HD4630 Workshop II

First- & Second-Level Analysis

Elizabeth DuPre

Human Neuroscience Institute
Department of Human Development
Cornell University

Workshop I Recap

- Preprocessing
  - Discard pre-steady state TRs
  - Slice-timing correction
  - Rigid-body motion correction
  - Coregistration of functional & anatomical
  - Normalization to standard template
  - Spatial filtering (Smoothing)
- Quality Control
  - Visual inspection
  - Censoring or ”scrubbing” motion

Plan for Today

- Discuss first- and second-level analyses in a traditional general linear model (GLM) framework
- Conduct a first-level, fixed-effects analysis in AFNI using `uber_subject.py`
- Conduct a second-level, random-effects analysis in AFNI using `uber_ttest.py`

First-level Analysis

- First-level analysis involves estimating the β-matrix in

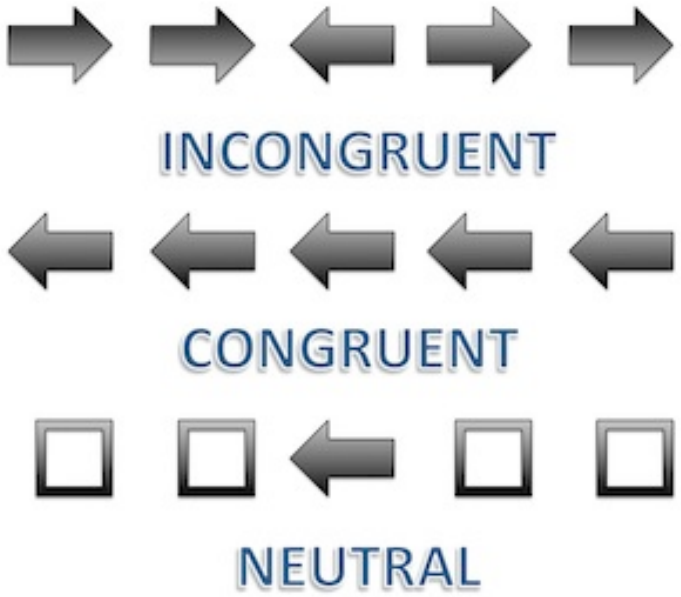
$$Y = X\beta + \epsilon$$

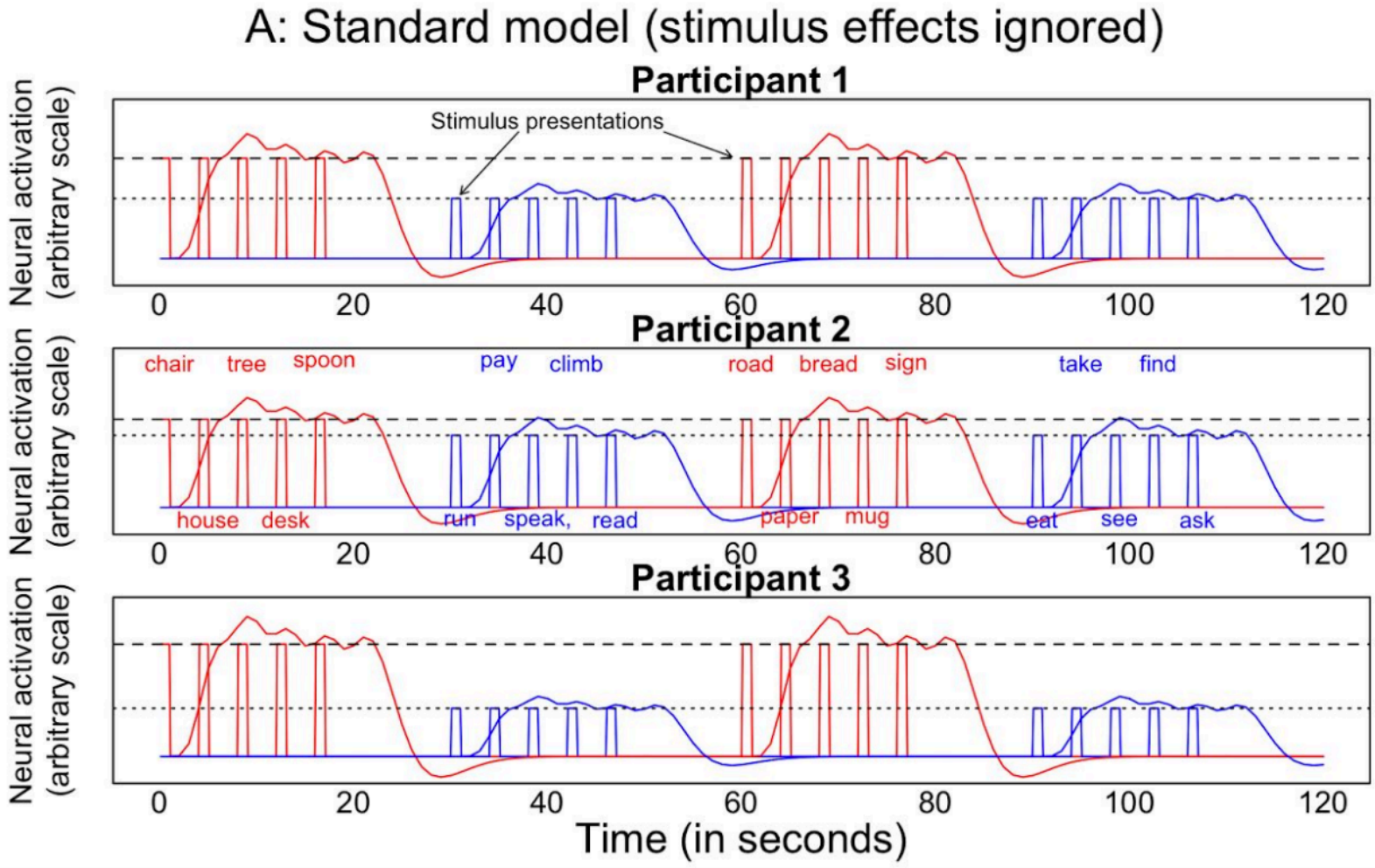
  by constructing the contrast matrix X

- A great review of the math underlying this is available from Mumford Brain Stats

