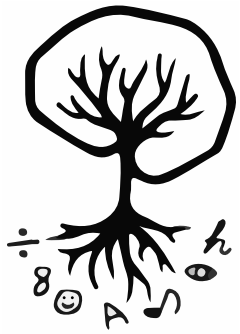

NPDL Functional MRI Analysis Manual

Release 1.0

Connor Lane

July 26, 2016

CHAPTER

ONE

SETUP AND DATA ORGANIZATION

1.1 Connecting to the Arwen server

Before you can start any analyses, you need to be able to connect to the lab's shared analysis server, "Arwen"[1]. Arwen is a Dell Precision T5600 workstation, equipped with 4 8-core Intel Xeon E5 processors, and 64 GB of DDR3 memory. As of 7/2016 it is running a variant of Linux: Ubuntu 12.04 LTS. Attached to Arwen is a Drobo redundant data storage system, with 14TB of usable space. The purpose of Arwen is to support all the analysis needs of lab members. Having a shared analysis workstation facilitates better collaboration and data sharing within the lab.

Arwen is housed in the KSAS server room, and is overseen by PBS IT. Since we do not have physical access to the machine, we must connect to it remotely. The hostname for Arwen is `pbs-mb-arwen.pbs.jhu.edu`. However you will only be able to access this domain from inside the Hopkins network. When working from home, you will need to have the JHU VPN enabled [2].

You can connect to Arwen using either the terminal-based `ssh` [3] utility, or by opening a full remote desktop session using the NoMachine client[4]. In both cases, you will use your JHED ID and password as your login credentials. Importantly though, not everyone with a JHED ID is allowed to login. You must be a part of the `pbs-mb-group` active directory group [5].

To log on using `ssh`, you will need to enter `ssh user@pbs-mb-arwen.pbs.jhu.edu`, where `user` is your username. You will then be prompted to enter your password. By default `ssh` only gives you access to the terminal. However it is possible to "tunnel" Gui applications through `ssh`, using X11 forwarding[6]. To do this, you just add the `-X` flag: `ssh -X user@pbs-mb-arwen.pbs.jhu.edu` (assuming X11 forwarding is enabled).

In order to have full access to a desktop session, you will need to use the NoMachine client to open an NX remote desktop connection. The NoMachine client download can be found here[7]. The steps for setting up a NoMachine connection, and for starting or resuming a desktop session are described in Figures 1.1 and 1.2.

1.2 Exploring the Godzilla server

Once data is collected, it's sent to the KKI Godzilla server for permanent storage. The first step in analysis then is to transfer the data from Godzilla to Arwen. As with Arwen, we will use `ssh` log onto Godzilla and transfer data.

[1] Our old IT person Jesse named it after the mystical elf from Lord of the Rings.
[2] Setting the VPN up can be tricky, so talk to PBS IT. As of 7/16, the Pulse Secure VPN client is required.
[3] If you're unfamiliar with how to use `ssh`, see this tutorial for example: https://mediatemple.net/community/products/dv/204403684/connecting-via-ssh-to-your-server
[4] https://www.nomachine.com/
[5] PBS IT can help you with this as well.
[6] http://unix.stackexchange.com/questions/12755/how-to-forward-x-over-ssh-from-ubuntu-machine
[7] https://www.nomachine.com/download

