'));

opt.dir.raw = fullfile(this_dir, '..', 'inputs', 'raw');
opt.dir.preproc = fullfile(this_dir, '..', 'inputs', 'fmriprep');
opt.dir.derivatives = fullfile(this_dir, '..', 'outputs', 'derivatives');
```

The actual `opt.dir.input` and `opt.dir.output` folders will usually be set automatically when running:

```
opt = checkOptions(opt)
```

But you can set those by hand if you prefer.

CHAPTER

FIVE

BIDS STATS MODEL JSON FILE

This file allows you to specify the GLM to run and which contrasts to compute.

It follows BIDS statistical model.

This type of JSON file is a bit more complicated than the usual JSON files, you might be acquainted with in BIDS. So make sure you have a read through the JSON 101 page.

Then have a look at the walkthrough that explains how to build a simple model.

5.1 Create a default BIDS model for a dataset

src.bids_model.**createDefaultStatsModel**(*BIDS*, *opt*)

Creates a default model json file for a BIDS dataset

USAGE:

```
opt = createDefaultStatsModel(BIDS, opt)
```

Parameters

- **BIDS** (struct or path) – dataset layout. See also: bids.layout, getData.
- **opt** (structure) – Options chosen for the analysis. See also: checkOptions() and loadAndCheckOptions().

Returns

opt

Outputs a model file in the current directory:

```
fullfile(pwd, 'models', ['model-default' opt.taskName '_smdl.json']);
```

This model has 3 "Nodes" in that order:

- Run level:
  - will create a GLM with a design matrix that includes all all the possible type of trial_types that exist across all subjects and runs for the task specified in opt, as well as the realignment parameters.
  - use DummyContrasts to generate contrasts for each trial_type for each run. This can be useful to run MVPA analysis on the beta images of each run.
- Subject level:

  - will create a GLM with a design matrix that includes all all the possible type of trial_types that exist across all subjects and runs for the task specified in `opt`, as well as the realignment parameters.
  - use `DummyContrasts` to generate contrasts for all each trial_type across runs

- Dataset level:
  - use `DummyContrasts` to generate contrasts for each trial_type for at the group level.

EXAMPLE:

```
opt.taskName = 'myFascinatingTask';
opt.dir.raw = fullfile(pwd, 'data', 'raw');
opt = checkOptions(opt);

[BIDS, opt] = getData(opt, opt.dir.raw);

createDefaultStatsModel(BIDS, opt);
```

5.2 Validate your model

5.2.1 In Visual Studio Code

You can add those lines to the `.vscode/settings.json` of your project to help you validate BIDS stats models as you write them.

```
{
    "json.schemas": [
        {
        "fileMatch": ["model-*_smdl.json"],
        "url": "https://bids-standard.github.io/stats-models/BIDSStatsModel.json"
        }
    ],
    "files.associations": {"*.m": "matlab"}
}
```

5.2.2 In the browser

Otherwise you can use the online validator and copy paste your model in it.

5.2.3 Using the BIDS stats model python package

```
pip install bsmschema
```

```
from bsmschema.models import BIDSStatsModel

BIDSStatsModel.parse_file('model-example_smdl.json')
```

5.3 Loading and interacting with a BIDS stats model

You can use the `BidsModel` class to create a bids model instance and interact with. This class inherits from bids-matlab `+bids.Model` class.

class src.bids_model.**BidsModel**(*varargin*)

getHRFderivatives(*varargin*)

returns the HRF derivatives of a node of a BIDS statistical model

getInclusiveMaskThreshold(*varargin*)

returns the threshold for inclusive masking of subject level GLM node of a BIDS statistical model

getModelMask(*varargin*)

returns the mask of a node of a BIDS statistical model

getSerialCorrelationCorrection(*varargin*)

returns the Serial Correlation Correction of a node of a BIDS statistical model

There are also extra functions to interact with those models.

src.bids_model.**getContrastsList**(*node*, *model*)

Get list of names of Contrast from this Node or gets its from the previous Nodes

USAGE:

```
contrastsList = getContrastsList(node, model)
```

Parameters

- **node** (`char or structure`) – node name or node content
- **model** (`BIDS stats model object`) –

Returns

contrastsList (cellstr)

src.bids_model.**getDummyContrastsList**(*node*, *model*)

Get list of names of DummyContrast from this Node or gets its from the previous Nodes

USAGE:

```
dummyContrastsList = getDummyContrastsList(node, model)
```

Parameters

- **node** (`char or structure`) – node name or node content
- **model** (`BIDS stats model object`) –

Returns

dummyContrastsList (cellstr)

5.4 bidspm implementation of the BIDS stats model

bidspm only implements a subset of what is currently theoretically possible with the BIDS stats model.

For example, at the subject level the bidspm can only access variables, that are in the `events.tsv` in tha raw dataset or in the `regressors.tsv` or `timeseries.tsv` generated by the preprocessing pipeline.

At the group level, it is only possible to access some variables from the `participants.tsv` file.

5.4.1 Transformation

The `Transformations` object allows you to define what you want to do to some variables, before you put them in the design matrix.

Currently bidspm can only transform variables contained in `events.tsv` files.

It uses bids-matlab transformers to run those transformations. Please see this bids-matlab documentation to know how to use them and call them in your JSON.

The advantage of this bids-matlab transformers is that they allow you to directly add on tsv files to quickly see what outcome a series of transformers will produce.

Below is an example on how to subtract 3 seconds from the event onsets of the conditions `motion` listed in the `trial_type` columns of the `events.tsv` file, and put the output in a variable called `motion`.

```
"Transformations": {
    "Transformer": "bidspm",
    "Instructions": [
        {
            "Name": "Subtract",
            "Input": [
              "onset"
            ],
            "Query": "trial_type==motion",
            "Value": 3,
            "Output": [
                "motion"
            ]
        }
    ]
}
```

At the subject level, bidspm can only access apply transformation on the content `events.tsv`.

5.4.2 HRF

For a given `Node`, `Model.X` defines the variables that have to be put in the design matrix.

Here `trans_?` means any of the translation parameters (in this case `trans_x`, `trans_y`, `trans_z`) from the realignment that are stored in `_confounds.tsv` files.

Similarly `*outlier*` means that ANY "scrubbing" regressors containing the word `outlier` created by fMRIprep or bidspm to detect motion outlier or potential dummy scans will be included.

```
"Model": {
    "Type": "glm",
    "X": [
        "motion",
        "static",
        "trans_?",
        "rot_?",
        "*outlier*"
    ],
    "HRF": {
        "Variables": [
            "motion",
            "static"
        ],
        "Model": "spm"
}
```

`HRF` specifies:

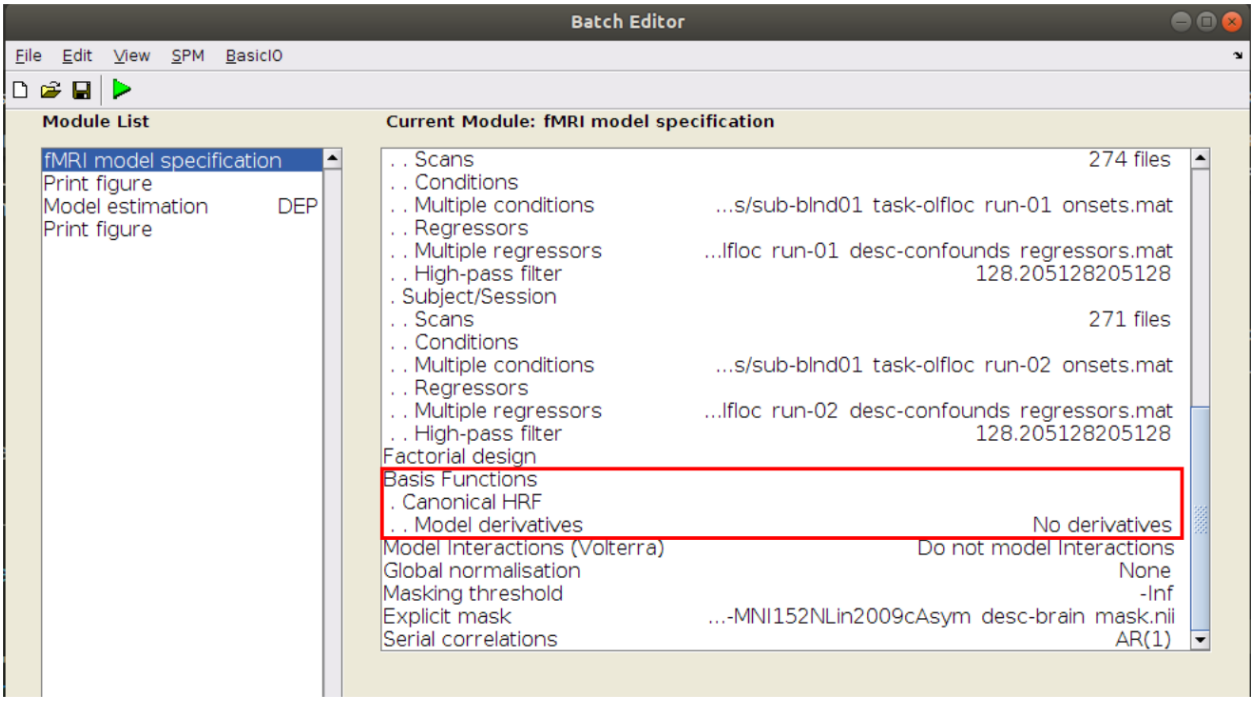
- which variables of `X` have to be convolved
- what HRF model to use to do so.

You can choose from:

- `"spm"`
- `"spm + derivative"`
- `"spm + derivative + dispersion"`

Not yet implementated:

- `"fir"`



5.4.3 Software

Note that if you wanted to change the *SerialCorrelation model* used by bidspm, you could do so via the `Software` object of the BIDS stats model.

Similar you can adapt directly in the model the threshold used by SPM to create an implicit inclusive mask when running a GLM (the value `defaults.mask.thresh` of SPM defaults.) .

```
{
    "Nodes": [
        {
            "Level": "Run",
            "Name": "run_level",
            "Model": {
                "X": ["trial_type.listening"],
                "HRF": {
                    "Variables": ["trial_type.listening"],
                    "Model": "spm"
                },
                "Type": "glm",
                "Software": {
                    "SPM": {
                        "SerialCorrelation": "AR(1)",
                        "InclusiveMaskingThreshold": "-Inf"
                    }
                }
            }
        }
    ]
}
```

5.4.4 Contrasts

Run level

To stay close to the way most SPM users are familiar with, all runs are analyzed in one single GLM.

Contrasts are the run level that are either specified using `DummyContrasts` or `Contrasts` will be computed and will have the run number appended to their name in the SPM gui as shown in *Contrast for run 1* and *Contrast for run 2*.

```
{
    "Level": "Run",
    "Name": "run_level",
    "X": [
        "olfid_eucalyptus_left",
        "olfid_eucalyptus_right",
        "olfid_almond_left",
        "olfid_almond_right",
        "olfloc_eucalyptus_left",
        "olfloc_eucalyptus_right",
        "olfloc_almond_left",
        "olfloc_almond_right",
        "resp_03",
```

(continues on next page)

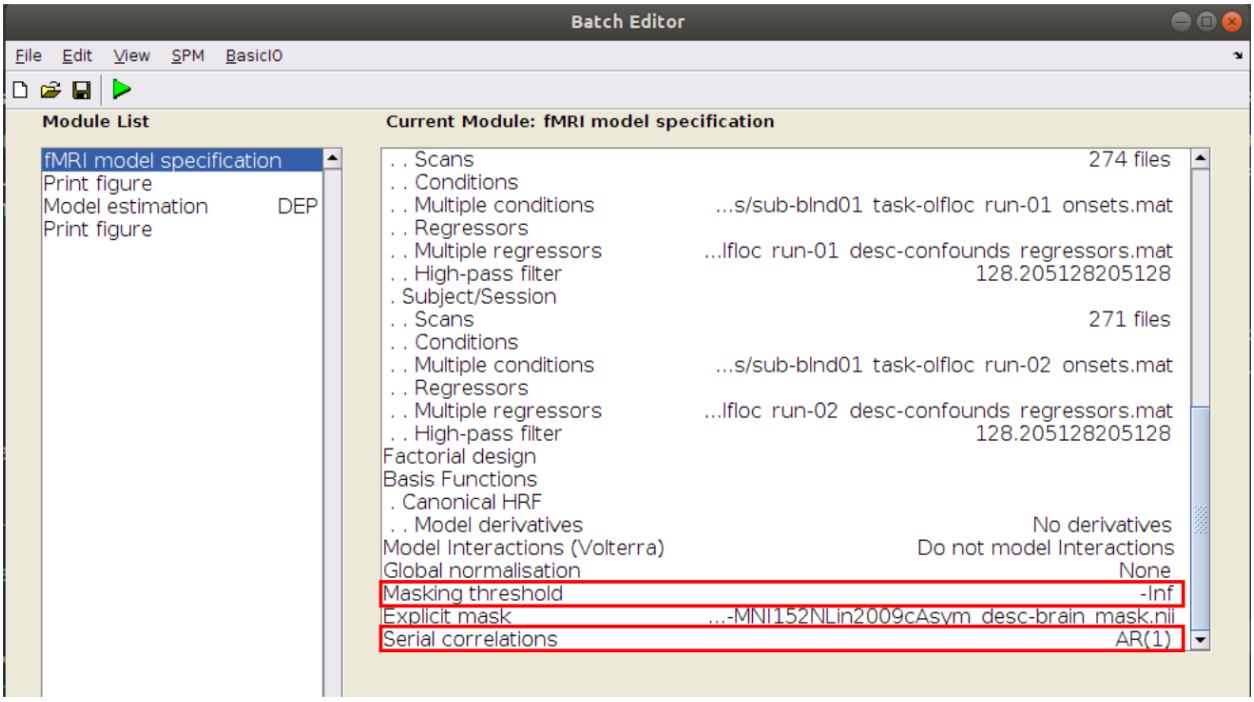


Fig. 1: Corresponding options in SPM batch

(continued from previous page)

```
        "resp_12",
        1
    ],
    "DummyContrasts": {
        "Contrasts": [
            "olfid_eucalyptus_left",
            "olfid_eucalyptus_right",
            "olfid_almond_left",
            "olfid_almond_right",
            "olfloc_eucalyptus_left",
            "olfloc_eucalyptus_right",
            "olfloc_almond_left",
            "olfloc_almond_right"
        ],
        "Test": "t"
    },
    "Contrasts": [
        {
            "Name": "olfid",
            "ConditionList": [
                "olfid_eucalyptus_left",
                "olfid_eucalyptus_right",
                "olfid_almond_left",
                "olfid_almond_right"
            ],
            "Weights": [
```

(continues on next page)

(continued from previous page)

```
                1,
                1,
                1,
                1
            ],
            "Test": "t"
        }
    ]
}
```

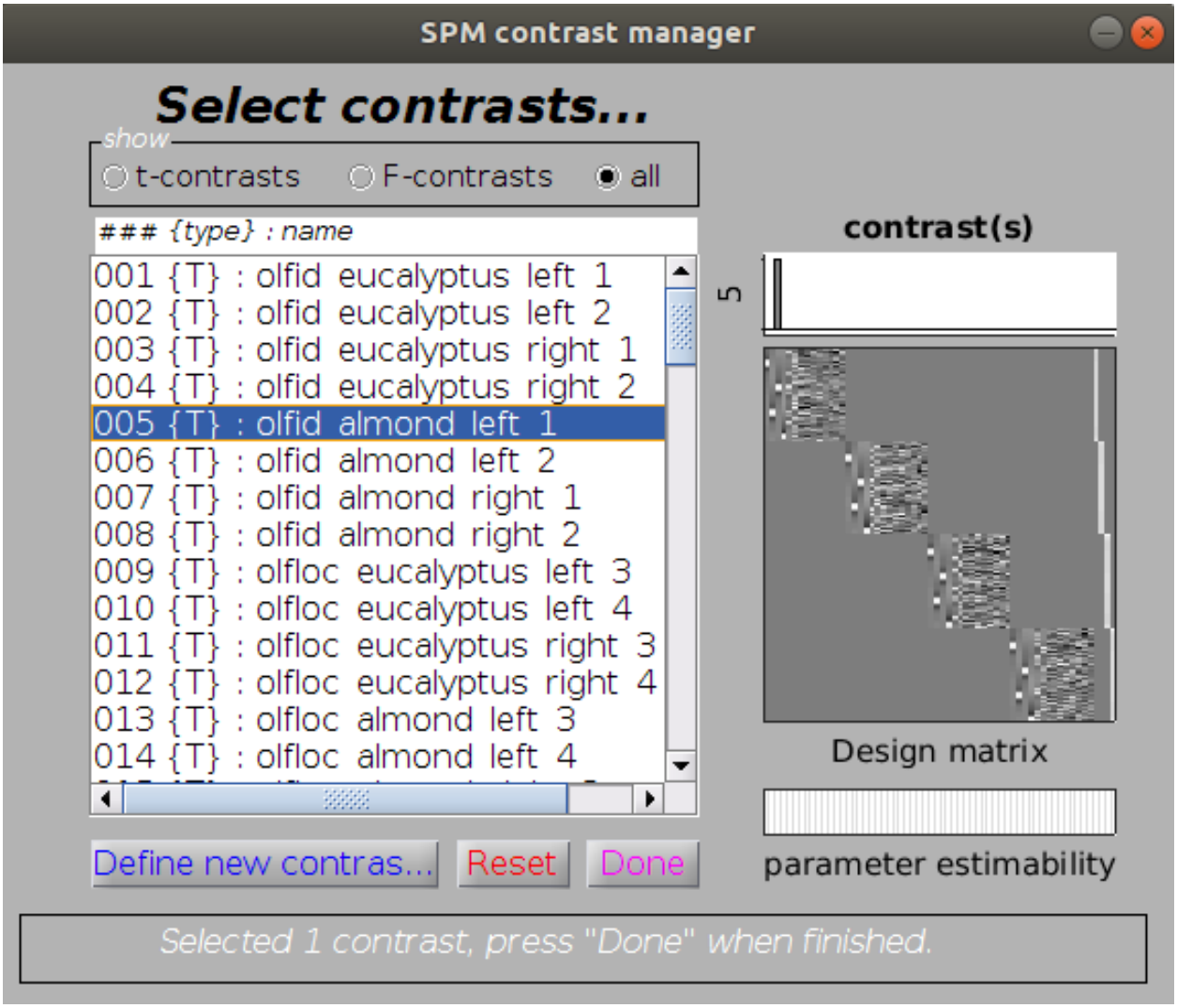


Fig. 2: Contrast for run 1

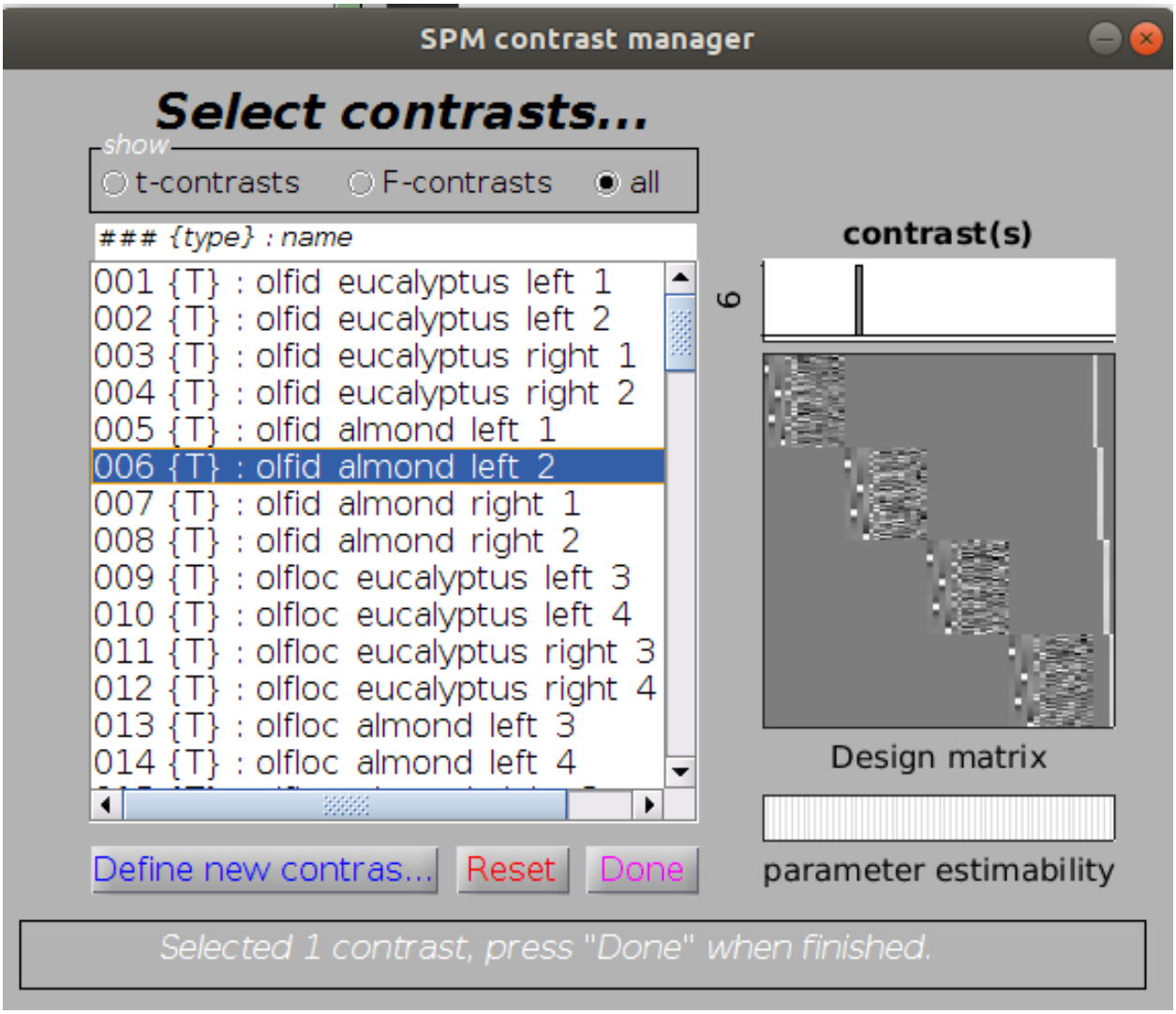


Fig. 3: Contrast for run 2

Subject level

At the moment the only type of model supported at the run level is averaging of run level contrasts.

```
{
    "Level": "Subject",
    "Name": "subject_level",
    "Description": "Only averaging at the subject level is supported for now.",
    "GroupBy": [
        "contrast",
        "subject"
    ],
    "Model": {
        "X": [
            1
        ],
        "Type": "glm"
    },
    "DummyContrasts": {
        "Test": "t"
    }
}
```

5.5 Dataset level

At the moment only, the only type of models that are supported are:

- one sample t-test: averaging across all subjects

```
{
  "Level": "Dataset",
  "Name": "dataset_level",
  "GroupBy": [
    "contrast"
  ],
  "Model": {
    "X": [
      1
    ],
    "Type": "glm"
  },
  "DummyContrasts": {
    "Test": "t"
  }
}
```

- one sample t-test: averaging across all subjects of a specific group

```
{
  "Level": "Dataset",
  "Name": "within_group",
  "Description": "one sample t-test for each group",
```

(continues on next page)

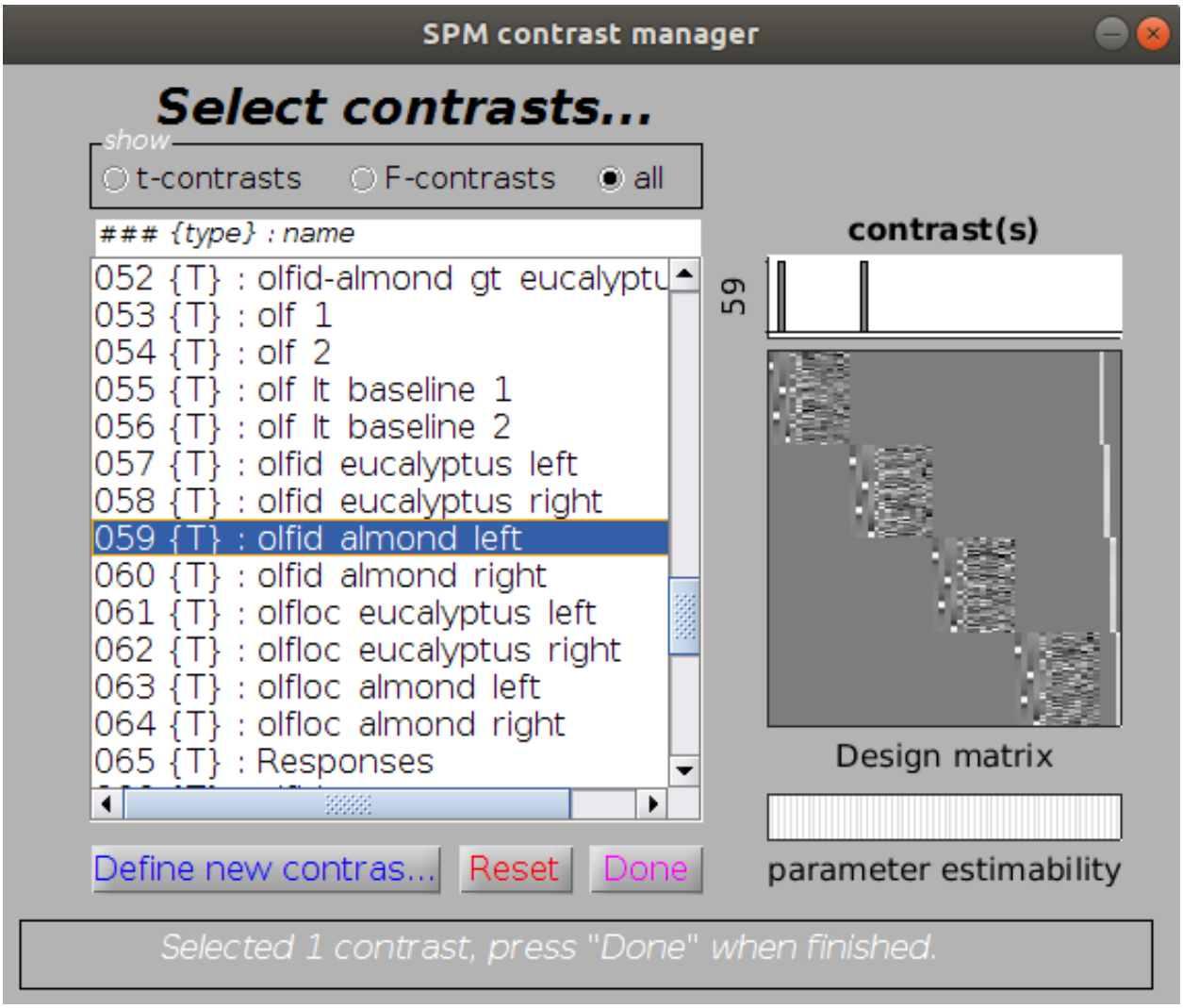


Fig. 4: Subject level contrast averaging beta of run 1 and 2

(continued from previous page)

```
  "GroupBy": [
    "contrast",
    "Group"
  ],
  "Model": {
    "Type": "glm",
    "X": [
      1
    ]
  },
  "DummyContrasts": {
    "Test": "t"
  }
}
```

- 2 samples t-test: comparing 2 groups

At the moment this can only be based on how participants are allocated to a group based on a `group` or `Group` column in the `participants.tsv` of in the raw dataset.

