CPP SPM

Release v1.1.3dev

the CPP SPM pipeline dev team

Jan 14, 2022

CONTENT

CHAPTER

ONE

INSTALLATION

1.1 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.

1.1.1 Octave compatibility

The following features do not yet work with Octave:

- anatomicalQA
- functionalQA
- slice_display toolbox

1.2 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 CPP_SPM itself, we STRONGLY suggest you use this template repository to create a new project with a basic structure of folders and with the CPP_SPM 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/CPP_SPM.git
```

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

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

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

1.3 Initialization

In general DO NOT ADD CPP SPM PERMANENTLY to your MATLAB / Octave path.

You just need to initialize for a given session with:

```
initCppSpm()
```

This will add all the required folders to the path.

You can also remove CPP_SPM from the path with:

```
uninitCppSpm()
```

1.4 Installation on a computing cluster

This relies on the fact that SPM and CPM SPM are Octave compatible, so you will be able to run most of CPP SPM 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 CPP SPM 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 CPP SPM

As before install and run an initialization:

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

Warning: There are some warnings thrown during initialization:

```
octave:1> initCppSpm
warning: addpath: /home/users/r/g/rgau/CPP_SPM/lib/spmup/utlilities/home/users/r/g/
↪rgau/CPP_SPM/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/CPP_SPM/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/CPP_SPM/src/workflows/stats/home/users/r/g/
↪rgau/CPP_SPM/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

TWO

SET UP

2.1 Configuration of the pipeline

2.1.1 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 we often label `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.

Set the task to analyze in the BIDS data set `opt.taskName = 'auditory'`

Selecting groups and subjects

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

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

Returns the subjects to analyze in `opt.subjects`

USAGE:

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

Parameters

- **`BIDS`** (`structure`) – output of bids.layout
- **`opt`** (`structure`) – Options chosen for the analysis. See `checkOptions()`.

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`.

2.1.2 BIDS model JSON files

This files allow you to specify the GLM to run and which contrasts to run. It follows BIDS statistical model

The model json file that describes:

- out to prepare the regressors for the GLM: `Transformation`
- the design matrix: `X`
- the contrasts to compute: `contrasts`

It also allows to specify those for different levels of the analysis:

- run
- session
- subject
- dataset

An example of json file could look something like that:

```
{
    "Name": "Basic",
    "Description": "",
    "Input": {
        "task": "motionloc"
    },
    "Nodes": [
        {
            "Level": "Subject",
```

(continues on next page)

(continued from previous page)

```
            "DummyContrasts": ["stim_type.motion", "stim_type.static"],
            "Contrasts": [
                {
                    "Name": "motion_vs_static",
                    "ConditionList": ["stim_type.motion", "stim_type.static"],
                    "weights": [1, -1],
                    "type": "t"
                }
            ]
        },
        {
            "Level": "Dataset",
            "DummyContrasts": [
                "stim_type.motion",
                "stim_type.static",
                "motion_vs_static"
            ]
        }
    ]
}
```

In brief this means:

- at the subject level automatically compute the t contrast against baseline for the condition `motion` and `static` and compute the t-contrats for motion VS static with these given weights.
- at the level of the data set (so RFX) do the t contrast of the `motion`, `static`, `motion VS static`.

We are currently using this to run different subject level GLM models for our univariate and multivariate analysis where in the first one we compute a con image that averages the beta images of all the runs where as in the latter case we get one con image for each run.

To create a default model for a given BIDS dataset you can do this:

```
path_to_dataset = fullfile(pwd, 'data', 'raw');
BIDS = bids.layout(path_to_dataset);

opt.taskName = 'myFascinatingTask';

createDefaultStatsModel(BIDS, opt);
```

CHAPTER

THREE

PIPELINE DEFAULTS

Defaults of the pipeline.

3.1 checkOptions

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

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

USAGE:

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

Parameters **opt** (structure) – structure or json filename containing the options.

Returns

- **opt** the option structure with missing values filled in by the defaults.

IMPORTANT OPTIONS (with their defaults):

- opt.taskName
- 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.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.model.file = '' - path to the BIDS model file that contains the model to speficy and the contrasts to compute.
- '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.
- opt.fwhm.func = 6 - FWHM to apply to the preprocessed functional images.

- `opt.fwhm.contrast = 6` - FWHM to apply to the contrast images before bringing them at the group level.
- `opt.query` - a structure used to specify other options to only run analysis on certain files. `struct('dir', 'AP', 'acq' '3p00mm')`. See `bids.query` to see how to specify.

OTHER OPTIONS (with their defaults):

- `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.anatReference.type = 'T1w'` - type of the anatomical reference
- `opt.anatReference.session = ''` - session label of the anatomical reference
- `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.funcVoxelDims = []` - Voxel dimensions to use for resampling of functional data at normalization.
- `opt.stc.skip = false` - boolean flag to skip slice time correction or not.
- `opt.stc.referenceSlice = []` - reference slice 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.glm.roibased.do = false` must be set to `true` to use the `bidsRoiBasedGLM` workflow
- `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.QA.func.carpetPlot = true` to plot carpet plot when running `functionaQA`
- `opt.QA.func` contains a lot of options used by `spmup_first_level_qa` in `functionaQA`
- `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'`
- `opt.QA.glm.do = true` - 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` and Accurate autocorrelation modeling substantially improves fMRI reliability for more info.

(C) Copyright 2019 CPP_SPM developers

3.2 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 defautls 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 script simpler.

(C) Copyright 2019 CPP_SPM developers

3.2.1 auto-correlation modelisation

Use of FAST 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.

Olszowy, W., Aston, J., Rua, C. et al.

Accurate autocorrelation modeling substantially improves fMRI reliability.

Nat Commun 10, 1220 (2019).

https://doi.org/10.1038/s41467-019-09230-w

Note that if you wanted to change this setting you could do so via the `Software` object of the BIDS stats model:

```
{
"Name": "auditory",
"BIDSModelVersion": "1.0.0",
"Description": "contrasts to compute for the FIL MoAE dataset",
"Input": {
    "task": "auditory"
},
"Nodes": [
    {
    "Level": "Run",
    "Name": "run_level",
    "Model": {
        "X": [
        "trial_type.listening"
        ],
        "Type": "glm",
        "Software": {
            "SPM": {
                "SerialCorrelation": "AR(1)",
                "HRFderivatives": "none"
            }
        }
```

(continues on next page)

(continued from previous page)

```
    }
    }
]
}
```

3.3 spm to BIDS filename conversion

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

set default map for renaming for cpp_bids

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`.

(C) Copyright 2021 CPP_SPM developers

CHAPTER

FOUR

DEMOS

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

The demos show show you different way to use CPP_SPM.

4.1 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_preproc**
: This script will download the dataset from the FIL for the block design SPM tutorial and will run the basic preprocessing.

  (C) Copyright 2019 Remi Gau

demos.MoAE.**moae_02_stats**
: This script will run the FFX and contrasts on it of the MoAE dataset

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

  (C) Copyright 2019 Remi Gau

demos.MoAE.**moae_03_create_roi_extract_data**
: This script shows how to create a ROI and extract data from it.

  FYI: 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_04_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

4.2 Face repetition

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

demos.face_repetition.**face_rep_01_preproc_func**
This script will download the face repetition dataset from SPM and will run the basic preprocessing.

(C) Copyright 2019 Remi Gau

demos.face_repetition.**face_rep_02_stats**
This script will run the FFX and contrasts on the the face repetition dataset from SPM.

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

(C) Copyright 2019 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_anat**
This show how an anat only pipeline would look like.

(C) Copyright 2019 Remi Gau

demos.face_repetition.**face_rep_resolution**
This script runs preprocessing with different final spatial resolution in MNI space It then runs the subject level GLMs

This can show how to script several analysis within the CPP_SPM framework

(C) Copyright 2019 Remi Gau

CHAPTER

FIVE

ARCHITECTURE

At the highest levels CPP SPM is organized in workflows: they all start with the prefix *bids* (for example `bidsRealignReslice`) and are in the folder *src.workflows* Workflows typical run on all the subjects specified in the `options` structure (see the *Set up* section).