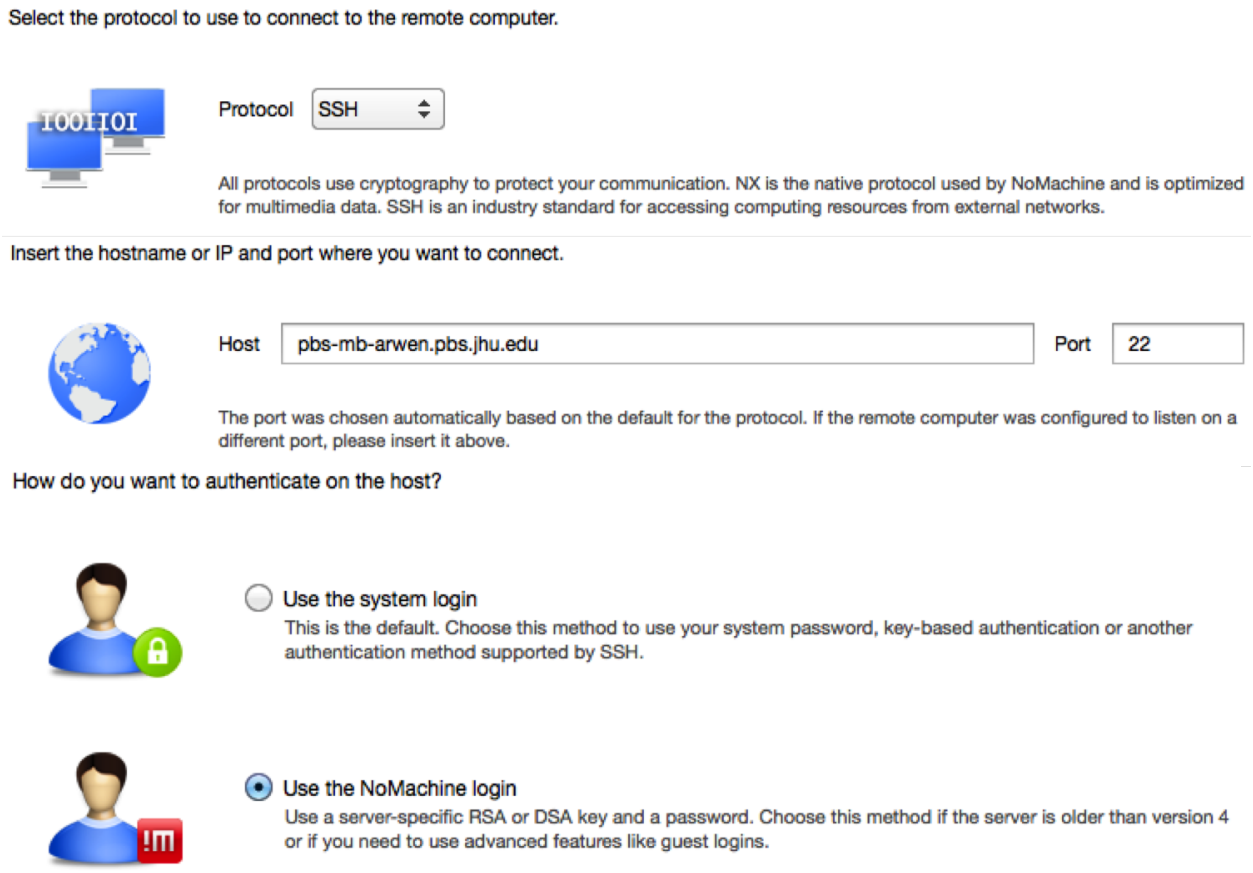

Figure 1.1: Once you've installed the NoMachine client, you need to set up a new connection to the Arwen server. You should select "SSH" rather than "NX" as the protocol in the first dialog. Next, you'll enter the Host: pbs-mb-arwen.pbs.jhu.edu. Last, make sure to tick the "Use the NoMachine login" option. (*Aside*: We use FreeNX on the server-side to handle NX connections. FreeNX is an open-source fork of NoMachine, built of their version 3.5.0 libraries. As a result, we have to use this backwards compatible login option, rather than the default.) For the rest of the settings, the default options are fine.

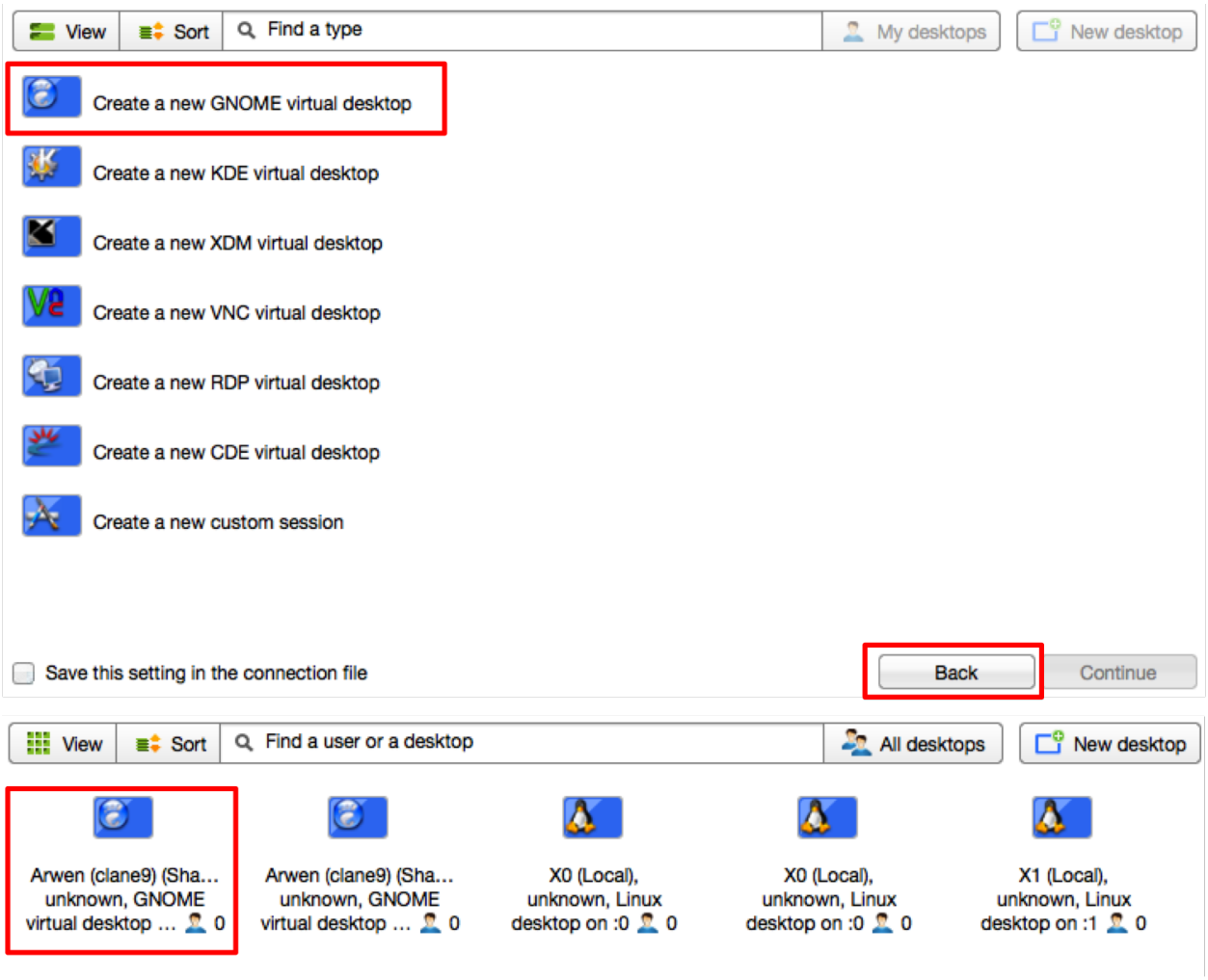

Figure 1.2: Once you've set up the connection, you should try logging in. After you've entered your user name and password, you'll be prompted to create a new virtual desktop. Because the server is running Ubuntu, you'll need to select the "GNOME" option. However, if you do not want to start a new desktop, but instead want to rejoin a previous session, you can select the "Back" button. This will bring you to a list of open sessions to choose from.

To log onto the Godzilla, you need to contact system administrator at KKI and request an account. Note that this is different than the scheduling account you use to book MRI sessions on the scanner.

To simplify the `ssh` log-in process, you can set up public key authentication. For a description of what this is and how to do it, see this tutorial[8].