First-Level Analysis, *cont.*

- We need to input the stimulus timing for each of the conditions in our task
  - We'll collapse the two Flanker 'congruent' and 'incongruent' conditions, ignoring participant accuracy





Westfall, Nichols, & Yarkoni

bioRxiv

Second-Level Analysis

- Second-level analysis as implemented in fMRI takes a summary-statistics approach

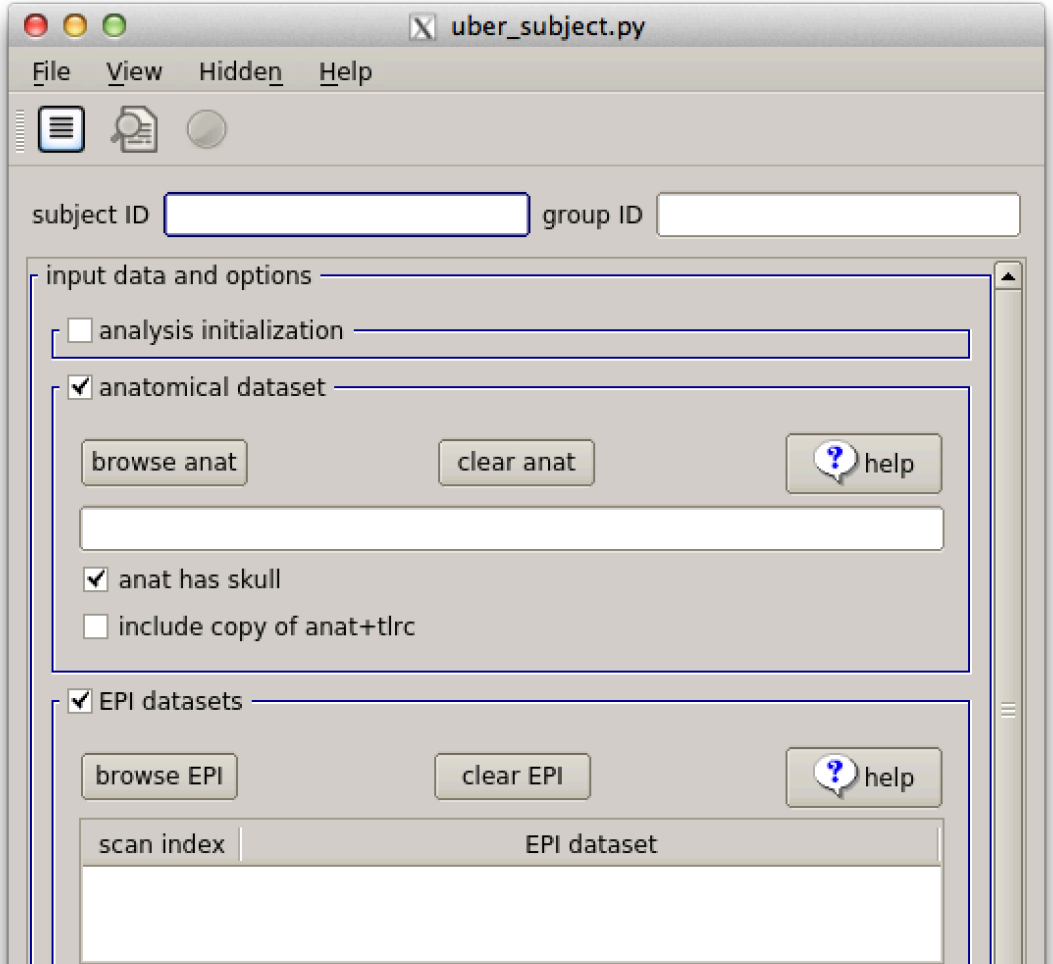
$$\hat{\beta} = X_g \beta_g + \eta^*$$

  where beta-hats for each contrast, for each subject are carried forward from the first-level

- A great review of the math underlying this is available from Mumford Brain Stats

To Do: Set Preprocessing Options

- These were discussed in the last workshop
  - For a review, see last week's slides on the course website

