

# Neuroimaging Processing R

John Muschelli

Johns Hopkins Bloomberg School of Public Health

April 3, 2015

# What makes a comprehensive (s)MRI library

- Read/Write NIfTI images
- Visualize Images
- Inhomogeneity/Bias-field Correction
- Segmentation
  - Brain Extraction
  - Tissue-Class Segmentation (WM, GM, CSF)
- Spatial "Normalization" /Registration
  - Within Visit
  - Across Visit/People
- Intensity Normalization

# FSL and fslr

- FSL is a comprehensive library of analysis tools for fMRI, MRI and DTI brain imaging data.
  - Collection of routines in C, C++
- fslr: port of FSL into R
- The two functions we focus on are:
  - 1 Brain Extraction Tool (BET)
  - 2 Image inhomogeneity correction and Tissue-class Segmentation (using FAST [6])

# ANTS and ANTsR

- Advanced normalization tools (ANTS) [1] is state-of-the art software that can perform many neuroimaging-related functions.
  - Collection of routines in C, C++, and some R
- ANTsR: port of ANTS into R using Rcpp
- The two functions we focus on are:
  - 1 Image inhomogeneity correction (N3 [4] and N4 [5])
  - 2 Tissue-Class Segmentation (Atropos/K-means)
  - 3 Image registration (Symmetric Normalization)

# extrantsr and WhiteStripe

- extrantsr:

**Extra ANTsR** functions.

ANTsR has some non-intuitive syntax and structures. I created extrantsr to make thing easier and create pipelines.

- WhiteStripe:

WhiteStripe from Shinohara et al. [3]: intensity-based normalization of T1 and T2 image that estimates a white matter mean and standard deviation on data rigidly-registered to the MNI template using histogram-based methods.

# Installing These Packages

First, you must Install FSL

<http://fsl.fmrib.ox.ac.uk/fsl/fslwiki/FslInstallation>.

`fs1r` is installed on CRAN, but the development arm of `fs1r` is most likely the best to install, using the `devtools` package (requires new version of `oro.nifti`)

`ANTsR` is currently (as of April 3, 2015) hosted on GitHub.

```
if (!require(devtools)){  
  install.packages('devtools')  
}  
devtools::install_github("muschellij2/oro.nifti")  
devtools::install_github("muschellij2/fs1r")  
install.package("WhiteStripe")  
devtools::install_github("stnava/cmaker")  
devtools::install_github("stnava/ITKR")  
devtools::install_github("stnava/ANTsR")
```

# Installing These Packages

Setup for JHPCE Cluster is here:

http:

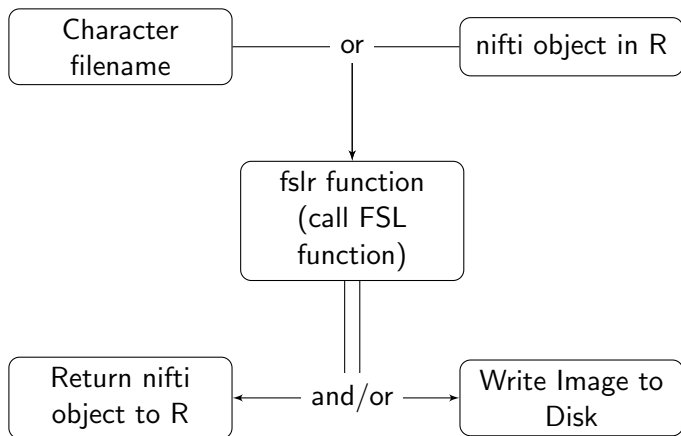
[//muschellij2.github.io/neuro\\_cluster/README.html](http://muschellij2.github.io/neuro_cluster/README.html)

# Unix

FSL works only on Unix-type machines, so `fs1r` only works on Unix-type machines.