Once you're logged in, navigate to `/g4/mbedny`. This is our lab folder on Godzilla. All of our studies are stored in this folder (e.g. BSYN, BRAILLE). Within each study folder, there will be a folder for each subject. And within each subject's folder, a par and rec file for each collected run, plus an MR folder, containing DICOM images for the MPRAGE scan:

```
/g4/mbedby/
  |- BSYN/
    |- BSYN_S_01/
      |- bsyn_s_01_3_1.par
      |- bsyn_s_01_3_1.rec
      |- bsyn_s_01_4_1.par
      |- bsyn_s_01_4_1.rec
    |- BSYN_S_02/
  |- BRAILLE1/
    |- BRAILLE1_CB_01
      |- MR/
        |- 1.3.46.670589.11.24058.5.0.1788.2014053014171806001
        |- 1.3.46.670589.11.24058.5.0.1788.2014053014171865002
      |- braille1_cb_01_3_1.par
      |- braille1_cb_01_3_1.rec
```

Typically, runs 1, 2 are the survey and reference scans respectively [9]. These scans are not usually sent to Godzilla during data collection. The first scan we keep is usually run 3, the MPRAGE.

1.3 The scan log

For each scanning session you must keep a scan log documenting the events of the session. The scan log is how we associate scanner runs with behavioral files. Without the scan logs we could not run any analyses.

You should keep both a written scan log and an electronic version. The format for the electronic version is as follows:

```
# Study: BSYN
# Subject ID: BSYN_S_01
# Scanner ID: BSYN_04
# Registration ID: 1403250900
# Date: 3/25/14 9:00
# Scanner: MR1 32ch
# Scanned by: TB

3 mprage
4 bsyn_01
5 bsyn_02
6 bsyn_03
7 bsyn_04 # participant got out to use the restroom.
8 bsyn_05
9 bsyn_06
```

[8] https://macnugget.org/projects/publickeys/

[9] In spring, 2016 the scanner software was updated so that the reference is built into the survey. Therefore, scan 2 is now often the MPRAGE.

```
# Notes:
# - Volume set to 2.5
# - Good performance on average
# - Subject a little claustrophobic for the first scan
```

The scan log starts with a header containing info about the scan session: the study name, the subject ID, the scanner subject ID (which may or may not be different), etc.

- Each line of the header must start with "#".
- The "key" for each line must be spelled exactly as shown.
- The keys must be separated from their values with a colon.

The next section of the scan log is a two-column matrix consisting of run number, run name pairs. The run number is the number of the scan, as it was collected. The run name typically has the format `{task}_{run num}`, with the run number being zero-padded to two places.

Last, there is an optional *Notes* section, where you can record miscellaneous information about the session or the participant. Each line of notes should start with "#".

1.4 Transferring data for analysis

Transferring data is a three part process:

1. Fetch the par and rec files from Godzilla.
2. Convert the par/recs to gzipped Nifti files.
3. Rename the converted files to something more convenient than the default scanner names.

All of these steps are accomplished with the `parfetch` command, which is part of the lab's suite of scripts

```
Usage: parfetch [options] <scan-log>

Fetch par and rec files from the scanner file server and convert
to gzipped nifti. File organization on the server is assumed to
follow the convention:

  {lab dir}/{study dir}/{subject ID}/*_{run #}_{acq #}.*

Arguments:
  <scan-log>   Scan log text file. Describes how files should be
               renamed. First column is run number, second column
               is new name. You may optionally specify the Study,
               Subject ID, and/or Scanner ID in a comment line
               (starting with #). All other lines starting with #
               will be ignored. See below for an example of the
               proper format.

Options:
  --study <study>     Name of study on server. Read from the scan log
                      by default (needs a '# Study: XXXX' line).
  --sub <scan-sub>    Scanner subject ID on server. Read from the scan
                      log by default (needs a '# Scanner ID: XXXX' line).
  --out <outdir>      Directory to put converted data. If this option
                      is not specified, the converted data will be placed
                      in {subject ID}/raw, in the working directory,
                      where {subject ID} is read from the scan log.
```

```
  --u <user>          Name of server user [default: clane9].
  --labdir <dir>      Lab directory on server [default: /g4/mbedny].
  --no-clean          Don't delete redundant rec files.
```

First `parfetch` reads the scan log for the scan session to determine where the data is located on Godzilla, and where it should be placed on the lab server. It uses the "Study" and "Scanner ID" values to determine where the data is located, and it uses the "Subject ID" value to decide where to put the data (defaulting to `{Subject ID}/raw` in the working directory.

Next, the command transfers the data to the lab server using the `scp` command. For this part to work it is essential that you can access Godzilla. And if you have public-key authentication set up, you won't have to enter your password. Next, `parfetch` uses `dcm2nii` to convert the data to gzipped Nifti files. See the Mricron[10] site for details on this part. Last, `parfetch` renames the converted files according to the names given in the second column in the scan log.

If we were to run `parfetch` on the example scan log above, the resulting raw folder would be structured like this:

```
BSYN_S_01/
  |- raw/
    |- bsyn_01.nii.gz
    |- bsyn_02.nii.gz
    |- mprage.nii.gz
    |- par/
      |- bsyn_04_3_1.par
      |- bsyn_04_4_1.par
      |- MR/
    |- parfetch.log
    |- sl.txt
```

[10] http://www.mccauslandcenter.sc.edu/mricro/mricron/dcm2nii.html

CHAPTER

TWO

SURFACE RECONSTRUCTION

2.1 Surfaces

Most of our analyses are performed on data that have been mapped to the cortical surface. There are two kinds of files used to represent surface-based data: anatomical surfaces, and metric overlays. Anatomical surfaces are like high-resolution T1 images. They represent the size and shape of the cortical sheet. Anatomical surfaces are geometrical structures made up of triangular faces that are "glued" together along the edges to form a closed "mesh". The points of the triangular faces are called vertices. Vertices are the surface analogues to voxels. They are the units of analysis.

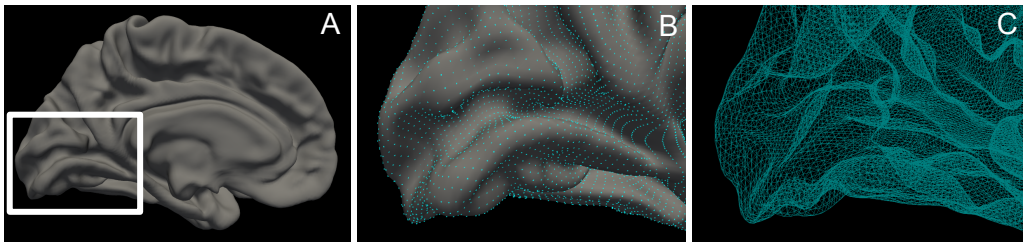

Figure 2.1: Renderings of an anatomical surface. **(A)** The entire surface shown from the medial view. V1 is boxed. **(B)** A close-up of V1, with vertices shown in teal. **(C)** Another close-up of V1, this time showing the mesh structure.

