**Use of FSL on Discovery Cluster**

scp -r file.to.move tinney.e@ood.northeastern.edu:/scratch/tinney.e

**Introduction and Overview of FSL on Discovery Cluster**

Welcome to FSL (FMRIB Software Library) on Discovery.

FSL is a comprehensive library of analysis tools for FMRI, MRI and DTI brain imaging data. Information on FSL specifics can be found at the helpful FSL wiki here: https://fsl.fmrib.ox.ac.uk/fsl/fslwiki

FSL is also a loadable module on the Discovery Cluster and can also be installed on your local computer. Certain aspects of FSL may function better on the cluster through a shell application, and some may be more accessible via Open On Demand (OOD) through Jupyter. We will discuss this shortly. This guide will cover:

1. FSL installation and access
2. DTI Pipeline
3. FSL best practice

**FSL: Cluster and at Home**

**Uses of FSL**

We can break down the use of FSL into three major components:

1. Command-line based computation. These include the uses of fslmaths, fslroi, and others.
2. Brain image warping, editing, and pathfinding. These can be done via command line and/or the cluster window, and include such eddy\_correct, bet, and bedpostX.
3. Viewing brain and DTI images, including ROIs (which can also be drawn). This can only be done using fsleyes.

So, how do these relate to the Cluster on shell versus what I can/need to do on OOD through browser or at home?

In simple terms:

1. Anything that can be done on the command line can be done on the cluster or at home. The cluster, however, is probably (a lot) more powerful than your own device, and will be able to run many fsl commands in a fraction of the time your personal computer would require. Therefore, **it is best practice to do most, if not all computation, image editing, and tractography on the cluster.** Examples of differences in time and cpu strain include the aforementioned bedpostX, which on Cluster GPUs can complete in a clean fifteen minutes while the author’s Windows Laptop would probably get it done within a few years.
2. **opening the fsl window and fsleyes is not possible on the cluster through a shell application, such as Ubuntu, PuTTY, or Terminal.** This may be resolved, but is currently the only reason using fsl on OOD through a browser is necessary if you want to look at the brain images. Alternatively, you can install FSL viewing software on your own device.
3. The fact that there is a disparity between what you can / should do on Cluster and your device means that software that can easily transfer files between both is highly recommended. Author’s personal choice is FileZilla. Installation instructions and button can be found here:

      https://filezilla-project.org/

In summary, you should do anything computationally intensive on Cluster through shell or browser, but you can only see the images you edit/create through Cluster on browser or your home computer, so plan on using both and get comfortable with switching between the two.

**How to access FSL viewing software (fsleyes)**

If you want to install FSL to view your images on your own device, follow this link. Warning: Complicated but manageable.

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

**PROS:** FSL on your computer, no need to switch to a browser to view images if you aren’t already using OOD, sense of pride and ownership.

**CONS**: Slow, requires file transfer and computer space.

If you want to use Open On Demand to access FSL on Cluster, read and follow the instructions provided here:

<https://rc-docs.northeastern.edu/en/latest/using-ood/introduction.html>

**PROS**: Easy, clean, can directly access files from cluster to view in fsleyes without file transfer.

**CONS**: Programs are subject to necessary timeouts (though this can be mitigated).

*The author uses both FSL / fsleyes on his device and on OOD, and recommends using OOD.*

**Using FSL on Discovery Cluster**

**Commands**

Unlike your home computer, you need to “load” fsl on Cluster every time you want to use it (since it’s a module). This module will unload whenever your account is not logged in to Discovery Cluster. To load the newest module as of 2020/20/08, type into the Cluster command line:

module load fsl/6.0.3-cuda9.1

Then, you can type

module list

to see that it has been loaded. If for any reason you want to unload the module, type

module unload fsl/6.0.3-cuda9.1

Tip: Save time by typing “fsl/6.0.3” and hitting tab once or twice. This will autofill the rest of the module name.

Having loaded the module, you may now use all fsl commands (with the exception of those that would open a seperate fsl page, such as “fsl” or “fsleyes”).

