AFNI PROCESSING MANUAL

Center for Advanced Brain Imaging 831 Marietta Street, Atlanta, GA 30318

Updated: March 2, 2018

Table of Contents

Introduction

Welcome to CABI’s AFNI processing manual. **A**nalysis of **F**unctional **N**euro**I**mages is a software package developed by Robert Cox and maintained by the NIH. It is a set of functions that we use to process and analyze fMRI data in various ways. This manual will show you how to use our standard AFNI processing stream developed in CABI to do a basic analysis of fMRI data.

This manual will take you from the very beginning of fMRI analysis to what is called second-level analysis where you calculate within- or between-subjects differences in contrasts of interest. We will be using a small set of fMRI data collected while participants were performing a simple finger tapping task. If you are new to AFNI (or neuroimaging in general), we encourage you to start at the beginning of this manual and work your way through the end. Even if your own study is very different from the design here, a thorough understanding of this processing stream will prepare you to run more advanced analyses that are modified to suit the needs of your own data. If you have any questions, please contact CABI's current researcher assigned to assist on AFNI processing.

Note that we run AFNI on a set of Linux computers at CABI. We operate AFNI via a set of commands run through the terminal. If you are new to Linux and working through a terminal, we recommend reviewing Section 6 – Bash scripting before you proceed with AFNI.

fMRI Task Overview

For purposes of demonstration, you will be working with fMRI data collected in a block design while participants performed a finger tapping task. This simple task produces strong activation patterns in the primary motor cortex and cerebellum, which you will be able to see upon completion of second-level analysis.

A total of 3 participants completed this task. Each participant completed three different conditions randomly ordered: Tapping with left fingers (L), tapping with right fingers (R), and rest (N). The duration of each condition was 12 seconds. For each participant, four runs of functional scans (i.e., scans that recorded the fMRI signal while participants performed the tapping task) and one anatomical scan (i.e., the 3D anatomical image of the brain) were collected. These data are saved on a permanent location on the CABI server.

Getting Started

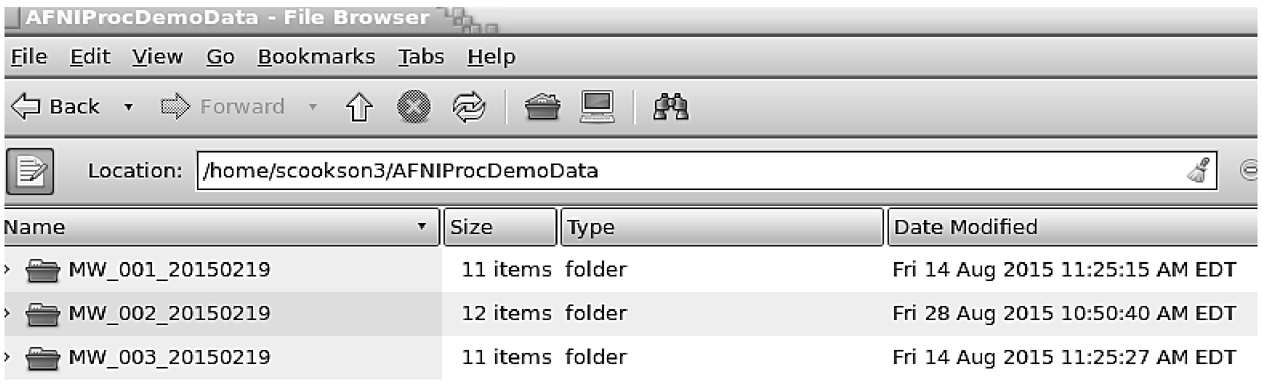
Before you begin, you need to download the latest version of the AFNI Processing Stream backbone files. These can be found on http://www.cabi.gatech.edu/CABI/resources/process-guides/ The files contained in this set are example/starter scripts that are designed to take one from the conversion of the raw DICOMs all the way to the individual analysis level with minimal additional coding requirements. Unzip and store these files in a folder on your CABI computer account. In this manual, this folder is referred to as "afni_processing".

Preprocessing

Step 1 – DICOM to NIFTI conversion

Raw fMRI data is saved in .dcm (dicom) files. Typically these .dcm files correspond to individual slices, and the resulting 3D image (or a time-series of 3D images) is saved in a .nii (nifti) file. (https://wiki.cam.ac.uk/bmuwiki/FMRI). The first step of fMRI analysis is converting our raw .dcm data to .nii format. To do this, you need to edit the data file and conversion script, and then run the conversion script

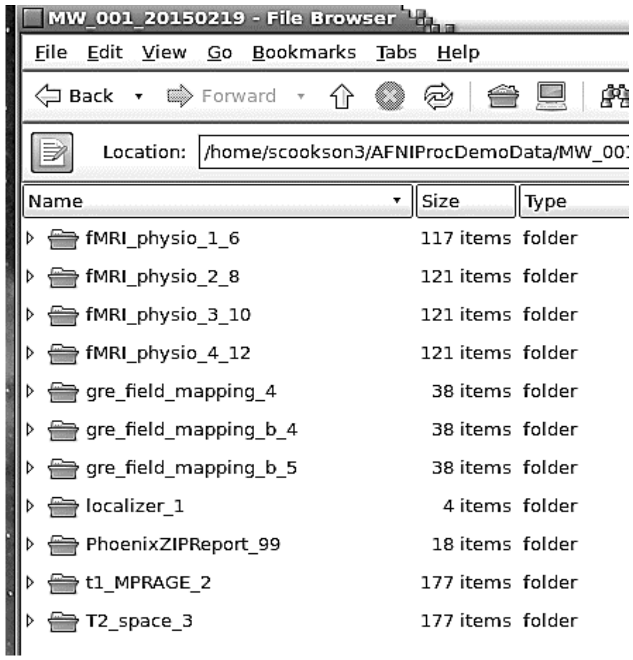
The raw data are stored here: /home/scookson3/AFNIProcDemoData
Here you see there are 3 folders for the 3 subjects you will analyze: MW_001_*, MW_002_*, and MW_003_*



/home/scookson3/AFNIProcDemoData

Name	Size	Type	Date Modified
MW_001_20150219	11 items	folder	Fri 14 Aug 2015 11:25:15 AM EDT
MW_002_20150219	12 items	folder	Fri 28 Aug 2015 10:50:40 AM EDT
MW_003_20150219	11 items	folder	Fri 14 Aug 2015 11:25:27 AM EDT

If you look inside any of the folders, you will find additional sets of folders that contain the data for the different scans (structural scans, functional scans, localizers, etc.). Here is what subject 1's folders look like:



Location: /home/scookson3/AFNIProcDemoData/MW_001

Name	Size	Type
fMRI_physio_1_6	117 items	folder
fMRI_physio_2_8	121 items	folder
fMRI_physio_3_10	121 items	folder
fMRI_physio_4_12	121 items	folder
gre_field_mapping_4	38 items	folder
gre_field_mapping_b_4	38 items	folder
gre_field_mapping_b_5	38 items	folder
localizer_1	4 items	folder
PhoenixZIPReport_99	18 items	folder
t1_MPRAGE_2	177 items	folder
T2_space_3	177 items	folder

The data that is relevant here is the t1 data (t1_MPRAGE_2) and the 4 functional task runs (fMRI_physio_1_6, fMRI_physio_2_8, fMRI_physio_3_10, fMRI_physio_4_12).

Step 1a – Editing the data file

The data file in the afni_processing folder contains a list of information for the data for each subject that will be converted from raw file format to .nii format. The CABI stream supports two different formats of conversion for your data. One uses a CSV file, and one uses an Excel file; the format for each of these is unique. Both CSV and Excel are equally effective; the one you use is based on your personal preference. Below we provide instructions for both formats. Note if you open a CSV file, you may be prompted with a text import option. Just click 'ok' to continue.

<u>CSV data file (exampleList_CSVversion.csv)</u> (this information can also be found in the exampleListCSV_readme.txt file from the backbone files)

The first line is a set of headers indicating what is contained in each column.

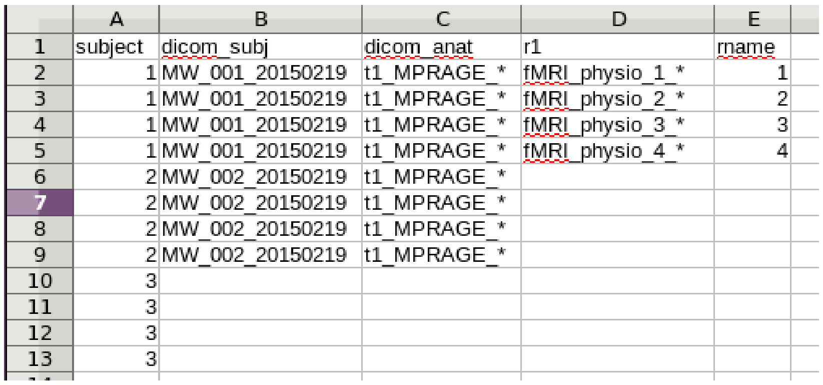
- subject: the identifier for each of your subjects. Typically the subject number, but does not have to be. Note that this will serve as the filename for each individual subject's converted data, and will trickle down through all of your analyses.

- dicom_subj: the folder name within the parent folder containing all of the raw data that corresponds to a given subject's scan. These are the names outputted by the scanner. Do NOT include the forward slash ("/") at the end of the folder name.
- dicom_anat: the name of the folder containing the anatomical scan DICOM files. This value can include wildcards.
- r1: the names of the folders containing each run of functional MRI DICOM files. These values can include wildcards.
- rname: this is the identifier that you would like to give the converted NIFTI files for each run. It will be appended to the built-in prefix, which is “fMRI_run”. The identifier should be the unique part of the name that identifies each particular file. Many people use numbers for this part so that the formatting of the rnames is “fMRI_run1”, “fMRI_run2”, etc.

Each line should represent a single run of data for a single subject. Note that in this analysis, there were 4 functional runs and 1 structural run per subject. The number of rows per subject in the CSV format is based on the number of functional runs. Even though there is only one structural run, it will fill each of the four cells in the dicom_anat column. Because of the consistency of the format of the outputted data coming from the scanner, you can use wildcards to let you copy and paste much of the information for a single subject for every subject.

Once you have set up your CSV file, you will need to go through each raw data folder and make sure that you have explicitly indicated which run to use in the case that a scan needed to be repeated – this sometimes happens when the researcher needs to restart a run, or when they decide to collect a second structural scan if the first was poor quality. In this case, you will not be able to use any wildcards, because a wildcard will not differentiate between the first (bad) scan and the second (good) scan (e.g., see subject MW_002) Also double-check the spelling of the folder name containing your subject's scans; failure to check both of these steps will prevent the converter from completing that subject's NIFTI setup.

Here is an example of a partially-complete CSV data file:



	A	B	C	D	E
1	subject	dicom_subj	dicom_anat	r1	rname
2	1	MW_001_20150219	t1_MPRAGE_*	fMRI_physio_1_*	1
3	1	MW_001_20150219	t1_MPRAGE_*	fMRI_physio_2_*	2
4	1	MW_001_20150219	t1_MPRAGE_*	fMRI_physio_3_*	3
5	1	MW_001_20150219	t1_MPRAGE_*	fMRI_physio_4_*	4
6	2	MW_002_20150219	t1_MPRAGE_*		
7	2	MW_002_20150219	t1_MPRAGE_*		
8	2	MW_002_20150219	t1_MPRAGE_*		
9	2	MW_002_20150219	t1_MPRAGE_*		
10	3				
11	3				
12	3				
13	3				

XLSX data file (exampleList_XLSXversion.xlsx) (this information can also be found in the exampleListXLSX_readme.txt file from the backbone files.

The first line is a set of headers indicating what is contained in each column.

- SubjectName: the identifier for each of your subjects. Typically the subject number, but does not have to be. CANNOT BE 0. Note that this will serve as the filename for each individual subject's converted data, and will trickle down through all of your analyses.

- Subject_Dicom_folder: the folder name within the parent folder containing all of the raw data that corresponds to a given subject's scan. These are the names outputted by the scanner. Do NOT include the forward slash ("/") at the end of the folder name.

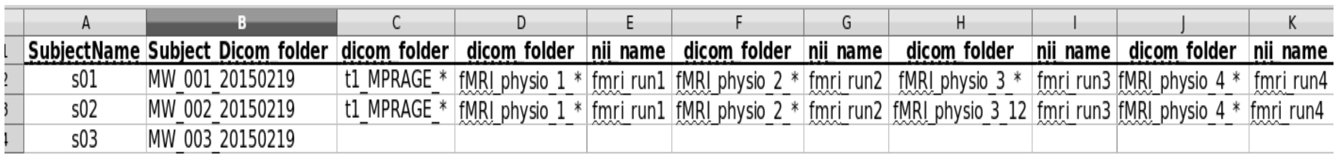
- dicom_folder: the name of the folder containing the DICOM files for a run of interest. This value can include wildcards.

- nii_name: this is the identifier that you would like to give the converted NIFTI files for each run. It will be appended to the built-in prefix, which is “fMRI_run”. The identifier should be the unique part of the name that identifies each particular file. Many people use numbers for this part so that the formatting of the nii_names is “fMRI_run1”, “fMRI_run2”, etc.

Each line should represent a single subject. You can repeat pairs of "dicom_folder" and "nii_name" entries for the number of runs you wish to analyze for that subject; subjects in a single excel file do not need to have the same number of runs. Note that the first set of files is your structural scan; this scan is automatically named "T1", so no nii_name value is included for this scan. Because of the consistency of the format of the outputted data coming from the scanner, you can use wildcards to let you copy and paste much of the information for a single subject for every subject.

Once you have set up your XLSX file, you will need to go through each raw data folder and make sure that you have explicitly indicated which run to use in the case that a scan needed to be repeated – this sometimes happens when the researcher needs to restart a run, or when they decide to collect a second structural scan if the first was poor quality. In this case, you will not be able to use any wildcards, because a wildcard will not differentiate between the first (bad) scan and the second (good) scan (e.g., see subject MW_002). Also double-check the spelling of the folder name containing your subject's scans; failure to check both of these steps will prevent the converter from completing that subject's NIFTI setup.

Here is an example of a partially-complete XLSX data file:



	A	B	C	D	E	F	G	H	I	J	K
1	SubjectName	Subject_Dicom_folder	dicom_folder	dicom_folder	nii_name	dicom_folder	nii_name	dicom_folder	nii_name	dicom_folder	nii_name
2	s01	MW_001_20150219	t1_MPRAGE_*	fMRI_physio_1_*	fmri_run1	fMRI_physio_2_*	fmri_run2	fMRI_physio_3_*	fmri_run3	fMRI_physio_4_*	fmri_run4
3	s02	MW_002_20150219	t1_MPRAGE_*	fMRI_physio_1_*	fmri_run1	fMRI_physio_2_*	fmri_run2	fMRI_physio_3_12	fmri_run3	fMRI_physio_4_*	fmri_run4
4	s03	MW_003_20150219									

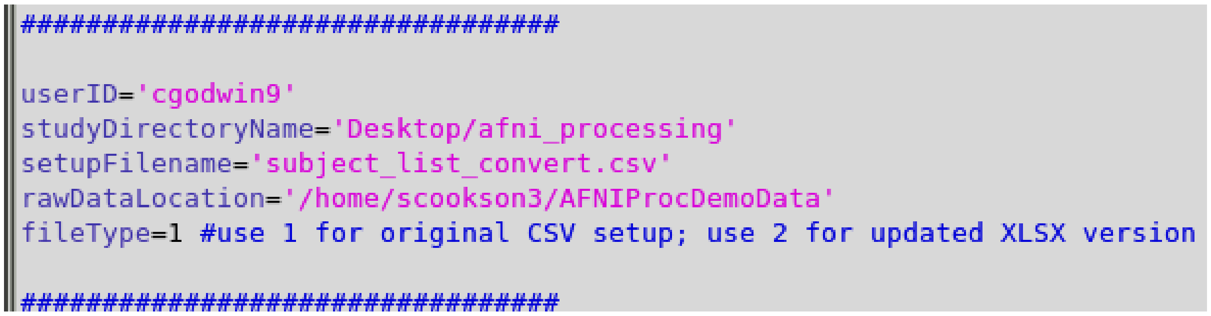
Step 1b – Editing the conversion script (conversion_script.sh)

We use a conversion script to actually convert the raw .dcm files to .nii format. The conversion script uses the information in the .csv or .xlsx files to correctly access the raw data.

The conversion script has two sections. The top section is the area that the user edits, and the bottom section runs the conversion. The script is set up so that the user only needs to put in the names of the files and folders that the conversion will need to run. These are each entered as strings, which are indicated by the single (') or double (")quotes.

- userID: your user ID on the CABI system
- studyDirectoryName: The folder that was created for the current analysis
- dataFilename: the name of the study-specific CSV file created in step 1.
- rawDataLocation: the full filepath of the raw data for the study (main directory)
- fileType: which data file structure you are using for the conversion. 1 for CSV, 2 for XLSX.