Metric overlays are used to represent assignments of data to the vertices of a surface. The numerical data assigned to each vertex can either be a vector, like a functional time series, or a single scalar, such as a z or t value. Metrics are also commonly used to represent ROIs, by assigning vertices within the ROI the value 1, and all other vertices 0. One important thing to remember is that metrics contain no anatomical structure. This is unlike volume-based images, where the spatial relationships between data points are encoded in the file format itself.

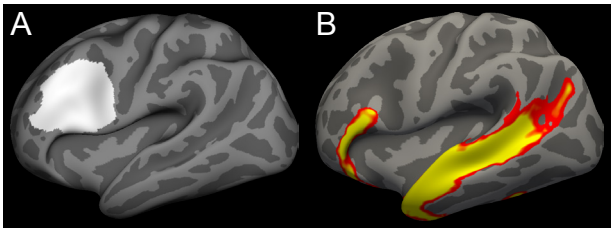

Figure 2.2: Examples of metric overlays. **(A)** An ROI in inferior frontal cortex, represented with 1's and 0's. **(B)** A z-stat image for a language contrast.

2.2 Introduction to Freesurfer

All of our surface-based analyses depend on the Freesurfer software package. Freesurfer consists of a set of command-line tools for generating and manipulating surface files. Generating anatomical surfaces from T1 MR images is the most important thing we use Freesurfer for. The command for that is called `recon-all`. Some other important uses of Freesurfer include:

- Volume and surface visualization (freeview[1], tksurfer[2])
- Volume and surface file conversion (`mri_convert`, `mris_convert`)
- Surface-constrained volume registration (`bbregister`)
- Surface resampling and smoothing (`mri_surf2surf`)
- Surface-based group fMRI statsistics (`mri_glmfit`)

Freesurfer uses a centralized workspace called the *Subjects Directory* for carrying out most of its processing. The location of the Subjects Directory is determined by the shell environment variable `SUBJECTS_DIR`. This variable is set automatically whenever you start a terminal session. To change the Subjects Directory, you need to re-define the `SUBJECTS_DIR` variable in the terminal. E.g:

```
cd /media/BednyDrobo/Projects/BSYN/SurfAnat
SUBJECTS_DIR=$(pwd)
```

The Subjects Directory contains many individual *subject folders*. These folders are generated when you run `recon-all`. They contain all of the output surface files and intermediate volume files created by `recon-all`. For example, the default Freesurfer Subjects Directory (`$FREESURFER_HOME/subjects`), contains a subject called "bert". bert's subject folder is organized as follows:

```
bert/
  |- bem/
  |- label/
  |- mri/
    |- orig.mgz
      * Raw T1 image.
    |- T1.mgz
      * Intensity-normalized T1 image.
    |- brain.mgz
      * Intensity normalized and brain-extracted T1 image.
  |- scripts/
    |- recon-all.log
      * Complete recon-all log.
    |- recon-all-status.log
      * Summary log, showing completion time ofeach reconstruction step.
  |- src/
  |- stats/
  |- surf/
    |- lh.pial
      * Left-hemisphere pial surface mesh (outer gray matter boundary).
    |- rh.pial
    |- lh.white
      * Left-hemisphere white surface mesh (inner gray matter boundary).
    |- rh.white
    |- lh.sphere.reg
      * Left-hemisphere spherical surface after alignment with fsaverage.
    |- rh.sphere.reg
```

[1] http://freesurfer.net/fswiki/FreeviewGuide
[2] https://surfer.nmr.mgh.harvard.edu/fswiki/tksurfer

```
|- tmp/
|- touch/
|- trash/
```

The mri and surf folders contain the output volume and surface files respectively. For example, T1.mgz is a copy of the raw T1 volume, after intensity bias correction has been applied. lh.white is an anatomical surface that follows the gray/white matter boundary in the left hemisphere.

Note: Freesurfer uses a custom file format to represent MRI volumes and surfaces. These files usually have the extension `.mgh` or `.mgz` (compressed). You can convert these files to more standard formats like Nifti and Gifti using `mri_convert` and `mris_convert` respectively.

A good way to get familiar with all of the Freesurfer output files is by following the Freesurfer output inspection tutorial[3], either with the tutorial data or your own. This is also a good way to get familiar with the primary Freesurfer visualization tool, freeview[4].

2.2.1 Additional Freesurfer resources

- The Freesurfer course materials: https://surfer.nmr.mgh.harvard.edu/fswiki/FsTutorial
- The Freesurfer youtube channel: https://www.youtube.com/channel/UCruQerP8aa-gYttXkAcyveA
  - In particular, the smoothing and registration talk, which is all about the benefits of surface-based analysis: https://www.youtube.com/watch?v=8WPvXoORoAw

2.3 Running `recon-all`

Freesurfer's entire surface reconstruction pipeline is completely automated. It only takes one command to get the process going, e.g.:

```
recon-all -subject BSYN_S_01 -i mprage.nii.gz -all
```

This command will create a subject folder called BSYN_S_01 inside the Subjects Directory and begin populating it with BSYN_S_01's anatomical surfaces. The only input is mprage.nii.gz, the high resolution T1 volume. There are 34 steps in the entire processing stream, starting with intensity normalization and skull stripping, and going through surface creation, surface inflation, and surface-based registration to the Freesurfer average space, *fsaverage*. Altogether it can take up to 10 hours to complete (on our machine it's usually about 6 hours). The `-all` option tells recon-all to do all 34 steps.

You can view the complete description of all processing steps either by looking at the `recon-all` help message, or by navigating to the online Freesurfer analysis pipeline overview[5]. If you're looking for more detail, try the recon-all table[6] which contains a complete outline of all the reconstruction steps, and the sub-commands involved.