Workflows run by creating matlab batches that are then passed to SPM to run. To do this they call "bacth creating functions" that all start with the prefix `setBatch` (for example `setBatchCoregistration`). and are in the folder *src.batches*.

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

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

5.1 Workflows

5.1.1 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`) – structure or json filename containing the options. See `checkOptions()` and `loadAndCheckOptions()`.

(C) Copyright 2021 CPP_SPM developers

5.1.2 Other

`src.workflows.`**`bidsCopyInputFolder`**(*opt*, *unzip*)

Copies the folders from the `raw` folder to the `derivatives` folder, and will copy the dataset description and task json files to the derivatives directory.

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])
```

Parameters

- **opt** (structure) – structure or json filename containing the options. See checkOptions() and loadAndCheckOptions().
- **unZip** (boolean) –

(C) Copyright 2019 CPP_SPM developers

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

Renames SPM output into BIDS compatible files.

USAGE:

```
bidsRename(opt)
```

Parameters **opt** (structure) – structure or json filename containing the options. See checkOptions() and loadAndCheckOptions().

See the spm_2_bis submodule and defaults.set_spm_2_bids_defaults for more info.

(C) Copyright 2019 CPP_SPM developers

5.1.3 Worflow helper functions

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

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

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])
```

Parameters

- **opt** (structure) – structure or json filename containing the options. See checkOptions and loadAndCheckOptions.
- **workflowName** (string) – name that will be printed on screen
- **bidsDir** –
- **bidsDir** – string

Returns

- **BIDS** (structure) returned by getData

- **opt** options checked
- **group**

(C) Copyright 2019 CPP_SPM developers

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** (string) – name of the batch
- **opt** (structure) – structure or json filename containing the options. See checkOptions and loadAndCheckOptions.
- **subLabel** (string) – subject label

(C) Copyright 2019 CPP_SPM developers

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

```
cleanUpWorkflow(opt)
```

(C) Copyright 2021 CPP_SPM developers

src.workflows.**returnDependency**(*opt*, *type*)
Use to create dependencies between batches in workflows.

USAGE:

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

(C) Copyright 2021 CPP_SPM developers

5.2 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) – BIDS layout returned by getData.

- **opt** (structure) – structure or json filename containing the options. See checkOptions() and loadAndCheckOptions().
- **subLabel** (string) – subject label

Returns

matlabbatch (structure)

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

- image type = opt.anatReference.type (default = T1w)
- session to select the anat from = opt.anatReference.session (default = 1)

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

(C) Copyright 2020 CPP_SPM developers

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

(C) Copyright 2020 CPP_SPM developers

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 checkOptions().
- **outputDir** (tring) –

Returns

- **matlabbatch** (structure)

(C) Copyright 2019 CPP_SPM developers

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) – BIDS layout returned by getData.
- **opt** (structure) – Options chosen for the analysis. See checkOptions().
- **subLabel** (string) – subject label

Returns

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

(C) Copyright 2020 CPP_SPM developers

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** (string) – name of the output file
- **outDir** (string) – output directory
- **expression** (string) – 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**

(C) Copyright 2020 CPP_SPM developers

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** (string) – 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

(C) Copyright 2020 CPP_SPM developers

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

Creates a batch for the lesion overlap map

USAGE:

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

Parameters **matlabbatch** (structure) – list of SPM batches

Returns

- **matlabbatch** (structure)

(C) Copyright 2021 CPP_SPM developers

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

Creates a batch to segment the anatomical image for lesion detection

USAGE:

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

Parameters **matlabbatch** (structure) – list of SPM batches

Returns

- **matlabbatch** (structure)

(C) Copyright 2021 CPP_SPM developers

src.batches.lesion.**setBatchLesionAbnormalitiesDetection**(*matlabbatch*, *opt*, *images*)
Creates a batch to detect lesion abnormalities

USAGE:

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

Parameters **matlabbatch** (structure) – list of SPM batches

Returns

- **matlabbatch** (structure)

(C) Copyright 2021 CPP_SPM developers

5.3 Low level functions description

5.3.1 Utility functions

src.utils.**cleanCrash**()
Removes any files left over from a previous unfinished run of the pipeline, like any *.png imgages

USAGE:

```
cleanCrash()
```

(C) Copyright 2020 CPP_SPM developers

src.utils.**saveOptions**(*opt*)
Short description of what the function does goes here.

USAGE:

```
saveOptions(opt)
```

Parameters **opt** (structure) – Options chosen for the analysis. See checkOptions().

(C) Copyright 2020 CPP_SPM developers

src.utils.**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** (string) – 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()`.

(C) Copyright 2020 CPP_SPM developers

`src.utils.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`) – returned by bids.layout when exploring a BIDS data set.
- **subLabel** (`string`) – 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`) – Used to find the task name and to pass extra `query` options.
- **info** (`string`) – `sessions`, `runs`, `filename`.
- **sessionLabel** (`string`) – session label (for *ses-001*, the label will be *001*)
- **runIdx** (`string`) – run index label (for *run-001*, the label will be *001*)
- **suffix** (`string`) – datatype (`bold`, `events`, `physio`)

(C) Copyright 2020 CPP_SPM developers

`src.utils.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 `checkOptions()`.
- **bidsDir** (`string`) – the directory where the data is ; default is : `fullfile(opt.dataDir, '..', 'derivatives', 'cpp_spm')`

Returns

- **opt** (structure)
- **BIDS** (structure)

(C) Copyright 2020 CPP_SPM developers

src.utils.**setDirectories**(*opt*)

USAGE:

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

(C) Copyright 2021 CPP_SPM developers

src.utils.**createDataDictionary**(*tsvContent*)

USAGE:

```
jsonContent = createDataDictionary(tsvContent)
```

(C) Copyright 2020 CPP_SPM developers

src.utils.**createDerivativeDir**(*opt*)

Creates the derivative folder if it does not exist.

USAGE:

```
opt = createDerivativeDir(opt)
```

Parameters **opt** (structure) – Options chosen for the analysis. See checkOptions().

(C) Copyright 2019 CPP_SPM developers

src.utils.**createGlmDirName**(*opt*)

USAGE:

```
glmDirName = createGlmDirName(opt)
```

(C) Copyright 2021 CPP_SPM developers

src.utils.**getAnatFilename**(*BIDS*, *opt*, *subLabel*)

Get the filename and the directory of an anat file for a given session and run. Unzips the file if necessary.

It several images are available it will take the first one it finds.

USAGE:

```
[anatImage, anatDataDir] = getAnatFilename(BIDS, subLabel, opt)
```

Parameters

- **BIDS** (structure) –
- **subLabel** –
- **subLabel** – string
- **opt** – structure

Returns

- **anatImage** (string)

- **anatDataDir** (string)

(C) Copyright 2020 CPP_SPM developers

src.utils.**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) – returned by bids.layout when exploring a BIDS data set.
- **subLabel** (string) – 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** (string) – session label (for *ses-001*, the label will be *001*)
- **runID** (string) – run index label (for *run-001*, the label will be *001*)
- **opt** (structure) – Mostly used to find the task name.

Returns

- **boldFilename** (string)
- **subFuncDataDir** (string)

(C) Copyright 2020 CPP_SPM developers

src.utils.**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) –
- **subLabel** (string) –
- **opt** (structure) – Options chosen for the analysis. See checkOptions().

Returns

- **meanImage** (string)
- **meanFuncDir** (string)

(C) Copyright 2020 CPP_SPM developers

src.utils.**getTpmFilename**(*BIDS*, *subLabel*, *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) –
- **opt** –
- **opt** – structure
- **subLabel** –
- **subLabel** – string
- **space** –
- **space** – string
- **res** –
- **res** – string

Returns

- **gm** (string) grey matter TPM
- **wm** (string) white matter TPM
- **csf** (string) csf matter TPM

(C) Copyright 2021 CPP_SPM developers

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 `checkOptions()`.
- **subFuncDataDir** –
- **fileName** –

Returns

- **voxDim**
- **opt**

(C) Copyright 2020 CPP_SPM developers

src.utils.**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) – output of getData or bids.layout
- **opt** (structure) – Options chosen for the analysis. See checkOptions().
- **opt** – filter for bids.query.