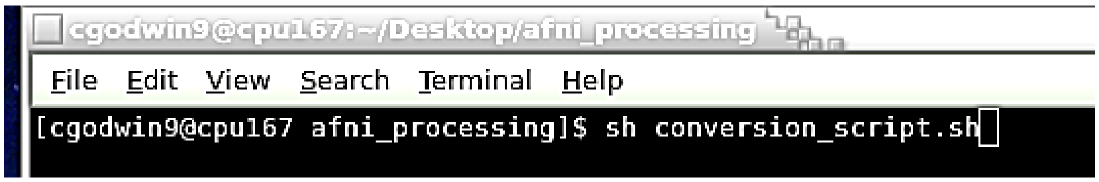
In the example below, the conversion is being run on the Desktop under the afni_processing folder, and the CSV setup option is selected. As you can see, I changed the name of the .csv file to 'subject_list_convert.csv'.



```
###################################

userID='cgodwin9'
studyDirectoryName='Desktop/afni_processing'
setupFilename='subject_list_convert.csv'
rawDataLocation='/home/scookson3/AFNIProcDemoData'
fileType=1 #use 1 for original CSV setup; use 2 for updated XLSX version

###################################
```

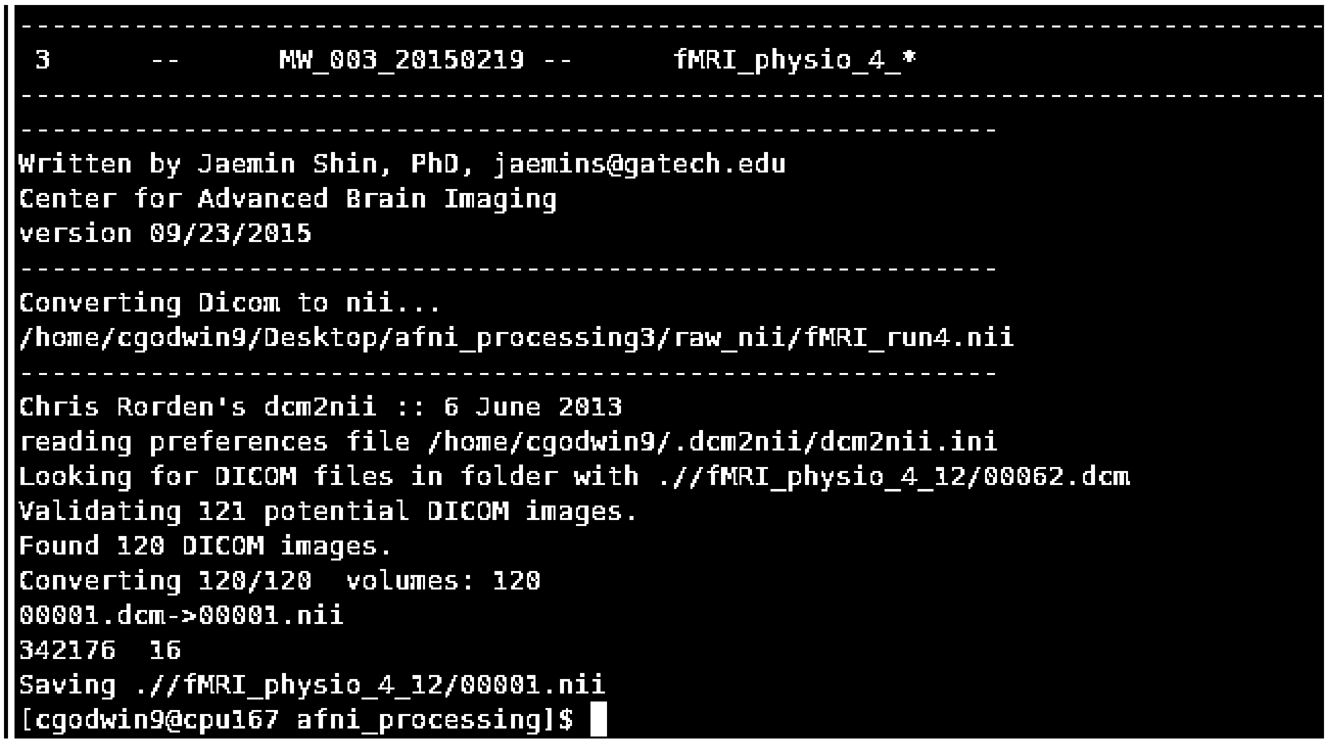
Step 1c – Running the conversion script

Once you have completed the steps above, go into Terminal and navigate to your study folder. From here, run the conversion script you edited for your specific study using the "sh" command (the process will take a few minutes):


cgodwin9@cpu167:~/Desktop/afni_processing
File Edit View Search Terminal Help

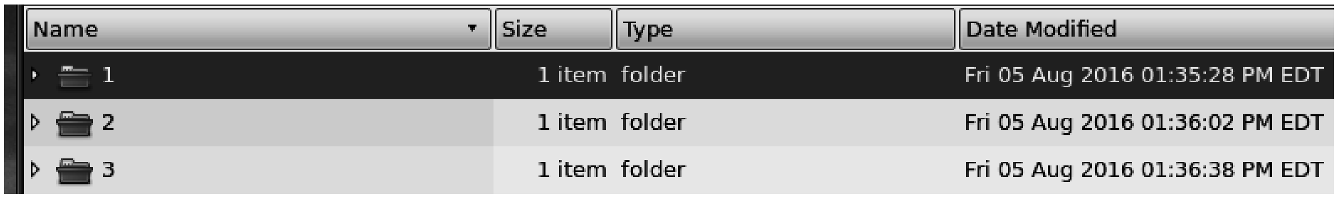
```
[cgodwin9@cpu167 afni_processing]$ sh conversion_script.sh
```

Example of typical output in the terminal as files convert:



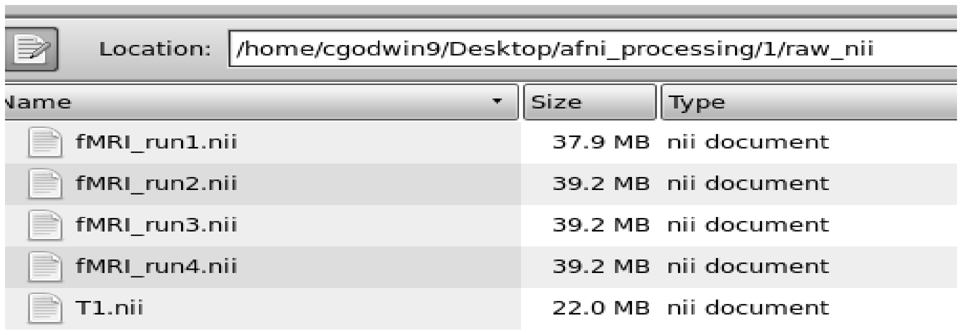
```
--------------------------------------------------------------------------------
 3      --      MW_003_20150219 --      fMRI_physio_4_*
--------------------------------------------------------------------------------
------------------------------------------------------------
Written by Jaemin Shin, PhD, jaemins@gatech.edu
Center for Advanced Brain Imaging
version 09/23/2015
------------------------------------------------------------
Converting Dicom to nii...
/home/cgodwin9/Desktop/afni_processing3/raw_nii/fMRI_run4.nii
------------------------------------------------------------
Chris Rorden's dcm2nii :: 6 June 2013
reading preferences file /home/cgodwin9/.dcm2nii/dcm2nii.ini
Looking for DICOM files in folder with .//fMRI_physio_4_12/00062.dcm
Validating 121 potential DICOM images.
Found 120 DICOM images.
Converting 120/120  volumes: 120
00001.dcm->00001.nii
342176  16
Saving .//fMRI_physio_4_12/00001.nii
[cgodwin9@cpu167 afni_processing]$ 
```

If the conversion is successful, you should see a number of additions to your study folder. Note that these folder names are the same as the subject number specified in the .csv file.



Name	Size	Type	Date Modified
1	1 item	folder	Fri 05 Aug 2016 01:35:28 PM EDT
2	1 item	folder	Fri 05 Aug 2016 01:36:02 PM EDT
3	1 item	folder	Fri 05 Aug 2016 01:36:38 PM EDT

There should now be a folder for each subject, labeled with the subject number indicated in your CSV file. Within each subject folder is a "raw_nii" folder. In this folder, you should see one file called T1.nii and an additional .nii file for each of your functional runs, named with the label you gave in your data file.



Location: /home/cgodwin9/Desktop/afni_processing/1/raw_nii

Name	Size	Type
fMRI_run1.nii	37.9 MB	nii document
fMRI_run2.nii	39.2 MB	nii document
fMRI_run3.nii	39.2 MB	nii document
fMRI_run4.nii	39.2 MB	nii document
T1.nii	22.0 MB	nii document

Step 2 – Stimulus Timing Files

Next, we need to create the stimulus timing files for this experiment. Stimulus timing files indicate when each of the events in your experiment take place. How many files you have depends on your experimental conditions. Here we will create just two timing files for each participant - one that indicates when people tapped with their left hand, and one that indicates when they tapped with their right. You can follow along with the instructions below with the example timing files for subject 1 provided in the backbone files download.

The format of each of these files is as follows: each file has one line for each functional run (recall that our data have 4 runs). On each line, then, we indicate the start time and duration, in seconds, for each occurrence of the event in that run. The start time and duration are separated by a colon, and events are separated by spaces. For example, if a participant started tapping with their left hand at 10s, 30s, 50s, and 70s and tapped for 10 seconds each time during run 1, then the first line of their timing file would look like this:

```
10:10 30:10 50:10 70:10
```

To make the timing files, we need more information. Specifically, we need to know when each person started each block, and whether they tapped with the left or right hand (or didn't tap at all). This information is contained in the AFNIDemoExcelData.xlsx spreadsheet in the for_timing_files folder. Open this file and take a look at its contents. The columns are labelled according to what data they contain. Each row represents a single block of our experiment, and for each block, you are given whether they tapped with their left (L), right (R), or rest (N). You are also given the time of the scan (in seconds) at which that block started. Recall that all blocks were 12 seconds long.

First, let's do an example file for subject 1's left tapping file. Open a new text file and look at the excel data for subject 1, run 1. Find the first block that logged a (L)eft tap and note the start time: the subject first tapped left at 100s into the experiment. In the text file, then, on the first line, write:

```
100:12
```

This indicates that the subject tapped starting at 100s and continued for 12s. Now, find the start time for the second block of L tapping: 212s, the final block of the run. Add this value to your text file the same way, so that the first line looks like this:

```
100:12 212:12
```

That completes the first run of blocks. Now, look at the second run and find the first L at 184s. To mark that we have started a new run, add this value to a new line in your text file:

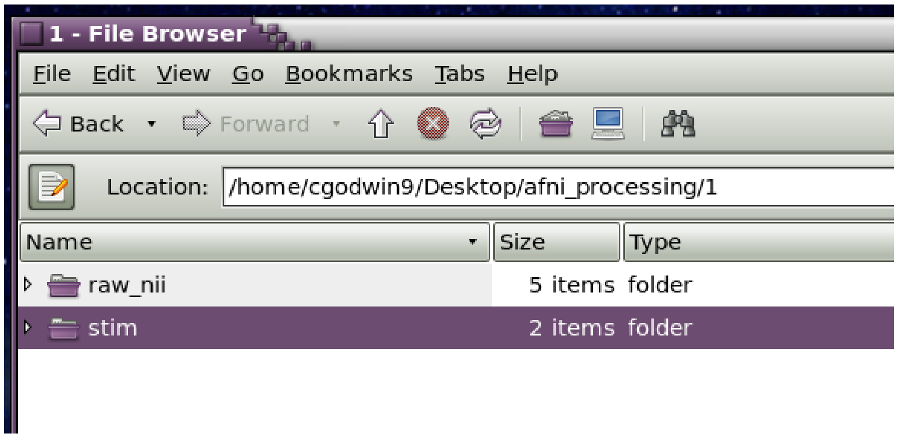
```
100:12 212:12
184:12
```

Repeat this process for each of the blocks for each run for subject 1. Your final text file for this subject should look like this:

```
100:12 212:12
184:12 212:12
16:12 156:12
100:12 212:12
```

This is also shown in the example file L.1D for subject 1.

Now, we need to save the file. The analysis stream needs a consistent filesystem convention to run automatically, so we need a system for saving each subject’s timing files in a consistent location and order. Go into your folder and go into the folder for subject 1. You should see folder called “raw_nii” that was created during the conversion process. Add a new folder called “stim” here; the processing scripts will look for your stimulus timing files for each subject under this name:



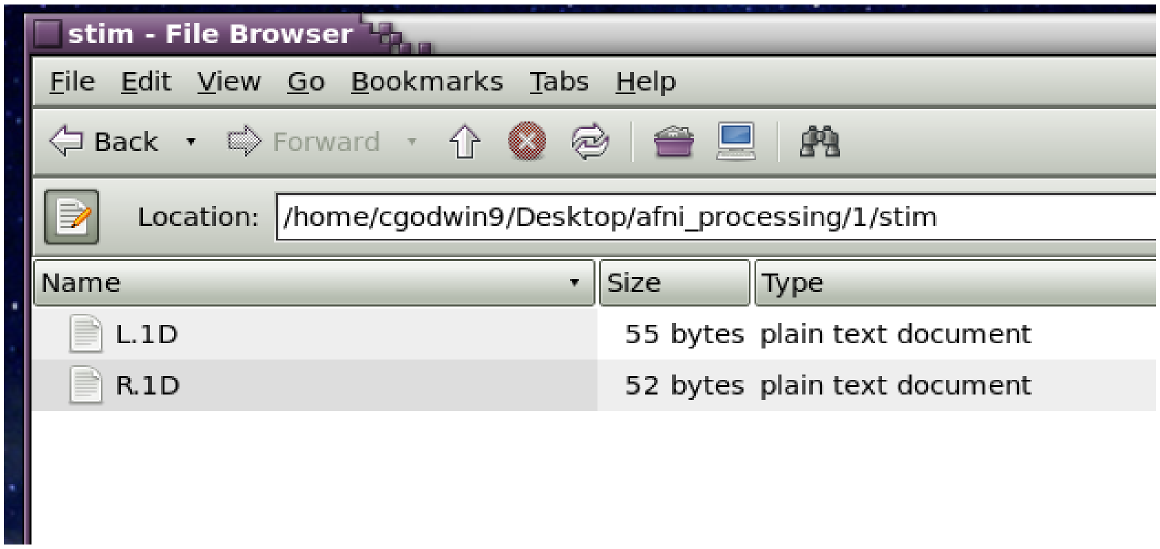
Location: /home/cgodwin9/Desktop/afni_processing/1

Name	Size	Type
raw_nii	5 items	folder
stim	2 items	folder

Save your newly created text file to this folder under the name “L.1D” (a 1D file is a different type of text file than .txt; AFNI will read any type of text file, but using 1D specifically for stimulus timing filetypes ensures that the only files the processing stream reads for stimulus timings are the ones you made for that purpose). When naming stimulus timing files, you will want to generally use a simple, systematic naming system. In this experiment, the simplest convention is to simply use “L” for our left-tap timing files, and “R” for our right-tap timing files. For more complicated experiments, it can be helpful to name your timing files according to the combination of conditions represented by each file. For example, if we had split the current experiment into both left and right tapping AND fast or slow tapping, we would want four files, one to represent each of the possible factor combinations (fast-left, slow-left, fast-right, slow-right). In this case, you might use F and S for the fast-slow distinction, thus naming your files FL, SL, FR, and SR. Ultimately, you will want to use the convention that is right for you and your study.

You will now need to repeat the above process for the right-hand tapping events for subject 1; an incomplete file has been included in the example to help you get started. Save this file as "R.1D" to the stim folder you just created under s1. Once you have completed subject 1's timing files, the same process needs to be done for subjects 2 and 3. For each, you will need to create a new stim folder and save their specific L and R files to that folder. Make sure that you don't accidentally save a file to a different subject's stim folder!

As a final note here, your timing files are automatically alphabetized as you save them; this will be important when indicating to AFNI what each of your files represents (more on that later):



Location: /home/cgodwin9/Desktop/afni_processing/1/stim

Name	Size	Type
L.1D	55 bytes	plain text document
R.1D	52 bytes	plain text document

Step 3 – Data Preprocessing and 1st-Level Analysis

Now that your data are converted into .nii format and you have created stimulus timing files, you are now ready to move on to actually processing fMRI data. There are two components to processing fMRI data at the individual level. The first is preprocessing, where you prepare the data for statistical analysis. Preprocessing typically consists of steps such as aligning the functional and structural scans, normalizing to a standard template, blurring, etc. This manual provides the standard preprocessing steps as the default in the scripts provided. However, as you advance in fMRI analysis, you may want to make the decision to add or remove preprocessing steps depending on what your data require. In the processing script here, your preprocessing steps are indicated under the "blocksDesired" input (see Step 3a).

The next step in individual-level fMRI analysis is the GLM, often referred to as the first-level analysis. This is where we run a general linear model based on our task conditions in order to discern how well our hypothesized model explains the collected data. You will indicate the task conditions under "regressorNames" and specify the contrasts you are interested in under "contrasts" (see Step 3a). In this manual, we are specifying two contrasts: 1) Examining just left-hand tapping activity (which gets compared to baseline), and 2) Comparing left-hand tapping activity relative to right-hand tapping activity.