2.4 Troubleshooting `recon-all`

99% of the time `recon-all` completes successfully. However there are a few things that will systematically cause `recon-all` to crash.

[3] http://freesurfer.net/fswiki/FsTutorial/OutputData_freeview
[4] http://freesurfer.net/fswiki/FreeviewGuide
[5] http://freesurfer.net/fswiki/FreeSurferAnalysisPipelineOverview
[6] http://freesurfer.net/fswiki/ReconAllTableStableV5.3

1. **The Subjects Directory is set incorrectly.** Before running `recon-all`, be sure to set the `SUBJECTS_DIR` environment variable to your project-specific Subjects Directory. Otherwise `recon-all` will try to write to the default Subjects Directory, which will cause a read-only permissions error.
2. **The orientation information in your anatomical image is incorrect.** Load your anatomical image in `freeview` or some other viewer. If one of the coordinate dimensions (anterior, posterior, superior, inferior) is labeled incorrectly, this will cause `recon-all` to fail dramatically.
3. **Too large of a FOV.** Some of the `recon-all` steps (e.g. volume-based registration) occasionally fail if your T1 image was acquired with an especially large field of view. In these cases it can help to first crop your T1 image using a tool such as FSL's `robustfov`.

2.5 Inspecting outputs

The critical outputs of the Freesurfer reconstruction process are four surfaces (two for each hemisphere). The left and right "white" surfaces mark the inner gray/white matter boundary. The left and right "pial" surfaces mark the outer dura/gray matter boundary. The primary goal of data inspection is to check that the volume between these surfaces includes all and only gray matter.

We visually inspect these surfaces overlaid onto the high-res anatomical image. To do this, we load the surfaces and the high-res anatomical into the Freesurfer viewer, freeview[7]. Freeview can be opened at the terminal by typing `freeview`. You can then use the GUI interface to load the surfaces and T1 image.

Alternatively, you can specify the images as command-line arguments to the `freeview` command.

To make loading images easier, we use the command `checksurf`, which only requires a subject ID argument (provided the `SUBJECTS_DIR` is set properly):

```
checksurf BSYN_S_01
```

All `checksurf` does is encapsulate what would be a very long `freeview` call: loading the original T1, the brain-extracted T1, the left and right white and pial surfaces (with gyral parcellations), as well as the left and right inflated surfaces:

```
# Change to subject's directory.
cd $SUBJECTS_DIR/BSYN_S_01

# Load volumes and surfaces.
freeview \
  -v \
  mri/brainmask.mgz \
  mri/orig.mgz:visible=0 \
  -f \
  surf/lh.white:edgecolor=blue:edgethickness=2 \
  surf/rh.white:edgecolor=blue:edgethickness=2 \
  surf/lh.pial:edgecolor=green:edgethickness=2:annot=label/lh.aparc.annot \
  surf/rh.pial:edgecolor=green:edgethickness=2:annot=label/lh.aparc.annot \
  surf/lh.inflated:edgethickness=0:visible=0 \
  surf/rh.inflated:edgethickness=0:visible=0
```

To check the surface boundaries, you will want to scroll through each slice of the image, looking for places where too much or too little has been included as "gray matter". You should pick either the sagittal, coronal, or axial view from the top menu bar, and work your way through the entire image. You can scroll the slices using Page Up/Page Down (Fn+Up Arrow and Fn+Down Arrow on a Mac). Some other useful shortcuts are:

[7] http://freesurfer.net/fswiki/FreeviewGuide

- Shift+Left click drag: Control brightness/contrast of the anatomical image
- Mouse scroll: Zoom in/out
- Ctrl+Left click: Zoom in at cursor
- Ctrl+Right click: Zoom out at cursor
- Left/Right/Up/Down Arrow: Move FOV

These shortcuts and others can also be found in Help –> Quick Reference in the Freeview window.

Some general pointers for checking surface accuracy:

- I prefer scrolling through the coronal view, since it's a smaller cross-sectional area to look at. I've also found that the frontal and temporal lobes exhibit different kinds of errors. By scrolling through from anterior to posterior, you can concentrate on one sort of error at a time.
- About 1-3 seconds per slice tends to be a good pace.
- When you see something that doesn't look right, try clicking on the spot and changing views. Getting a different perspective usually helps you decide if what you're seeing is really an error.
- It also helps to pass over a problem area a few times, scrolling back and forth. Getting a better idea of the context around a potential reconstruction error will usually help you decide what to do about it.

2.6 Common reconstruction errors

Most of the time the surfaces produced by Freesurfer are just fine and don't need any intervention (say ~50% of subjects). When there are surface problems, they'll usually fall into one of the following categories.

Note: Many of these errors are covered in the Freesurfer failure modes talk[8], and the fixes are described in the Freesurfer troubleshooting tutorial[9].

2.6.1 Bad white matter segmentation

The segmentation of white matter from gray matter relies on white matter voxels having a consistent intensity. When you get a region of white matter that is especially bright or dark, this can cause Freesurfer to mislabel it. You'll often see this problem in the long, thin gyri located in anterior temporal lobe.

To fix this kind of error, you'll need to place some *control points* on or around the mislabeled voxels, and re-run `recon-all`. For a full description of the steps to take, see the Freesurfer control points tutorial[10].

Note: This type of error is often visible in several adjacent slices. If you see what looks like a white matter segmentation error in one isolated slice, you should probably leave it be. This is most likely just Freesurfer making a difficult partial-volume decision.

2.6.2 Skull segmented as gray matter

The skull stripping step can sometimes fail to remove parts of the skull or dura, especially around the eyes, around orbitofrontal cortex, and near the top of the head. This extra skull can sometimes get mislabeled as

[8] http://surfer.nmr.mgh.harvard.edu/pub/docs/freesurfer_failuremodes_ani.ppt
[9] https://surfer.nmr.mgh.harvard.edu/fswiki/FsTutorial/TroubleshootingData
[10] https://surfer.nmr.mgh.harvard.edu/fswiki/FsTutorial/ControlPoints_freeview

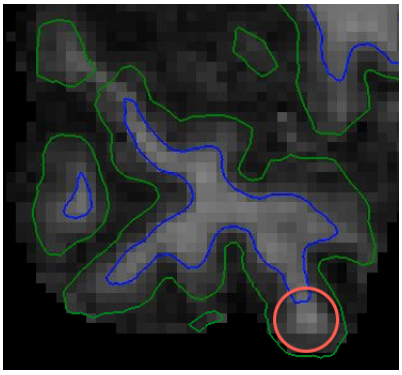

Figure 2.3: Example of bad white matter segmentation. A cluster of white matter voxels are classified as gray matter. It is difficult to see here, but the circled white matter voxels have intensity values well below the expected value, 110.

gray matter, which you'll have to fix by erasing the voxels manually. For a full description of the steps to take, see the Freesurfer skull-strip fix tutorial[11].