```
{
  "Level": "Dataset",
  "Name": "between_groups",
  "Description": "2 sample t-test between groups",
  "GroupBy": [
    "contrast"
  ],
  "Model": {
    "Type": "glm",
    "X": [
      1,
      "group"
    ]
  },
  "Contrasts": [
    {
      "Name": "blind_gt_control",
      "ConditionList": [
        "Group.blind",
        "Group.control"
      ],
      "Weights": [
        1,
        -1
      ],
      "Test": "t"
    }
  ]
}
```

5.5.1 Method section

It is possible to write a draft of method section based on a BIDS statistical model.

```
opt.model.file = fullfile(pwd, ...
                          'models', ...
                          'model-faceRepetition_smdl.json');
opt.fwhm.contrast = 0;
opt = checkOptions(opt);

opt.designType = 'block';

outputFile = boilerplate(opt, ...
                         'outputPath', pwd, ...
                         'pipelineType', 'stats');
```

```
## fMRI statistical analysis

The fMRI data were analysed with BIDSpm (v2.2.0; https://github.com/cpp-lln-lab/bidspm;␣
↪DOI: https://doi.org/10.5281/zenodo.3554331)
using statistical parametric mapping
(SPM12 - 7771; Wellcome Center for Neuroimaging, London, UK;
https://www.fil.ion.ucl.ac.uk/spm; RRID:SCR_007037)
using MATLAB 9.4.0.813654 (R2018a)
on a unix computer (Linux version 5.15.0-52-generic (buildd@lcy02-amd64-032) (gcc␣
↪(Ubuntu 11.2.0-19ubuntu1) 11.2.0, GNU ld (GNU Binutils for Ubuntu) 2.38) #58-Ubuntu␣
↪SMP Thu Oct 13 08:03:55 UTC 2022
).

The input data were the preprocessed BOLD images in IXI549Space space for the task "␣
↪facerepetition ".

### Run / subject level analysis

At the subject level, we performed a mass univariate analysis with a linear
regression at each voxel of the brain, using generalized least squares with a
global  AR(1)  model to account for temporal auto-correlation
 and a drift fit with discrete cosine transform basis ( 128 seconds cut-off).

Image intensity scaling was done run-wide before statistical modeling such that
the mean image would have a mean intracerebral intensity of 100.

We modeled the fMRI experiment in a  event  design with regressors
entered into the run-specific design matrix. The onsets
were convolved with SPM canonical hemodynamic response function (HRF)
 and its temporal and dispersion derivatives for the conditions:
  - `famous_1`,
 - `famous_2`,
 - `unfamiliar_1`,
 - `unfamiliar_2`,
 .

 Nuisance covariates included:
```

(continues on next page)

(continued from previous page)

```
 - `trans_?`,
 - `rot_?`,

to account for residual motion artefacts,
 .



 ## References

This method section was automatically generated using BIDSpm
(v2.2.0; https://github.com/cpp-lln-lab/bidspm; DOI: https://doi.org/10.5281/zenodo.
↪3554331)
and octache (https://github.com/Remi-Gau/Octache).
```

5.6 Parametric modulation

Those are not yet fully implemented but there is an example of how to get started in the face repetition demo folder.

```
{
   "Name": "parametric modulation",
   "BIDSModelVersion": "1.0.0",
   "Description": "model for face repetition",
   "Input": {
      "task": [
         "facerepetition"
      ],
      "space": [
         "IXI549Space"
      ]
   },
   "Nodes": [
      {
         "Level": "Run",
         "Name": "parametric",
         "GroupBy": [
            "run",
            "subject"
         ],
         "Transformations": {
            "Description": "merge the familiarity and repetition column to create the␣
↪trial type column, also create parametric modulation variable",
            "Transformer": "bidspm",
            "Instructions": [
               {
                  "Name": "Concatenate",
                  "Input": [
                     "face_type",
```

(continues on next page)

(continued from previous page)

```
                "repetition_type"
            ],
            "Output": "trial_type"
        },
        {
            "Name": "Copy",
            "Input": "lag",
            "Output": "pmod_lag"
        },
        {
            "Name": "Power",
            "Input": "pmod_lag",
            "Value": 2,
            "Output": "pmod_lag_squared"
        }
    ]
},
"Model": {
    "X": [
        "trial_type.famous_1",
        "trial_type.famous_2",
        "trial_type.unfamiliar_1",
        "trial_type.unfamiliar_2",
        "trans_?",
        "rot_?"
    ],
    "HRF": {
        "Variables": [
            "trial_type.famous_1",
            "trial_type.famous_2",
            "trial_type.unfamiliar_1",
            "trial_type.unfamiliar_2"
        ],
        "Model": "spm + derivative"
    },
    "Type": "glm",
    "Options": {
        "HighPassFilterCutoffHz": 0.0078,
        "Mask": {
            "suffix": [
                "mask"
            ],
            "desc": [
                "brain"
            ]
        }
    },
    "Software": {
        "SPM": {
            "SerialCorrelation": "AR(1)"
        }
    }
```

(continues on next page)

(continued from previous page)

```
        },
        "DummyContrasts": {
            "Test": "t",
            "Contrasts": [
                "trial_type.famous_1",
                "trial_type.famous_2",
                "trial_type.unfamiliar_1",
                "trial_type.unfamiliar_2"
            ]
        },
        "Contrasts": [
            {
                "Name": "faces_gt_baseline",
                "ConditionList": [
                    "trial_type.famous_1",
                    "trial_type.famous_2",
                    "trial_type.unfamiliar_1",
                    "trial_type.unfamiliar_2"
                ],
                "Weights": [
                    1,
                    1,
                    1,
                    1
                ],
                "Test": "t"
            },
            {
                "Name": "faces_lt_baseline",
                "ConditionList": [
                    "trial_type.famous_1",
                    "trial_type.famous_2",
                    "trial_type.unfamiliar_1",
                    "trial_type.unfamiliar_2"
                ],
                "Weights": [
                    -1,
                    -1,
                    -1,
                    -1
                ],
                "Test": "t"
            }
        ]
    }
  ]
}
```

See the help section of `convertOnsetTsvToMat` for more information.

5.7 Examples

There are several examples of models in the model zoo along with links to their datasets.

Several of the *demos* have their own model and you can find several "dummy" models (without corresponding data) used for testing

in this folder.

An example of JSON file could look something like that:

```
{
  "Name": "vislocalizer",
  "BIDSModelVersion": "1.0.0",
  "Description": "contrasts for the visual localizer",
  "Input": {
    "task": [
      "vislocalizer"
    ],
    "space": [
      "IXI549Space"
    ]
  },
  "Nodes": [
    {
      "Level": "Run",
      "Name": "run_level",
      "GroupBy": [
        "run",
        "subject"
      ],
      "Model": {
        "Type": "glm",
        "X": [
          "trial_type.VisMot",
          "trial_type.VisStat",
          "trial_type.missing_condition",
          "trans_?",
          "rot_?"
        ],
        "HRF": {
          "Variables": [
            "trial_type.VisMot",
            "trial_type.VisStat"
          ],
          "Model": "spm+derivative"
        },
        "Options": {
          "HighPassFilterCutoffHz": 0.008
        },
        "Software": {
          "SPM": {
            "InclusiveMaskingThreshold": 0,
            "SerialCorrelation": "FAST"
```

(continues on next page)

(continued from previous page)

```
        }
      }
    },
    "DummyContrasts": {
      "Test": "t",
      "Contrasts": [
        "trial_type.VisMot",
        "trial_type.VisStat"
      ]
    },
    "Contrasts": [
      {
        "Name": "VisMot_&_VisStat",
        "ConditionList": [
          "trial_type.VisMot",
          "trial_type.VisStat"
        ],
        "Weights": [
          1,
          1
        ],
        "Test": "t"
      },
      {
        "Name": "VisMot_&_VisStat_lt_baseline",
        "ConditionList": [
          "trial_type.VisMot",
          "trial_type.VisStat"
        ],
        "Weights": [
          -1,
          -1
        ],
        "Test": "t"
      }
    ]
  },
  {
    "Level": "Subject",
    "Name": "subject_level",
    "GroupBy": [
      "contrast",
      "subject"
    ],
    "Model": {
      "Type": "glm",
      "X": [
        1
      ]
    },
    "DummyContrasts": {
      "Test": "t"
```

(continues on next page)

(continued from previous page)

```
      }
    },
    {
      "Level": "Dataset",
      "Name": "dataset_level",
      "GroupBy": [
        "contrast"
      ],
      "Model": {
        "Type": "glm",
        "X": [
          1
        ]
      },
      "DummyContrasts": {
        "Test": "t"
      }
    }
  ]
}
```

CHAPTER

SIX

PIPELINE DEFAULTS

Defaults of the pipeline.

6.1 checkOptions

src.defaults.**checkOptions**(*opt*)

Check the option inputs and add any missing field with some defaults

USAGE:

```
opt = checkOptions(opt)
```

Parameters

opt (structure) – Options chosen for the analysis. See also: checkOptions

Returns

- **opt**

  the option structure with missing values filled in by the defaults.

IMPORTANT OPTIONS (with their defaults):

- **generic**
  - opt.dir: TODO EXPLAIN
  - opt.groups = {''} - group of subjects to analyze
  - opt.subjects = {[]} - suject to run in each group space where we conduct the analysis are located. See setDerivativesDir() for more information.
  - opt.space = {'individual', 'IXI549Space'} - Space where we conduct the analysis
  - opt.taskName
  - opt.query = struct('modality', {{'anat', 'func'}}) - a structure used to specify subset of files to only run analysis on. Default = struct('modality', {{'anat', 'func'}}) See bids.query to see how to specify.

    Warning: opt.query might be progressively deprecated in favor of opt.bidsFilterFile that allows using different filters for T1w and bold data.

  - `opt.bidsFilterFile` - Sets how to define a typical images "bold", "T1w"... in terms of their bids entities. The default value is:

```
struct('fmap', struct('modality', 'fmap'), ...
       'bold', struct('modality', 'func', 'suffix', 'bold'), ...
       't2w',  struct('modality', 'anat', 'suffix', 'T2w'), ...
       't1w',  struct('modality', 'anat', 'space', '', 'suffix', 'T1w'),␣
↪...
       'roi',  struct('modality', 'roi', 'suffix', 'mask'));
```

- **preprocessing**
  - `opt.realign.useUnwarp = true`
  - `opt.useFieldmaps = true` - when set to `true` the preprocessing pipeline will look for the voxel displacement maps (created by `bidsCreateVDM()`) and will use them for realign and unwarp.
  - `opt.fwhm.func = 6` - FWHM to apply to the preprocessed functional images.
- **statistics**
  - `opt.model.file = ''` - path to the BIDS model file that contains the model to speficy and the contrasts to compute.
  - `opt.fwhm.contrast = 6` - FWHM to apply to the contrast images before bringing them at the group level.
  - `'opt.model.designOnly'` = if set to `true`, the GLM will be set up without associating any data to it. Can be useful for quick design matrix inspection before running estimation.

OTHER OPTIONS (with their defaults):

- **generic**
  - `opt.verbosity = 1;` - Set it to `0` if you want to see less output on the prompt.
  - `opt.dryRun = false` - Set it to `true` in case you don't want to run the analysis.
  - `opt.pipeline.type = 'preproc'` - Switch it to `stats` when running GLMs.
  - `opt.pipeline.name`
  - `opt.zeropad = 2` - number of zeros used for padding subject numbers, in case subjects should be fetched by their number `1` and not their label `01'`.
  - `opt.rename = true` - to skip renaming files with `bidsRename()`. Mostly for debugging as the ouput files won't be usable by any of the stats workflows.
  - `opt.msg.color = blue` - default font color of the prompt messages.
- **preprocessing**
  - `opt.anatOnly = false` - to only preprocess the anatomical file
  - `opt.segment.force = false` - set to `true` to ignore previous output of the segmentation and force to run it again
  - `opt.skullstrip.mean = false` - to skulstrip mean functional image
  - `opt.skullstrip.threshold = 0.75` - Threshold used for the skull stripping. Any voxel with `p(grayMatter) + p(whiteMatter) + p(CSF) > threshold` will be included in the mask.
  - `opt.skullstrip.do = true` - Set to `true` to skip skullstripping
  - `opt.stc.skip = false` - boolean flag to skip slice time correction or not.

  - `opt.stc.referenceSlice = []` - reference slice (in seconds) for the slice timing correction. If left emtpy the mid-volume acquisition time point will be selected at run time.
  - `opt.stc.sliceOrder = []` - To be used if SPM can't extract slice info. NOT RECOMMENDED, if you know the order in which slices were acquired, you should be able to recompute slice timing and add it to the json files in your BIDS data set.
  - `opt.funcVoxelDims = []` - Voxel dimensions to use for resampling of functional data at normalization.

- **preprocessing QA** (see `functionalQA`)

  opt.QA.anat.do = true;

  `opt.QA.func` contains a lot of options used by `spmup_first_level_qa`

  - `opt.QA.func.do = true` skips QA if set to false
  - `opt.QA.func.carpetPlot = true` to plot carpet plot
  - `opt.QA.func.MotionParameters = 'on'`
  - `opt.QA.func.FramewiseDisplacement = 'on'`
  - `opt.QA.func.Voltera = 'on'`
  - `opt.QA.func.Globals = 'on'`
  - `opt.QA.func.Movie = 'on'` ; set it to `off` to skip generating movies of the time series
  - `opt.QA.func.Basics = 'on'`

- **statistics**
  - `opt.glm.roibased.do = false` must be set to `true` to use the `bidsRoiBasedGLM` workflow
  - `opt.glm.useDummyRegressor = false` to add dummy regressors when a condition is missing from a run. See `bidsModelSelection()` for more information.
  - `opt.glm.maxNbVols = Inf` sets the maximum number of volumes to include in a run in a subject level GLM. This can be useful if some time series have more volumes than necessary.
  - `opt.glm.keepResiduals = false` keeps the subject level GLM residuals
  - `opt.QA.glm.do = false` - If set to `true` the residual images of a GLM at the subject levels will be used to estimate if there is any remaining structure in the GLM residuals (the power spectra are not flat) that could indicate the subject level results are likely confounded. See `plot_power_spectra_of_GLM_residuals.m` and Accurate autocorrelation modeling substantially improves fMRI reliability for more info.

`src.defaults.`**`setDirectories`**`(`*`opt`*`)`

USAGE:

```
opt = setDirectories(opt)
```

`src.defaults.`**`defaultResultsStructure`**`()`

`src.defaults.`**`defaultContrastsStructure`**`()`

6.2 spm_my_defaults

Some more SPM options can be set in the `src.defaults.spm_my_defaults.m()`.

`src.defaults.`**spm_my_defaults**`()`

USAGE:

```
spm_my_defaults()
```

This is where we set the defaults we want to use. These will overide the spm defaults. When "not enough" information is specified in the batch files, SPM falls back on the defaults to fill in the blanks. This allows to make the scripts simpler.

6.3 statistics defaults

Note that some of the defaults value may be over-ridden by the content of the `opt` structure but also by the content of your BIDS stats model.

6.3.1 auto-correlation modelisation

Use of FAST [OARW19] and not AR1 for auto-correlation modelisation.

Using FAST does not seem to affect results on time series with "normal" TRs but improves results when using sequences: it is therefore used by default in this pipeline.

Check the *the relevant section of the BIDS stats model* to know how to change this value.

6.4 SPM to BIDS filename conversion

`src.defaults.`**set_spm_2_bids_defaults**`(`*opt*`)`

set default map for renaming for bidspm

USAGE:

```
opt = set_spm_2_bids_defaults(opt)
```

Further renaming mapping can then be added, changed or removed through the `opt.spm_2_bids object`.

6.5 list of defaults

CHAPTER

SEVEN

DEMOS

```
demos/
    ├── face_repetition
    ├── lesion_detection
    ├── MoAE
    ├── openneuro
    ├── tSNR
    └── vismotion
```

The demos show show you different way to use bidspm.

7.1 MoAE

```
/demos/MoAE
    ├── models
    └── options
```

This "Mother of All Experiments" is based on the block design dataset of SPM.

In the `options` folder has several examples of how to encode the options of your analysis in a json file.

In the `models` shows the BIDS statistical model used to run the GLM of this demo.

demos.MoAE.**moae_01_bids_app**

MoAE demo

demos.MoAE.**moae_fmriprep**

This script will run the FFX and contrasts on it of the MoAE dataset using the fmriprep preprocessed data

If you want to get the preprocessed data and you have datalad on your computer you can run the following commands to get the necesary data:

```
datalad install --source git@gin.g-node.org:/SPM_datasets/spm_moae_fmriprep.git \
        inputs/fmriprep
cd inputs/fmriprep && datalad get *.json \
                  */*/*tsv \
                  */*/*json \
                  */*/*desc-preproc*.nii.gz \
                  */*/*desc-brain*.nii.gz
```

Otherwise you also grab the data from OSF: https://osf.io/vufjs/download

demos.MoAE.**moae_02_create_roi_extract_data**

This script shows how to create a ROI and extract data from it.

> **Warning:** This is "double dipping" as we use the same data to create the ROI we are going to extract the value from.

(C) Copyright 2021 Remi Gau

demos.MoAE.**moae_03_slice_display**

This script shows how to display the results of a GLM by having on the same image:

- the beta estimates
- the t statistics
- ROI contours

(C) Copyright 2021 Remi Gau

7.2 Face repetition

This is based on the event related design dataset of SPM.

demos.face_repetition.**face_rep_01_bids_app**

This script will download the face repetition dataset from SPM and will run the basic preprocessing.

Download

- downloads and BIDSify the dataset from the FIL website

Preprocessing

- copies the necessary data from the raw to the derivative folder,
- runs slice time correction
- runs spatial preprocessing

those are otherwise handled by the workflows:

- `bidsCopyInputFolder.m`
- `bidsSTC.m`
- `bidsSpatialPrepro.m`

type *bidspm help* or *bidspm('action', 'help')* or see this page: https://bidspm.readthedocs.io/en/stable/bids_app_api.html for more information on what parameters are obligatory or optional

(C) Copyright 2022 Remi Gau

demos.face_repetition.**face_rep_01_anat_only**

This show how an anat only pipeline would look like.

Download

- downloads and BIDSify the dataset from the FIL website

Preprocessing

- copies the necessary data from the raw to the derivative folder,

- runs spatial preprocessing

those are otherwise handled by the workflows:

- `bidsCopyInputFolder.m`
- `bidsSpatialPrepro.m`

type *bidspm help* or *bidspm('action', 'help')* or see this page: https://bidspm.readthedocs.io/en/stable/bids_app_api.html for more information on what parameters are obligatory or optional

(C) Copyright 2022 Remi Gau

`demos.face_repetition.face_rep_02_stats`

> **Warning:** This script assumes you have already preprocessed the data with face_rep_01_bids_app.m

stats

This script will run the FFX and contrasts on the the face repetition dataset from SPM.

- GLM specification + estimation
- compute contrasts
- show results

that are otherwise handled by the workflows

- `bidsFFX.m`
- `bidsResults.m`

Note: Results might be a bit different from those in the SPM manual as some default options are slightly different in this pipeline (e.g use of FAST instead of AR(1), motion regressors added)

---

type *bidspm help* or *bidspm('action', 'help')* or see this page: https://bidspm.readthedocs.io/en/stable/bids_app_api.html for more information on what parameters are obligatory or optional

(C) Copyright 2022 Remi Gau

`demos.face_repetition.face_rep_03_roi_analysis`

Creates a ROI in MNI space from the retinotopic probablistic atlas.

Creates its equivalent in subject space (inverse normalization).

Then uses marsbar to run a ROI based GLM

(C) Copyright 2019 Remi Gau

`demos.face_repetition.face_rep_resolution`

7.3 Visual motion localizers

Small demo using visual motion localizer data to show how to set up an analysis with BIDSpm from scratch with datalad.

7.3.1 Using BIDSpm and datalad

Ideally better to use the datalad fMRI template we have set up, this shows a set by step approach.

Note: The bash script `vismotion_demo.sh` will run all the steps described here in one fell swoop.

You can run it by typing the following from within the bidspm/demos/vismotion

```
bash vismotion_demo.sh
```

Set up

Create a new datalad dataset with a YODA config

```
datalad create -c yoda visual_motion_localiser
cd visual_motion_localiser
```

Add the BIDSpm code as a sub-dataset, checkout the dev branch ands initializes

all submodules.

```
datalad install \
    -d . \
    -s https://github.com/cpp-lln-lab/bidspm.git \
    --branch dev \
    -r \
    code/bidspm
```

In case you get some errors when installing the submodules you might have to initialize them manually, and update your dataset with that update

```
cd code/bidspm
git checkout dev
git submodule update --init --recursive && git submodule update --recursive
cd ..
datalad save -m 'update BIDSpm submodules'
```

Now let's get the raw data as a subdataset and put it in an `inputs/raw` folder.

The data from the CPP lab is openly available on GIN: https://gin.g-node.org/cpp-lln-lab/Toronto_VisMotionLocalizer_MR_raw

Note that to install it you will need to have set up Datalad to play nice with GIN: see the datalad handbook

This will install the data:

```
datalad install -d . \
                -s git@gin.g-node.org:/cpp-lln-lab/Trento_VisMotionLocalizer_MR_raw.git \
                --recursive \
                inputs/raw
```

After this your datalad dataset should look something like this:

```
├── code
│   └── bidspm
└── inputs
    └── raw
        ├── derivatives
        │   └── fmriprep
        ├── sub-con07
        ├── sub-con08
        └── sub-con15
```

To finish the setup you need to download the data:

```
cd inputs/raw
datalad get .
```

Note that you could have installed the dataset and got the data in one command:

Running the analysis

Start matlab and run the `step_1_preprocess.m` and `step_2_stats.m` scripts.

In the end your whole analysis should look like this.