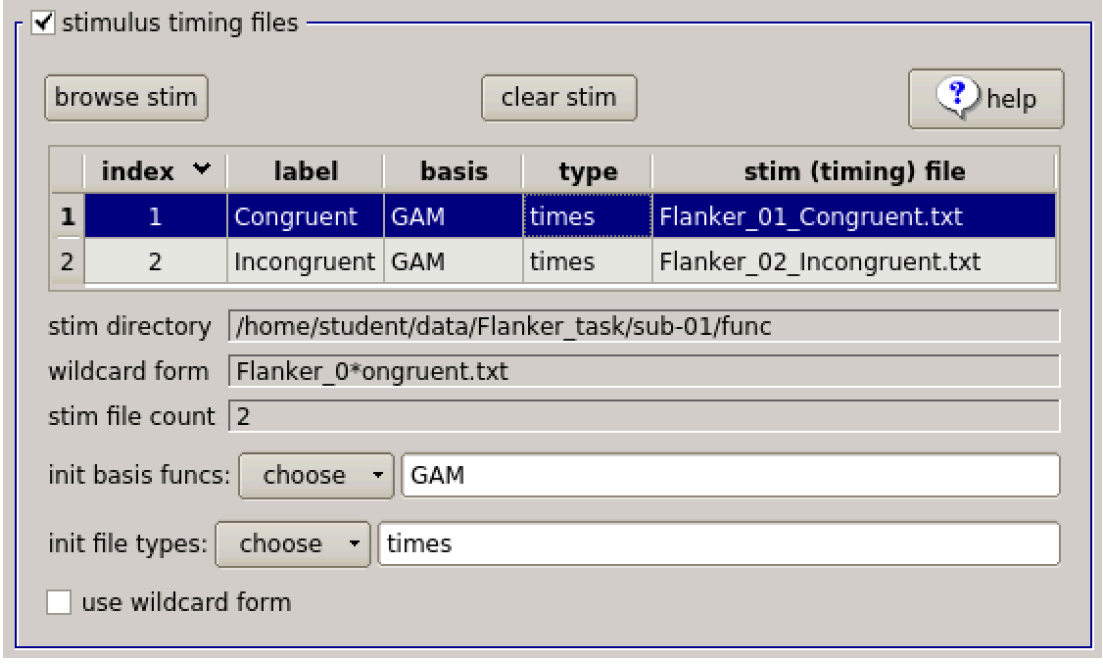


Stimulus Timing Information

- All stimulus timing information is included in the `*events.tsv` files for each run
  - However, we need to convert this information into text files that AFNI will accept
- You'll be provided with a folder containing
  - A Python script to convert these files
  - The created text files themselves

To Do: Enter Stimulus Timing

- In `uber_subject.py`, select the provided text files in the section 'Stimulus Timing Files'
- You can also specify the basis function and the file 'type' (see `3dDeconvolve -help` for relevant options)

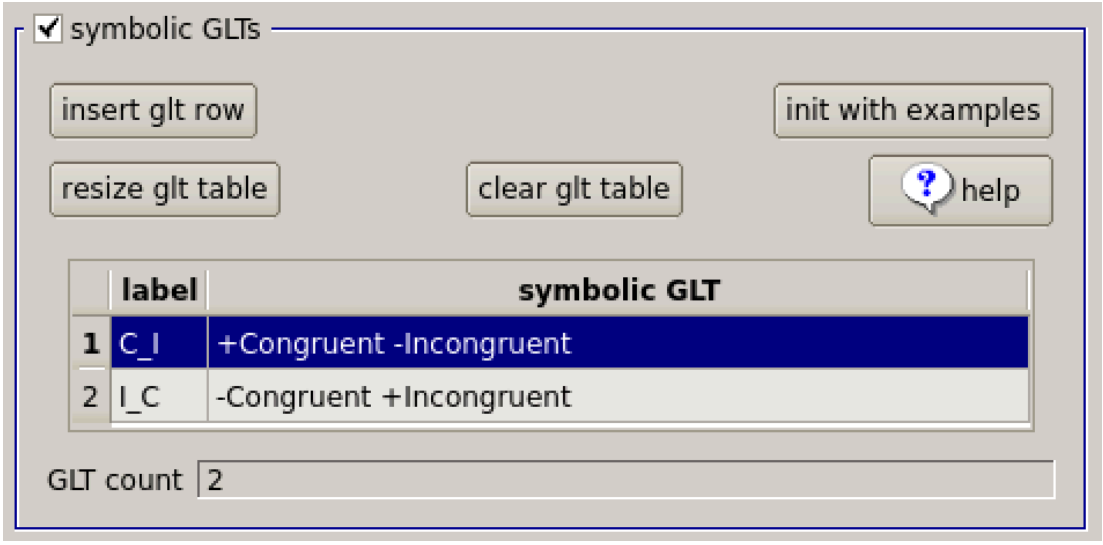


General Linear Tests (GLTs)

- AFNI refers to contrasts between conditions as General Linear Tests (GLTs)
  - This is because we're working in the General Linear Model (GLM)
- We'll need to specify what tests we would like to conduct
  - *Congruent* > *Incongruent*
  - *Incongruent* > *Congruent*