Note: Most of the time skull strip errors won't cause surface errors. No need to do anything in this case. The surfaces are all we care about.

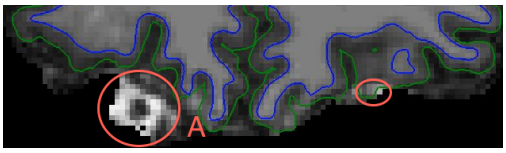

Figure 2.4: Skull strip errors that don't affect the cortical surface.

2.6.3 White Matter Lesions

We sometimes see subjects with small lesions scattered in their white matter. These lesions look like dark spots in the white matter volume. They are more common in older subjects, and usually having one means having many. When these lesions occur near the gray/white matter boundary, they can cause the white surface to errantly "dip" into (what should be) white matter.

Adding control points won't help in this situation. You can only use control points to correct intensity issues due to scanner bias. Putting a control point on a voxel tells Freesurfer "This voxel is white matter, so you should scale it's intensity (and that of neighboring voxels) to match other white matter voxels." Since lesion voxels don't actually contain white matter, adding control points here is the wrong choice. Instead, you will have to manually fill in the lesion area in the subject's wm.mgz volume. For complete instructions, see the Freesurfer white-matter edit tutorial[12].

2.6.4 Midline weirdness

The paths that the surfaces take through the midline structures and the corpus callosum are usually pretty nonsensical. **This is to be expected.** Don't worry about it. The lines here are more or less arbitrary, since there is no cortex to follow.

[11] https://surfer.nmr.mgh.harvard.edu/fswiki/FsTutorial/SkullStripFix_freeview
[12] https://surfer.nmr.mgh.harvard.edu/fswiki/FsTutorial/WhiteMatterEdits_freeview

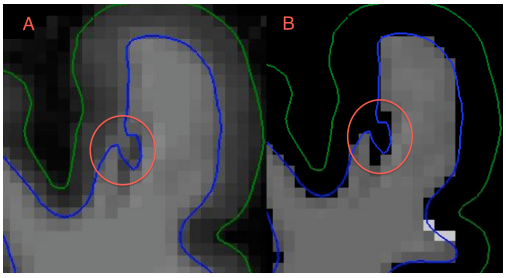

Figure 2.5: A white matter lesion causing a reconstruction error. **(A)** The error shown on the brainmask.mgz volume. **(B)** The error shown on the wm.mgz volume. Note how implausible the surface curvature is here. Also, see that the error is more obvious in **(B)** than in **(A)**.

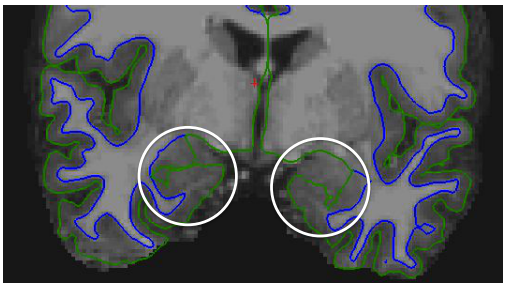

Figure 2.6: Uninterpretable pial surface through midline subcortical structures.

2.6.5 Deciding when to correct an error

Reconstruction errors come in different sizes. Some require correction before analysis can proceed, while others are insignificant. When deciding whether to take action, keep in mind how many voxels are affected by the error. If several tens or even hundreds of voxels are affected, you should absolutely correct the error. Otherwise, use your judgement. When in doubt, you should be conservative in taking action. Freesurfer makes principled, objective decisions about the surface boundaries. Human observers sometimes don't. **Above all, be consistent across your study sample.**

2.7 Quality control record keeping

It's very important to take notes while inspecting your data. This way you can return months later and feel confident that the quality assurance was done right. As a lab, we try to stick to a common format for quality control record keeping.

The quality control spreadsheet, usually called *recon_QC.xls* has two tabs. The first tab, called *Recon check*, is for keeping track of who checked which subjects when, and whether the reconstruction was ok, or required fixing.

Subject	Recon OK	Date	Checked by
BSYN_CB_02	yes	8/4/15	CL
BSYN_S_02	no	8/4/15	CL

The second tab, called *Recon errors*, is for keeping track of errors that you find in the surface reconstruction. Each row corresponds to an error. In the columns, you record the subject that you found the error in, a description of the error, the voxel coordinates for the error, what fix you applied (if any) and whether the fix worked. Keep in mind that this sheet is both for errors that need intervention, and for problem spots you

just want to note, but don't need any further action. If you're not sure how to describe an error you see, or how to fix it, you should still take down at least the voxel coordinates. This way you can return to it later.

Subject	Error	Coordinates	Fix	Fixed
BSYN_CB_02	Questionable partial-volume decision	87, 153, 87	N/A	N/A
BSYN_S_02	Skull included around right eye	154, 144, 175	N/A	N/A
BSYN_S_02	Skull included in right orbital frontal	160, 151, 156	Removed extra skull	yes

2.8 Post reconstruction processing

After Freesurfer reconstruction is complete, some additional surface processing is required before you can move on to later analysis steps. These steps are carried out using the `postrecon` script, and include:

- Nonlinear registration of the subject's T1.mgz with the MNI 152 template using FSL's `fsl_anat`.
- Converting primary surfaces to Gifti format using `mris_convert`.
- Constructing HCP-style midthickness and inflated surfaces using `wb_command -surface-average` and `wb_command -surface-generate-inflated`.
- Resampling Gifti surfaces into the HCP group average space, "32k_fs_LR", using `wb_command -surface-resample`.