```
├── code
│   └── bidspm
│       ├── binder
│       ├── demos
│       │   ├── face_repetition
│       │   ├── lesion_detection
│       │   ├── MoAE
│       │   ├── openneuro
│       │   ├── tSNR
│       │   └── vismotion          # <--- your scrips are there
│       ├── docs
│       ├── lib
│       ├── manualTests
│       ├── notebooks
│       ├── src
│       ├── templates
│       └── tests
├── inputs
│   └── raw                        # <--- input data
│       ├── derivatives
│       │   └── fmriprep           # <--- fmriprep data
│       ├── sub-con07
│       │   └── ses-01
```

(continues on next page)

(continued from previous page)

```
│       ├── sub-con08
│       │   └── ses-01
│       └── sub-con15
│           └── ses-01
└── outputs
    └── derivatives
        ├── bidspm-preproc        # <--- smoothed data
        │   ├── jobs
        │   ├── sub-con07
        │   ├── sub-con08
        │   └── sub-con15
        └── bidspm-stats          # <--- stats output
            ├── jobs
            ├── sub-con07
            ├── sub-con08
            └── sub-con15
```

7.4 Openneuro based demos

7.4.1 Demos based on openneuro datasets

- ds000001: one task, one session, several runs
- ds000114: several tasks, several sessions, one or several runs depending on task
- ds001168: resting state, several sessions, several acquisition, fieldmaps, physio data
- ds002799: resting state and taks, several sessions, with fmriprep data

Download with datalad

All those data can be installed with datalad.

Datalad datasets can be accessed via their siblings on: https://github.com/OpenNeuroDatasets

Check the content of the `Makefile` to see the code snippets you need to run to install those datasets.

Otherwise you can also get them by using the Datalad superdataset.

For example:

```
datalad install ///
cd datasets.datalad.org/
datalad install openneuro
datalad install openneuro/dsXXXXXX
cd openneuro/dsXXXXXX
# get rest data first subject
datalad get /openneuro/dsXXXXXX/sub-0001/func/sub-0001*
```

CHAPTER

EIGHT

ARCHITECTURE

At the highest levels bidspm is organized in workflows:

- they all start with the prefix `bids` (for example `bidsRealignReslice`)
- they are in the folder *src.workflows*
- they run on all the subjects specified in the `options` structure (see the *Set up* section).

Most workflows run by creating matlab batches that are saved as `.mat` files in a `jobs` then passed to the SPM jobman to run. To do this the workflows call "batch creating functions":

- all start with the prefix `setBatch` (for example `setBatchCoregistration`).
- are in the folder *src.batches*.

Many workflows include some post-processing steps (like file renaming) after the execution of the batch, so in many cases the output of running just the batch and running the whole workflow will be different.

Preprocessing, *Statistics* and *Fieldmaps* handling have their own document pages.

Other workflows, batches and related helper functions are listed below.

8.1 Workflows

8.1.1 HRF estimation

Relies on the resting-state HRF toolbox.

`src.workflows.`**`bidsRsHrf`**`(`*opt*`)`

Use the rsHRF to estimate the HRF from resting state data.

USAGE:

```
bidsRsHrf(opt)
```

Parameters

`opt` (`structure`) – Options chosen for the analysis. See also: checkOptions `checkOptions()` and `loadAndCheckOptions()`.

8.1.2 Other

src.workflows.**bidsCopyInputFolder**(*varargin*)

Copies data from the `opt.dir.input` folder to the `opt.dir.output` folder

Then it will search the derivatives directory for any zipped `*.gz` image and uncompress the files for the task of interest.

USAGE:

```
bidsCopyInputFolder(opt, 'unzip', true, 'force', true)
```

Parameters

- **opt** (`structure`) – Options chosen for the analysis. See also: `checkOptions()` and `loadAndCheckOptions()`.
- **unzip** (`boolean`) – defaults to true
- **unzip** – defaults to true

See also: bids.copy_to_derivative

src.workflows.**bidsRename**(*opt*)

Renames SPM output into BIDS compatible files.

USAGE:

```
bidsRename(opt)
```

Parameters

opt (`structure`) – Options chosen for the analysis. See also: checkOptions `checkOptions()` and `loadAndCheckOptions()`.

See the spm_2_bids submodule and `defaults.set_spm_2_bids_defaults` for more info.

8.1.3 Worflow helper functions

To be used if you want to create a new workflow.

src.workflows.**setUpWorkflow**(*opt*, *workflowName*, *bidsDir*, *indexData*)

Calls some common functions to:

- check the configuraton,
- remove some old files from an eventual previous crash
- loads the layout of the BIDS dataset
- tries to open a graphic window

USAGE:

```
[BIDS, opt, group] = setUpWorkflow(opt, workflowName, [bidsDir], indexData)
```

Parameters

- **opt** (`structure`) – Options chosen for the analysis. See also: checkOptions

- **workflowName** (char) – name that will be printed on screen
- **bidsDir** –
- **bidsDir** – fullpath
- **indexData** – Set to `false` if you want to skip the datindexing with `getData`. Can be useful for some group level workflow where indexing will happen later at the batch level. This will also skip updating the subject list done by `getData`. Default to `true`.
- **indexData** – boolean

Returns

- **BIDS**
  (structure) returned by `getData`
- **opt**
  options checked

src.workflows.**saveAndRunWorkflow**(*matlabbatch*, *batchName*, *opt*, *subLabel*)

Saves the SPM matlabbatch and runs it

USAGE:

```
saveAndRunWorkflow(matlabbatch, batchName, opt, [subLabel])
```

Parameters

- **matlabbatch** (structure) – list of SPM batches
- **batchName** (char) – name of the batch
- **opt** (structure) – Options chosen for the analysis. See also: checkOptions
- **subLabel** (char) – subject label

src.workflows.**cleanUpWorkflow**(*opt*)

USAGE:

```
cleanUpWorkflow(opt)
```

src.workflows.**returnDependency**(*opt*, *type*)

Use to create dependencies between batches in workflows.

USAGE:

```
dep = returnDependency(opt, type)
```

8.2 Batches

src.batches.**setBatchSelectAnat**(*matlabbatch*, *BIDS*, *opt*, *subLabel*)

Creates a batch to set an anatomical image

USAGE:

```
matlabbatch = setBatchSelectAnat(matlabbatch, BIDS, opt, subLabel)
```

Parameters

- **matlabbatch** (structure) – list of SPM batches
- **BIDS** (structure) – dataset layout. See also: bids.layout, getData.
- **opt** (structure) – Options chosen for the analysis. See also: `checkOptions()` and `loadAndCheckOptions()`.
- **subLabel** (char) – subject label

Returns

matlabbatch
(structure)

matlabbatch = setBatchSelectAnat(matlabbatch, BIDS, opt, subLabel)

- image type = opt.bidsFilterFiler.t1w.suffix (default = T1w)
- session to select the anat from = opt.bidsFilterFiler.t1w.ses (default = 1)

We assume that the first anat of that type is the "correct" one

src.batches.**setBatchPrintFigure**(*matlabbatch*, *opt*, *figureName*)

template to creae new setBatch functions

USAGE:

```
matlabbatch = setBatchPrintFigure(matlabbatch, figureName)
```

Parameters

- **matlabbatch** –
- **figureName** (string) –

Returns

- **matlabbatch**
(structure) The matlabbatch ready to run the spm job

src.batches.**setBatchMeanAnatAndMask**(*matlabbatch*, *opt*, *outputDir*)

Creates batxh to create mean anatomical image and a group mask

USAGE:

```
matlabbatch = setBatchMeanAnatAndMask(matlabbatch, opt, funcFWHM, outputDir)
```

Parameters

- **matlabbatch** (structure) –
- **opt** (structure) – Options chosen for the analysis. See also: checkOptions
- **outputDir** (string) –

Returns

- **matlabbatch**
(structure)

src.batches.**setBatchRsHRF**(*matlabbatch*, *BIDS*, *opt*, *subLabel*)

Set the batch for realign / realign and reslice / realign and unwarp

USAGE:

```
matlabbatch = setBatchRsHRF(matlabbatch, BIDS, opt, subLabel)
```

Parameters

- **matlabbatch** (structure) – SPM batch
- **BIDS** (structure) – dataset layout. See also: bids.layout, getData.
- **opt** (structure) – Options chosen for the analysis. See also: `checkOptions()` and `loadAndCheckOptions()`.
- **subLabel** (char) – subject label

Returns

- **matlabbatch**
  (structure) (dimension)

src.batches.**setBatchImageCalculation**(*varargin*)

Set a batch for a image calculation

USAGE:

```
matlabbatch = setBatchImageCalculation(matlabbatch, input, output, outDir,␣
↪expression)
```

Parameters

- **matlabbatch** (structure) –
- **input** (cell) – list of images
- **output** (char) – name of the output file
- **outDir** (char) – output directory
- **expression** (char) – mathematical expression to apply (for example '(i1+i2)>3')
- **expression** – data type that must be one of the following: - 'uint8' - 'int16' (default) - 'int32' - 'float32' - 'float64' - 'int8' - 'uint16' - 'uint32'

See `spm_cfg_imcalc.m` for more information:

```
``edit(fullfile(spm('dir'), 'config', 'spm_cfg_imcalc.m'))``
```

Returns

- **matlabbatch**

src.batches.**setBatch3Dto4D**(*matlabbatch*, *opt*, *volumesList*, *RT*, *outputName*, *dataType*)

Set the batch for 3D to 4D conversion

USAGE:

```
matlabbatch = setBatch3Dto4D(matlabbatch, volumesList, RT, [outputName], [dataType])
```

Parameters

- **matlabbatch** (structure) –
- **volumesList** (array) – List of volumes to be converted in a single 4D brain
- **outputName** (char) – The string that will be used to save the 4D brain
- **dataType** (integer) – It identifies the data format conversion
- **RT** (float) – It identifies the TR in seconds of the volumes to be written in the 4D file header

Returns

- **matlabbatch**
  (structure) The matlabbatch ready to run the spm job

dataType:

- 0: SAME
- 2: UINT8 - unsigned char
- 4: INT16 - signed short
- 8: INT32 - signed int
- 16: FLOAT32 - single prec. float
- 64: FLOAT64 - double prec. float

src.batches.**saveMatlabBatch**(*matlabbatch*, *batchType*, *opt*, *subLabel*)

#ok<INUSL>

Saves the matlabbatch job in a .m file. Environment information are saved in a .json file.

% USAGE:

```
saveMatlabBatch(matlabbatch, batchType, opt, [subLabel])
```

Parameters

- **matlabbatch** (structure) –
- **batchType** (char) –
- **opt** (structure) – Options chosen for the analysis. See also: checkOptions
- **subLabel** (char) –

The .m file can directly be loaded with the SPM batch or run directly by SPM standalone or SPM docker.

The .json file also contains heaps of info about the "environment" used to set up that batch including the version of:

- OS,
- MATLAB or Octave,
- SPM,
- bidspm

This can be useful for methods writing though if the the batch is generated in one environment and run in another (for example set up the batch with Octave on Mac OS and run the batch with Docker SPM), then this information will be of little value in terms of computational reproducibility.

src.batches.lesion.**setBatchLesionOverlapMap**(*matlabbatch*, *BIDS*, *opt*, *subLabel*)

Creates a batch for the lesion overlap map

Requires the ALI toolbox: https://doi.org/10.3389/fnins.2013.00241

USAGE:

```
matlabbatch = setBatchLesionOverlapMap(matlabbatch, BIDS, opt, subLabel)
```

Parameters
matlabbatch (structure) – list of SPM batches

Returns

- **matlabbatch**
  (structure)

src.batches.lesion.**setBatchLesionSegmentation**(*matlabbatch*, *BIDS*, *opt*, *subLabel*)

Creates a batch to segment the anatomical image for lesion detection

Requires the ALI toolbox: https://doi.org/10.3389/fnins.2013.00241

USAGE:

```
matlabbatch = setBatchSegmentationDetectLesion(matlabbatch, BIDS, opt, subLabel)
```

Parameters
matlabbatch (structure) – list of SPM batches

Returns

- **matlabbatch**
  (structure)

src.batches.lesion.**setBatchLesionAbnormalitiesDetection**(*matlabbatch*, *opt*, *images*)

Creates a batch to detect lesion abnormalities

Requires the ALI toolbox: https://doi.org/10.3389/fnins.2013.00241

USAGE:

```
matlabbatch = setBatchLesionAbnormalitiesDetection(matlabbatch, BIDS, opt, subLabel)
```

Parameters
matlabbatch (structure) – list of SPM batches

Returns

- **matlabbatch**
  (structure)

CHAPTER

NINE

STATISTICS

Make sure you are familiar with the *BIDS stats model JSON file*, before you embark on to statistical analysis.

9.1 Statistics workflows

Note: The illustrations in this section mix what the files created by each workflow and the functions and are called by it. In this sense they are not pure DAGs (directed acyclic graphs) as the `*.m` files mentioned in them already exist.

9.1.1 Subject level

`src.workflows.stats.`**`bidsFFX`**(*varargin*)

- specify the subject level fMRI model
- estimates it
- do both in one go
- or compute the contrasts

To run this workflows get the BOLD input images from derivatives BIDS dataset that contains the preprocessed data and get the condition, onsets, durations from the events files in the raw BIDS dataset.

For the model specification, if `opt.model.designOnly` is set to `true`, then it is possible to specify a model with no data: this can useful for debugging or to quickly inspect designs specification.

For the model estimation, it is possible to do some rough QA, by setting `opt.QA.glm.do = true`.

USAGE:

```
bidsFFX(action, opt, 'nodeName', 'run_level')
```

Parameters
`action` (`char`) – Action to be conducted

- `'specify'` to specify the fMRI GLM
- `'specifyAndEstimate'` for fMRI design + estimate
- `'contrasts'` to estimate contrasts.

Parameters

- **opt** (structure) – Options chosen for the analysis. See also: checkOptions() and loadAndCheckOptions().
- **nodeName** (char) – Only for action 'contrasts'. Specifies which Node to work on.

See also: setBatchSubjectLevelGLMSpec, setBatchSubjectLevelContrasts

After the specification step an output folder is created. To get the fullpath of that folder you can use:

```
getFFXdir(subLabel, opt)
```

A typical folder will contain:

```
bidspm-stats/sub-01/stats/task-audio_space-IXI549Space_FWHM-6
├── SPM.mat
├── sub-01_task-audio_space-IXI549Space_desc-beforeEstimation_designmatrix.png
├── sub-01_task-audio_run-01_desc-confounds_regressors.mat
├── sub-01_task-audio_run-01_desc-confounds_regressors.tsv
├── sub-01_task-audio_run-01_onsets.mat
└── sub-01_task-audio_run-01_onsets.tsv
```

Each run should have a pair of tsv/mat files:

- One that summarises the onsets used for that design.
- One that summarises the regressors confounds used for that design.

In most cases those are going to be a subset of the content:

- of the _events.tsv from the raw BIDS dataset
- of the _regressors.tsv from the deriratives BIDS dataset containing the preprocessed data.

What part of the _events.tsv and _regressors.tsv gets into the final GLM specification depends on the BIDS statistical model used.

The mat files can directly be ingested by SPM: the TSV files are there for both logging and interoperability.

Fig. 1: Subject level GLM specification workflow for model specification

src.workflows.stats.**bidsConcatBetaTmaps**(*opt*, *deleteTmaps*)

Make 4D images of beta and t-maps for the MVPA.

USAGE:

```
concatBetaImgTmaps(opt, deleteIndTmaps)
```

Parameters

- **opt** (structure) – Options chosen for the analysis. See also: checkOptions
- **deleteIndTmaps** ((boolean)) – decide to delete t-maps. Default to false.

A valid BIDS stats model is required for this workflow: this is because the beta images to concatenate are those of the conditions mentioned in the DummyContrasts of the RUN level of the BIDS stats model.

When concatenating betamaps:

- Ensures that there is only 1 image per "contrast".
- Creates a tsv that lists the content of the 4D image.
- This TSV is in the subject level GLM folder where the beta map came from.
- This TSV file is named sub-subLabel_task-taskName_space-space_labelfold.tsv.

9.1.2 Group level

src.workflows.stats.**bidsRFX**(*varargin*)

- smooths all contrast images created at the subject level

OR

- creates a mean structural image and mean mask over the sample

OR

- specifies and estimates the group level model,
- computes the group level contrasts.

USAGE:

```
bidsRFX(action, opt, 'nodeName', '')
```

Parameters

- **action** (char) – Action to be conducted: 'smoothContrasts' or 'RFX' or 'meanAnatAndMask' or 'contrast'
- **opt** (structure) – Options chosen for the analysis. See also: checkOptions checkOptions() and loadAndCheckOptions().
- **nodeName** (char) – name of the BIDS stats model to run analysis on

9.1.3 Compute results

src.workflows.stats.**bidsResults**(*varargin*)

Computes the results for a series of contrast that can be specified at the run, subject or dataset step level (see contrast specification following the BIDS stats model specification).

USAGE:

```
bidsResults(opt,'nodeName', '')
```

Parameters

- **opt** (structure) – Options chosen for the analysis. See also: checkOptions() and loadAndCheckOptions().
- **nodeName** (char or cellstr) – name of the BIDS stats model Node(s) to show results of

See also: setBatchSubjectLevelResults, setBatchGroupLevelResults

Below is an example of how specify the option structure to getsome speific results outputs for certain contrasts.

See the online documentation for example of those outputs.

The field opt.results allows you to get results from several Nodes from the BIDS stats model. So you could run bidsResults once to view results from the subject and the dataset level.

Specify a default structure result for this node:

```
opt.results(1) = returnDefaultResultsStructure();
```

Specify the Node name (usually "run_level", "subject_levle" or "dataset_level"):

```
opt.results(1).nodeName = 'subject_level';
```

Specify the name of the contrast whose resul we want to see. This must match one of the existing contrats (dummy contrast or contrast) in the BIDS stats model for that Node:

```
opt.results(1).name = 'listening_1';
```

For each contrat, you can adapt:

- voxel level threshold (p) [between 0 and 1]
- cluster level threshold (k) [positive integer]
- type of multiple comparison (MC):
  - 'FWE' is the defaut
  - 'FDR'
  - 'none'

You can thus specify something different for a second contrast:

```
opt.results(2).name = {'listening_lt_baseline'};
opt.results(2).MC =  'none';
opt.results(2).p = 0.01;
opt.results(2).k = 0;
```

Specify how you want your output (all the following are on false by default):

```
% simple figure with glass brain view and result table
opt.results(1).png = true();

% result table as a .csv: very convenient when comes the time to write papers
opt.results(1).csv = true();

% thresholded statistical map
opt.results(1).threshSpm = true();

% binarised thresholded statistical map (useful to create ROIs)
opt.results(1).binary = true();
```

You can also create a montage to view the results on several slices at once:

```
opt.results(1).montage.do = true();

% slices position in mm [a scalar or a vector]
opt.results(1).montage.slices = -0:2:16;

% slices orientation: can be 'axial' 'sagittal' or 'coronal'
% axial is default
opt.results(1).montage.orientation = 'axial';

% path to the image to use as underlay
% Will use the SPM MNI T1 template by default
opt.results(1).montage.background = ...
     fullfile(spm('dir'), 'canonical', 'avg152T1.nii');

% Can also be a structure to pick up the correct file for each subject
% opt.results(1).montage.background = struct('suffix', 'T1w', ...
%                                            'desc', 'preproc', ...
%                                            'modality', 'anat');
```

Finally you can export as a NIDM results zip files.

NIDM results is a standardized results format that is readable by the main neuroimaging softwares (SPM, FSL, AFNI). Think of NIDM as BIDS for your statistical maps. One of the main other advantage is that it makes it VERY easy to share your group results on neurovault (which you should systematically do).

- NIDM paper
- NIDM specification
- *NIDM results viewer for SPM <https://github.com/incf-nidash/nidmresults-spmhtml>*

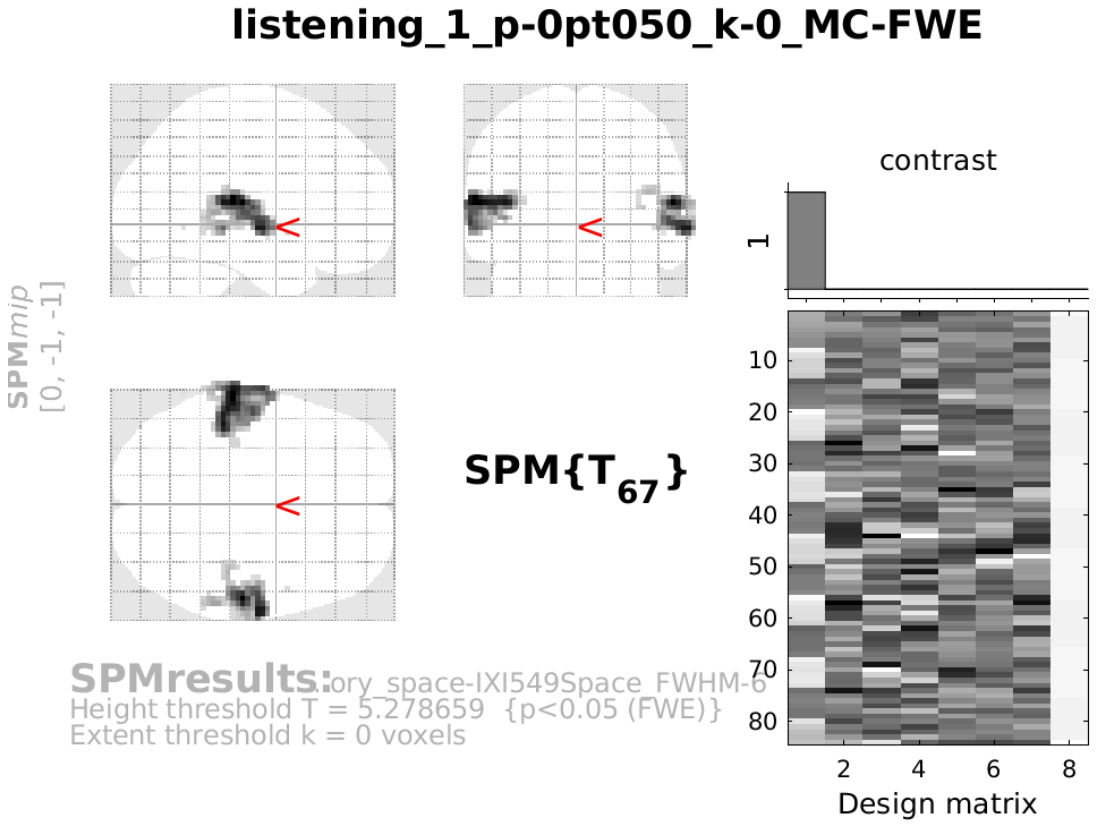
To generate NIDM results zip file for a given contrats simply:

