Using FSL to preprocess FMRI data – week 4

Introductions

This workshop has been adapted with permission from the freely available introductory workshop on the Oxford FMRIB FSL website. I have removed a few things to save time and added slightly more information in some places. I've also reorganised the material from the website so that essential things are in the first part of this document and more advanced and technical things are in the second (optional) part.

To start..

Assuming you have completed the tutorial on bash shell, and have successfully launched a CINN Virtual Machine, you can find the folder we will use for PYM0FM using the command "`cd`" and list the files, and folders within that folder using the command "`ls`" (that's a lower-case L and a lower-case s).

Not also that in the grey box, the "$" represents the prompt for the bash, i.e. where you will type the command; you should **not** type the "$", just type what comes afterwards, as it is shown.

```
$ cd /storage/silver/pym0fm/

$ ls
```

At this point, bash should show you a list of folders, including one that has named after your IT login.

Find the data

In the command terminal, if you haven't done it already, go to your folder on the PYM0FM folder, by typing the below, and replacing your ITS login accordingly

```
$ cd /storage/silver/pym0fm/<YOUR ITS LOGIN>
```

Then use the "`ls`" (that's an "l" as in lullaby) command to see a list of files in the directory, simply type:

```
$ ls
```

You will see a directory that contains some of the fsl course data.

```
fsl_course_data
```

Now navigate into this folder

```
$ cd fsl_course_data
```

If you *ls* again, you will now see the names of 3 folders that contain the data from 3 parts of the fsl course:

```
fmri

intro

registration
```

1. LOADING FSL

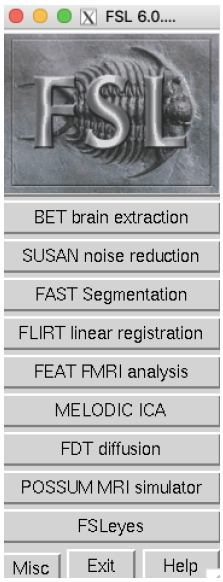
On the command line, type:

```
$ module avail
```

"module" is a little tool that provides the ability to configure the system. In CINN, we use it to make sure that we can access several versions of the same software, without them colliding with each other. The command "module avail" shows all the modules that are available. To load the version of fsl we want to use, type:

```
$ module load fsl6.0
```

Now, when we type 'fsl' into the command line, the main fsl GUI should appear.



Mini Exercise: Have a little explore of the various tabs - think a little about what they might all be doing. Note that hovering your mouse over certain fields will cause a help-dialogue to pop up. Pressing the *Help* button on each tab will also direct you to online help resources. Close the fsl window when you are ready to move on.

2. FSLUTILS

First let's navigate to the *intro* folder.

```
$ cd /storage/silver/pym0fm/<YOUR ITS LOGIN>/fsl_course_data/intro
```

fslinfo

fslinfo is a utility for retrieving information from nifti files.

Type `fslinfo highres` and `fslinfo thresh_zstat1` and note their different image matrix dimensions (dim1-3 = X Y Z) and voxel sizes (pixdim1-3 = X Y Z).

Note that some images (highres) are of integer datatype, while others (thresh_zstat1) are of floating point datatype. Integer means that the intensity values can only take on whole numbers - no fractions - raw image data is normally of this type. Floating point means that intensity values can be fractional - the result of applying most statistical processing algorithms to image data results in images of floating point type.

QUESTION: Now type `fslinfo filtered_func_data.nii.gz.` What does dim4 represent? Why was dim4 only 1 for the previous two images?

3. BET (Brain Extraction Tool)

Let us now navigate to the registration directory:

```
$ cd /storage/silver/pym0fm/<YOUR ITS LOGIN>/fsl_course_data/registration
```

Most FSL programs can be run from the command-line without using a GUI, by typing fully lowercase names (e.g. bet). Most programs also have a GUI, which can be started via the fsl mini-GUI (type fsl &) or by typing a capitalised version of the command name (e.g. Bet &).

In any analysis, you should use the GUI to explore the data and try out analyses, but really, if you are going to do any serious analyses, you should write a script that will call out the tools you use.