Running `postrecon` only requires a subject ID argument, e.g.:

```
postrecon BSYN_S_01
```

Since you need to run `recon-all` and `postrecon` in sequence, it's convenient to run them in one command line:

```
recon-all -subject BSYN_S_01 -i mprage.nii.gz -all && postrecon BSYN_S_01
```

`postrecon` modifies subject folders in place. The outputs of the nonlinear registration with MNI 152 are placed under mri/T1.anat, while the generated Gifti surfaces are saved in the subject's surf folder. Some of the key outputs include:

```
BSYN_S_01/
  |- mri/T1.anat/
    |- T1_to_MNI_nonlin.nii.gz
      * T1 image resampled into MNI space (2mm^2 resolution).
    |- T1_to_MNI_coeff.nii.gz
      * T1 to MNI Warp coeffiecients file.
    |- T1_biascorr_brain_mask.nii.gz
      * brain mask after MNI 152 informed brain extraction.
  |- surf/
    |- ?h.white.surf.gii
      * subject's white surface converted to Gifti format.
    |- ?h.pial.surf.gii
      * subject's pial surface converted to Gifti format.
    |- ?h.midthickness.surf.gii
      * average of white and pial surfaces, corresponding to the middle
        cortical thickness.
    |- ?h.32k_fs_LR.midthickness.surf.gii
      * midthickness surface, resampled into 32k_fs_LR space.
```

After running `postrecon`, you should check that the nonlinear registration with MNI 152 is accurate. One way to do this is to load the resampled T1 image "T1_to_MNI_nonlin.nii.gz" into Freeview, along with the MNI 152 template. The MNI template is packaged with FSL. For example, you might run.

```
freeview -v BSYN_S_01/mri/T1.anat/T1_to_MNI_nonlin.nii.gz \
  $FSLDIR/data/standard/MNI_152_2mm_brain.nii.gz
```

You can gauge the degree of overlap between the two images by adjusting the opacity slider for one of the images.

2.9 A note on surface spaces

A surface *space* or *mesh* is defined by a set of vertices and their connections. The same surface mesh can be used to represent different anatomical surfaces–white, pial, inflated, etc.–only by changing the (X, Y, Z) coordinates of the vertices.

During reconstruction, a subject-specific surface mesh is generated, along with a range of anatomical surfaces. This "native" mesh typically contains over 150,000 vertices, which roughly amounts to 0.8mm^2 per vertex. The high resolution of the native mesh is part of why Freesurfer's surfaces are so accurate. However, such a high resolution space is innapropriate for fMRI analysis, since fMRI data is acquired at a much lower resolution (e.g. 3mm ^{3} isotropic voxels). This is the primary reason for introducing the "32k_fs_LR" space during `postrecon` processing. The 32k_fs_LR mesh contains only 32,492 vertices, which amounts to 3.75mm^2 per vertex.

During `postrecon` The subject's native surfaces are resampled onto the 32k_fs_LR mesh. The resampled surfaces still represent the subject's specific anatomy, albeit at a resolution more comparable to that of fMRI data. These resampled surfaces then serve as the targets in later preprocessing for projecting volumetric data to the surface.

An important additional feature of the 32k_fs_LR space is *vertex-wise anatomical correspondence*. This means that once surfaces are resampled to 32k_fs_LR, vertex *N* will correspond to the same anatomical location in all subjects. Importantly this is still true even though each subject has a slightly different surface anatomy.

This kind of surface-based alignment is notably different than volume-based alignment. In the volume regime, you cannot put two MR images into voxel-wise correspondence without disrupting the geometry of one of them. This is because a voxel's index just *is* its spatial location. By contrast, the number or index of a surface vertex and its spatial position are totally independent. This is why we can have aligned surfaces that retain a subject's specific anatomy.

Lastly, not only are surfaces aligned across subjects in 32k_fs_LR space, they are also aligned across hemispheres. This means that within a single subject vertex *N* will fall in contra-lateral positions in the left and right hemispheres.

Note: Try to convince yourself of the vertex-wise correspondence by loading two subject's lh.32k_fs_LR.midthicknes.surf.gii surfaces in Freeview and looking up different vertices. You can also test the hemisphere correspondence by loading both of a subject's ?h.32k_fs_LR.midthickness.surf.gii surfaces.

CHAPTER

THREE

FUNCTIONAL PREPROCESSING

3.1 Preprocessing overview

Functional MRI data must be preprocessed before you can run any statistical analyses. fMRI data are often corrupted by nuisance factors, like subject head motion and low-frequency scanner noise. It is critical to remove the effects of these factors before running any statistical analyses.

Our preprocessing pipeline is built on the publicly available toolboxes from FSL, Freesurfer, and the Human Connectome Project (HCP). It is partly based on the "minimal preprocessing" pipeline used in the HCP [Glasser2013]. We have broken the pipeline into 7 stages:

1. Volume-based preprocessing using FSL's FEAT tool[1].
   (a) Motion correction [Jenkinson2002].
   (b) Slice-timing correction *[optional]*.
   (c) Grand-mean intensity scaling (the entire 4D data series is scaled so that the mean whole-brain intensity across time is 10,000).
2. Surface-constrained registration of functional and anatomical MR volumes, using Freesurfer's `bbregister` [Greve2009].
3. ICA with automatic artifact classification.
   (a) Using FSL's FIX[2] (more general, but very sensitive to acquisition parameters; requires hand-training) *[optional]*.
   (b) Using FSL's AROMA[3] (more robust than FIX, but only detects motion artifacts) *[optional]*.
4. Denoising.
   (a) ICA artifact removal *[optional]*.
   (b) Band-pass filtering *[high-pass filtering recommended; low-pass optional]*.
   (c) White-matter/CSF signal removal *[regressors saved by default; removal optional]*.
5. Motion spike detection, following [Power2012].
   (a) Finding time-points with relative frame-wise displacement (FDRMS) greater than some threshold (e.g. 1.5mm) [Jenkinson1999].

[1] http://fsl.fmrib.ox.ac.uk/fsl/fslwiki/FEAT
[2] http://fsl.fmrib.ox.ac.uk/fsl/fslwiki/FIX
[3] https://github.com/rhr-pruim/ICA-AROMA

   (b) Finding outlier time-points with respect to the DVARS metric [Power2012].