**DTI on Cluster**

FSL has many uses, and this guide will mostly focus on its use in DTI (Diffusion Tensor Imaging). The pipeline for DTI has been streamlined through the efforts of the Northeastern University Research Computing department, and many of the steps have been significantly expedited since myself and others doing DTI began working with them. If you have further suggestions as to how FSL can be improved upon on Cluster, they are available on the Northeastern Research Computing website.

**The DTI Pipeline**

DTI is a several-step process for understanding linkages between regions of interest in FMRI brain images. An example pipeline may look like this:

1. Start with raw FMRI brain data (FA images).
2. Using FSL on your device or on OOD, open fsleyes and view these images.
3. On Cluster, use the eddy\_correct or eddy\_cuda command to correct the brain image for artefacts.
4. Using FSL on your device or on OOD, open fsleyes and view these images.
5. On Cluster, run fslroi and bet2 on the FA images to create a mask.
6. Using FSL on your device or on OOD, open fsleyes and view these images.
7. Using the mask, eddy\_corrected result, bvals and bvecs (which you should already have), run dtifit on Cluster to prep the brain for DTI.
8. Using FSL on your device or on OOD, open fsleyes and view these images.
9. Reconfigure directories on Cluster and run bedpostx (explained further on) to find all tracts within the brain.
10. Use Cluster to run TBSS.
11. Use FSL on your device or OOD with fslmaths and fsleyes to configure, draw, and view Regions of Interest (ROIs) on standard-space brains.
12. Use Cluster to run flirt, fnirt, invwarp, and applywarp to put your subject brains into standard space and match the ROIs to them.
13. Using FSL on your device or on OOD, open fsleyes and view these images.
14. Use Cluster to run probtrackx, which views the probability of connection between two given regions of interest.
15. Threshold, binarize, and begin comparison of subject brains. This is one of the ultimate goals of DTI and is the end of this example pipeline.

**DTI TIPS**

For your DTI pipeline, the first word of command is that of your lab or organization. This guide is intended to smooth wrinkles, and will walk you through an example process, but yours may vary from this example. For further assistance with DTI pipelining and comprehension, consult FSL wiki (linked above).

Here is a non-exhaustive list of tips and helpful tools for DTI.

**Beginning and Organization**

* Having each subject’s data in an individual, uniquely-named directory for that subject is essentially a must. This will make everything easier.
* Make sure that your subject directories contain files with the following endings (or corresponding tags):

1. subject\_dwi.nii.gz (image of subject brain)
2. subject.bval (subject bvals)
3. subject.bvec (subject bvecs)

* This author recommends creating a separate directory for scripts.

**Viewing**

* To view your images, either open FSL on your computer or, if using OOD, open OOD by heading to this link:

<https://rc-docs.northeastern.edu/en/latest/using-ood/fileexplore.html>

and clicking the link under step 1. Log in.

1. On the top bar, click “Interactive Apps” and mouse down to “FSL.” Launch FSL. (No need to change the numbers here).
2. Click “Launch FSL.”
3. A new window will open with the FSL interactive interface loaded into it. Drag the interface up and click on the bottom button, “FSLEYES.” It will take a minute to open.
4. Once open, use File->Add from file or File->Add from directory to add images from the cluster to the viewing console.
5. Further navigation of fsleyes can be found here:

<https://fsl.fmrib.ox.ac.uk/fslcourse/lectures/practicals/intro1/index.html>

NOTE: Make sure to Delete your job once you’ve finished using fsleyes!

**First Steps: eddy\_correct**

* General information on eddy can be found here [**https://fsl.fmrib.ox.ac.uk/fsl/fslwiki/eddy/UsersGuide**](https://fsl.fmrib.ox.ac.uk/fsl/fslwiki/eddy/UsersGuide)
* , but use will vary lab to lab. Thus an overview will be given here.

REMEMBER: BEFORE RUNNING **ANY** GIVEN COMMANDS, LOG IN TO A COMPUTE NODE WITH

srun --pty /bin/bash

This will allow the cluster to allocate proper processing power to your job.

Back to eddy:

* eddy\_correct is our current most powerful command. You’ll want to use

eddy\_correct <input\_file> <output\_name> <reference\_number>

in the command line after loading the fsl module.