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

(C) Copyright 2020 CPP_SPM developers

src.utils.**getSubjectList**(*BIDS*, *opt*)

Returns the subjects to analyze in opt.subjects

USAGE:

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

Parameters

- **BIDS** (structure) – output of bids.layout
- **opt** (structure) – Options chosen for the analysis. See checkOptions().

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`.

(C) Copyright 2021 CPP_SPM developers

src.utils.**renameSegmentParameter**(*BIDS*, *subLabel*, *opt*)

USAGE:

```
renameSegmentParameter(BIDS, subLabel, opt)
```

(C) Copyright 2020 CPP_SPM developers

src.utils.**renameUnwarpParameter**(*BIDS*, *subLabel*, *opt*)

USAGE:

```
renameUnwarpParameter(BIDS, subLabel, opt)
```

(C) Copyright 2020 CPP_SPM developers

src.utils.**rmTrialTypeStr**(*conName*)

(C) Copyright 2019 CPP_SPM developers

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** (string) –
- **overwrite** (boolean) –

Returns

- **structure** (structure)

(C) Copyright 2020 CPP_SPM developers

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

Also save some basic environnment info.

USAGE:

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

Parameters

- **matlabbatch** (structure) –
- **batchType** (string) –
- **opt** (structure) – Options chosen for the analysis. See `checkOptions()`.
- **subLabel** (string) –

(C) Copyright 2019 CPP_SPM developers

src.utils.**unzipAndReturnsFullpathName**(*fullpathName*, *opt*)

Unzips a file if necessary

USAGE:

```
unzippedFullpathName = unzipAndReturnsFullpathName(fullpathName)
```

Parameters **fullpathName** (string array) –

Returns

- **unzippedFullpathName** (string)

(C) Copyright 2020 CPP_SPM developers

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** (string) – Directory where the search will be conducted.
- **fileName** (string) – file name pattern. Can be a regular expression except for the starting `^` and ending `$`. For example: `'sub-.*_ses-.*_task-.*_bold.nii'`.
- **prefix** (string) – 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]');

(C) Copyright 2019 CPP_SPM developers

5.3.2 Print and error handling

src.messages.**printToScreen**(*msg*, *opt*)

USAGE:

```
printToScreen(msg, opt)
```

(C) Copyright 2021 CPP_SPM developers

src.messages.**errorHandling**(*varargin*)

USAGE:

```
errorHandling(functionName, id, msg, tolerant, verbose)
```

Parameters

- **functionName** (string) –
- **id** (string) – Error or warning id
- **msg** (string) – 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

(C) Copyright 2018 CPP_SPM developers

src.messages.**printWorkflowName**(*workflowName*, *opt*)

(C) Copyright 2019 CPP_SPM developers

src.messages.**printBatchName**(*batchName*, *opt*)

(C) Copyright 2019 CPP_SPM developers

src.messages.**printCredits**(*opt*)

TODO use the .CFF to load contributors

(C) Copyright 2019 CPP_SPM developers

src.messages.**printProcessingRun**(*groupName*, *iSub*, *subLabel*, *iSes*, *iRun*, *opt*)

(C) Copyright 2019 CPP_SPM developers

src.messages.**printProcessingSubject**(*iSub*, *subLabel*, *opt*)

(C) Copyright 2019 CPP_SPM developers

5.3.3 Infrastructure related functions

src.infra.**checkDependencies**(*opt*)
Checks that that the right dependencies are installeda and loads the spm defaults.

USAGE:

```
checkDependencies()
```

(C) Copyright 2019 CPP_SPM developers

src.infra.**checkToolbox**(*toolboxName*)
Checks that that the right dependencies are installeda and loads the spm defaults.

USAGE:

```
checkToolbox()
```

(C) Copyright 2021 CPP_SPM developers

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)

(C) Copyright 2020 CPP_SPM developers

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)

(C) Copyright 2020 CPP_SPM developers

src.infra.**isOctave**()
Returns true if the environment is Octave.

USAGE:

```
retval = isOctave()
```

Returns

retval (boolean)

(C) Copyright 2020 Agah Karakuzu (C) Copyright 2020 CPP_SPM developers

src.infra.**setGraphicWindow**(*opt*)

Short description of what the function does goes here.

USAGE:

```
[interactiveWindow, graphWindow, cmdLine] = setGraphicWindow(opt)
```

Parameters **opt** (structure) –

Returns

- **interactiveWindow**
- **graphWindow**
- **cmdLine** (boolean)

(C) Copyright 2019 CPP_SPM developers

CHAPTER

SIX

PREPROCESSING

6.1 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.

6.1.1 Slice Time Correction

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

Performs the slice timing correction of the functional data.

USAGE:

```
bidsSTC(opt)
```

Parameters **`opt`** (`structure`) – structure or json filename containing the options. See `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.

(C) Copyright 2019 CPP_SPM developers

More info available on this page of the SPM wikibook.

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

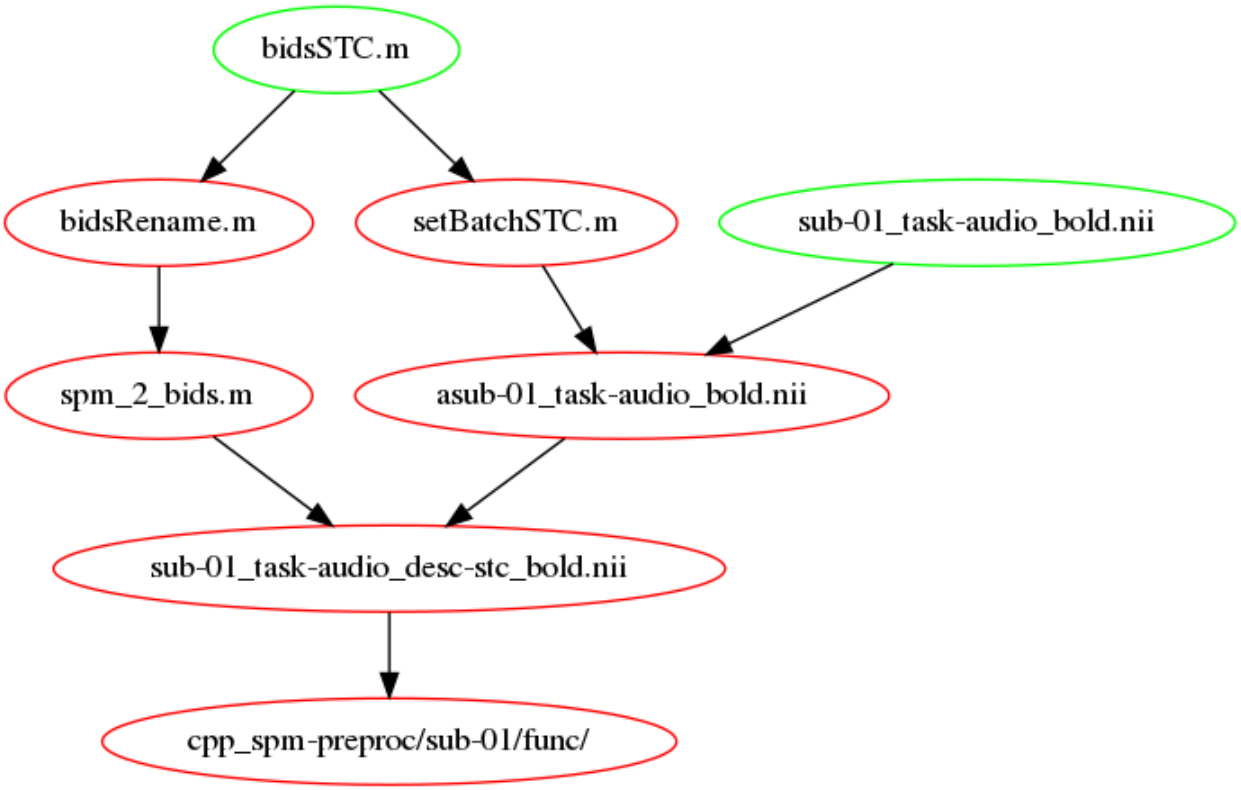


Fig. 1: Slice timing correction workflow

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.

