pyMRI USER GUIDE

2022

Instructions, rationale and example scripts to perform subject and group level processing with pyMRI package.

Rev. 0.3, 16/05/2022.

Table of Contents

PYMRI: USER GUIDE

Overview

This guide is intended to explain all the processing steps necessary to perform subject and group level processing with pyMRI, a set of python classes and utilities interacting with either FSL and SPM software. The following analysis are currently implemented: VBM and Cortical Thickness, TBSS and tractography, melodic and seed-based functional connectivity (SBFC) analyses.

The first step will convert and store subjects' images data and perform the first pre-processing at the subject level. It will perform a set of operations thought to ease the further processing, like scalping, segmentation, main co-registrations among different sequences, single subject melodic and confound signal removal from epi data for SBFC, DTI model fitting, bedpostX and xTract individual tractography.
The last three sections of this guide will instead focus on the three techniques (melodic, sbfc and tracto), explaining all the steps to perform single-subject and group-level and the final statistical analyses

Naming conventions

Original images files and all those created by pyMRI processing will be named according to rigid and fixed rules. This important assumption, which involves the creation of standardized folder and file naming scheme, will allow the present coding architecture to automatically process data through a set of automatized and highly parameterized scripts. Each deviation from the standard naming scheme will prevent the correct functioning of the analyses scripts. Project and subject fixed file system schema will be extensively explained in the following paragraphs. To avoid file indexing issues, each subject identification name (SUBJLABEL) must not contain neither any special characters (+-*/!"£$%&/ ...etc) nor the space " ". It may contain only the underscore ("_").

Code architecture: python framework and old bash framework (bashMRI)

The present python framework originally derived from the porting of a previous one, written in bash language (bashMRI), and then underwent several improvements. Some processing is still to be ported, and are available only in the old framework, which is fully compatible with the new one, since they share the same files and folders naming. In both frameworks two kind of script exist: framework and project scripts. The former are basically a set of classes and convenient scripts that implement the required processing, they are predetermined and *users* **do not have** to modify them. Users must create and edit *project scripts* according to the desired analysis.
The folder pyMRI_FOLDER/examples/ contains an example of all the project scripts used for each implemented analysis.

Multi-threaded scripting

Some methods can be invoked in a multi-threaded manner. In that case, in the calling project script you can define the number of CPU to be used, providing the list of multiple cases (subjects, folders, glm files) to the framework that will automatically implement the multi-threading process.

Interaction with other tools

pyMRI is thought to interplay mainly with FSL and SPM software. It does it in two radically different ways. With FSL (and all the other linux executable) it directly runs its executables file, each performing a different processing, by means of a set of convenient functions (the most important is called *rrun*). Interaction with SPM is more complex. It is based on the *"matlab engine for Python"* a set of python functions, provided by Matlab, which allow Python to start several matlab engines, call their functions and access their workspaces. Besides few specific matlab functions, called directly through the framework, most of the interactions with SPM is done by instructing pyMRI to edit a batch file (with most of its parts set by default and some other edited according to the calling methods) and run it. Batch files is the standard method used by SPM to run complex and long processes. They are normally edited with a proper SPM user interface. pyMRI uses instead predefined static templates with some dynamic parts (e.g. subjects label and folder, the value of some parameters) that are overwritten during the requested processing. More details will follow.

Basic coding concepts

There are few coding (software developing) concepts that pyMRI user must understand to properly use it.

Variable: is an entity that store values, having a specific type.
Type: can be: numbers, string, lists and dictionaries (and many others).

- numbers can be integer (*int*, e.g. 2) or decimals (*float*, e.g. 2.21).
- *string* is a list of characters (e.g. "Hello").
- *list* is a list of elements (e.g. mylist=["a", "b", …., "z"]).
    - I can read a list's values by its index in the list (starting from zero)
        - val = mylist[0] => "a".
        - val = mylist[1] => "b".
    - I can modify an existing element with: mylist[2] = "c"
    - I can add a new element at the end of the list with: mylist.append("gigi")
- Dictionary is a python type, a sort of list in which element are indexed by name and not my number. mydict = {"name1":val1, "name2":val2}
    - I can read an element with: val = mydict["name1"] => val1
    - write in the opposite way: mydict["name1"] = val1

Functions
Functions is an entity that performs a set of predefined commands.

```
function hello(){  print("ciao") }      hello()       => ciao
```

they can be customizable accepting parameters that alter its functioning

```
function hello(text){  print(text) }   hello("ciao")   => ciao,  hello("addio")  => addio
```

they can also return a value to the calling code

```
function sum(a,b){  return a+b } c = sum(1,2) => c = 3
```

Class
Another key type in coding is the Class. A Class is an entity that contain variables (called properties) and functions (called methods) and typically contains and manipulate logically-related things. For example, in pyMRI there will be a class managing each subject, by implementing all the subject-level processing, one managing the GroupAnalysis stuffs, and many others.
You use a class by creating an *instance* of that class. Assuming the existence of a class called Subject which need as first parameter the name of that subject. You can create two instances:

```
subj_gigi = Subject("gigi")
subj_pino = Subject("pino")
```

They contain different information (variables), e.g. the path of the T1 file, but can perform the same operations. That is, across many instances of a same class, properties values are different but have the same methods. In pyMRI, user will always deal with instances and will call its methods.
In the guide, when describing a method, it will be used the notation *ClassName.methodname*. User will have to first create an instance of class *ClassName* and then call: instancename. *methodname(params)*

Installation

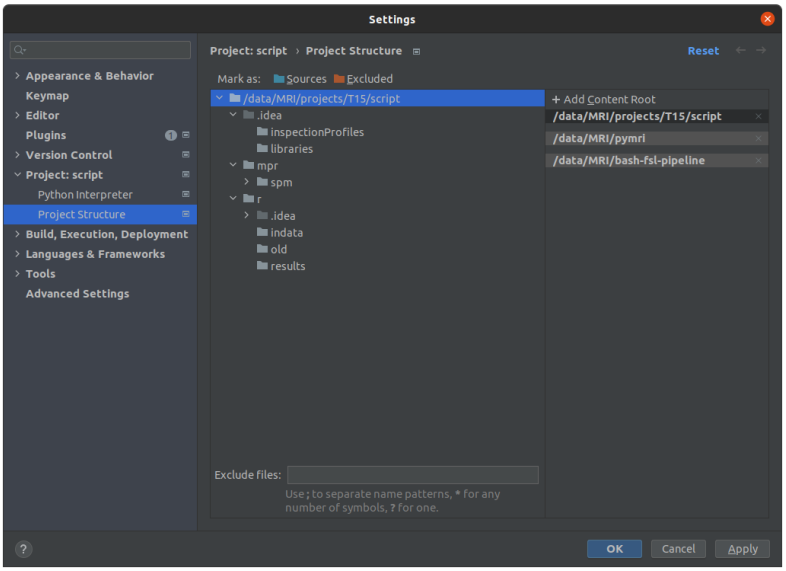
The suggested IDE to use pyMRI is PYCHARM and all the following instructions will focus on it. The suggested python version is 3.8 but user must also install the 2.7 version, in order to use the ICA-AROMA package.
This is the list of the needed packages:
csv, gzip, matlab engine, matplotlib, ntpath, numpy, re, shutil, traceback

"Matlab engine for Python" can be installed following these instructions:
https://www.mathworks.com/help/matlab/matlab_external/install-the-matlab-engine-for-python.html

Each MRI project compatible with pyMRI has the following folders:

Project A
- script
- subjects
- group_analysis

When you want to setup a project you must open, in pycharm, its *script* folder.
Then press **File -> Settings -> Project: script -> project Structure**
Here press **Add Content Root** and select both the pyMRI folder and the bashMRI framework (called bash-fsl-pipeline)



ANALYSIS TOOLS

Project script header

All project scripts must start with this specific header.

```
import os
from Global import Global
from Project import Project

if __name__ == "__main__":

    fsl_code            = "604"
    try:
        globaldata = Global(fsl_code)

    except Exception as e:
        print(e)
        exit()

    # ======================================================================
    # HEADER
    # ======================================================================
    proj_dir    = "/data/MRI/projects/past_controls"
    project     = Project(proj_dir, globaldata)
    SESS_ID     = 1
    num_cpu     = 19
    group_label = "test"
    subjects    = project.load_subjects(group_label, SESS_ID)
```

This header defines a set of global parameters (globaldata) and creates an instance of the Project class (indicating the project folder) which store all the information related to that specific project. Using a method of the *Project* class (*load_subjects*), it loads a list of subjects belonging to it into the variable *subjects,* which is a *list* of *Subject* instances.

Description of the 4 parameters

proj_dir:
User must edit in order to make it point to the correct project folder

SESS_ID:
This regulates which subject session manage. pyMRI can in fact manage also longitudinal studies where more than one recording sessions is present for each subject.
Subject are stored in the project folder *subjects,* each session will be stored in a subfolder called s1, s2, s3 etc…
This value actually could be omitted, since all methods that have session as parameter, set it to 1 by default.

num_cpu:
most of the subject's methods can be run in parallel. With this parameters user can decide how many methods run. This values should never be higher than the PC available cores. Actually, the proper value

must also consider the available RAM. Some processing can use up to 2 GB of RAM, which should never be completely used. Thus user must choose a number below the available cores and that not deplete the free RAM.

group_label:
with this variable user can select the list of subjects to be processed.

Subjects lists

One of the most important topic in pyMRI is managing subjects lists. In a project with many subjects, it often happens to have many subgroups that are processed separately. In order to manage this, user must create and edit a file called *subjects_lists.json* located in this path:
PROJ_DIR/script/subjects_lists.json
Is a text file, following the JSON format.

```
{
"subjects":[
              {"label":"single"  , "list":[“subj1”]},
              {"label":"group_1" , “list":["s1", "s2", …., "sn"]},
              {"label":"group_2" , “list":["s6", "s10", …., "sy"]},
              ……
            ]
}
```

subjects_lists has only one field called *“subjects”* whose value is a list of items. These items represent a subjects lists and contain the fields “label” (to name it and retrieve it) and “list”. The latter is a list of string, each defining the name of a subject belonging to such subjects list. When, during the analysis, the user wants to perform an analysis over a specific subset of subjects, it creates a new list and use it. By running this code

```
subjects    = project.load_subjects(group_label, SESS_ID)
```

the project class looks into all the subjects’ list defined in the json file and retrieve the list of subjects *instances* there defined. The group label must be defined in the json file, otherwise an error is raised.
Several methods calls in pyMRI accept a parameters that can either be the label of one the subjects list defined in *subjects_lists.json* or directly a list of subjects. These methods also return valid lists, that is they check whether all the subjects in such list are actually present in the file system.
Examples are:

Project. get_subjects_labels(grouplabel_or_subjlist, sess_id) => list of valid subjects’ labels
Project. get_subjects(self, group_or_subjlabels, sess_id=1) => list of valid subjects’ instances

But also those regarding group analysis
GroupAnalysis.tbss_run_fa(grouplabel_or_subjlist, …..)

Which can be called by either passing the group label of a subjects list or directly a list of subjects’ labels or instances.

Subject Processing

Once user load a subjects' list into the workspace, it can get the reference of each of them by name:

```
subj = project.get_subject_by_label("subjname1")
```

or by position within the loaded list

```
subj = subjects[2]
```

and then can call whichever *Subject*'s method.

```
subj.rename("new_subjname1")
subj.dti.xtract_viewer()
```

Multi-threading processing

Most of the Subject methods can be run in parallel. Since they work only on the given subject, within-subjects parallel processing does not interfere one with each other.
User can do it calling the Project's method *run_subjects_methods.*

```
run_subjects_methods(method_type, method_name, kwparams, ncore=1, subj_labels=None)
```

method_type and method_name

They define the subject method to be run in parallel.
method_type can be one of the following "", "mpr", "epi", "dti", "transform" and define whether the method is defined in Subject class ("") or in one of its four main properties.

For example, the Subject method *rename*, which is defined in Subject, is called in this way:
run_subjects_methods("", "rename"….)
The method *eddy_correct,* defined in the class SubjectDti, can be called in this way:

```
run_subjects_methods("dti", "eddy_corrent", ….)
```

kwparams

The *kwparams* parameter is of type list and allows to specify the parameters of the method. Those parameters are specified as a python dictionary, e.g. {"param1":value1, "param2":value2, …}.
Two cases can be distinguished: a) when parameters are the same for each subject or b) when they differ between them. In the first case, the kwparams contain only one element, the dictionary defining all the common parameters. In the second case, the kwparams must contain one dictionary for each subject processed.
Let's see some example.
Assuming user want to reslice all sagittal images to axial orientation. The *reslice_image* method has a parameter called *direction*. All methods need the same value, so I can simply define the kwparams as a list with just one dictionary *{"direction":"sag->axial"}*

```
project.run_subjects_methods("", "reslice_image", [{"direction":"sag->axial"}],
ncore=num_cpu)
```

The rename function has the following signature: *rename(new_label, session_id=1)* and requires instead a different value for each subject. Thus, I create an empty list and I fill it with a dictionary for each subject.

```
kwparams = []
for subj in subjects:
       kwparams.append({"new_label":subj.label + "_prefix"}
project.run_subjects_methods("", "rename", kwparams, ncore=num_cpu)
```

ncore

This parameter defines the number of cores (and thus subject's methods) to be run in parallel. Note that operations involving the GPU cannot be run in parallel, since GPU is optimized to run only one process at time, and thus ncore must be set to 1.

subj_labels

This parameter can be used to define the list of subjects to process. In case no value is specified (default behavior), it uses the list of presently loaded subjects. Otherwise user must input a list of string explicating all the subjects to be processed.

```
wellcome(self, do_anat=True, odn="anat", imgtype=1, smooth=10,
biascorr_type=BIAS_TYPE_STRONG,
do_reorient=True, do_crop=True,
do_bet=True, betfparam=[0.5],
do_sienax=False, bet_sienax_param_string="-SNB -f 0.2",
do_reg=True, do_nonlinreg=True,
do_seg=True, do_spm_seg=True, spm_seg_over_bet=False, spm_seg_over_fs=False,
do_cleanup=True, do_strongcleanup=False, do_overwrite=False,
use_lesionmask=False, lesionmask="",
do_freesurfer=False,
do_first=False, first_struct="", first_odn="",
do_epirm2vol=0, do_aroma=True, do_nuisance=True, hpfsec=100, feat_preproc_odn="resting",
feat_preproc_model="singlesubj_feat_preproc", do_featinitreg=False,
do_melodic=True, mel_odn="resting", mel_preproc_model="singlesubj_melodic",
do_melinitreg=False,
do_dtifit=True, do_bedx=True, do_bedx_cuda=False, bedpost_odn="bedpostx",
do_autoptx_tract=False,
do_struct_conn=False, struct_conn_atlas_path="freesurfer", struct_conn_atlas_nroi=0,
std_image=""):
for p in range(len(subjects)):
       kwparams.append({"do_spm_seg":False, "do_cat_seg":False, "do_epirm2vol":146,
       "do_dtifit":True,"feat_preproc_model":"singlesubj_feat_preproc_noreg_melodic"})
project.run_subjects_methods("", "wellcome", kwparams, project.get_subjects_labels(),
nthread=num_cpu)
```

Standard subjects pipeline

It will be now described the standard procedure to import a subject in an existing project with pyMRI.

Obtain dicom from server

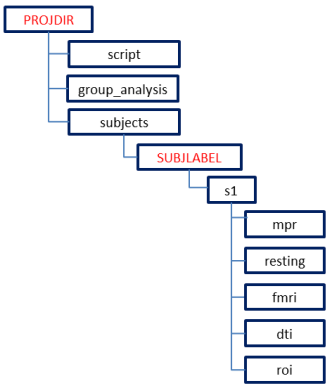
In San Martino Hospital, research images are stored in an internal dicom server that can be reached at the following web address: http://10.187.186.69:3333/main
Once logged in, user can search for subject name, see all his recording sessions and download each of them as separate zip files. Each zip file contains the hundreds/thousands of 2D dicom images composing each sequence. When downloading a session zip, please save it as the final subject identification name (SUBJLABEL). P.s. remember that this label should not contain any special character, only the "_" is allowed.

Create subject filesystem

Each project has a "subjects" folder containing one subfolder for each subject. The first step to perform is to create such subject folder and all its subfolders.

```
project.run_subjects_methods("", "create_file_system", [],
ncore=num_cpu)
```

MAIN PROJECT AND SUBJECT FOLDERS



It first creates a folder for each experimental session. This is thought to manage longitudinal studies where one than more sessions creates a folder for each type of recorded sequence. *mpr* contains the anatomical t1, *fmri* can contain more than one session, *dti* the diffusion sequence and *resting* the epi sequence at rest. In addition, it creates a folder called *roi* whom content will be explained later on.

The scanner writes sequences' images as a set of 2D dicom images contained in zip file. FSL and SPM deals instead with 3D nifty images, original images thus need to be extracted from the zip and converted in nifty. To this aim, pyMRI uses the console program dcm2niix, which takes a folder, recognize to which sequence each 2D dicom image belongs to and creates the corresponding 3D nifty image.

unzipping scanner output

```
for subj in subjects:
     kwparams.append({"src_zip":"/data/MRI/OT/dicom/past_bp/" + subj.label + ".zip",
                      "dest_dir":"/data/MRI/OT/dicom/past_bp/"+ subj.label})
project.run_subjects_methods("", "unzip_data", kwparams, nthread=num_cpu)
```

User must specify where the zip file is located (src_zip parameter of the dictionary object) and where it must create the folder with all the dicom images (dest_dir).