* Your input is your base subject dwi file.
* Your output is named by you and should probably be called something like (subjectname)\_eddycorrected
* The reference number is, in the case of B0 data (which yours most likely is), just 0.
* So:

eddy\_correct subject-001\_dwi.nii.gz subject-001\_dwi\_eddycorrected 0

is an example of eddy\_correct run on subject-001\_dwi.nii.gz

If you aren’t using B0 data, reference the FSL wiki eddy page (or, for that matter, the web) for alternative reference numbers (sometimes called reference volumes.

Before moving on, make sure to look at your new images in fsleyes to make sure eddy\_correct removed at least a few artefacts.

**Next Steps: fslroi and bet2**

* fslroi gives us a b0 value for a brain image, and bet2 uses that value to give us a mask. These are both necessary for the next step, dtifit.
* fslroi takes your base subject brain images and yields a b0 output in this case. Formatting is as follows:

fslroi <subject\_dwi\_image> <output\_name> 0 1

to generate a b0 file. An example for our example subject-001\_dwi.nii.gz file is as follows:

fslroi subject-001\_dwi.nii.gz subject-001\_dwi\_b0\_output 0 1

Then we run bet2 using the output of fslroi. bet2 will give us a nice mask and is formatted as follows for this purpose:

bet2 <b0\_input> <output\_name> -m -f 0.1

Example with subject-001\_dwi.nii.gz:

bet2 dwi\_b0\_output.nii.gz betoutput -m -f 0.1

This will give you a file ending in mask.nii.gz. This is important.

**DTIFIT**

Now let’s run dtifit. Make sure you have:

1. The output of eddy\_correct
2. The mask from bet2
3. Your files ending in bval and bvec

all in the same place.

Let’s get running!

dtifit -k <eddy\_result> -o <output\_name> -m <mask> -r <bvec> -b <bval>

*Or*

dtifit -k subject-001\_dwi.nii.gz -o subject-001\_dtifitoutput -m \*mask.nii.gz -r \*bvec -b \*bval

Congratulations! You’ve run dtifit! Now you have an FA brain image, which is essential for the rest of the steps. It’s the go-to brain you’ll be looking at from now on for each subject. Now let’s get to bedpostX and a note on parallel computing.

**bedpostX and Parallel Computing**

* bedpostX finds all possible tracts from each voxel in the brain to each other voxel. Because of this, its base rate is *extremely* slow. Using the power of GPU on the Cluster, however, we can expedite this process by a factor of nearly one hundred.
* While bedpostx\_gpu, the newest command, takes only 15 minutes (as opposed to the 24 hours of past iterations), when multiplying this time by your subject count the time spent can be quite high. Because of this, computing in parallel is recommended for this and any other jobs that take more than a few minutes. Contact Research Computing for assistance with parallel computing.

To run bedpostX with GPUs, we need to create a GPU-based environment using a script. Here is an example of a parallel submission script:

#!/bin/bash

while read i

do

sed "s/SUBJECTNAME/$i/g" template.bash > submit$i.bash

sbatch submit$i.bash

sleep 1

done</scratch/myusername/scripts/sub\_folder\_names

While here is template.bash, the script to be submitted which creates the GPU environment and runs bedpostx\_gpu.

#!/bin/bash

#SBATCH -N 1

#SBATCH -n 1

#SBATCH -p gpu

#SBATCH --gres=gpu:p100:1

#SBATCH --output=SUBJECTNAME.out

#SBATCH --error=SUBJECTNAME.err

#SBATCH -J SUBJECTNAME\_gpu

cd /scratch/mysubjectfolders/SUBJECTNAME

module load fsl/6.0.3-cuda9.1

module load cuda/9.1

export FSL\_GPU=p100

. ${FSLDIR}/etc/fslconf/fsl.sh

bedpostx\_gpu runbedpostx -n 3 -w 1 -b 1000

NOTE: If you don’t understand how/why these scripts work, please schedule a Consultation with Research Computing on parallel scripting. Understanding these concepts is essential for efficient and safe computing on the Cluster.

