EPItome-xl

As long as our brain is a mystery, the universe, the reflection of the structure of the brain will also be a mystery.

SANTIAGO RAMÓN Y CAJAL

Joseph D. Viviano
Department of Biology
York University
Toronto, ON, CA
joseph.d.viviano@gmail.com

Contents

1 Introduction

EPItome-xl is a program designed for the flexible construction of MRI pre-processing pipelines, with a focus on functional MRI images and their associated problems. Its primary function is to take BASH modules and chain them together in any way the user desires to create a set of batch-processing scripts for an MRI experiment. These modules are not necessarily dependent on one another, allowing users of this package to easily extend the functionality of EPItome-xl by simply depositing a shell script into the appropriate module folder and writing the associated python wrapper command (and documentation!)

The goal of this design is to allow multiple levels of control for users of different skills with the same interface. EPItome facilitates the construction of very robust pre-processing scripts that can be run on your computer or in a distributed computing environment by only answering a few high-level questions, hopefully making it easy for beginners to get started. The scripts that are output by these commands are otherwise fully tweak-able and well commented – they encourage experimentation. These modified modules could eventually evolve into new features altogether, which are easily added to the existing pool.

This system is also designed to facilitate easy-to-reproduce research, as these scripts can be easily re-purposed for new experiments that follow the EPItome folder structure. In this way, the outputs of EPItome act as your lab notebook, and can be shared with collaborators or reviewers.

This manual will progress to more advanced topics in the end. First, I will explain the basic use of EPItome-xl. Next, I'll walk the user through a few common pre-processing tasks. I'll then explain the modules one-by-one, and finish with an explanation on how to add new modules.

2 Requirements & Dependencies

EPItome contains a small number of programs that actually manipulate data, but also makes heavy use of widely-used MRI analysis tools and a number of python distributions. The user is assumed to have properly installed and configured FSL, AFNI, Freesurfer, and the python packages numpy, scipy, and matplotlib.

The program itself was built and tested on a Ubuntu 12.04 server. I imagine it will work well in any Linux environment. It should run on Mac OS X as well, but this remains unverified. There will be no support for Windows.

3 Setting Up & Using EPItome-xl

Up to date EPItome code & installation instructions are available from github). Briefly,

```
git clone https://github.com/josephdviviano/EPItome-xl.git
easy_install EPItome-xl
```

After installation, EPItome requires you to specify a few paths. These settings can be found in the /epitome/config.py file.

- `dir_data`: should point to your MRI experiment folder.
- `dir_pipe`: should point to your installation of EPItome-xl.
- `dir_afni`: should point to your installation of AFNI.
- `cores`: could be set to the maximum number of cores you wish EPItome to use (defaults to the maximum possible -1).

EPItome-xl comes with a few command-line interfaces. EPItome is used to inspect data in the MRI directory, returns information on the currently-available modules, can be used to construct new pipelines, and to remove unwanted data from the MRI directory cleanly. EPIphysio is a tool built to parse physiological data from the BIOPAK 150 unit we have installed at York University (Toronto), and might need to be adapted / extended to work with other units. EPIfolder is used to generate an appropriate folder structure in the MRI directory for the EPItome pipeline to work on.

The MRI directory itself must be organized as follows:

```
/EPItome
 |__EXPERIMENTS
 |   |__SUBJECTS
 |       |__MODE
 |           |__SESS
 |               |__RUN
 |__FREESURFER
     |__SUBJECTS
```

The Freesurfer subject directory should point to the Freesurfer directory in your MRI folder – this can easily be accomplished using a symbolic link. The remaining folder structure is perhaps best generated by using the included `EPIfolder` tool (or via a home-brew script).

3.1 Folders

The folder structure is integral to EPItome – if it is flawed, the pipeline will fail in mysterious ways. The structure itself is designed to be thought of as a tree. At the roots of the tree are the individual files collected at the scanner. As we ascend the tree, files are combined across sessions, image modalities, and subjects, so one finds experiment-wide outputs at the highest levels. The `EPIfolder` program will help you set up these folders appropriately.

3.1.1 EXPERIMENTS

This is a set of folders containing entire experiments. There are no important naming conventions, but it seems advisable (for consistency) to make the folder names all capitals, and short (e.g., `LINGASD` for 'language study on those with autism spectrum disorder').

3.1.2 SUBJECTS

Once again, these are simply folders with participant names. They follow no convention, but should be consistent for your own sake.