```
opt.results(1).nidm = true();
```

`CSV output example`

bidspm also includes the `slice_display` code that allows you to plot on the same figure:

- beta values
- t values
- cluster boundaries



Statistics: ***p-values adjusted for search volume***

set-level		cluster-level				peak-level					mm	mm	mm
p	c	$p_{FWE\text{-}corr}$	$q_{FDR\text{-}corr}$	k_E	p_{uncorr}	$p_{FWE\text{-}corr}$	$q_{FDR\text{-}corr}$	T	(Z_E)	p_{uncorr}			
0.000	5	0.000	0.000	399	0.000	0.000	0.000	12.16	Inf	0.000	-63	-28	11
						0.000	0.000	11.33	Inf	0.000	-45	-34	11
						0.000	0.000	10.12	7.84	0.000	-66	-10	-1
		0.000	0.000	214	0.000	0.000	0.000	12.10	Inf	0.000	57	-22	11
						0.000	0.000	11.29	Inf	0.000	66	-13	-4
						0.000	0.003	7.07	6.09	0.000	60	-37	5
		0.001	0.020	5	0.012	0.006	0.200	5.81	5.21	0.000	69	-25	-4
		0.015	0.221	1	0.221	0.039	0.851	5.35	4.86	0.000	51	5	-7
		0.015	0.221	1	0.221	0.050	0.998	5.28	4.81	0.000	-63	-58	-4

table shows 3 local maxima more than 8.0mm apart

Height threshold: T = 5.28, p = 0.000 (0.050) Degrees of freedom = [1.0, 67.0]
Extent threshold: k = 0 voxels FWHM = 9.9 9.9 8.3 mm mm mm; 3.3 3.3 2.8 {voxels
Expected voxels per cluster, <k> = 0.715 Volume: 1784484 = 66092 voxels = 1953.4 resels
Expected number of clusters, <c> = 0.07 Voxel size: 3.0 3.0 3.0 mm mm mm; (resel = 30.35 vo
FWEp: 5.279, FDRp: 6.440, FWEc: 1, FDRc: 5

Fig. 2: Example of subject level results from the MoAE demo

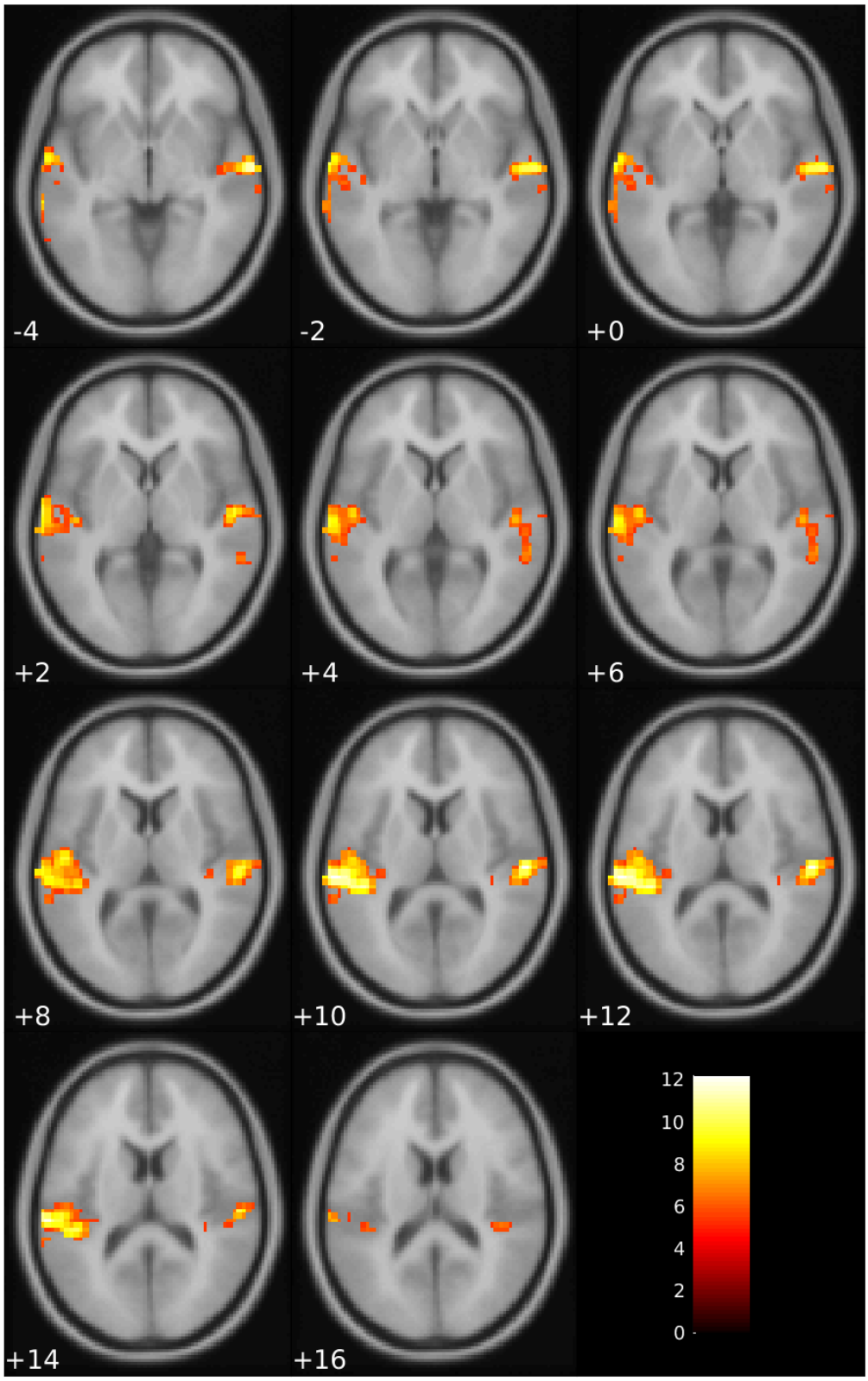


Fig. 3: Example of subject level montage from the MoAE demo

- ROI boundaries

An example of how to use it is available in the `moae_04_slice_display.m` script in the MoAE demo.

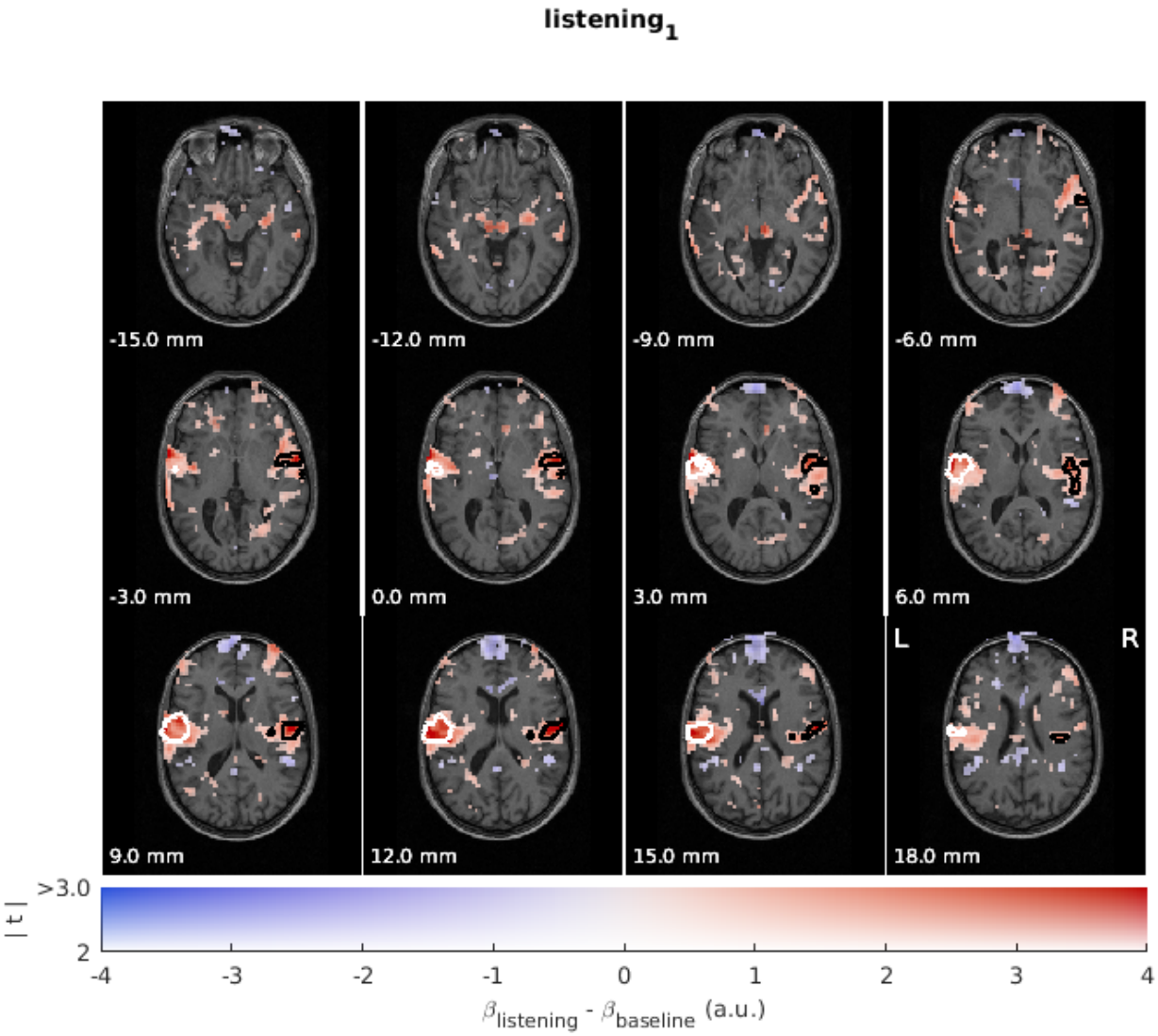


Fig. 4: Example of subject level slice display from the MoAE demo

9.1.4 Model selection

`src.workflows.stats.`**`bidsModelSelection`**(*varargin*)

Uses the MACS toolbox to perform model selection.

USAGE:

```
bidsModelSelection(opt, 'action', 'all')
```

Parameters

- **opt** (structure) – Options chosen for the analysis. See also: checkOptions See `checkOptions()` and `loadAndCheckOptions()`.
- **action** (char) – any of `'all'`, `'modelSpace'`, `'cvLME'`, `'posterior'`, `'BMS'`

Steps are performed in that order:

1. MA_model_space: defines a model space
2. MA_cvLME_auto: computes cross-validated log model evidence
3. MS_PPs_group_auto: calculate posterior probabilities from cvLMEs
4. MS_BMS_group_auto: perform cross-validated Bayesian model selection
5. MS_SMM_BMS: generate selected models maps from BMS

- `'all'` : performs 1 to 5
- `'modelSpace'`: : performs step 1
- `'cvLME'`: performs steps 1 and 2
- `'posterior'`: performs steps 1 and 3, assuming step 2 has already been run
- `'BMS'`: performs 1, 4 and 5, assuming step 2 and 3 have already been run

This way you can run all steps at once:

```
bidsModelSelection(opt, 'action', 'all');
```

Or in sequence (can be useful to split running cvLME in several batches of subjects)

bidsModelSelection(opt, 'action', 'cvLME'); bidsModelSelection(opt, 'action', 'posterior'); bidsModelSelection(opt, 'action', 'BMS');

Requirements:

- define the list of BIDS stats models in a cell string of fullpaths

```
opt.toolbox.MACS.model.files
```

- all models must have the same `space` and `task` defined in their inputs
- for a given subject / model, all runs must have the same numbers of regressors This requires to create dummy regressors in case some subjects are missing a condition or a confound. This can be done by using the *bidsFFX(opt)* with the option *opt.glm.useDummyRegressor* set to *true*.

Note: Adding dummy (empty) regressors will make your model non-estimable by SPM, where as the MACS toolbox can deal with this.

- specify each model for each subject:

```
opt = opt_stats_subject_level();

opt.glm.useDummyRegressor = true;

models = opt.toolbox.MACS.model.files
```

(continues on next page)

(continued from previous page)

```
for i = 1:numel(models)
  opt.model.file = models{i};
  bidsFFX('specify', opt);
end
```

For more information see the toolbox manual in the folder `lib/MACS/MACS_Manual`.

Links:

- MACS toolbox repo

If you use this workflow, please cite the following paper:

```
@article{soch2018jnm,
  title={MACS - a new SPM toolbox for model assessment, comparison and selection.},
  author={Soch J, Allefeld C},
  journal={Journal of Neuroscience Methods},
  year={2018},
  volume={306},
  doi={https://doi.org/10.1016/j.jneumeth.2018.05.017}
}
```

If you use cvBMS or cvBMA, please also cite the respective method:

```
@article{soch2016nimg,
  title={How to avoid mismodelling in GLM-based fMRI data analysis:
         cross-validated Bayesian model selection.},
  author={Soch J, Haynes JD, Allefeld C},
  journal={NeuroImage},
  year={2016},
  volume={141},
  doi={https://doi.org/10.1016/j.neuroimage.2016.07.047}
}

@article{soch2017nimg,
  title={How to improve parameter estimates in GLM-based fMRI data analysis:
         cross-validated Bayesian model averaging.},
  author={Soch J, Meyer AP, Haynes JD, Allefeld C},
  journal={NeuroImage},
  year={2017},
  volume={158},
  doi={https://doi.org/10.1016/j.neuroimage.2017.06.056}
}
```

9.1.5 Region of interest analysis

src.workflows.roi.**bidsCreateROI**(*opt*)

Use CPP_ROI and marsbar to create a ROI in MNI space based on a given atlas and inverse normalize those ROIs in native space if requested.

Parameters
opt (structure) – Options chosen for the analysis. See also: checkOptions `checkOptions()` and `loadAndCheckOptions()`.

USAGE:

```
opt = get_option();
opt.roi.atlas = 'wang';
opt.roi.name = {'V1v', 'V1d'};
opt.roi.space = {'IXI549Space', 'individual'};
opt.dir.stats = fullfile(opt.dir.raw, '..', 'derivatives', 'bidspm-stats');

bidsCreateROI(opt);
```

src.workflows.roi.**bidsRoiBasedGLM**(*opt*)

Will run a GLM within a ROI using MarsBar.

USAGE:

```
bidsRoiBasedGLM(opt)
```

Parameters
opt (structure) – Options chosen for the analysis. See also: `checkOptions()` and `loadAndCheckOptions()`.

Returns:

- skipped:

Will compute the absolute maximum percent signal change and the time course of the events or blocks of contrast specified in the BIDS model and save and plot the results in tsv / json / jpeg files.

Warning: If your blocks are modelled as series of fast paced "short" events, the results of this workflow might be misleading. It might be better to make sure that the each block has a single event with a "long" duration.

Adapted from the MarsBar tutorial: lib/CPP_ROI/lib/marsbar-0.44/examples/batch

See also: bidsCreateRoi, plotRoiTimeCourse, getEventSpecificationRoiGlm

9.2 Statistics batches

9.2.1 Subject level

src.batches.stats.**setBatchSubjectLevelGLMSpec**(*varargin*)

Sets up the subject level GLM

USAGE:

```
matlabbatch = setBatchSubjectLevelGLMSpec(matlabbatch, BIDS, opt, subLabel)
```

Parameters

- **matlabbatch** (structure) –
- **BIDS** (structure) – dataset layout. See also: bids.layout, getData.
- **opt** (structure) – Options chosen for the analysis. See also: `checkOptions()` and `loadAndCheckOptions()`.
- **subLabel** (char) –

Returns

- **matlabbatch**
  (structure)

src.batches.stats.**setBatchEstimateModel**(*matlabbatch*, *opt*, *nodeName*, *contrastsList*, *groups*)

Set up the estimate model batch for run/subject or group level GLM

USAGE:

```
matlabbatch = setBatchEstimateModel(matlabbatch, opt)
matlabbatch = setBatchEstimateModel(matlabbatch, opt, nodeName, contrastsList,␣
↪groups)
```

Parameters

- **matlabbatch** (structure) –
- **opt** (structure) – Options chosen for the analysis. See also: checkOptions
- **nodeName** (char) –
- **contrastsList** (cell string) –

Returns

- **matlabbatch**
  (structure)

9.2.2 Group level model

src.batches.stats.**setBatchContrasts**(*matlabbatch*, *opt*, *spmMatFile*, *consess*)

Short description of what the function does goes here.

USAGE:

```
matlabbatch = setBatchContrasts(matlabbatch, opt, spmMatFile, consess)
```

Parameters

- **matlabbatch** (cell) –
- **opt** (structure) – Options chosen for the analysis. See also: checkOptions
- **spmMatFile** (char) –
- **consess** (cell) –

Returns

- **matlabbatch**
  (structure)

src.batches.stats.**setBatchFactorialDesign**(*matlabbatch*, *opt*, *nodeName*)

Handles group level GLM specification

USAGE:

```
[matlabbatch, contrastsList] = setBatchFactorialDesign(matlabbatch, opt, nodeName)
```

Parameters

- **matlabbatch** (structure) –
- **opt** (structure) – Options chosen for the analysis. See also: checkOptions
- **nodeName** (char) –

Returns

- **matlabbatch**
  (structure)

src.batches.stats.**setBatchSubjectLevelContrasts**(*matlabbatch*, *opt*, *subLabel*, *nodeName*)

set batch for run and subject level contrasts

USAGE:

```
matlabbatch = setBatchSubjectLevelContrasts(matlabbatch, opt, subLabel, funcFWHM)
```

Parameters

- **matlabbatch** (structure) –
- **opt** (structure) – Options chosen for the analysis. See also: `checkOptions()` and `loadAndCheckOptions()`.
- **subLabel** (char) –

Returns

- **matlabbatch**

See also: bidsFFX, specifyContrasts, setBatchContrasts

src.batches.stats.**setBatchGroupLevelContrasts**(*matlabbatch*, *opt*, *nodeName*)

USAGE:

```
matlabbatch = setBatchGroupLevelContrasts(matlabbatch, opt, nodeName)
```

Parameters

- **matlabbatch** (structure) –
- **opt** (structure) – Options chosen for the analysis. See also: `checkOptions()` and `loadAndCheckOptions()`.
- **nodeName** (char) –

Returns

- **matlabbatch**

See also: setBatchContrasts, specifyContrasts, setBatchSubjectLevelContrasts

9.2.3 Compute results

src.batches.stats.**setBatchResults**(*matlabbatch*, *result*)

Outputs the typical matlabbatch to compute the result for a given contrast

Common for all type of results: run, session, subject, dataset

USAGE:

```
matlabbatch = setBatchResults(matlabbatch, opt, result)
```

Parameters

- **matlabbatch** (structure) –
- **results** –

Returns

- **matlabbatch**
  (structure)

See also: setBatchSubjectLevelResults, setBatchGroupLevelResults

src.batches.stats.**setBatchSubjectLevelResults**(*varargin*)

USAGE:

```
matlabbatch = setBatchSubjectLevelResults(matlabbatch, opt, subLabel, result)
```

Parameters

- **matlabbatch** (structure) –
- **opt** (structure) – Options chosen for the analysis. See also: `checkOptions()` and `loadAndCheckOptions()`.

- **subLabel** (char) –

Returns

- **matlabbatch**
  (structure)

See also: bidsResults, setBatchResults

src.batches.stats.**setBatchGroupLevelResults**(*varargin*)

USAGE:

```
matlabbatch = setBatchGroupLevelResults(matlabbatch, opt, result)
```

Parameters

- **matlabbatch** (structure) –
- **opt** (structure) – Options chosen for the analysis. See also: checkOptions
- **result** (structure) –

Returns

- **matlabbatch**
  (structure)

9.3 Statistics functions

9.3.1 Subject level

src.subject_level.**createAndReturnOnsetFile**(*opt*, *subLabel*, *tsvFile*)

For a given `_events.tsv` file and `_model.json`, it creates a `_onset.mat` file that can directly be used for the GLM specification of a subject level model.

The file is moved directly into the folder of the GLM.

USAGE:

```
onsetFilename = createAndReturnOnsetFile(opt, subLabel, tsvFile)
```

Parameters

- **opt** (structure) – Options chosen for the analysis. See also: `checkOptions()` and `loadAndCheckOptions()`.
- **subLabel** (char) –
- **tsvFile** (char) – fullpath name of the tsv file.

Returns

onsetFilename
(path) fullpath name of the file created.

See also: convertOnsetTsvToMat

src.subject_level.**getFFXdir**(*subLabel*, *opt*)

Sets the name the FFX directory and creates it if it does not exist

USAGE:

```
ffxDir = getFFXdir(subLabel, opt)
```

Parameters

- **subLabel** (char) –
- **opt** (structure) – Options chosen for the analysis. See also: checkOptions

Returns

- **ffxDir**
  (string)

src.subject_level.**getBoldFilenameForFFX**(*varargin*)

Gets the filename for this bold run for this task for the FFX setup and check that the file with the right prefix exist

USAGE:

```
boldFilename = getBoldFilenameForFFX(BIDS, opt, subLabel, funcFWHM, iSes, iRun)
```

Parameters

- **BIDS** (structure) – dataset layout. See also: bids.layout, getData.
- **opt** (structure) – Options chosen for the analysis. See also: `checkOptions()` and `loadAndCheckOptions()`.
- **subLabel** (char) –
- **iSes** (integer) –
- **iRun** (integer) –

Returns

- **boldFilename**
  (string)

src.subject_level.**deleteResidualImages**(*ffxDir*)

USAGE:

```
deleteResidualImages(ffxDir)
```

Parameters
ffxDir (char) –

src.subject_level.**specifyContrasts**(*SPM*, *model*, *nodeName*)

Specifies the first level contrasts

USAGE:

```
contrasts = specifyContrasts(SPM, model)
```

Parameters

- **SPM** (structure) – content of SPM.mat
- **model** (bids model object) –
- **nodeName** (char) – name of the node to return name of

Returns

- **contrasts**
  (structure)

To know the names of the columns of the design matrix, type : `strvcat(SPM.xX.name)`

See also: setBatchSubjectLevelContrasts, setBatchGroupLevelContrasts

Functions to deal with onsets files and confounds regressors.

`src.subject_level.convertOnsetTsvToMat(opt, tsvFile)`

Converts an events.tsv file to an onset file suitable for SPM subject level analysis.

USAGE:

```
fullpathOnsetFilename = convertOnsetTsvToMat(opt, tsvFile)
```

Parameters

- **opt** (structure) – Options chosen for the analysis. See also: `checkOptions()` and `loadAndCheckOptions()`.
- **tsvFile** (char) –

Use a BIDS stats model specified in a JSON file to:

- loads events.tsv and apply the `Node.Transformations` to its content
- extract the trials (onsets, durations) of the conditions that should be convolved as requested from `Node.Model.HRF.Variables`

It then stores them in in a .mat file that can be fed directly in an SPM GLM batch as 'Multiple conditions'

Parametric modulation can be specified via columns in the TSV file starting with `pmod_`. These columns can be created via the use of `Node.Transformations`. Only polynomial 1 are supported. More complex modulation should be precomputed via the Transformations.

if `opt.glm.useDummyRegressor` is set to `true`, any missing condition will be replaced by a DummyRegressor.

Returns

fullpathOnsetFilename
(string) name of the output `.mat` file.

EXAMPLE:

```
tsvFile = fullfile(pwd, 'data', 'sub-03_task-VisuoTact_run-02_events.tsv');

opt.model.file = fullfile(pwd, 'models', 'model-VisuoTact_smdl.json');
opt.verbosity = 2;
opt.glm.useDummyRegressor = false;

fullpathOnsetFilename = convertOnsetTsvToMat(opt, tsvFile);
```

See also: createAndReturnOnsetFile, bids.transformers

src.subject_level.**convertRealignParamToTsv**(*rpTxtFile*, *opt*, *rmInput*)

Convert SPM typical realignement files to a BIDs compatible TSV one.

USAGE:

```
rpTsvFile = convertRealignParamToTsv(rpTxtFile, opt, rmInput)
```

Parameters

- **rpTxtFile** (path) – path to SPM realignement parameter txt file.
- **opt** (structure) – Options chosen for the analysis. See also: checkOptions `checkOptions()` and `loadAndCheckOptions()`.
- **rmInput** (logical) – Optional. Default to `false`. If true remove original txt file.

src.subject_level.**createAndReturnCounfoundMatFile**(*opt*, *tsvFile*)

Creates a `_regressors.mat` in the subject level GLM folder.

For a given `_regressors.tsv` file and `_model.json`, it creates a `_regressors.mat` file that can directly be used for the GLM specification of a subject level model.

The file is moved directly into the folder of the GLM.

USAGE:

```
counfoundMatFile = createAndReturnCounfoundMatFile(opt, tsvFile)
```

Parameters

- **opt** (structure) – Options chosen for the analysis. See also: `checkOptions()` and `loadAndCheckOptions()`.
- **tsvFile** (char) – fullpath name of the tsv file.

Returns

counfoundMatFile
(string) fullpath name of the file created.

See also: setBatchSubjectLevelGLMSpec, createConfounds

src.subject_level.**getConfoundsRegressorFilename**(*BIDS*, *opt*, *subLabel*, *session*, *run*)

Gets the potential confounds files for a given subject, session, run

USAGE:

```
realignParamFile = getRealignParamFile(BIDS, subLabel, session, run, opt)
```

Parameters

- **BIDS** (structure) – dataset layout. See also: bids.layout, getData.
- **subLabel** (char) – label of the subject ; in BIDS lingo that means that for a file name `sub-02_task-foo_bold.nii` the subLabel will be the string `02`
- **session** (char) – session label (for *ses-001*, the label will be *001*)
- **run** (char) – run index label (for *run-001*, the label will be *001*)

- **opt** (structure) – Options chosen for the analysis. See also: checkOptions() and loadAndCheckOptions().

Returns

- **filename**
  (string)

src.subject_level.**getRealignParamFilename**(*BIDS*, *subLabel*, *session*, *run*, *opt*)

Gets the realignement parameter file produced by SPM (rp_*.txt) for a given subject, session, run

USAGE:

```
realignParamFile = getRealignParamFile(BIDS, subLabel, session, run, opt)
```

Parameters

- **BIDS** (structure) – dataset layout. See also: bids.layout, getData.
- **subLabel** (char) – label of the subject ; in BIDS lingo that means that for a file name sub-02_task-foo_bold.nii the subLabel will be the string 02
- **session** (string) – session label (for *ses-001*, the label will be *001*)
- **run** (string) – run index label (for *run-001*, the label will be *001*)
- **opt** (structure) – Options chosen for the analysis. See also: checkOptions() and loadAndCheckOptions().

Returns

- **realignParamFile**
  (string)

9.3.2 Group level model

src.group_level.**getRFXdir**(*varargin*)

Sets the name the group level analysis directory and creates it if it does not exist

USAGE:

```
rfxDir = getRFXdir(opt, nodeName, contrastName, thisGroup)
```

Parameters

- **opt** (structure) – Options chosen for the analysis. See also: checkOptions
- **nodeName** (char) –
- **contrastName** (char) –
- **thisGroup** (cellstr) –

Returns

rfxDir
(string) Fullpath of the group level directory

Typical output:

- opt.dir.derivatives/bidspm-stats/derivatives/bidspm-groupStats/bidspm-stats

```
['sub-ALL-task-',     model.Input.task, ...
 '_space-'    model.Input.space, ...
 '_FWHM-',    num2str(opt.fwhm.func), ...
 '_conFWHM-', opt.fwhm.contrast, ...
 'node-', model.Input.Nodes(dataset_level).Name, ...             % optional
 'contrast-', model.Input.Nodes(dataset_level).Contrast(i).Name  % if ~= from
↪"dataset_level"
 ]
```

9.3.3 Compute results

src.results.**defaultOuputNameStruct**(*opt*, *result*)

USAGE:

```
outputName = defaultOuputNameStruct(opt, result)
```

Parameters

- **opt** (structure) – Options chosen for the analysis. See also: checkOptions() and loadAndCheckOptions().
- **result** (structure) –

Returns

- **outputName**
  (structure)

See also: setBatchSubjectLevelResults, bidsResults

src.results.**setMontage**(*result*)

USAGE:

```
montage = setMontage(result)
```

CHAPTER

TEN

PREPROCESSING

10.1 Preprocessing workflows

Note: The illustrations in this section mix what the files created by each workflow and the functions and are called by it. In this sense they are not pure DAGs (directed acyclic graphs) as the `*.m` files mentioned in them already exist.

10.1.1 Remove dummies

src.workflows.preproc.**bidsRemoveDummies**(*varargin*)

Removes dummies from functional files

USAGE:

```
bidsRemoveDummies(opt, 'dummyScans', someInteger, 'force', false)
```

Parameters

- **opt** (structure) – Options chosen for the analysis. See also: checkOptions `checkOptions()` and `loadAndCheckOptions()`.
- **dummyScans** (integer >= 0) – number of volumes to remove
- **force** (boolean) – use `'force', true` to remove dummy scans even if metadata say they have already been removed

EXAMPLE:

```
opt.taskName = 'auditory';
opt.dir.input = fullfile(pwd, 'inputs', 'raw');
bidsRemoveDummies(opt, 'dummyScans', 4, 'force', false);
```

10.1.2 Slice Time Correction

src.workflows.preproc.**bidsSTC**(*opt*)

Performs the slice timing correction of the functional data.

USAGE:

```
bidsSTC(opt)
```

Parameters

opt (structure) – Options chosen for the analysis. See also: checkOptions() and loadAndCheckOptions().

STC will be performed using the information provided in the BIDS data set. It will use the mid-volume acquisition time point as as reference.

In general slice order and reference slice is entered in time unit (ms) (this is the BIDS way of doing things) instead of the slice index of the reference slice (the "SPM" way of doing things).

If no slice timing information is available from the file metadata this step will be skipped.

See also: setBatchSTC, getAndCheckSliceOrder

See the documentation for more information about slice timing correction.

More info available on this page of the SPM wikibook.

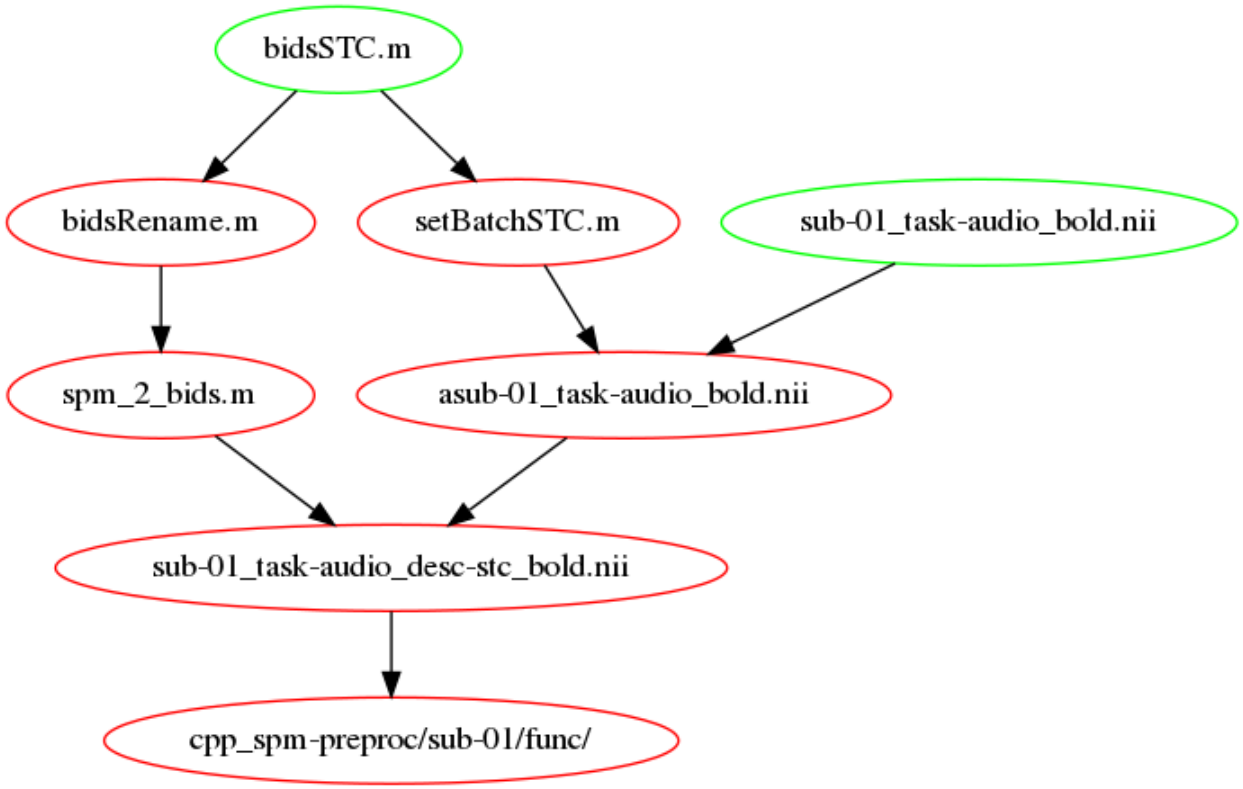


Fig. 1: Slice timing correction workflow

Some comments from here on STC, when it should be applied

At what point in the processing stream should you use it?

This is the great open question about slice timing, and it's not super-answerable. Both SPM and AFNI recommend you do it before doing realignment/motion correction, but it's not entirely clear why. The issue is this:

If you do slice timing correction before realignment, you might look down your non-realigned time course for a given voxel on the border of gray matter and CSF, say, and see one TR where the head moved and the voxel sampled from CSF instead of gray. This would results in an interpolation error for that voxel, as it would attempt to interpolate part of that big giant signal into the previous voxel. On the other hand, if you do realignment before slice timing correction, you might shift a voxel or a set of voxels onto a different slice, and then you'd apply the wrong amount of slice timing correction to them when you corrected - you'd be shifting the signal as if it had come from slice 20, say, when it actually came from slice 19, and shouldn't be shifted as much.

There's no way to avoid all the error (short of doing a four-dimensional realignment process combining spatial and temporal correction - Remi's note: fMRIprep does it), but I believe the current thinking is that doing slice timing first minimizes your possible error. The set of voxels subject to such an interpolation error is small, and the interpolation into another TR will also be small and will only affect a few TRs in the time course. By contrast, if one realigns first, many voxels in a slice could be affected at once, and their whole time courses will be affected. I think that's why it makes sense to do slice timing first. That said, here's some articles from the SPM e-mail list that comment helpfully on this subject both ways, and there are even more if you do a search for "slice timing AND before" in the archives of the list.

10.1.3 Spatial Preprocessing

Perform spatial preprocessing by running `bidsSpatialPrepro`

`src.workflows.preproc.`**`bidsSpatialPrepro`**(*opt*)

Performs spatial preprocessing of the functional and anatomical data.

The anatomical data are segmented, skull-stripped [and normalized to MNI space].

The functional data are re-aligned (unwarped), coregistered with the anatomical, [and normalized to MNI space].

Assumes that `bidsSTC()` has already been run if `opt.stc.skip` is not set to `true`.

USAGE:

```
bidsSpatialPrepro([opt])
```

Parameters

`opt` (`structure`) – Options chosen for the analysis. See also: checkOptions `checkOptions()` and `loadAndCheckOptions()`.

If you want to:

- only do realign and not realign AND unwarp, make sure you set `opt.realign.useUnwarp` to `false`.
- normalize the data to MNI space, make sure `opt.space` includes `IXI549Space`.

See the *Preprocessing* section of the FAQ to know at what resolution files are resampled during normalization.

If you want to:

- use another type of anatomical data than `T1w` as a reference or want to specify which anatomical session is to be used as a reference, you can set this in `opt.bidsFilterFiler.t1w`:

```
opt.bidsFilterFiler.t1w.suffix = 'T1w';
opt.bidsFilterFiler.t1w.ses = 1;
```

The figures below show the `bidsSpatialPrepro` workflow as it would run using realign and unwarp (default) and with normalization to SPM MNI space (`IXI549Space`).

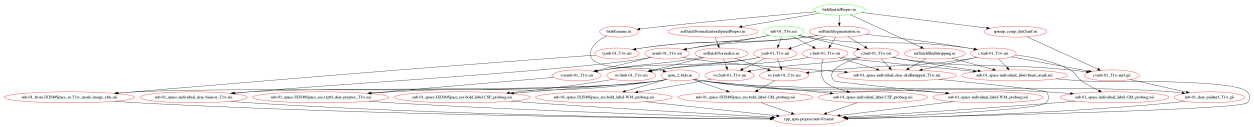


Fig. 2: Anatomical component of the spatial preprocessing workflow

Fig. 3: Functional component of the spatial preprocessing workflow

`src.workflows.preproc.`**`bidsRealignReslice`**`(`*`opt`*`)`

Realigns and reslices the functional data of a given task.

USAGE:

```
bidsRealignReslice(opt)
```

Parameters

`opt` (`structure`) – Options chosen for the analysis. See also: checkOptions `checkOptions()` and `loadAndCheckOptions()`.

Assumes that `bidsSTC()` has already been run if `opt.stc.skip` is not set to `true`.

`src.workflows.preproc.`**`bidsRealignUnwarp`**`(`*`opt`*`)`

Realigns and unwarps the functional data of a given task.

USAGE:

```
bidsRealignReslice(opt)
```

Parameters

`opt` (`structure`) – Options chosen for the analysis. See also: checkOptions `checkOptions()` and `loadAndCheckOptions()`.

Assumes that `bidsSTC` has already been run.

If the `bidsCreateVDM()` workflow has been run before the voxel displacement maps will be used unless `opt.useFieldmaps` is set to `false`.

10.1.4 Smoothing

Perform smoothing of the functional data by running `bidsSmoothing`

`src.workflows.preproc.`**`bidsSmoothing`**`(`*opt*`)`

This performs smoothing to the functional data using a full width half maximum smoothing kernel of size "mm_smoothing".

USAGE:

```
bidsSmoothing(opt)
```

Parameters

`opt` (`structure`) – Options chosen for the analysis. See also: checkOptions `checkOptions()` and `loadAndCheckOptions()`.

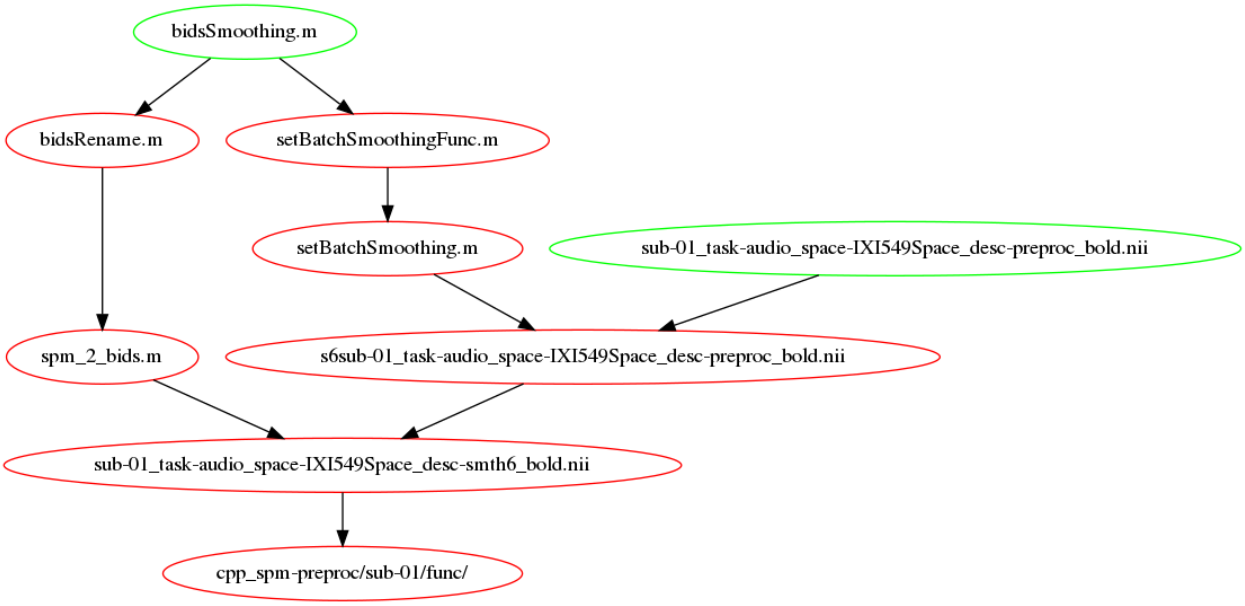


Fig. 4: Smoothing workflow

10.1.5 Others

`src.workflows.preproc.`**`bidsResliceTpmToFunc`**`(`*opt*`)`

Reslices the tissue probability map (TPMs) from the segmentation to the mean functional and creates a mask for the bold mean image

USAGE:

```
bidsResliceTpmToFunc(opt)
```

Parameters

`opt` (`structure`) – Options chosen for the analysis. See also: checkOptions `checkOptions()` and `loadAndCheckOptions()`.

Assumes that the anatomical has already been segmented by `bidsSpatialPrepro()` or `bidsSegmentSkullStrip()`.

It is necessary to run this workflow before running the `functionalQA` pipeline as the computation of the tSNR by `spmup` requires the TPMs to have the same dimension as the functional.

`src.workflows.preproc.`**`bidsSegmentSkullStrip`**`(`*`opt`*`)`

Segments and skullstrips the anatomical image. This workflow is already included in the bidsSpatialPrepro workflow.

USAGE:

```
bidsSegmentSkullStrip(opt)
```

Parameters

`opt` (`structure`) – Options chosen for the analysis. See also: checkOptions `checkOptions()` and `loadAndCheckOptions()`.

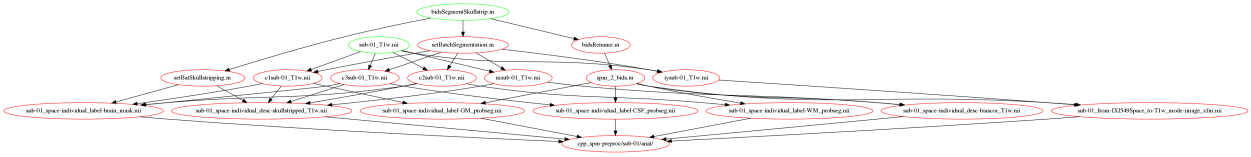


Fig. 5: Segment and skullstrip workflow

`src.workflows.preproc.`**`bidsWholeBrainFuncMask`**`(`*`opt`*`)`

Create segmented-skull stripped mean functional image

10.2 Preprocessing batches

10.2.1 Slice Time Correction

`src.batches.preproc.`**`setBatchSTC`**`(`*`varargin`*`)`

Creates batch for slice timing correction

USAGE:

```
matlabbatch = setBatchSTC(matlabbatch, BIDS, opt, subLabel)
```

Parameters

- **`BIDS`** (`structure`) – dataset layout. See also: bids.layout, getData.
- **`opt`** (`structure`) – Options chosen for the analysis. See also: `checkOptions()` and `loadAndCheckOptions()`.
- **`subLabel`** (`char`) – subject label

Returns

- **matlabbatch**
  (structure) The matlabbatch ready to run the spm job

Slice timing units is in seconds to be BIDS compliant and not in slice number as is more traditionally the case with SPM.

If no slice order can be found, the slice timing correction will not be performed.

If not specified in the options, this function will take the mid-volume time point as reference to do the slice timing correction.

10.2.2 Spatial Preprocessing

src.batches.preproc.**setBatchRealign**(*varargin*)

Set the batch for realign / realign and reslice / realign and unwarp

USAGE:

```
[matlabbatch, voxDim] = setBatchRealign(matlabbatch, ...
                                        BIDS, ...
                                        opt, ...
                                        subLabel, ...
                                        [action = 'realign'])
```

Parameters

- **matlabbatch** (cell) – SPM batch
- **BIDS** (structure) – dataset layout. See also: bids.layout, getData.
- **opt** (structure) – Options chosen for the analysis. See also: `checkOptions()` and `loadAndCheckOptions()`.
- **subLabel** (char) – subject label
- **action** (char) – `realign`, `realignReslice`, `realignUnwarp`, `'reslice'`

Returns

- **matlabbatch**
  (structure) (dimension)
- **voxDim**
  (array) (dimension)

src.batches.preproc.**setBatchReslice**(*matlabbatch*, *opt*, *referenceImg*, *sourceImages*, *interp*)

Set the batch for reslicing source images to the reference image resolution

USAGE:

```
matlabbatch = setBatchReslice(matlabbatch, opt, referenceImg, sourceImages, interp)
```

Parameters

- **matlabbatch** (structure) – list of SPM batches
- **opt** (structure) – Options chosen for the analysis. See also: checkOptions
- **referenceImg** (char or cellstring) – Reference image (only one image)
- **sourceImages** (char or cellstring) – Source images

- **interp** (integer >= 0) – type of interpolation to use (default = 4). Nearest neighbour = 0.

Returns

- **matlabbatch**
  (structure) The matlabbatch ready to run the spm job

src.batches.preproc.**setBatchSegmentation**(*matlabbatch*, *opt*, *imageToSegment*)

Creates a batch to segment the anatomical image

USAGE:

```
matlabbatch = setBatchSegmentation(matlabbatch, opt)
```

Parameters

- **matlabbatch** (structure) – list of SPM batches
- **opt** (structure) – Options chosen for the analysis. See also: checkOptions

Returns

matlabbatch
(structure)

src.batches.preproc.**setBatchSkullStripping**(*matlabbatch*, *BIDS*, *opt*, *subLabel*)

Creates a batch to compute a brain mask based on the tissue probability maps from the segmentation.

USAGE:

```
matlabbatch = setBatchSkullStripping(matlabbatch, BIDS, opt, subLabel)
```

Parameters

- **matlabbatch** (structure) – list of SPM batches
- **BIDS** (structure) – dataset layout. See also: bids.layout, getData.
- **opt** (structure) – Options chosen for the analysis. See also: `checkOptions()` and `loadAndCheckOptions()`.
- **subLabel** (char) – subject label

Returns

- **matlabbatch**
  (structure) The matlabbatch ready to run the spm job

This function will get its inputs from the segmentation batch by reading the dependency from `opt.orderBatches.segment`. If this field is not specified it will try to get the results from the segmentation by relying on the `anat` image returned by `getAnatFilename`.

The threshold for inclusion in the mask can be set by:

```
opt.skullstrip.threshold (default = 0.75)
```

Any voxel with p(grayMatter) + p(whiteMatter) + p(CSF) > threshold will be included in the skull stripping mask.

It is also possible to segment a functional image by setting `opt.skullstrip.mean` to `true`

Skullstripping can be skipped by setting `opt.skullstrip.do` to `false`

src.batches.preproc.**setBatchNormalize**(*matlabbatch*, *deformField*, *voxDim*, *imgToResample*)

Short description of what the function does goes here.

USAGE:

```
matlabbatch = setBatchNormalize(matlabbatch [, deformField] [, voxDim] [,␣
↪imgToResample])
```

Parameters

- **matlabbatch** (structure) –
- **deformField** –
- **voxDim** –
- **imgToResample** –

Returns

- **matlabbatch**
  (structure)

src.batches.preproc.**setBatchNormalizationSpatialPrepro**(*matlabbatch*, *BIDS*, *opt*, *voxDim*)

Short description of what the function does goes here.

USAGE:

```
matlabbatch = setBatchNormalizationSpatialPrepro(matlabbatch, opt, voxDim)
```

Parameters

- **matlabbatch** (structure) –
- **opt** (array) – Options chosen for the analysis. See also: checkOptions
- **voxDim** –

Returns

- **matlabbatch**
  (structure)

src.batches.preproc.**setBatchCoregistrationFuncToAnat**(*matlabbatch*, *BIDS*, *opt*, *subLabel*)

Set the batch for coregistering the functional images to the anatomical image.

USAGE:

```
matlabbatch = setBatchCoregistrationFuncToAnat(matlabbatch, BIDS, subLabel, opt)
```

Parameters

- **matlabbatch** (structure) – list of SPM batches
- **BIDS** (structure) – dataset layout. See also: bids.layout, getData
- **opt** (structure) – Options chosen for the analysis. See also: `checkOptions()` and `loadAndCheckOptions()`.
- **subLabel** (char) – subject label

Returns

- **matlabbatch**
  (structure) The matlabbatch ready to run the spm job

src.batches.preproc.**setBatchCoregistration**(*varargin*)

Set the batch for coregistering the source images into the reference image

USAGE:

```
matlabbatch = setBatchCoregistration(matlabbatch, opt, ref, src, other)
```

Parameters

- **matlabbatch** (structure) – list of SPM batches
- **opt** (structure) – Options chosen for the analysis. See also: checkOptions
- **ref** (char) – Reference image
- **src** (char) – Source image
- **other** (cell string) – Other images to apply the coregistration to

Returns

- **matlabbatch**
  (structure) The matlabbatch ready to run the spm job

src.batches.preproc.**setBatchSaveCoregistrationMatrix**(*matlabbatch*, *BIDS*, *opt*, *subLabel*)

Short description of what the function does goes here.

USAGE:

```
matlabbatch = setBatchSaveCoregistrationMatrix(matlabbatch, BIDS, opt, subLabel)
```

Parameters

- **matlabbatch** (structure) –
- **BIDS** (structure) – dataset layout. See also: bids.layout, getData.
- **opt** (structure) – Options chosen for the analysis. See also: `checkOptions()` and `loadAndCheckOptions()`.
- **subLabel** (char) –

Returns

- **matlabbatch**

10.2.3 Smoothing

src.batches.preproc.**setBatchSmoothConImages**(*matlabbatch*, *opt*)

Creates a batch to smooth all the con images of all subjects

USAGE:

```
matlabbatch = setBatchSmoothConImages(matlabbatch, opt)
```

Parameters

- **matlabbatch** (structure) –
- **opt** (structure) – Options chosen for the analysis. See also: checkOptions

Returns

- **matlabbatch**

See also: bidsRFX, setBatchSmoothing, setBatchSmoothingFunc

src.batches.preproc.**setBatchSmoothingFunc**(*matlabbatch*, *BIDS*, *opt*, *subLabel*)

Creates a batch to smooth the bold files of a subject

USAGE:

```
matlabbatch = setBatchSmoothingFunc(matlabbatch, BIDS, opt, subLabel)
```

Parameters

- **matlabbatch** (structure) –
- **BIDS** (structure) – dataset layout. See also: bids.layout, getData.
- **opt** (structure) – Options chosen for the analysis. See also: `checkOptions()` and `loadAndCheckOptions()`.
- **subLabel** (char) – subject label

Returns

- **matlabbatch**
  (structure)

See also: bidsSmoothing, setBatchSmoothing

src.batches.preproc.**setBatchSmoothing**(*matlabbatch*, *opt*, *images*, *fwhm*, *prefix*)

Small wrapper to create smoothing batch

USAGE:

```
matlabbatch = setBatchSmoothing(matlabbatch, opt, images, fwhm, prefix)
```

Parameters

- **matlabbatch** (structure) –
- **opt** (structure) – Options chosen for the analysis. See also: checkOptions
- **images** (fullpath) –
- **fwhm** (positive integer) –

- **prefix** (char) –

Returns

- **matlabbatch**
  (structure)

See also: bidsSmoothing, bidsRFX, setBatchSmoothingFunc, setBatchSmoothConImages

CHAPTER

ELEVEN

OUTPUTS OF BIDSPM

11.1 jobs and logs

Each batch is saved in `jobs` folder in time a stamped m-file: those are more human readable and interoperable than their equivalent `.mat` file, and they can still be loaded in the SPM batch interface.

It is accompanied by a `.json` file that contains information about the environment in which the batch was run (operating system, bidspm version…).

```
bidspm-stats
├── jobs
│   └── taskName
│       ├── sub-01
│       │   ├── batch_batchName_task-taskName_space-space_FWHM-0_YYYY-MM-HHTMM-SS.json
│       │   └── batch_batchName_task_taskName_space_space_FWHM_0_YYYY-MM-HHTMM-SS.m
│       ├── sub-02
```

11.2 preprocessing

For a complete list of how SPM outputs are renamed into BIDS derivatives see the *Mapping* page.

11.2.1 func

SPM	BIDS
s6usub-01_task-auditory_bold.nii	sub-01_task-auditory_space-individual_desc-smth6_bold.nii
s6rsub-01_task-auditory_bold.nii	sub-01_task-auditory_space-individual_desc-smth6_bold.nii
s6uasub-01_task-auditory_bold.nii	sub-01_task-auditory_space-individual_desc-smth6_bold.nii
s6rasub-01_task-auditory_bold.nii	sub-01_task-auditory_space-individual_desc-smth6_bold.nii
s6wusub-01_task-auditory_bold.nii	sub-01_task-auditory_space-IXI549Space_desc-smth6_bold.nii
s6wrsub-01_task-auditory_bold.nii	sub-01_task-auditory_space-IXI549Space_desc-smth6_bold.nii
s6wuasub-01_task-auditory_bold.nii	sub-01_task-auditory_space-IXI549Space_desc-smth6_bold.nii
s6wrasub-01_task-auditory_bold.nii	sub-01_task-auditory_space-IXI549Space_desc-smth6_bold.nii
s6sub-01_task-auditory_bold.nii	sub-01_task-auditory_desc-smth6_bold.nii
rp_sub-01_task-auditory_bold.txt	sub-01_task-auditory_desc-confounds_regressors.tsv
rp_asub-01_task-auditory_bold.txt	sub-01_task-auditory_desc-confounds_regressors.tsv
usub-01_task-auditory_bold.nii	sub-01_task-auditory_space-individual_desc-preproc_bold.nii
uasub-01_task-auditory_bold.nii	sub-01_task-auditory_space-individual_desc-preproc_bold.nii
rsub-01_task-auditory_bold.nii	sub-01_task-auditory_space-individual_desc-preproc_bold.nii
rasub-01_task-auditory_bold.nii	sub-01_task-auditory_space-individual_desc-preproc_bold.nii
std_usub-01_task-auditory_bold.nii	sub-01_task-auditory_space-individual_desc-std_bold.nii
std_uasub-01_task-auditory_bold.nii	sub-01_task-auditory_space-individual_desc-std_bold.nii

11.2.2 anat

Note: Not listed:

- some of the outputs of the segmentation done by the ALI toolbox for lesion detection

SPM	BIDS
wc1sub-01_T1w.nii	sub-01_space-IXI549Space_res-bold_label-GM_probseg.nii
wc2sub-01_T1w.nii	sub-01_space-IXI549Space_res-bold_label-WM_probseg.nii
wc3sub-01_T1w.nii	sub-01_space-IXI549Space_res-bold_label-CSF_probseg.nii
rc1sub-01_T1w.nii	sub-01_space-individual_res-bold_label-GM_probseg.nii
rc2sub-01_T1w.nii	sub-01_space-individual_res-bold_label-WM_probseg.nii
rc3sub-01_T1w.nii	sub-01_space-individual_res-bold_label-CSF_probseg.nii
wmsub-01_T1w.nii	sub-01_space-IXI549Space_res-r1pt0_T1w.nii
wmsub-01_desc-skullstripped_T1w.nii	sub-01_space-IXI549Space_res-r1pt0_desc-preproc_T1w.nii
msub-01_desc-skullstripped_T1w.nii	sub-01_space-individual_desc-preproc_T1w.nii
wsub-01_desc-skullstripped_T1w.nii	sub-01_space-individual_res-r1pt0_desc-preproc_T1w.nii
msub-01_space-individual_desc-something_label-brain_mask.nii	sub-01_space-individual_label-brain_mask.nii
c1sub-01_T1w.surf.gii	sub-01_desc-pialsurf_T1w.gii

11.3 Statistics

At the subject level each folder contains for each run modeled:

- a pair of `*_onsets.mat` / `*_onsets.tsv`

The `*_onsets.mat` file contains the `names`, `onsets`, `durations`, `pmod` required by SPM to build the "multi condition" section of the model specification. The `*_onsets.tsv` is a human readable equivalent organised like BIDS `events.tsv` files.

- a pair of `*_desc-confounds_regressors.mat` / `*_desc-confounds_regressors.tsv`

The `*_desc-confounds_regressors.mat` file contains the `names`, `R` required by SPM to build the "multi regressor" section of the model specification. The `*_desc-confounds_regressors.tsv` is a human readable equivalent organised like BIDS derivatives `timeseries.tsv` files.