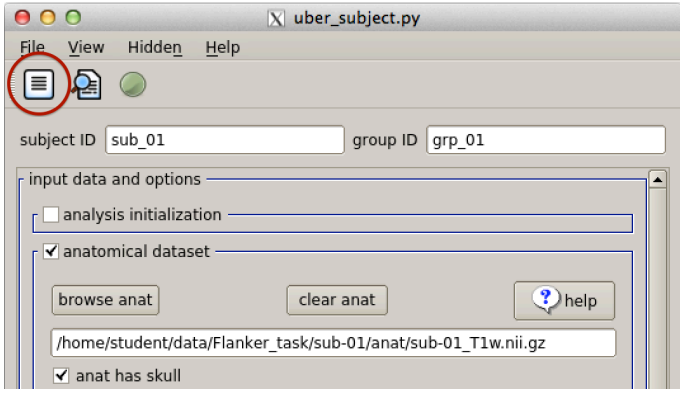
To Do: Specify GLTs

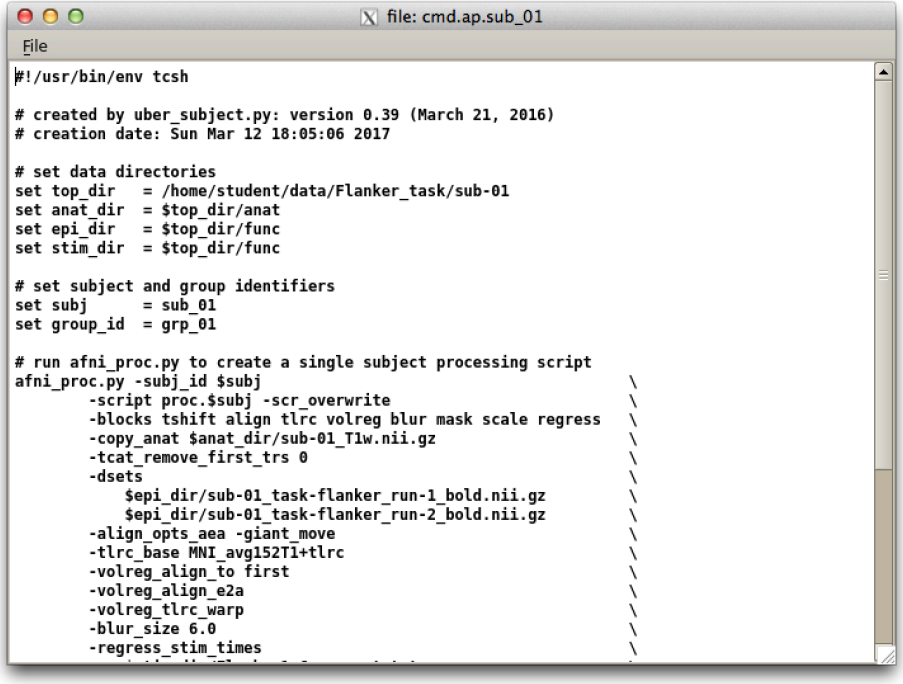
- When inputting stimulus timing files, AFNI will automatically associate each condition with your provided label
- You can use those labels to enter your desired GLTs
  - Select 'init with examples' to see example GLTs with your conditions



To Do: Review Processing Script

- Once you've entered all relevant parameters for preprocessing and first-level analysis, we can review and run the associated script
- Estimated run time is 40 minutes per subject





The Design Matrix

- All of the information we have entered thus far will form the X or 'design' matrix
- This is entered into the model

$$Y = X\beta + \epsilon$$

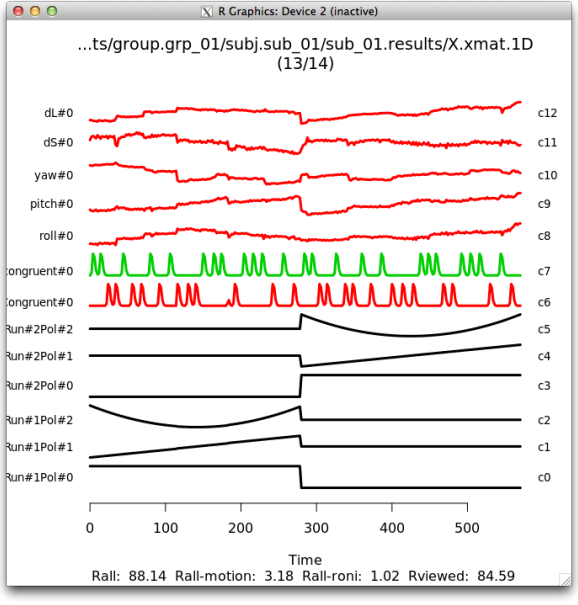
and will allow us to estimate the β-matrix for each participant

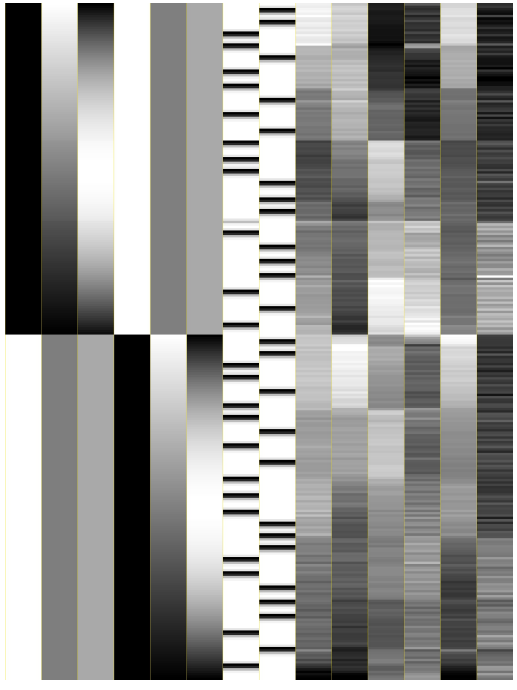
The Design Matrix, *cont.*

- After `uber_subject.py`, the design matrix is output as a `*.xmat.1D` file
- Multiple versions of this file exist, including
  - A list of input stimulus timings (`X.stim.xmat.1D`)
  - A full design matrix with no motion censoring (`X.nocensor.xmat.1D`)
  - A full design matrix with motion censoring (`X.xmat.1D`)
- In this class, we'll be working with the `X.xmat.1D` file