Step 3a – Editing the processing file (processing_shell.sh)

The script is set up so that the user only needs to put in minimal study-specific information that AFNI will need for the processing. These are each entered as strings, which are indicated by the single quotes (').

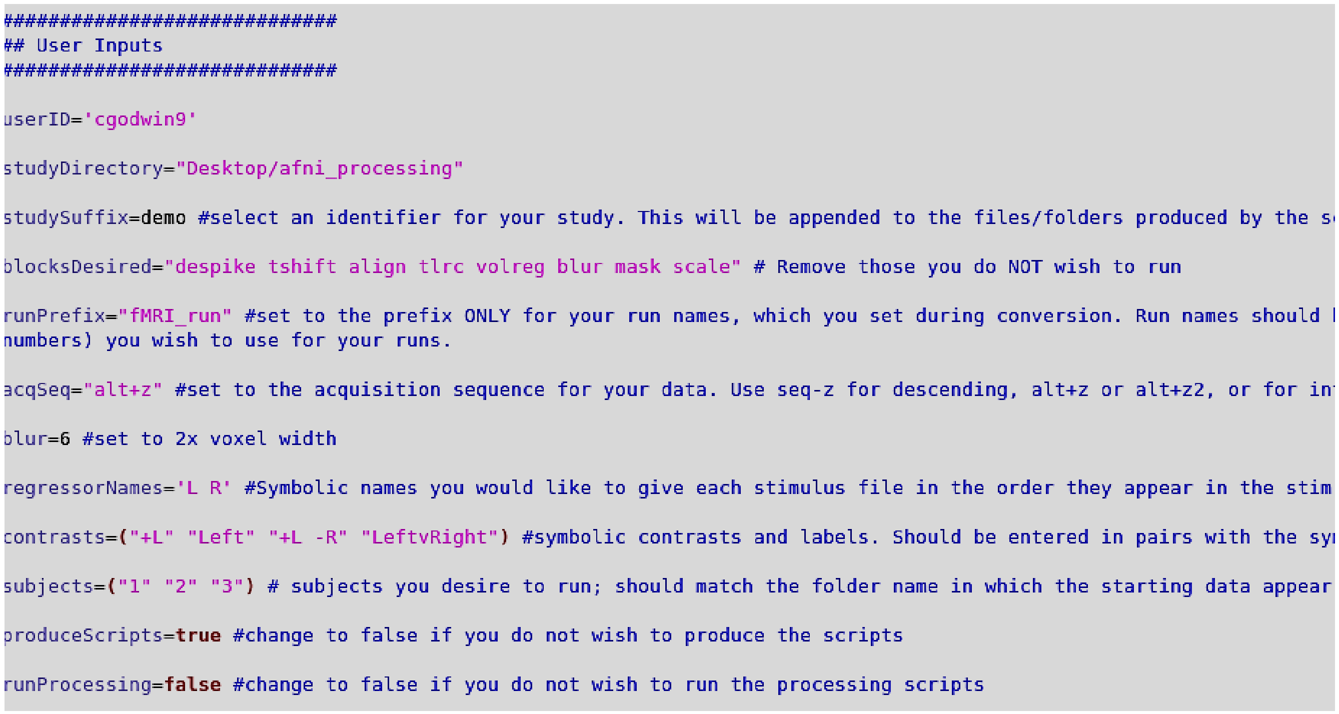
This processing script does two things: it produces the tcsh scripts you'll need to actually process the individual data, and then it runs those scripts on the individual data. You can run both of these steps or either, depending on your goals. You indicate which steps to run by setting produceScripts and runProcessing to either true or false (see below). When you are processing your data for the first time, I recommend just producing the tcsh scripts first by setting produceScripts to 'true' and runProcessing to 'false'. Then you can open the scripts and review them to make sure everything looks accurate (See Step 3b). After confirming your scripts are good, you can confidently run your processing by setting produceScripts to 'false' (because you already have them) and setting runProcessing to 'true'.

- **userID**: your user ID on the CABI system
- **studyDirectory**: The folder that was created for the current analysis
- **studySuffix**: an identifier for your study. This will be appended to the files/folders produced by the script.
- **blocksDesired**: a variable indicating the specific processing steps you want to run. Remove those you do NOT wish to run. The default is standard processing.
- **runPrefix**: a string indicating the prefix for your runs, that is, the prefix of the file names for your runs found in the folders /raw_nii
- **acqSeq**: the acquisition sequence that was used to collect the data. allows for seq-z, alt+z, and alt+z2 options. Use seq-z for descending, alt+z for interleaved with an odd number of slices, and alt+z2 for interleaved with an even number of slices. If you do not know the acquisition sequence, check your scan protocol (can be obtained from the MRI technician). In this manual, the acquisition sequence was interleaved with an odd number of slices. See Text Box 1 for more information on acquisition sequences.
- **blur**: size of the kernel used for smoothing. Default value is 2x your voxel width; most studies will use 6.
- **regressorNames**: Symbolic names you would like to give each stimulus file in alphabetical order (the order they appear in the stim folder). Minimal characters are

recommended.

- **contrasts**: symbolic contrasts and labels. Should be entered in pairs with the symbolic expression first; use the included example for a formatting guide. Symbols used in the contrast expressions must match those included in the regressor names. The two contrasts conducted here are 1) Examining just left-hand tapping activity (which gets compared to baseline), and 2) Comparing left-hand tapping activity relative to right-hand tapping activity.
- **subjects**: subjects you desire to run; should match the folder name in which the starting data appear

The produceScripts and runProcessing variables can be used to indicate whether you want to skip one or the other step when running the processing; use these if you have already made the tcsh processing scripts (produceScripts=false) or if you just want to produce the scripts and check them before running the processing proper (runProcessing=false).



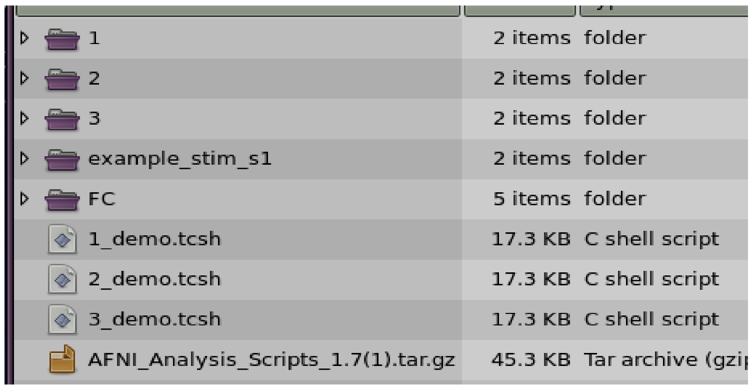
```
##############################
## User Inputs
##############################

userID='cgodwin9'

studyDirectory="Desktop/afni_processing"

studySuffix=demo #select an identifier for your study. This will be appended to the files/folders produced by the s

blocksDesired="despike tshift align tlrc volreg blur mask scale" # Remove those you do NOT wish to run

runPrefix="fMRI_run" #set to the prefix ONLY for your run names, which you set during conversion. Run names should
numbers) you wish to use for your runs.

acqSeq="alt+z" #set to the acquisition sequence for your data. Use seq-z for descending, alt+z or alt+z2, or for in

blur=6 #set to 2x voxel width

regressorNames='L R' #Symbolic names you would like to give each stimulus file in the order they appear in the stim

contrasts=("+L" "Left" "+L -R" "LeftvRight") #symbolic contrasts and labels. Should be entered in pairs with the sy

subjects=("1" "2" "3") # subjects you desire to run; should match the folder name in which the starting data appear

produceScripts=true #change to false if you do not wish to produce the scripts

runProcessing=false #change to false if you do not wish to run the processing scripts
```

Step 3b – Generate and review tcsh scripts

In this manual, I set runProcessing to false, so in this step I'm only producing the tcsh scripts.

Once you have completed the steps above, go into Terminal and navigate to your study folder. From here, run the processing script you renamed and edited for your specific study using the "sh" command (the process will take a few minutes).

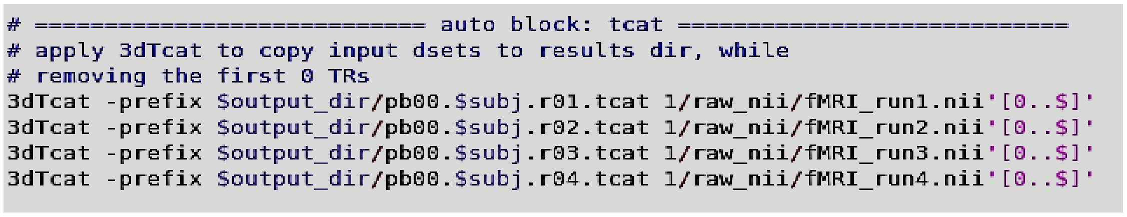
You should see some output in the terminal indicating that the tcsh scripts are being produced. Upon completion, you will see the tcsh scripts appear in your processing folder. Here, my scripts appear as 1_demo.tcsh, 2_demo.tcsh etc because 'demo' is what I specified for the studySuffix.



1	2 items	folder
2	2 items	folder
3	2 items	folder
example_stim_s1	2 items	folder
FC	5 items	folder
1_demo.tcsh	17.3 KB	C shell script
2_demo.tcsh	17.3 KB	C shell script
3_demo.tcsh	17.3 KB	C shell script
AFNI_Analysis_Scripts_1.7(1).tar.gz	45.3 KB	Tar archive (gzi

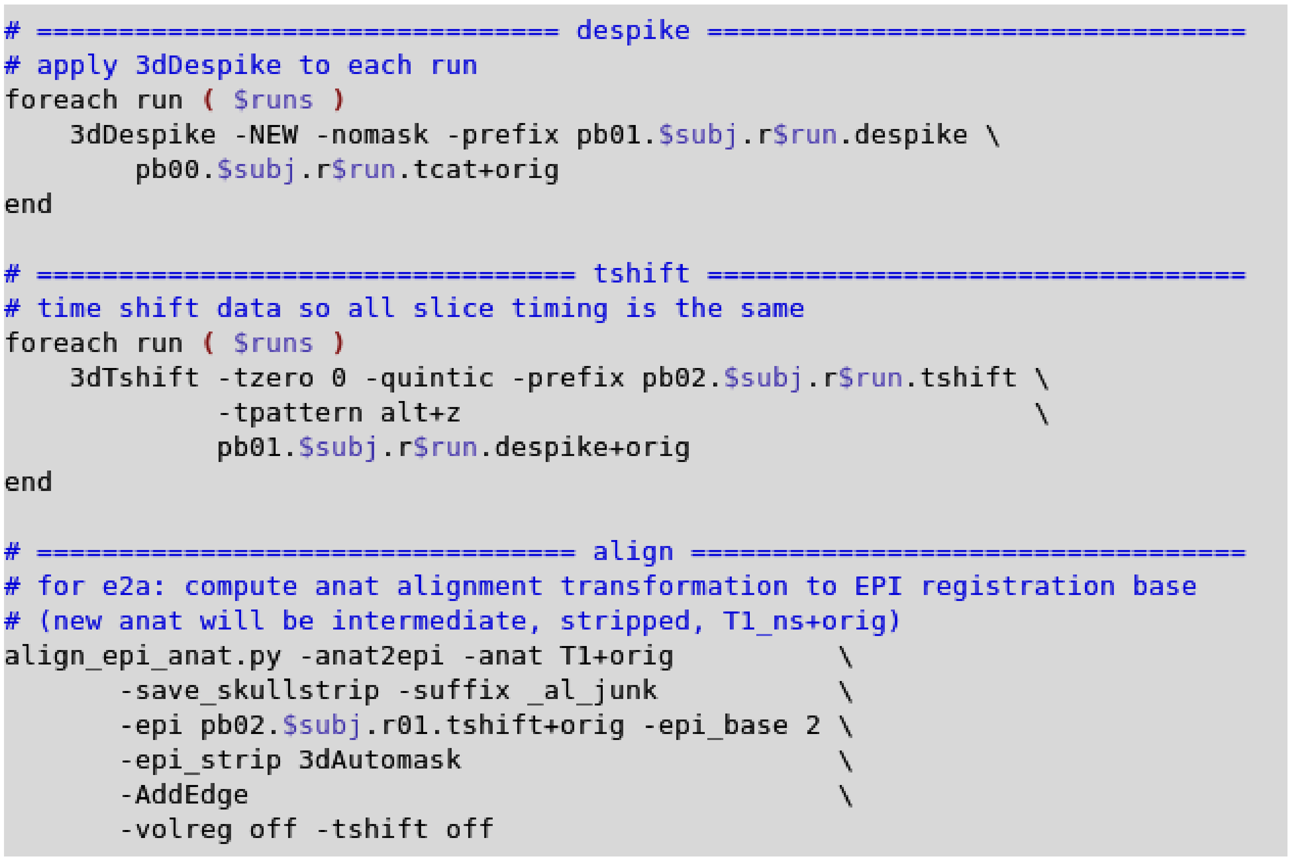
At this point, it's a good idea to open one of your tcsh scripts to review the contents. These are the scripts that will actually process your data, so it's important they are accurate. If you are new to fMRI processing, you may not understand every part of these scripts. However, there are a few things you can take note of:

1) confirm that all functional runs are indicated under auto block:tcat



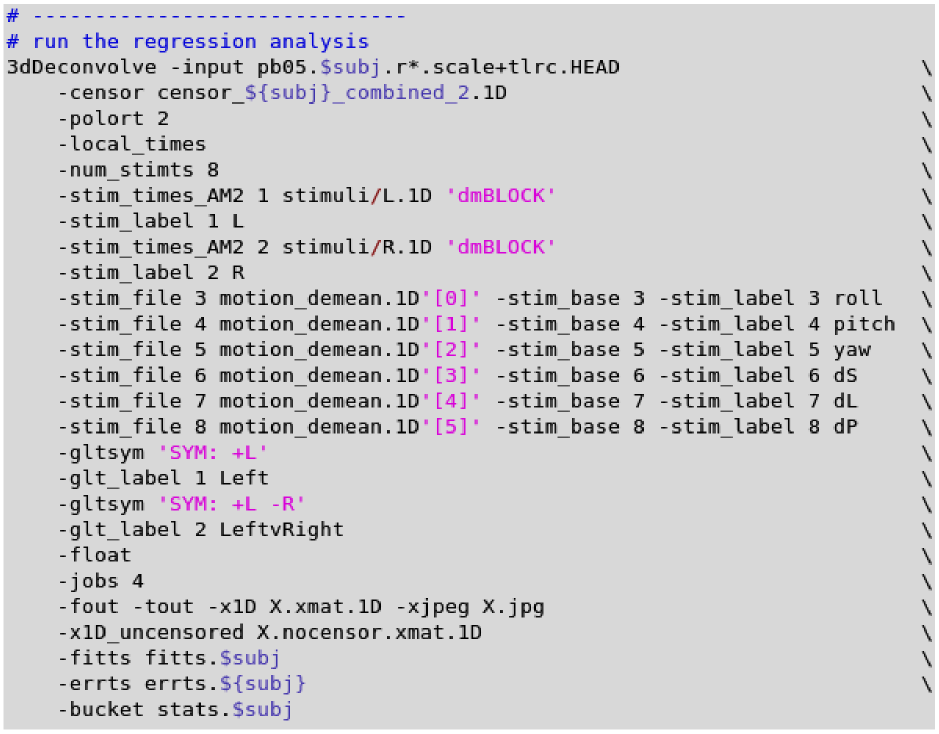
```
# ============================ auto block: tcat ============================
# apply 3dTcat to copy input dsets to results dir, while
# removing the first 0 TRs
3dTcat -prefix $output_dir/pb00.$subj.r01.tcat 1/raw_nii/fMRI_run1.nii'[0..$]'
3dTcat -prefix $output_dir/pb00.$subj.r02.tcat 1/raw_nii/fMRI_run2.nii'[0..$]'
3dTcat -prefix $output_dir/pb00.$subj.r03.tcat 1/raw_nii/fMRI_run3.nii'[0..$]'
3dTcat -prefix $output_dir/pb00.$subj.r04.tcat 1/raw_nii/fMRI_run4.nii'[0..$]'
```

2) check for the preprocessing blocks you specified in the processing_shell.sh (e.g., despike, tshift, align, etc.)



```
# ================================ despike =================================
# apply 3dDespike to each run
foreach run ( $runs )
    3dDespike -NEW -nomask -prefix pb01.$subj.r$run.despike \
        pb00.$subj.r$run.tcat+orig
end

# ================================= tshift =================================
# time shift data so all slice timing is the same
foreach run ( $runs )
    3dTshift -tzero 0 -quintic -prefix pb02.$subj.r$run.tshift \
             -tpattern alt+z                                   \
             pb01.$subj.r$run.despike+orig
end

# ================================= align ==================================
# for e2a: compute anat alignment transformation to EPI registration base
# (new anat will be intermediate, stripped, T1_ns+orig)
align_epi_anat.py -anat2epi -anat T1+orig          \
       -save_skullstrip -suffix _al_junk           \
       -epi pb02.$subj.r01.tshift+orig -epi_base 2 \
       -epi_strip 3dAutomask                       \
       -AddEdge                                    \
       -volreg off -tshift off
```