```
bidspm-stats
├── sub-01
│   └── task-taskName_space-space_FWHM-0_node-nodeName
│       |
│       ├── beta_0001.nii  ----------
│       ├── beta_*.nii              |
│       ├── con_0001.nii           |
│       ├── con_*.nii              |
│       ├── mask.nii               | Regular SPM output
│       ├── ResMS.nii              |
│       ├── RPV.nii                |
│       ├── SPM.mat                |
│       ├── spmT_0001.nii          |
│       ├── spmT_*.nii     ----------
│       |
│       ├── sub-blnd01_task-taskName_space-space_desc-contrastName_label-0039_p-0pt050_k-
↪10_MC-FWE_montage.png
│       |
│       ├── sub-blnd01_task-taskName_space-space_desc-afterEstimation_designmatrix.png
│       ├── sub-blnd01_task-taskName_space-space_desc-beforeEstimation_designmatrix.png
│       |
│       ├── sub-blnd01_task-taskName_run-01_desc-confounds_regressors.mat ------
│       ├── sub-blnd01_task-taskName_run-01_desc-confounds_regressors.tsv      |
│       ├── sub-blnd01_task-taskName_run-01_onsets.mat                         |
│       ├── sub-blnd01_task-taskName_run-01_onsets.tsv                         | Files␣
↪used for model specification
│       ├── sub-blnd01_task-taskName_run-*_desc-confounds_regressors.mat       |
│       ├── sub-blnd01_task-taskName_run-*_desc-confounds_regressors.tsv       |
│       ├── sub-blnd01_task-taskName_run-*_onsets.mat                         |
│       └── sub-blnd01_task-taskName_run-*_onsets.tsv --------------------------
...
```

CHAPTER

TWELVE

MAPPING

input	output
m	*space-individual_desc-biascor
c1	*space-individual_label-GM_probseg
c2	*space-individual_label-WM_probseg
c3	*space-individual_label-CSF_probseg
iy_	*from-IXI549Space_to-T1w_mode-image_xfm
y_	*from-T1w_to-IXI549Space_mode-image_xfm
segparam_	*segparam
a	*space-individual_desc-stc
au	*space-individual_desc-stc
unwarpparam_	*unwarpparam
u	*space-individual_desc-preproc
ua	*space-individual_desc-preproc
rp_	*motion.tsv
rp_a	*motion.tsv
rp_au	*motion.tsv
wc1	*space-IXI549Space_label-GM_res-bold_probseg
wc2	*space-IXI549Space_label-WM_res-bold_probseg
wc3	*space-IXI549Space_label-CSF_res-bold_probseg
s	*space-individual_desc-smth
sua	*space-individual_desc-smth
sau	*space-individual_desc-smth
sra	*space-individual_desc-smth
su	*space-individual_desc-smth
sr	*space-individual_desc-smth
sw	*space-IXI549Space_desc-smth
swua	*space-IXI549Space_desc-smth
swau	*space-IXI549Space_desc-smth
swra	*space-IXI549Space_desc-smth
swu	*space-IXI549Space_desc-smth
swr	*space-IXI549Space_desc-smth
w	*space-IXI549Space_desc-preproc
wm	*space-IXI549Space_res-r1pt0
wau	*space-IXI549Space_desc-preproc
wua	*space-IXI549Space_desc-preproc
wra	*space-IXI549Space_desc-preproc
wu	*space-IXI549Space_desc-preproc
wr	*space-IXI549Space_desc-preproc

continues on next page

Table 1 – continued from previous page

input	output
ra	*space-individual_desc-preproc
mean	*space-individual_desc-mean
meanu	*space-individual_desc-mean
meanua	*space-individual_desc-mean
meanau	*space-individual_desc-mean
wmean	*space-IXI549Space_desc-mean
wmeanu	*space-IXI549Space_desc-mean
wmeanua	*space-IXI549Space_desc-mean
wmeanau	*space-IXI549Space_desc-mean
rc1	*space-individual_label-GM_res-bold_probseg
rc2	*space-individual_label-WM_res-bold_probseg
rc3	*space-individual_label-CSF_res-bold_probseg
s6w	*space-IXI549Space_desc-smth6
s6wua	*space-IXI549Space_desc-smth6
s6wra	*space-IXI549Space_desc-smth6
s6wu	*space-IXI549Space_desc-smth6
s6wr	*space-IXI549Space_desc-smth6
s6ua	*space-individual_desc-smth6
s6ra	*space-individual_desc-smth6
s6u	*space-individual_desc-smth6
s6r	*space-individual_desc-smth6
s6	*desc-smth6
mean_ua	*space-individual_desc-mean
mean_r	*space-individual_desc-mean
mean_u	*space-individual_desc-mean
mean_	*space-individual_desc-mean
std_ua	*space-individual_desc-std
std_u	*space-individual_desc-std
std_r	*space-individual_desc-std
std_	*space-individual_desc-std
mdesc-skullstripped_T1w.nii	*space-individual_desc-preproc
mlabel-brain_mask.nii	*label-brain
c1add-joker_T1w.surf.gii	*desc-pialsurf.gii
wdesc-skullstripped_T1w.nii	*space-IXI549Space_desc-preproc_res-r1pt0
wdesc-brain_mask.nii	*space-IXI549Space_desc-brain_res-r1pt0
wmdesc-skullstripped_T1w.nii	*space-IXI549Space_desc-preproc_res-r1pt0

CHAPTER

THIRTEEN

METHODS SECTION

13.1 Dataset description

Use the `reportBIDS` function to description of your dataset that can be used for your methods section

`src.reports.`**`reportBIDS`**`(`*`opt`*`)`

Prints out a human readable description of a BIDS data set for every subject in `opt.subjects`.

The output is a markdown file save in the directory:

```
opt.dir.output, 'reports', ['sub-' subLabel]
```

USAGE:

```
opt.dir.input = "path_to_dataset"
reportBIDS(opt)
```

Parameters

`opt` (`structure`) – Options chosen for the analysis. See also: checkOptions

13.2 Preprocessing & GLM

This can be generated with the `boilerplate` function.

`src.reports.`**`boilerplate`**`(`*`varargin`*`)`

USAGE:

```
outputFile = boilerplate(opt, ...
                         'outputPath', outputPath, ...
                         'pipelineType', pipelineType, ...
                         'partialsPath', partialsPath, ...
                         'verbosity', 2)
```

Parameters

- **`opt`** (`structure`) – Options chosen for the analysis. See also: checkOptions
- **`outputPath`** (`char`) –
- **`pipelineType`** (`char`) – `'preproc'` or `'stats`
- **`partialsPath`** (`path`) –

- **verbose** (boolean) –

EXAMPLE:

```
opt.model.file = path_to_model;
opt.designType = 'event';
opt = checkOptions(opt);

outputFile = boilerplate(opt, ...
                         'outputPath', pwd, ...
                         'pipelineType', 'stats', ...
                         'verbosity', 2)
```

13.2.1 Output example - Preprocessing

Pre processing

The (f)MRI data were pre-processed with BIDSpm (v2.2.0; https://github.com/cpp-lln-lab/bidspm; DOI: https://doi.org/10.5281/zenodo.3554331) using statistical parametric mapping (SPM12 - 7771; Wellcome Center for Neuroimaging, London, UK; https://www.fil.ion.ucl.ac.uk/spm; RRID:SCR_007037) using MATLAB 9.4.0.813654 (R2018a) on a unix computer (Linux version 5.15.0-52-generic (buildd@lcy02-amd64-032) (gcc (Ubuntu 11.2.0-19ubuntu1) 11.2.0, GNU ld (GNU Binutils for Ubuntu) 2.38) #58-Ubuntu SMP Thu Oct 13 08:03:55 UTC 2022). .

The preprocessing of the functional images was performed in the following order:

- removing of dummy scans
- slice timing correction
- realignment and unwarping
- segmentation and skullstripping
- normalization MNI space
- smoothing

{{nb}} dummy scans were removed to allow for signal stabilization.

Slice timing correction was performed taking the ^th slice as a reference (interpolation: sinc interpolation).

Functional scans from each participant were realigned and unwarped using the mean image as a reference (SPM single pass; number of degrees of freedom: 6 ; cost function: least square) (Friston et al, 1995).

The anatomical image was bias field corrected. The bias field corrected image was segmented and normalized to MNI space (target space: IXI549Space; target resolution: 1 mm; interpolation: 4th degree b-spline) using a unified segmentation.

The tissue propability maps generated by the segmentation were used to skullstripp the bias corrected image removing any voxel with p(gray matter) + p(white matter) + p(CSF) > 0.75.

The mean functional image obtained from realignement was co-registered to the bias corrected anatomical image (number of degrees of freedom: 6 ; cost function: normalized mutual information) (Friston et al, 1995). The transformation matrix from this coregistration was applied to all the functional images.

The deformation field obtained from the segmentation was applied to all the functional images (target space: IXI549Space; target resolution: equal to that used at acquisition; interpolation: 4th degree b-spline).

Preprocessed functional images were spatially smoothed using a 3D gaussian kernel (FWHM = 6 mm).

References

This method section was automatically generated using BIDSpm (v2.2.0; https://github.com/cpp-lln-lab/bidspm; DOI: https://doi.org/10.5281/zenodo.3554331) and octache (https://github.com/Remi-Gau/Octache).

13.2.2 Output example - GLM subject level

fMRI statistical analysis

The fMRI data were analysed with BIDSpm (v2.2.0; https://github.com/cpp-lln-lab/bidspm; DOI: https://doi.org/10.5281/zenodo.3554331) using statistical parametric mapping (SPM12 - 7771; Wellcome Center for Neuroimaging, London, UK; https://www.fil.ion.ucl.ac.uk/spm; RRID:SCR_007037) using MATLAB 9.4.0.813654 (R2018a) on a unix computer (Linux version 5.15.0-52-generic (buildd@lcy02-amd64-032) (gcc (Ubuntu 11.2.0-19ubuntu1) 11.2.0, GNU ld (GNU Binutils for Ubuntu) 2.38) #58-Ubuntu SMP Thu Oct 13 08:03:55 UTC 2022).

The input data were the preprocessed BOLD images in IXI549Space space for the task " facerepetition ".

Run / subject level analysis

At the subject level, we performed a mass univariate analysis with a linear regression at each voxel of the brain, using generalized least squares with a global AR(1) model to account for temporal auto-correlation and a drift fit with discrete cosine transform basis (128 seconds cut-off).

Image intensity scaling was done run-wide before statistical modeling such that the mean image would have a mean intracerebral intensity of 100.

We modeled the fMRI experiment in a event design with regressors entered into the run-specific design matrix. The onsets were convolved with SPM canonical hemodynamic response function (HRF) and its temporal and dispersion derivatives for the conditions:

- `famous_1`,
- `famous_2`,
- `unfamiliar_1`,
- `unfamiliar_2`, .

Nuisance covariates included:

- `trans_?`,
- `rot_?`,

to account for residual motion artefacts, .

References

This method section was automatically generated using BIDSpm (v2.2.0; https://github.com/cpp-lln-lab/bidspm; DOI: https://doi.org/10.5281/zenodo.3554331) and octache (https://github.com/Remi-Gau/Octache).

13.2.3 Output example - GLM Group level

```
`{warning} WORK IN PROGRESS `
```

CHAPTER

FOURTEEN

DO IT YOURSELF

Listing here some basic commands you might need to know to combine masks by hand.

Also good way to learn about some basic low level functions of SPM.

- `spm_vol`: reads the header of a 3D or 4D Nifti images
- `spm_read_vols`: given a header it will get the data of Nifti image
- `spm_write_vol`: given a header it will get the data of Nifti image

For more info about basic files, check the SPM wikibooks.

14.1 Merging 2 masks

```
path_to_mask_1 = 'FIX_ME';
path_to_mask_2 = 'FIX_ME';

% get header of Nifti images
header_1 = spm_vol(path_to_mask_1);
header_2 = spm_vol(path_to_mask_2);

% if you want to make sure that images are in the same space
% and have same resolution
masks = char({path_to_mask_1; path_to_mask_2});
spm_check_orientations(spm_vol(masks));

% get data of Nifti images
mask_1 = spm_read_vols(header_1);
mask_2 = spm_read_vols(header_2);

% concatenate data along the 4th dimension
merged_mask = cat(4, mask_1, mask_2);

% keep any voxel that has some value along the 4th dimension
merged_mask = any(merged_mask, 4);

% create a new header of the final mask
merged_mask_header = header_1;
merged_mask_header.fname = 'new_mask.nii';

spm_write_vol(merged_mask_header, merged_mask);
```

CHAPTER

FIFTEEN

FREQUENTLY ASKED QUESTIONS

15.1 General

15.1.1 How can I prevent from having SPM windows pop up all the freaking time?

Running large number of batches when the GUI of MATLAB is active can be annoying, as SPM windows will always pop up and become active instead of running in the background like most users would prefer to.

One easy solution is to add a `spm_my_defaults` function with the following content in the MATLAB path, or in the directory where you are running your scripts or command from.

```
function spm_my_defaults

  global defaults

  defaults.cmdline = true;

end
```

This should be picked up by bidspm and SPM upn initialization and ensure that SPM runs in command line mode.

15.1.2 How can I run any of this from the command line?

Related to the previous question but more radical. You can run most analysis from within your terminal without starting the MATLAB graphic interface.

For this you first need to know where is the MATLAB application. Here are the typical location depending on your operating system (where `XXx` corresponds to the version you use).

- Windows: `C:\Program Files\MATLAB\R20XXx\bin\matlab.exe`
- Mac: `/Applications/Matlab_R20XXx.app/bin/matlab`
- Linux: `/usr/local/MATLAB/R20XXx/bin/matlab`

You can then launch from a terminal in a command line only with the following arguments: `-nodisplay -nosplash -nodesktop`

So on Linux for example:

```
/usr/local/MATLAB/R2017a/bin/matlab -nodisplay -nosplash -nodesktop
```

If you are on MAc or Linux, we would recommend adding those aliases to your `.bashrc` or wherever else you keep your aliases.

```
matlab=/usr/local/MATLAB/R20XXx/bin/matlab
matlabcli='/usr/local/MATLAB/R20XXx/bin/matlab -nodisplay -nosplash -nodesktop'
```

15.1.3 What happens if we run same code twice? Are there timestamps on the files or are we overwriting them?

In the vast majority of cases, if you have not touched anything to your options, you will overwrite the output.

Two exceptions that actually have time stamps and are not over-written:

- The `matlabbatches` saved in the `jobs` folders as `.mat` and `.json` files.
- If you have saved your options with `saveOptions`, then the output `.json` file is saved with a time stamp too. Most of the default `getOptions` templates include `saveOptions` as a last function call.

In most of other cases if you don't want to overwrite previous output, you will have to change the output directory.

For the preprocessing workflows, in general you would have to specify a different `opt.dir.output`.

For the statistics workflows, you have a few more options as the name of the output folders includes information that comes from the options and / or the BIDS stats model.

The output folder name (generated by `getFFXdir()` for the subject level and by `getRFXdir()` for the dataset level) should include the FWHM used on the BOLD images as well as info specified in the `Inputs` section of the BIDS stats model JSON file (like the name of the task or the MNI space of the input images).

```
$ ls demos/MoAE/outputs/derivatives/bidspm-stats/sub-01/stats

# Folder name for a model on the auditory task in SPM's MNI space
# on data smoothed with a 6mm kernel
task-auditory_space-IXI549Space_FWHM-6
```

But also the GLM folder will include the `Name` of the BIDS stats model as description (`desc`) if this `Name` is not just the name `opt.taskName`.

For example, here with the following BIDS stats model.

```
$ head tests/dummyData/models/model-nback_smdl.json

{
  "Name": "nback MVPA",     # <---- this gets appended to the folder name as␣
↪description
  "BIDSModelVersion": "1.0.0",
  "Description": "for folder naming",
  "Input": {
    "task": "nback"
  }
  ...
```

And this code to set things up

```
subLabel = '02';
opt.taskName = 'nback';
opt.space = 'individual';

opt.dir.stats = 'outputs/derivatives/bidspm-stats';
```

(continues on next page)

(continued from previous page)

```
ffxDir = getFFXdir(subLabel, opt);
```

Here is the expected output folder name

```
expectedOutput = fullfile(opt.dir.stats, 'sub-02', 'stats', ...
                          'task-nback_space-individual_FWHM-6_desc-nbackMVPA');
```

15.1.4 How can I know that things are set up properly before I run an analysis?

If you want to set things up but not let SPM actually run the batches you can use the option:

`opt.dryRun = true()`

This can be useful when debugging. You may still run into errors when SPM jobman takes over and starts running the batches, but you can at least see if the batches will be constructed without error and then inspect with the SPM GUI to make sure everything is fine.

15.1.5 What is the BIDS way to name and store (Regions of Interest) ROIs?

There is no "official" way to name ROI in BIDS, but you can apply BIDS naming principles to name those.

The closest things to ROI naming are the `masks` for the BIDS derivatives.

Here is an example from the :ref:`face repetition demo`::

```
bidspm-roi
├── group
│   ├── hemi-L_space-MNI_label-V1d_desc-wang_mask.json
│   ├── hemi-L_space-MNI_label-V1d_desc-wang_mask.nii
│   ├── hemi-L_space-MNI_label-V1v_desc-wang_mask.json
│   ├── hemi-L_space-MNI_label-V1v_desc-wang_mask.nii
│   ├── hemi-R_space-MNI_label-V1d_desc-wang_mask.json
│   ├── hemi-R_space-MNI_label-V1d_desc-wang_mask.nii
│   ├── hemi-R_space-MNI_label-V1v_desc-wang_mask.json
│   └── hemi-R_space-MNI_label-V1v_desc-wang_mask.nii
└── sub-01
    └── roi
        ├── sub-01_hemi-L_space-individual_label-V1d_desc-wang_mask.nii
        ├── sub-01_hemi-L_space-individual_label-V1v_desc-wang_mask.nii
        ├── sub-01_hemi-R_space-individual_label-V1d_desc-wang_mask.nii
        └── sub-01_hemi-R_space-individual_label-V1v_desc-wang_mask.nii
```

ROIs that are defined in some MNI space are going to be the same across subjects, so you could store a "group" folder (this is not BIDSy but is less redundant than having a copy of the same file for each subject).

The `desc` entity (description) here is used to denotate the atlas the ROI taken from, so if you are building yours from a localizer you might not need to use it.

Ideally you would want to add a JSON file to add metadata about those ROIs.

You can use bids-matlab to help you create BIDS valid filenames.

```
>> name_spec.ext = '.nii';
>> name_spec.suffix = 'mask';
>> name_spec.entities = struct( ...
                               'hemi', 'R', ...
                               'space', 'MNI', ...
                               'label', 'V1v', ...
                               'desc', 'wang');
>> file = bids.File(name_spec);
>> file.filename

   hemi-R_space-MNI_label-V1v_desc-wang_mask.nii
```

15.1.6 How can run my script only only certain files, like just the session 02 for example?

Currently there are 2 ways of doing this.

- using a `bids_filter_file.json` file or its counterpart field `opt.bidsFilterFile`
- using the `opt.query` option field. On the long run the plan is to use only the `bids_filter_file.json`, but for now both possibilities should work.

bids filter file

This is similar to the way you can "select" only certain files to preprocess with fmriprep.

You can use a `opt.bidsFilterFile` field in your options to define a typical images "bold", "T1w" in terms of their BIDS entities. The default value is:

```
struct('fmap', struct('modality', 'fmap'), ...
       'bold', struct('modality', 'func', 'suffix', 'bold'), ...
       't2w',  struct('modality', 'anat', 'suffix', 'T2w'), ...
       't1w',  struct('modality', 'anat', 'space', '', 'suffix', 'T1w'), ...
       'roi',  struct('modality', 'roi', 'suffix', 'mask'));
```

Similarly when using the BIDS app bidspm you can use the argument `bids_filter_file` to point to a JSON file that would also define typical images "bold", "T1w"…

The default content in this case would be:

```
{
    "fmap": { "datatype": "fmap" },
    "bold": { "datatype": "func", "suffix": "bold" },
    "t2w": { "datatype": "anat", "suffix": "T2w" },
    "t1w": { "datatype": "anat", "space": "", "suffix": "T1w" },
    "roi": { "datatype": "roi", "suffix": "mask" }
}
```

So if you wanted to run your analysis on say run 02 and 05 of session 02, you would define this file like this:

```
{
    "fmap": { "datatype": "fmap" },
    "bold": {
```

(continues on next page)

(continued from previous page)

```
        "datatype": "func",
        "suffix": "bold",
        "ses": "02",
        "run": ["02", "05"]
    },
    "t2w": { "datatype": "anat", "suffix": "T2w" },
    "t1w": { "datatype": "anat", "space": "", "suffix": "T1w" },
    "roi": { "datatype": "roi", "suffix": "mask" }
}
```

opt.query

You can select a subset of your data by using the `opt.query`.

This will create a "filter" that bids-matlab will use to only "query" and retrieve the subset of files that match the requirement of that filter

In "pure" bids-matlab it would look like:

```
BIDS = bids.layout(path_to_my_dataset)
bids.query(BIDS, 'data', opt.query)
```

So if you wanted to run your analysis on say run `02` and `05` of session `02`, you would define your filter like this:

```
opt.query.ses = '02'
opt.query.run = {'02', '05'}
```

15.2 Preprocessing

15.2.1 What images are resampled during preprocessing and to what resolution?

In the spatial preprocessing workflow (`bidsSpatialPrepro`):

1. When no normalization is requested

This is the case when `opt.space = 'individual'`, functional images resolution is not changed. This cannot be overridden.

2. During normalization to MNI

By default, functional images resolution is not changed. Override possible by setting `opt.funcVoxelDims` to the desired resolution.

The anatomical images are resampled at 1 mm.

Tissue probability maps downsampled at resolution of functional images mostly to help with potential with creation of tissue-based mask and also quality control pipelines.

For several files, you can guess their resolution if they have `res` entity in their filename:

- `res-bold` means that the image is resampled at the resolution of the BOLD timeseries
- `res-r1pt0` means that the image is resampled at a resolution of 1.00 mm isometric

```
sub-01
├── anat
│   ├── sub-01_space-individual_desc-biascor_T1w.nii                    # native res
│   ├── sub-01_space-individual_desc-skullstripped_T1w.nii              # native res
│   ├── sub-01_space-individual_label-brain_mask.nii                    # native res
│   ├── sub-01_space-individual_label-CSF_probseg.nii
│   ├── sub-01_space-individual_label-GM_probseg.nii
│   ├── sub-01_space-individual_label-WM_probseg.nii
│   ├── sub-01_space-individual_res-r1pt0_desc-preproc_T1w.nii          # 1.0 mm
│   ├── sub-01_space-IXI549Space_res-bold_label-CSF_probseg.nii         # bold res
│   ├── sub-01_space-IXI549Space_res-bold_label-GM_probseg.nii          # bold res
│   ├── sub-01_space-IXI549Space_res-bold_label-WM_probseg.nii          # bold res
│   └── sub-01_space-IXI549Space_res-r1pt0_T1w.nii                      # 1.0 mm
└── func
    ├── sub-01_task-auditory_space-individual_desc-mean_bold.nii        # native res
    ├── sub-01_task-auditory_space-individual_desc-preproc_bold.nii     # native res
    ├── sub-01_task-auditory_space-individual_desc-std_bold.nii         # native res
    ├── sub-01_task-auditory_space-IXI549Space_desc-mean_bold.nii       # native res
    └── sub-01_task-auditory_space-IXI549Space_desc-preproc_bold.nii    # native res
```

See those slides for some pointers on how to make choices for the resolution to choose for your analysis.

15.3 Statistics

15.3.1 How should I structure my data to run my statistical analysis?

The main thing to remember is that bidspm will read the events.tsv files from your raw BIDS data set and will read the bold images from a `bidspm-preproc` folder.

If your data was preprocessed with fmriprep, bidspm will first need to copy, unzip and smooth the data into a `bidspm-preproc` folder

Here is an example of how the data is organized for the MoAE fmriprep demo and what the `bidspm` BIDS call would look like.