3.1.3 MODE

Image modality folders separate images of different kinds: anatomicals, EPI's collected using differing sequences, or EPI's of different task-types (e.g., rest vs. two-back matching). The T1 directory **must** exist for each subject at the very minimum in `SESS01`.

This is a good place to separate scans you would like to have analyzed in different ways, or to test multiple pre-processing strategies on the same set of subjects. For example, it may be that your `TASK` set is being prepared for a GLM or partial least squares analysis, and should be processed more minimally than your `REST` data, which will undergo things such as low-pass filtering and nuisance variable regression. In another example, it may be that you are curious about how your choice of pre-processing steps influences your results. Here, you could have a set of identical scans under REST_1 and REST_2. You could build two sets of pipelines using EPItome with the unique identifiers '1' and '2', and run them on each modality separately.

EPItome has no modules built for DTI scans at the moment, but they could easily be added here under their own DTI modality. Note that a set of DTI-friendly modules would need to be built for these kinds of scans.

3.1.4 SESS

The session folders are used to separate scans taken on different days. They must begin with SESS and end with a zero-padded 2-digit number (e.g., 02 or 10). EPItome does not currently support experiments where participants were scanned on more than 99 days.

These session folders are currently used to match EPIs with the T1 taken on the same day. Best practice is to collect a T1 with every EPI scan. The pipeline is also able to use the T1 collected on the first day as the target for all sessions. This will be automatically decided by the pipeline: if the number of T1s does not equal the number of sessions, only the first T1 will be used.

IMPORTANT: While it is normally advisable for the sessions to align chronologically, in the case that the only T1 collected was *not* on the first day, it should still be entered as SESS01.

3.1.5 RUN

Each RUN folder should contain one and only one .nii or .nii.gz formatted file. Appropriate companion files should also be entered here: physiological noise recordings (extension .PHYS, and/or custom slice timing files (extension .1D). If more than NIFTI file is in this folder, the pipeline will fail. Any other files kept in this folder will remain untouched, so this is a fine place to keep run-specific notes.

3.2 Running EPItome

EPItome contains a set of helper subroutines and two main functions: 'run' and 'clean'. Typing EPItome into your command line after installation should show each function and a brief description of each, so I won't reiterate that here. I will mention that EPItome check <experiment> allows you to check the total number of raw NIFTIs in the `RUN` folders of each image modality. This allows you to quickly find empty `RUN` folders, and ensure you have properly imported all of your data.

3.2.1 EPItome run

This is the heart and soul of `EPItome`. This command line interface will walk you through the construction of a pipeline for a single image modality within a single experiment. Every run of EPItome begins with a lengthly run of Freesurfer on all T1s, and `init_EPI`, which does the most basic kinds of MRI pre-processing. Following that, you are free to chain together modules as you see fit.

In order to retain the modular and easily-customized structure of EPItome, this program allows you to shoot yourself in the foot. In fact, if you aren't clear on what to do, you are more likely to make a malformed pipeline than you are to making a good one. Therefore, I recommend reading the Pipeline section of this manual at least once, and perhaps skimming the Presets section following, to get a sense of reasonable usage.

The pipeline will allow you to chain together various modules in order until you give it the `stop` command, when it will switch over to asking you to submit a set of desired QC outputs. Internally, this program is simply looking through the `/qc` module directory instead of the `/pre` module directory. These QC outputs will only properly work once all of your subjects are pre-processed, as they output single PDFs detailing some feature of the entire experiment.

Finally, a few outputs will be deposited in your experiment directory: a `master` script, a `proclist` script, and a set of `cmd` scripts. A copy of all the current modules at the time of running will be deposited in an `EPItome` directory in your home folder. The `master` script generates everything else from these copied modules, and can be edited by-hand to produce new pipelines. In fact, those who know what they are doing can generate new pipelines directly from an old `master` script, instead of interacting with the command line interface again. The `cmd` scripts are the actual set of commands run on each participant. These are essentially the module script files concatenated in the way defined by the user with the appropriate variables filled in. These files are meant to be well-commented, and should be easily edited by-hand for those adventurous types who would like to tweak various settings, try new methods, or debug beguiling problems, on a subject wise basis. Alternatively, the modules in your home folder can be edited, the master script re-run, and the changes will be applied to every subject's `cmd` script. The `proclist` is simply a set of calls to these various scripts in order. It can be called directly, to run the subjects serially, or submitted to a batch queuing system to be analyzed in a cluster environment.

EPItome scripts are written to never re-do done work. Therefore, to replace a bad set of outputs with good ones, one must first delete the bad outputs. This can be done by hand, or via a set of helper cleanup scripts, detailed below.