**A few things before you run this script:**

1. Within your scripts directory, create a text file called sub\_folder\_names and put each subject directory name on a new line. The first few lines might read like this:

sub-001

sub-002

sub-003

…

1. Within each subject directory, make a directory named runbedpostx and copy the following files into it:
2. Your bval file
3. Your bvec file
4. Your eddy\_correct output
5. Your \*mask.nii.gz from bet2

VERY IMPORTANT

Rename each of these files to:

1. bvals
2. bvecs
3. data.nii.gz  = Diff\_topup.nii
4. nodif\_brain\_mask.nii.gz = dti\_mask.nii

Respectively.

This is because bedpostx formatting reads for these files specifically.

After you run bedpostx\_gpu on the runbedpostx directory, you’ll notice a new directory is created called runbedpostx\_bedpostX. This is your output directory. This should contain a few different types of files, including those with headers “dyads,” “mean,” and “merged.”

Congratulations! You’re done with bedpostX.

./bedpostx\_gpu\_parallel.sh

View file: cat MCI\_020.out

**TBSS**

TBSS is a brief and self-explanatory process which is detailed on the FSL wiki following this link:

<https://fsl.fmrib.ox.ac.uk/fsl/fslwiki/TBSS/UserGuide>

Email me with questions at [r.burtonpatel@northeastern.edu](mailto:r.burtonpatel@northeastern.edu)

**Drawing Regions of Interest (ROIs)**

The point of DTI is to observe the connectivity between two different Regions of Interest within the brain, or ROIs. You can use FSL commands to find ROIs, or hand-draw them yourself. Each of these will be specific to your lab or organization, but here is a basic guide to hand-drawing ROIs.

Open fsleyes and go to File->Add standard. Depending on the space of your brain images you will use different standards; in the author's case FMRIB58\_FA\_1mm was used.

Go to Tools->Edit Mode. This will enable you to draw your ROI.

Use the pen and eraser tool to construct your ROI. You’ll probably have a template based on the particular Region you’re looking at, and each Region differs from the rest. The author recommends this video for general ROI construction tips, and specifics will come from your lab or organization.

<https://www.youtube.com/watch?v=Vaj7BBxqXt0&t=221s>

Make sure to save your ROIs once you’ve drawn them by clicking the save icon next to the file name.

If you can’t see the file name list, go to Settings->Ortho View 1->Overlay List and select Overlay List.

ADD:

Bvec, bval, dti\_mask, diff\_topup\_bet

**Flirt, Fnirt, Invwarp, and Applywarp**

These 4 steps are used to prepare your ROI (drawn in standard space) to be converted to native space (fitted to each subject’s individual brain). This is a lot easier than drawing an ROI for each subject.

**Flirt**

flirt takes the subject brain and the standard brain and makes a transformation matrix to match one to the other.

Format:

flirt -in <FA\_brain> -ref <standard\_brain> -out <output\_name> -omat <output\_matrix\_name>

With subject-001\_FA.nii.gz as our example:

flirt -in dti\_FA.nii.gz -ref ${FSLDIR}/data/standard/FMRIB58\_FA\_1mm.nii.gz -out OA\_091\_FA\_flirted -omat OA\_091\_FA\_flirted\_mat

The important output of this is the matrix.

**Fnirt**

fnirt gives you a warp coefficient based on the transformation matrix which can be applied to ROIs.

Format:

fnirt --in=<FA\_brain> --aff=<flirt\_matrix> --config=<config\_file> --cout=<output\_name> --logout=<output\_log\_name> --ref=<standard\_brain>

NOTE: The config file is a helper tool. More information can be found here:

<https://fsl.fmrib.ox.ac.uk/fsl/fslwiki/FNIRT/UserGuide#Configuration_files>

Example:

fnirt --in=dti\_FA.nii.gz --aff=OA\_091\_FA\_flirted\_mat --config=${FSLDIR}/etc/flirtsch/FA\_2\_FMRIB58\_1mm.cnf --cout=OA\_091\_FA\_warpcoeff --logout=OA\_091\_FA\_to\_FMRIB58\_FA\_1mm.log --ref=${FSLDIR}/data/standard/FMRIB58\_FA\_1mm.nii.gz

**Invwarp**

Invwarp reverses non-linear mapping, and its output is necessary for the coming steps. NOTE: Invwarp is computationally intensive and may take longer than the previous steps.