For those FSL programs that can be run from the command line without a GUI the usage can be obtained by typing the lowercase name. Type *bet* and press return to see the usage for bet. This tells you that to use bet from the command line you need to give it the name of an input image, a filename for the output images, and you can also choose various options concerning the output and how brain extraction is achieved. Start the fsl GUI and click on BET, to see how this all works. Now exit both of these GUIs and start the Bet GUI directly from the terminal by typing Bet (note the capital letter here)

Using the Bet GUI, set the input file to *STRUCT.nii.gz*; use the right-hand file selector rather than typing this in by hand. Turn on various optional outputs (brain extracted image, binary brain mask and skull surface image - see *'Advanced Options'* tab) but leave the other settings as they are. Note that the GUI suggests the default output name *STRUCT_brain.nii.gz*. When done click on "Go" to run BET and then exit the GUI once it's done. Note that a bet command is spat out to the terminal when you press "Go".

When this has finished (it wont take long) use ls to see what files got created and view the various outputs. Hint: if you type

```
$ ls -lrt
```

then the listing is sorted according to file creation date, hence it's very easy to see what the most recently created files in a directory are. View the brain mask overlaid onto the original image by first loading the original image into fsleyes and then adding the mask image. Change the

transparency of the overlay so that you can get a better view of the success of the brain extraction.

Type ‘bet’ to get the usage description again and make sure you understand the command that the GUI printed in the terminal when you pressed OK. For instance, one thing we learn from the output is that the -f parameter controls the fractional intensity threshold - and that smaller values lead to larger brain estimates

Next, to illustrate the effect of manipulating this parameter, as well as running bet from the command line, let's run bet again, but changing -f away from the default (which is 0.5). To do this, type:

```
$  bet STRUCT.nii.gz STRUCT_brain2.nii.gz -f 0.1 -m -s -o
```

Here we are creating a second extracted brain (STRUCT_brain2.nii.gz), but we are lowering the fractional intensity threshold. This should give us a larger brain estimate. Confirm this by viewing STRUCT.nii.gz, STRUCT_brain_mask.nii.gz & STRUCT_brain2_mask.nii.gz in fsleyes simultaneously.

QUESTION: Which of the two brain extractions has led to a better brain estimate?

4. mcflirt (motion correction) and slictetimer (slice timing correction).

Now let's illustrate two other pre-processing operations from the command line. First, let's take a look at FUNC.nii.gz. Looking at a movie of your 4D time series data is a very good way of spotting problems with the data such as excessive head motion, sudden image intensity changes (spikes), and ghosting. It is always worth doing this as an initial quality check on your data.

Open **FUNC.nii.gz** with fsleyes. Now watch this as a movie by pressing the film-strip button. It is quite clear that the participant has moved their head around a large degree during the scan. Therefore, let us run some motion correction. First, let's type *mcflirt* into the command line to see what the various options are.

Now lets run the following call to mcflirt:

```
$ mcflirt -in FUNC.nii.gz -out FUNC_motion_corrected.nii.gz -plots -report
```

After it has completed, we will have produced a file called **FUNC_motion_corrected.nii.gz.par .** Lets take a look at the contents of this file, as follows:

```
$ nano FUNC_motion_corrected.nii.gz.par
```

In the terminal an array of numbers will be printed

QUESTION: Thinking back to the preprocessing lecture, what do you think the values in this file represent? What are the columns, what are the rows?

QUESTION: Scroll down the .par file (using down key) in nano until you find a row of values that are all zeros. What do you think this means? Why is this particular row populated with zeros?

When you are satisfied that you have worked this out, press ctrl-x to close the file.

You will also notice that a file called **FUNC_motion_corrected.nii.gz** has been created. This is the motion corrected functional volume. Load this and the original **FUNC.nii.gz** into fsleyes and view them in movie mode.