6.1.2 Spatial Preprocessing

Perform spatial preprocessing by running `bidsSpatialPrepro`

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

Performs spatial preprocessing of the functional and anatomical data.

USAGE:

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

Parameters **opt** (structure) – structure or json filename containing the options. See `checkOptions()` and `loadAndCheckOptions()`.

The anatomical data are segmented, skulls-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`.

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`.

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.anatReference`:

```
opt.anatReference.type = 'T1w';
opt.anatReference.session = 1;
```

(C) Copyright 2019 CPP_SPM developers

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

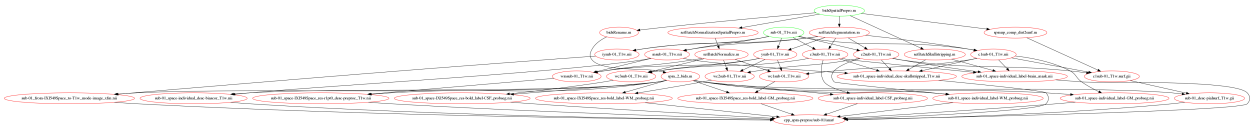


Fig. 2: Anatomical component of the spatial preprocessing workflow

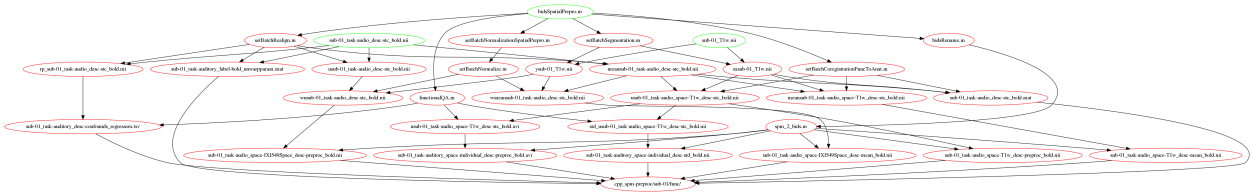


Fig. 3: Functional component of the spatial preprocessing workflow

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

Realigns and reslices the functional data of a given task.

USAGE:

```
bidsRealignReslice(opt)
```

Parameters **opt** (structure) – structure or json filename containing the options. See `checkOptions()` and `loadAndCheckOptions()`.

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

(C) Copyright 2020 CPP_SPM developers

src.workflows.preproc.**bidsRealignUnwarp**(*opt*)
Realigns and unwarps the functional data of a given task.

USAGE:

```
bidsRealignReslice(opt)
```

Parameters **opt** (structure) – structure or json filename containing the options. See 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.

(C) Copyright 2020 CPP_SPM developers

6.1.3 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) – structure or json filename containing the options. See checkOptions() and loadAndCheckOptions().

(C) Copyright 2020 CPP_SPM developers

6.1.4 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) – structure or json filename containing the options. See 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.

(C) Copyright 2020 CPP_SPM developers

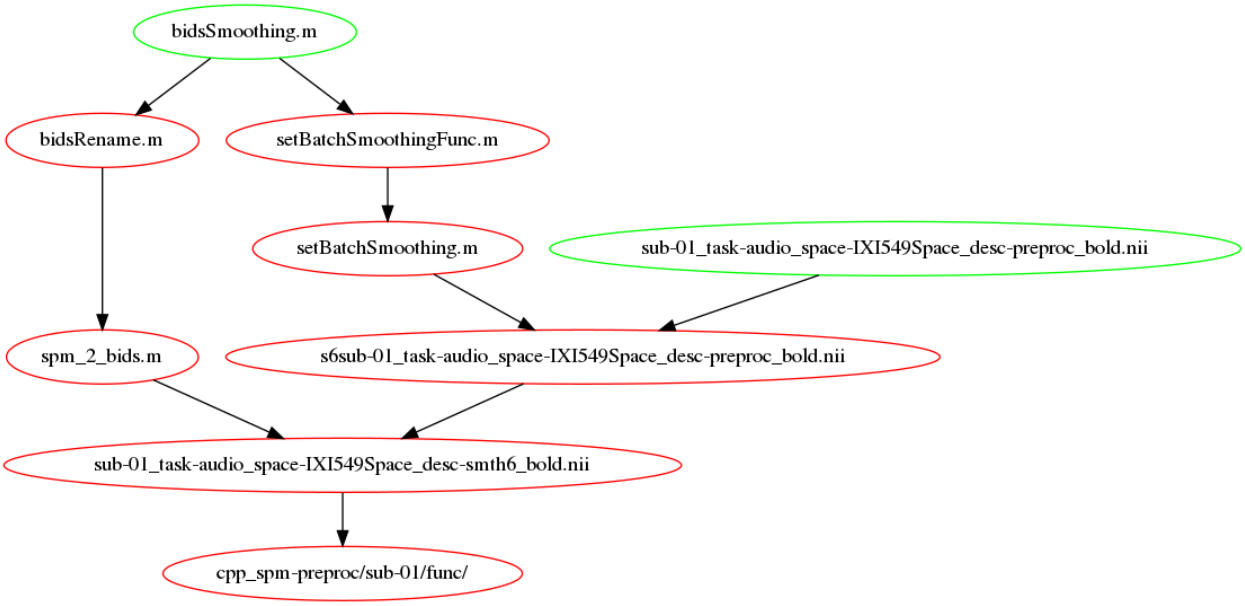


Fig. 4: Smoothing workflow

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) – structure or json filename containing the options. See `checkOptions()` and `loadAndCheckOptions()`.

(C) Copyright 2020 CPP_SPM developers

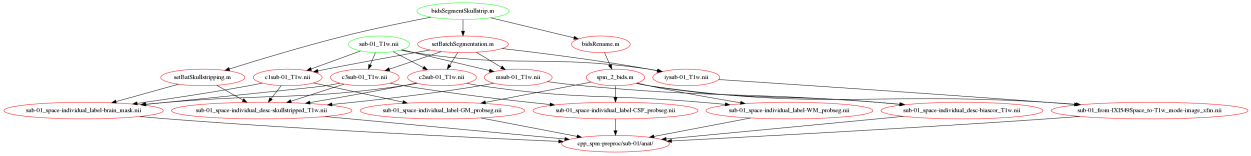


Fig. 5: Segment and skullstrip workflow

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

Create segmented-skull stripped mean functional image

(C) Copyright 2020 CPP_SPM developers

6.2 Preprocessing batches

6.2.1 Slice Time Correction

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

Creates batch for slice timing correction

USAGE:

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

Parameters

- **BIDS** (structure) – BIDS layout returned by getData.
- **opt** (structure) – structure or json filename containing the options. See checkOptions() and loadAndCheckOptions().
- **subLabel** (string) – 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 this function will take the mid-volume time point as reference to do the slice timing correction.