Format:

invwarp -w <warp\_coefficient> -o <output\_name> -r <FA\_brain>

Example:

invwarp -w OA\_073\_FA\_warpcoeff.nii.gz -o OA\_073\_FA\_invwarp -r dti\_FA

invwarp -w OA\_091\_FA\_warpcoeff.nii.gz -o OA\_091\_AD\_invwarp -r dti\_AD

invwarp -w OA\_091\_FA\_warpcoeff.nii.gz -o OA\_091\_RD\_invwarp -r dti\_RD

invwarp -w OA\_091\_FA\_warpcoeff.nii.gz -o OA\_091\_MD\_invwarp -r dti\_MD

**Applywarp**

Applywarp applies the warp you’ve been making to the ROI itself and maps it onto the subject FA brain. The outputs of Applywarp are the ROIs you’ll use as reference for tractography, and are unique to each subject.

Format:

applywarp -i <ROI\_file> -o <output\_name> -r <FA\_brain> -w <invwarp\_output>

Example:

applywarp -i IFG\_PO\_left\_handdrawn\_standard.nii.gz -o IFG\_PO\_left\_handdrawn\_native -r subject-001\_FA.nii.gz -w subject-001\_FA\_invwarp.nii.gz

applywarp -i PMC\_mask\_L\_BA4a.nii.gz -o AD\_PMCL\_native.nii.gz -r dti\_AD -w MCI\_014\_AD\_invwarp.nii.gz

applywarp -i PMC\_mask\_R\_BA4a.nii.gz -o AD\_PMCR\_native.nii.gz -r dti\_AD -w MCI\_014\_AD\_invwarp.nii.gz

applywarp -i PMC\_mask\_L\_BA4a.nii.gz -o AD\_PMCL\_native.nii.gz -r dti\_MD -w MCI\_014\_MD\_invwarp.nii.gz

applywarp -i PMC\_mask\_R\_BA4a.nii.gz -o AD\_PMCR\_native.nii.gz -r dti\_MD -w MCI\_014\_MD\_invwarp.nii.gz

applywarp -i PMC\_mask\_L\_BA4a.nii.gz -o AD\_PMCL\_native.nii.gz -r dti\_FA -w MCI\_014\_FA\_invwarp.nii.gz

applywarp -i PMC\_mask\_R\_BA4a.nii.gz -o AD\_PMCR\_native.nii.gz -r dti\_FA -w MCI\_014\_FA\_invwarp.nii.gz

applywarp -i PMC\_mask\_L\_BA4a.nii.gz -o AD\_PMCL\_native.nii.gz -r dti\_RD -w MCI\_014\_RD\_invwarp.nii.gz

applywarp -i PMC\_mask\_R\_BA4a.nii.gz -o AD\_PMCR\_native.nii.gz -r dti\_RD -w MCI\_014\_RD\_invwarp.nii.gz

applywarp -i SMA\_mask\_L\_BA3b.nii.gz -o AD\_SMAL\_native.nii.gz -r dti\_AD -w MCI\_014\_AD\_invwarp.nii.gz

applywarp -i SMA\_mask\_R\_BA3b.nii.gz -o AD\_SMAR\_native.nii.gz -r dti\_AD -w MCI\_014\_AD\_invwarp.nii.gz

applywarp -i SMA\_mask\_L\_BA3b.nii.gz -o AD\_SMAL\_native.nii.gz -r dti\_MD -w MCI\_014\_MD\_invwarp.nii.gz

applywarp -i SMA\_mask\_R\_BA3b.nii.gz -o AD\_SMAR\_native.nii.gz -r dti\_MD -w MCI\_014\_MD\_invwarp.nii.gz

applywarp -i SMA\_mask\_L\_BA3b.nii.gz -o AD\_SMAL\_native.nii.gz -r dti\_FA -w MCI\_014\_FA\_invwarp.nii.gz