File conversion and renaming

As previously stated, pyMRI needs that each file has specific names and is located in specific folder, whilst the name of the converted image corresponds to the sequence name that was defined in the scanner's protocol. pyMRI thus, after data conversion, have to rename and move the created nifty image toward its final path. To perform these two operations, convert and rename, the *renameNifti* method exists.

```
renameNifti(extpath, associations, options="-z o -f %f_%p_%t_%s_%d ", cleanup=0,
convert=True, rename=True):
```

extpath contains the folder where all the dicom file are located
associations defines a list of rules to find specific sequences and rename as requested. Each rule is represented as a python dictionary with the following fields:

- contains when a sequence's name contains the substring here specified, it apply the conversion
- postfix new name will be: subject label + postfix

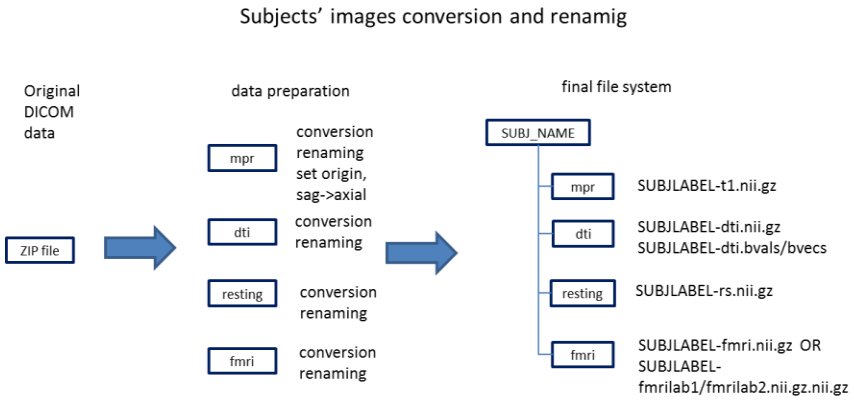
- type specify which sequence is

By default, *renameNifti* performs both the operations. Nevertheless, it can just do one of the two. For example, when importing for the very first time a subject from a new protocol and user doesn't know how each sequence will be called, it is useful to call it with the parameter *rename=False*. After the conversion, user takes notes of the name of each sequence and properly fill in the associations list in order to correctly rename each file of interest.
Assuming that the mpr image name contains the substring FSPGR, the dti the substring TENSOR and the resting state epi the substring "Resting state", the code to properly convert and rename imported sequence would be:

```
associations    = []
associations.append({"contains": "FSPGR", "postfix":"-t1", "type": Subject.TYPE_T1})
associations.append({"contains": "TENSOR", "postfix":"-dti", "type": Subject.TYPE_DTI})
associations.append({"contains": "Resting_state", "postfix":"-rs", "type": Subject.TYPE_RS})

for subj in in range(len(subjects)):
      kwparams.append({"extpath": "/data/MRI/OT/dicom/past_bp/" + subjects[p].label,
                       "associations": associations, "cleanup": 0, "rename":True})
project.run_subjects_methods("", "renameNifti", kwparams, nthread=num_cpu)
```

this code would create the following files:

```
PROJDIR\subjects\SUBJLABEL\s1\mpr\SUBJLABEL-t1.nii.gz
PROJDIR\subjects\SUBJLABEL\s1\dti\SUBJLABEL-dti.nii.gz
PROJDIR\subjects\SUBJLABEL\s1\dti\SUBJLABEL-dti.bval
PROJDIR\subjects\SUBJLABEL\s1\dti\SUBJLABEL-dti.bvec
PROJDIR\subjects\SUBJLABEL\s1\resting\SUBJLABEL-rs.nii.gz
```



Reslicing to axial

It may happen that, particularly anatomical images, are scanned in sagittal orientation. Since MNI templates is in axial orientation. It is advisable to reslice images to the axial orientation. User can do it with:

```
project.run_subjects_methods("", "reslice_image", [{"direction":"sag->axial"}],
ncore=num_cpu)
```

check images

Particularly when converting dozens of subjects, it is useful to verify whether all subjects have all the expected sequences. With the following code, user can check if the sequence specified with True are available. In case not, it outputs in the command line the name of the subject and his missing sequences.

```
project.run_subjects_methods("", "check_images", [{"t1":True, "rs":True, "dti":True,
"t2":False, "fmri":None}], ncore=num_cpu)
```

Note that *fmri* is a list of string or None (= do not check), not a boolean variable. Since there could be more than one session, each named in a different way, user can specify their names. For example, assuming fmri sequences are called SUBJLABEL-seq1 and SUBJLABEL-seq2, user could check their presence with the following

```
project.run_subjects_methods("","check_images",[{"fmri":["-seq1","-seq2"]}], ncore=num_cpu)
```

set origin

SPM wants user to specify the position of the anterior commissure (AC) on each t1 image. This helps SPM internal algorithms to better segment and normalize the t1. AC can only be manually set in the proper SPM user interface. pyMRI helps user to do it with a three-steps procedure:

1) user can call the method *prepare_mpr_for_setorigin1(group_label, sess_id=1, replaceOrig=False, overwrite=False),* which unzips the SUBJLABEL-t1.nii.gz (SPM does not work with compressed images) and rename it as SUBJLABEL-t1_temp.nii. The method by default creates a backup copy of the original image, but this can be omitted (replaceOrig=True). If a the _temp.nii is already present, the method returns without doing anything, unless user asks (overwrite=True) to re-uncompress and overwrite the existing _temp.nii.

```
project.prepare_mpr_for_setorigin1(group_label)
```

2) User then open SPM (matlab -> spm fmri), load the SUBJLABEL-t1_temp.nii image (press “display images”), find the AC, press “Set Origin” to set it, then press ….. and select the same file SUBJLABEL-t1_temp.nii.
3) then user call

```
project.prepare_mpr_for_setorigin2(group_label)
```

which compresses SUBJLABEL-t1_temp.nii to SUBJLABEL-t1.nii.gz

welcome

Now images are ready to be processed with the method *wellcome*. This method launches all the available individual processing for each four (mpr, dti, resting, fmri) sequence. It has more than 50 parameters, all with a default value.

```
Wellcome:

do_anat=True, odn="anat", imgtype=1, smooth=10,
biascorr_type=SubjectMpr.BIAS_TYPE_STRONG,
do_reorient=True, do_crop=True,
do_bet=True, betfparam=[0.5],
do_sienax=False, bet_sienax_param_string="-SNB -f 0.2",
do_reg=True, do_nonlinreg=True, do_seg=True,
do_spm_seg=False, spm_seg_templ="", spm_seg_over_bet=False,
do_cat_seg=False, cat_seg_over_bet=False, cat_use_dartel=False, do_cat_surf=True,
```

```
do_cat_seg_long=False, cat_long_sessions=[1],
do_cleanup=True, do_strongcleanup=False, do_overwrite=False,
use_lesionmask=False, lesionmask="lesionmask",
do_freesurfer=False, do_complete_fs=False,
do_first=False, first_struct="", first_odn="",
do_epirm2vol=0, do_susc_corr=False, do_aroma=True, do_nuisance=True, hpfsec=100,
feat_preproc_odn="resting", feat_preproc_model="singlesubj_feat_preproc_noreg_melodic",
do_featinitreg=False,
do_melodic=True, mel_odn="postmel", mel_preproc_model="singlesubj_melodic_noreg",
do_melinitreg=False,
replace_std_filtfun=False,
do_dtifit=True, do_bedx=False, do_bedx_gpu=False, bedpost_odn="bedpostx",
do_xtract=False, xtract_odn="xtract", xtract_refspace="native", xtract_gpu=False,
xtract_meas="vol,prob,length,FA,MD,L1,L23",
do_struct_conn=False, struct_conn_atlas_path="freesurfer", struct_conn_atlas_nroi=0):
```

Here comes the list of all the implemented processing, divided by the sequence involved and with a brief description of the parameters used to activate such processing the determine its characteristics.

MPR processing

The mpr pipeline derives from the standard fsl one. It involves several preprocessing to be run in parallel. The most important is the scalping step, where scalp and skull are individuated and an image containing only the beneath tissues (white and grey matter and CSF), called SUBJLABEL-t1_brain.nii.gz, is created.

The importance of brain scalping

This step may fail, resulting in a "_brain" image that contains part of the skull or where parts of the gray matter are missing. In this case, all the following preprocessing step will be invalid as the subject shall not be correctly normalized. A properly normalized t1 is important not only for anatomical processing, but also for all the other group analysis. DTI and EPI (resting state and fmri) individual images are in fact co-registered to the standard template (necessary for group analysis) by means of the t1->standard transformation. The scalping has to be repeated with other tools (spm, cat or freesurfer), as later specified, until a valid "_brain" image is created.

Anatomical processing can be omitted setting *do_anat=False*. Outputs are saved in *odn="anat"* folder. It can be changed in case user want to test a second kind of pipeline. The algorithm skips the already completed steps or can repeat them again, overwriting the existing images (*do_overwrite=True*). At the end of the preprocessing, user can manage how deeply remove intermediate files (*do_cleanup=True, do_strongcleanup=False*). It is done on T1 images, but could be hypothetically run also on t2 (imgtype=2). Some smoothing operations use the default value of 10 mm that should not be changed (*smooth=10*)

Prebet

This step prepares t1 for scalping by running the following:

- fixing negative values
- do_reorient=True: reorientation to standard
- *do_crop=True* : cropping
- *use_lesionmask=False, lesionmask=lesionmask*: reorient and crop the given lesion mask
- biascorr_type=SubjectMpr.BIAS_TYPE_STRONG: perform bias field correction

Bet

The default schema is *do_reg=True, do_bet=True and do_nonlinreg=True* which performs the following:

- co-register the individual T1 to the standard MNI head (not brain)
- calculate the inverse transformation
- apply such inverse transformation to the standard MNI brain mask
- fill the zeros of this mask
- apply this mask to the individual T1 => individual SUBJLABEL-t1_brain.nii.gz

In case a linear registration is done, brain extraction is performed in a classical way using bet and user may specify more than one erosion values (*betfparam=[0.5]*) and compare their outputs.

SPM segmentation

User can perform standard segmentation with SPM (*do_spm_seg=True*). It uses a precalculated tissues probability map (TPM.nii) image that can be changed (spm_seg_templ=/../…/customTPM.nii) whether another, still normalized but more similar to the investigated cohort, is available. By default, it also outputs images ready for DARTEL template creation (rc* images). Its output is not deterministic, in the created images, one for each image components (white matter, grey matter, csf, skull, scalp, outer regions), voxels' values represent the probability (from 0 to 1) that that voxel belongs to that image. Accordingly, SPM does not create a "_brain" image. pyMRI creates it by summing gray and white images, removing values below 0.1 and filling internal hole and then by summing the liquor image.

```
rrun("fslmaths " + c1img + " -add " + c2img + " -thr 0.1 -fillh " + brain_mask, logFile=log)
rrun("fslmaths " + c1img + " -add " + c2img + " -add " + c3img + " -thr 0.1 -bin " +
skullstripped_mask, logFile=log)
```

The output image is only seldom acceptable as a good scalped image. Nevertheless, user can decide to use this version to replace the "_brain" image created by the pipeline (*spm_seg_over_bet=True*).
NOTE: if user want to run VBM with the dartel method, it must select this step.

CAT segmentation

User can perform standard segmentation with CAT12, an SPM toolbox (do_cat_seg *=True*). It is similar to SPM method, with some, according to authors, some improvements. In pyMRI it is mostly used to perform surface analysis (*do_cat_surf=True*), necessary for Cortical Thickness evaluation. CAT by default use an own template (the shooting template), but can also use the dartel one (*cat_use_dartel=True*).
CAT segmentation can also be run in a longitudinal fashion (*do_cat_seg_long=True*), when each subject has more than one session recorded in different timepoints. Sessions can be specified with the variable *cat_long_sessions=[1,2,3]*.

freesurfer

Free surfer is a very famous software, historically the gold standard for cortical thickness, before CAT12 started to become popular. It has a huge t1 processing pipeline concerning both volume and surface processing. A full subject analysis may last up to 24 hours. pyMRI selected the FSL processing T1 pipeline as its standard one, but freesurfer brain mask creation is indeed very precise and represents the favourite "plan B" when FSL brain extraction fails.

Freesurfer can be run with *do_freesurfer=True* and user can select whether running the full pipeline (*do_complete_fs=True*) or only the part necessary to create a brainmask (*do_complete_fs=False*)

Postbet

In this step the following process are done:

- Fsl segmentation with fast software
- Brain volume estimation
- Clean up

Finalize

Some final file renaming and moving is done

Sienax

SIENA is a package for both single-time-point ("cross-sectional") and two-time-point ("longitudinal") analysis of brain change, in particular, the estimation of atrophy (volumetric loss of brain tissue). Can be invoked with *do_sienax=True*, internally uses the bet software, whom parameter can be changed altering: *bet_sienax_param_string="-SNB -f 0.2"*.

First

FIRST is a model-based segmentation/registration tool suited for subcortical structures, run when *do_first=True.* User can select to which mpr subfolder save its results (by changing *first_odn*) and which subcortical structures segment by specifying their comma separated list in *first_struct.*
This is the list of accepted structures:
L_Accu,L_Amyg,L_Caud,L_Hipp,L_Pall,L_Puta,L_Thal,R_Accu,R_Amyg,R_Caud,R_Hipp,R_Pall,R_Puta,R_Thal,BrStem

transform_mpr

finally, the last step of T1 processing is to calculate its transformations from and to the standard MNI templates, both at 2mm and 4mm resolution. This means calculating linear and non-linear:
hr2std, std2hr, hr2std4, std42hr

Resting State processing

The resting state processing implemented in pyMRI is aimed at cleaning and postprocess data and then run the standard FSL pipeline: run a group melodic analysis and then the dual regression to perform within networks analysis. Before real processing, some steps must be completed. At the end of these steps there will be two main outputs:

1) The cleaned sequence in individual space:

PROJDIR/subjects/sX/SUBJLABEL/resting/SUBJ/….. _preproc_aroma_nuisance_melodic

2) If *replace_std_filtfun=True*, the cleaned sequence in standard space, together with other files, will be located in a folder called:

PROJDIR/subjects/sX/SUBJLABEL/resting/reg_std

The former file will be used by seed-based functional connectivity (SBFC) analysis, the latter folder will be used for group melodic and dual-regression analysis.

Remove first volumes

Historically, the first volumes of an EPI session have been considered affected by minor artefacts induced by incomplete magnet stabilization. The solution is to remove them. User can do this by setting the number of final requested volumes. Assuming user recorded 200 volumes, by setting *do_epirm2vol=196,* the algorithms removes the first four.

Susceptibility correction

For high field scanners (≥ 3T), it is advisable to record also a few volumes sequence with a phase encoding direction inverted with respect to original sequence. The PE direction, in fact, induces artefacts along such direction that can be corrected once a sequence with the opposite direction is given to a couple of specific algorithms called topup/applytopup. User must set *do_susc_corr=True* and be sure to insert into the same folder a sequence called:
PROJDIR/subjects/sX/SUBJLABEL/resting/SUBJLABEL-rs_PA.nii.gz.
Now resting state image is ready to be processed

Pre-processing

Echo planar imaging (EPI) preprocessing is run with the FSL's command line instruction *feat*, that reads a properly edited fsf configuration file. Each project must have an own file that, by default, must be called as:
feat_preproc_model="singlesubj_feat_preproc_noreg_melodic"
and is located in the *PROJDIR/script/glm/templates folder.* The script opens this file replace some of its part with subject information and runs it. Saving outputs in a sub-folder of *PROJDIR/subjects/sX/SUBJLABEL/resting,* named as: *feat_preproc_odn="resting"*.
It executes the following steps:

- High pass filtering
- motion correction (mcflirt)
- spatial smoothing
- melodic exploration.

The latter si done to have an idea of which artefacts are present in the original data and verify whether following AROMA/FIX analysis could remove them. Registration toward the standard space is here omitted. pyMRI opted for a centralized between-sequences co-registration. These steps are done once and transformation matrices (linear registration) and warps (non-linear registration) are stored each in a specific subfolder of SUBJDIR/roi.

Between sequences transformation

This script performs all linear and non-linear registration between resting EPI, T1, standard and standard 4mm. Registration between resting and t1, presently do NOT use the fsl suggested BBR method. Reasons have to be inquired (TODO). This step is done now, since its results are needed by ICA-AROMA.

ICA-Aroma (also Fix in the future)

ICA-AROMA is a data-driven method to identify and remove motion-related ICA components from FMRI data (https://github.com/maartenmennes/ICA-AROMA). It exploits a small, but robust set of theoretically motivated features specifically aimed at selecting motion-related components from FSL-MELODIC output. It still uses Python2.7. Its main advantage is that is simple and completely automatic, user just have to run it (*do_aroma=True*) without any other effort. In the present pyMRI implementation, the aroma script uses the melodic output folder calculated in the previous step.

Nuisance removal

This step regress-out the signal present in non-brain tissues, like liquor and whole head signal which are evidently artefacts. It has been considered for years a mandatory step, while recently it was questioned. pyMRI still does it by default, but can also skip it (*do_nuisance=False*). The default high pass frequency is 1/period = 1/100 = 0.01 Hz and can be modified by setting the desired period (*hpfsec=100*).

Final melodic

The last melodic step does not process image (no filtering, no motion correction, no registration, no smoothing) but only does a melodic analysis (whether *do_melodic=True*), using a template file located in PROJDIR/script/glm/template and named *mel_preproc_model="singlesubj_melodic_noreg"*, writing a folder named *mel_odn="postmel"*, to let user evaluate the quality of the previous steps and the possible presence of artefacts.