3) review the GLM: your stim_times and stim_labels should refer to your task conditions (left and right tapping), there should be 6 motion regressors (e.g., motion_demean.1D), and your contrasts should be specified under gltsym



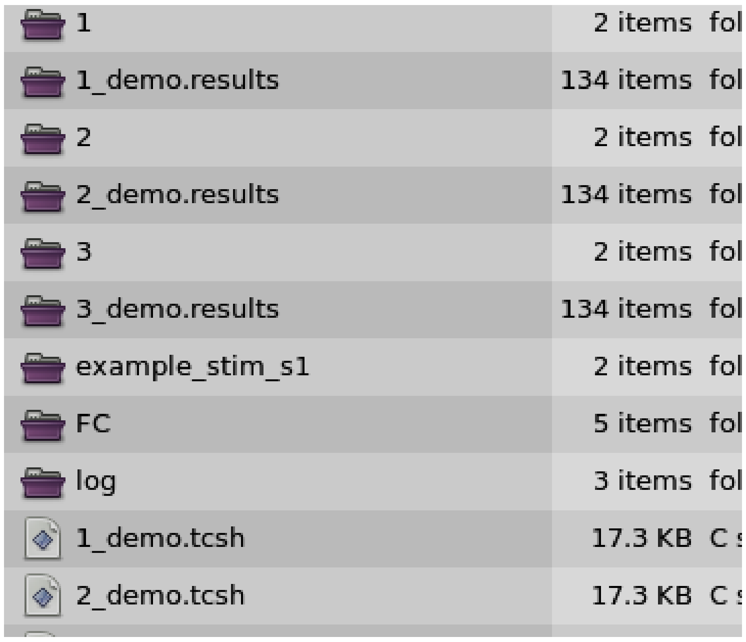
```
# ------------------------------
# run the regression analysis
3dDeconvolve -input pb05.$subj.r*.scale+tlrc.HEAD                   \
    -censor censor_${subj}_combined_2.1D                              \
    -polort 2                                                         \
    -local_times                                                      \
    -num_stimts 8                                                     \
    -stim_times_AM2 1 stimuli/L.1D 'dmBLOCK'                          \
    -stim_label 1 L                                                   \
    -stim_times_AM2 2 stimuli/R.1D 'dmBLOCK'                          \
    -stim_label 2 R                                                   \
    -stim_file 3 motion_demean.1D'[0]' -stim_base 3 -stim_label 3 roll  \
    -stim_file 4 motion_demean.1D'[1]' -stim_base 4 -stim_label 4 pitch \
    -stim_file 5 motion_demean.1D'[2]' -stim_base 5 -stim_label 5 yaw   \
    -stim_file 6 motion_demean.1D'[3]' -stim_base 6 -stim_label 6 dS    \
    -stim_file 7 motion_demean.1D'[4]' -stim_base 7 -stim_label 7 dL    \
    -stim_file 8 motion_demean.1D'[5]' -stim_base 8 -stim_label 8 dP    \
    -gltsym 'SYM: +L'                                                 \
    -glt_label 1 Left                                                 \
    -gltsym 'SYM: +L -R'                                              \
    -glt_label 2 LeftvRight                                           \
    -float                                                            \
    -jobs 4                                                           \
    -fout -tout -x1D X.xmat.1D -xjpeg X.jpg                           \
    -x1D_uncensored X.nocensor.xmat.1D                                \
    -fitts fitts.$subj                                                \
    -errts errts.${subj}                                              \
    -bucket stats.$subj
```

Step 3c – Run tcsh scripts

Once you have generated and reviewed the tcsh scripts, edit processing_shell.sh to set produceScripts = false and runProcessing = true.

Again, go into Terminal and navigate to your study folder. From here, run the processing script. The process can take a while; processing time depends on the number of subjects and size of run data per subject to be processed. Terminal output simply states that the jobs have been submitted. To check whether your processing is running, you can type "qstat" in terminal to see which jobs are still in the system, or refresh your experiment folder to see the folders/files being added and changing size. Processing is done when the qstat command does not produce an output in terminal.

Once processing is complete, you will see "results" folders for each of your subjects



1	2 items	fol
1_demo.results	134 items	fol
2	2 items	fol
2_demo.results	134 items	fol
3	2 items	fol
3_demo.results	134 items	fol
example_stim_s1	2 items	fol
FC	5 items	fol
log	3 items	fol
1_demo.tcsh	17.3 KB	C
2_demo.tcsh	17.3 KB	C

Inside each folder are many files. Each step of preprocessing is saved separately, with the stats file being the processed functional data file that you'll be interested in. The final anatomical data is saved as “anat_final”. Other files that will be of interest in the upcoming sections are the @ss_review_driver, which helps with quality assurance, and the X.jpg, which is the design matrix and should always be reviewed to ensure correct model specification.

Text Box 1: About Acquisition Sequences

When collecting scans in the MRI, it is convenient to talk about the TRs, or volumes, we collect as if they are acquired simultaneously. However, this is not the case. The slices of the volume must be collected one (or a few, in the case of multiband) at a time over the course of the TR. This, of course, means that the first slice and the last slice in a single volume can be separated in time by .5-3 seconds, depending on your TR. In fact, the last slice in your volume is actually closer in time to the next volume you collect!

To account for these time differences, many researchers apply slice timing correction procedures to their data. This involves interpolating between slices to estimate what a volume would look like if it had been collected as a whole instantaneously. The CABI processing stream likewise uses this slice timing correction. However, in order to apply the slice timing procedure correctly, we need to tell the program how your scans were collected.

There are a number of ways in which the MRI can collect a volume; that is, the order in which it collects each slice. The precise pattern in which the slices of a volume are collected is called the “acquisition sequence”. While ostensibly the MRI could collect these slices in any order, there are a select few patterns that are recommended for scanning.

Descending Acquisition with Gap
A descending acquisition pattern with slice gap is recommended for most protocols. Descending means that the scanner collects the scans in a single direction from the top of the head to the bottom. The gap is the amount of space to add between slices, which prevents residual excitation from biasing subsequent slices. The gap is specified in a percentage of slice width, and is typically 20% (although as little as 10-15% has been used, it is not recommended by CABI). In the processing stream, to indicate that your slice timing correction should use a descending acquisition, use the option “seq-z”. You do not need to specify the gap size.

Interleaved Acquisition

Interleaved acquisition collects scans starting from one side of the brain, collecting every other slice (i.e., all odds or all evens depending on parameters) in one direction, then returning to the starting point and collecting the opposite slices (i.e., the evens and odds that remained from the first pass). Collecting scans in this way still prevents excitation from carrying over into subsequent slices, as skipping every other slice width accomplishes the same idea as the gap in descending acquisition. However, because the pattern ultimately leaves no space in between slices, this pattern allows for increased spatial coverage. However, there are biophysical reasons why interleaved patterns may increase artifacts in your data. CABI recommends that this pattern be used only if spatial coverage is a particular factor for your experiment.

To indicate that your slice timing correction should use an interleaved acquisition, you will need to know both the direction of collection and the number of slices in your volume. If your volumes are collected in the ascending direction and you have an odd number of slices, use the option “alt+z”. If you have an even number of slices, use “alt+z2”. Likewise, for the descending direction with odd slices, use “alt-z”, and for even slices “alt-z2”.

Multiband Sequence

CABI has the capability to collect multiband sequences. This sequence allows for the collection of multiple slices simultaneously, which can afford increased temporal resolution while maintaining spatial resolution. Multiband sequencing is a relatively new technology, and there are not well-established methods of processing data collected in this way. Researchers interested in this sequence should work with CABI staff to determine if multiband sequencing is appropriate for their experiments and to understand the options for processing these data. Multiband sequence data cannot be processed using the CABI processing stream at this time.

Quality Assurance (QA)

Note: the following information is intended to apply specifically to data analyzed under the CABI processing stream; however, in most cases, it will apply to any afni_proc-based analysis. This QA procedure should be run on the CABI computer cluster for best results.

Step 1 – AFNI Guided Review (review_driver)

The majority of your quality assurance will be conducted with the guidance of AFNI's review driver, which leads you step by step through the different aspects of quality control. The review driver provides a cursory level of information on how to interpret the results of each step, and this guide is intended to provide more thorough information and guidance for using the program effectively.

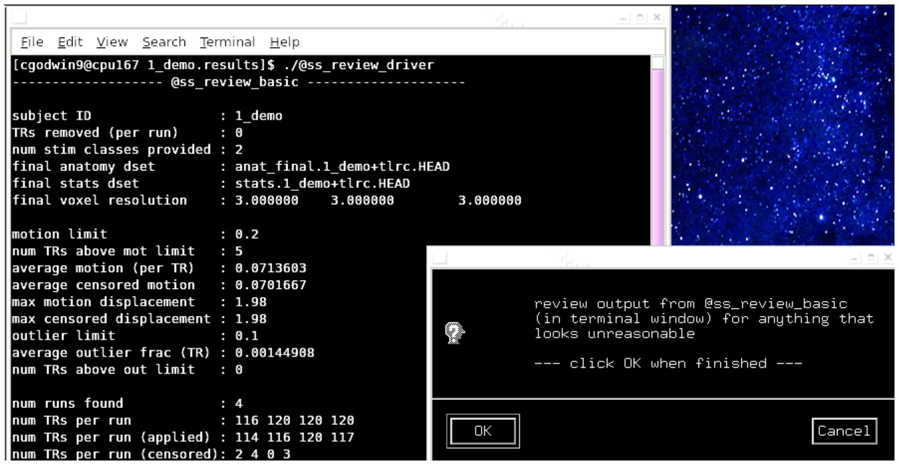
Step 1a – Starting the program

To run the quality check on one of your participants, enter terminal and navigate to that person's results folder. Run the summary file by typing "csh @ss_review_driver" (no quotes) in the terminal window and pressing "enter". The program will then produce a set of summary notes, plots, and images. The program will pause briefly after printing to the terminal, then continue, so just be patient while everything loads.

Step 1b – TR/Motion censoring

First, the program will print a summary of TR/motion censoring to the terminal. This information tells you how many TR's were excluded from the analysis. Motion censoring occurs when the estimated motion parameters indicate a change in position that passes a threshold displacement. Other types of censoring look for unexpected magnitude differences or artifacts in the BOLD signal. The TR censoring values in the summary represent the total censoring, including motion, so the difference in TR and motion sensor in values represents these other censoring results. When reviewing this information, start by looking at the total censored time points; they should be minimized for both motion and overall censoring. Typical "good" values may lie between zero and 20, although the exact range of acceptable values will vary with the number of time points and the population of your study. You may find that, across all of your participants, the censoring values follow a modal distribution, where most participants fall within a "good" range, and a minority show much larger values. You might use this to determine your censoring cut off.

Next, look at the "by run" and "by condition" breakdowns. These values show the number of time points censored for each run or condition, respectively. These values allow you to assess if there may be time- or task-related artifacts in your design. Typically (and for the most part with minimal consequence to standard analyses), you will see a slight increase in motion censoring across the runs, as participants get more restless over time. If you find that one of your conditions incited more censoring then the others in multiple participants, consider how your design may result in this confound and ways to control for it and future participants.

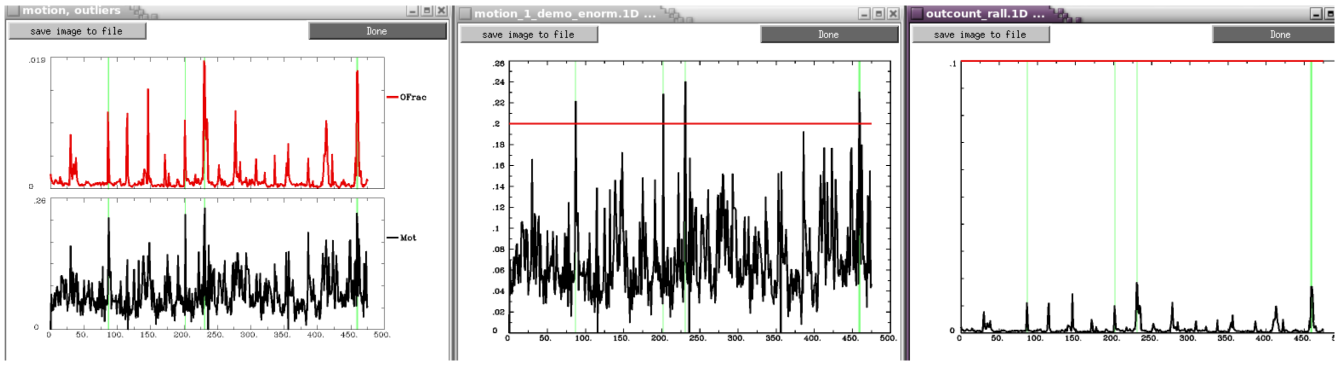


```
[cgodwin9@cpu167 1_demo.results]$ ./@ss_review_driver
------------------ @ss_review_basic -------------------

subject ID                : 1_demo
TRs removed (per run)     : 0
num stim classes provided : 2
final anatomy dset        : anat_final.1_demo+tlrc.HEAD
final stats dset          : stats.1_demo+tlrc.HEAD
final voxel resolution    : 3.000000    3.000000        3.000000

motion limit              : 0.2
num TRs above mot limit   : 5
average motion (per TR)   : 0.0713603
average censored motion   : 0.0701667
max motion displacement   : 1.98
max censored displacement : 1.98
outlier limit             : 0.1
average outlier frac (TR) : 0.00144908
num TRs above out limit   : 0

num runs found            : 4
num TRs per run           : 116 120 120 120
num TRs per run (applied) : 114 116 120 117
num TRs per run (censored): 2 4 0 3
```

```
review output from @ss_review_basic
(in terminal window) for anything that
looks unreasonable

--- click OK when finished ---
```

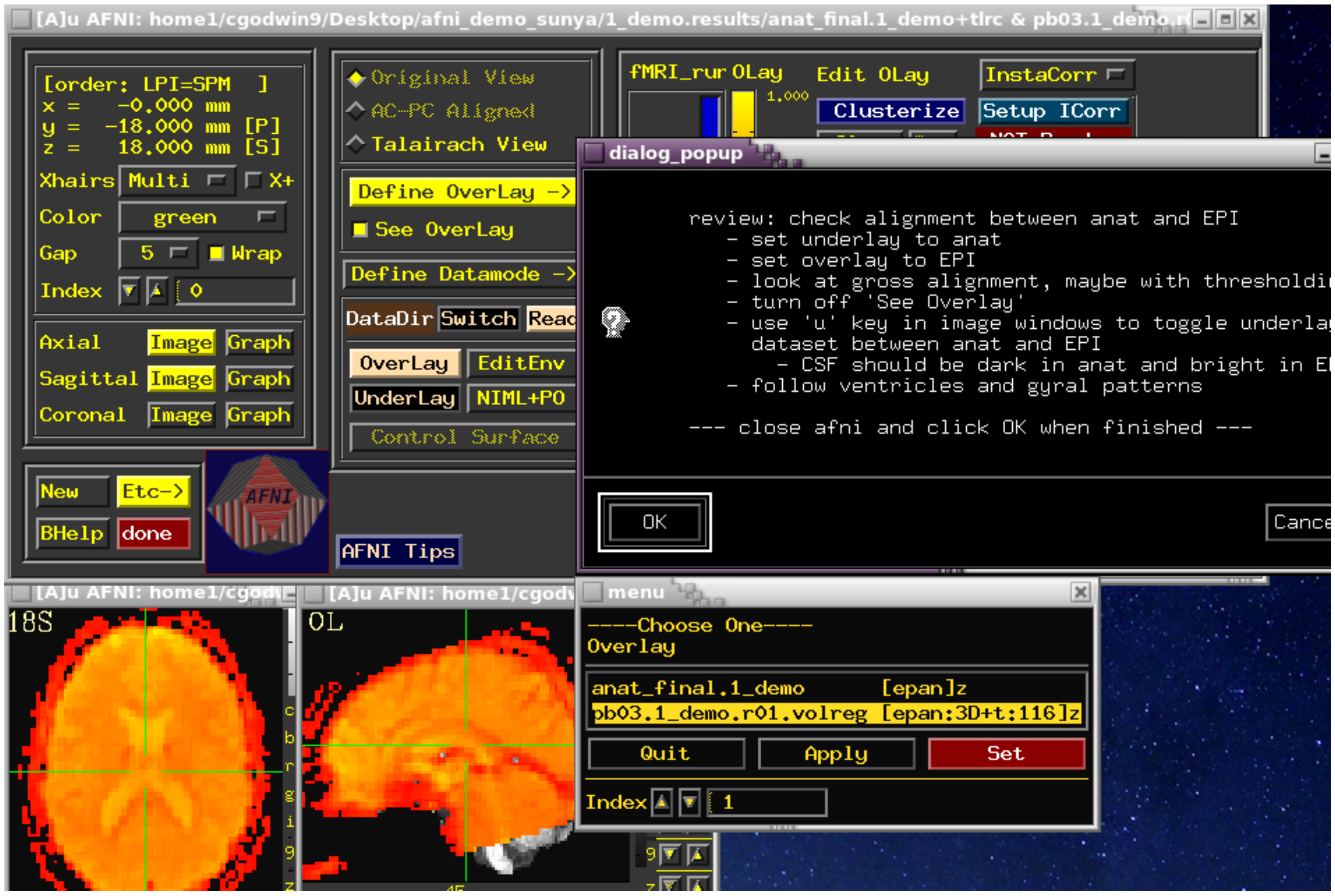
Step 1c – Motion and outlier plots

After clicking ‘ok’ in the dialog pop-up, the program will produce three plots. One is titled “motion, outliers”. This is the summary graph of your censoring data, where the green bars represent time points that have been censored. The top graph represents your outliers, and the bottom represents motion. The other two graphs, “outcount_rall” and “motion….enorm” show the thresholding limits and results for the outlier and motion data, respectively. These allow you to visually inspect the same information discussed in Step 1b above to confirm your previous conclusions. Look for groupings of censored TRs and for overlap between the outlier and motion censored time points. Large groupings may suggest the removal of a particular run, or may indicate a condition-driven confound. More overlap in the outlier and motion censored TRs is quite common and means less overall data loss versus uncorrelated outlier/motion censoring. Close each plot when finished by pressing “Done”.