To Do: Review the Design Matrix

- `ExamineXmat` reads in the generated `*.xmat.1D` file to visualize the time series of the design matrix
- The generated X.jpg depicts a graph of the design matrix
  - You can recreate this using `1dgrayplot` to read in the `*.xmat.1D` file



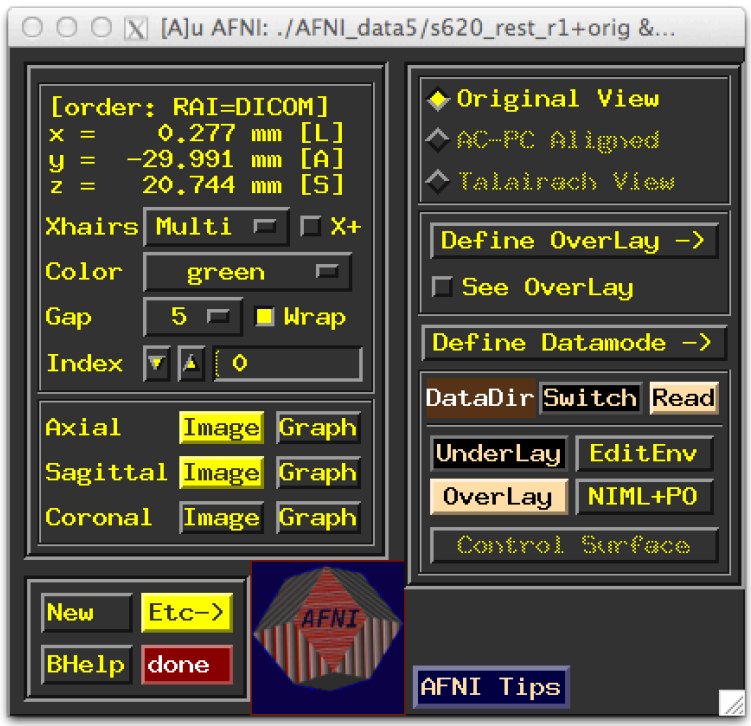


Pulling First-Level β Coefficients

- We need to carry forward the beta coefficients from the first- to second-level
  - We'll need their index in the created stats file to do that
- To get the correct index, we can view our first-level results in AFNI

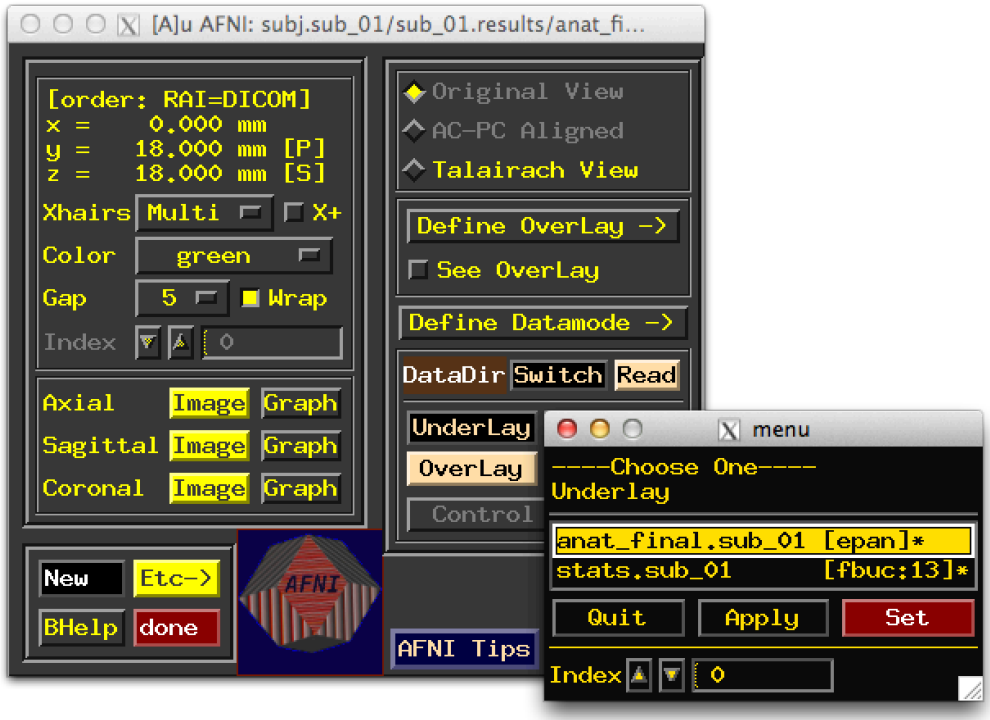
To Do: View First-Level Results

- Navigate to the output subject results directory (e.g., `cd subj.sub_01/sub_01.results/`)
- Open `afni`
  - You can also change the directory within AFNI using the 'Read' button



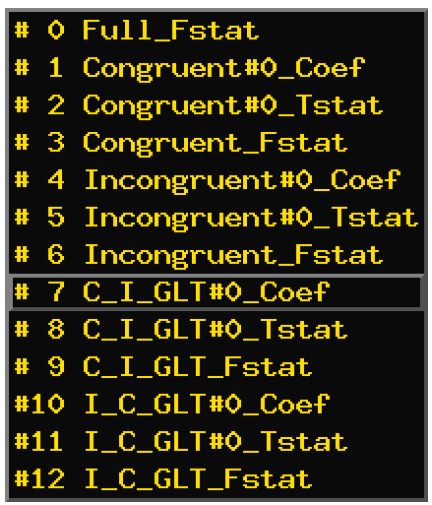
To Do: View First-Level Results, *cont.*