Data finalization

If *replace_std_filtfun=True*, pyMRI fills the folder *PROJDIR/subjects/sX/SUBJLABEL/resting/reg_std* with cleaned epi, mask and background image saved in standard 4mm reference space, the one used by melodic-dual regression analysis to represent resting state networks and perform group analysis.

These two settings should not be modified: *do_featinitreg=False, do_melinitreg=False*.

DTI processing

Diffusion tensor imaging (DTI) is a rather simple method to measure white matter integrity. It is performed by default (*do_dtifit=True*).

Artefacts correction

Before any other operation on DTI image, user need to correct for movement and eddy current artefacts. There are two methods: if user also recorded a DTI sequence with the phase encoding (PE) direction inverted, referred as dti_pa, *wellcome* method will run the novel eddy tool (*do_pa_eddy=True*), otherwise (*do_pa_eddy=False*) it will run the classical eddy_correct tool. The former is a more advanced tool that necessitates that user recorded also few volumes of dti with an inverse phase encoding (PE) direction. Normally dti sequences are recorded with an anterior-to-posterior direction, the sequences used to correct dti are recorded with a posterior-to-anterior direction.

DTI fit

Once the original image is corrected, the DTI tensor can be fitted and the FA, MD, L1 and L23, each representing the voxel-wise values of such metric, are created. By default, the latter image is created

as the mean between L2 and L3 and represents the radial diffusivity (RD). These images will be later used in TBSS group analysis.

Bedpostx

The second processing that can be done on DTI images is the tractography, the possibility to reconstruct the integrity of brain axons fibers. Before running tractography, a preliminary bedpostx processing must be done (*do_bedx=True*), with output folder set with *bedpost_odn="bedpostx"*. This is a long-lasting process (up to 24 hours) that can be run using the CPU or the GPU (respectively setting *do_bedx_gpu=False/True*). With GPU the duration drops to 2 hours. User must evaluate whether is faster using several CPUs (when lots of ram and cores are available) or one cpu and the gpu.

Probtrackx

This tool is the standard software to run tractography in FSL. It is not implemented in pyMRI, user can refer to the bashMRI to run it.

xTract

pyMRI opted for implementing the xTract tool (https://fsl.fmrib.ox.ac.uk/fsl/fslwiki/XTRACT). XTRACT (cross-species tractography) can be used to automatically extract a set of carefully dissected tracts in humans and macaques. XTRACT reads the standard space protocols and performs probabilistic tractography (probtrackx2) in the subject's native space. Resultant tracts may be stored in either the subject's native space or in standard space. The user must provide the crossing fibres fitted data (bedpostx) and diffusion to standard space registration warp fields (and their inverse). xTract is run setting *do_xtract=True* and user can change the output dir (*xtract_odn="xtract"*). Tractography is run in native space by default, but can be alternatively run in standard space by setting *xtract_refspace= <refimage> <diff2ref> <ref2diff>*. In this case, user must provide three images, the standard space and the transformations: individual dti -> reference and reference -> individual dti. xTract can be run with cpu or gpu (respectively xtract_gpu=False/True). At the end of xTract tractographies, pyMRI calls a proper function that creates a summary files with the measures specified in: *xtract_meas="vol,prob,length,FA,MD,L1,L23"*.

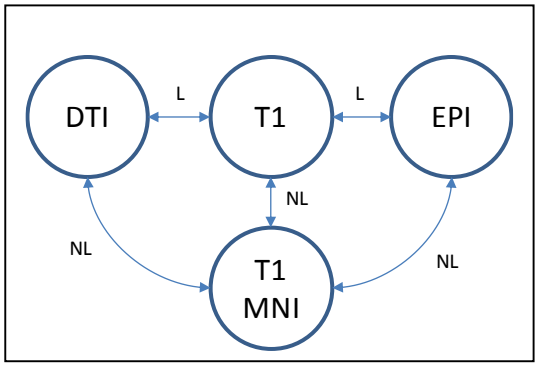
Structural connectivity

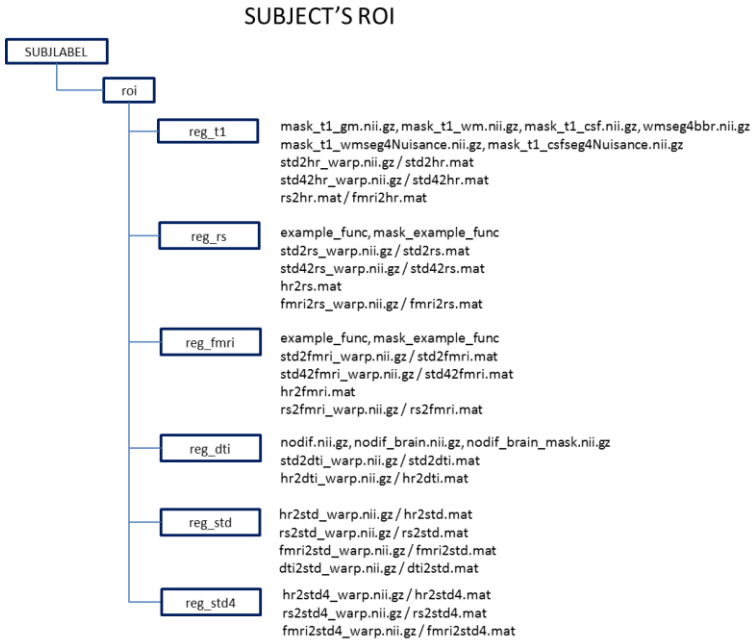
It is not implemented in pyMRI, user can refer to the bashMRI to run it.

```
do_struct_conn=False, struct_conn_atlas_path="freesurfer", struct_conn_atlas_nroi=0
```

Within-sequences coregistration

Most of the group analysis are voxel-wise (vertex-wise in case of cortical thickness). This means that the i-th voxel of sequence A of subject X, must correspond to the same anatomical location of i-th voxel of sequence A of subject Y. In order to make it happen, individual images must be co-registered to a same anatomical template, all whom voxels have been classified and clustered in specific Brodmann areas. The standard template is the MNI T1 template, available at 1, 2 and 4 mm of spatial resolution. FSL-Eyes, the fsl's image viewer software (and many others) have a tool to indicate to which cortical area the selected voxel belongs to. During *wellcome* script, each individual image is co-registered to such common space and transformation between sequences are calculated.





The output of these procedures is the *roi* folder that have one subfolder for each recorded sequence. Within each of these folder, all images are in that specific reference system. They contain:

- The one/first volume image of EPI and DTI images (e.g. the B0 of DTI)
- the linear (.mat) and non-linear (_warp.nii.gz) transformation from the other reference systems.
- ROIs (regions-of-interest) obtained from other modalities, results, literature

All these procedures rely on the non-linear co-registration between individual T1 and MNI T1 and on all the linear co-registrations between individual T1 versus all other individual images. A good-quality and well processed individual T1 is the key aspect of a valid group analysis. In case for example the T1 was badly recorded, that subject shall not be used for group analysis.

Post-wellcome processing

After this huge pipeline, and before starting any group analysis, results must be carefully checked. In particular, two things must be checked:

1) All BET went right
2) Co-registrations from each sequence to standard space were correct

In order to fulfil the latter requirement, and thus may proceed to group analysis, user must first check the former.

1- Compare Brain Extraction

pyMRI provides the method *Project.compare_brain_extraction* with the following signature:

```
compare_brain_extraction(outdir, subjs_labels=None, num_cpu=1)
```

outdir: full path of the output folder
subjs_labels: list of subject labels to check

which does two tasks:

1) for each subjects, it creates its "_brain" image done with different tools, namely BET, freesurfer and SPM
2) move these on-the-fly generated images to a given folder and run the FSL's slicesdir command.

slicesdir is an important routine that takes a folder and create a web page containing 8 snapshots (at different orientations and slices) of each image contained in that folder. This tool allows an easy and fast method to verify the quality of many images at a time. All the images must be in the same reference space. The three methods provide slightly different results in terms of liquor erosion. User should use the same method for all the subjects. Exceptions to this rule can be considered, after visual inspection of individual results.

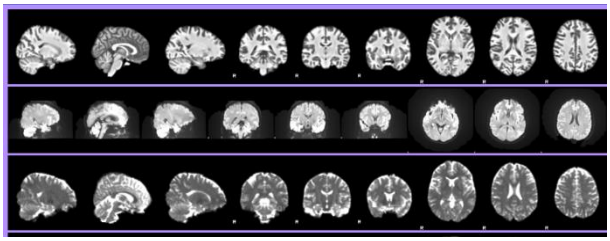


Figure 1:Example of a slicesdir command that display the subject T1, DTI and EPI images coregisterd to the standard template

2 - Co-registration check

pyMRI provides the method *Project.check_all_coregistration* with the following signature:

```
check_all_coregistration(outdir, subjs_labels=None, _from=None, _to=None, num_cpu=1,
overwrite=False)
```

outdir: full path of the output folder
subjs_labels: list of subject labels to check
_from: list of sequences name to co-register from
_to: list of sequences name to co-register to.
The allowed values are = *["hr", "rs", "fmri", "dti", "t2", "std", "std4"]*

Before starting any group analysis, user is requested to check how the three most common sequences (epi, dti, t1) co-register toward the standard space

```
outdir      = os.path.join(project.group_analysis_dir, "registration_check_2_std")
project.check_all_coregistration(outdir, _from=["hr", "rs", "dti"], _to=["std"],
num_cpu=num_cpu)
```

Further processing on some single subjects

If user must do some specific processing to a subject, it's possible to obtain its instance (and then call its methods) with this:

```
subj = project.get_subject_by_label("subject_name")
```

Subject must belong to the currently loaded list. Otherwise, user can do:

```
subj = Subject("subject_name", project)
```

Group Processing

In the MRI world we talk first (subject) and second (group) level analyses. The former, seen in previous paragraphs, regards preprocessing individual sequences (temporal and spatial filtering, segmentation and co-registration) and eventually modelling some metrics (DTIfit, task related bold signal). The latter consists in all those analyses that process subject level analysis output in order to extract group-level results by running a specific statistical model. pyMRI defines several classes to perform the latter.
Group analysis often consists in two steps:

- create a group template, co-registering all the individual images to a given template.
  This step is not a simple co-registration but involves different and more accurate processing that depends on the specific method. Some group analysis does not require it and individual images must be simply normalized to a common template
- perform statistical analysis
  in order to find out differences between given groups and/or correlation between specific variables (demographic, clinical, performance) and mri measures.

Kind of statistical analysis

There are two main tools to perform group level statistical analysis: the fsl's randomize and the SPM batch tool. Both approaches run a specific general linear model (GLM) on given data. GLM is a statistical model where user can specify one or more experimental factors of several levels and many covariates. Other available method is fsl's Feat, for SBFC analysis, or extract subjects' values and analyze in them in own favorite statistic suite (e.g. R, spss, statistica etc.).

Presently, pyMRI (or bashMRI) implements the following group analysis:

Seq.	Analysis	gr templ	normaliz	framework	stats
T1	VBM_FSL	X		shMRI	randomize
	VBM_DARTEL	X		pyMRI	spm
	Cortical thickness		X	pyMRI	spm
DTI	TBSS	X		pyMRI	randomize
	Tractography/xtract		X	pyMRI	R/spss
RS	Melodic/dual-regr	X		shMRI	randomize
	Seed-based FC		X	shMRI	feat
	fslNets	X		shMRI	randomize
fMRI	Task-fmri		X	pyMRI	spm

Preliminary check

The *Project* class contain a method to check whether specific analyses can be done, by verifying the presence of the images they need.
The method is:
Project.can_run_analysis(analysis_type,analysis_params=None,group_or_subjlabels=None, sess_id=1)
Its parameters are:

analysis_type: vbm_fsl, vbm_spm, ct, tbss, bedpost, xtract, melodic, sbfc, fmri
analysis_params: can be any type, depending on the method checked
group_or_subjlabels: can be a group label or a list of subjects' labels

Get subjects for group analysis

All the methods that use a list of subjects have a parameter called *grouplabel_or_subjlist*. This parameter can be either a string or a list of string or *Subject* instances. In the first case it must correspond to one of the subjects' list present in the subjects_lists.json file. In the latter case, it must be a list of subjects' labels or instances.
Subjects' labels are obtained with:

```
project.get_subjects_labels(grouplabel_or_subjlist)
```

Subjects' instances are obtained with:

```
project.get_subjects(grouplabel_or_subjlist)
```

Both methods also validate the list before returning it, that is, they check whether each subject really exist in the file system.

External data

Statistical analysis always involves using non-mri data, like subjects age and gender, participants behavioural or performance measurements and/or patients' clinical scales. These data are usually stored in excel or tab-separated text files. Regardless of the file type, these files are arranged as a matrix, with one row for each subject and one column for each measure. pyMRI likes the latter and have a specific class (*SubjectsDataDict*) that read these file and manipulate subjects' columns. Given such a file, pyMRI transforms it in a Python dictionary:
{"subj1": {"age":25, "gender":"m", "var1":23 }, "subj2": {"age":20, "gender":"f", "var1":20 }, ...}

This is an example to extract data from a tab separated data file.

```
group_label  = "all_46_seq1"
subjects     = project.load_subjects(group_label, SESS_ID)

datafile     = os.path.join(project.script_dir, "data.dat")
data         = SubjectsDataDict(datafile)

# to extract all ages as a list
age          = data.get_column("age")

# to extract all ages as a single string, with values in separate lines.
age_str      = data.get_column_str("age")

# to extract a subset of the whole list
age          = data.get_filtered_column("age", project.get_subjects_labels("test"))

# to extract those having a given value
age          = data.get_filtered_column_by_value"cat_dist", 0)

# to extract those having column values between two given values
age          = data.get_filtered_column_within_values("age", 18, 25)
```

T1

VBM

Voxel-Based Morphometry (VBM) is an analysis technique that measure the gray matter density of human brain. VBM can be run with many tools, pyMRI (or shMRI) implements two of them, one run with dartel toolbox of SPM (pyMRI), the other with FSL (shMRI).
Dartel (https://www.fil.ion.ucl.ac.uk/spm/course/slides10-vancouver/09_Morphometry.pdf) is considered one of the best (if not the best) method to co-register several segmented images into a group template. Once all individual segmented T1 are co-registered together in a normalized reference space, analysis is done voxel-wise, that is each single voxel is compared across experimental conditions.

VBM SPM

The first step needed to run dartel-based VBM is to run the wellcome script with the param *do_spm_seg=True.* This step creates the individual rc1T1, rc2T1, c1T1 images that are needed.
This method starts creating a T1 group template using all the individual T1 processed (segmented and saved in dartel-compatible format). This step also determines the following statistical analyses, strictly speaking, user can compare only normalized T1 that were co-registered to the same template. If the subgroup investigated is highly different to the entire group
Then user must call the method *GroupAnalysis.create_vbm_spm_template_normalize(name, subjects_list)* indicating:

name:	name of the generated template (e.g. controls56, bd_man21, patients78, where the number indicates the number of subjects
subjects_list:	list of subjects' instances

DTI

Dti group analysis consists in two main methods: TBSS (tracto-based spatial statistics) and tractography. The former provides a voxel-wise comparison of FA, MD, RD, LD maps, the latter allow to estimate white matter integrity at tract level. The two methods can be combined: individual tracts are individuated, and mean FA, MD, RD, LD values within them can be calculated and statistically interpreted. TBSS in fact, provide simple voxel-wise analysis and often reports differences in some voxels of some tracts. Investigating diseases like multiple sclerosis, that is known to produce focal lesions, this analysis makes perfectly sense. In most of the cases instead, a reduced FA in some voxel of a tract is quite meaningless, better is calculate the mean FA value of an entire tract and thus be able report that the overall FA of a specific tract is affected by a certain experimental factor.

TBSS

TBSS (https://fsl.fmrib.ox.ac.uk/fsl/fslwiki/TBSS) methods starts with the creation of a group template through the method:
tbss_run_fa(subjects_list, odn, sessid=1, prepare=True, proc=True, postreg="S", prestat_thr=0.2)
It takes a list of subjects' instances and creates a folder called as *odn* parameter in the group_analysis/tbss folder.
In this folder, it creates the individual skeletons projected toward the FA skeleton template.
For testing and or experiments, user can select whether just preparing (*prepare=True*) and/or processing (*proc=True*).

More important is the parameter *postreg* which regulate toward which project the individual skeletons, the allowed values are:
S: create a skeleton group template and project over it
T: project individual skeletons over the FMRIB58_FA mean FA image
The *prestat_thr* should be leaved as this.

The output of this step is 4D image containing, the FA/MD/RD/LD white matter skeleton of each subject, projected on the group template (S) or the FMRIB58_FA (T).