(C) Copyright 2019 CPP_SPM developers

6.2.2 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) – BIDS layout returned by getData.
- **opt** (structure) – Options chosen for the analysis. See checkOptions().
- **subLabel** (string) – subject label
- **action** (string) – realign, realignReslice, realignUnwarp, 'reslice'

Returns

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

(C) Copyright 2020 CPP_SPM developers

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, referenceImg, sourceImages, interp = 4)
```

Parameters

- **matlabbatch** (structure) – list of SPM batches
- **referenceImg** (string or cellstring) – Reference image
- **sourceImages** (string or cellstring) – Source images

Returns

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

(C) Copyright 2020 CPP_SPM developers

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) – structure or json filename containing the options. See checkOptions() and loadAndCheckOptions().

Returns

matlabbatch (structure)

(C) Copyright 2020 CPP_SPM developers

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) – BIDS layout returned by getData.

- **opt** (structure) – structure or json filename containing the options. See checkOptions() and loadAndCheckOptions().
- **subLabel** (string) – 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.

(C) Copyright 2020 CPP_SPM developers

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** –
- **voxDim** –
- **imgToResample** –
- **imgToResample** –

Returns

- **matlabbatch** (structure)

(C) Copyright 2019 CPP_SPM developers

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) –

- **voxDim** –

Returns

- **matlabbatch** (structure)

(C) Copyright 2019 CPP_SPM developers

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) – BIDS layout returned by getData.
- **opt** (structure) – structure or json filename containing the options. See checkOptions() and loadAndCheckOptions().
- **subLabel** (string) – subject label

Returns

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

(C) Copyright 2020 CPP_SPM developers

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

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

USAGE:

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

Parameters

- **matlabbatch** (structure) – list of SPM batches
- **opt** (cell string) – Other images to apply the coregistration to
- **ref** (string) – Reference image
- **src** (string) – Source image
- **other** (cell string) – Other images to apply the coregistration to

Returns

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

(C) Copyright 2020 CPP_SPM developers

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) – BIDS layout returned by getData.
- **opt** (Options chosen for the analysis. See checkOptions().) –
- **subLabel** (string) –

Returns

- **matlabbatch**

(C) Copyright 2020 CPP_SPM developers

6.2.3 Smoothing

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

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

USAGE:

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

Parameters

- **matlabbatch** –
- **group** –
- **opt** – Options chosen for the analysis. See checkOptions().

Returns

- **matlabbatch**

(C) Copyright 2019 CPP_SPM developers

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

Short description of what the function does goes here.

USAGE:

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

Parameters

- **matlabbatch** (structure) –
- **BIDS** (structure) – BIDS layout returned by getData.
- **opt** (string) – Options chosen for the analysis. See checkOptions().
- **subLabel** –

Returns

- **matlabbatch** (structure)

(C) Copyright 2019 CPP_SPM developers

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

Small wrapper to create smoothing batch

USAGE:

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

Parameters

- **matlabbatch** (structure) –
- **images** –
- **fwhm** –
- **prefix** –

Returns

- **matlabbatch** (structure)

(C) Copyright 2019 CPP_SPM developers

CHAPTER

SEVEN

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`) – structure or json filename containing the options. See `checkOptions()` and `loadAndCheckOptions()`.

Inspired from spmup `spmup_BIDS_preprocess` (@ commit 198c980d6d7520b1a99) (URL missing)

(C) Copyright 2020 CPP_SPM developers

`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`) – BIDS layout returned by `getData`.

- **opt** (structure) – structure or json filename containing the options. See checkOptions() and loadAndCheckOptions().
- **subLabel** (string) –

Returns

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

TODO implement for 'phase12', 'fieldmap', 'epi'

(C) Copyright 2020 CPP_SPM developers

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) – BIDS layout returned by getData.
- **opt** (structure) – structure or json filename containing the options. See checkOptions() and loadAndCheckOptions().
- **subLabel** (string) – subject label

Returns

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

TODO implement for 'phase12', 'fieldmap', 'epi'

(C) Copyright 2020 CPP_SPM developers

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** –
- **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)

(C) Copyright 2020 CPP_SPM developers

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

(C) Copyright 2020 CPP_SPM developers

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) – BIDS layout returned by `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

(C) Copyright 2020 CPP_SPM developers

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`)

(C) Copyright 2020 CPP_SPM developers

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) –
- **opt** (string) – Options chosen for the analysis. See checkOptions().
- **boldFilename** –

Returns

- **vdmFile** (string)

CHAPTER

EIGHT

STATISTICS

8.1 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.

8.1.1 Subject level

`src.workflows.stats.`**`bidsFFX`**(*action*, *opt*)

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

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)
```

Parameters

- **`action`** (`string`) – Action to be conducted:`specifyAndEstimate` or `contrasts`.
- **`opt`** (`structure`) – structure or json filename containing the options. See `checkOptions()` and `loadAndCheckOptions()`.

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

See also: setBatchSubjectLevelGLMSpec, setBatchSubjectLevelContrasts

(C) Copyright 2020 CPP_SPM developers

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:

```
cpp_spm-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*, *deleteIndBeta*, *deleteIndTmaps*)

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

USAGE:

```
concatBetaImgTmaps(opt, [deleteIndBeta = true,] [deleteIndTmaps = true])
```

Parameters

- **opt** (structure) – options structure
- **deleteIndBeta** ((boolean)) – decide to delete beta-maps
- **deleteIndTmaps** ((boolean)) – decide to delete t-maps

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.

(C) Copyright 2019 CPP_SPM developers

8.1.2 Group level

src.workflows.stats.**bidsRFX**(*action*, *opt*)

- 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)
```

Parameters

- **action** (string) – Action to be conducted: 'smoothContrasts' or 'RFX' or 'meanAnatAndMask'
- **opt** (structure) – structure or json filename containing the options. See checkOptions() and loadAndCheckOptions().

(C) Copyright 2020 CPP_SPM developers

8.1.3 Compute results

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

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)
```

Parameters **opt** (structure) – structure or json filename containing the options. See `checkOptions()` and `loadAndCheckOptions()`.

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.result.Nodes` 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.result.Nodes(1) = returnDefaultResultsStructure();
```

Specify the Node level type (run, subject or dataset):

```
opt.result.Nodes(1).Level = 'subject';
```

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.result.Nodes(1).Contrasts(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.result.Nodes(1).Contrasts(2).Name = 'listening_lt_baseline';
opt.result.Nodes(1).Contrasts(2).MC =  'none';
opt.result.Nodes(1).Contrasts(2).p = 0.01;
opt.result.Nodes(1).Contrasts(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.result.Nodes(1).Output.png = true();

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

% thresholded statistical map
opt.result.Nodes(1).Output.thresh_spm = true();

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

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

```
opt.result.Nodes(1).Output.montage.do = true();

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

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

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

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.result.Nodes(1).Output.NIDM_results = true();
```