3.2.2 EPItome clean

This program works similarly to `run`, but produces scripts for deleting intermediate files. Generally, it is good practice to inspect the outputs of EPItome first, and if problems are identified, to work backwards through the pipeline until the problem arises to determine the origin of the issue. After the outputs have been vetted, these cleanup scripts will go a long way to keeping your hard drives and system administrators happy.

4 The Pipeline

What follows is a description of what each module in EPItome-xl does. These modules can be easily chained together manually, or by using the command-line interface included with the pipeline. New modules are simply bash scripts which call various programs, including FSL, Freesurfer, AFNI, and custom python programs. Therefore, functionality of EPItome is easily extended without perturbing the function of older modules. Any script found in the modules directories will be added to the command line interface automatically, but will not work properly unless a matching wrappper function is added to `commands.py`.

The types of modules included can be roughly split into 4 categories: freesurfer, pre-processing, quality-control, and cleanup.

4.1 Freesurfer

Right now, the default freesurfer recon-all is run on every participant before further processing. This is to produce surface files that can be used for cortical smoothing / data visualization, and the automatic generation of tissue masks which can be used for the generation of nuisance regressors.

4.1.1 fsrecon.py

Usage: `fsrecon.py <data_directory> <experiment> <modality> <cores>`

> data_directory – full path to your MRI/WORKING directory.
> experiment – name of the experiment being analyzed.
> modality – image modality to import (normally T1).
> cores – number of cores to dedicate (one core per run).

This sends each subject's T1s through the Freesurfer pipeline. It uses multiple T1s per imaging session, but does not combine them between sessions. Data is output to the dedicated FREESURFER directory, and should be exported to the MRI analysis folders using `fsexport.py`.

4.1.2 fsexport.py

Usage: `fsexport.py <data_directory> <experiment>`

> data_directory – full path to your MRI/WORKING directory.
> experiment – name of the experiment being analyzed.

Imports processed T1s from Freesurfer to the experiment directory.

4.2 Pre-Processing

This contains the lion's share of the pipeline. Every run of EPItome begins with `init_EPI`, which contains a non-contentious set of pre-processing steps for EPI images. THe following stages can be chained together at will to preform de-noising, spatial transformations, projections to surface-space, and spatial smoothing.

4.2.1 init_EPI

Usage: `init_EPI <data_quality> <del_tr> <t_pattern> <normalization> <masking>`

> data_quality – 'low' for poor internal contrast, otherwise 'high'.
> del_tr – number of TRs to remove from the beginning of the run.
> t_pattern – slice-timing at acquisition (from AFNI's 3dTshift). normalization – voxel wise time series normalization. One of 'zscore', 'pct', 'demean'.

masking – EPI brain masking tolerance. One of ‘loose’, ‘normal’ ‘tight’.

Works from the raw data in each RUN folder. It performs general pre-processing for all fMRI data:

- Orients data to RAI
- Deletes initial time points (optionally)
- Removes data outliers
- Slice time correction
- Deobliques & motion corrects data
- Creates session mean deskulled EPIs and whole-brain masks
- Scales and optionally normalizes each time series
- Calculates various statistics + time series

Times series normalization can be accomplished in one of two ways: percent signal change, and scaling. For percent signal change, the data is normalized by the mean of each time series to mean = 100. A deviation of 1 from this mean value indicates 1% signal change in the data. This is helpful for analyzing only relative fluctuations in the signal and is best at removing inter-session/subject/run variability, although it can also introduce rare artifacts in small localized regions of the images and may not play well with multivariate techniques such as partial least squares without accounting for these artifacts. Alternatively, one can scale the data, which applies single scaling factor to all voxels such that the global mean of the entire run = 1000. This will help normalize baseline shifts across sessions, runs, and participants. Your selection here might be motivated by personal preference, or in rarer cases, analytic requirements. When in doubt, it is safe to select ‘off’, as scaling can be done later by hand, or ‘scale’ if one is doing a simple GLM-style analysis. ‘pct’ should be used by those with a good reason.

Masking options are provided to improve masking performance across various acquisition types, but it is very hard to devise a simple one-size fits all solution for this option. Therefore the QC outputs will be very important for ensuring good masking, and these options may need to be tweaked on a site-by-site basis. Luckily, many analysis methods do not rely heavily on mask accuracy. In cases that do, such as partial least squares / ICA / PCA analysis, close attention should be paid to the output of this step. Hopefully the ‘loose’, ‘normal’, and ‘tight’ nomenclature are self-explanatory. Generally, it is best to start with normal, and adjust if required.