# Structure of fslr functions



# Reading and Writing Images

The `oro.nifti` does most of the reading/writing of NIfTI images:

```
library(oro.nifti)
nim = readNIfTI("Output_3D_File.nii.gz",
reorient=FALSE)
nim
```

# Reading in Images using ANTsR

Reading in images using ANTsR requires 2 changes compared to `readNIfTI` from `oro.nifti`:

- 1 The extension of the filename (e.g. `.nii.gz`) must be specified
- 2 The dimension of the image (usually 3) must be supplied (could be 2, 3, or 4)

```
library(ANTsR)
aimg = antsImageRead("Output_3D_File.nii.gz",
dimension = 3)
```

# ANTsR images

The `aimg` object is an object of `antsImage`, which consists of:

- `pixeltype` - how is the image stored (integers versus fractional numbers (floats))
- `dimension` - how many dimensions does the image have
- `pointer` - where the data is stored

```
class(aimg)
aimg
slotNames(aimg)
```

# Visualizing Images

```
orthographic(nim)
```

# Visualizing Images with an Overlay

Viewing the image and overlaying in red values  $> 200$ )

```
library(fslr)
ortho2(nim, nim > 200, col.y=alpha("red", 0.5))
```

# Inhomogeneity/Bias-Field Correction

ANTsR has `n3BiasFieldCorrection` and `n4BiasFieldCorrection`. These require `antsImage` objects to run.

`extrantsr::bias_correct` will perform either of these, you just need to switch correction

```
library(extrantsr)
n4img = bias_correct(nim, correction = "N4", retimg=TRUE)
```

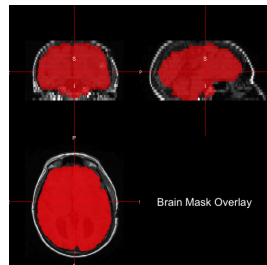
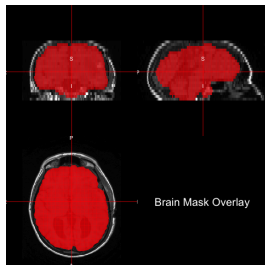
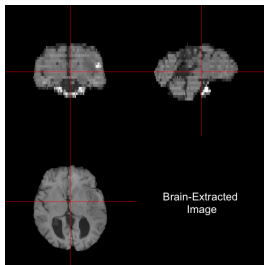
## fslr: Brain Extraction

FSL's Brain Extraction Tool (BET) can be used for skull stripping. It is fast, robust, and one of the most popular for this task. `fslr::fslbet` is used to call the FSL commands `bet2`, which does brain extraction or `bet`, which does brain extraction with additional options.

```
brain_img = fslbet(infile=n4img, retimg=TRUE)
### rerurn with new cog
cog = cog(brain_img, ceil=TRUE)
brain_img2 = fslbet(infile=n4img,
                    retimg=TRUE,
                    opts = paste("-c", paste(cog, collapse=
```



# fslr: Brain Extraction Results



# ANTsR: Tissue class Segmentation

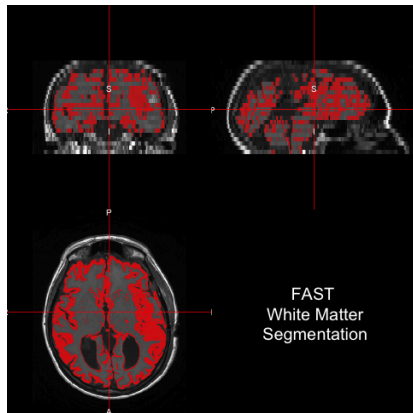
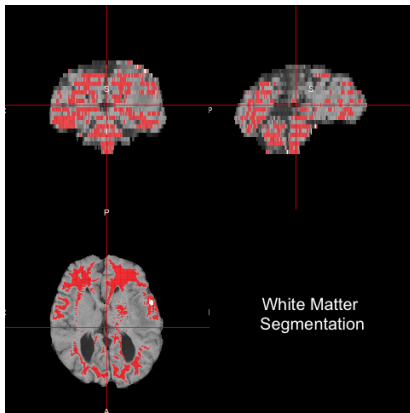
Below is some code to do k-means segmentation of an image (k=5) and then finding the one with the highest intensity (white matter). (discuss ants2oro)

```
library(fslr)
a = oro2ants(brain_img2)
x = oro2ants((brain_img2>0)*1)
k = 5; km = kmeansSegmentation(a, k=k, kmask=x)
res = ants2oro(km$segmentation)
voysize = prod(voxdim(n4img))/1000
ks = seq(k); sizes = sapply(ks, function(i) sum(res == i))
keep_ks = ks[ sizes > 200 ] # Arbitrary size - need to be
wm = cal_img(res == max(keep_ks))
```