applywarp -i SMA\_mask\_R\_BA3b.nii.gz -o AD\_SMAR\_native.nii.gz -r dti\_FA -w MCI\_014\_FA\_invwarp.nii.gz

applywarp -i SMA\_mask\_L\_BA3b.nii.gz -o AD\_SMAL\_native.nii.gz -r dti\_RD -w MCI\_013\_RD\_invwarp.nii.gz

applywarp -i SMA\_mask\_R\_BA3b.nii.gz -o AD\_SMAR\_native.nii.gz -r dti\_RD -w MCI\_013\_RD\_invwarp.nii.gz

Note the lack of distinction of subject number for the output. This is because it is good practice to have your native applywarp-ed ROIs in individually named subject folders.

**ProbtrackX**

ProbtrackX is the end of this pipeline. It uses bedpostX results and two ROIs to map a path between them, effectively creating your desired tractography image. It is computationally intensive and can take fifteen minutes or longer per subject, per ROI pair. As these numbers grow multiplicatively with subject and ROI count, use of parallel computing is strongly recommended.

Before running, create a new directory in each subject folder called probtrackx. Within it, create a directory **for each pathway between ROIs.** For example if your ROIs were called A, B, and C, your directories would look like:

AtoB

BtoA

AtoC

CtoA

BtoC

CtoB

Make sure you also have your ROIs in an accessible location, as well as your bedpostX folder on hand (you will need all files including “merged” as well as the nodif\_brain\_mask)

Example Format for 2 ROIs (used in an example study; your steplength, fiberthresh and other values may vary) (all bolded values must be specified):

probtrackx2  -x **<1st ROI>**  -l --onewaycondition -c 0.2 -S 2000 --steplength=0.5 -P 5000 --fibthresh=0.01 --distthresh=0.0 --sampvox=0.0 --forcedir --opd -s **<all bedpostX merged files, format specifics in example below>** -m **<your nodif\_brain\_mask from bedpostX>**  --dir=**<directory to contain this specific pathway, as discussed before>** --waypoints=**<2nd ROI>** --waycond=AND

probtrackx2 -x AD\_BCC\_native.nii.gz -l --onewaycondition -c 0.2 -S 2000 --steplength=0.5 -P 5000 --fibthresh=0.01 --distthresh=0.0 --sampvox=0.0 --forcedir --opd -s merged -m ./nodif\_brain\_mask --dir=/scratch/tinney.e/bedpostX/datasets/MCI\_001/probtractX/AD\_BCC2PMCL/ --waypoints=AD\_PMCL\_native --waycond=AND

probtrackx2 -x AD\_BCC\_native.nii.gz -l --onewaycondition -c 0.2 -S 2000 --steplength=0.5 -P 5000 --fibthresh=0.01 --distthresh=0.0 --sampvox=0.0 --forcedir --opd -s merged -m ./nodif\_brain\_mask --dir=/scratch/tinney.e/bedpostX/datasets/MCI\_001/probtractX/AD\_BCC2PMCR/ --waypoints=AD\_PMCR\_native --waycond=AND

probtrackx2 -x FA\_BCC\_native.nii.gz -l --onewaycondition -c 0.2 -S 2000 --steplength=0.5 -P 5000 --fibthresh=0.01 --distthresh=0.0 --sampvox=0.0 --forcedir --opd -s merged -m ./nodif\_brain\_mask --dir=/scratch/tinney.e/bedpostX/datasets/MCI\_001/probtractX/FA\_BCC2PMCL/ --waypoints=FA\_PMCL\_native --waycond=AND

probtrackx2 -x FA\_BCC\_native.nii.gz -l --onewaycondition -c 0.2 -S 2000 --steplength=0.5 -P 5000 --fibthresh=0.01 --distthresh=0.0 --sampvox=0.0 --forcedir --opd -s merged -m ./nodif\_brain\_mask --dir=/scratch/tinney.e/bedpostX/datasets/MCI\_001/probtractX/FA\_BCC2PMCR/ --waypoints=FA\_PMCR\_native --waycond=AND