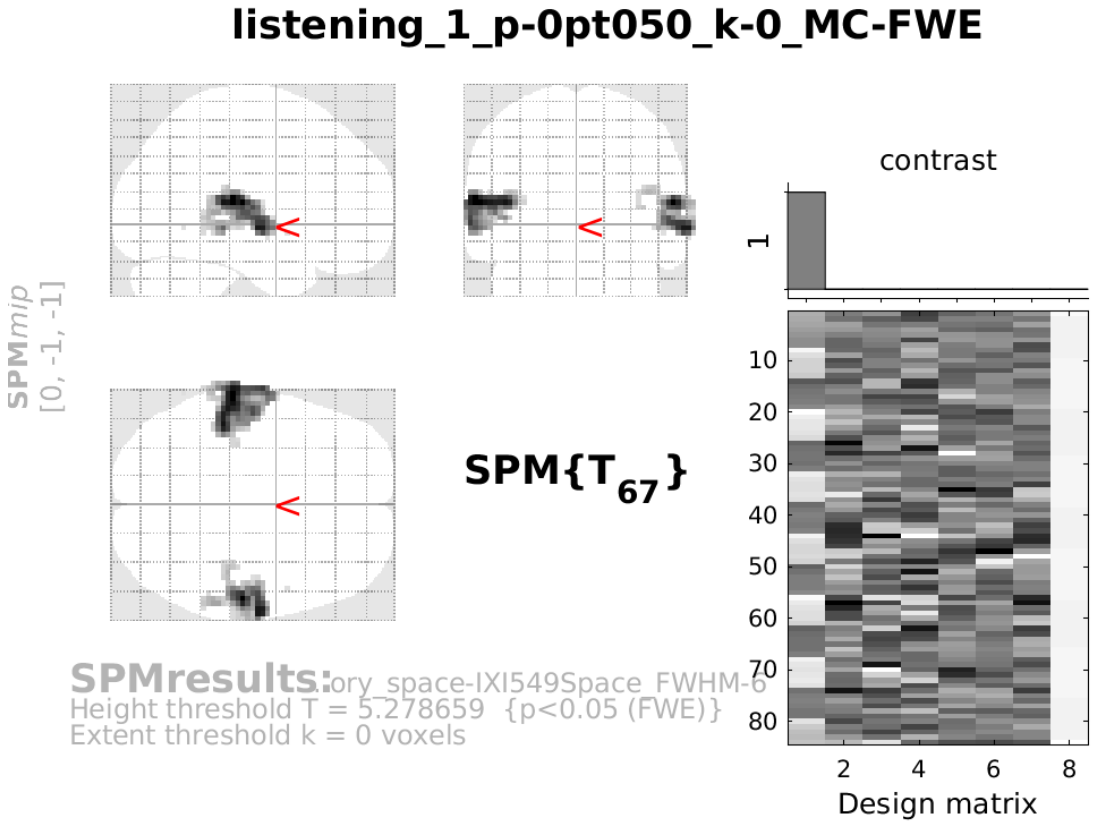
(C) Copyright 2020 CPP_SPM developers

`CSV output example`

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

- beta values
- t values
- cluster boundaries
- ROI boundaries

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



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

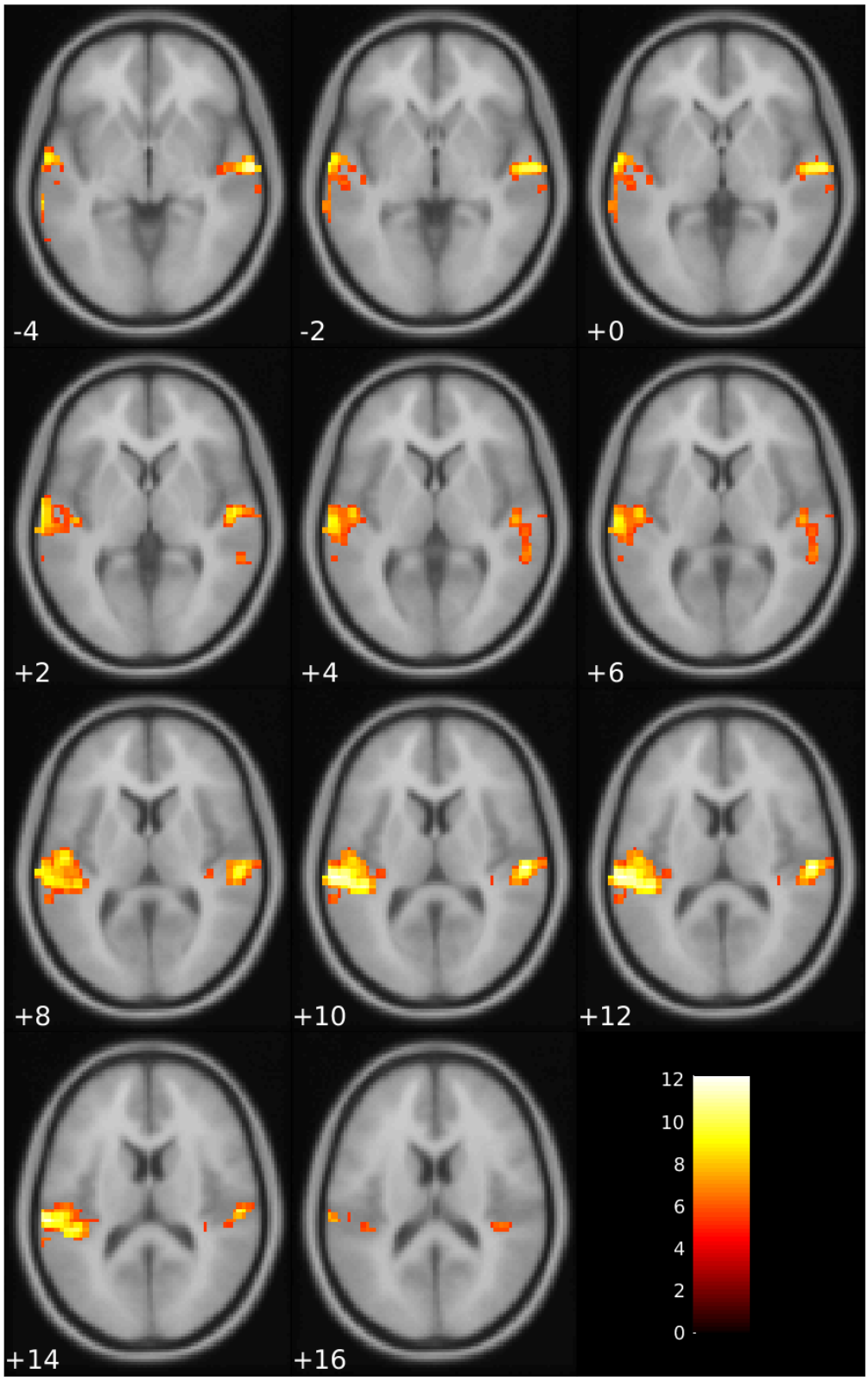


Fig. 3: Example of subject level montage from the MoAE demo

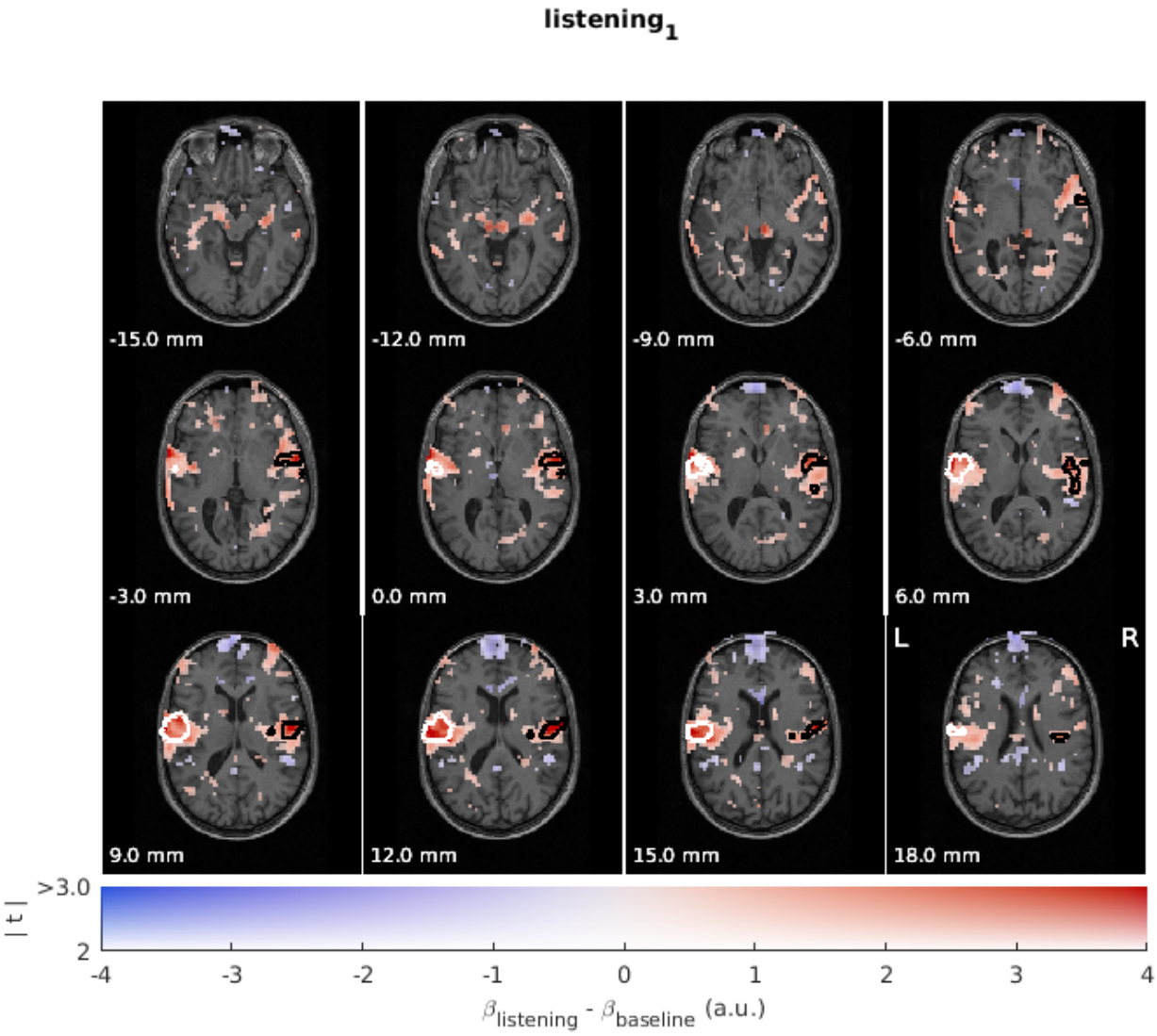


Fig. 4: Example of subject level slice display from the MoAE demo

8.1.4 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) – structure or json filename containing the options. See 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', 'cpp_spm-stats');

bidsCreateROI(opt);
```

(C) Copyright 2021 CPP_SPM developers

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

Will run a GLM within a ROI using MarsBar.

USAGE:

```
bidsRoiBasedGLM(opt)
```

Parameters **opt** (structure) – structure or json filename containing the options. See checkOptions() and loadAndCheckOptions().

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

(C) Copyright 2021 CPP_SPM developers

8.2 Statistics batches

8.2.1 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) –
- **opt** (structure) –
- **subLabel** (string) –

Returns

- **matlabbatch** (structure)

(C) Copyright 2019 CPP_SPM developers

src.batches.stats.**setBatchEstimateModel**(*matlabbatch*, *opt*, *grpLvlCon*)
Short description of what the function does goes here.

USAGE:

```
matlabbatch = setBatchEstimateModel(matlabbatch, grpLvlCon)
```

Parameters

- **matlabbatch** (structure) –
- **grpLvlCon** –

Returns

- **matlabbatch** (structure)

(C) Copyright 2019 CPP_SPM developers

8.2.2 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) –
- **spmMatFile** (string) –
- **consess** (cell) –

Returns

- **matlabbatch** (structure)

(C) Copyright 2019 CPP_SPM developers

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

Short description of what the function does goes here.

USAGE:

```
matlabbatch = setBatchFactorialDesign(matlabbatch, opt)
```

Parameters

- **matlabbatch** (structure) –
- **opt** (structure) –

Returns

- **matlabbatch** (structure)

(C) Copyright 2019 CPP_SPM developers

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

Short description of what the function does goes here.

USAGE:

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

Parameters

- **matlabbatch** (structure) –
- **opt** (structure) –
- **subLabel** (string) –

Returns

- **matlabbatch**

(C) Copyright 2019 CPP_SPM developers

src.batches.stats.**setBatchGroupLevelContrasts**(*matlabbatch*, *opt*, *grpLvlCon*, *rfxDir*)

(C) Copyright 2019 CPP_SPM developers

8.2.3 Compute results

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

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

USAGE:

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

Parameters

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

Returns

- **matlabbatch** (structure)

See also: setBatchSubjectLevelResults, setBatchGroupLevelResults

(C) Copyright 2019 CPP_SPM developers

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

Short description of what the function does goes here.

USAGE:

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

Parameters

- **matlabbatch** (structure) –
- **opt** (structure) –
- **subLabel** (string) –

Returns

- **matlabbatch** (structure)

See also: bidsResults, setBatchResults

(C) Copyright 2019 CPP_SPM developers

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

USAGE:

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

Parameters

- **matlabbatch** (structure) –
- **opt** (structure) –
- **result** (structure) –

Returns

- **matlabbatch** (structure)

(C) Copyright 2019 CPP_SPM developers

8.3 Statistics functions

8.3.1 Subject level

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

Creates an `_onset.mat` in the subject level GLM folder.

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, funcFWHM)
```

Parameters

- **opt** (structure) –
- **subLabel** (string) –