Step 1d – EPI/Anatomical registration

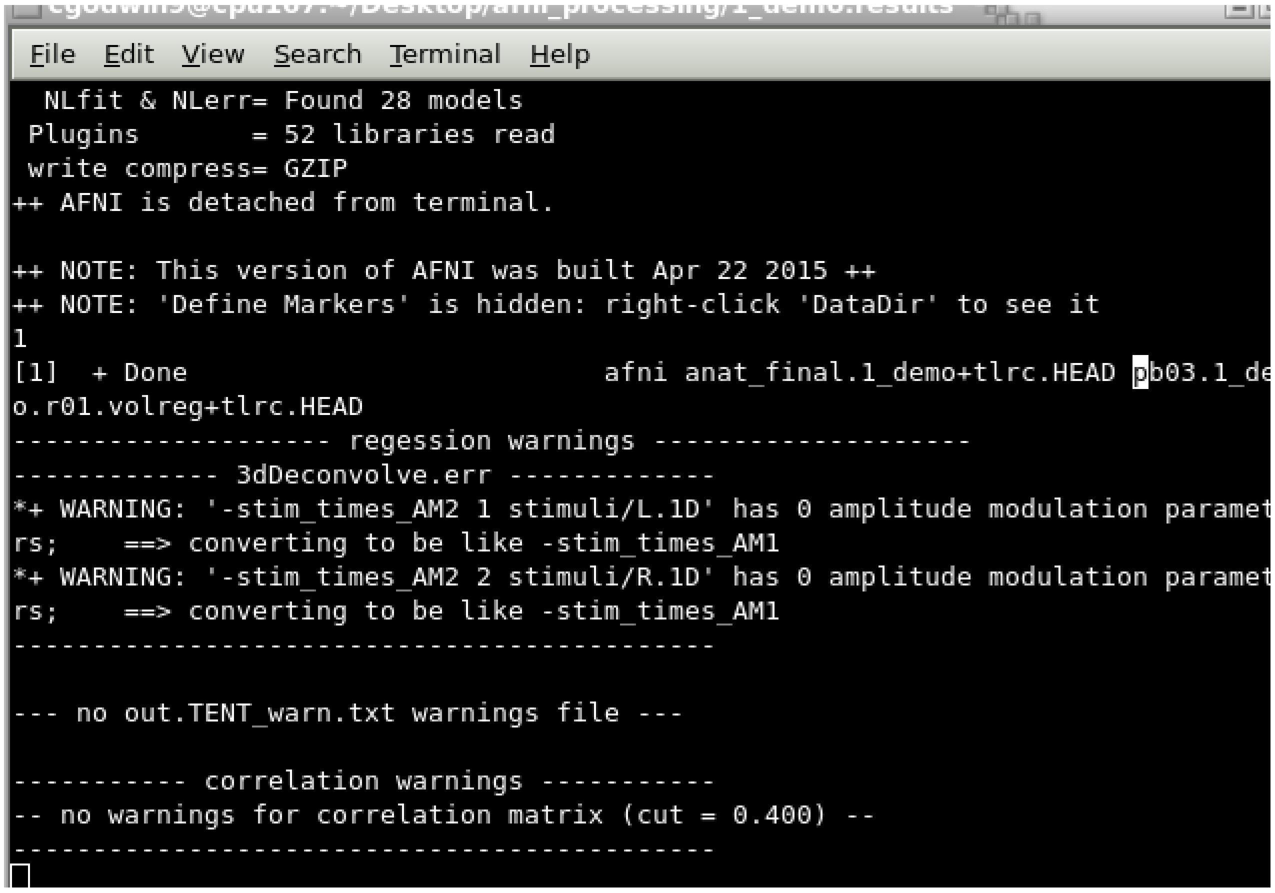
The next summary is a guided visual confirmation in the AFNI GUI to confirm that your anatomical registration is correctly aligned. The instructions window for this summary is particularly informative; therefore, you should follow the instructions in this window. You should overlay the functional EPI scan (the volreg file) on top of the anatomical image for your participant. You can toggle the functional scan on and off. You should use the natural edges present in the images (e.g., cortical edges, ventricles, mid sagittal line) to check that the EPI is properly registered to the anatomical scan. When you have finished, press "Done" in the GUI window to close AFNI and "OK" in the instruction screen to continue to the next summary.



```
review: check alignment between anat and EPI
   - set underlay to anat
   - set overlay to EPI
   - look at gross alignment, maybe with thresholdi
   - turn off 'See Overlay'
   - use 'u' key in image windows to toggle underla
     dataset between anat and EPI
       - CSF should be dark in anat and bright in E
   - follow ventricles and gyral patterns

--- close afni and click OK when finished ---
```

Step 1e - Warnings

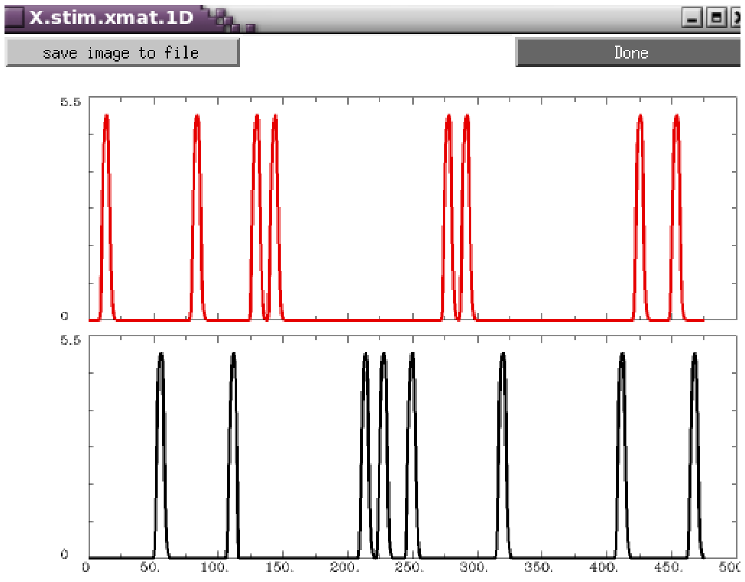
Next, the program will print out a list of the warning messages to the terminal window that were generated during the analyses. You should review these warnings to determine if any of the specifications in your script resulted in any unexpected behavior of the analysis. Not every warning automatically indicates an error; in the example here, anytime your regressor is the same duration across events or is a binary (on dash off) rather than a parametric modulation, it will warn you that the analysis will treat the regressor as a simpler event type in order to reduce the computational overhead of the analysis. When you are finished reviewing the warning messages, press OK.



```
  NLfit & NLerr= Found 28 models
 Plugins       = 52 libraries read
 write compress= GZIP
++ AFNI is detached from terminal.

++ NOTE: This version of AFNI was built Apr 22 2015 ++
++ NOTE: 'Define Markers' is hidden: right-click 'DataDir' to see it
1
[1]  + Done                          afni anat_final.1_demo+tlrc.HEAD pb03.1_de
o.r01.volreg+tlrc.HEAD
-------------------- regession warnings --------------------
------------- 3dDeconvolve.err -------------
*+ WARNING: '-stim_times_AM2 1 stimuli/L.1D' has 0 amplitude modulation paramet
rs;    ==> converting to be like -stim_times_AM1
*+ WARNING: '-stim_times_AM2 2 stimuli/R.1D' has 0 amplitude modulation paramet
rs;    ==> converting to be like -stim_times_AM1
--------------------------------------------

--- no out.TENT_warn.txt warnings file ---

----------- correlation warnings -----------
-- no warnings for correlation matrix (cut = 0.400) --
--------------------------------------------
```

Step 1f – X.stim and sum plots

This section shows you the visual representation of your stimulus regressors after they have been convolved with the hemodynamic response function to be incorporated into your full design matrix. Here we are focused on the X.stim plot. Each row represents a regressor in the order in which your stimulus timing files were included in the afni_proc command call. The horizontal axis represents time. Make sure that these timings align with your expected stimulus presentation times and orders according to your timing files. Press “Done” on each plot to close it and press “OK” in the instruction screen to continue.



Step 1h – Stats file

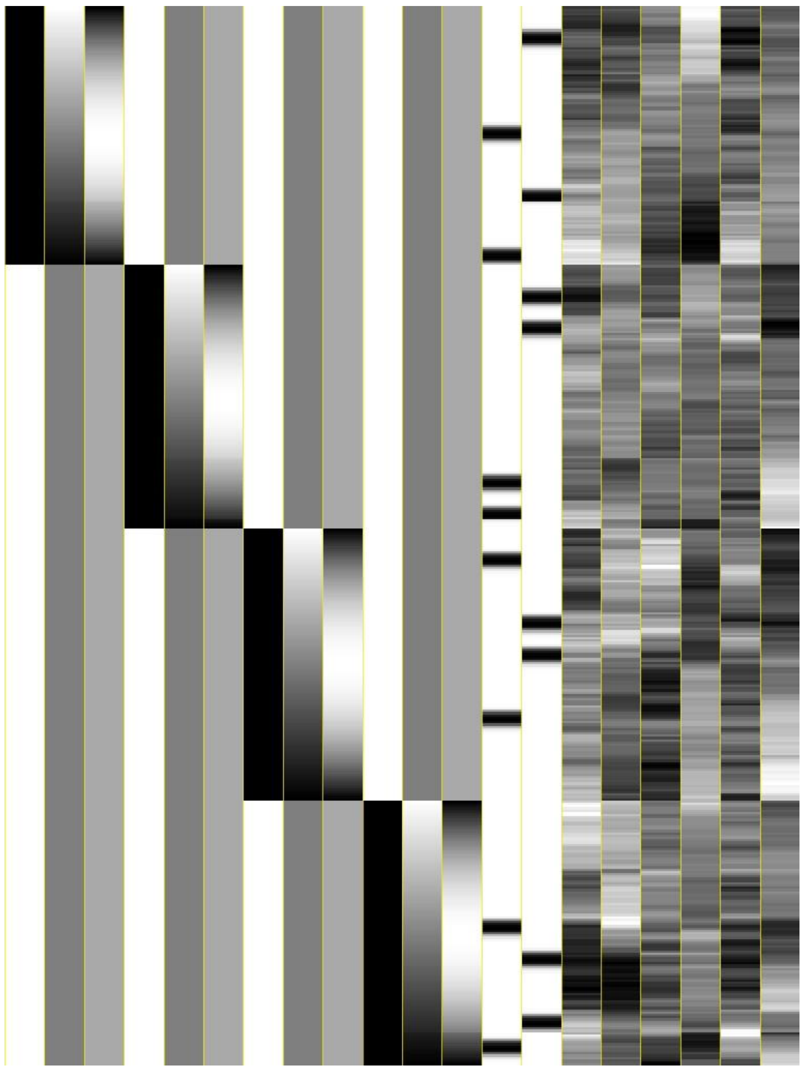
Finally, the program will open the statistics bin file in AFNI for you to look at the results of the individual-level analysis. It will open the file using the Full F-Statistic; however, at this point, you can use the AFNI viewer as you normally would (see AFNI Viewer Tutorial section). Be sure to press OK to close the review program when finished.

Step 2 – Design Matrix

This review step is not included in the AFNI-produced QA procedure, but is nonetheless a worthwhile step. In your subject's results folder, open the image file "X.jpg". This image shows a visual representation of the design matrix used in your GLM analysis. In this image, time progresses down the vertical axis, and each column represents a regressor. White coloring indicates a value of zero (or off); black indicates a value of one (or on); and grays represent the intervening values (parametric regressors only).

In most cases, the column order is as follows. First are the polynomial regressors, which remove drift and other systematic noise. The standard CABI stream uses 0°, 1°, and 2°

polynomial regressors. Polynomial regressors are defined separately for each run, which results in a pattern of white columns with repeating segments along the diagonal. This pattern should indicate to you where your scans stop and start for separate runs. Following these columns are your task condition regressors; there should be one column per regressor. The color blocks in your regressor column should align with the onsets in duration (and amplitudes, if applicable) in your stimulus timing file. Finally, the last columns represent the motion regressors produced in preprocessing. The CABI stream uses the typical six-parameter motion correction procedure; accordingly, there will be six regressor columns.



Group Level Analysis

Once you have completed the first-level analysis and have reviewed each subject’s results for quality assurance, you can move on to the group level analysis. This is the part of fMRI analysis that you are probably most interested in – it will tell you whether the contrasts that you are interested in are statistically significant, that is, does one task condition show significantly greater (or less) activation than the other condition. Group analyses in fMRI can be very complicated and intricate, as there are many ways to analyze data at this level. For introductory purposes, this guide will walk you through how to run standard t-tests on your data to test whether your contrasts of interest differ from each other (or from zero).

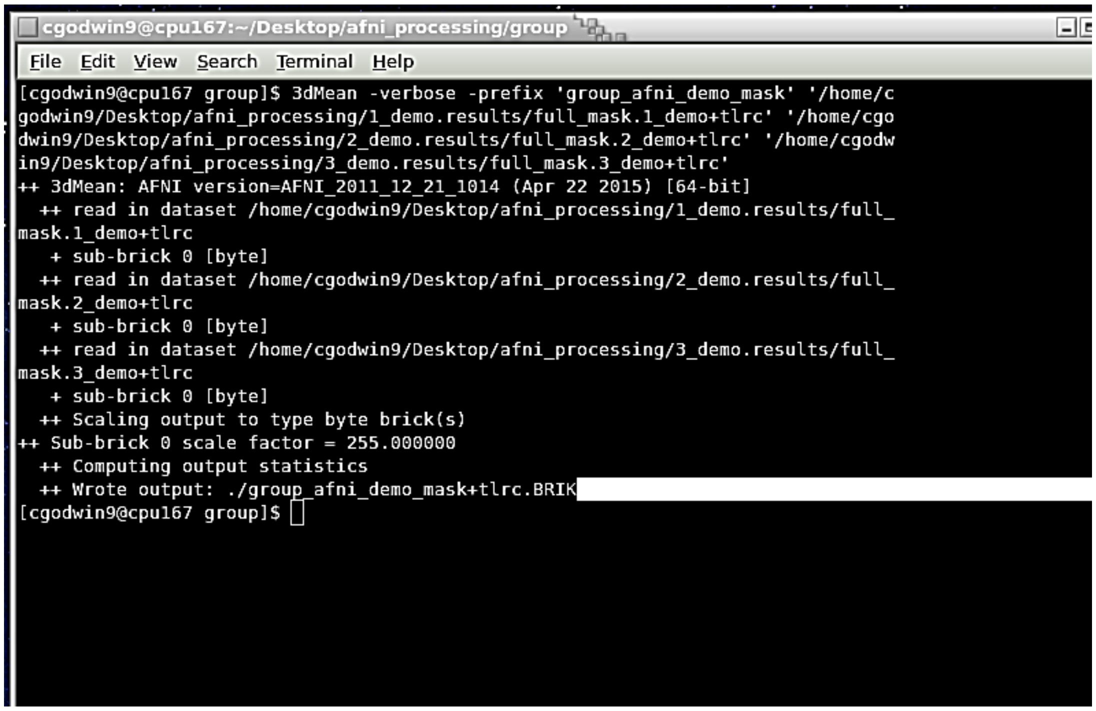
Step 1 – Masking

Step 1a – Average your subject masks

First you will average across all your subjects' functional data to create a group whole brain mask. This will allow you to constrain your analyses to the brain area consistent across all subjects. To do this, you will use the AFNI function 3dMean in the terminal, which allows you to average across datasets. 3dMean will give you a new datafile that contains, for each voxel, the percent of subjects for which activity in that voxel is present. If all subjects have activity in a particular voxel, the value that 3dMean calculates is 1 (100%). If only 1 subject had signal from a particular voxel, then the 3dMean calculation would produce 1/(total # of subjects); in this example, that would be .33 (33%).

You will average across the *full masks* that are produced for each subject; you can find the full mask file for each subject in the subject's results folder (e.g., full_mask.1_demo+tlrc). Note that these full mask files only contain 1's and 0's – where a voxel has activity, there is a 1, where there is no activity, there is a 0. When running 3dMean, you will want to specify "verbose" so that AFNI prints out additional useful information regarding the calculation. In addition, you will need to specify under the "prefix" option the name you want for your whole brain mask file. Here, I am naming the file 'group_afni_demo_mask'. After you specify the prefix, you will then list all the full masks from the subjects that you want to use to create the group whole brain mask. Make sure to include the full path for these file names so that you can run the command from your group folder.