probtrackx2 -x MD\_BCC\_native.nii.gz -l --onewaycondition -c 0.2 -S 2000 --steplength=0.5 -P 5000 --fibthresh=0.01 --distthresh=0.0 --sampvox=0.0 --forcedir --opd -s merged -m ./nodif\_brain\_mask --dir=/scratch/tinney.e/bedpostX/datasets/MCI\_001/probtractX/MD\_BCC2PMCL/ --waypoints=MD\_PMCL\_native --waycond=AND

probtrackx2 -x MD\_BCC\_native.nii.gz -l --onewaycondition -c 0.2 -S 2000 --steplength=0.5 -P 5000 --fibthresh=0.01 --distthresh=0.0 --sampvox=0.0 --forcedir --opd -s merged -m ./nodif\_brain\_mask --dir=/scratch/tinney.e/bedpostX/datasets/MCI\_001/probtractX/MD\_BCC2PMCR/ --waypoints=MD\_PMCR\_native --waycond=AND

probtrackx2 -x RD\_BCC\_native.nii.gz -l --onewaycondition -c 0.2 -S 2000 --steplength=0.5 -P 5000 --fibthresh=0.01 --distthresh=0.0 --sampvox=0.0 --forcedir --opd -s merged -m ./nodif\_brain\_mask --dir=/scratch/tinney.e/bedpostX/datasets/MCI\_001/probtractX/RD\_BCC2PMCL/ --waypoints=RD\_PMCL\_native --waycond=AND

probtrackx2 -x RD\_BCC\_native.nii.gz -l --onewaycondition -c 0.2 -S 2000 --steplength=0.5 -P 5000 --fibthresh=0.01 --distthresh=0.0 --sampvox=0.0 --forcedir --opd -s merged -m ./nodif\_brain\_mask --dir=/scratch/tinney.e/bedpostX/datasets/MCI\_001/probtractX/RD\_BCC2PMCR/ --waypoints=RD\_PMCR\_native --waycond=AND

probtrackx2 -x AD\_BCC\_native.nii.gz -l --onewaycondition -c 0.2 -S 2000 --steplength=0.5 -P 5000 --fibthresh=0.01 --distthresh=0.0 --sampvox=0.0 --forcedir --opd -s merged -m ./nodif\_brain\_mask --dir=/scratch/tinney.e/bedpostX/datasets/MCI\_001/probtractX/AD\_BCC2SMAL/ --waypoints=AD\_SMAL\_native --waycond=AND

probtrackx2 -x AD\_BCC\_native.nii.gz -l --onewaycondition -c 0.2 -S 2000 --steplength=0.5 -P 5000 --fibthresh=0.01 --distthresh=0.0 --sampvox=0.0 --forcedir --opd -s merged -m ./nodif\_brain\_mask --dir=/scratch/tinney.e/bedpostX/datasets/MCI\_001/probtractX/AD\_BCC2SMAR/ --waypoints=AD\_SMAR\_native --waycond=AND

probtrackx2 -x FA\_BCC\_native.nii.gz -l --onewaycondition -c 0.2 -S 2000 --steplength=0.5 -P 5000 --fibthresh=0.01 --distthresh=0.0 --sampvox=0.0 --forcedir --opd -s merged -m ./nodif\_brain\_mask --dir=/scratch/tinney.e/bedpostX/datasets/MCI\_001/probtractX/FA\_BCC2SMAL/ --waypoints=FA\_SMAL\_native --waycond=AND

probtrackx2 -x FA\_BCC\_native.nii.gz -l --onewaycondition -c 0.2 -S 2000 --steplength=0.5 -P 5000 --fibthresh=0.01 --distthresh=0.0 --sampvox=0.0 --forcedir --opd -s merged -m ./nodif\_brain\_mask --dir=/scratch/tinney.e/bedpostX/datasets/MCI\_001/probtractX/FA\_BCC2SMAR/ --waypoints=FA\_SMAR\_native --waycond=AND