A tbss scripts is:

```
proj_dir    = "/data/MRI/Projects/past_controls"
project     = Project(proj_dir, globaldata)
SESS_ID     = 1
num_cpu     = 1
analysis    = GroupAnalysis(project)

group_label         = "all_rs_dti"
population_label    = "controls57_FMRIB58"
postreg_option      = "T"
subjects            = project.get_subject(group_label)
main_folder         = analysis.tbss_run_fa(subjects, population_label, postreg="T")
```

This script performs skeletons projection of the FA images only, to also project the other modalities fitted by DTIFIT method, user must call the following method:

```
analysis.tbss_run_alternatives(subjects, main_folder, ["MD", "L1", "L23"])
```

The TBSS pipeline continues with statistical analysis (performed with randomize) and results view (using fsleyes), both these steps are currently not implemented in pyMRI and the bashMRI must be used.

xTract

At the subject level, the wellcome script performs individual tractography of xTract's predetermined tracts (from now, simply called xtract). This provides a parcelization of subject's white matter in well-known tracts and allow user to refer to specific tracts rather that voxels, likely belonging to a specific tract.
The most common usage of tractography (xTract is just an automated method to perform individual tractography over known tracts) is calculate mean FA/MD/RD/LD values over a tract. The tract thus acts as a mask for tbss results.
xTract contains a convenient method to create a summary file for each subject's tract, pyMRI has a method that takes a list of subjects and create a single file containing the asked metrics of all those subjects.
xtract_export_group_data(subjs_or_group, ofp, tracts=None, values=None, ifn="stats.csv"):
Its parameters are:

- subjs_or_group: label of a group list defined in *subjects_lists.json* or a list of subjects instances (not labels)
- ofp: full path of the output file
- tracts: list of tracts to output, valid values are those defined by xtract tool
- values: list of dtift metrics to summarize

```
sign_tract_in_tbss  = ["af_r", "atr_l", "atr_r", "cbd_l", "cbp_l",….,"slf2_r" , "slf3_r"]
res_file            = os.path.join(project.tbss_dir, "valid_xtract_fa_rd.dat")
analysis.xtract_export_group_data(group_label,res_file,values=["mean_FA","mean_L23"],
                                  tracts=sign_tract_in_tbss)
```

xTract and TBSS

The two methods can be integrated. One possible scientific question is: to which tract, the significant tbss voxels belong to? at which percentage a specific tract overlapped with a tbss-derived map? To answer to this question, two pyMRI methods can be used.
The former has to be run just once, it performs an INTERSECTION operation between the FMRIB58 FA skeleton and each xtract (set 1 to each voxel in both the skeleton and the given tract, set 0 in all other cases), that is, it creates a mask of each xtract in the FMRIB space.
Then user, having obtained a result map with tbss, can investigate its overlap degree with the xtracts by calling:
GroupAnalysis.tbss_clusterize_results_by_atlas(tbss_result_image,out_folder,log_file="log.txt",tracts_labels,tracts_dir,thr=0.95)

tbss_result_image:	significant tbss maps to process
tracts_labels:	xtract labels,
tracts_dir:	folder containing the clusterization of the FMRIB58 FA template xtract

```
measure      = "L23" # or "FA"
out_folder   = os.path.join(tbss_folder, ……, measure)
tbss_img     = os.path.join(……)

analysis.tbss_clusterize_results_by_atlas(tbss_img,out_folder,measure+"_tbss_segm_on_xtract.
txt", globaldata.dti_xtract_labels, globaldata.dti_xtract_dir)
```

This method creates a folder containing one image for each not-empty intersection between tbss image and each xtract's tract (that is, if tbss image does not overlap with a tract, it doesn't create any image). Then, it calculates the number of voxels of these overlapping tract and its coverage percentage ((number of overlapping voxels / total number of voxels) * 100. Moreover, it creates an image containing all those tbss voxels that could not be classified as belonging to any xtract, to check how many voxels could not be properly classified. This method can help user understand how much a specific tract was affected by the experimental condition tested in that tbss result map.

Then …TO BE COMPLETED

```
var, labs  = data.get_filtered_column("variable") # tuple[2] of  values, subj_label
out_folder = os.path.join(tbss_folder, "results", "results_var_xtract_fmrib58", measure)
analysis.tbss_summarize_clusterized_folder (out_folder, var, "variable", tbss_folder)
```

Statistical Analysis

Statistical analysis is divided in two steps:

1. Model definition
2. Post Model (Contrasts definition and results correction)

1) Define a model means specify the experimental factors and nuisance/covariates best explaining our data. Typical experimental factors can be group (with each population being one of its *level*) and/or different conditions in a task. Nuisance/Covariates are additional values (subjects' age and gender, clinical scales scores, behavioral performances, etc…) which can be used to correct data for (removing their effect from data), and we talk about Nuisance or directly investigated (Covariates), e.g. studying the effect of age over an MRI metric (e.g. cortical thickness).
2) User investigates the effect of each factor or covariate by setting a so-called contrast, which represent a scientific question (is level A of factor 1 higher that level B of the same factor?). Then, since analysis are voxel-based, and voxels can be dozens/hundreds of thousands, results must be corrected for multiple comparison, specifying a method (FWE or FDR), a p-value and a cluster size.

SPM

The SPM batch tool is a useful instrument to plan serial processing on subjects' data. It has a GUI to let user graphically select desired processing, pickup from the file system the required images and set the needed parameters. Moreover, and more importantly, is also able to write the created batch (***batch file***) in a text file. By modifying such text files, e.g changing the subject(s) name(s), user can conveniently reuse such batches. pyMRI perform statistical analysis (but also some subject level one) by changing these so-called template files editing some given parameters (images paths, subjects' data, output folders, etc.).
In pyMRI the two phases (model definition and post model) are executed by one method that runs two different SPM batches.

Available models

Presently pyMRI can use these following statistical models:

- One group multiple regression
- Two samples t-test
- One-way ANOVA
- Two-way ANOVA

All these four methods are implemented in the SPMModels class. There are versions for VBM, CT and fMRI, and all have the same parameters.
For example, to run a two-samples T-test analysis, user must call the following method:

```
spm_analysis = SPMModels(project)
analysis.batchrun_cat_thickness_stats_factdes_2samplesttest(
```

root_outdir	folder where the template and/or the subjects' files are present
analysis_name	name of the analysis folder (subfolder of root_outdir) and prefix to batch files
groups_instances	list of subjects' instances
covs	list of either Nuisance or Covariates instances.
data_file	path to a data file. If None, use the one loaded by default in project
glob_calc	valid values are: "subj_icv" | "subj_tiv" | "", uses subject's values or do not correct
cov_interaction	list of flags (0|1) indicating whether each covariate interact with data

expl_mask	specify a mask to restrict results
spm_template_name	name of the spm template that perform stats
post_model	instance of PostModel class , specifying value for contrasts and results analysis
runit	set whether running the analysis (True) or only write batch files

```
proj_dir     = "/data/MRI/projects/T15"
project      = Project(proj_dir, globaldata)
spm_analysis = SPMModels(project)

SESS_ID      = 1
subjects     = project.get_subjects(group_label, SESS_ID)

covs         = [Nuisance("gender"), Covariate("age")] # get correlation with age,
                                                      # correcting for gender
anal_name    = "multregr_age_gender"
statsdir     = os.path.join(project.group_analysis_dir, "mpr/thickness")
post_model   = PostModel("spm_stats_contrasts_results", covs, [],
                         res_params=ResultsParams("FWE", 0.01, 10))
spm_analysis.batchrun_cat_thickness_stats_factdes_1group_multregr(statsdir, anal_name,
                                                    subjects, covs, post_model=post_model)
```

Three classes were here used for the first time: Nuisance, Covariate and PostModel.
The first two are used to specify whether the inserted values must be used only to correct data, thus act as *Nuisance* variables, or to be investigated (*Covariate*) and thus have an own contrast in the postmodel phase. The string set as the only parameters must correspond to a column within the subjects' data file.
PostModel is class containing the following properties:

- *template_name* : template file name (without the ending "_job.m"), present in pyMRI
  full path of a file (without the extension ".m") given by user
- *regressors* : list of either covariate/nuisance instances
- *contr_names* : list of contrasts names
- *res_params* : multiple correction params: method (FWE, FDR, none), p-value (< 0.05), cluster extension (value $\geq$ 0)
- *isSpm* : indicate whether is a standard SPM (True) or a CAT (False) analysis

FSL (glm & randomize)

Cross-projects group analysis

The suggested way to manage different populations, e.g. patients and controls, in pyMRI, is to create one project containing only controls and one the population of interest. This allow reusing controls data in several projects without duplicating files. All the subject-level analyses are in fact usually the same. Group analysis steps thus pick individual images from different projects and output group templates and statistical analysis to only one project's (usually the patient one) "group_analysis" folder. This is how to do it in pyMRI:

```
ctrl_proj_dir       = "/data/MRI/projects/controls"
ctrl_project        = Project(ctrl_proj_dir, globaldata)
pat_proj_dir        = "/data/MRI/projects/patients"
pat_project         = Project(pat_proj_dir, globaldata)


spm_analysis = SPMModels(pat_project)   # reference project for group-level analysis is the
                                          patients' one

ctrl_group_label    = "ctrl_for_patients_study" # an age-matched subset of all controls
patient_group_label = "all_patients"            # the patient group of interest
SESS_ID             = 1
ctrl_subjects       = ctrl_project.get_subjects(ctrl_group_label, SESS_ID)
pat_subjects        = ctrl_project.get_subjects(patient_group_label, SESS_ID)

cov_names    = ["gender", "age"]
anal_name    = "an_analysis"
statsdir     = os.path.join(pat_project.group_analysis_dir, "mpr/thickness/" + anal_name)


all_subjects = ctrl_subjects.append(pat_subjects)    # create a list concatenating controls'
                                                       list with patient one
.create_vbm_spm_template_normalize("ctrl_patients", all_subjects)

spm_analysis.batchrun_cat_thickness_stats_factdes_1group_multregr(statsdir, subjects,
cov_names, spm_contrasts_template_name="")
```

Python Classes

The *Subject* class

The *Subject* class is the main data structure of pyMRI. It holds all the properties and methods needed to perform data analysis on individual data. The properties are the full path of each file and folder of interest (more than 100) and four special properties

```
self.transform    = SubjectTransforms(self, self._global)
self.mpr          = SubjectMpr(self, self._global)
self.dti          = SubjectDti(self, self._global)
self.epi          = SubjectEpi(self, self._global)
```

which are instances of four classes that each deals with a specific kind of sequence (SubjectMpr, SubjectDti, SubjectEpi) or implement transformation among them (SubjectTransforms).

The *Subject* class public methods

create_file_system(

check_images(

reslice_image(

welcome(

renameNifti(

mri_merger(

The *SubjectMpr* class public methods

prebet

bet(odn="anat", imgtype=1, smooth=10, biascorr_type=BIAS_TYPE_STRONG, do_reorient=True, o_crop=True, do_bet=True, do_overwrite=False, use_lesionmask=False, lesionmask="")

spm_segment(odn="anat", imgtype=1, do_overwrite=False, do_bet_overwrite=False, dd_bet_mask=False, set_origin=False, seg_templ="", spm_template_name="spm_segment_tissuevolume")

spm_segment_check(check_dartel=True)

cat_segment(odn="anat", imgtype=1, do_overwrite=False, do_bet_overwrite=False, add_bet_mask=True, set_origin=False, seg_templ="", coreg_templ="", calc_surfaces=0, num_proc=1, use_existing_nii=True, use_dartel=True, spm_template_name="cat27_segment_customizedtemplate_tiv_smooth")

cat_segment_check(calc_surfaces=True)

cat_segment_longitudinal(sessions, odn="anat", imgtype=1, do_overwrite=False, do_bet_overwrite=False, add_bet_mask=True, set_origin=False, seg_templ="", coreg_templ="", calc_surfaces=0, num_proc=1, use_existing_nii=True, spm_template_name="cat_segment_longitudinal_customizedtemplate_tiv_smooth")

cat_segment_longitudinal_check(sessions, calc_surfaces=0)

cat_surfaces_complete_longitudinal(sessions, num_proc=1)

cat_surf_resample(session=1, num_proc=1, isLong=False, mesh32k=1, endengine=True, eng=None)

cat_tiv_calculation(session=1, isLong=False, endengine=True, eng=None)

spm_tissue_volumes(spm_template_name="spm_icv_template", endengine=True, eng=None)

surf_resampled_longitudinal_diff(sessions, outdir="", matlab_func="subtract_gifti")

postbet(odn="anat", imgtype=1, smooth=10, betfparam=0.5, do_reg=True, do_nonlinreg=True, do_seg=True, do_cleanup=True, do_strongcleanup=False, do_overwrite=False, use_lesionmask=False, lesionmask="lesionmask")

finalize(odn="anat", imgtype=1)

first(structures="", t1_image="", odn="")

fs_reconall(step="-all", do_overwrite=False, backtransfparams=" RL PA IS ")

use_fs_brainmask(backtransfparams=" RL PA IS ", erosiontype=" -kernel boxv 5 ", is_interactive=True, do_clean=True)

use_fs_brainmask_exec(do_clean=True)

use_spm_brainmask(backtransfparams=" RL PA IS ", erosiontype=" -kernel boxv 5 ", is_interactive=True, do_clean=True)

use_spm_brainmask_exec(do_clean=True)

compare_brain_extraction(tempdir, backtransfparams=" RL PA IS ")

cleanup(lvl=Global.CLEANUP_LVL_MIN)

The *SubjectDti* class public methods

get_nodiff(logFile=None)

ec_fit(logFile=None)

bedpost(out_dir_name="bedpostx", use_gpu=False, logFile=None)

probtrackx(

xtract(outdir_name="xtract", bedpostx_dirname="bedpostx", refspace="native", use_gpu=False, species="HUMAN", logFile=None)

xtract_check(in_dir="xtract")

xtract_viewer(xtract_dir="xtract", structures="", species="HUMAN")

xtract_stats(xtract_dir="xtract", refspace="native", meas="vol,prob,length,FA,MD,L1", structures="",logFile=None)

xtract_read_file(tracts=None, values=None, ifn="stats.csv", logFile=None)

conn_matrix(atlas_path="freesurfer", nroi=0)

The *SubjectEpi* class public methods

get_example_function

pepolar_correction(motionfirst=True, epi_ref_vol=-1, ref_image_pe="", ref_volume_pe=-1)

fsl_feat(epi_label, in_file_name, out_dir_name, model, do_initreg=False, std_image="", tr="", te="")

aroma(epi_label, input_dir, md="", mc="", aff="", warp="", ofn="ica_aroma", upsampling=0, logFile=None)

remove_nuisance(in_img_name, out_img_name, epi_label="rs", ospn="", hpfsec=100)

get_slicetiming_params(nslices, scheme=1, params=None)

spm_motioncorrection(ref_vol=1, ref_image=None, epi2correct=None, spm_template_name="spm_fmri_realign_estimate_reslice_to_given_vol")

get_closest_volume(ref_image_pe="", ref_volume_pe=-1)

prepare_for_spm(in_img, subdirmame="temp_split")

spm_fmri_preprocessing_motioncorrected(num_slices, TR, TA=-1, acq_scheme=0, ref_slice=-1, slice_timing=None)

spm_fmri_preprocessing(num_slices, TR, TA=-1, acq_scheme=0, ref_slice=-1, slice_timing=None, epi_image=None, spm_template_name='spm_fmri_preprocessing')

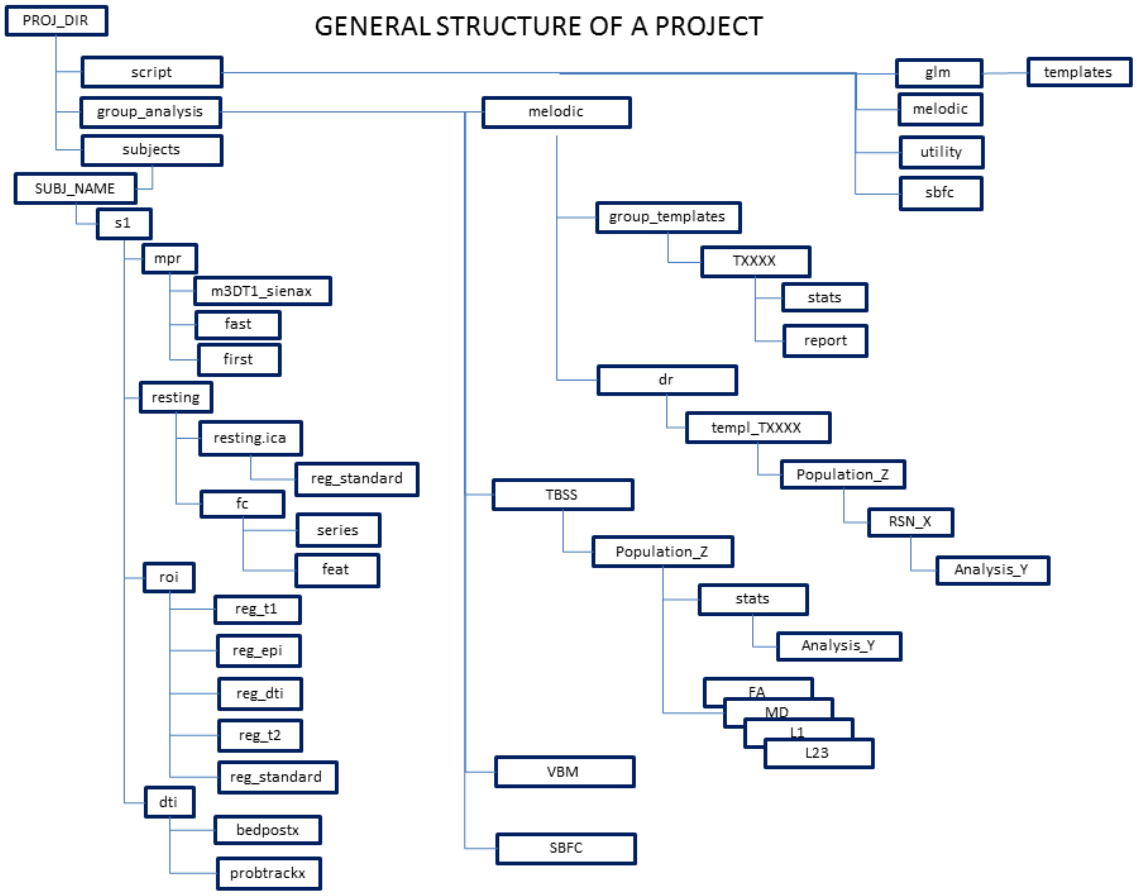
spm_fmri_1st_level_analysis(analysis_name, TR, num_slices, conditions_lists, events_unit="secs", spm_template_name='spm_fmri_stats_1st_level', rp_filemame="")

cleanup(lvl=Global.CLEANUP_LVL_MIN)

adopt_rs_preproc_step(step_label, outsuffix="")

adopt_rs_preproc_folderoutput(proc_folder)

eg_copy_feat(epi_label, std_image="")


GENERAL STRUCTURE OF A PROJECT
PROJ_DIR
script
group_analysis
subjects
SUBJ_NAME
s1
mpr
m3DT1_sienax
fast
first
resting
resting.ica
reg_standard
fc
series
feat
roi
reg_t1
reg_epi
reg_dti
reg_t2
reg_standard
dti
bedpostx
probtrackx
melodic
group_templates
TXXXX
stats
report
dr
templ_TXXXX
Population_Z
RSN_X
Analysis_Y
TBSS
Population_Z
stats
Analysis_Y
FA
MD
L1
L23
VBM
SBFC
glm
templates
melodic
utility
sbfc

MELODIC

The melodic package allows you to evaluate the **within-network** functional connectivity. It decomposes the brain rest activity in resting-state networks (RSN) and then, independently for each RSN, assess the functional connectivity of each voxel contained within the RSN to the other voxel of the same network. This measure can be then compared between groups or correlated with clinical, behavioral and demographic variables.