- Once in AFNI, you can load the processed subject anatomical as an underlay
  (e.g., `anat_final.sub_01`)
- Then load the first-level results as an overlay
  (e.g., `stats.sub_01`)



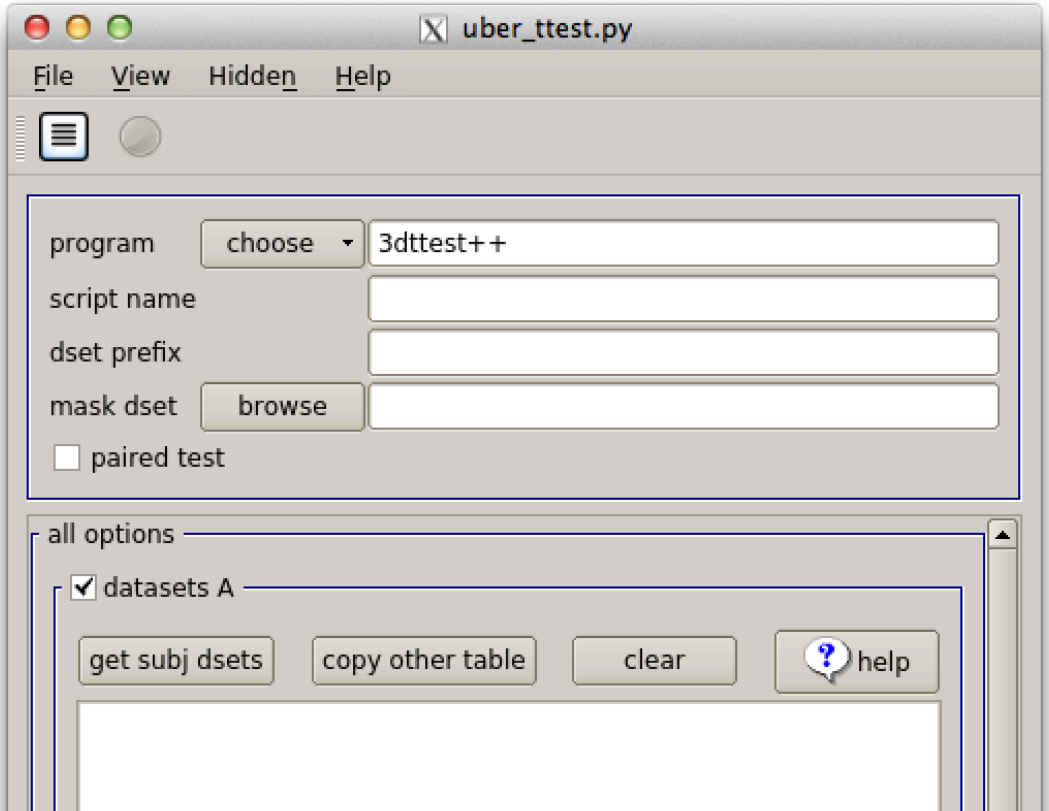
To Do: View First-Level Results, *cont.*

- Click 'Define Overlay' to open the overlay menu
- Hover over 'Olay;' it should read 'Choose overlay sub-brick'
- Clicking on 'Olay' will bring up a menu with the beta coefficients we created
  - **Note the index** for the two GLTs β coefficients; in the window below, they are 7 and 10



Second-Level Analysis

- Once we're confident at the first-level, we can move forward to second-level analysis
- AFNI also provides a GUI to do this!
  - `uber_ttest.py`

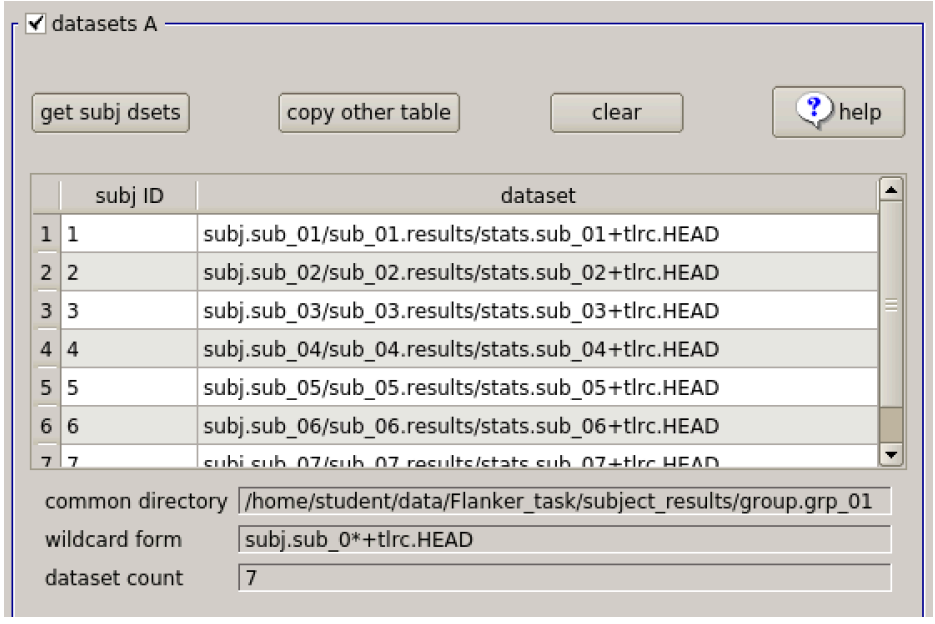


To Do: Specify Second-Level

- For each GLT, we need to conduct a second-level analysis
- In `uber_ttest.py`, the relevant parameters will be:
  - 'dset prefix': the name of the GLT
  - 'subj dsets': the subject-specific stats files
  - 'group or class name': we'll set as 'subjs'
  - 'data index or label': the indices you noted previously