probtrackx2 -x MD\_BCC\_native.nii.gz -l --onewaycondition -c 0.2 -S 2000 --steplength=0.5 -P 5000 --fibthresh=0.01 --distthresh=0.0 --sampvox=0.0 --forcedir --opd -s merged -m ./nodif\_brain\_mask --dir=/scratch/tinney.e/bedpostX/datasets/MCI\_001/probtractX/MD\_BCC2SMAL/ --waypoints=MD\_SMAL\_native --waycond=AND

probtrackx2 -x MD\_BCC\_native.nii.gz -l --onewaycondition -c 0.2 -S 2000 --steplength=0.5 -P 5000 --fibthresh=0.01 --distthresh=0.0 --sampvox=0.0 --forcedir --opd -s merged -m ./nodif\_brain\_mask --dir=/scratch/tinney.e/bedpostX/datasets/MCI\_001/probtractX/MD\_BCC2SMAR/ --waypoints=MD\_SMAR\_native --waycond=AND

probtrackx2 -x RD\_BCC\_native.nii.gz -l --onewaycondition -c 0.2 -S 2000 --steplength=0.5 -P 5000 --fibthresh=0.01 --distthresh=0.0 --sampvox=0.0 --forcedir --opd -s merged -m ./nodif\_brain\_mask --dir=/scratch/tinney.e/bedpostX/datasets/MCI\_001/probtractX/RD\_BCC2SMAL/ --waypoints=RD\_SMAL\_native --waycond=AND

probtrackx2 -x RD\_BCC\_native.nii.gz -l --onewaycondition -c 0.2 -S 2000 --steplength=0.5 -P 5000 --fibthresh=0.01 --distthresh=0.0 --sampvox=0.0 --forcedir --opd -s merged -m ./nodif\_brain\_mask --dir=/scratch/tinney.e/bedpostX/datasets/MCI\_001/probtractX/RD\_BCC2SMAR/ --waypoints=RD\_SMAR\_native --waycond=AND

Note: No practical example is given in this case due to high variability of input type, number, etc. The omission is made with the intent to not cause confusion.

Outputs for ProbtrackX are numerous, but most importantly include files called fdtpaths. These are the files used for the rest of the pipeline, and are distinguished by their location within labeled pathway directories.

**Thresholding, Binarizing, and Analysis**

Thresholding is run on the path and selects a percentile of voxels and makes a new pathway with all voxels below that threshold removed. It helps remove extraneous or confounding activity.

Example format:

fslmaths <fdtpaths> -thrP <threshold\_percentile> <output\_filename>

Practical example:

fslmaths fdt\_paths.nii.gz -thrP 50 fdt\_paths\_thr50

In this example, the lower 50th percentile of active voxels would be removed in the new file.

Binarizing takes all different levels in voxels and sets them to either 1 (non-zero) or 0 (no activity). Run it on your thresholded pathway.

Example format:

fslmaths <subject\_thresholded\_file> -bin <output\_filename>

Practical example:

fslmaths fdt\_paths\_thr50.nii.gz -bin fdt\_paths\_thr50\_bin

This will solidify the color of your pathway.

Lastly, we will turn the pathway into a mask so we can run analysis on it.

We will use fslmaths to map the pathway over the subject FA brain.

Example format:

fslmaths <subject\_FA> -mas <subject\_thresholded\_binarized\_pathway) <output\_filename>

Practical Example:

fslmaths dti\_AD.nii.gz -mas fdt\_paths\_thr50\_bin fdt\_thr50\_FA

fslmaths dti\_FA.nii.gz -mas fdt\_paths\_thr50\_bin fdt\_thr50\_FA

fslmaths dti\_MD.nii.gz -mas fdt\_paths\_thr50\_bin fdt\_thr50\_FA

fslmaths dti\_RD.nii.gz -mas fdt\_paths\_thr50\_bin fdt\_thr50\_FA

Your output file is ready for analysis by your institution. Congratulations! You’ve completed this pipeline.

FSL is a complicated and multifaceted software interface. If you have any questions, consult the FSLWiki or email me at [r.burtonpatel@northeastern.edu](mailto:r.burtonpatel@northeastern.edu). Additionally, if you have any recommendations for how this user guide can be improved, please don’t hesitate to contact me.

Activate script:

Chmod +x

./ script name