The processing pipeline include: o) subject welcome, i) single-subject analysis, ii) template creation, iii) dual-regression of subjects' data to the template, iv) statistical analysis, v) data visualization and vi) final packaging for publication.

1: subjects_single_melodic:

GUI usage

The analysis is normally performed by the GUI application called Melodic. This tool requires that you define:

- subject RS image: resting.nii.gz
- the anatomical scalped brain image SUBJLABEL-t1_brain.nii.gz
- the TR and TE values of the sequence
- Information over the normalization to standard anatomical template
    - anatomical template path
    - spatial re-sampling dimension in mm. (usually 4 mm)
    - registration approach (linear vs non-linear)
- the spatial smoothing (5 or 6 mm),
- the high-pass filter cutoff (between 100-150)
- optionally the output folder if different from the default one (resting.ica)

NOTE: the TE values must be edited manually in the fsf file produced by the GUI application. Hence it is advisable to create a study template file for 1st-level melodic analysis, load that file and modify subject-dependent information.

Scripted usage

The global script for this step is a multi-threated script; in the project script you must define this information:

- number of CPU to be used
- the global script ($GLOBAL_SUBJECT_SCRIPT_DIR/execute_subject_melodic.sh)
- fsf template
- subjects list string name (variable defined in $PROJ_SCRIPT_DIR/subjects_list.sh)
- custom output folder (optionally)

```
. $PROJ_SCRIPT_DIR/subjects_list.sh

EXECUTE_SH=$GLOBAL_SUBJECT_SCRIPT_DIR/execute_subject_melodic.sh
melodic_fsf_template=$PROJ_SCRIPT_DIR/glm/singlesubj_melodic
declare -i NUM_CPU=2
postfix_output_folder_name="non-default_name"  # optional string appended to the label :
$SUBJ_NAME-rs
#=============================================================

# standard call....read: SUBJ_NAME/resting/resting.nii.gz,  create a folder:
SUBJ_NAME/resting/resting.ica
. $MULTICORE_SCRIPT_DIR/define_thread_processes.sh $NUM_CPU $EXECUTE_SH "$arr_subj"
$PROJ_DIR -model $melodic_fsf_template

# non-standard input file....  read: SUBJ_NAME/resting/resting_skip4vol.nii.gz, create a
folder SUBJ_NAME/resting/resting.ica
. $MULTICORE_SCRIPT_DIR/define_thread_processes.sh $NUM_CPU $EXECUTE_SH "$arr_subj"
$PROJ_DIR -ifn resting_skip4vol -model $melodic_fsf_template

# non-standard input file and folder....  read: SUBJ_NAME/rs2/resting_skip4vol.nii.gz,
create a folder SUBJ_NAME/rs2/resting.ica
. $MULTICORE_SCRIPT_DIR/define_thread_processes.sh $NUM_CPU $EXECUTE_SH "$arr_subj"
$PROJ_DIR -ifn resting_skip4vol -idn rs2 -model $melodic_fsf_template

# non-standard input file and folder and output dir....  read:
SUBJ_NAME/rs2/resting_skip4vol.nii.gz, create a folder SUBJ_NAME/rs2/resting _denoised.ica
. $MULTICORE_SCRIPT_DIR/define_thread_processes.sh $NUM_CPU $EXECUTE_SH "$arr_subj"
$PROJ_DIR -ifn resting_skip4vol -idn rs2 -model $melodic_fsf_template -odn resting_denoised
```

The output of such analysis is a folder (SUBJECTS_DIR/SUBJ_NAME/resting/resting.ica) containing the subject-level analysis of resting state data. The most important files created are the

- resting.ica /filtered_func_data.nii.gz
- resting.ica/reg_standard/filtered_func_data.nii.gz (registered to the anatomical template)

The latter is of special interest as it will be later used by the dual regression process. They represent the filtered data.
This step is used to verify the quality of subjects' data, how his movement affected the RSN identification and if he needs a specific denoising. As later discussed

1a: denoising (optional)

It is possible to remove specific artifacts from the original data after a preliminary melodic analysis. The procedure is realized by visually inspecting the melodic output, take note of the artefactual components id and invoke the fsl_regfilt command which remove those components from the signal and create a denoised file.

<u>There are two approaches:</u>

- simply correct the rs data without changing its final name (substitute the original file which is in turn renamed as filtered_func_data_original.nii.gz)
- preserve original file name and create a denoised version (filtered_func_data_denoised)

The former is used when you plan to analyze original data and you had just to correct the data of few subjects. On the contrary, the latter approach is used when you plan to denoise all subjects data, thus it creates a reg_standard_denoised folder, later used by dual-regression analyses on denoised data.

Few subjects correction

```
declare -a arr_subjects2denoise=(DYT_B_prsic_svetislav)
declare -a arr_ic2remove=("1,2,3,4,5,6,7,8,9,10, 12,13,14,15, 17,18,19, 22,23,25,26,27,28")

declare -i cnt=0
for SUBJ_NAME in ${arr_subjects2denoise[@]}
do
  echo "$SUBJ_NAME"
  . $GLOBAL_SCRIPT_DIR/subject_init_vars.sh
    mv $RS_DATA.ica/reg_standard $RS_DATA.ica/reg_standard_original
    mv $RS_DATA.ica/filtered_func_data.nii.gz
$RS_DATA.ica/filtered_func_data_original.nii.gz
    $FSLDIR/bin/fsl_regfilt -i $RS_DATA.ica/filtered_func_data_original.nii.gz -o
$RS_DATA.ica/filtered_func_data.nii.gz -d $RS_DATA.ica/filtered_func_data.ica/melodic_mix -f
"${arr_ic2remove[cnt]}"
    $FSLDIR/bin/featregapply $RS_DATA.ica
    $FSLDIR/bin/fslroi $RS_DATA.ica/filtered_func_data.nii.gz
$RS_DATA.ica/filtered_func_data_skip4vol.nii.gz 4 196
    $FSLDIR/bin/fslroi $RS_DATA.ica/reg_standard/filtered_func_data.nii.gz
$RS_DATA.ica/reg_standard/filtered_func_data_skip4vol.nii.gz 4 196
     cnt=$cnt+1
done
```

Whole population correction

```
declare -a arr_ic2remove=("1,2,3,4,5,6,7,8,9,10, 12,13,14,15, 17,18,19, 22,23,25,26,27,28"
"…..” “……” “……” ……)

declare -i cnt=0
for SUBJ_NAME in ${arr_patients[@]}    #variable found in subjects_list.sh
do
  echo "$SUBJ_NAME"
  .   $GLOBAL_SCRIPT_DIR/subject_init_vars.sh
  mv $RS_DATA.ica/reg_standard $RS_DATA.ica/reg_standard_original
  mv $RS_DATA.ica/filtered_func_data.nii.gz $RS_DATA.ica/filtered_func_data_original.nii.gz

  $FSLDIR/bin/fsl_regfilt -i $RS_DATA.ica/filtered_func_data_original.nii.gz -o
$RS_DATA.ica/filtered_func_data.nii.gz -d
  $RS_DATA.ica/filtered_func_data.ica/melodic_mix -f "${arr_ic2remove[cnt]}"
   $FSLDIR/bin/featregapply $RS_DATA.ica
   $FSLDIR/bin/fslroi $RS_DATA.ica/filtered_func_data.nii.gz
$RS_DATA.ica/filtered_func_data_skip4vol.nii.gz 4 196
   $FSLDIR/bin/fslroi $RS_DATA.ica/reg_standard/filtered_func_data.nii.gz
$RS_DATA.ica/reg_standard/filtered_func_data_skip4vol.nii.gz 4 196
    cnt=$cnt+1
done
```

Note1

In this phase is it advisable to be very conservative: in case of doubt keep the IC, it may hide some good signal, and delete only those IC related to subjects movements, which highly differs between subjects. For instance, the artifact related to cardiac impulse and blood flow is present in each subjects and have similar spatio-frequency pattern, thus the group melodic algorithm is perfectly able to find it in every

subjects and associate it to a common IC which will not be considered in the group template. The criteria to define artefactual IC are out of the scope of the present guide.

Note2

In the two examples, the first 4 volumes of the data were removed from the filtered_func_data. This approach is used by several researchers, particularly when their scanners are old or not perfectly set-up. The signal present in the first volumes can be in fact altered by the not perfect signal stabilization. It is advisable to view some subject melodic in order to decide if this procedure is necessary and how many volumes should be removed.

2: template definition

This step creates the group template representing the RSN spatial pattern that will be later investigated with randomise.
The subjects used to create the template must belong to one of the following populations:

- healthy controls, better if age-matched, **different from those included** in the study
- entire population of the study (controls + patients)

Otherwise you can use the templates located at the following paths.

$GLOBAL_SCRIPT_DIR/data_templates/rsn/fsl_20/rsn20_444.nii.gz
$GLOBAL_SCRIPT_DIR/data_templates/rsn/bishwal/metaICA_2mm.nii.gz

After having created the group template (a) at
$PROJ_GROUP_ANALYSIS_DIR/melodic/group_templates/XXX,
you must inspect the IC by opening the web page at /.../XXX/report/00index.html and then (b) create the template script file, an sh file containing the variables which defines the template characteristics.

Template creation

It is performed using the following script, where user must define i) the TR of the sequence, ii) the name of subjects resting-state image input folder (rs), iii) the name of the RS images (rs), iv) the template name and v) use the proper subjects list (whom corresponding variables are defined in the subjects_list.sh file)

```
TR_VALUE=3.0
SUBJECTS_INPUT_RS_DIR_NAME=resting
SUBJECTS_INPUT_RS_NAME=resting

output_template_name=belgrade_dyt_controls21_patients45_skip4vol
arr_ctrl=${arr_controls21[@]}
arr_patients=${arr_patients45[@]}
input_file_name=filtered_func_data_skip4vol


CTRL_SUBJECTS_DIR=/gnappo/home2/dati/.../HC/subjects
MELODIC_OUTPUT_DIR=$PROJ_GROUP_ANALYSIS_DIR/melodic/group_templates/$output_template_name
```

```
mkdir -p $MELODIC_OUTPUT_DIR

filelist=$MELODIC_OUTPUT_DIR/.filelist_$template_name

echo "creating file lists"
bglist=""
masklist=""

for SUBJ_NAME in ${arr_ctrl[@]}
do
reg_standard_dir=$CTRL_SUBJECTS_DIR/$SUBJ_NAME/s$SESS_ID/$SUBJECTS_INPUT_RS_DIR_NAME/$SUBJEC
TS_INPUT_RS_NAME.ica/reg_standard
  bglist="$bglist $reg_standard_dir/bg_image"
  masklist="$masklist $reg_standard_dir/mask"
  echo "$reg_standard_dir/$input_file_name" >> $filelist
done

for SUBJ_NAME in ${arr_patients[@]}
do
  . $GLOBAL_SCRIPT_DIR/subject_init_vars.sh

reg_standard_dir=$SUBJECT_DIR/$SUBJECTS_INPUT_RS_DIR_NAME/$SUBJECTS_INPUT_RS_NAME.ica/reg_st
andard
  bglist="$bglist $reg_standard_dir/bg_image"
  masklist="$masklist $reg_standard_dir/mask"
  echo "$reg_standard_dir/$input_file_name" >> $filelist
done

echo "merging background image"
$FSLDIR/bin/fslmerge -t $MELODIC_OUTPUT_DIR/bg_image $bglist
$FSLDIR/bin/fslmaths $MELODIC_OUTPUT_DIR/bg_image -inm 1000 -Tmean
$MELODIC_OUTPUT_DIR/bg_image -odt float
echo "merging mask image"
$FSLDIR/bin/fslmerge -t $MELODIC_OUTPUT_DIR/mask $masklist

echo "start group melodic !!"
$FSLDIR/bin/melodic -i $filelist -o $MELODIC_OUTPUT_DIR -v --nobet --bgthreshold=10 --
tr=$TR_VALUE --report --guireport=$MELODIC_OUTPUT_DIR/report.html --
bgimage=$MELODIC_OUTPUT_DIR/bg_image -d 0 --mmthresh=0.5 --Ostats -a concat
```

Template script file

User must define the parameters of a group template by creating a proper file which declares some variables used by the subsequent processing steps. These file must be stored in the folder:

$GLOBAL_SCRIPT_DIR/melodic_templates/
by calling : . $GLOBAL_SCRIPT_DIR/melodic_templates/fsl_rsn20_444.nii.gz
you have all these variables available in your project script.

```
template_name=fsl_rsn20

TEMPLATE_MELODIC_IC             /..path to/melodic_IC.nii.gz or which ever 4D images
MASK_IMAGE                      / path to / mask image
BG_IMAGE                        / path to / bg_image.nii.gz
TEMPLATE_STATS_FOLDER                 / path to thresh_tstatsXXX images
TEMPLATE_MASK_FOLDER                  / path to / folder containing single RSN mask.

str_pruning_ic_id="0,1,3,4,6,7,12,15"         /  0-based ids of RSN networks of interest
```

```
str_arr_IC_labels="SM,AUDIO,DMN,..."                / labels of RSN networks of interest
declare -a arr_IC_labels=(SM AUDIO DMN .. .. ..)    / same elements of previous parameters
but as an array
```

VERY IMPORTANT !!!!:

In *str_pruning_id* variable, RSN ID are 0-based. That is, when you select your RSN of interest in the report folder, you must subtract 1 from its component number (DMN is at component 6, you must note its ID as 5)

Template RSN masks

If you want to restrict randomize analyses to the RSN mask you have two options. One is to use the thresh_zstat maps calculated by group-melodic step (located in: group_templates/templ_name/stats folder). The second consists in calculating the mean map of each component with randomize. A script is available to perform this step: it performs a randomize, then it masks them with a threshold of 0.998 creating a mask for each RSN previously defined as network of interests.

```
SINGLE_IC_PRUNING_SCRIPT=$GLOBAL_GROUP_SCRIPT_DIR/dual_regression_split2singleIC.sh
EXECUTE_STATS_SH=$GLOBAL_GROUP_SCRIPT_DIR/dual_regression_randomize_singleIC_multiple_folder
s_mean_mask.sh

NUM_PERM=5000
NUM_CPU=3
NUM_SUBJECTS=78                  # preserved for backward compatibility

. $GLOBAL_SCRIPT_DIR/melodic_templates/belgrade_controls.sh      # load template-related
variables
TEMPLATE_DIR=$PROJ_GROUP_ANALYSIS_DIR/melodic/group_templates/$template_name
filelist=$TEMPLATE_DIR/.filelist_$template_name
DR_DIR=$TEMPLATE_DIR/dr

echo "start DR SORT !!"
. $GLOBAL_GROUP_SCRIPT_DIR/dual_regression_sort.sh $TEMPLATE_MELODIC_IC 1 $DR_DIR `cat
$filelist`

echo "start DR SPLIT 2 SINGLE ICs !!"
. $GLOBAL_GROUP_SCRIPT_DIR/dual_regression_split2singleIC.sh $TEMPLATE_MELODIC_IC $DR_DIR
$DR_DIR $NUM_SUBJECTS "$str_pruning_ic_id" "$str_arr_IC_labels"

str_folders="$DR_DIR/${arr_IC_labels[0]}"
for ic in ${arr_IC_labels[@]:1}
do
  str_folders="$str_folders $DR_DIR/$ic"
done

. $MULTICORE_SCRIPT_DIR/define_thread_processes.sh $NUM_CPU $EXECUTE_STATS_SH "$str_folders"
$PROJ_DIR -nperm $NUM_PERM -maskf $GLOBAL_DATA_TEMPLATES/gray_matter/mask_T1_gray_4mm.nii.gz

mkdir -p $TEMPLATE_DIR/mask

for ic in ${arr_IC_labels[@]}
do
  input_file=$DR_DIR/$ic/mean/$ic"_mean_mask_tfce_corrp_tstat1.nii.gz"
  output_file=$TEMPLATE_DIR/mask/"mask_"$RSN_LABEL.nii.gz
  fslmaths $input_file -thr 0.998 -bin $output_file
done
```

```
rm -rf $DR_DIR
```

The last parameter is represented by a mask of gray-matter only in standard space resampled to 4mm. This in order to obtain mask representing the mean RSN map limited to gray-matter

NOTE, differences between the two masks: The latter mask corresponds to the group-melodic RSN image seen in the corresponding web page. The former instead is larger, including areas which are not usually described in the literature as belonging to that network. Nevertheless, some reviewer might not appreciate the smaller mask as you limit your analysis running the risk to lose some unexpected activation. The larger map should be accepted universally.