# fslr: FAST tissue-class segmentation

```
fast_img = readNIfTI("test_fast_seg.nii.gz", reorient=False)
```

# Tissue-Extraction Results



# Image Registration

- fslr
  - flirt - linear (rigid/affine) registration
  - fnirt - non-linear registration
- ANTsR: antsRegistration
  - Rigid/Affine
  - SyN - symmetric Normalization (REVERSIBLE non-linear registration)

# Image Registration: to Template

`extrantsr::ants_regwrite` will register images (nifti objects or filenames), write outfiles,

```
brain_to_temp = ants_regwrite(file = brain_img2,  
                              template.file = "MNI152_T1_1r  
                              typeofTransform = "SyN")
```

# CRAN/NITRC is not good enough





- CRAN is great for R.
- CRAN is bad for neuroimaging.
- NITRC is not good enough for neuroimaging (Neuroimaging Informatics Tools and Resources Clearinghouse)

# GitHub has what we need

- Version control (obviously)
- Install packages via `devtools::install_github`
- Stars for GitHub Packages
- Travis CI for checking builds
- Issue pages
- Data Packages (if `github.com`)
- Collaborations
- Live development version



# References I

-  Brian B Avants et al. “A reproducible evaluation of ANTs similarity metric performance in brain image registration”. In: *Neuroimage* 54.3 (2011), pp. 2033–2044.
-  Régis Guillemaud and Michael Brady. “Estimating the bias field of MR images”. In: *Medical Imaging, IEEE Transactions on* 16.3 (1997), pp. 238–251.
-  Russell T Shinohara et al. “Normalization Techniques for Statistical Inference from Magnetic Resonance Imaging”. In: (2013).
-  John G Sled, Alex P Zijdenbos, and Alan C Evans. “A nonparametric method for automatic correction of intensity nonuniformity in MRI data”. In: *Medical Imaging, IEEE Transactions on* 17.1 (1998), pp. 87–97.

# References II



Nicholas J Tustison et al. “N4ITK: improved N3 bias correction”. In: *Medical Imaging, IEEE Transactions on* 29.6 (2010), pp. 1310–1320.



Yongyue Zhang, Michael Brady, and Stephen Smith. “Segmentation of brain MR images through a hidden Markov random field model and the expectation-maximization algorithm”. In: *Medical Imaging, IEEE Transactions on* 20.1 (2001), pp. 45–57.

# Interactive/GUI vs. Terminal R

In general, GUI-based apps do not inherit the shell environment (aka if FSLDIR is defined in your Terminal, RStudio doesn't see it). For fslr to work, it must know where the directory FSL was installed. If FSLDIR is found, it will be used. You can check this by 2 ways:

```
Sys.getenv("FSLDIR")  
library(fslr)  
have.fsl()
```

If `have.fsl() = FALSE` then you must specify the path using:

```
options(fsl.path="/my/path/to/fsl")
```

# Functions of Note

`extrantsr::preprocess_mri_within` - will N4 (or N3) correct images, co-register scans to the first (T1w usually) scan, skull strip the image (using BET)