- **tsvFile** (string) – fullpath name of the tsv file.

Returns

onsetFilename (string) fullpath name of the file created.

See also: convertOnsetTsvToMat

(C) Copyright 2019 CPP_SPM developers

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

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

USAGE:

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

Parameters

- **subLabel** (string) –
- **opt** –
- **opt** – structure

Returns

- **ffxDir** (string)

(C) Copyright 2019 CPP_SPM developers

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) –
- **opt** (structure) –
- **subLabel** (string) –
- **iSes** (integer) –
- **iRun** (integer) –

Returns

- **boldFilename** (string)

(C) Copyright 2020 CPP_SPM developers

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

USAGE:

```
deleteResidualImages(ffxDir)
```

Parameters **ffxDir** (string) –

(C) Copyright 2020 CPP_SPM developers

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

Specifies the first level contrasts

USAGE:

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

Parameters

- **SPM** (structure) – content of SPM.mat
- **opt** (structure) –

Returns

- **contrasts** (type) (dimension)

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

(C) Copyright 2019 CPP_SPM developers

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.

The function extracts from the events.tsv file the trials (with type, onsets, and durations) of the conditions of interest as requested in the model.json. It then stores them in the GLM folder in a .mat file that can be fed directly in an SPM GLM batch.

USAGE:

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

Parameters

- **opt** (structure) –
- **tsvFile** (string) –

Returns

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

(C) Copyright 2019 CPP_SPM developers

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

(C) Copyright 2019 CPP_SPM developers

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

Creates an `_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) –
- **tsvFile** (string) – fullpath name of the tsv file.

Returns

counfoundMatFile (string) fullpath name of the file created.

See also: setBatchSubjectLevelGLMSpec, createConfounds

(C) Copyright 2019 CPP_SPM developers

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

Gets the `_confounds.tsv` for a given subject, session, run

USAGE:

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

Parameters

- **BIDS** (structure) – returned by bids.layout when exploring a BIDS data set.
- **subLabel** (string) – 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) – Mostly used to find the task name.

Returns

- **filename** (string)

(C) Copyright 2021 CPP_SPM developers

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) – returned by bids.layout when exploring a BIDS data set.
- **subLabel** (string) – 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) – Mostly used to find the task name.

Returns

- **realignParamFile** (string)

(C) Copyright 2020 CPP_SPM developers

8.3.2 Group level model

src.group_level.**getRFXdir**(*opt*)

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

USAGE:

```
rfxDir = getRFXdir(opt)
```

Parameters **opt** (structure) – Options chosen for the analysis. See checkOptions().

Returns

rfxDir (string) Fullpath of the group level directory

(C) Copyright 2019 CPP_SPM developers

8.3.3 Compute results

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

Short description of what the function does goes here.

USAGE:

```
matlabbatch = setBatchSubjectLevelResults(matlabbatch, opt, subLabel, funcFWHM,␣
↪iNode, iCon)
```

Parameters

- **opt** (structure) –
- **result** (structure) –

Returns

- **outputNameStructure** (structure)

See also: setBatchSubjectLevelResults, bidsResults

(C) Copyright 2021 CPP_SPM developers

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

USAGE:

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

(C) Copyright 2019 CPP_SPM developers

CHAPTER

NINE

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 `checkOptions()`.