An example of the syntax to use can be found in your processing folder in the file "group_analysis_readme.txt". Once you have completed this syntax, it is easiest to just copy and paste this syntax into your terminal (make sure to first navigate to your group folder). This is the output you should see in the terminal after this whole brain mask is created:


cgodwin9@cpu167:~/Desktop/afni_processing/group
File Edit View Search Terminal Help

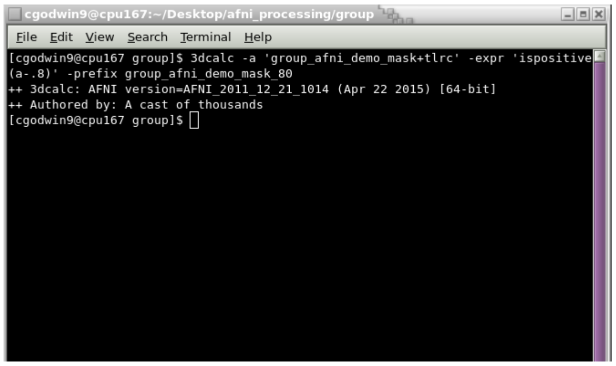
```
[cgodwin9@cpu167 group]$ 3dMean -verbose -prefix 'group_afni_demo_mask' '/home/c
godwin9/Desktop/afni_processing/1_demo.results/full_mask.1_demo+tlrc' '/home/cgo
dwin9/Desktop/afni_processing/2_demo.results/full_mask.2_demo+tlrc' '/home/cgodw
in9/Desktop/afni_processing/3_demo.results/full_mask.3_demo+tlrc'
++ 3dMean: AFNI version=AFNI_2011_12_21_1014 (Apr 22 2015) [64-bit]
  ++ read in dataset /home/cgodwin9/Desktop/afni_processing/1_demo.results/full_
mask.1_demo+tlrc
   + sub-brick 0 [byte]
  ++ read in dataset /home/cgodwin9/Desktop/afni_processing/2_demo.results/full_
mask.2_demo+tlrc
   + sub-brick 0 [byte]
  ++ read in dataset /home/cgodwin9/Desktop/afni_processing/3_demo.results/full_
mask.3_demo+tlrc
   + sub-brick 0 [byte]
  ++ Scaling output to type byte brick(s)
++ Sub-brick 0 scale factor = 255.000000
  ++ Computing output statistics
  ++ Wrote output: ./group_afni_demo_mask+tlrc.BRIK
[cgodwin9@cpu167 group]$
```

Step 1b – Constrain the mask

The next step is to edit the whole brain mask so that it is constrained to voxels/brain areas common across most (or all) subjects. To do this we use the AFNI function 3dcalc. You will need to decide how strict you want your mask to be – do you want your mask to only include voxels that belong to ALL subjects (100% overlap), or are you ok with a mask that has a more generous threshold? Typically a 100% overlap between subjects is too conservative. For example, if 19 out of 20 subjects had an overlap between a voxel, that voxel would still be excluded from the whole-brain mask because it is not common to all subjects. A good compromise is to require an 80% overlap, where voxels are only required to be common across 80% of your subjects in order to be included in the group mask.

The syntax for the 3dcalc function can be found in your group_analysis_readme file. As with other AFNI functions, you can also read more about 3dcalc in AFNI's online help section. In the 3dcalc syntax, you will want to set your current group mask for option 'a'. 3dcalc works by using the logical function 'ispositive' and setting it to be (a – 0.80). That will take every voxel from your group mask, subtract .80, and every voxel that is greater than .80 will survive and be saved as your constrained mask (note that the greatest value possible in your group mask is 1). Make sure to specify a name for your constrained mask under the prefix option. When you're ready, run the command in the terminal.

Output from running 3dcalc:



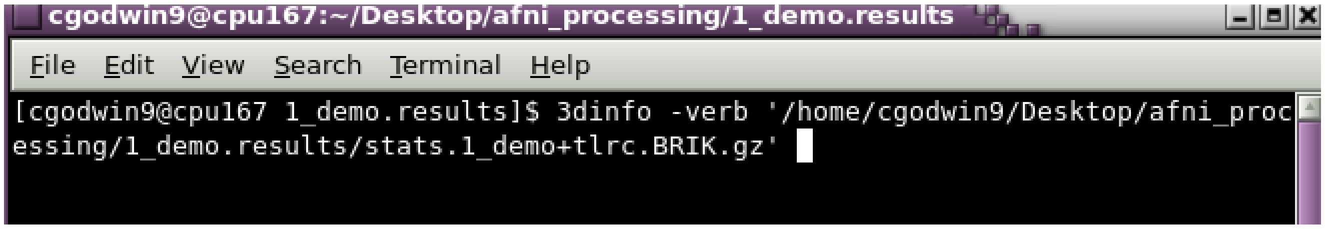
```
[cgodwin9@cpu167 group]$ 3dcalc -a 'group_afni_demo_mask+tlrc' -expr 'ispositive
(a-.8)' -prefix group_afni_demo_mask_80
++ 3dcalc: AFNI version=AFNI_2011_12_21_1014 (Apr 22 2015) [64-bit]
++ Authored by: A cast of thousands
[cgodwin9@cpu167 group]$ 
```

Step 2 – Running Statistics

Now you are ready to run your statistical tests. In this example, we specified two contrasts in the GLM when we completed the processing_shell.sh. Here we will test if those contrasts are significantly different from zero.

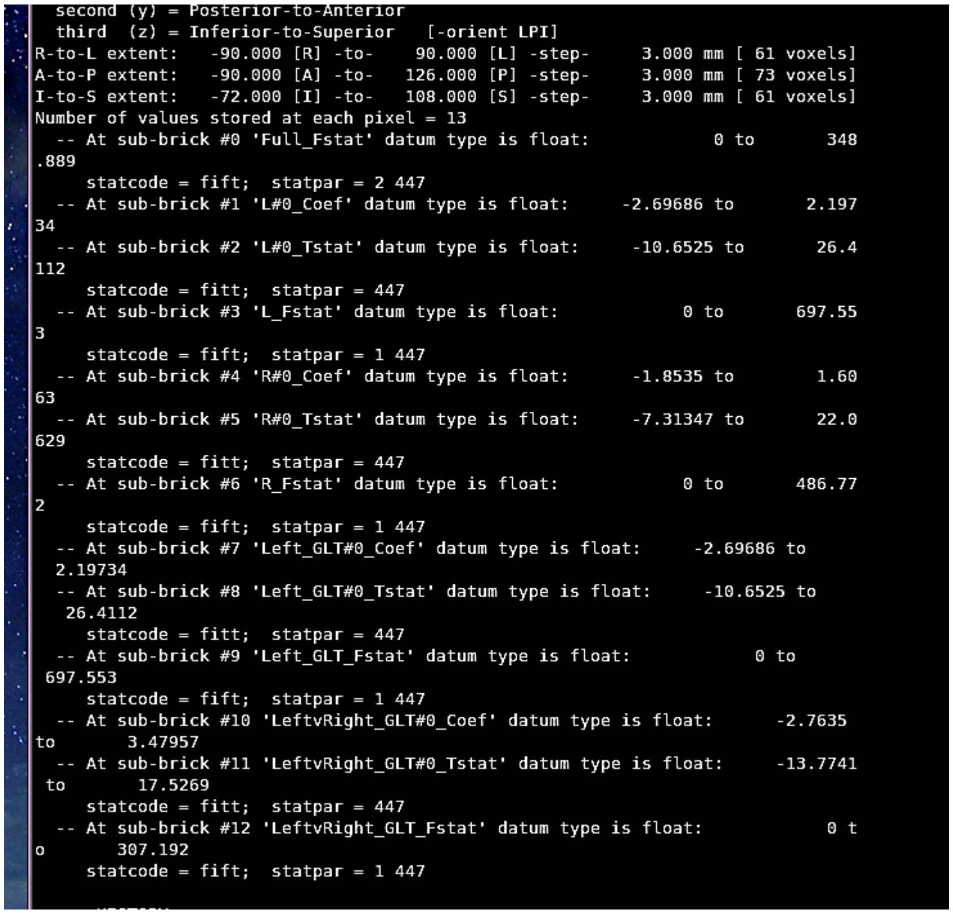
Step 2a – Indexing the data

Each subject's stats file contains a variety of information regarding the individual-level statistical tests you ran earlier. You will need to index for each subject the coefficients of these tests so they can be used in the group analysis. To find these indexes, navigate to one of your subject's results folders. Open a terminal in that directory, and use the function 3dinfo with the option 'verb'. Then specify that subject's stats file (instead of writing out the entire filename, you can drag the file over to the terminal and the filename will appear in your command):



```
[cgodwin9@cpu167 1_demo.results]$ 3dinfo -verb '/home/cgodwin9/Desktop/afni_proc
essing/1_demo.results/stats.1_demo+tlrc.BRIK.gz' 
```

When you press 'enter', you will see a large amount of information pertaining to 'sub-bricks'. Sub-bricks are arrays that store data, for example, here you can see there are sub-bricks that contain F statistics, t-values, and coefficients for various tests. You will want the coefficients for the contrasts you indicated in your GLM: For the LeftvRight contrast and for just the L contrast (which gets compared to baseline). Here, the coefficients for LeftvRight are stored at sub-brick 10. The coefficients for the L contrast are stored in sub-brick 1 [note that AFNI starts indexing at 0, not at 1!]. Take note of these values – you will need them in the next section.

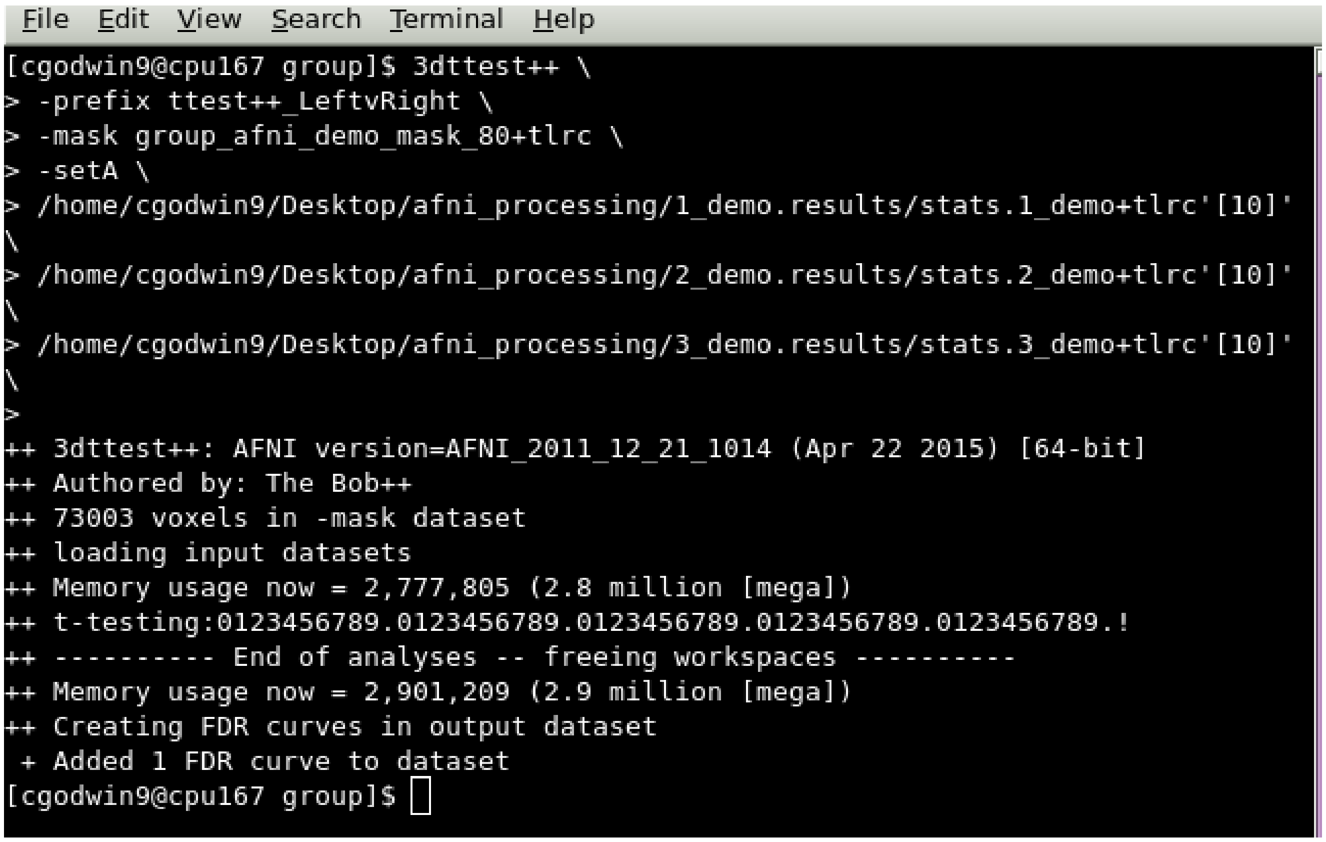


```
  second (y) = Posterior-to-Anterior
  third  (z) = Inferior-to-Superior   [-orient LPI]
R-to-L extent:   -90.000 [R] -to-    90.000 [L] -step-     3.000 mm [ 61 voxels]
A-to-P extent:   -90.000 [A] -to-   126.000 [P] -step-     3.000 mm [ 73 voxels]
I-to-S extent:   -72.000 [I] -to-   108.000 [S] -step-     3.000 mm [ 61 voxels]
Number of values stored at each pixel = 13
  -- At sub-brick #0 'Full_Fstat' datum type is float:            0 to       348
.889
     statcode = fift;  statpar = 2 447
  -- At sub-brick #1 'L#0_Coef' datum type is float:     -2.69686 to       2.197
34
  -- At sub-brick #2 'L#0_Tstat' datum type is float:     -10.6525 to       26.4
112
     statcode = fitt;  statpar = 447
  -- At sub-brick #3 'L_Fstat' datum type is float:            0 to       697.55
3
     statcode = fift;  statpar = 1 447
  -- At sub-brick #4 'R#0_Coef' datum type is float:      -1.8535 to        1.60
63
  -- At sub-brick #5 'R#0_Tstat' datum type is float:     -7.31347 to       22.0
629
     statcode = fitt;  statpar = 447
  -- At sub-brick #6 'R_Fstat' datum type is float:            0 to       486.77
2
     statcode = fift;  statpar = 1 447
  -- At sub-brick #7 'Left_GLT#0_Coef' datum type is float:     -2.69686 to
  2.19734
  -- At sub-brick #8 'Left_GLT#0_Tstat' datum type is float:     -10.6525 to
   26.4112
     statcode = fitt;  statpar = 447
  -- At sub-brick #9 'Left_GLT_Fstat' datum type is float:            0 to
 697.553
     statcode = fift;  statpar = 1 447
  -- At sub-brick #10 'LeftvRight_GLT#0_Coef' datum type is float:      -2.7635
to       3.47957
  -- At sub-brick #11 'LeftvRight_GLT#0_Tstat' datum type is float:     -13.7741
 to       17.5269
     statcode = fitt;  statpar = 447
  -- At sub-brick #12 'LeftvRight_GLT_Fstat' datum type is float:            0 t
o       307.192
     statcode = fift;  statpar = 1 447
```

*Note that we only looked up the sub-bricks for subject 1. If your subjects are organized in the same manner (which they should be), these sub-bricks will be the same for each person. However, I recommended viewing the stats files of a handful of your subjects to at least make sure.

Back in your group folder, you will run the command 3dttest++. An example of the syntax is provided for you in the group_analysis_readme. Again, you can read more about 3dttest++ on the AFNI site. For each t-test, make sure to specify the name of the datafile that will be output in this analysis. Indicate the mask that you will be using (here we are using our 80% overlap mask). Then you will individually indicate the subjects' stats files under the option -setA. In brackets you specify the sub-brick of data you want to test against zero. This is where you put the index values that we looked up in the previous step. When you're ready, run the command in the terminal from your group folder.