3: dual regression and RSN splitting

Note: All the following analyses are template-dependent. You must repeat each of the following steps for every template used.

This step is the core process of all the analysis. It basically analyzes each filtered_func_data image trying to reconstruct the subject version of the independent components found in the specified template. Then it sorts these subjects component in the same order as the template and creates a 4D image (the fourth dimension is the subjects' one) for each component. Additionally to normal dual regression script, as defined by fsl, here components are explicitly splitted into RSN of interest (chosen by the user and defined in the template script file) and stored in specific folders.
$PROJ_GROUP_ANALYSIS_DIR/melodic/dr/templ_XXX/population_XX/RSN1

In the project script you must define these variables:

```
SESS_ID=1
. $PROJ_SCRIPT_DIR/subjects_list.sh                        # load subjects list variables

SUBJECTS_INPUT_RS_DIR_NAME=resting
SUBJECTS_INPUT_RS_NAME=resting

#===== operations ========================================
DO_SORT=1
DO_SPLIT=1

. $GLOBAL_SCRIPT_DIR/melodic_templates/belgrade_controls.sh  # load template-related variables
NUM_SUBJECTS=78                                              # preserved for backward compatibility

out_population_name=controls_md_skip4vol
CTRL_SUBJECTS_DIR=/. . . /belgrade_controls/subjects # when using subjects from different projects
#=========================================
```

a) dual regression over a specific population (sort)

In this stage you also define which kind of analysis you will do, mainly which population(s) you will investigate and/or compare. For example consider a 3 groups study, here you define if compare groups A vs B vs C, A vs B, A vs C or B vs C. for each of this comparison you will create a specific folder

PROJ_GROUP_ANALYSIS_DIR/melodic/dr/templ_XXX/population_XX

This step basically sorts subjects IC according to template schema. It creates #IC files dr_stage2_ic00XX.nii.gz containing one volume for each subject.

```
DR_DIR=$PROJ_GROUP_ANALYSIS_DIR/melodic/dr/templ_$template_name/$out_population_name

if [ $DO_SORT -eq 1 ]
then
  mkdir -p $DR_DIR
  filelist=$DR_DIR/.filelist_$out_population_name

  echo "creating file lists"
  for SUBJ_NAME in ${arr_controls_md[@]}
  do
    echo
"$CTRL_SUBJECTS_DIR/$SUBJ_NAME/s$SESS_ID/$SUBJECTS_INPUT_RS_DIR_NAME/$SUBJECTS_INPUT_RS_NAME.ica/reg_st
andard/filtered_func_data_skip4vol" >> $filelist
  done

  echo "creating file lists"
  for SUBJ_NAME in ${arr_md_corr[@]}
  do
    . $GLOBAL_SCRIPT_DIR/subject_init_vars.sh
    echo
"$SUBJECT_DIR/$SUBJECTS_INPUT_RS_DIR_NAME/$SUBJECTS_INPUT_RS_NAME.ica/reg_standard/filtered_func_data_s
kip4vol" >> $filelist
  done

  echo "start DR SORT !!"
  . $GLOBAL_GROUP_SCRIPT_DIR/dual_regression_sort.sh $TEMPLATE_MELODIC_IC 1 $DR_DIR `cat $filelist`
fi
```

b) components split

Within a population folder (eg. /../templ_XXX/A_B), it creates a specific folder for each RSN defined in the arr_IC_labels, variable defined in the template script file, and merge all subjects RSN in a 4D file.

PROJ_GROUP_ANALYSIS_DIR/melodic/dr/templ_XXX/population_XX /RSN1/dr_stage2_ic0000.nii.gz

which will be later used by randomise to perform statistical analysis

```
if [ $DO_SPLIT -eq 1 ]
then
  echo "start DR SPLIT 2 SINGLE ICs !!"
  . $GLOBAL_GROUP_SCRIPT_DIR/dual_regression_split2singleIC.sh $TEMPLATE_MELODIC_IC $DR_DIR $DR_DIR
    $NUM_SUBJECTS "$str_pruning_ic_id" "$str_arr_IC_labels"
fi
```

4: statistical analysis

This step consists in performing the statistical analysis, matching the 4D subject data of each RSN against a specific General Linear Model. Hence, you need to define:

- a GLM model
- which RSN networks investigate
- the population folder previously created
- the number of CPU and permutations.
- the mask

The processing steps are:

a) create GLM models, and verify that model.con and model.mat are present. These two files will be searched for by the script once you provide the model file path and name (**without extension**).
b) (**optional**) you can calculate the gray matter 4D file used to correct for gray matter differences
c) do randomize in selected networks

If you want to execute the same GLM to different RSN folder, call the following script

$GLOBAL_GROUP_SCRIPT_DIR/dual_regression_randomize_singleIC_multiple_folders.sh

You will obtain your results in => input_folder/analysis_name

```
NUM_CPU=2
NUM_PERM=5000
EXECUTE_STATS_SH=$GLOBAL_GROUP_SCRIPT_DIR/dual_regression_randomize_singleIC_multiple_folder
s.sh
#-------------------------------------------------------------------------------------------
----------
. $GLOBAL_SCRIPT_DIR/melodic_templates/belgrade_controls.sh # define templates variables

in_population_name=dyt_b_st_wc_skip4vol       # as defined in step3 (sorting)
$out_population_name
in_GLM_FILE=$PROJ_SCRIPT_DIR/glm/b_st_wc_x_age       # as defined by GLM program
out_analysis_name=dyt_b_st_wc_maskrsn         #
group_analysis/melodic/dr/$in_population_name
  /RSN_LABEL/$out_analysis_name

ANALYSIS_OUTPUT_DIR_ROOT=$PROJ_GROUP_ANALYSIS_DIR/melodic/dr/templ_$template_name/$in_popula
tion_name
# do STATS !!
#arr_IC_labels=(DMN)   # remove the "#" to restrict analysis to some networks, or change
analysis order

str_folders="$ANALYSIS_OUTPUT_DIR_ROOT/${arr_IC_labels[0]}"
for ic in ${arr_IC_labels[@]:1}
do
  str_folders="$str_folders $ANALYSIS_OUTPUT_DIR_ROOT/$ic"
done

# use default dual regression derived mask
. $MULTICORE_SCRIPT_DIR/define_thread_processes.sh $NUM_CPU $EXECUTE_STATS_SH "$str_folders"
$PROJ_DIR -model $GLM_FILE -nperm $NUM_PERM -odn $out_analysis_name
# specify a file mask
. $MULTICORE_SCRIPT_DIR/define_thread_processes.sh $NUM_CPU $EXECUTE_STATS_SH "$str_folders"
$PROJ_DIR -model $GLM_FILE -nperm $NUM_PERM -odn $out_analysis_name -maskf
$path_to_specific_mask
# specify a folder which must  contain several files called   mask_RSNLABEL.nii.gz
. $MULTICORE_SCRIPT_DIR/define_thread_processes.sh $NUM_CPU $EXECUTE_STATS_SH "$str_folders"
$PROJ_DIR -model $GLM_FILE -nperm $NUM_PERM -odn $out_analysis_name -maskd
$TEMPLATE_MASK_FOLDER
# GM correction... insert GLM column and path to gm demeaned 4d_file.  N.B. if 2 want to use
a default mask: write "mask"
. $MULTICORE_SCRIPT_DIR/define_thread_processes.sh $NUM_CPU $EXECUTE_STATS_SH "$str_folders"
$PROJ_DIR -model $GLM_FILE -nperm $NUM_PERM -odn $out_analysis_name -vxl 3 -vxf
$path_to_gm_demeaned_4d_file
```

```
wait
```

It assumes that:

1) input_folder (#1) contains the input path with the last folder corresponding to the RSN label.
thus extract last folder name and use it as prefix for creating output dr_stage3 files.

2) by default it masks the analysis using $INPUT_DIR/mask.nii.gz . Then it's possible to define:

-maskf: uses a specific file as mask
-maskd: specify a folder that **must** contain "mask_$RSN_LABEL.nii.gz

The output of this step is a set of corrected and uncorrected zstat image for each of the contrasts present in the GLM. The results file name is composed by:
RSN_LABEL"_"GLM_name_masktype

5: results visualization

A proper global script called: show_dr_results_in_singleIC_subfolders.sh

is available to search and visualize all the analysis results, contained within a RSN folder, which pass the requested level of significance. It also calculates the activation parameters by mean of the FSL *cluster* command, which calculate the position and the Z score of both maxima and centre-of-gravity. The script needs:

- the input folder
- the type of searched images (corrected or uncorrected)
- the name of the output text file where it stores the results.
- the background image
- the requested significance value
- the type of searched images.

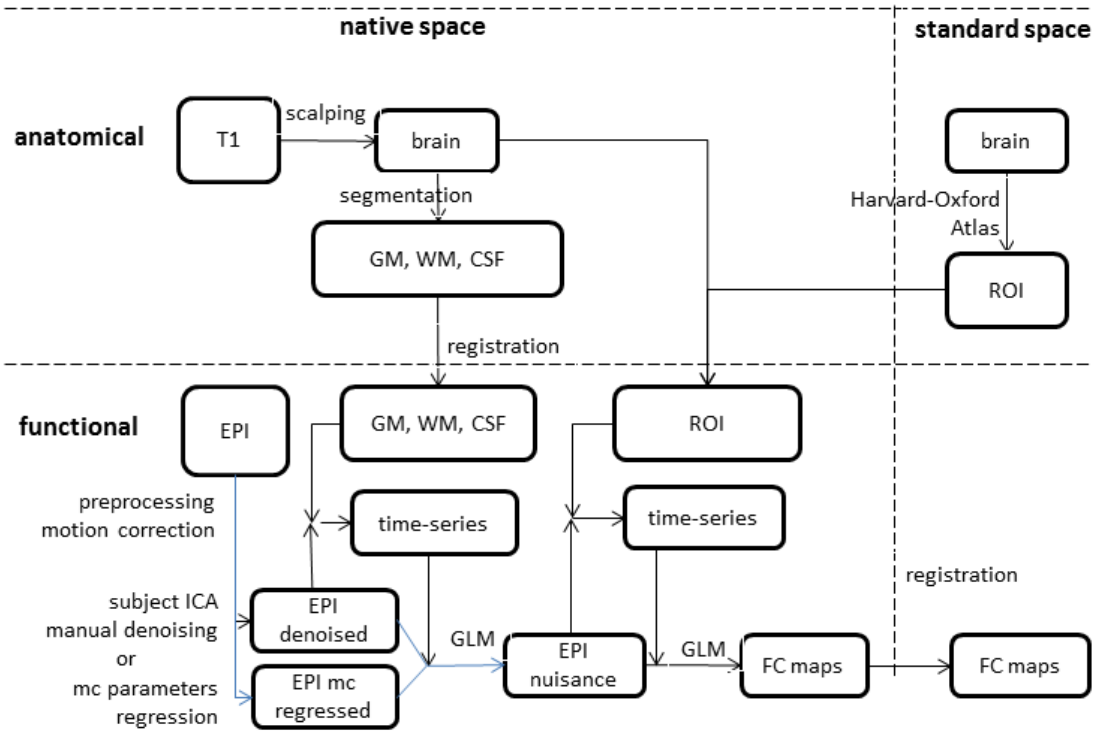
```
#============================================
SHOW_IMAGE=1
WRITE_ONLY_SIGNIFICANT=1
THRESH_VALUE="0.95"
stats_type="corrp"

. $GLOBAL_SCRIPT_DIR/melodic_templates/controls_belgrade18_cab45_earlypd_skip4vol.sh
POPULATION_DIR=controls28_pd66_denoised

# results file is written by default in $MELODIC_POPULATION_DIR/results/RSN_$ICLABEL
#=================================================================
MELODIC_POPULATION_DIR=$PROJ_GROUP_ANALYSIS_DIR/melodic/dr/templ_$template_name/$POPULATION_
DIR
for IC_NAME in ${arr_IC_labels[@]}
do
  INPUT_FOLDER=$MELODIC_POPULATION_DIR/$IC_NAME
```

```
  . $GLOBAL_SCRIPT_DIR/utility/show_dr_results_in_singleIC_subfolders.sh $INPUT_FOLDER -
bgimg $TEMPLATE_BG_IMAGE -ifn "$stats_type" -shimg $SHOW_IMAGE -wrsign
$WRITE_ONLY_SIGNIFICANT -thr $THRESH_VALUE
done
```

SBFC : Seed-based functional connectivity with FEAT

The melodic package allows you to evaluate the **whole-brain** functional connectivity among one or more region-of-interest (ROI) and the rest of the brain. The output of this method are functional connectivity maps (FC), at either individual or group level, representing those voxels whom bold signal time-series (their temporal evolution) are correlated with the ROI's one. There are two possible approaches to pre-process the data, a) calculate motion correction and regress out motion effect over the data using the FEAT module, or b) perform a subject-level MELODIC analysis and manually denoise movement (and non-movement) related components. Moreover, before calculating the functional connectivity of a ROI with the rest of the brain, it is necessary to regress out the confound signals generated by white matter, csf and the whole brain, then subject-level FC maps can be generated.

The processing pipeline include: o) subject welcome, i) subject pre-processing with either FEAT or MELODIC, ii) confounds signals regression, iii) roi creation, iv) single/multiple roi(s) functional connectivity, v) group-level statistical analysis.



1-2: motion pre-processing and nuisance signal regression

The step 1 and 2 described in the analysis pipeline are performed by a single script. Nevertheless, there are two different approaches for doing this step, which regards the way the movement-related artifacts

are removed from the analysis. In the conventional approach (as used for example by the fc1000 projects) you can use FEAT to calculate motion-correction and add the calculated motion parameters to multiple regression GLM that remove their effect from the data. The second approach involves instead the execution of a subject-level MELODIC and a manual denoising using the regfilt function. Both these steps require the presence of two template fsf files which contain some general settings valid for all the subjects.
After having corrected for motion, the effect of the nuisance signals must be regressed out by the data. This can be accomplished with the FEAT module using the output of the previous step. Since such correction is better performed in the native space, WM, CSF, and whole brain masks derived from T1 segmentation must be coregistered to epi native space. A proper script has been designed to perform these steps. The final output of this analysis is a residual 4D files which contains the bold signal after having regressed out the effect of movement artifacts and confound signals. Such file will be stored in $RSFC_DIR and be called (by default) nuisance_10000.nii.gz

a: Motion parameters regression with FEAT

Template creation:

Open a FEAT window and select First-level analysis & Pre-stats + Stats

Data tab:

- the TR value of the sequence
- the high-pass filter cutoff (between 100-150)

“Select 4D data” and “Output directory” will be overwritten by the script.

Pre-stats tab:

you must select:
Motion correction: MCFLIRT
BET brain extraction
Spatial Smoothing FWHM: 5/6 mm
Temporal Filtering: Highpass

Stats tab:

Select:
Use FILM prewhitening
Add motion parameters to model

Press: “Full model setup”, Create a model with a single dummy EV, selecting as Basic shape: Empty (all zeros) and one contrast with filled with an “1”.

Registration:

Keep unchecked.

Usage

A global multi-threaded script, called rsfc_motion_nuisance_feat.sh, is available to perform such analysis.
You have to define the number of CPU, the script name, the subjects array (in string version). The script will i) perform a FEAT to regress out the motion parameters calculated with MCFLIRT, ii) extract their mean time-series, writing the corresponding text files in the `$RSFC_DIR/series` folder.

```
. $PROJ_SCRIPT_DIR/subjects_list.sh

NUM_CPU=1
EXECUTE_SH=$GLOBAL_SCRIPT_DIR/process_subject/rsfc_motion_nuisance_feat.sh
. $MULTICORE_SCRIPT_DIR/define_thread_processes.sh $NUM_CPU $EXECUTE_SH "$str_arr_subj"
$PROJ_DIR
```

Optionally you can decide to process a different *data* from the standard one. In order to do this you can add a further parameter. The script will use the $RS_DIR/$INPUT_NAME.

```
ALTERNATIVE_INPUT_NAME="resting_skip4vol"
. $MULTICORE_SCRIPT_DIR/define_thread_processes.sh $NUM_CPU $EXECUTE_SH "$str_arr_subj"
$PROJ_DIR $ALTERNATIVE_INPUT_NAME
```

b: Melodic denoising

Template creation:

Open a Melodic window

Data / Pre-stats / Registration tabs :

Same as method a).

Stats tab:

Select:
Variance-normalize timecourses
Automatic dimensionality estimation.
Single-session ICA

Post-stats tab:

Select:
Threshold IC maps 0.5
Background image: Mean highres

NOTE: After having saved the fsf template, user must manually edit such file, modifying the TE value of the sequence.

1b: manual denoising after MELODIC

Instead of regressing out the motion parameters, for example when you suspect the presence of strong artifact, either related or not to head movements, it is possible to perform a deeper artifact removal procedure using the MELODIC package. The procedure is realized by executing a single-subject melodic processing, visually inspecting its output, taking note of the artefactual components id and invoking the fsl_regfilt command which remove those components from the signal and create a denoised file.