(C) Copyright 2020 CPP_SPM developers

`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';
```

(continues on next page)

(continued from previous page)

```
opt.QA.func.Globals = 'on';
opt.QA.func.Movie = 'on';
opt.QA.func.Voltera = 'on';
opt.QA.func.carpetPlot = true;
```

Parameters **opt** (structure) – Options chosen for the analysis. See `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.

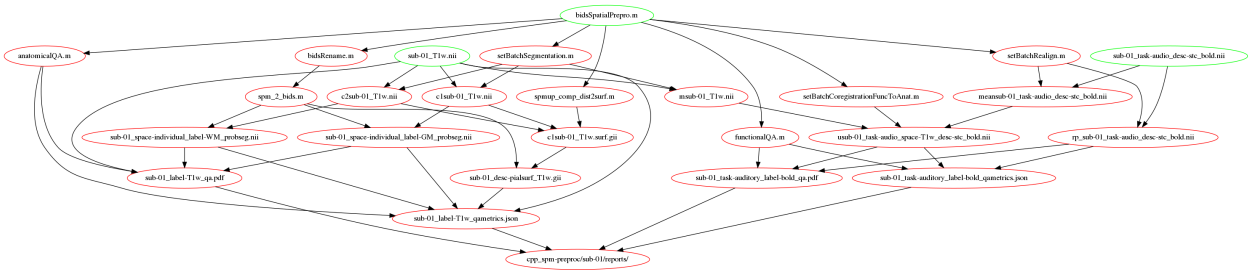


Fig. 1: workflows for QA as part of the spatial preprocessing workflow

`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** (string) – Path to a bids _events.tsv file.
- **opt** (structure) – Options chosen for the analysis with the content below.

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');
```

(continues on next page)

(continued from previous page)

```
bids.util.tsvwrite('events.tsv', tsv);

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** (string) – 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

TEN

FREQUENTLY ASKED QUESTIONS

10.1 Results

10.1.1 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.result.Nodes(1).Output.montage.slices = -12:4:60;

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

CHAPTER

ELEVEN

BOILERPLATE METHODS SECTION

11.1 Preprocessing

The fMRI data were pre-processed and analyzed using statistical parametric mapping (SPM12 - {v7487}; Wellcome Center for Neuroimaging, London, UK; www.fil.ion.ucl.ac.uk/spm) running on {octave 4.{??} / matlab 20{XX} (Mathworks)}.

The preprocessing of the functional images was performed in the following order: removing of dummy scans, {slice timing correction}, realignment, normalization to MNI, smoothing.

{XX} dummy scans were removed to allow signal stabilization.

{Slice timing correction was then performed taking the {XX} th slice as a reference (interpolation: sinc interpolation).}

Functional scans from each participant were realigned using the mean image as a reference (SPM 2 passes ; number of degrees of freedom: 6 ; cost function: least square) (Friston et al, 1995).

The mean image obtained from realignement was then co-registered to the anatomical T1 image (number of degrees of freedom: 6 ; cost function: normalized mutual information) (Friston et al, 1995). The transformation matrix from this coregistration was then applied to all the functional images.

The anatomical T1 image was bias field corrected, segmented and normalized to MNI space (target space resolution: 1 mm ; interpolation: 4th degree b-spline) using a unified segmentation. The deformation field obtained from this step was then applied to all the functional images (target space resolution equal that used at acquisition ; interpolation: 4th degree b-spline)

Functional MNI normalized images were then spatially smoothed using a 3D gaussian kernel (FWHM = {XX} mm).

11.2 fMRI data 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 FAST model to account for temporal auto-correlation (Corbin et al, 2018) 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 will have mean intracerebral intensity of 100.

We modeled the fMRI experiment in an event related design with regressors entered into the run-specific design matrix after convolving the onsets of each event with a canonical hemodynamic response function (HRF).

Nuisance covariates included the 6 realignment parameters to account for residual motion artefacts.

Contrast images were computed for the following condition and spatially smoothed using a 3D gaussian kernel (FWHM = {XX} mm).

11.3 References

Friston KJ, Ashburner J, Frith CD, Poline J-B, Heather JD & Frackowiak RSJ (1995) Spatial registration and normalization of images Hum. Brain Map. 2:165-189

Corbin, N., Todd, N., Friston, K. J. & Callaghan, M. F. Accurate modeling of temporal correlations in rapidly sampled fMRI time series. Hum. Brain Mapp. 39, 3884-3897 (2018).

—

Use the report function to get a print out of the content of a dataset.

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

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

USAGE:

```
reportBIDS(opt)
```

Parameters **`opt`** (`structure`) – Options chosen for the analysis. See `checkOptions()`.

CHAPTER

TWELVE

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()`).

(C) Copyright 2004-2009 JH (C) Copyright 2009_2012 DSS (C) Copyright 2012_2019 Remi Gau (C) Copyright 2020 CPP BIDS SPM-pipeline developers

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

THIRTEEN

DOCKER

13.1 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/cpp_spm:stable
```

This will create an image with the tag name `cpp_spm:stable`

Running `make docker_img` will also build the `stable` version and a `latest` version.

13.2 Run docker image

The following command would pull from our docker hub and start the docker image:

```
docker run -it --rm cpplab/cpp_spm: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/cpp_spm:latest
```

For example, you could run the demos by doing this:

```
code_folder=/home/remi/github/CPP_SPM/demos/MoAE

docker run -it --rm \
    -v $code_folder:/code \
    cpplab/cpp_spm:latest

# once inside the docker image
moae_01_preproc
```

CHAPTER

FOURTEEN

LINKS AND REFERENCES

14.1 SPM starters

If you start from zero, go through the 2 first tutorials of SPM:

https://www.fil.ion.ucl.ac.uk/spm/data/

14.2 Andrew Jahn videos and blogs

http://andysbrainblog.blogspot.com/search/label/marsbar

http://andysbrainblog.blogspot.com/search/label/SPM

http://andysbrainblog.blogspot.com/search/label/SPM.mat

https://www.youtube.com/watch?v=qbcBLXJhzZg&list=PLIQIswOrUH689KpRPCa5-h6U-m9CddWM6

14.3 SPM code snippets

SPM wikibook has some very useful sections: https://en.wikibooks.org/wiki/SPM

From John Ashburner on Tom Nichols blog: https://blogs.warwick.ac.uk/nichols/tag/johns-gems/

From Rik Henson: http://www.mrc-cbu.cam.ac.uk/people/rik.henson/personal/analysis/

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: https://github.com/Remi-Gau/advanced_fMRI_course/blob/master/Practical%231/practical_1.m

HRF, convolution and GLM ("by hand"): https://github.com/Remi-Gau/advanced_fMRI_course/blob/master/Practical%232/practical_2.m

Design efficiency: https://github.com/Remi-Gau/advanced_fMRI_course/blob/master/Practical%233/practical_3.m

14.4 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

14.4.1 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`

14.4.2 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

14.4.3 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)

14.4.4 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.

14.4.5 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

14.4.6 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.

14.4.7 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).

14.4.8 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

14.4.9 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

14.4.10 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`

14.4.11 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)

14.4.12 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

14.4.13 info on variance of the error

- `SPM.VResMS` - file struct of ResMS image handle
  - `SPM.VResMS.fname` - variance of error file name

14.4.14 info on mask

- `PM.VM` - file struct of Mask image handle
  - `PM.VM.fname` - name of mask nii file

14.4.15 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

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

CHAPTER

FIFTEEN

FEATURES

15.1 Preprocessing

If your data is fairly "typical" (for example whole brain coverage functonal 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 can't be handled well by fmriprep then CPP_SPM has some automated workflows to perform amongst other things:

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

All preprocessed outputs are saved as BIDS derivatives with BIDS compliant filenames.

15.2 Statistics

The model specification are done via 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

15.3 Quality control

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

It can also prepare the data to run an MVPA analysis by running a GLM for each subject on non-normalized images and get one beta image for each condition to be used in the MVPA.

CHAPTER

SIXTEEN

ASSUMPTIONS

At the moment this pipeline makes some assumptions:

- it assumes that the dummy scans have been removed from the BIDS data set and it can jump straight into pre-processing,

CHAPTER

SEVENTEEN

INDICES AND TABLES

- genindex
- modindex
- search

MATLAB MODULE INDEX

INDEX

I

L

M

P

R

S

U

V