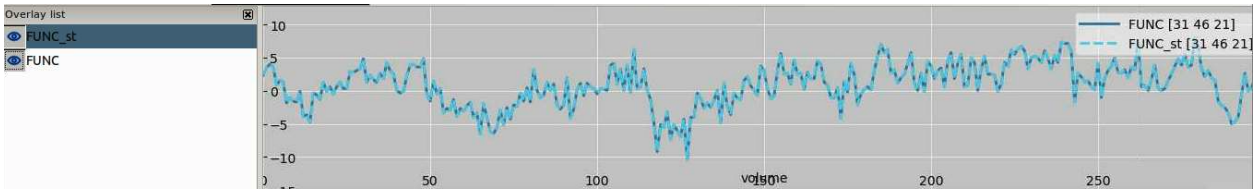
QUESTION: Does the brain appear more stationary (still) in the corrected file? Is there still some residual motion?

Now let's try out using fsl's slicetiming utility: *slicetimer*. As before, let's start by typing *slicetimer* in the command line to view the various options.

Let's just run this with all the defaults, specifying only the infile (-i) and outfile (-o).

```
$ slicetimer -i FUNC.nii.gz -o FUNC_st.nii.gz
```

After it has finished running, open both the infile and newly created outfile in fsleyes. As shown in the fsl demo, click around in various slices and compare the timecourses to observe the temporal shifts that have occurred between the two volumes (tip - you may have to 'zoom in' by changing the x axis so that any shifts can be more visible).

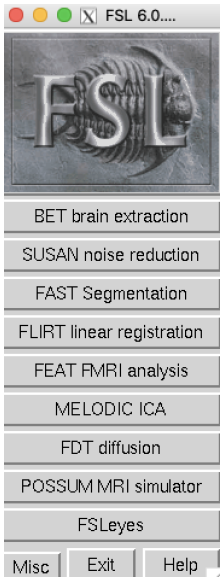


QUESTION: Can you find a slice that has no correction? Why might this be the case?

5. Registration of volumes.

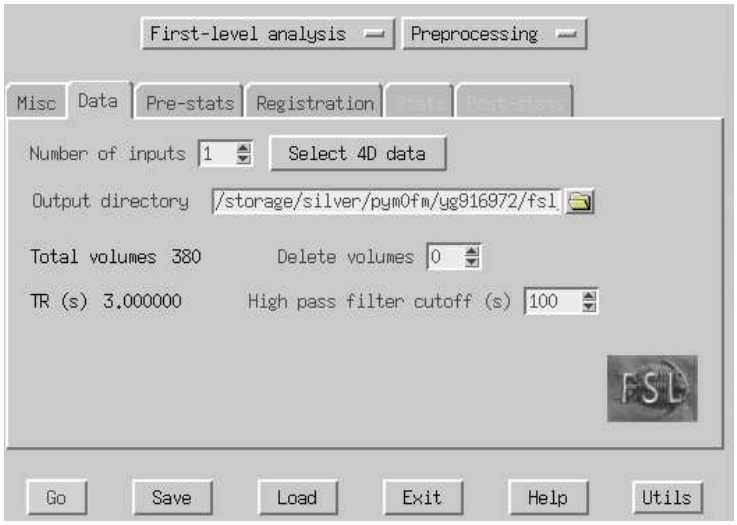
Here we are going to perform registration of some functional and structural data. To start this, type in *fsl* into the command line, which will open up the main GUI

Next, navigate to the ‘FEAT fmri analysis’ tab.



From here, we want to change the ‘*full analysis*’ dropdown to ‘’*preprocessing*’.

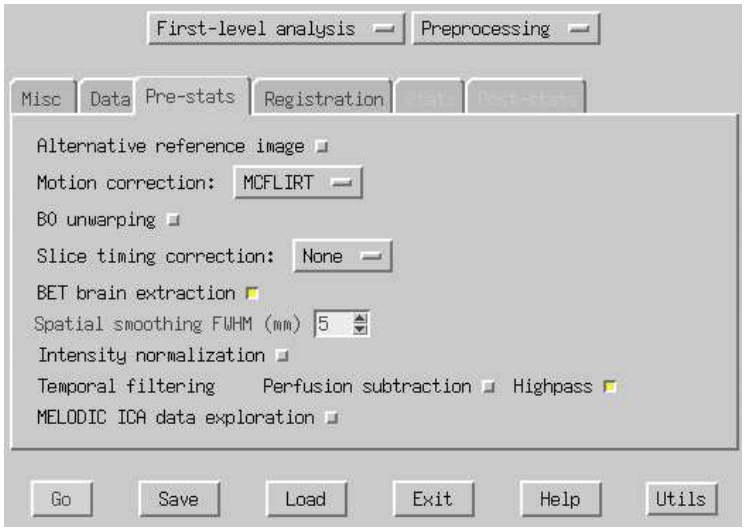
First we want to select our functional data. Navigate to the ‘Data’ tab and select *FUNC.nii.gz* as our 4D data.