Subject-level melodic is implemented through a multi-threated global script, in the project script you must define these information:
- number of CPU to be used
- the global script ($GLOBAL_SUBJECT_SCRIPT_DIR/execute_subject_melodic.sh)
- fsf template previously created
- subjects' list string name (variable defined in $PROJ_SCRIPT_DIR/subjects_list.sh)

```
. $PROJ_SCRIPT_DIR/subjects_list.sh
EXECUTE_SH=$GLOBAL_SUBJECT_SCRIPT_DIR/execute_subject_melodic.sh
melodic_fsf_template=$PROJ_SCRIPT_DIR/glm/singlesubj_melodic
declare -i NUM_CPU=2
. $MULTICORE_SCRIPT_DIR/define_thread_processes.sh $NUM_CPU $EXECUTE_SH "$arr_subj"
$PROJ_DIR -model $melodic_fsf_template
```

The output of such analysis is a folder (SUBJECTS_DIR/SUBJ_NAME/resting/resting.ica) containing the subject-level analysis of resting state data. In the file ./filtered_func_data.ica/report.html you find the html page showing the calculated independent component.
More detail on melodic analysis can be found in the "melodic_methods.doc" user guide.
In order to denoise your subjects there are two approaches:

- simply correct the rs data without changing its final name (substitute the original file which is in turn renamed as filtered_func_data_original.nii.gz)
- preserve original file name and create a denoised version (filtered_func_data_denoised)

The former is used when you plan to analyze original data and you had just to correct the data of few subjects. On the contrary, the latter approach is used when you plan to denoise all subjects data, thus it creates a reg_standard_denoised folder, later used by dual-regression analysis, for each subject.

Note
When you will correct just few subjects, you should be very conservative: in case of doubt keep the IC, it may hide some good signal, and delete only those IC related to subjects movements, which highly differs between subjects. If you instead plan to denoise all the subjects, you can decide to remove other

structured noise like those related to cardiac impulse, blood flow and scanner artifact, which are present in each subjects. In fact, here data won't be analyzed with group melodic (able to individuate IC pattern present in all the subjects), so such kind of artifact might be here removed

- **Usage**

In order to proceed with such analysis user must use a different function respect to normal SBFC pre-processing. You can define the input Compared to the previous mode, in order to compose the input file name, you must specify the melodic ICA output dir (`ICA_DIR_NAME`) and the denoised file name (`INPUT_IMAGE_NAME`). That will be used as follows:

```
ICA_DIR=$RS_DIR/$ICA_DIR_NAME
INPUT_IMAGE=$ICA_DIR/$INPUT_IMAGE_NAME.nii.gz
```

Moreover, a third parameter must be specified (`OUTPUT_POSTFIX_NAME`) in order to:

1) select a different FEAT output folder name to discriminate this analysis from standard one (e.g. not involving denoised data)
2) append the output WM, CSF, BRAIN time series names.

```
csf/wm/global"_"$OUTPUT_POSTFIX_NAME"_ts.txt"
```

3) append output nuisance file (`$RSFC_DIR/nuisance"_"$OUTPUT_POSTFIX_NAME"_10000".nii.gz`)

An example of this call is

```
. $PROJ_SCRIPT_DIR/subjects_list.sh
SESS_ID=1
NUM_CPU=1
EXECUTE_SH=$GLOBAL_SCRIPT_DIR/process_subject/rsfc_nuisance_from_feat.sh

# reads /SUBJ_NAME/resting/resting.ica/filtered_func_data_denoised.nii.gz, writes
$RSFC_DIR/nuisance_denoised_10000.nii.gz
. $MULTICORE_SCRIPT_DIR/define_thread_processes.sh $NUM_CPU $EXECUTE_SH "$str_arr_subj"
$PROJ_DIR -idn resting.ica -ifn filtered_func_data_denoised -odn "denoised"
```

3: ROI creation

There are basically two situations: a) roi derives from group analysis results, b) they are obtained in the subjects native space (most commonly the anatomical one). In both cases, ROI must be registered to subject native space.
The subject preprocessing step created all kinds of cross-modal linear and nonlinear registration, so user just have to simply apply those transformation to starting rois.

4: Functional connectivity maps

This is the final subject-level step, which is performed again with a FEAT analysis. The global script that will implement this process needs a template fsf file that can be created as following.

Usage

There are 2 possible multi-threaded scripts that allows to

1) calculate R-roi FEATs of one roi over S-subjects => rsfc_multiple_subject_several_1roi_feat
2) calculate one FEAT of R-rois over S-subjects => rsfc_multiple_subject_1multiroi_feat

Each of them are multi-threaded scripts, which extracts the ROI timeseries from the input image (usually $RSFC_DIR/nuisance_10000) using the input roi as binary masks. Input roi are expected to be stored in a subfolder of $ROI_DIR. Although this process is usually done in the EPI space, in order to be more versatile, the reg_epi subfolder must be specified in the project script. The script will append the relative path of the input roi mask (reg_epi/mask_t_thal_epi.nii.gz) to $ROI_DIR

calculate R-roi FEATs of one roi over S-subjects

"str_arr_subjects" usual string containing the list of subjects
$PROJ_DIR: usual project directory
-model: full path of alternative model, the default one used by the script is normally the right one
-ifn: ALTERNATIVE_INPUT_FILE_NAME. file stored in $RSFC_DIR (e.g. nuisance_denoised)
-son: OUTPUT_SERIES_POSTFIX_NAME, name appended to rois timeseries
<….> At the end of these parameters, you must specify all the ROIs names.

The final OUTPUT feat folder name is defined in this way:
feat_$ROINAME"_"$OUTPUT_SERIES_POSTFIX_NAME

This is an example.

```
SESS_ID=1
NUM_CPU=2
EXECUTE_SH=$GLOBAL_SCRIPT_DIR/process_subject/rsfc_multiple_subject_several_1roi_feat.sh

# base name of ROI: final name used by the script will be
$ROI_DIR/reg_epi/mask_ROINAME_epi.nii.gz
declare -a arr_roi=(l_caudate_hos_fsl l_pallidum_hos_fsl l_putamen_hos_fsl
l_thalamus_hos_fsl)

ALTERNATIVE_TEMPL_FSF=$PROJ_SCRIPT_DIR/glm/templates/template_feat_roi.fsf  # can be
omitted...as it is already the default template used by the script

# alternative call: define input file name and output series postfix name

OUTPUT_DIR_NAME2=roi_left_caud_pall_put_thal_ortho_denoised
ALTERNATIVE_INPUT_NUISANCE_FILE="nuisance_denoised_10000 "
OUTPUT_SERIES_POSTFIX_NAME="denoised" # "skip4vol"
#================================================================================
declare -a final_roi=()
declare -i cnt=0

for roi in ${arr_roi[@]};
do
  final_roi[cnt]=reg_epi/mask_$roi"_epi.nii.gz"
  cnt=$cnt+1
```

```
done

## !!!!! the OUTPUT feat folder name is defined in this way:
feat_$ROINAME"_"$OUTPUT_SERIES_POSTFIX_NAME

# default call: read $RSFC_DIR/nuisance_10000.nii.gz and use template_feat_roi.fsf
. $MULTICORE_SCRIPT_DIR/define_thread_processes.sh $NUM_CPU $EXECUTE_SH "$arr_subj"
$PROJ_DIR ${final_roi[@]} # -model $ALTERNATIVE_TEMPL_FSF if u want a special feat setup
# default call but with a custom template : read $RSFC_DIR/nuisance_10000.nii.gz and use
ALTERNATIVE_TEMPL_FSF
. $MULTICORE_SCRIPT_DIR/define_thread_processes.sh $NUM_CPU $EXECUTE_SH "$arr_subj"
$PROJ_DIR -model $ALTERNATIVE_TEMPL_FSF ${final_roi[@]}
# alternative call: read $RSFC_DIR/nuisance_denoised_10000.nii.gz
. $MULTICORE_SCRIPT_DIR/define_thread_processes.sh $NUM_CPU $EXECUTE_SH "$arr_subj"
$PROJ_DIR -ifn $ALTERNATIVE_INPUT_NUISANCE_FILE -son $OUTPUT_SERIES_POSTFIX_NAME
${final_roi[@]}
wait
```

calculate one FEAT of R-rois over S-subjects

“str_arr_subjects” usual string containing the list of subjects
$PROJ_DIR: usual project directory
-model: full path of model templates
-odn: OUTPUT_DIR_NAME, name appended to $PROJ_GROUP_ANALYSIS_DIR/sbfc/
-ifn: ALTERNATIVE_INPUT_FILE_NAME. file stored in $RSFC_DIR (e.g. nuisance_denoised)
-son: OUTPUT_SERIES_POSTFIX_NAME, name appended to rois timeseries
<….> At the end of these parameters, you must specify all the ROIs names.

This is an example.

```
SESS_ID=1
NUM_CPU=2
EXECUTE_SH=$GLOBAL_SCRIPT_DIR/process_subject/rsfc_multiple_subject_1multiroi_feat.sh

# base name of ROI: final name used by the script will be
$ROI_DIR/reg_epi/mask_ROINAME_epi.nii.gz
declare -a arr_roi=(l_caudate_hos_fsl l_pallidum_hos_fsl l_putamen_hos_fsl
l_thalamus_hos_fsl)

TEMPL_FSF=$PROJ_SCRIPT_DIR/glm/templates/template_feat_4roi_ortho

# standard call: define output dir name
OUTPUT_DIR_NAME=roi_left_caud_pall_put_thal_ortho

# alternative call: define output dir name, input file name and output series postfix name
OUTPUT_SERIES_POSTFIX_NAME="denoised"
ALTERNATIVE_OUTPUT_DIR_NAME=roi_left_caud_pall_put_thal_ortho_denoised
ALTERNATIVE_INPUT_NUISANCE_FILE="nuisance_denoised_10000 "
OUTPUT_SERIES_POSTFIX_NAME="denoised"
#================================================================================
declare -a final_roi=()
declare -i cnt=0

for roi in ${arr_roi[@]};
do
  final_roi[cnt]=reg_epi/mask_$roi"_epi.nii.gz"
  cnt=$cnt+1
```

```
done

# default call: read $RSFC_DIR/nuisance_10000.nii.gz
. $MULTICORE_SCRIPT_DIR/define_thread_processes.sh $NUM_CPU $EXECUTE_SH "$arr_subj"
$PROJ_DIR -model $TEMPL_FSF -odn $OUTPUT_DIR_NAME ${final_roi[@]}
# default call: read $RSFC_DIR/nuisance_denoised_10000.nii.gz
. $MULTICORE_SCRIPT_DIR/define_thread_processes.sh $NUM_CPU $EXECUTE_SH "$arr_subj"
$PROJ_DIR -ifn $ALTERNATIVE_INPUT_NUISANCE_FILE -model $TEMPL_FSF -odn
$ALTERNATIVE_OUTPUT_DIR_NAME -son $OUTPUT_SERIES_POSTFIX_NAME ${final_roi[@]}
wait
```

5: Group level analysis

Group analysis is performed with another FEAT analysis. User must first create one or more FEAT fsf template files, indicating the threshold masking option and the GLM group models.
Then two global group scripts are available performing:

- Test multiple GLM models over one 1st-level analysis
- Execute the same GLM model over N 1st-level analyses

Test multiple GLM models over one 1st-level analysis

User must define an array of FEAT fsf file (containing masking options and group GLMs), provide an 1st level input folder name and an output group folder name,

"str_arr_fsf" string containing the full path of several fsf files.
$PROJ_DIR: usual project directory
-odp: OUTPUT_DIR, full path of output folder
-ncope: number of copes contained in the first level folders.
<….> At the end of these parameters, you must specify the full path of all the first level analyses

```
SESS_ID=1
NUM_CPU=1
EXECUTE_SH=$GLOBAL_SCRIPT_DIR/process_group/rsfc_multiple_model_group_feat.sh

. $PROJ_SCRIPT_DIR/subjects_list.sh

INPUT_1stlevel_DIR="roi_right_caud_pall_put_thal_ortho_denoised"

OUTPUT_DIR=$PROJ_GROUP_ANALYSIS_DIR/sbfc/$INPUT_1stlevel_DIR
declare -a
arr_fsf_templates=($PROJ_SCRIPT_DIR/glm/templates/groupfeat_ctrl28_treated45_naive21_maskgm)
str_arr_fsf_templates=`echo ${arr_fsf_templates[@]}`

CONTROLS_SUBJ_DIR=/media/data/MRI/projects/CAB/fsl_resting_belgrade_controls/subjects

# create 1st level feat dir list
first_level_feat_paths=""

for SUBJ_NAME in ${arr_controls28[@]}
do
  first_level_feat_paths="$first_level_feat_paths
$CONTROLS_SUBJ_DIR/$SUBJ_NAME/s$SESS_ID/resting/fc/feat/$INPUT_1stlevel_DIR"
```

```
done

for SUBJ_NAME in ${arr_treated45[@]}
do
  . $GLOBAL_SCRIPT_DIR/subject_init_vars.sh first_level_feat_paths="$first_level_feat_paths
$RSFC_DIR/feat/$INPUT_1stlevel_DIR"
done

for SUBJ_NAME in ${arr_naive21[@]}
do
  . $GLOBAL_SCRIPT_DIR/subject_init_vars.sh
   first_level_feat_paths="$first_level_feat_paths $RSFC_DIR/feat/$INPUT_1stlevel_DIR"
done

#============================================================================================
. $MULTICORE_SCRIPT_DIR/define_thread_processes.sh $NUM_CPU $EXECUTE_SH
"$str_arr_fsf_templates" $PROJ_DIR -odp $OUTPUT_DIR -ncope 8 $first_level_feat_paths
wait
```

Execute the same GLM model over N 1st-level analyses

"str_arr_1stlvl_feat_name" string containing the name of the 1st level feat analysis
$PROJ_DIR: usual project directory
-odp: OUTPUT_DIR, full path of output folder
-ncope: number of copes contained in the first level folders.
-model: full path of the template fsf file
<….> At the end of these parameters, you must specify the subjects' directory that contains the 1st level analyses subfolders

```
SESS_ID=1
NUM_CPU=1
EXECUTE_SH=$GLOBAL_SCRIPT_DIR/process_group/rsfc_multiple_roi_group_feat.sh

. $PROJ_SCRIPT_DIR/subjects_list.sh

declare -a arr_1stlevel_input_roi=(roi_right_caud roi_right_pall roi_right_put
roi_right_thal)
str_arr_1stlevel_input_roi=`echo ${arr_1stlevel_input_roi[@]}`

OUTPUT_DIR=$PROJ_GROUP_ANALYSIS_DIR/rsfc/ctrl_treated_naive
fsf_template=$PROJ_SCRIPT_DIR/glm/templates/groupfeat_ctrl28_treated45_naive21_maskgm

CONTROLS_SUBJ_DIR=/media/data/MRI/projects/CAB/fsl_resting_belgrade_controls/subjects

# create 1st level feat roots list
first_level_feat_roots=""

for SUBJ_NAME in ${arr_controls28[@]}
do
  first_level_feat_roots="$first_level_feat_roots
$CONTROLS_SUBJ_DIR/$SUBJ_NAME/s$SESS_ID/resting/fc/feat"
done

for SUBJ_NAME in ${arr_treated45[@]}
do
  . $GLOBAL_SCRIPT_DIR/subject_init_vars.sh
```

```
  first_level_feat_roots="first_level_feat_roots $RSFC_DIR/feat"
done

for SUBJ_NAME in ${arr_naive21[@]}
do
  . $GLOBAL_SCRIPT_DIR/subject_init_vars.sh
  first_level_feat_roots="$first_level_feat_roots $RSFC_DIR/feat"
done

#================================================================================
. $MULTICORE_SCRIPT_DIR/define_thread_processes.sh $NUM_CPU $EXECUTE_SH
"$str_arr_1stlevel_input_roi" $PROJ_DIR -model $fsf_template -odp $OUTPUT_DIR -ncope 8
$first_level_feat_roots
wait
```

Tractography

Probtrackx

A main multi-threaded script, called dti_multiple_subject_probrtackx.sh is available to perform probtrackx analysis. Through its input parameters is possible to define all (most of) the standard operations, that is, user can define seed and stop masks, define several waypoints and the file containing target images for Classification Targets approach.

The script allows user to define absolute or relative (to $ROI_DIR) paths separately for all the involved images. That is you can set some path as absolute, some other as relative. For example, if dti images are in the standard space, you can have co-registered the seed using subject's T1 and T2 and thus store them in reg_dti folder but you may use a common image for stop mask.

Three further operations are included:

a) If Classification Target procedure is selected, the function "find_the_biggest" is automatically called.
b) NORMALIZATION
   By default, the script create a normalized version of fdt_paths file (fdt_paths_norm) by dividing the original file by the number of tracts contained in the waypoint file
c) THRESHOLDING
   If you provide a further parameters (-thrP) followed by a comma-separated list of N values, the script perform N thrP thresholding of fdt_paths_norm file and move the resulting N masks in ROI_DIR/reg_dti folder.

