Final Project

Max Egan

2022-12-08

# Building a Preprocessing Pipeline for fMRI Data in R

## Background

There has been no development more impactful to the field of cognitive psychology than the invention of magnetic resonance imaging (MRI). Developed in the early 1990’s, MRI uses a superconducting magnet to generate magnetic gradient fields that selectively excite electrons and record the subsequent activity. From the neuroscience perspective, this has given us great insight into the brain on an anatomical level. The response properties of the electrons vary depending on tissue type, allowing us to generate a very high resolution image of the brain that can accurately depict white and gray matter, cerebral spinal fluid, bone, tumors and many other tissues. Different scanning methodologies also allow us to exploit other laws of physics to generate statistically-based representations of brain anatomy, for example in the case of diffusion tensor imaging using the anisotropic properties of water to develop maps of white matter tracts in the brain (Basser et al., 2000). The field of cognitive neuroscience has benefitted greatly from one such methodology developed in 1992, termed functional MRI (fMRI) (Ogawa et al., 1992).

The methodology of fMRI centers around the use of cerebral blood flow as a measure of neural activity. The electrical properties measured by the MR machine rely on the fact that deoxygenated hemoglobin is paramagnetic whereas oxygenated hemoglobin is diamagnetic, or more basically, deoxygenated blood has a small magnetic pull whereas oxygenated blood does not. The underlying premise is that areas of the brain that are actively being used will receive greater blood flow, so by measuring the ratio of oxygenated to deoxygenated blood at any given location over a course of time one can track the activation of that location. This measured quantity is called the blood oxygenation level dependent (BOLD) signal. While the spatial resolution is still much higher than other imaging modalities, the nature of recording the BOLD response means the functional image does not have quite the spatial quality of an anatomical MRI scan. Still, with a carefully controlled experimental design, researchers are able to accurately tease apart which locations in the brain are ‘active’ during a particular set of tasks, for example the visual cortex when presented with a flashing checkerboard.

## The Problem

After recording an fMRI scan, researchers are given a set of files that contain information about the BOLD response at each voxel in the brain, a volumetric measurement typically about 2x2x2 mm^3. Each file represents one volume of the brain taken by the MR scanner, or a full set of images of the brain for each time the scanner samples it, typically between 1.5-2s. The experimenter then uses some statistical software to indicate the timepoints that they are interested in, for example at each point an auditory stimulus is heard, and the program will calculate which voxels contain a significantly different BOLD response at those time points as compared to a separate and/or baseline condition.

These raw files are known as DICOM files (Digital Imaging and Communications in Medicine), and represent raw scanner data. These are then converted to NifTi files(Neuroimaging Informatics Technology Initiative), hereafter referred to as NII files, to be used in data analysis. The conversion from DICOM to NII,however, represents only the very first step in the data analysis pipeline. These fresh NII files are unique to each subject, rife with artifacts from both the scanner and head motion, and functionally cannot be used for any true data analysis. In order to be useable, they must undergo numerous preprocessing steps that normalize them to the neuroimaging field.

## The Project

Typically the preprocessing pipeline is done using various statistical packages that are implemented through MATLAB or Python, with the latter becoming more common as the default choice. There are several packages that perform the core functions of preprocessing, but currently the most common are SPM (Statistical Parametric Mapping, typically used with MATLAB), and FSL (FMRIB Software Library, typically implemented in Python). In my case, our lab has worked primarily with SPM in a MATLAB environment to varying degrees of efficiency. Particularly, as scanning methods become more advanced and our experiments become more complex, the raw data coming out of the scanner increases drastically in terms of volume count and file size (on the order of 200mb per session of 3 runs to 1.3GB per individual run). This represents an obvious drastic increase in computational demands in which SPM, for particular functions, is beginning to lag behind a bit.