Prerequisites: None.

4.2.2 combine_volumes

Usage: `combine_volumes <func1_prefix> <func2_prefix>`

> func1_prefix – functional data prefix (eg., smooth in func_smooth).
> func2_prefix – functional data prefix (eg., smooth in func_smooth).

Combines two functional files via addition. Intended to combine the outputs of `surfsmooth` & `surf2vol` with `volsmooth`, but could be used to combine other things as well. The functional files should not have non-zeroed regions that overlap, or the output won’t make much sense.

Prerequisites: Two EPI modules with unique output prefixes. Intended to be used to combine the outputs of `volsmooth` and `surfsmooth` in a single volume.

4.2.3 linreg_calc_AFNI

Usage: `linreg_calc_AFNI <cost> <reg_dof> <data_quality>`

> cost – cost function minimized during registration.
> reg_dof – ‘big_move’ or ‘giant_move’ (from align_EPI_anat.py).
> data_quality – ‘low’ for poor internal contrast, otherwise ‘high’.

Uses AFNI’s align_EPI_anat.py to calculate linear registration between EPI <–> T1 <–> MNI152, and generate an EPI template registered to T1 & T1 registered to EPI (sessionwise). Specific options can be found in the command-line interface’s help function.

Prerequisites: `init_EPI`.

4.2.4 linreg_calc_FSL

Usage: `linreg_calc_FSL <cost> <reg_dof> <data_quality>`

> cost – cost function minimized during registration (see FSL FLIRT).
> reg_dof – 6, 7, 9, or 12 degrees of freedom (see FSL FLIRT),
> data_quality – ‘low’ for poor internal contrast, otherwise ‘high’.

Uses FSL's FLIRT to calculate linear registration between EPI <–> T1 <–> MNI152, and generate an EPI template registered to T1 & T1 registered to EPI (session-wise). Specific options can be found in the command-line interface's help function.

Prerequisites: `init_EPI`.

4.2.5 linreg_EPI2MNI_AFNI

Usage: `linreg_EPI2MNI_AFNI <func_prefix> <voxel_dims>`

> func_prefix – functional data prefix (eg.,smooth in func_smooth).
> voxel_dims – target voxel dimensions (isotropic).

Prepares data for analysis in MNI standard space.

Prerequisites: `init_EPI`, `linreg_calc_AFNI`.

4.2.6 linreg_EPI2MNI_FSL

Usage: `linreg_EPI2MNI_FSL <func_prefix> <voxel_dims>`

> func_prefix – functional data prefix (eg., smooth in func_smooth).
> voxel_dims – target voxel dimensions (isotropic).

Prepares data for analysis in MNI standard space.

Prerequisites: `init_EPI`, `linreg_calc_FSL`.

4.2.7 linreg_FS2EPI_AFNI

Usage: `linreg_FS2EPI_AFNI`
Brings Freesurfer atlases in register with single-subject EPIs.

Prerequisites: `init_EPI`, `linreg_calc_AFNI`.

4.2.8 linreg_FS2EPI_FSL

Usage: `linreg_FS2EPI_FSL`
Brings Freesurfer atlases in register with single-subject EPIs.

Prerequisites: `init_EPI`, `linreg_calc_FSL`.

4.2.9 linreg_FS2MNI_FSL

Usage: `linreg_FS2MNI_FSL`
Brings Freesurfer atlases in register with MNI standard space.
Prerequisites: `init_EPI`, `linreg_calc_FSL`.

4.2.10 gen_regressors

Usage: `gen_regressors <func_prefix>`

func_prefix – functional data prefix (eg.,smooth in func_smooth).

Creates a series of regressors from fMRI data and a freesurfer segmentation:

- white matter + eroded mask
- ventricles + eroded mask
- grey matter mask
- brain stem mask
- dialated whole-brain mask
- draining vessels mask
- local white matter regressors + 1 temporal lag
- ventricle regressors + 1 temporal lag
- draining vessel regressors + 1 temporal lag

Prerequisites: `init_EPI`, `linreg_calc_AFNI/FSL`, `linreg_FS2EPI_AFNI/FSL`.

4.2.11 gen_gcor

Usage: `gen_gcor <func_prefix>`

func_prefix – functional data prefix (eg.,smooth in func_smooth).

Calls an AFNI script to calculate the global correlation for each concatenated set of runs (across all sessions). Useful for resting state functional connectivity experiments.
Prerequisites: `init_EPI`.