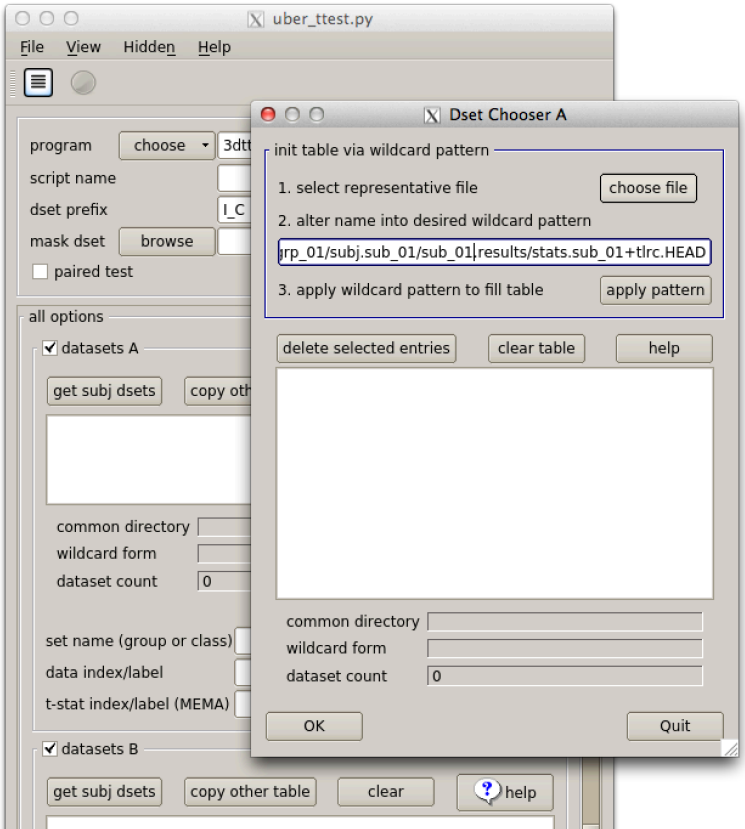
To Do: Specify Second-Level, *cont.*

- We need to populate our 'subj dsets' so they look like the table below (with stat files from all processed subjects)
- To do this, we'll create a wildcard pattern to match our subject-specific stat files



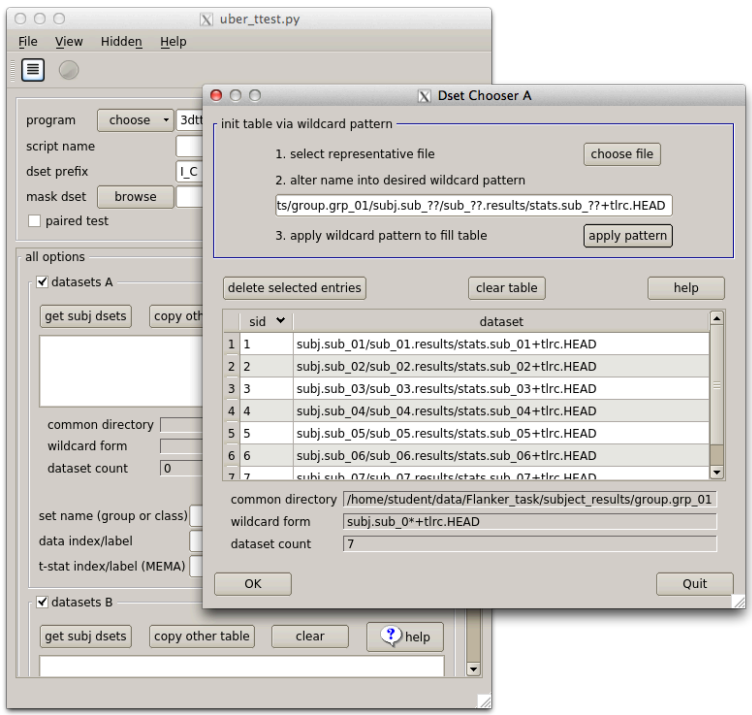
To Do: Create Wildcard Pattern

- Clicking on 'get subj dsets' will bring up a new menu
- There, you can navigate to and select a stats file for one subject
  - In the image below, I've selected a stats file for sub-01



To Do: Create Wildcard Pattern, *cont.*

- Alter the name to be pattern, replacing subject-specific characters with question marks
  - e.g., `sub_01` becomes `sub_??`
- Then click 'apply pattern'
  - This should populate the table as below



To Do: Run `uber_ttest.py`

- Review the generated `tcsh` script to be sure your indices were properly selected
  - If you're satisfied with the parameters, run it by clicking on the green circle as in `uber_subject.py`
- For each GLT you conducted, you'll need to repeat `uber_ttest.py`. For each test, you should edit
  - The 'dset prefix' to the name of the GLT
  - The 'data index or label' to the index you noted in viewing first level results

Viewing Second-Level Results

- Once we've run our second-level analysis using `uber_ttest.py`, we need to view the results
  - This will provide us information on which brain regions are active during each condition (e.g., *Congruent* > *Incongruent*)
- This can be done by navigating to the `group_results` directory and launching `afni`
  - Make sure you've previously copied in the desired template to the results folder(s) for each GLT!