While thinking about topics for this project, I came across an implementation of the FSL package tailored to the R environment, FSLR, that seemed perfect for this course (Jeninson et. al, 2012, Muschelli et. al, 2015). The goal of this project is to implement as much of a preprocessing pipeline as possible in the R environment using the FSL wrapper to see if it is a viable option moving forward for our data, particularly in R’s utility to be used in a cluster environment. Note that this isn’t a direct comparison of the two packages/softwares, but more of “Can I even do this in R?” type of project.

## The Pipeline

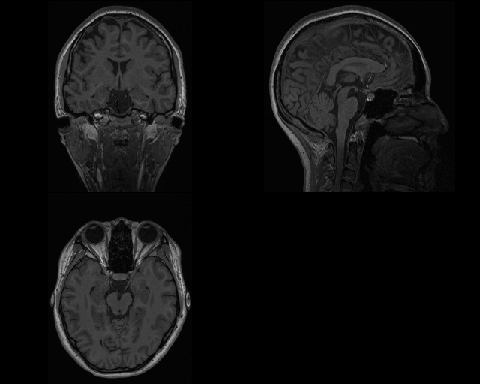
### What we are working with

First, it’s helpful to understand what we are starting with when working with fMRI data (we’ll assume that all files have already been converted to NII). There are two types of scans we are concerned with for this pipeline that are taken from each subject.

The first is the anatomical scan. This is a high-resolution scan that contains information only on the structural parts of the brain, allowing us to view with great detail the bone, white matter, and grey matter of the brain. This scan contains no information about blood flow, and thus no information about indirect neural activity.

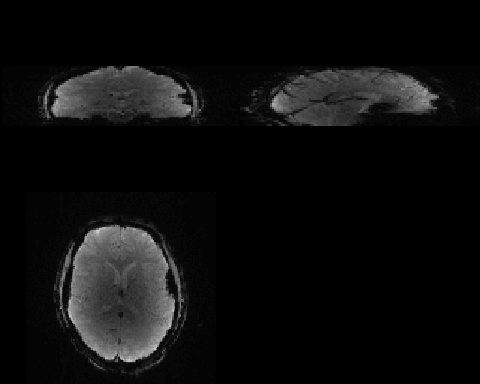
library(fslr)  
options(fsl.path="/home/maxkegan/fsl")  
options(fsl.outputtype = "NIFTI")

anatfile <- "~/Final\_Project/subject\_anatomical.nii"  
anatfile <- readNIfTI(anatfile, reorient = FALSE)  
ortho2(anatfile, crosshairs = FALSE)



The second are the functional scans, the scans that contains information about the BOLD response and thus the actual neural activity within the brain.

funcfile <- "~/Final\_Project/subject\_functional\_TC.nii"  
funcfile <- readNIfTI(funcfile, reorient = FALSE)  
ortho2(funcfile[,,,1], crosshairs = FALSE)



If your first thought is “… one of those is not like the other”, you would be correct! The functional scans are absolutely terrible compared to the anatomical scan in terms of spatial resolution by virtue of how the BOLD signal is acquired in the scanner. The physics behind it aren’t relevant here, but what is important is that we somehow need to bridge the world between these two. Also, note that the code required for the functional file was slightly more complex. If we look at the dimensions of the two files we see why.

dim(anatfile)

## [1] 192 256 256

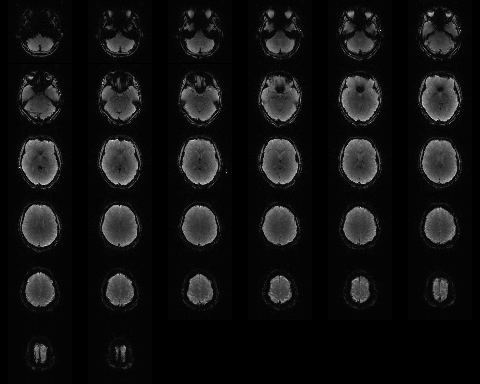
dim(funcfile)

## [1] 128 128 32 338

The anatomy file is 3d, whereas the functional file is 4d. The first three numbers for each scan represents the X,Y, and Z dimensions of the image, but the 4th column in the functional file represents all of the functional scans that are taken during the session. Read: we take a full volume of the brain 2256 times per session for each subject, and in this particular experiment, there are 6 sessions per subject. The data problem should be becoming readily apparent.