Terminal output after running 3dttest++:



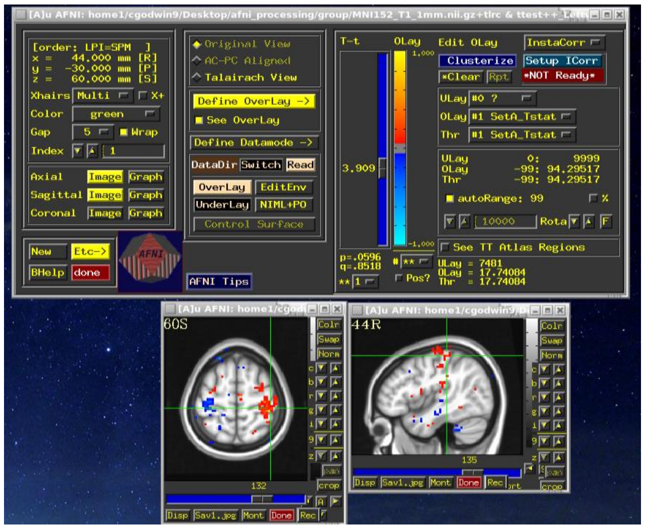
```
[cgodwin9@cpu167 group]$ 3dttest++ \
> -prefix ttest++_LeftvRight \
> -mask group_afni_demo_mask_80+tlrc \
> -setA \
> /home/cgodwin9/Desktop/afni_processing/1_demo.results/stats.1_demo+tlrc'[10]'
\
> /home/cgodwin9/Desktop/afni_processing/2_demo.results/stats.2_demo+tlrc'[10]'
\
> /home/cgodwin9/Desktop/afni_processing/3_demo.results/stats.3_demo+tlrc'[10]'
\
>
++ 3dttest++: AFNI version=AFNI_2011_12_21_1014 (Apr 22 2015) [64-bit]
++ Authored by: The Bob++
++ 73003 voxels in -mask dataset
++ loading input datasets
++ Memory usage now = 2,777,805 (2.8 million [mega])
++ t-testing:0123456789.0123456789.0123456789.0123456789.0123456789.!
++ ---------- End of analyses -- freeing workspaces ----------
++ Memory usage now = 2,901,209 (2.9 million [mega])
++ Creating FDR curves in output dataset
 + Added 1 FDR curve to dataset
[cgodwin9@cpu167 group]$ 
```

Step 3 – Viewing the Results

If you are not familiar with visualizing data in AFNI, first see Section 5 AFNI Viewer on how to navigate through the AFNI viewer.

Once you've run your t-tests, you can view the results in AFNI to determine what is significant. First make sure that the MNI template (MNI152_T1_1mm.nii.gz) is in your group folder. This is your anatomical underlay that will help you determine the anatomical regions that are significant.

Open AFNI in the terminal. With your underlay set to the MNI template, you will select your OverLay as the t-test file of interest. Here let's focus on the LeftvRight contrast. Overlay the ttest++_LeftvRight file. On the right-hand side of the viewer, set your Olay and Thr to SetA_Tstat. In the middle under the color bar, set the double-asterisk option on the left to 1. Adjust your threshold so that p = 0.05. At this point, some clear patterns of results should appear on the brain. In particular, you should see a large number of warm-colored voxels on the dorsal part of the cortex over the right primary motor cortex. You should see cool-colored voxels on the dorsal part of the left primary motor cortex. The warm (orange) colors represent significant voxels where activity increased in the “left” condition compared to the “right”. The cool (blue) colors represent significant voxels where activity decreased in the “left” compared to “right” conditions. Why do you think there is activation over the right primary motor cortex and deactivation over the left primary motor cortex? Think about the task subjects were required to perform. Scroll through the brain and find other prominent regions with clusters of significant activity.


```
[A]u AFNI: home1/cgodwin9/Desktop/afni_processing/group/MNI152_T1_1mm.nii.gz+tlrc & ttest++_Left

[order: LPI=SPM  ]
x =    44.000 mm [R]
y =   -30.000 mm [P]
z =    60.000 mm [S]
Xhairs  Multi      X+
Color   green
Gap     5          Wrap
Index   1

Original View
AC-PC Aligned
Talairach View
Define OverLay ->
See OverLay
Define Datamode ->
DataDir Switch Read
OverLay  EditEnv
UnderLay NIML+PO
Control Surface

T-t    OLay   Edit OLay   InstaCorr
       1.000  Clusterize  Setup ICorr
              *Clear Rpt  *NOT Ready*
3.909         ULay #0 ?
              OLay #1 SetA_Tstat
              Thr  #1 SetA_Tstat
              ULay      0:     9999
              OLay    -99: 94.29517
              Thr     -99: 94.29517
              autoRange: 99          %
              10000  Rota  F
       -1.000 See TT Atlas Regions
p=.0596   # **     ULay = 7481
q=.8518   Pos?     OLay = 17.74084
** 1               Thr  = 17.74084

Axial     Image Graph
Sagittal  Image Graph
Coronal   Image Graph
New   Etc->
BHelp done
AFNI Tips
```

*Note that in this example, we set our threshold to be p = 0.05. If you look below, you see there is also a 'q' threshold as well. This q-value is an FDR-corrected threshold, which is a common way to correct for multiple comparisons in fMRI analysis. Correcting for multiple comparisons is extremely important in fMRI because we have run our t-tests on thousands of voxels – there is guaranteed to be voxels that are significant just by chance, so it is important to account for this. In general, a common threshold for the q-value is also 0.05. In this example, you can see that we are not able to reach that value. That is because we only have 3 subjects in this example. If we had a complete dataset, then we would have much more power and would be able to draw inferences from our data based on the corrected q-value rather than the p-value. In your own analyses, you should aim to use the q-value for your statistical threshold (or implement one of the other ways to correct for multiple comparisons in fMRI data).

AFNI Viewer Tutorial

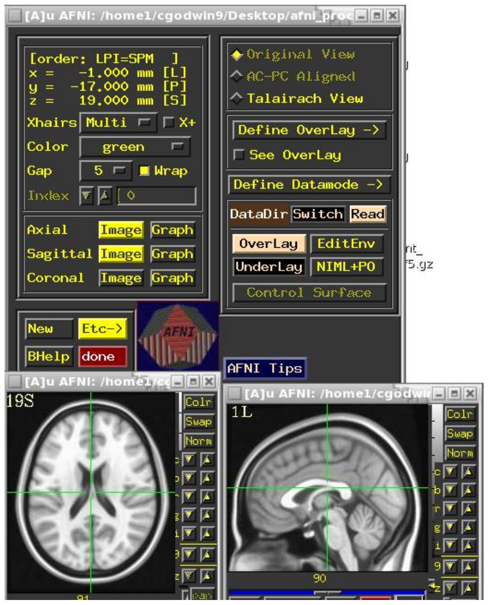
In this section, we introduce the viewer program that is used for visualizing results. It is an essential tool for interpreting and reviewing data.

The main file format AFNI uses is to separate data into two files: a BRIK file that contains the data, and a HEAD file that contains header labels, so that the data points are labeled and organized properly. AFNI needs both of these files with the same name for the data to load properly.

To run this part of the tutorial, make sure you are in the folder 'viewer_tutorial'. You should see a ttest results file that will give you the opportunity to view fMRI results. Also make sure to copy the MNI152_T1_1mm_brain.nii.gz template into this folder.

Step 1 - Navigating the brain

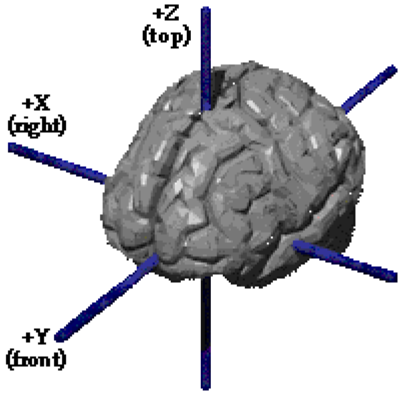
The first part of this section will introduce you to a standard structural image of the brain, the MNI template. To start, navigate to the viewer_tutorial folder. Open the terminal, type 'afni' (without quotes), and press enter. Once AFNI starts up, it should have three windows that look like this:

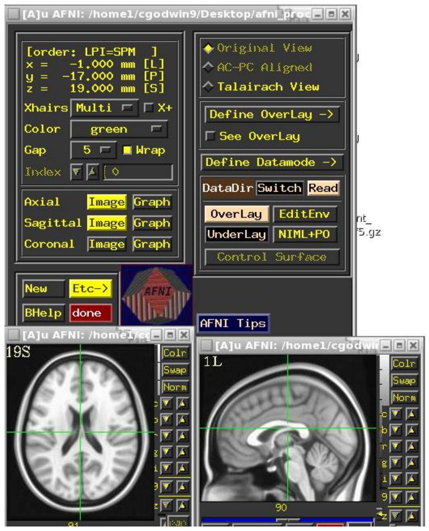


```
[order: LPI=SPM  ]
x =   -1.000 mm [L]
y =  -17.000 mm [P]
z =   19.000 mm [S]
```

Xhairs Multi, Color green, Gap 5, Wrap

Let's focus first on the main settings window. The coordinates at the top left show where in the brain where the crosshairs currently focus. The letters in brackets show you if you are on the [L]eft or the [R]ight (x axis), [A]nterior or [P]osterior (y axis), and [S]uperior or [I]nferior (z axis), as illustrated below.





```
[order: LPI=SPM  ]
x =    -1.000 mm [L]
y =   -17.000 mm [P]
z =    19.000 mm [S]
```

Click on 'Multi' next to 'Xhairs' and turn it to 'Off'. The cross hairs below disappear! This is useful if you are saving an image of a brain map and don't want to crosshairs in the way. For now, the cross hairs are useful, so turn 'Xhairs' back to 'Multi'. Set 'Color' to your favorite color.

Below in the main settings are options to select which views of the brain are displayed in separate windows. By default, the Axial and Sagittal views are displayed. But the Coronal view is nice too, so click 'Image' next to Coronal. The Coronal view pops up on top of the other windows. Move the window over to the side of the other brain images, so that the windows don't overlap.

Clicking 'New' would open another instance of AFNI. This is useful if you want to compare different results, as the crosshair positions are linked between the two AFNI instances. ‘BHelp’ is useful if you’re not sure what a button in AFNI does. Click ‘BHelp’ then click on any other button in AFNI. A pop-up window will give you a description of what the button does. Try this out on a button you’re curious about. Clicking the ‘done’ button twice will close AFNI, but don’t do that right now, we’re not done.

The ‘Define Datamode’ has many additional options, but we won’t use most of them.

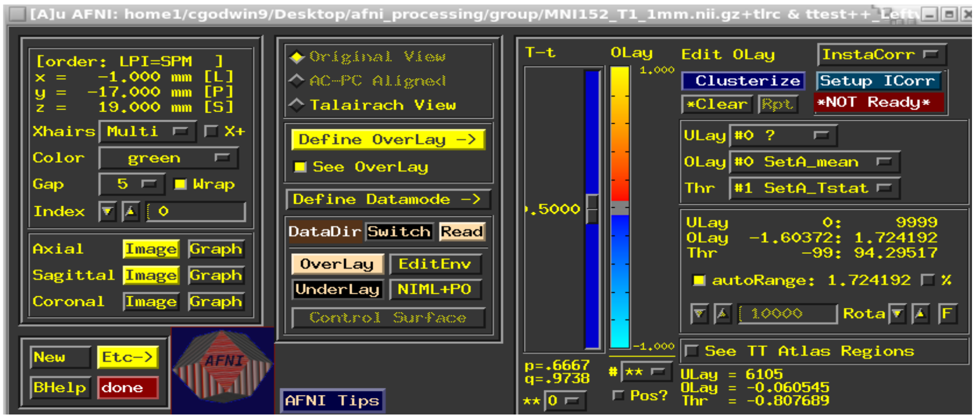
The next buttons we’ll use are the OverLay and UnderLay buttons used to choose which files of brain images are loaded in the other windows. The UnderLay button loads the anatomical brain image and the OverLay button loads the activation maps that you want to display on top of the brain.

Step 2 – Loading imaging into the viewer

Click the ‘UnderLay’ button. A pop-up window will appear with a list of all the brain image files in the folder you are in. If it is not already selected, click on the ‘MNI152_T1_1mm_brain.nii.gz’ file and then click ‘Set’ to set the underlay. You should see the anatomical brain in the other viewing windows. Try clicking around the brain. Notice that the views change on the other windows when you move the crosshairs. Also notice that the coordinates at the top of the settings window change to different values as you move the crosshairs.

Now click the ‘OverLay’ button. Select “ttest_LeftvRight+tlrc” and click ‘Set’.

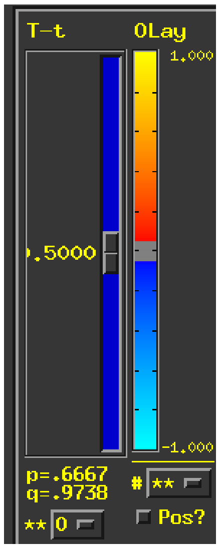
Once you have loaded an overlay, you will notice that the settings window expands to the right with some new buttons, like this:


```
[A]u AFNI: home1/cgodwin9/Desktop/afni_processing/group/MNI152_T1_1mm.nii.gz+tlrc & ttest++_Left

[order: LPI=SPM   ]
x =    -1.000 mm [L]
y =   -17.000 mm [P]
z =    19.000 mm [S]
Xhairs Multi   X+
Color  green
Gap    5   Wrap
Index  0

ULay #0 ?
OLay #0 SetA_mean
Thr  #1 SetA_Tstat

ULay         0:    9999
OLay  -1.60372: 1.724192
Thr      -99: 94.29517
autoRange: 1.724192  %
10000  Rota

ULay = 6105
OLay = -0.060545
Thr  = -0.807689

p=.6667
q=.9738
```

The Overlay file can actually contain several 'sub-bricks', or arrays of data, within the one file. Also, AFNI starts indexing from 0 rather than 1. This file has two sub-bricks: one for the mean (the average difference in activation across participants) and one for the t-value. We are interested in what is statistically significant, so we need to select the t-value sub-brick. On the right, select sub-brick #1 (SetA_Tstat) from the drop down menu next to “OLay” and “Thr”. Be sure to set both of them to the same sub-brick.

Step 3 – Setting the color bar

Right click the rainbow color bar in the middle of the settings.



Click “Choose Colorscale” (You may need to hold down the mouse button after right clicking to accomplish this).

Choose “Spectrum:yellow_to_cyan+gap”, then click “Set”. This is the standard colorscale that we use, where positive activation is orange and negative activation is blue. For this example, orange-colored brain areas were more active when tapping with the left hand (compared to right hand) and blue-colored brain areas were less active when tapping with the left hand (compared to right hand).

Step 4 – Saving the layout

Any changes you make to AFNI (loaded files, window placement, color bar) will be reset when you close and open it again. In order to save the layout for the next time you open AFNI, click “Define Datamode ->”, then “Misc”, then “Save Layout”. Don’t write anything, but click the “Set” button. Click “Define Datamode” again to hide the Datamode options, so you can see the color bar again.

Now when you exit AFNI and start it up again, the settings you changed will be saved. Note that the layout is only saved in the current terminal folder where you opened AFNI. If you open AFNI in another folder, the default settings will load.

Step 5 – Setting and interpreting a threshold

The blue bar next to the color scale can be moved up and down to change the threshold of the stats displayed on the brain. Each voxel in the brain has a positive or negative *t*-value associated with the contrast that was run. However, you only want to display voxels that reach statistical significance. The number next to the blue bar is the cutoff for the *t*-value that you set. All voxels of greater magnitude will be displayed; all voxels of smaller magnitude will be transparent.

To set the threshold, first click on the number next to two asterisks (**) at the bottom. Set this to 1. This is the order of magnitude for the *t*-scale, and you should set it to "1" because the cutoff you want is probably between 0 and 10.

Now drag the slider on the blue bar up and down. Notice that the values for "p" and "q" change beneath it. "p" is the uncorrected *p*-value of the T-test; uncorrected because it doesn't take into account the huge number of voxels that *t* values were calculated for. "q" is the false discovery rate (FDR), which does take into account the number of tests. You want to adjust the slider up or down until q = .0500, equivalent to a 5 percent false discovery rate. If the fickle data gods are not smiling upon you, you may not be able to raise the bar to q =.05, or it may almost totally wipe out any activity in the brain. If this is the case it is common to set p =.001, though results at this lenient threshold should be interpreted with caution, as there is an inflated chance of false positives. When exploring your data, it is good to check out the pattern of activation at more lenient thresholds like p = .001, p = .01, or p = .05. Activity that is below a properly corrected threshold might be significant in subsequent analyses (e.g., regions of interest).

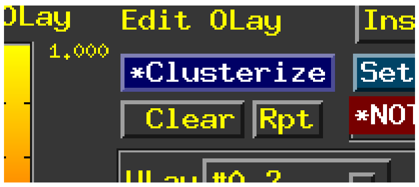
In this example, we only have 3 subjects and little statistical power. Therefore, we cannot reach q = 0.05. For viewing purposes, go ahead and set p = 0.01.

If you click the button that says "Pos?" below the colorbar, the overlay will display only positive values. Since we want to see both positive and negative values, be sure that this button is not on. You should be able to see both orange and blue spots on the brain images.