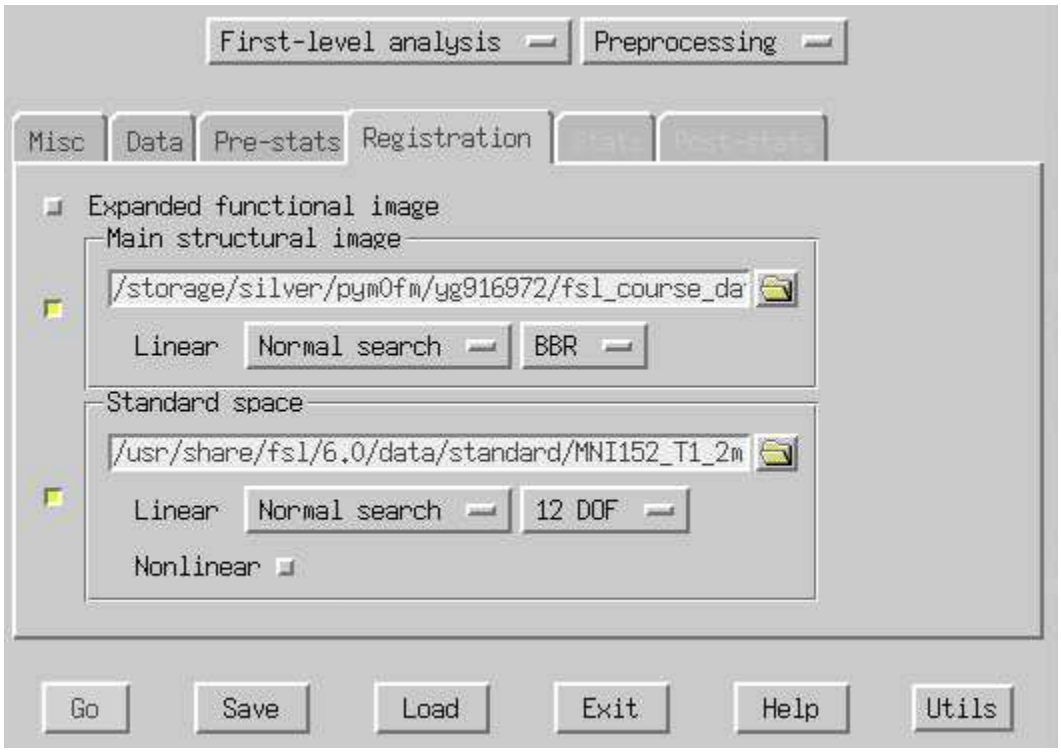
Note that the TR (3s) and total volumes (380) fields are populated when we select this file.

We next need to select an output directory. Click the folder icon next to *output directory* and type *test_reg* into the *selection* box.

Now navigate to the *prestats* tab. Note that various preprocessing stages appear here, including the motion correction and slice timing correction that we explored before.



Leave this tab and move onto the *registration* tab - which is the main focus of this exercise.



Click *main structural image*. Select the *STRUCT_brain* file that we made earlier in the BET exercise.

Note in the *standard space* section our images are to be registered to MNI 152 space. Leave everything here as the default.

QUESTION: Note that transformation to the standard space is listed as having 12 degrees of freedom. This is somewhat more than a rigid-body registration (which is 6 degrees of freedom). Why might this be the case?

Press *go* - this will open up a window in your browser. Progress of the preprocessing, including the registrations will periodically be updated here. It should only take a few minutes to finish.

When feat has finished - you will see a 'Finished at' string instead of STILL RUNNING, as shown above.