It’s useful to show what I mean when I say that we take a full volume of the brain. When we take a single volume, the way the MR physics works is that you can only excite small slices of the brain at a time in order to measure the resulting hydrogen proton realignment. Each volume in this experiment represents 32 slices taken by the scanner at different sections of the brain. Shown visually:

image(funcfile, w=1)



The images should actually be lined up directly rather than having a few of the brain slices cut like that, definitely a knock towards FSLR right here, but ultimately we don’t visualize the images like this in the pipeline code so not necessarily a deal breaker. The take home message from the image is that each individual volume is actually comprised of 32 distinct slices of the brain. This is done over the course of what is called a TR (repetition time), which in this case is 500ms. So, in total, we are taking 32 slices every 500ms for 2256 scans, which means the total length of each run is .5\*2256=1128s, or 18 minutes and 48 seconds per run. Yes, it’s long, and yes, the subjects do get very tired.

This brings us to the first major step of the pipeline:

### Slice timing correction

Because we are exciting 32 individual slices at different times during the 500ms period, we have to have some way to have each slice accurately reflect it’s activation during the time it is not actively being excited during the 500ms. To do this, we temporal interpolation via a process called slice timing correction. This interpolates across all slices such that each voxel accurately represents a continuous activation during the whole acquisition period.

fslslicetimer(funcfile, outfile="a\_subject\_functional.nii", tr=.5, acq\_order="interleaved", reorient = FALSE, verbose=FALSE)

## NIfTI-1 format  
## Type : nifti  
## Data Type : 16 (FLOAT32)  
## Bits per Pixel : 32  
## Slice Code : 0 (Unknown)  
## Intent Code : 0 (None)  
## Qform Code : 1 (Scanner\_Anat)  
## Sform Code : 1 (Scanner\_Anat)  
## Dimension : 128 x 128 x 32 x 338  
## Pixel Dimension : 1.8 x 1.8 x 3.85 x 0.5  
## Voxel Units : mm  
## Time Units : sec

The output of the run shows the first volume of the functional scans and the offset of all 32 slices. The resulting Nifti output shows that we still have the correct dimensions from the original functional scan. We are left with a new file that has an “a\_” as a prefix to indicate that we’ve performed slice timing correction. On to the next step.

funcfile <- readNIfTI("a\_subject\_functional.nii", reorient = FALSE)

### Head motion correction

One of the biggest artifacts that exists in fMRI data is simple head motion. When we consider that a whole volume of the brain is roughly 12cm in length and the voxels are typical 2mm cubes, head movement of even a single mm can massive change the data within a voxel, especially if the tissue type changes, such as a space that was previously gray matter suddenly being filled with bone or CSF. The most basic way to deal with this problem is called head motion correction, and is done by the process “mcflirt” in FSLR. This extract the parameters for the 6 rigid head motion movements, which correspond to the translational movement in the X, Y, and Z direction, along with the rotational movement.

mcflirt(funcfile, outfile = "ra\_subject\_functional.nii", reorient=FALSE, verbose=FALSE, opts="-plots")

## NIfTI-1 format  
## Type : nifti  
## Data Type : 16 (FLOAT32)  
## Bits per Pixel : 32  
## Slice Code : 0 (Unknown)  
## Intent Code : 0 (None)  
## Qform Code : 1 (Scanner\_Anat)  
## Sform Code : 1 (Scanner\_Anat)  
## Dimension : 128 x 128 x 32 x 338  
## Pixel Dimension : 1.8 x 1.8 x 3.85 x 0.5  
## Voxel Units : mm  
## Time Units : sec