The input parameters of the script are coded as follows:

```
-idn)         INPUT_MERGED_DIR, appended to $BEDPOSTX_DIR
-odn)         OUTPUT_DIR_NAME, appended to $PROBTRACKX_DIR
-mask)        mask file path relative to $ROI_DIR
-maskp)              full path of mask file
-seed)        seed file path relative to $ROI_DIR
-seedp)              full path of seed file
-stop)        stop file path relative to $ROI_DIR
-stopp)              full path of stop file
-target)      TARGET_FILE=$2
-targetp)     full path of target
-wp)          define that waypoints file listed must be considered full paths
-thrP)               "20,30,40" list of -thrP values to be applied to fdt_paths_norm

All the remaining parameters are the list of waypoints file to be used, if -wp is set, they
are full paths, otherwise they are path relative to $ROI_DIR

declare -a WP_FILES=( "$@" )
```

Here follow an example of Classification target analysis:

```
SESS_ID=1
. $PROJ_SCRIPT_DIR/subjects_list.sh
BASH_SCRIPT=$GLOBAL_SUBJECT_SCRIPT_DIR/dti_multiple_subject_probtrackx.sh
NUM_CPU=1
#================================================

DO_OVERWRITE_TARGET_FILE=0

SEED_IMAGE_r=reg_dti/mask_R_Thal_dti.nii.gz
SEED_IMAGE_l=reg_dti/mask_L_Thal_dti.nii.gz

STOP_IMAGE_r=$GLOBAL_DATA_TEMPLATES/gray_matter/MNI152_T1_2mm_brain_left.nii.gz
STOP_IMAGE_l=$GLOBAL_DATA_TEMPLATES/gray_matter/MNI152_T1_2mm_brain_right.nii.gz

OUTPUT_DIR_NAME_r=r_thalamus_to_8lobes
OUTPUT_DIR_NAME_l=l_thalamus_to_8lobes

# ----- target list ------------------------------------------------------------------------
----------
input_roi_dir=$GLOBAL_SCRIPT_DIR/data_templates/roi/2mm/lobes
target_list_file_r=$input_roi_dir/r_8lobes_list.txt
target_list_file_l=$input_roi_dir/l_8lobes_list.txt
declare -a target_image_list_r=(r_mask_1_pfc r_mask_2_premotor r_mask_3_precentral
r_mask_4_postcentral r_mask_5_parietal_lobes r_mask_6_temporal_lobes r_mask_7_tempoccip
r_mask_8_occipital_lobes)
declare -a target_image_list_l=(l_mask_1_pfc l_mask_2_premotor l_mask_3_precentral
l_mask_4_postcentral l_mask_5_parietal_lobes l_mask_6_temporal_lobes l_mask_7_tempoccip
l_mask_8_occipital_lobes)

if [ ! -f $target_list_file_r -o $DO_OVERWRITE_TARGET_FILE -eq 1 ]; then
  echo "$input_roi_dir/${target_image_list_r[0]}.nii.gz" > $target_list_file_r
  for f in ${target_image_list_r[@]:1}; do echo "$input_roi_dir/$f.nii.gz" >>
$target_list_file_r; done
fi

if [ ! -f $target_list_file_l -o $DO_OVERWRITE_TARGET_FILE -eq 1 ]; then
  echo "$input_roi_dir/${target_image_list_l[0]}.nii.gz" > $target_list_file_l
  for f in ${target_image_list_l[@]:1}; do echo "$input_roi_dir/$f.nii.gz" >>
$target_list_file_l; done
fi
#================================================================================================
==========
. $MULTICORE_SCRIPT_DIR/define_thread_processes.sh $NUM_CPU $BASH_SCRIPT "$str_arr_pd65"
$PROJ_DIR -odn $OUTPUT_DIR_NAME_r -maskp "mask" -seed $SEED_IMAGE_r -targetp
$target_list_file_r -stopp $STOP_IMAGE_r
wait
. $MULTICORE_SCRIPT_DIR/define_thread_processes.sh $NUM_CPU $BASH_SCRIPT "$str_arr_pd65"
$PROJ_DIR -odn $OUTPUT_DIR_NAME_l -maskp "mask" -seed $SEED_IMAGE_l -targetp
$target_list_file_l -stopp $STOP_IMAGE_l
wait

declare -i cnt=0
for SUBJ_NAME in ${arr_pd65[@]}
do
  . $GLOBAL_SCRIPT_DIR/subject_init_vars.sh
  cnt=1
  for ROI_NAME in ${target_image_list_r[@]}
  do
```

```
    $FSLDIR/bin/fslmaths $PROBTRACKX_DIR/r_thalamus_to_8lobes/biggest -thr $cnt -uthr $cnt -
bin $ROI_DIR/reg_dti/$ROI_NAME
    cnt=$cnt+1
  done
  cnt=1
  for ROI_NAME in ${target_image_list_l[@]}
  do
    $FSLDIR/bin/fslmaths $PROBTRACKX_DIR/l_thalamus_to_8lobes/biggest -thr $cnt -uthr $cnt -
bin $ROI_DIR/reg_dti/$ROI_NAME
    cnt=$cnt+1
  done
done
```

Calculate mean tract dtifit values

One of two main application of tractography is the possibility to use the reconstructed tract to calculate its mean FA, MD etc values.

```
. $PROJ_SCRIPT_DIR/subjects_list.sh

thrPvalue=20
OUTPUT_DIR_NAME_l=l_cst_ped2mi_2wp_1s
OUTPUT_DIR_NAME_r=r_cst_ped2mi_2wp_1s
mask_thrp_file_name_r=mask_$OUTPUT_DIR_NAME_r"_P"$thrPvalue.nii.gz
mask_thrp_file_name_l=mask_$OUTPUT_DIR_NAME_l"_P"$thrPvalue.nii.gz

for SUBJ_NAME in ${arr_ela_dti[@]}
do
  echo $SUBJ_NAME
  . $GLOBAL_SCRIPT_DIR/subject_init_vars.sh

  # calculate mean FA/MD in CST
  [ ! -f $ROI_DIR/reg_dti/$mask_thrp_file_name_l ] && $FSLDIR/bin/fslmaths
$PROBTRACKX_DIR/$OUTPUT_DIR_NAME_l/fdt_paths_norm.nii.gz -thrP $thrPvalue
$ROI_DIR/reg_dti/$mask_thrp_file_name_l

  $FSLDIR/bin/fslmeants -i $DTI_DIR/$DTI_FIT_LABEL"_FA.nii" -m
$ROI_DIR/reg_dti/$mask_thrp_file_name_l -o
$ROI_DIR/reg_dti/FA_meants_$OUTPUT_DIR_NAME_l"_P"$thrPvalue.txt
      $FSLDIR/bin/fslmeants -i $DTI_DIR/$DTI_FIT_LABEL"_MD.nii" -m
$ROI_DIR/reg_dti/$mask_thrp_file_name_l -o
$ROI_DIR/reg_dti/MD_meants_$OUTPUT_DIR_NAME_l"_P"$thrPvalue.txt

  [ ! -f $ROI_DIR/reg_dti/$mask_thrp_file_name_r ] && $FSLDIR/bin/fslmaths
$PROBTRACKX_DIR/$OUTPUT_DIR_NAME_r/fdt_paths_norm.nii.gz -thrP $thrPvalue
$ROI_DIR/reg_dti/$mask_thrp_file_name_r

  $FSLDIR/bin/fslmeants -i $DTI_DIR/$DTI_FIT_LABEL"_FA.nii" -m
$ROI_DIR/reg_dti/$mask_thrp_file_name_r -o
$ROI_DIR/reg_dti/FA_meants_$OUTPUT_DIR_NAME_r"_P"$thrPvalue.txt
      $FSLDIR/bin/fslmeants -i $DTI_DIR/$DTI_FIT_LABEL"_MD.nii" -m
$ROI_DIR/reg_dti/$mask_thrp_file_name_r -o
$ROI_DIR/reg_dti/MD_meants_$OUTPUT_DIR_NAME_r"_P"$thrPvalue.txt
done
#============================================================================================
==========
# collect subjects' FA & MD means and store in a single file for statistical analysis
mkdir -p $PROJ_GROUP_ANALYSIS_DIR/results
group_fa_means=$PROJ_GROUP_ANALYSIS_DIR/results/group_fa_means.txt
```

```
group_md_means=$PROJ_GROUP_ANALYSIS_DIR/results/group_md_means.txt

echo "subj    r_fa   l_fa" > $group_fa_means
echo "subj    r_md   l_md" > $group_md_means

for SUBJ_NAME in ${arr_ela_dti[@]}
do
  . $GLOBAL_SCRIPT_DIR/subject_init_vars.sh
  wr=$(cat $ROI_DIR/reg_dti/FA_meants_$OUTPUT_DIR_NAME_l"_P20".txt | tr -d ' ')
  wl=$(cat $ROI_DIR/reg_dti/FA_meants_$OUTPUT_DIR_NAME_r"_P20".txt | tr -d ' ')
  echo "$SUBJ_NAME  $wr    $wl" >> $group_fa_means

  wr=$(cat $ROI_DIR/reg_dti/MD_meants_$OUTPUT_DIR_NAME_r"_P20".txt | tr -d ' ')
  wl=$(cat $ROI_DIR/reg_dti/MD_meants_$OUTPUT_DIR_NAME_l"_P20".txt | tr -d ' ')
  echo "$SUBJ_NAME  $wr    $wl" >> $group_md_means
done
```

Appendix

Revisions

0.3:

Changes to group analysis. Added PostModel, Nuisance and Covariates to specify contrasts and Posthoc (multiple comparisons).
Added group analysis example files

Example Files

Cat-thickness group analysis

```
import os
import traceback

from Global import Global
from Project import Project
from group.GroupAnalysis import GroupAnalysis
from group.SPMModels import SPMModels
from group.SPMStatsUtils import Covariate, Nuisance, PostModel

if __name__ == "__main__":

    # ============================================
    # check global data and external toolboxes
    # ============================================
    fsl_code = "604"
    try:
        globaldata = Global(fsl_code)

        # ======================================
        # HEADER
        # ======================================
        proj_dir = "/data/MRI/projects/test"
        project = Project(proj_dir, globaldata)     # automatically load
                                                    # PROJDIR/script/data.dat if present
        SESS_ID = 1
        num_cpu = 1
        group_label = "all"

        # ======================================
        # PROCESSING
        # ======================================
        subjects = project.load_subjects(group_label, SESS_ID)

        analysis            = GroupAnalysis(project)
        spm_analysis        = SPMModels(project)


        # ======================================
        # THICKNESS DATA:
        # ======================================
        groups_instances    = [project.get_subjects("grp1", SESS_ID)]
        covs                = [Nuisance("gender"), Nuisance("age")]
        anal_name           = "multregr_age_gender"
        statsdir            = os.path.join(project.group_analysis_dir, "mpr/thickness")
        post_model          = PostModel("cat_stats_contrasts_results", regressors=covs,
                                        isSpm=False)

      spm_analysis.batchrun_cat_thickness_stats_factdes_1group_multregr(statsdir,anal_name,
                           groups_instances, covs, post_model=post_model, runit=False)


        groups_instances    = [ project.get_subjects("grp1"),
                                project.get_subjects("grp2")]
        covs                = [Nuisance("gender"), Nuisance("age")]
        anal_name           = "2stt_age_gender"
        statsdir            = os.path.join(project.group_analysis_dir, "mpr/thickness")
        post_model          = PostModel("cat_stats_2samples_ttest_contrasts_results",
                                        regressors=covs, isSpm=False)
```

```
    spm_analysis.batchrun_cat_thickness_stats_factdes_2samplesttest(statsdir, anal_name,
                            groups_instances, covs, post_model=post_model, runit=False)


    groups_instances    = [ project.get_subjects("grp1"),
                            project.get_subjects("grp2"),
                            project.get_subjects("grp3")]
    covs                = [Nuisance("gender"), Nuisance("age")]
    anal_name           = "1Wanova_3_groups_age_gender"
    statsdir            = os.path.join(project.group_analysis_dir, "mpr/thickness")
    post_model          = PostModel("fullpath2template", regressors=covs, isSpm=False)
    spm_analysis.batchrun_cat_thickness_stats_factdes_1Wanova(statsdir, anal_name,
                            groups_instances, covs, post_model=post_model, runit=False)


    factors             = {"labels":["f1", "f2"], "cells":
                           [[project.get_subjects("grp1"), project.get_subjects("grp2")],
                           [project.get_subjects("grp3"), project.get_subjects("grp4")]]}
    covs                = [Nuisance("gender"), Nuisance("age")]
    anal_name           = "2Wanova_age_gender"
    statsdir            = os.path.join(project.group_analysis_dir, "mpr/thickness")
    post_model          = PostModel("fullpath2template", regressors=covs, isSpm=False)
    spm_analysis.batchrun_cat_thickness_stats_factdes_2Wanova(statsdir, anal_name,
                                    factors, covs, post_model=post_model, runit=False)


except Exception as e:
    traceback.print_exc()

    print(e)
    exit()
```

spm-dartel group analysis

```
import os
import traceback

from Global import Global
from Project import Project
from group.GroupAnalysis import GroupAnalysis
from group.SPMModels import SPMModels
from group.SPMStatsUtils import Covariate, Nuisance, PostModel, ResultsParams

if __name__ == "__main__":

    # ================================================
    # check global data and external toolboxes
    # ================================================
    fsl_code = "604"
    try:
        globaldata = Global(fsl_code)

        # ================================================
        # HEADER
        # ================================================
        proj_dir = "/data/MRI/projects/test"
        project = Project(proj_dir, globaldata)     # automatically load
                                                     # PROJDIR/script/data.dat if present
        SESS_ID = 1
        num_cpu = 1
        group_label = "all"

        # ================================================
        # PROCESSING
        # ================================================
        subjects = project.load_subjects(group_label, SESS_ID)
        #project.add_icv_to_data(group_label) # add icv to all data

        analysis            = GroupAnalysis(project)
        spm_analysis        = SPMModels(project)

        # template
        vbm_template_name   = "a_template"
        vbm_template_dir    = os.path.join(project.vbm_dir, vbm_template_name) #
         vbm_template_dir    = analysis.create_vbm_spm_template_normalize(vbm_template_name,
                                                                          subjects)

        # STATS

        groups_instances    = [project.get_subjects("grp1", SESS_ID)]
        covs                = [Covariate("gender"), Covariate("age")]
        anal_name           = "multregr_age_gender"
        postmodel           = PostModel("spm_stats_contrasts_results", covs, [],
                                        res_params=ResultsParams("FWE", 0.01, 10))
        spm_analysis.batchrun_spm_vbm_dartel_stats_factdes_1group_multregr(vbm_template_dir,
anal_name, groups_instances, covs, post_model=postmodel, runit=False)


        groups_instances    = [ project.get_subjects("grp1"),project.get_subjects("grp2")]
        covs                = [Nuisance("gender"), Nuisance("age")]
        anal_name           = "2stt_age_gender"
        postmodel           = PostModel("spm_stats_2samples_ttest_contrasts_results",
                                        covs, ["grp1 > grop2", "grp2 > grp1"],
                                        ResultsParams("FWE", 0.01, 10))
        spm_analysis.batchrun_spm_vbm_dartel_stats_factdes_2samplesttest(vbm_template_dir,
```

```
                anal_name, groups_instances, covs, post_model=postmodel, runit=False)


    groups_instances    = [ project.get_subjects("grp1"),
                            project.get_subjects("grp2"),
                            project.get_subjects("grp3")]
    covs                = [Nuisance("gender"), Nuisance("age")]
    anal_name           = "1Wanova_3_groups_age_gender"
    postmodel           = PostModel("fullpath2template", regressors=covs)
    spm_analysis.batchrun_spm_vbm_dartel_stats_factdes_1Wanova(vbm_template_dir,
                 anal_name, groups_instances, covs, post_model=postmodel, runit=False)


    factors    = {"labels":["f1", "f2"], "cells": [[project.get_subjects("grp1"),
                                                    project.get_subjects("grp2")],
                                                   [project.get_subjects("grp3"),
                                                    project.get_subjects("grp4")]]}
    covs                = [Nuisance("gender"), Nuisance("age")]
    anal_name           = "2Wanova_age_gender"
    postmodel           = PostModel("fullpath2template", regressors=covs)
    spm_analysis.batchrun_spm_vbm_dartel_stats_factdes_2Wanova(vbm_template_dir,
                            anal_name, factors, covs, post_model=postmodel, runit=False)


except Exception as e:
    traceback.print_exc()

    print(e)
    exit()
```

cross-projects group analysis

```
import traceback

from Global import Global
from Project import Project
from group.GroupAnalysis import GroupAnalysis
from group.SPMModels import SPMModels

if __name__ == "__main__":

  # ================================================================================
  # check global data and external toolboxes
  # ================================================================================
  fsl_code = "604"
  try:
    globaldata = Global(fsl_code)

    # ================================================================================
    # HEADER
    # ============================================================
    SESS_ID = 1
    ctrl_proj_dir   = "/data/MRI/projects/controls"
    ctrl_project    = Project(ctrl_proj_dir, globaldata)

    pat_proj_dir    = "/data/MRI/projects/patients"
    pat_project     = Project(pat_proj_dir, globaldata)

    group_analysis  = GroupAnalysis(pat_project)  # reference project for group-
                                                     level analysis is the patients' one
    spm_analysis    = SPMModels(pat_project)

    ctrl_group_label    = "ctrl_for_patients_study"     # an age-matched subset of all
                                                           controls
    patient_group_label = "all_patients"                # the patient group of interest
    ctrl_subjects       = ctrl_project.get_subjects(ctrl_group_label, SESS_ID)
    pat_subjects        = ctrl_project.get_subjects(patient_group_label, SESS_ID)

    cov_names           = ["gender", "age"]
    populations_name    = "a_population"
    analysis_name       = "an_analysis"

    all_subjects   = ctrl_subjects.append(pat_subjects)  # create a list concatenating
                                                          controls' list with patient one

    population_dir = group_analysis.create_vbm_spm_template_normalize(populations_name,
                          all_subjects) # create a template of the given population
    post_model     = PostModel("cat_stats_2samples_ttest_contrasts_results",
                               regressors=covs, isSpm=False)

   spm_analysis.batchrun_vbm_dartel_stats_factdes_2samplesttest(population_dir,analysis_name
             ,[ctrl_subjects, pat_subjects], cov_names, spm_contrasts_template_name="")

    except Exception as e:
        traceback.print_exc()

        print(e)
        exit()
```