```
├── inputs
│   ├── fmriprep                    # fmriprep preprocessed BIDS dataset
│   |   ├── dataset_description.json
│   |   └── sub-01
│   |       ├── anat
│   |       ├── figures
│   |       └── func
│   |           ├── sub-01_task-auditory_desc-confounds_timeseries.json
│   |           ├── sub-01_task-auditory_desc-confounds_timeseries.tsv
│   |           ├── sub-01_task-auditory_space-MNI152NLin6Asym_desc-brain_mask.json
│   |           ├── sub-01_task-auditory_space-MNI152NLin6Asym_desc-brain_mask.nii.gz
│   |           ├── sub-01_task-auditory_space-MNI152NLin6Asym_desc-preproc_bold.json
│   |           └── sub-01_task-auditory_space-MNI152NLin6Asym_desc-preproc_bold.nii.gz
│   └── raw                         # raw BIDS dataset
│       ├── dataset_description.json
│       ├── README
│       ├── sub-01
```

(continues on next page)

(continued from previous page)

```
        │   ├── anat
        │   │   └── sub-01_T1w.nii
        │   └── func
        │       ├── sub-01_task-auditory_bold.nii
        │       └── sub-01_task-auditory_events.tsv
        └── task-auditory_bold.json
├── models                              # models used to run the GLM
│   ├── model-MoAEfmriprep_smdl.json
│   ├── model-MoAEindividual_smdl.json
│   └── model-MoAE_smdl.json
├── options
└── outputs
    └── derivatives
        └── bidspm-preproc          # contains either data taken from fmriprep and␣
↪smoothed
            ├── dataset_description.json
            ├── README
            ├── jobs
            │   └── auditory
            │       └── sub-01
            └── sub-01
                ├── anat
                └── func
                    ├── sub-01_task-auditory_desc-confounds_timeseries.json
                    ├── sub-01_task-auditory_desc-confounds_timeseries.tsv
                    ├── sub-01_task-auditory_space-MNI152NLin6Asym_desc-brain_mask.json
                    ├── sub-01_task-auditory_space-MNI152NLin6Asym_desc-brain_mask.nii
                    ├── sub-01_task-auditory_space-MNI152NLin6Asym_desc-preproc_bold.json
                    ├── sub-01_task-auditory_space-MNI152NLin6Asym_desc-preproc_bold.nii
                    ├── sub-01_task-auditory_space-MNI152NLin6Asym_desc-smth8_bold.json
                    └── sub-01_task-auditory_space-MNI152NLin6Asym_desc-smth8_bold.nii
```

```
WD = fileparts(mfilename('fullpath'));

subject_label = '01';

bids_dir = fullfile(WD, 'inputs', 'raw');
output_dir = fullfile(WD, 'outputs', 'derivatives');
preproc_dir = fullfile(output_dir, 'bidspm-preproc');

model_file = fullfile(pwd, 'models', 'model-MoAEfmriprep_smdl.json');

bidspm(bids_dir, output_dir, 'subject', ...
        'participant_label', {subject_label}, ...
        'action', 'stats', ...
        'preproc_dir', preproc_dir, ...
        'model_file', model_file, ...
        'fwhm', 8, ...
        'options', opt);
```

15.4 Results

15.4.1 How can I change which slices are shown in a montage?

> In the `bidsResults.m` I get an image with the overlay of different slices. | How can I change which slices are shown?

When you define your options the range of slices that are to be shown can be changed like this (see `bidsResults` help section for more information):

```
% slices position in mm [a scalar or a vector]
opt.results(1).montage.slices = -12:4:60;

% slices orientation: can be 'axial' 'sagittal' or 'coronal'
% axial is default
opt.results(1).montage.orientation = 'axial';
```

CHAPTER

SIXTEEN

FIELDMAPS

In a nutshell, the information we need to create a VDM in SPM (see `calculate VDM` module in SPM batch):

- `blip direction`
- `echo time`
- `total EPI readout time`

Inferring `blip direction` and `echo time` from a dataset that has sufficient metadata is usually simple.

But `total EPI readout time` is not mentioned, so it has to be computed from the information we have, it is not entirely clear how (see the comments with a lot of `???` in `getTotalReadoutTime`).

Things that are yet unclear:

- is it actually possible to compute total EPI readout time that SPM needs from the info in a typical dataset with fieldmaps like `openneuro/ds001168`?
- If it is not then that is an issue because it means some BIDS dataset are not usable with SPM.

Things to keep an eye on: the code from this repo from the fMRIprep team could have answers for us.

`src.workflows.preproc.`**`bidsCreateVDM`**`(`*`opt`*`)`

Creates the voxel displacement maps from the fieldmaps of a BIDS dataset.

USAGE:

```
bidsCreateVDM(opt)
```

Parameters

`opt` (`structure`) – Options chosen for the analysis. See also: checkOptions `checkOptions()` and `loadAndCheckOptions()`.

Inspired from spmup `spmup_BIDS_preprocess` (@ commit 198c980d6d7520b1a99) (URL missing)

`src.batches.preproc.`**`setBatchCoregistrationFmap`**`(`*`matlabbatch`*`, `*`BIDS`*`, `*`opt`*`, `*`subLabel`*`)`

Set the batch for the coregistration of field maps

USAGE:

```
matlabbatch = setBatchCoregistrationFmap(matlabbatch, BIDS, opt, subLabel)
```

Parameters

- **`BIDS`** (`structure`) – dataset layout. See also: bids.layout, getData.

- **opt** (structure) – Options chosen for the analysis. See also: checkOptions() and loadAndCheckOptions().
- **subLabel** (char) –

Returns

- **matlabbatch**
  (structure) The matlabbatch ready to run the spm job

TODO implement for 'phase12', 'fieldmap', 'epi'

src.batches.preproc.**setBatchCreateVDMs**(*matlabbatch*, *BIDS*, *opt*, *subLabel*)

Short description of what the function does goes here.

USAGE:

```
matlabbatch = setBatchCreateVDMs(matlabbatch, BIDS, opt, subLabel)
```

Parameters

- **matlabbatch** (structure) –
- **BIDS** (structure) – dataset layout. See also: bids.layout, getData.
- **opt** (structure) – Options chosen for the analysis. See also: checkOptions() and loadAndCheckOptions().
- **subLabel** (char) – subject label

Returns

- **matlabbatch**
  (structure) The matlabbatch ready to run the spm job

TODO implement for 'phase12', 'fieldmap', 'epi'

src.batches.preproc.**setBatchComputeVDM**(*matlabbatch*, *fmapType*, *refImage*)

Short description of what the function does goes here.

USAGE:

```
matlabbatch = setBatchComputeVDM(matlabbatch, fmapType, refImage)
```

Parameters

- **matlabbatch** (structure) – list of SPM batches
- **fmapType** (char) – ''phasediff'' or 'phase&mag'
- **refImage** – Reference image

Returns

- **matlabbatch**
  (structure) The matlabbatch ready to run the spm job

matlabbatch = setBatchComputeVDM(type)

adapted from spmup get_FM_workflow.m (@ commit 198c980d6d7520b1a996f0e56269e2ceab72cc83)

src.fieldmaps.**getBlipDirection**(*metadata*)

Gets the total read out time of a sequence.

USAGE:

```
blipDir = getBlipDirection(metadata)
```

Parameters
metadata (strcuture) – image metadata

Returns

- **blipDir**

Used to create the voxel dsiplacement map (VDM) from the fieldmap

src.fieldmaps.**getMetadataFromIntendedForFunc**(*BIDS*, *fmapMetadata*)

Gets metadata of the associated bold file: - finds the bold file a fmap is intended for, - parse its filename, - get its metadata.

USAGE:

```
[totalReadoutTime, blipDir] = getMetadataFromIntendedForFunc(BIDS, fmapMetadata)
```

Parameters

- **BIDS** (structure) – dataset layout. See also: bids.layout, getData.
- **fmapMetadata** (structure) –

Returns

totalReadoutTime
(type) (dimension)

blipDir
(type) (dimension)

At the moment the VDM is created based on the characteristics of the last func file in the IntendedFor field

src.fieldmaps.**getTotalReadoutTime**(*metadata*)

Gets the total read out time of a sequence. Used to create the voxel dsiplacement map (VDM) from the fieldmap

USAGE:

```
totalReadoutTime = getTotalReadoutTime(metadata)
```

Parameters
metadata (strcuture) – image metadata

Returns

- **totalReadoutTime**
(float) in millisecond

Currently this relies on the user adding extra metadata in the json of the functional files as the metadata queried are not "official" BIDS metadata but can usuall be found in the DICOM headers (for example: PixelBandwidth)

src.fieldmaps.**getVdmFile**(*BIDS*, *opt*, *boldFilename*)

returns the voxel displacement map associated with a given bold file

USAGE:

```
vdmFile = getVdmFile(BIDS, opt, boldFilename)
```

Parameters

- **BIDS** (structure) – dataset layout. See also: bids.layout, getData.
- **opt** (structure) – Options chosen for the analysis. See also: checkOptions() and loadAndCheckOptions().
- **boldFilename** (path) –

Returns

- **vdmFile**
  (string)

CHAPTER

SEVENTEEN

QUALITY CONTROL

Note: The illustrations in this section mix what the files created by each workflow and the functions and are called by it. In this sense they are not pure DAGs (directed acyclic graphs) as the `*.m` files mentioned in them already exist.

`src.QA.`**`anatomicalQA`**`(`*`opt`*`)`

Computes several metrics for anatomical image.

Is run as part of:

- `bidsSpatialPrepro`

USAGE:

```
anatomicalQA(opt)
```

Parameters
`opt` (`structure`) – Options chosen for the analysis. See also: checkOptions

`src.QA.`**`functionalQA`**`(`*`opt`*`)`

Is run as part of:

- `bidsSpatialPrepro`

USAGE:

```
functionalQA(opt)
```

For each run works on the realigned (and unwarped) data:

- plots motion, global signal, framewise displacement
- make a movie of the realigned time series
- computes additional confounds regressors depending on the options asked
- gets temporal SNR (TODO)
- creates a carpet plot of the data (TODO) ; warning this is slow

Relevant options:

```
opt.QA.func.Basics = 'on';
opt.QA.func.Motion = 'on';
opt.QA.func.FD = 'on';
opt.QA.func.Globals = 'on';
```

(continues on next page)

(continued from previous page)

```
opt.QA.func.Movie = 'on';
opt.QA.func.Voltera = 'on';
opt.QA.func.carpetPlot = true;
```

Parameters

opt (`structure`) – Options chosen for the analysis. See also: checkOptions

Warning: Because of a bug in spm_up, if `Voltera = 'on'`, then the confound regressors of framewise displacement, RMS and global signal will not be saved.

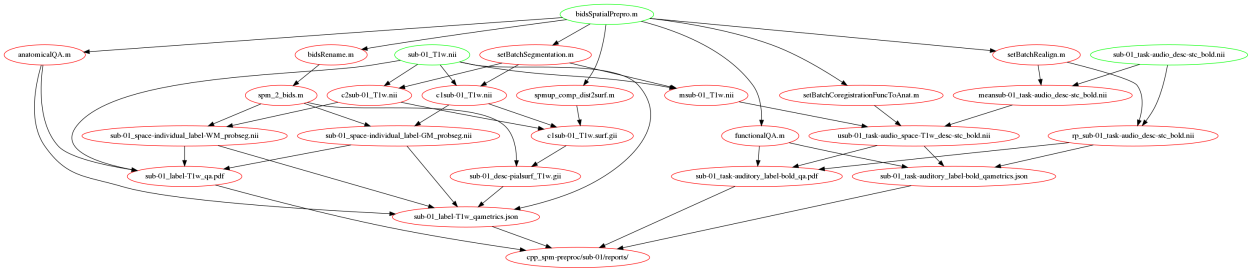


Fig. 1: workflows for QA as part of the spatial preprocessing workflow

`src.QA.computeDesignEfficiency`(*tsvFile*, *opt*)

Calculate efficiency for fMRI GLMs. Relies on Rik Henson's fMRI_GLM_efficiency function.

For more information on design efficiency, see Jeanette Mumford excellent videos and the dedicated videos from the Principles of fMRI Part 2, Module 7-9.

Warning: This function should NOT be used for proper design efficiency optimization as there are better tools for this.

In general see the BrainPower doc but more specifically the tools below:

- neuropower
- some of the Canlab tools

USAGE:

```
e = computeDesignEfficiency(tsvFile, opt)
```

Parameters

- **tsvFile** (`char`) – Path to a bids _events.tsv file.
- **opt** (`structure`) – Options chosen for the analysis. See also: `checkOptions()` and `loadAndCheckOptions()`.

Required:

- `opt.model.file`: path to bids stats model file
- `opt.TR`: inter-scan interval (s) - can be read from the `_bold.json`

Optional:

- `opt.t0`: initial transient (s) to ignore (default = 1)
- `opt.Ns`: number of scans

See also: fMRI_GLM_efficiency

—

EXAMPLE:

```
%% create stats model JSON
json = createEmptyStatsModel();
runStepIdx = 1;
json.Steps{runStepIdx}.Model.X = {'trial_type.cdt_A', 'trial_type.cdt_B'};
json.Steps{runStepIdx}.DummyContrasts = {'trial_type.cdt_A', 'trial_type.cdt_B'};

contrast = struct('type', 't', ...
                  'Name', 'A_gt_B', ...
                  'weights', [1, -1], ...
                  'ConditionList', {{'trial_type.cdt_A', 'trial_type.cdt_B'}});

json.Steps{runStepIdx}.Contrasts = contrast;

bids.util.jsonwrite('smdl.json', json);

%% create events TSV file
conditions = {'cdt_A', 'cdt_B'};
IBI = 5;
ISI = 0.1;
stimDuration = 1.5;
stimPerBlock = 12;
nbBlocks = 10;

trial_type = {};
onset = [];
duration = [];

time = 0;

for iBlock = 1:nbBlocks
  for cdt = 1:numel(conditions)
    for iTrial = 1:stimPerBlock
      trial_type{end + 1} = conditions{cdt};
      onset(end + 1) = time;
      duration(end + 1) = stimDuration;
      time = time + stimDuration + ISI;
    end
    time = time + IBI;
  end
end

tsv = struct('trial_type',  {trial_type}, 'onset', onset, 'duration', duration');

bids.util.tsvwrite('events.tsv', tsv);
```

(continues on next page)

(continued from previous page)

```
opt.TR = 2;

opt.model.file = fullfile(pwd, 'smdl.json');

e = computeDesignEfficiency(fullfile(pwd, 'events.tsv'), opt);
```

src.QA.**plotEvents**(*eventsFile*, *modelFile*)

USAGE:

```
plotEvents(eventsFile, modelFile)
```

Parameters

- **eventsFile** (char) – Path to a bids _events.tsv file.
- **modelFile** (structure) – Optional. Path to a bids statistical model file to filter what events to plot.

EXAMPLE:

```
dataDir = fullpath('bids-examples', 'ds001');

eventsFile = bids.query(dataDir, ...
                        'data', ...
                        'sub', '01', ...
                        'task', 'balloonanalogrisktask', ...
                        'suffix', 'events');

plotEvents(eventsFile{1});
```

CHAPTER

EIGHTEEN

LOW LEVEL FUNCTIONS DESCRIPTION

18.1 BIDS related functions

src.bids.**initBids**(*varargin*)

Initialize a BIDS dataset and updates dataset description.

USAGE:

```
initBids(opt, 'description', '', 'force', false)
```

Parameters
opt (structure) – Options chosen for the analysis. See also: checkOptions

src.bids.**addStcToQuery**(*BIDS*, *opt*, *subLabel*)

USAGE:

```
opt = addStcToQuery(opt, subLabel)
```

In case slice timing correction was performed this update the query to fetch the correct files for realignment.

src.bids.**removeEmptyQueryFields**(*query*)

src.bids.**getROIs**(*varargin*)

Get the rois from :

- the group folder when running analysis in MNI space
- the sub-*/roi/sub-subLabel folder when in individual space

USAGE:

```
[roiList, roiFolder] = getROIs(opt, subLabel)
```

src.bids.**getInfo**(*BIDS*, *subLabel*, *opt*, *info*, *varargin*)

Wrapper function to fetch specific info in a BIDS structure returned by spm_bids.

USAGE:

```
varargout = getInfo(BIDS, subLabel, opt, info, varargin)
```

If info = sessions, this returns name of the sessions and their number:

```
[sessions, nbSessions] = getInfo(BIDS, subLabel, opt, 'sessions')
```

If info = runs, this returns name of the runs and their number for a specified session:

```
[runs, nbRuns] = getInfo(BIDS, subLabel, opt, 'runs', sessionID)
```

If info = filename, this returns the name of the file for a specified session and run:

```
filenames = getInfo(BIDS, subLabel, opt, 'filename', sessionID, runID, suffix)
```

Parameters

- **BIDS** (structure) – dataset layout. See also: bids.layout, getData.
- **subLabel** (char) – label of the subject ; in BIDS lingo that means that for a file name sub-02_task-foo_bold.nii the subLabel will be the string 02
- **opt** (structure) – Options chosen for the analysis. See also: checkOptions() and loadAndCheckOptions().
- **info** (char) – sessions, runs, filename.
- **sessionLabel** (char) – session label (for *ses-001*, the label will be *001*)
- **runIdx** (char) – run index label (for *run-001*, the label will be *001*)
- **suffix** (char) – datatype (bold, events, physio)

src.bids.**getSubjectList**(*BIDS*, *opt*)

Returns the subjects to analyze in opt.subjects

USAGE:

```
opt = getSubjectList(BIDS, opt)
```

Parameters

- **BIDS** (structure) – dataset layout. See also: bids.layout, getData.
- **opt** (structure) – Options chosen for the analysis. See also: checkOptions() and loadAndCheckOptions().

Returns

- **opt**
  (structure)

If no group or subject is speficied in opt then all subjects are included. This is equivalent to the default:

```
opt.groups = {''};
opt.subjects = {[]};
```

If you want to run the analysis of some subjects only based on the group they belong to **as defined in the ``participants.tsv``** file, you can do it like this:

```
opt.groups = {'control'};
```

This will run the pipeline on all the control subjects.

If your subject label is blnd02 (as in sub-blnd02) but its group affiliation in the participants.tsv says control, then this subject will NOT be included if you run the pipeline with opt.groups = {'blnd'}.

If you have more than 2 groups you can specify them like this:

```
opt.groups = {'cont', 'cat'};
```

You can also directly specify the subject label for the participants you want to run:

```
opt.subjects = {'01', 'cont01', 'cat02', 'ctrl02', 'blind01'};
```

And you can combine both methods:

```
opt.groups = {'blind'};
opt.subjects = {'ctrl01'};
```

This will include all `blind` subjects and `sub-ctrl01`.

src.bids.**getAndCheckRepetitionTime**(*varargin*)

Gets the repetition time for a given bids.query filter (for several files) Throws an error if it returns empty or finds inconsistent repetition times.

USAGE:

```
repetitionTime = getAndCheckRepetitionTime(BIDS, filter)
```

Parameters

- **BIDS** (structure) – dataset layout. See also: bids.layout, getData.
- **filter** (structure) – obligatory argument.

Returns

- **repetitionTime**
  (float) (1x1)

Example:

```
filter = opt.query;
filter.sub =  subLabel;
filter.suffix = 'bold';
filter.extension = {'.nii', '.nii.gz'};
filter.prefix = '';
filter.task = opt.taskName;

TR = getAndCheckRepetitionTime(BIDS, filter);
```

src.bids.**getAndCheckSliceOrder**(*BIDS*, *opt*, *filter*)

Get the slice order information from the BIDS metadata. If inconsistent slice timing is found across files it returns empty and throws a warning.

USAGE:

```
sliceOrder = getAndCheckSliceOrder(opt)
```

Parameters

- **BIDS** (structure) – dataset layout. See also: bids.layout, getData.
- **opt** (structure) – Options chosen for the analysis. See also: `checkOptions()` and `loadAndCheckOptions()`.

Returns

- **sliceOrder**
  a vector of the time when each slice was acquired in in a volume or indicating the order of acquisition of the slices.

`getAndCheckSliceOrder` will try to read the `opt` structure for any relevant information about slice timing. If this is empty, it queries the BIDS dataset to ee if there is any consistent slice timing information for a given `filter`

See also: bidsSTC, setBatchSTC

`src.bids.`**`getTpmFilename`**`(`*BIDS*, *anatImage*, *res*, *space*`)`

Gets the fullpath filenames of the tissue probability maps (TPM)

USAGE:

```
[gm, wm, csf] = getTpmFilenames(BIDS, opt, subLabel, space, res)
```

Parameters

- **BIDS** (`structure`) – dataset layout. See also: bids.layout, getData.
- **opt** (`structure`) – Options chosen for the analysis. See also: `checkOptions()` and `loadAndCheckOptions()`.
- **anatImage** –
- **anatImage** – char
- **space** –
- **space** – char
- **res** –
- **res** – char

Returns

- **gm**
  (string) grey matter TPM
- **wm**
  (string) white matter TPM
- **csf**
  (string) csf matter TPM

`src.bids.`**`getMeanFuncFilename`**`(`*BIDS*, *subLabel*, *opt*`)`

Get the filename and the directory of an mean functional file.

USAGE:

```
[meanImage, meanFuncDir] = getMeanFuncFilename(BIDS, subLabel, opt)
```

Parameters

- **BIDS** (`structure`) – dataset layout. See also: bids.layout, getData.
- **subLabel** (`char`) –

- **opt** (structure) – Options chosen for the analysis. See also: checkOptions() and loadAndCheckOptions().

Returns

- **meanImage**
  (string)
- **meanFuncDir**
  (string)

src.bids.**getBoldFilename**(*varargin*)

Get the filename and the directory of a bold file for a given session / run.

Unzips the file if necessary.

USAGE:

```
[boldFilename, subFuncDataDir] = getBoldFilename(BIDS, subLabel, sessionID, runID,␣
↪opt)
```

Parameters

- **BIDS** (structure) – dataset layout. See also: bids.layout, getData.
- **subLabel** (char) – label of the subject ; in BIDS lingo that means that for a file name sub-02_task-foo_bold.nii the subLabel will be the string 02
- **sessionID** (char) – session label (for *ses-001*, the label will be *001*)
- **runID** (char) – run index label (for *run-001*, the label will be *001*)
- **opt** (structure) – Options chosen for the analysis. See also: checkOptions() and loadAndCheckOptions().

Returns

- **boldFilename**
  (string)
- **subFuncDataDir**
  (string)

src.bids.**getAnatFilename**(*varargin*)

Get the filename and the directory of some anat files for a given session and run. Unzips the files if necessary.

It several images are available it will take the first one it finds.

USAGE:

```
[anatImage, anatDataDir] = getAnatFilename(BIDS, subLabel, opt, nbImgToReturn,␣
↪tolerant)
```

Parameters

- **BIDS** (structure) – dataset layout. See also: bids.layout, getData.
- **subLabel** (char) –
- **opt** (structure) – Options chosen for the analysis. See also: checkOptions() and loadAndCheckOptions().

Returns

- **anatImage**
  (string)
- **anatDataDir**
  (string)

18.2 Input / Output

src.IO.**getData**(*varargin*)

Reads the specified BIDS data set and updates the list of subjects to analyze.

USAGE:

```
[BIDS, opt] = getData(opt, bidsDir)
```

Parameters

- **opt** (structure) – Options chosen for the analysis. See also: checkOptions
- **bidsDir** (char) – the directory where the data is ; default is : fullfile(opt.dataDir, '..', 'derivatives', 'bidspm')

Returns

- **opt**
  (structure)
- **BIDS**
  (structure)

src.IO.**saveOptions**(*opt*)

Saves options in a JSON file in a cfg folder.

USAGE:

```
saveOptions(opt)
```

Parameters

opt (structure) – Options chosen for the analysis. See also: checkOptions

src.IO.**loadAndCheckOptions**(*optionJsonFile*)

Loads the json file provided describing the options of an analysis. It then checks its content and fills any missing fields with the defaults.

If no argument is provived, it checks in the current directory for any opt_task-*.json files and loads the most recent one by name (using the date- key).

USAGE:

```
opt = loadAndCheckOptions(optionJsonFile)
```

Parameters

optionJsonFile (char) – Fullpath to the json file describing the options of an analysis. It can also be an opt structure containing the options.

Returns

opt
(structure) Options chosen for the analysis. See `checkOptions()`.

src.IO.**overwriteDir**(*directory*, *opt*)

USAGE:

```
overwriteDir(directory, opt)
```

src.IO.**createDerivativeDir**(*opt*)

Creates the derivative folder if it does not exist.

USAGE:

```
opt = createDerivativeDir(opt)
```

Parameters
opt (`structure`) – Options chosen for the analysis. See also: checkOptions

src.IO.**saveSpmScript**(*varargin*)

Saves a matlabbatch as .m file

USAGE:

```
outputFilename = saveSpmScript(input, outputFilename)
```

Parameters

- **input** – a `matlabbatch` variable (cell) or the fullpath to a `.mat` file containing such `matlabbatch` variable.
- **outputFilename** (`path`) – optional. Path to output file

Returns

- **outputFilename**
  (path)

src.IO.**unzipAndReturnsFullpathName**(*fullpathName*, *opt*)

Unzips a file if necessary

USAGE:

```
unzippedFullpathName = unzipAndReturnsFullpathName(fullpathName)
```

Parameters
fullpathName (`char array`) –

Returns

- **unzippedFullpathName**
  (string)

src.IO.**onsetsMatToTsv**(*varargin*)

Takes an SPM _onset.mat file and converts it to a _onsets.mat file.

Onsets are assumed to be in seconds.

USAGE:

```
onsetTsvFile = onsetMatToTsv(onsetMatFile)
```

Parameters
onsetMatFile (fullpath) – obligatory argument.

Returns

- **onsetTsvFile**
  (path)

src.IO.**regressorsMatToTsv**(*varargin*)

Takes an SPM _desc-confounds_regressors.mat file and converts it to a _desc-confounds_regressors.tsv file.

USAGE:

```
regressorsTsvFile = regressorsMatToTsv(regressorsMatFile)
```

Parameters
regressorsMatFile (fullpath) – obligatory argument.

Returns

- **regressorsTsvFile**
  (path)

src.IO.**renameUnwarpParameter**(*BIDS*, *subLabel*, *opt*)

USAGE:

```
renameUnwarpParameter(BIDS, subLabel, opt)
```

src.IO.**renameSegmentParameter**(*BIDS*, *subLabel*, *opt*)

USAGE:

```
renameSegmentParameter(BIDS, subLabel, opt)
```

src.IO.**cleanCrash**()

Removes any files left over from a previous unfinished run of the pipeline, like any *.png imgages

USAGE:

```
cleanCrash()
```

18.3 Utility functions

src.utils.**createDataDictionary**(*tsvContent*)

USAGE:

```
jsonContent = createDataDictionary(tsvContent)
```

src.utils.**createGlmDirName**(*opt*)

USAGE:

```
glmDirName = createGlmDirName(opt)
```

src.utils.**getFuncVoxelDims**(*opt*, *subFuncDataDir*, *fileName*)

Short description of what the function does goes here.

USAGE:

```
[voxDim, opt] = getFuncVoxelDims(opt, subFuncDataDir, fileName)
```

Parameters

- **opt** (structure) – Options chosen for the analysis. See also: checkOptions
- **subFuncDataDir** –
- **fileName** –

Returns

- **voxDim**
- **opt**

src.utils.**removeDummies**(*varargin*)

Short description of what the function does goes here.

USAGE:

```
removeDummies(inputFile, dummyScans, metadata, 'force', false, 'verbose', true)
```

Parameters

- **inputFile** (structure) –
- **dummyScans** (positive integer) – number of dummy scans to remove
- **metadata** (structure) –
- **force** (boolean) –
- **verbose** (boolean) –

src.utils.**returnVolumeList**(*varargin*)

USAGE:

```
volumes = returnVolumeList(opt, boldFile)
```

Parameters

- **opt** (structure) – Options chosen for the analysis. See also: checkOptions
- **boldFile** (fullpath) –

Returns

- **volumes**
  (cell string)

src.utils.**volumeSplicing**(*varargin*)

Removes specific set of volumes from a nifti time series.

USAGE:

```
outputFileFullPath = volumeSplicing(inputFile, volumesToRemove)
```

Parameters

- **inputFile** (path) –
- **volumesToRemove** (1xn or nX1 array) –
- **outputFile** (char) – optional parameter. default: will overwrite inputFile. If only a filename is given, the file will be created in the same folder as the input file.

Returns

- **outputFileFullPath**

Example:

```
outputFileFullPath = volumeSplicing(inputFile, volumesToRemove, 'outputFile', 'foo.
↪nii.gz');
```

src.utils.**labelActivations**(*varargin*)

Adds MNI labels to a csv output file from SPM and saves it as SPM.

Can choose which atlas to use.

USAGE:

```
tsvFile = labelActivations(csvFile, 'atlas', 'Neuromorphometrics')
```

Parameters

- **csvFile** (path) –
- **atlas** (char) – Any of {'Neuromorphometrics', 'AAL'}. Defaults to 'Neuromorphometrics'

Returns

- **tsvFile**
  (path)

src.utils.**getContrastNb**(*result*, *opt*, *SPM*)

Identify the contrast nb actually has the name the user asked

The search is regex based and any string (like 'foo') will be by default regexified (into '^foo$').

USAGE:

```
contrastNb = getContrastNb(result, opt, SPM)
```

src.utils.**getRegressorIdx**(*cdtName*, *SPM*)

Gets from the SPM structure the regressors index corresponding to the a condition convolved with the canonical HRF. This can also look for non convolved conditions to identify a confound regressor.

Throws a warning if there is no regressor for that condition.

USAGE:

```
[cdtName, regIdx, status] = getRegressorIdx(cdtName, SPM)
```

Parameters

- **cdtName** (`char or cellstr`) – name of the condition to look for
- **SPM** (`structure`) – content of SPM.mat

Returns

- **cdtName**
  (char) name of the condition stripped of any eventual `'trial_type.'` prefix
- **regIdx**
  (logical) vector of the columns of the design matrix containing the regressor of interest
- **status**
  (logical) is `false` if no regressor was found for that condition

src.utils.**getDist2surf**(*varargin*)

Loads the pial surface and computes the mean distance to the surface.

USAGE:

```
davg = getDist2surf(anatImage, opt)
```

Parameters

- **anatImage** (`cell`) –
- **opt** (`structure`) – Options chosen for the analysis. See also: checkOptions

Returns

- **davg**
  (float) (1 x 1)

src.utils.**computeMeanValueInMask**(*image*, *mask*)

USAGE:

```
value = computeMeanValueInMask(image, mask)
```

image: image filename mask: mask filename

src.utils.**computeTsnr**(*boldImage*)

calculate temporal SNR from single run of fMRI timeseries data

USAGE:

```
[tsnrImage, volTsnr] = computeTsnr(boldImage)
```

Parameters

boldImage (path) – path to the 4D nifti image. The file must have a BIDS like name (example: key1_label1_key2-label2_suffic.nii)

Output:

- tsnrImage: fullpath filename of the tSNR output image
- volTsnr: 3D volume of the tSNR image

Adapted from fmrwhy: https://github.com/jsheunis/fMRwhy/blob/master/fmrwhy/qc/fmrwhy_qc_calculateStats.m

Copyright 2019 Stephan Heunis

src.utils.**setFields**(*structure*, *fieldsToSet*, *overwrite*)

Recursively loop through the fields of a target structure and sets the values as defined in the structure fieldsToSet if they don't exist.

Content of the target structure can be overwritten by setting the overwrite```to ``true.

USAGE:

```
structure = setFields(structure, fieldsToSet, overwrite = false)
```

Parameters

- **structure** –
- **fieldsToSet** (char) –
- **overwrite** (boolean) –

Returns

- **structure**
  (structure)

src.utils.**validationInputFile**(*dir*, *fileNamePattern*, *prefix*)

Looks for file name pattern in a given directory and returns all the files that match that pattern but throws an error if it cannot find any.

A prefix can be added to the filename.

This function is mostly used that a file exists so that an error is thrown early when building a SPM job rather than at run time.

USAGE:

```
files = validationInputFile(dir, fileName, prefix)
```

Parameters

- **dir** (char) – Directory where the search will be conducted.
- **fileName** (char) – file name pattern. Can be a regular expression except for the starting ^ and ending $. For example: 'sub-.*_ses-.*_task-.*_bold.nii'.

- **prefix** (char) – prefix to be added to the filename pattern. This can also be a regular expression (ish). For example ,f looking for the files that start with `c1` or `c2` or `c3`, the prefix can be `c[123]`.

Returns

files
(string array) returns the fullpath file list of all the files matching the required pattern.

See also: spm_select

Example: % % tissueProbaMaps = validationInputFile(anatDataDir, anatImage, 'c[12]');

18.4 Print and error handling

`src.messages.`**printToScreen**(*varargin*)

USAGE:

```
printToScreen(msg, opt, 'format', 'blue')
```

`src.messages.`**errorHandling**(*varargin*)

USAGE:

```
errorHandling(functionName, id, msg, tolerant, verbose)
```

Parameters

- **functionName** (char) –
- **id** (char) – Error or warning id
- **msg** (char) – Message to print
- **tolerant** (boolean) – If set to `true` errors are converted into warnings
- **verbose** (boolean) – If set to `0` or `false` this will silence any warning

EXAMPLE:

```
msg = sprintf('this error happened with this file %s', filename)
id = 'thisError';
errorHandling(mfilename(), id, msg, true, opt.verbosity)
```

adapted from bids-matlab

`src.messages.`**createUnorderedList**(*list*)

turns a cell string or a structure into a string that is an unordered list to print to the screen

USAGE:

```
list = createUnorderedList(list)
```

Parameters

list (cell string or structure) – obligatory argument.

src.messages.**printAvailableContrasts**(*SPM*, *opt*)

src.messages.**printWorkflowName**(*workflowName*, *opt*)

src.messages.**printBatchName**(*batchName*, *opt*)

src.messages.**printCredits**(*opt*)

(C) Copyright 2019 bidspm developers

src.messages.**printProcessingSubject**(*iSub*, *subLabel*, *opt*)

USAGE:

```
printProcessingSubject(iSub, subLabel, opt)
```

18.5 Infrastructure related functions

src.infra.**checkDependencies**(*opt*)

Checks that that the right dependencies are installeda and loads the spm defaults.

USAGE:

```
checkDependencies()
```

src.infra.**checkToolbox**(*varargin*)

Checks that a given SPM toolbox is installed. Possible to install it if necessary.

USAGE:

```
status = checkToolbox(toolboxName, 'verbose', false, 'install', false)
```

Parameters

- **toolboxName** (char) – obligatory argument. Any of {'ALI', 'MACS', 'mp2rage'}.
- **verbose** (boolean) – parameter
- **install** (boolean) – parameter

EXAMPLE:

```
checkToolbox('MACS', 'verbose', true, 'install', true)
```

src.infra.**getEnvInfo**(*opt*)

Gets information about the environement and operating system to help generate data descriptors for the derivatives.

USAGE:

```
[OS, generatedBy] = getEnvInfo()
```

Returns

OS
(structure) (dimension)

generatedBy
(structure) (dimension)

src.infra.**getVersion**()

Reads the version number of the pipeline from the txt file in the root of the repository.

USAGE:

```
versionNumber = getVersion()
```

Returns

versionNumber
(string) Use semantic versioning format (like v0.1.0)

src.infra.**isOctave**()

Returns true if the environment is Octave.

USAGE:

```
retval = isOctave()
```

Returns

retval
(boolean)

src.infra.**setGraphicWindow**(*opt*)

Short description of what the function does goes here.

USAGE:

```
[interactiveWindow, graphWindow, cmdLine] = setGraphicWindow(opt)
```

Parameters
opt (structure) – Options chosen for the analysis. See also: checkOptions

Returns

- **interactiveWindow**
- **graphWindow**
- **cmdLine**
  (boolean)

CHAPTER

NINETEEN

MANUAL COREGISTRATION

Manual coregistration tools

---

lib.mancoreg.**mancoreg**(*varargin*)

This function displays 2 SPM ortho-views of a `targetimage` and a `sourceimage` image that can be manually coregistered.

USAGE:

```
mancoreg('targetimage', [], 'sourceimage', [], 'stepsize', 0.01)
```

Parameters

- **targetimage** (`string`) – Filename or fullpath of the target image. If none is provided you will be asked by SPM to select one.
- **sourceimage** (`string`) – Filename or fullpath of the source image. If none is provided you will be asked by SPM to select one.
- **stepsize** (`positive float`) – step size for each rotation and translation

Manual coregistration tool

The source image (bottom graph) can be manually rotated and translated with 6 slider controls. In the source graph the source image can be exchanged with the target image using a radio button toggle. This is helpful for visual fine control of the coregistration. The transformation matrix can be applied to a selected set of volumes with the `apply transformation` button. If the transformation is to be applied to the original source file that file will also need to be selected. If the sourceimage or targetimage are not passed the user will be prompted with a file browser.

The code is loosely based on `spm_image()` and `spm_orthoviews()` It requires the m-file with the callback functions for the user controls (`mancoregCallbacks()`).

lib.mancoreg.**mancoregCallbacks**(*operation*)

Callback routines for `mancoreg()`: defines the different actions for the different buttons.

USAGE:

```
mancoreg_callbacks(operation)
```

Parameters

operation (string) – Can be any of the following: move, toggle_off, toggle_on, reset, apply, plotmat

CHAPTER

TWENTY

DOCKER

20.1 Build docker image locally

If you want to build the docker image locally and not pull it from the docker hub:

```
docker build . -f docker/Dockerfile -t cpplab/bidspm:stable
```

This will create an image with the tag name `bidspm:stable`

Running `make docker_img` will also build the `stable` version and a `latest` version.

20.2 Run docker image

The following command would pull from our docker hub and start the docker image:

```
docker run -it --rm cpplab/bidspm:latest
```

The image is set up to start Octave in the `/code` folder.

The following command would do do the same, but it would also map 2 folders from your computer to the `output` and `code` folder inside the container image:

```
code_folder=fullpath_to_your_code
output_folder=fullpath_to_your_output_folder

docker run -it --rm \
    -v $output_folder:/output \
    -v $code_folder:/code \
    cpplab/bidspm:latest
```

For example, you could run the demos by doing this:

```
code_folder=/home/remi/github/bidspm/demos/MoAE

docker run -it --rm \
    -v $code_folder:/code \
    cpplab/bidspm:latest

# once inside the docker image
moae_01_preproc
```

CHAPTER

TWENTYONE

LINKS AND REFERENCES

21.1 SPM starters

If you start from zero, go through the 2 first tutorials of SPM

21.2 Andrew Jahn videos and blogs

- video playlist
- marsbar
- SPM
- SPM.mat

21.3 SPM code snippets

SPM wikibook has some very useful sections.

From John Ashburner on Tom Nichols blog

From Rik Henson

Some follow along tutorials written a long time ago, and that probably should be turned into notebooks and updated.

- Basic file / image manipulation with SPM
- HRF, convolution and GLM ("by hand")
- Design efficiency

21.4 Content of SPM.mat

This is here because SPM has the sad (and bad) Matlabic tradition of using variable names that have often attempted to replicate the notation in the papers to make engineers and the generally math enclined happy, rather then the `TypicalLongVariableNames` that many programmers and new comers would prefer to see to help with code readibility.

Adapted from: http://andysbrainblog.blogspot.com/2013/10/whats-in-spmmat-file.html

21.4.1 details on experiment

- `SPM.xY.RT` - TR length (RT =”repeat time”)
- `SPM.xY.P` - matrix of file names
- `SPM.xY.VY` - (number of runs x 1) struct array of mapped image volumes (.nii file info)
- `SPM.modality` - the data you’re using (PET, FMRI, EEG)
- `SPM.stats.[modality].UFp` - critical F-threshold for selecting voxels over which the non-sphericity is estimated (if required) [default: `0.001`]
- `SPM.stats.maxres` - maximum number of residual images for smoothness estimation
- `SPM.stats.maxmem` - maximum amount of data processed at a time (in bytes)
- `SPM.SPMid` - version of SPM used
- `SPM.swd` - directory for SPM.mat and nii files. default is `pwd`

21.4.2 basis function

- `SPM.xBF.name` - name of basis function
- `SPM.xBF.length` - length in seconds of basis
- `SPM.xBF.order` - order of basis set
- `SPM.xBF.T` - number of subdivisions of TR
- `SPM.xBF.T0` - first time bin (see slice timing)
- `SPM.xBF.UNITS` - options: `'scans'` or `'secs'` for onsets
- `SPM.xBF.Volterra` - order of convolution
- `SPM.xBF.dt` - length of time bin in seconds
- `SPM.xBF.bf` - basis set matrix

21.4.3 Session structure

Note that in SPM lingo sessions are equivalent to a runs in BIDS.

user-specified covariates/regressors

e.g. motion

- `SPM.Sess([sesssion]).C.C` - (n x c) double regressor (`c` is number of covariates, `n` is number of sessions)
- `SPM.Sess([sesssion]).C.name` - names of covariates

conditions & modulators specified

i.e. input structure array

- `SPM.Sess([sesssion]).U(condition).dt`: - time bin length (seconds)
- `SPM.Sess([sesssion]).U(condition).name` - names of conditions
- `SPM.Sess([sesssion]).U(condition).ons` - onset for condition's trials
- `SPM.Sess([sesssion]).U(condition).dur` - duration for condition's trials
- `SPM.Sess([sesssion]).U(condition).u` - (t x j) inputs or stimulus function matrix
- `SPM.Sess([sesssion]).U(condition).pst` - (1 x k) peri-stimulus times (seconds)

parameters/modulators specified

- `SPM.Sess([sesssion]).U(condition).P` - parameter structure/matrix
- `SPM.Sess([sesssion]).U(condition).P.name` - names of modulators/parameters
- `SPM.Sess([sesssion]).U(condition).P.h` - polynomial order of modulating parameter (order of polynomial expansion where 0 is none)
- `SPM.Sess([sesssion]).U(condition).P.P` - vector of modulating values
- `SPM.Sess([sesssion]).U(condition).P.P.i` - sub-indices of `U(i).u` for plotting

scan indices for sessions

- `SPM.Sess([sesssion]).row`

effect indices for sessions

- `SPM.Sess([sesssion]).col`

F Contrast information for input-specific effects

- `SPM.Sess([sesssion]).Fc`
- `SPM.Sess([sesssion]).Fc.i` - F Contrast columns for input-specific effects
- `SPM.Sess([sesssion]).Fc.name` - F Contrast names for input-specific effects
- `SPM.nscan([session])` - number of scans per session (or if e.g. a t-test, total number of con*.nii files)

21.4.4 global variate/normalization details

- `SPM.xGX.iGXcalc` - either `'none'` or `'scaling'`

For fMRI usually is `none` (no global normalization). If global normalization is `scaling`, see `spm_fmri_spm_ui` for parameters that will then appear under `SPM.xGX`.

21.4.5 design matrix information

- SPM.xX.X - design matrix (raw, not temporally smoothed)
- SPM.xX.name - cellstr of parameter names corresponding to columns of design matrix
- SPM.xX.I - (nScan x 4) matrix of factor level indicators. first column is the replication number. Other columns are the levels of each experimental factor.
- SPM.xX.iH - vector of H partition (indicator variables) indices
- SPM.xX.iC - vector of C partition (covariates) indices
- SPM.xX.iB - vector of B partition (block effects) indices
- SPM.xX.iG - vector of G partition (nuisance variables) indices
- SPM.xX.K - cell. low frequency confound: high-pass cutoff (seconds)
- SPM.xX.K.HParam - low frequency cutoff value
- SPM.xX.K.X0 - cosines (high-pass filter)
- SPM.xX.W - Optional whitening/weighting matrix used to give weighted least squares estimates (WLS). If not specified spm_spm will set this to whiten the data and render the OLS estimates maximum likelihood i.e. W*W' inv(xVi.V).
- SPM.xX.xKXs - space structure for K*W*X, the 'filtered and whitened' design matrix
  - SPM.xX.xKXs.X - matrix of trials and betas (columns) in each trial
  - SPM.xX.xKXs.tol - tolerance
  - SPM.xX.xKXs.ds - vectors of singular values
  - SPM.xX.xKXs.u - u as in X u*diag(ds)*v'
  - SPM.xX.xKXs.v - v as in X u*diag(ds)*v'
  - SPM.xX.xKXs.rk - rank
  - SPM.xX.xKXs.oP - orthogonal projector on X
  - SPM.xX.xKXs.oPp - orthogonal projector on X'
  - SPM.xX.xKXs.ups - space in which this one is embedded
  - SPM.xX.xKXs.sus - subspace
- SPM.xX.pKX - pseudoinverse of K*W*X, computed by spm_sp
- SPM.xX.Bcov - xX.pKX*xX.V*xX.pKX - variance-covariance matrix of parameter estimates (when multiplied by the voxel-specific hyperparameter ResMS of the parameter estimates (ResSS/xX.trRV ResMS))
- SPM.xX.trRV - trace of R*V
- SPM.xX.trRVRV - trace of RVRV
- SPM.xX.erdf - effective residual degrees of freedom (trRV^2/trRVRV)
- SPM.xX.nKX - design matrix (xX.xKXs.X) scaled for display (see spm_DesMtx('sca',... for details)
- SPM.xX.sF - cellstr of factor names (columns in SPM.xX.I, i think)
- SPM.xX.D - struct, design definition
- SPM.xX.xVi - correlation constraints (see non-sphericity below)
- SPM.xC - struct. array of covariate info

21.4.6 header info

- SPM.P - a matrix of filenames
- SPM.V - a vector of structures containing image volume information.
  - SPM.V.fname - the filename of the image.
  - SPM.V.dim - the x, y and z dimensions of the volume
  - SPM.V.dt - a (1 x 2) array. First element is datatype (see spm_type). The second is 1 or 0 depending on the endian-ness.
  - SPM.V.mat - a (4 x 4) affine transformation matrix mapping from voxel coordinates to real world coordinates.
  - SPM.V.pinfo - plane info for each plane of the volume.
  - SPM.V.pinfo(1,:) - scale for each plane
  - SPM.V.pinfo(2,:) - offset for each plane The true voxel intensities of the j:sup:*th* image are given by: val*V.pinfo(1,j) + V.pinfo(2,j)
  - SPM.V.pinfo(3,:) - offset into image (in bytes). If the size of pinfo is 3x1, then the volume is assumed to be contiguous and each plane has the same scale factor and offset.

21.4.7 structure describing intrinsic temporal non-sphericity

- SPM.xVi.I - typically the same as SPM.xX.I
- SPM.xVi.h - hyperparameters
- SPM.xVi.V xVi.h(1)*xVi.Vi{1} + ...
- SPM.xVi.Cy - spatially whitened (used by ReML to estimate h)
- SPM.xVi.CY - <(Y -)*(Y -)'> (used by spm_spm_Bayes)
- SPM.xVi.Vi - array of non-sphericity components
  - defaults to {speye(size(xX.X,1))} - i.i.d.
  - specifying a cell array of contraints ((Qi)
  - These contraints invoke spm_reml to estimate hyperparameters assuming V is constant over voxels that provide a high precise estimate of xX.V
- SPM.xVi.form - form of non-sphericity (either 'none' or 'AR(1)' or 'FAST')
- SPM.xX.V - Optional non-sphericity matrix. CCov(e)sigma^2*V. If not specified spm_spm will compute this using a 1st pass to identify signifcant voxels over which to estimate V. A 2nd pass is then used to re-estimate the parameters with WLS and save the ML estimates (unless xX.W is already specified).

21.4.8 filtering information

- SPM.K - filter matrix or filtered structure
  - SPM.K(s) - struct array containing partition-specific specifications
  - SPM.K(s).RT - observation interval in seconds
  - SPM.K(s).row - row of Y constituting block/partitions
  - SPM.K(s).HParam - cut-off period in seconds
  - SPM.K(s).X0 - low frequencies to be removed (DCT)
- SPM.Y - filtered data matrix

21.4.9 masking information

- SPM.xM - Structure containing masking information, or a simple column vector of thresholds corresponding to the images in VY.
- SPM.xM.T - (n x 1) double - Masking index
- SPM.xM.TH - (nVar x nScan) matrix of analysis thresholds, one per image
- SPM.xM.I - Implicit masking (0 –> none; 1 –> implicit zero/NaN mask)
- SPM.xM.VM - struct array of mapped explicit mask image volumes
- SPM.xM.xs - (1 x 1) struct ; cellstr description

21.4.10 design information

self-explanatory names, for once

- SPM.xsDes.Basis_functions - type of basis function
- SPM.xsDes.Number_of_sessions
- SPM.xsDes.Trials_per_session
- SPM.xsDes.Interscan_interval
- SPM.xsDes.High_pass_Filter
- SPM.xsDes.Global_calculation
- SPM.xsDes.Grand_mean_scaling
- SPM.xsDes.Global_normalisation

21.4.11 details on scanner data

e.g. smoothness

- SPM.xVol - structure containing details of volume analyzed
  - SPM.xVol.M - (4 x 4) voxel –> mm transformation matrix
  - SPM.xVol.iM - (4 x 4) mm –> voxel transformation matrix
  - SPM.xVol.DIM - image dimensions - column vector (in voxels)

  - SPM.xVol.XYZ - (3 x S) vector of in-mask voxel coordinates
  - SPM.xVol.S - Lebesgue measure or volume (in voxels)
  - SPM.xVol.R - vector of resel counts (in resels)
  - SPM.xVol.FWHM - Smoothness of components - FWHM, (in voxels)

21.4.12 info on beta files

- SPM.Vbeta - struct array of beta image handles
  - SPM.Vbeta.fname - beta nii file names
  - SPM.Vbeta.descrip - names for each beta file

21.4.13 info on variance of the error

- SPM.VResMS - file struct of ResMS image handle
  - SPM.VResMS.fname - variance of error file name

21.4.14 info on mask

- SPM.VM - file struct of Mask image handle
  - SPM.VM.fname - name of mask nii file

21.4.15 contrast details

added after running contrasts

- SPM.xCon - Contrast definitions structure array. See also spm_FcUtil.m for structure, rules & handling.
  - SPM.xCon.name - Contrast name
  - SPM.xCon.STAT - Statistic indicator character ('T', 'F' or 'P')
  - SPM.xCon.c - Contrast weights (column vector contrasts)
  - SPM.xCon.X0 - Reduced design matrix data (spans design space under Ho)
    - Stored as coordinates in the orthogonal basis of xX.X from spm_sp (Matrix in SPM99b)
    - Extract using X0 spm_FcUtil('X0', ...
  - SPM.xCon.iX0 - Indicates how contrast was specified:
    - If by columns for reduced design matrix then iX0 contains the column indices.
    - Otherwise, it's a string containing the spm_FcUtil 'Set' action: Usually one of {'c','c+','X0'} defines the indices of the columns that will not be tested. Can be empty.
  - SPM.xCon.X1o - Remaining design space data (X1o is orthogonal to X0)
    - Stored as coordinates in the orthogonal basis of xX.X from spm_sp (Matrix in SPM99b)
    - Extract using X1o spm_FcUtil('X1o', ...
  - SPM.xCon.eidf - Effective interest degrees of freedom (numerator df)

        * Or effect-size threshold for Posterior probability
    - `SPM.xCon.Vcon` - Name of contrast (for 'T's) or ESS (for 'F's) image
    - `SPM.xCon.Vspm` - Name of SPM image

21.5 Bilbiography

CHAPTER

TWENTYTWO

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[OARW19] Wiktor Olszowy, John Aston, Catarina Rua, and Guy B. Williams. Accurate autocorrelation modeling substantially improves fMRI reliability. *Nature Communications*, 10(1):1220, 2019. URL: http://www.nature.com/articles/s41467-019-09230-w (visited on 2022-06-10), doi:10.1038/s41467-019-09230-w.

MATLAB MODULE INDEX

INDEX

I

L

M

O

P

R

S

U

V