4.2.12 filter

Usage: `filter <func_prefix> <det> <gs> <vent> <dv> <wm_loc> <wm_glo>`

func_prefix – functional data prefix (eg.,smooth in func_smooth).
det – polynomial order to detrend each voxel against.
gs – if == on, regress mean global signal from each voxel.
vent – if == on, regress mean ventricle signal from each voxel.
dv – if == on, regress mean draining vessel signal from each voxel.
wm_loc – if == on, regress local white matter from target voxels.
wm_glo – if == on, regress global white matter for all voxels.

This computes detrended nuisance time series, fits each run with a computed noise model, and subtracts the fit. Computes temporal SNR. This program always regresses the motion parameters & their first lags, as well as physiological noise regressors generated my McRetroTS if they are available. The rest are optional, and generally advisable save global mean regression.

Prerequisites: `init_EPI`, `linreg_calc_AFNI/FSL`, `linreg_FS2EPI_AFNI/FSL`, `gen_regressors`.

4.2.13 TRdrop

Usage: `TRdrop <func_prefix> <head_size> <FD_thresh> <DV_thresh>`

func_prefix – functional data prefix (eg.,smooth in func_smooth).
head_size – head radius in mm (def. 50 mm).
thresh_FD – censor TRs with Δ motion > x mm (def. 0.3 mm).
thresh_DV – censor TRs with Δ GS change > x % (def. 1000000 %).

This removes motion-corrupted TRs from fMRI scans and outputs shortened versions for connectivity analysis (mostly). By default, DVARS regression is set of OFF by using a very, very high threshold.

Prerequisites: `init_EPI`.

4.2.14 lowpass

Usage: `lowpass <func_prefix> <mask_prefix> <filter> <cutoff>`

func_prefix – functional data prefix (eg.,smooth in func_smooth).
mask_prefix – mask data prefix (eg., EPI_mask in anat_EPI_mask).
filter – filter type: 'median', 'average', 'kaiser', or 'butterworth'.
cutoff – filter cuttoff: either window length, or cutoff frequency.

This low-passes input data using the specified filter type and cutoff.

Both 'median' and 'average' filters operate in the time domain and therefore, the best cutoff values are odd (and must be larger than 1 to do anything). Time-domain filters are very good at removing high-frequency noise from the data without introducing any phase-shifts or ringing into the time series. When in doubt, a moving average filter with window length of 3 is a decent and conservative choice.

Alternatively, the 'kaiser' and 'butterworth' filters work in the frequency domain and accept a cutoff in Hz (people tend to use a default of 0.1). Both are implemented as bi-directional FIR filters. The kaiser window is high order and permits reasonably sharp rolloff with minimal passband ringing for shorter fMRI time series. The butterworth filter is of low order and achieves minimal passband ringing at the expense of passband roll off (in layman's terms, butterworth filters will retain more high-frequency content than a kaiser filter with equivalent cutoff). The effect of the passband ringing is an empirical question that would be best tested by the User.

Prerequisites: `init_EPI`.

4.2.15 surfsmooth

Usage: `surfsmooth <func_prefix> <FWHM>`

> func_prefix – functional data prefix (eg.,smooth in func_smooth).
> FWHM – full-width half-maximum of the gaussian kernel convolved with the surface data.

This spatially-smooths cortical data along the surface mesh, estimated by Freesurfer.

Prerequisites: `init_EPI`, `vol2surf`.

4.2.16 surf2vol

Usage: `surf2vol <func_prefix> <target_prefix>`

> func_prefix – functional data prefix (eg.,smooth in func_smooth).
> target_prefix – target data prefix (eg.,smooth in func_smooth).

This projects surface data back into a functional volume with the same properties as <target_prefix>.

Prerequisites: `init_EPI`, `vol2surf`.

4.2.17 vol2surf

Usage: `vol2surf <func_prefix>`

> func_prefix – functional data prefix (eg.,smooth in func_smooth).

Projects functional data from volume space to a Freesurfer generated cortical mesh.

Prerequisites: `init_EPI`.

4.2.18 volsmooth

Usage: `volsmooth <func_prefix> <mask_prefix> <FWHM>`

> func_prefix – functional data prefix (eg., smooth in func_smooth).
> mask_prefix – mask data prefix (eg., EPI_mask in anat_EPI_mask).
> func_prefix – functional data prefix (eg.,smooth in func_smooth).

Re-samples a mask containing one or more labels to the functional data and smooths within unique values. All zero values in the mask are zeroed out in the output. The output of this can be combined with the outputs of `surfsmooth` & `surf2vol` using `combine_volumes`.

Prerequisites: `init_EPI`.

4.3 Quality Control

5 Further Reading