To Do: Load the Results

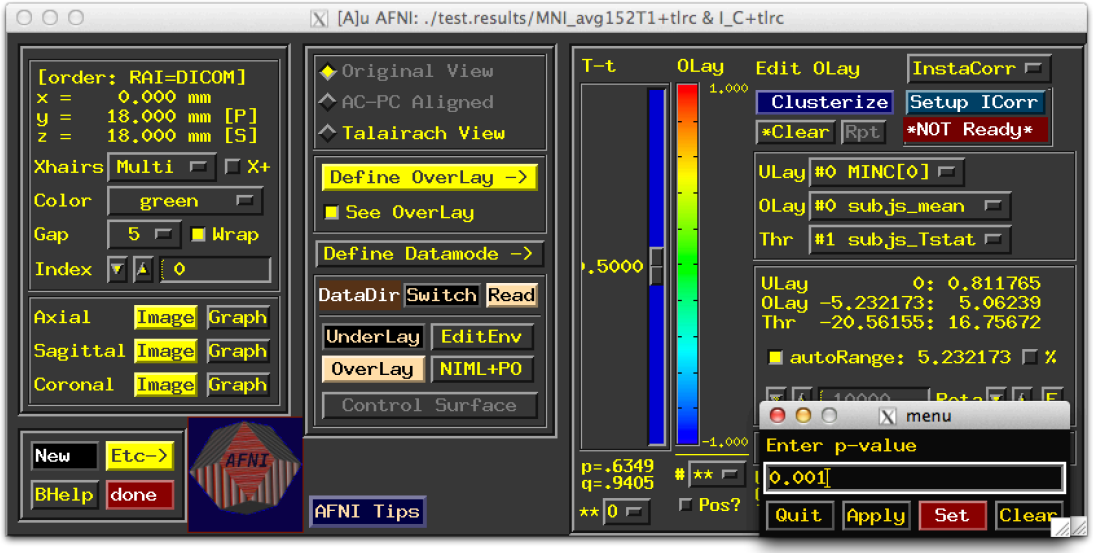
- Load the template as an underlay as discussed in *View First-Level Results*
- Load the GLT second-level results image as an Overlay as discussed in *View First-Level Results*
  - The GLT second-level results will be named the 'dset prefix' you input into `uber_ttest.py`

To Do: Threshold the Results

- Load the template as an underlay as discussed in *View First-Level Results*
- Load the GLT second-level results image as an Overlay as discussed in *View First-Level Results*
  - The GLT second-level results will be named the 'dset prefix' you input into `uber_ttest.py`

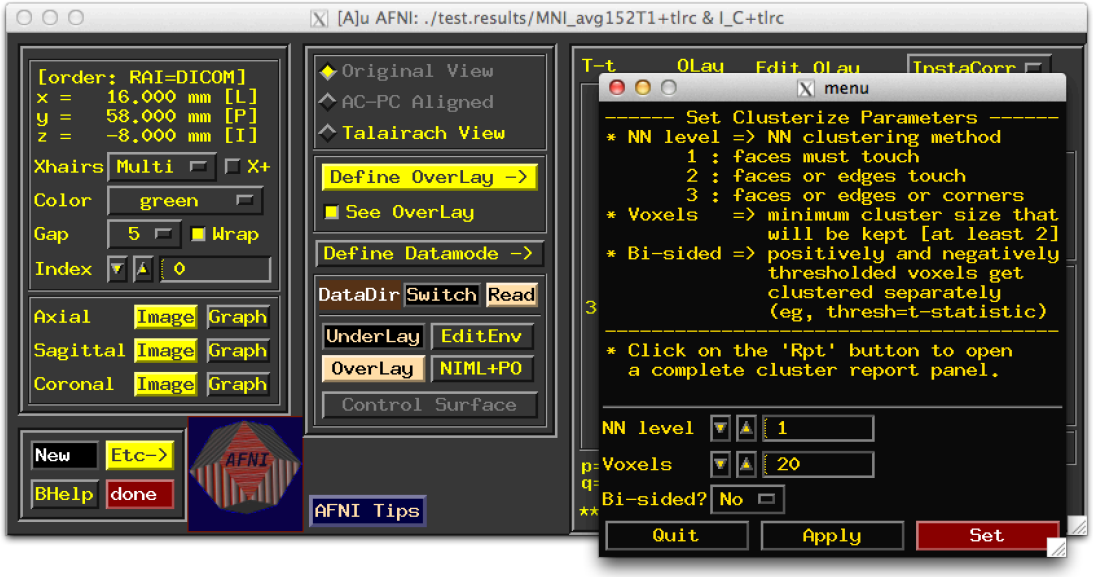
To Do: Threshold the Results, *cont.*

- After loading the image click on 'Define Overlay,' then hover over the listed p-value (below the color bar)
- Right-clicking on the p-value will bring up a menu where you can enter your desired threshold



To Do: Threshold the Results, *cont.*

- We next need to set our cluster-threshold, or the number of contiguous voxels that must be in a cluster
  - This is often reported as *k*
- Clicking on 'Clusterize' will load a menu where we can select our *k* value– the default is 20



Reporting Second-Level Results

- Once we have appropriately thresholded our results, we need to summarize them
  - This is usually done by providing figures and tables

To Do: Report Second-Level Results

- Clicking on the 'Rpt' button under 'Clusterize' will generate a results table
  - This can be used to jump to clusters for creating figures
  - The table itself can also be saved by clicking 'SaveTabl'