The resulting output is a motion corrected file now with the “ra\_” prefix attached, but what it corrected can be seen in the secondary file that gets put out, the “ra\_subject\_functional.par” file. We load it now to view its contents:

motion\_params = readLines("/home/maxkegan/Final\_Project/ra\_subject\_functional.par")  
motion\_params = strsplit(motion\_params, split = " ")  
motion\_params = sapply(motion\_params, function(x) {  
 as.numeric(x[ !(x %in% "")])  
})  
motion\_params = t(motion\_params)  
head(motion\_params)

## [,1] [,2] [,3] [,4] [,5] [,6]  
## [1,] -0.00267664 9.17437e-05 -0.000944615 -0.0644757 -0.0503353 -0.1132670  
## [2,] -0.00267664 1.71230e-04 -0.000875531 -0.0452882 -0.0562928 -0.1155040  
## [3,] -0.00281189 1.17937e-04 -0.000768518 -0.0454023 -0.0451456 -0.0991743  
## [4,] -0.00267664 -9.10210e-06 -0.000769602 -0.0422054 -0.0502565 -0.0943494  
## [5,] -0.00288299 1.17956e-05 -0.000776741 -0.0401849 -0.0444573 -0.0898928  
## [6,] -0.00264887 3.43254e-05 -0.000742004 -0.0365720 -0.0387190 -0.0920634

Each of these columns represents one of the 6 rigid head motion parameters, 3 translational and 3 rotational. These can later be plugged into a GLM as confounds to the data so any activations that directly align with the head motion will be accounted for. Now that we have corrected the head motion, it’s time to move to the next step: aligning the crude functional images with the crisp stuctural image of the subject.

### Coregistration

### Normalization to standard space

### Smoothing

## Conclusions

Even though I didn’t fully complete the pipeline due to time constraints, I have to say that I’m not too terribly impressed with the wrapper functionality in R. It’s not bad per se, but I definitely think to spend the amount of time I did on this just for basic preprocessing steps isn’t worth the effort. I much rather would code in MATLAB because it’s more familiar or Python because there is simply more power there. I do feel like I gained a much greater understand for how these functions work though just through troubleshooting them and trying to get them to work with the data properly. Additionally, thought I don’t have hard testing numbers on it, I also feel like the processes just generally ran faster in R. I believe part of this is due to the archaic way that MATLAB saves files, particularly large ones, but again I also think that these gains would be seen in Python.

All in all I could see a world were a pipeline for fMRI preprocessing could be used in R via the FSLR wrapper, but ultimately I think a lab would be better served by simply switching to Python.

## Future directions

Clearly I didn’t get as far in this as I would have liked, the three major steps that are still needed are co-registration, normalization to a standard space, followed by smoothing. Additionally, I would have liked to begin analysis of the behavioral data that came out of the actual experiment. It’s a very dense experiment, and the utility that comes with wrangling data in R would clearly be an advantage here.

#### References

1. Basser PJ, Pajevic S, Pierpaoli C, Duda J, Aldroubi A (2000) In vivo fiber tractography using DT-MRI data. Magn Reson Med 44:625–632.
2. Ogawa S, Tank DW, Menon R, Ellermann JM, Kim SG, Merkle H, Ugurbil K (1992) Intrinsic signal changes accompanying sensory stimulation: functional brain mapping with magnetic resonance imaging. Proc Natl Acad Sci U S A 89:5951–5955.
3. Jenkinson M, Beckmann CF, Behrens TE, Woolrich MW, Smith SM. FSL. Neuroimage. 2012 Aug 15;62(2):782-90. doi: 10.1016/j.neuroimage.2011.09.015. Epub 2011 Sep 16. PMID: 21979382.
4. Muschelli J, Sweeney E, Lindquist M, Crainiceanu C. fslr: Connecting the FSL Software with R. R J. 2015 Jun;7(1):163-175. PMID: 27330830; PMCID: PMC4911193.