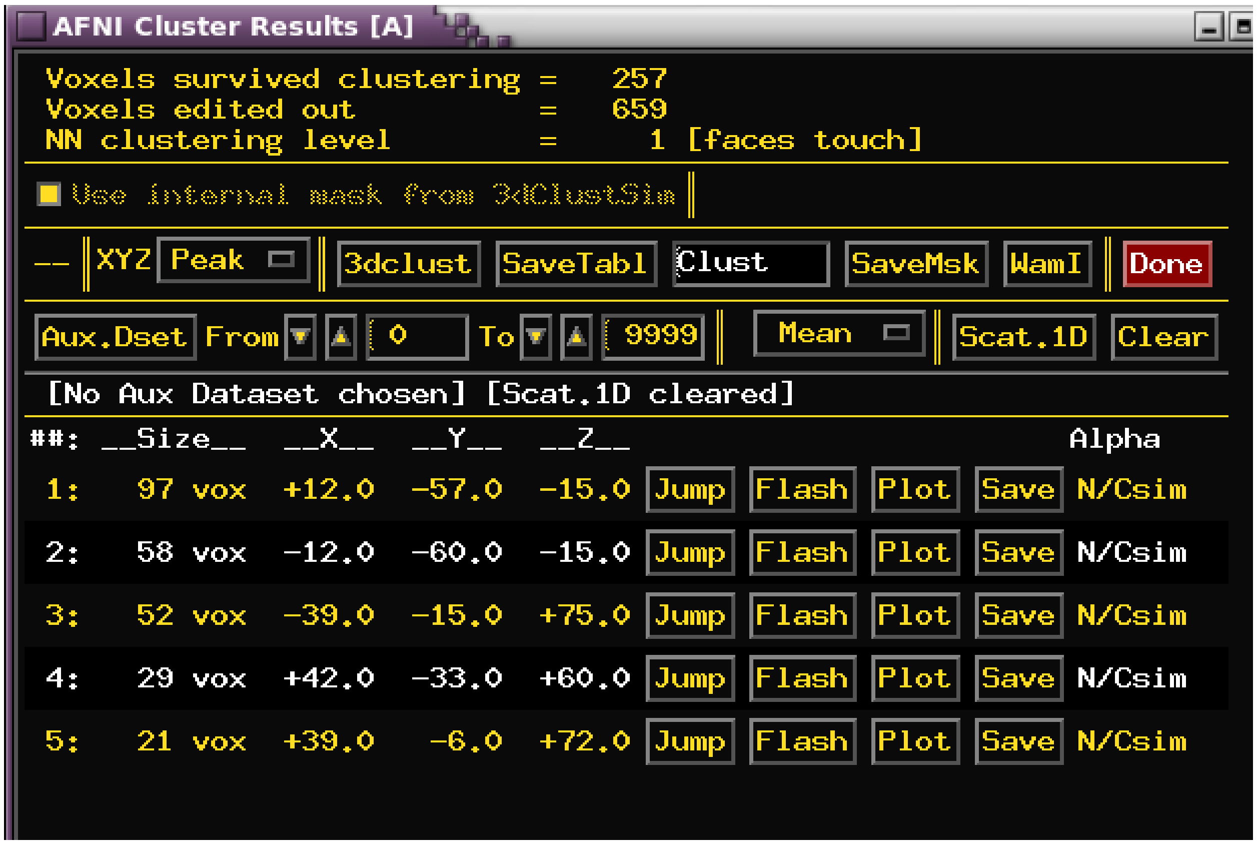
Step 6 - Clustering

The brain activity you are interested in spans regions of more than a single voxel. Due to noise, there will likely be stray voxels across the brain that survived the significance threshold. To get rid of this noise, we **clusterize**: we choose a number of voxels that have to be touching to be displayed as significant activation. **Note**: if you have properly corrected your threshold, even a single significant voxel is of interest and might indicate surrounding brain activity that is just below the threshold. The following instructions for clustering will help to present clear result images and create tables of activation, but be careful to not ignore any potentially interesting results that do not meet cluster size requirements.

Click the "Clusterize" button to the right of the color bar. On the pop-up menu, leave NN-level set to 1 and Bi-Sided set to No. Choose the number of touching voxels that you want. This can depend on the size of the brain region you are interested in, but 20 is typically a good number. Click "Set".



Now, to see more info about the significant clusters, click "Rpt" directly under "Clusterize" for a cluster report. It should look like this:



```
AFNI Cluster Results [A]
Voxels survived clustering =   257
Voxels edited out          =   659
NN clustering level        =     1 [faces touch]
```

Use internal mask from 3dClustSim
-- XYZ Peak 3dclust SaveTabl Clust SaveMsk WamI Done
Aux.Dset From 0 To 9999 Mean Scat.1D Clear

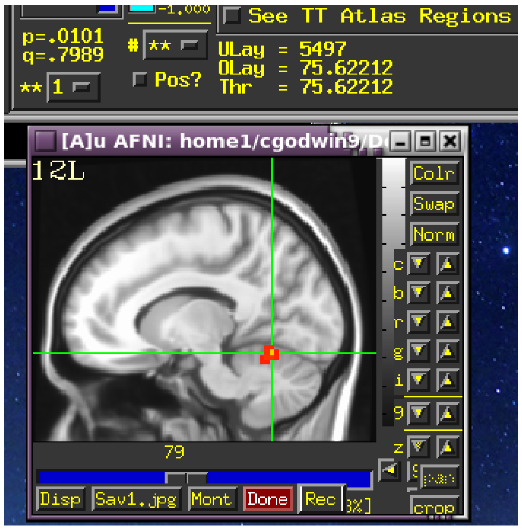
```
[No Aux Dataset chosen] [Scat.1D cleared]
##: __Size__  __X__  __Y__  __Z__                              Alpha
 1:   97 vox  +12.0  -57.0  -15.0 Jump Flash Plot Save N/Csim
 2:   58 vox  -12.0  -60.0  -15.0 Jump Flash Plot Save N/Csim
 3:   52 vox  -39.0  -15.0  +75.0 Jump Flash Plot Save N/Csim
 4:   29 vox  +42.0  -33.0  +60.0 Jump Flash Plot Save N/Csim
 5:   21 vox  +39.0   -6.0  +72.0 Jump Flash Plot Save N/Csim
```

This is a list of the significant clusters, from largest to smallest. The “Size” is the voxel size, the X Y Z coordinates are the voxel within the cluster with peak activation.

Click “Jump” on line 1. On the brain views, the crosshairs jump to the peak voxel for that cluster.

Click “Flash”. The cluster flickers on the brain view. This is useful for seeing the extent of one cluster and discriminating between nearby clusters.

Look back on the bottom right corner of the settings window, it should look something like below. The value after “OLay” is the t-value of the voxel under the crosshairs. This information is useful for creating tables of activation that include the *t*-value at the peak voxel of each cluster. Here is an example from one of the clusters:



Back on the cluster report, click on "WamI" at the top. This may take a minute to load. A window should pop up, comparing each of your clusters to anatomical brain atlases. This is useful for a table of activation: you can determine the probable extent of overlap with gyri and Brodmann's areas for each of your clusters.

Step 7 – Navigating the brain

Return to the brain view windows. Since you can only look at one slice at a time, it is nice to scan through the whole brain to see the full pattern of significant activity. Click on the axial, sagittal, or coronal viewing window and leave the mouse over the window. Now press "v" on the keyboard. The window should scan through the slices, so that you can see the whole brain pass. Press "b" to stop the automatic progression.

As you click around the brain, you might at times be curious about the anatomical structure of wherever the crosshairs are pointing. To find out, hold down the right click button over a brain view window, and select "Where Am I?" A window will pop up giving you the coordinates in MNI and Talairach space, as well as probabilistic labels for the underlying anatomical structure (e.g. right cingulate gyrus), based on several brain atlases. Once the “where am I” window is open, try clicking around the brain some more and notice how the coordinates and brain regions change with the cross hairs.

Bash Scripting

To successfully perform fMRI analysis in AFNI, it is important to have at least a basic understanding of Linux and interacting with the command line. When you work in the command line, you use a program called *shell* that takes the keyboard commands and passes them to the operating system so that they can be carried out. This shell program is called "bash", and scripting in this program is often referred to as bash scripting.

This tutorial provides two examples of basic bash scripting. Follow along with them below. For your convenience, the complete scripts have been provided for you in separate .sh files: bashExample.sh and testIteration.sh

The information below is also provided in the scripts' readme files. You may find it helpful to open the readme files in a text editor like gedit, which will help with formatting. Open the readme files in one window and each script in another window and follow along until you are ready to run the scripts.

Example 1 (bashExample.sh)

Before completing this example, navigate to the "Bash" folder in your AFNI Tutorial directory and run the following:

```
sh bashExample.sh
```

What happens? The following readme will walk you through the scripting basics that produced this output.

The following script acts as a wrapper function to run our example process some number of times. The pound (#) sign allows you to comment in your script. This is helpful for long scripts or scripts that make multiple calls to other scripts to help explain to others (or, often, yourself) what your thought process was when you first made the script. The comments in this script will help explain the different functionalities being demonstrated here.

The first two lines remove any files created from previous runs of this script.

The following two lines demonstrate variable assignment and calling in bash. To assign a value to a variable, simply type the name of your variable, an equals sign (WITH NO SPACES), and the value you will be assigning to your variable.

Bash variables default to string values; that is, it assumes they are lines of text. This does not usually cause issues for us in AFNI, because if we are using variables in our scripts it is typically to designate file and folder names that we will be operating on.

The second line of this pair uses the "echo" command. Echo is a built-in bash command that prints strings to an output location for you to read. In this example, calling our variable1 with the "$" operator causes echo to print our variable value. But to where?

When you run the script in terminal, you will see the line "Bash Example Output Log printed to the terminal screen. However, if you look in your current directory, you will see a new file called "testOutput.txt". If you look at the echo command, you will see the operators "|" and "tee", along with this testOutput.txt filename. The | operator tells echo the end of the desired echoed string and print it to the terminal. The tee command, then, tells echo to also append (versus write, which replaces anything in the file currently) the string to a text file. This is good for data logging, especially when running multiple scripts and processes.

In this example, what will the first line of testOutput.txt read?

```
variable1='Bash Example Output Log'

echo $variable1 | tee -a testOutput.txt
```

The next section of code is enclosed in a "for" loop. For loops are powerful tools that allow us to run the same operations repeatedly on different sets of data. Most often, you will use for loops to allow your script to analyze multiple subjects and/or runs in one go.

A few key points surround for loops. First is the syntax of the first line. When you call a for loop, you follow the format "for [VARIABLENAME] in [LIST OF VALUES YOU ARE OPERATING ON]". The variable name you use must start with a letter and may not have spaces or special characters. This variable will serve as a placeholder in the remainder of your code to indicate which values in your code should take on the various values in your list.

The list of values you give can have as many individual values as you want. Values are read in as strings one at a time, where the values are separated by spaces. On each pass through the loop, your variable name will take on the next value in your list, and the full block of code contained in the loop will be executed for that value. Your loop will make exactly as many passes through the code block as there are values in your list.

The remaining key points are the "do" and "done" lines. These lines indicate what portion of your script is to be executed as a part of the for loop. Failure to include these lines in your script will cause bash to misread the organization of your code in what is called a “parsing” error. The error you will typically get if this happens will tell you that bash reached the end of file before it finished reading the file, because it was waiting for the remainder of the loop.

In the example below:

What is our variable name?
How many times will we pass through the loop?
Which commands will not be executed in the loop? When will they be executed?

```
for iteration in 1 2 3 4 5

do

     echo 'Performing Iteration #'$iteration': ' | tee -a testOutput.txt
```

This snippet of code uses an “if” statement to execute different blocks of code depending on whether a set of conditions is fulfilled or not. The backbone of this code block is the “if [TEST COMMAND]; then [CONSEQUENT COMMANDS] else [CONSEQUENT COMMANDS] fi” structure. This indicates where the statement begins, where different conditions may be tested, what to do in each case, and where the statement ends. The if statement looks to see if the test condition is true. If it is, it executes the commands associated with the “then” section; otherwise, it executes the commands after the “else”, which serves as a catch-all.

There are two additional options to this basic structure. If there is no code to be executed if your test command turns out to be false, then you can simply leave off the “else” section, which will cause the statement to jump straight to the “fi” line and do nothing within the if statement. On the other hand, if you have different types of code that you want to execute depending on which of several conditions is met (for example, for odd numbered versus even numbered subjects), you can insert any number of “else if” statements with the format “elif [TEST COMMAND 2]; then [CONSEQUENT COMMAND]”. In this case, your statement will look for the first true test command and execute whatever code is associated with that if or elif block.

When using if statements, try to ensure that your test commands are mutually exclusive and exhaustive for your data. This is the easiest way to avoid errors in your analyses caused by failing to handle special cases in your data. Note that your code may not crash if you have misspecified your if statement; if, for example, you have divided your subject numbers into odd and even, but subject 5 needs to be run with the even group, your code will likely not let you know if you forgot to specify that!

In the example below, the test command is looking to see if there is a directory (-d command) in the folder from which you are running our example. The "!" operator means "not" - that is, we are specifically testing if there is NOT a directory called "bashTestOutput.

What happens if there is a directory called bashTestOutput"?

What happens if there is not a directory?

```
if [ ! -d bashTestOutput ] ;
        then
            echo '    Making test directory.' | tee -a testOutput.txt
            mkdir bashTestOutput
      else
            echo '    Test directory exists.' | tee -a testOutput.txt
      fi

      echo '    Running test script for this iteration… ' | tee -a testOutput.txt
```

The following line demonstrates the power of scripting once you have mastered the basics. The "sh" command likely looks familiar; it is the same command you use in terminal to run your scripts. When you use this command in bash, the parent script runs the called script just like in terminal. Doing this starts what is called a "stack" - that is, the order of operations in which your scripts are being run. Once you call a script from a parent, that new script executes to completion before returning to the parent script to continue its processing. This functionality can be used to complete analyses for subjects one at a time, or to ensure that you complete one process before starting another.

When calling scripts, you can pass values in to be operated on. These can be set values, as in strings or numbers, or can be variable values that were produced earlier in the parent scripts. These values are indicated after the name of the called script in the order in which they are assigned in the code in the called script (more on that later).

In this example:

what script is the parent?

what script is being called?

which script will finish running first?

what value is passed to the called script on the 3rd pass through the for loop?

```
      echo '    Test script output for iteration #'$iteration': ' | tee -a
testOutput.txt
      sh testIteration.sh ${iteration}
```

```
done
```

These last lines use the built in “date” function to display what time your script completed running, just for fun. You can find more information about this and other built-in functions online. Bash is both very powerful and at the same time very limited. There is usually a way to accomplish whatever processing you might have in mind; see what others have done to determine what is possible and what might be more cumbersome than it is worth.

```
hh=$(date +%k)
mm=$(date +%M)
echo 'Bash Example Complete '$hh':'$mm'. Thank you for playing!' | tee -a
testOutput.txt
```

At this point, you should proceed to the testIteration.sh file to continue the tutorial example.

Example 2 (testIteration.sh)

The following script looks short, but it is actually the script that in most cases will do the most work in our scripting designs. Here you will put the heavy computations or command calls that will be run on each iteration of your loop.

The main reason for this script is to show you how argument passing works in scripting. In the parent script, we called this script along with the name of a variable. Bash passes this information to the called script in the order it is inputted. In our case, we only passed one variable; however, depending on what you might want to accomplish, it may help to pass multiple values, such as subject numbers, directory names, run numbers, etc.

To use these passed values in the called script, we first need to assign that value to a variable in this script. As a general rule, scripts cannot access variables from each other unless they are directly passed through inputs and outputs and captured in some temporary variable form. This ensures that an error in one script does not propagate to others with no control processes in place.

To assign our variable, we simply select a variable name and set it equal to “$1”. This is the same format as any other variable setting; however, because the script does not otherwise know what input is coming into it, bash automatically assigns these inputs to the variables “1”, “2”, etc. To assign additional variables, you would just set your variable name to the serial position of the input value in the call.

Extra inputs that are never assigned to a variable will not break your script, but will not be available for use in the called script; however, failure to pass enough variables to your script will cause an error.

$RANDOM is a built-in variable in bash that produces, unsurprisingly, a random integer (0 - 32767). Like any other variable, it is called with the $ operator.

```
var1=$1
randNum=$RANDOM
```

In the conditional test expressions here, you will see a % symbol. This is used in bash to call the modulo operator. This operator divides the two arguments (here, the random number and your input variable) and returns the remainder. For example, 5 modulo 2 would equal 1, because 5/2 equals 2 with a remainder of 1.

The test condition is driven by the -eq command. This command compares the values on its left and right and determines if they are equal, returning a true or false value. The if statement then uses this value to determine which code block to run.

There are many other ways to manipulate numerical variables; if you can think of it, it is probably possible. Google is your friend!

```
if [ $[$randNum%$var1] -eq 0 ];
  then
      echo '        Sally sells seashells by the seashore. ('$randNum')' | tee -a
testOutput.txt
elif [ $[$randNum%$var1] -eq 1 ];
  then
      echo '        Are we there yet? ('$randNum')' | tee -a testOutput.txt
elif [ $[$randNum%$var1] -eq 2 ];
  then
      echo '        Scripting is fun! ('$randNum')' | tee -a testOutput.txt
elif [ $[$randNum%$var1] -eq 3 ];
  then
      echo '        DON'T PANIC ('$randNum')' | tee -a testOutput.txt
else
      echo '        You win! ('$randNum')' | tee -a testOutput.txt
fi
```

Can you walk through this script in your head and determine what will happen for the following
input values?
3
5
12

In many cases, you will call additional scripts or AFNI functions here. These work just like the call we made in our parent function with sh or the commands we have been running in the terminal; however, scripting allows you to:

1) Run multiple commands in a row
2) Run the same commands for multiple datasets, which is helpful for preprocessing
3) Format your commands on multiple lines for readability using the \ operator
4) Debug your code for a test subject so it can be run at once without errors on future subjects.