QUESTION: Click on the registration tab. What do each one of these figures show?

QUESTION: Click on the *prestats* tab. How do these plots relate to the .par file that was created in the mcflirt exercise?

Now navigate to the registration directory produced by feat.

```
$ cd /test_reg/reg
```

Lets load the MNI brain and the registered structural and functional images into fsleyes

```
$ fsleyes standard.nii.gz highres2standard.nii.gz
example_func2standard.nii.gz
```

Manipulate the transparency of the volumes to visualize how well-aligned they are.

QUESTION: Are there still significant differences between the standard and highres brain? Why might this be the case?

QUESTION: What further transformation might we be able to do to improve the alignment of these images? What action could we have taken in the FEAT gui to do this?

Part 2: optional advanced FSL procedures

The following exercises are not essential to the course, but they provide good experience of using Linux, and they will also make you aware of some useful FSL utility programs for manipulating data and image files. You can do these if you have spare time in the workshop or in your own time.

```
$ cd /storage/silver/pym0fm/<YOUR ITS LOGIN>/fsl_course_data/intro
```

fslsplit and fslmerge

run

```
$ fslsplit egfmri
```

to split up the 4D input image into the individual 3D images, or volumes, (vol0000, vol0001, vol0002, etc.). To list these new files do

```
$ ls vol*
```

(note that the * is expanded by the terminal to fit any characters in all possible filenames present, and so can be used in any commands, not just ls. * is called a wildcard).

As an example let us remove the 3rd and 6th images from this sequence and then remerge the remaining ones. Remove the images by doing

```
$ rm vol0002* vol0005*
```

Check that these files have been deleted using looking at the directory contents using ls

Now remerge the remaining 3D images into a new 4D image with

```
$ fslmerge -t newfmri vol*
```

The "-t" tells fslmerge to merge the images in time (you could also do other merges, such as stacking them on top of each other in space!) After "newfmri" (the name of the output image),

fslmerge expects a list of images to be merged. For all FSL commands the image names can be used with or without including the filetype extension (extensions are .hdr .img .nii.gz etc.).

View the resulting image in fsleyes and check that there are only 8 volumes in the timeseries. You can also check the size of the timeseries with fslinfo. Also try concatenating in z instead of t and view in fsleyes; why does nothing happen when you try to turn on the movie loop?

Now tidy up by removing the redundant vol files using the rm command.

fslroi

We will now fix an image that has somehow gotten "wrapped", with the nose sitting behind the back of the head. This sort of thing actually happens if you have a participant with a larger than average head and you don't have enough "slice oversampling" in your protocol. The method used here might enable you to "recover" data that would otherwise be wasted.

```
$ fsleyes wrapped &
```

You will use fslroi to split this image into two (front and back parts) and fslmerge to remerge them back together in the right order. In fsleyes, look at an inferior slice so that you can work out the y coordinate (in the anterior-posterior direction [= front to back]) where you want to make the split. Record the y-coordinate (in voxels, not mm).

Now use fslinfo to find out what the image matrix dimensions are (dim1,2,3). You're now ready to extract the back part of the head into a temporary file. If your y-coordinate was 76 you would do:

```
$ fslroi wrapped back 0 256 76 180 0 128
```

Now just type fslroi to see the usage. There are several ways to run fslroi: You can specify just a 3D ROI, as we have done here - this is the first of the usage possibilities. The second option is to extract a temporal ROI from a 4D dataset. The final option extracts a 4D ROI from a 4D dataset, controlling the region to be extracted in space and time. The program knows which option you want from the number of "arguments" that you type. Note that if you use the first option (3D ROI) on a 4D input file, the output will still be 4D, with the number of timepoints unchanged, and each volume cut down in 3D in the same way, as specified by the arguments.

Make sure that you understand the exact numbers used in the above command, that created back; note in particular that the second number in each pair is the length of the ROI in the relevant dimension, not the end point - the <ysize> was 180 (256-76) not 256.

Now work out how to run a similar command to create "front", from the back part of wrapped.

Finally, use fslmerge to merge front and back together correctly, and verify with fsleyes. Note that you could also have done a similar thing with the slight wraparound at the top of the image.

.