6. Mapping data to the cortical surface, using the HCP Workbench[4].
   (a) Resampling volumetric data onto the subject's native high-resolution surface, using the HCP's ribbon-constrained volume-to-surface mapping algorithm.
   (b) Downsampling surface data to the lower resolution `32k_fs_LR` surface mesh, which is more appropriate for fMRI data.
   (c) 2mm FWHMM surface-based smoothing to regularize data on the surface.
   (d) Filling in missing data on the surface with nearby values *[optional]*.
7. Quality-assurance reporting.

3.2 Basic `preproc` usage

The tool we use for carrying out this procedure is called `preproc`. In most circumstances, you will only need to run `preproc` with the default options enabled. In this case, the basic usage pattern is:

```
preproc <subject> <input-run> <outdir>
```

where `<subject>` is the subject ID, `<input-run>` is a path to the raw Nifti functional data, and `<outdir>` is the output directory. The subject ID must match a surface reconstruction folder in the `SUBJECTS_DIR`, the input run must be a valid 4D Nifti file, and the output directory should not already exist. For example, you might run something like:

```
preproc BSYN_S_01 BSYN_S_01/raw/bsyn_01.nii.gz BSYN_S_01/preproc/bsyn_01
```

By default, all optional processing steps are disabled. The data will first be motion-corrected and grand-mean scaled. Next, they are high-pass filtered with a default cutoff period of 128 seconds/cycle. Motion spikes are detected next. The volumetric data are then resampled to the cortical surface, and the surface-based based data are minimally smoothed (2mm FWHM) to regularize the data. Finally, an html quality-assurance report is generated.

Note: Since `preproc` is based on FSL's Feat, output directories have the `.feat` extension. In the example above, the `.feat` was left off of the `<outdir>` argument. But it's also fine to leave it on. E.g.:

```
preproc BSYN_S_01 BSYN_S_01/raw/bsyn_01.nii.gz BSYN_S_01/preproc/bsyn_01.feat
```

3.3 Full options

The behavior of `preproc` can be modified by providing different options. The full list of options is included below, along with a short description of each. Defaults for the options are enclosed in brackets. When no default is given, the option is disabled by default.

```
--slice <order>     Perform slice timing correction. Order can be: u (up), d
                    (down), or i (interleaved) (e.g. --slice u).
--aroma <mode>      Run ICA-AROMA denoising. Specify which steps to perform
                    with required <mode> argument. Steps include: c (artifact
                    classification), and r (artifact removal). Concatenate to
                    perform both (E.g. --aroma cr).
```

[4] http://www.humanconnectome.org/software/connectome-workbench.html

```
--fix <mode>        Run FIX denoising. Specify which steps to perform with
                    required <mode> argument. Steps include: i (ica), c
                    (classification), and r (removal). Concatenate to perform
                    a sequence (E.g. --fix icr).
--fixW <weights>    Specify the FIX training weights file to use for FIX.
                    (Only relevant if running FIX classification.)
                    [default: Standard.RData]
--fixT <thr>        Specify the FIX classification threshold. Higher values
                    mean more components classified as noise. Sensible values
                    are 5-20. (Only relevant if running FIX.) [default: 10]
--bptf <hp> <lp>    Band-pass temporal filter settings in seconds. <hp>
                    (<lp>) is the longest (shortest) temporal period that
                    will remain in the data. Set either to -1 to turn off.
                    [default: 128 -1]
--nuis-reg <mode>   Regress out nuisance covariates: white matter signal, CSF
                    signal, or linear trend. Mode should be a comma-
                    separated list of: wm, csf, or lin (E.g. --nuis-reg wm,lin).
                    (Note: You can always include these in your GLM, rather
                    regress out here, which will be less aggressive.)
--fdrms <thr>       Detect motion outliers using FDRMS metric. (See FMRIB
                    Technical Report TR99MJ1.) Set your own threshold in mm.
                    [default: 1.5]
--dvars <thr>       Detect motion outliers using DVARS metric (see Power et
                    al 2012). Set you own threshold in PSC * 10.
                    [default: 75% + 1.5*IQR]
--dil <mm>          Dilate the surface mapped data. (To fill in small holes
                    with nearby values.)
--fwhm <mm>         Smoothing kernel in mm. [default: 2]
```

To run `preproc` with options selected, pass the option arguments before the 3 positional arguments.

(Example 1) Standard `preproc` with ascending slice-timing correction:

```
preproc --slice u BSYN_S_01 BSYN_S_01/raw/bsyn_01.nii.gz BSYN_S_01/preproc/bsyn_01.sl
```

(Example 2) FIX ICA with automatic artifact classification and removal:

```
preproc --fix icr  --fixW fix_weights.RData \
  BSYN_S_01 BSYN_S_01/raw/bsyn_01.nii.gz BSYN_S_01/preproc/bsyn_01.fix
```

(Example 3) A preprocessing pipeline for a resting-state analysis, including slice-timing correction, AROMA ICA with automatic artifact removal, band-pass temporal filtering, nuisance signal regression, and 6mm FWHM spatial smoothing.

```
preproc --slice u --aroma cr --bptf 128 10 --nuis-reg wm,lin,csf --fwhm 6.0 \
  BSYN_S_01 BSYN_S_01/raw/resting.nii.gz BSYN_S_01/preproc/resting
```

3.4 Stopping and restarting

If you don't want to run the entire preprocessing pipeline, you can give the `--stop <stage>` option. Recall that the stages of preprocessing are: (1) FEAT volume-based preprocessing, (2) functional-to-anatomical registration, (3) ICA artifact detection, (4) denoising, (5) motion spike detection, (6) volume-to-surface mapping, and (7) QA reporting. The stage you give will be the last stage run. For example, to stop preprocessing after ICA artifact detection you might run:

```
preproc --stop 3 BSYN_S_01 BSYN_S_01/raw/bsyn_01.nii.gz BSYN_S_01/preproc/bsyn_01
```

Stopping preprocessing part way through can give you a chance to inspect the quality of intermediate outputs before proceeding.

In order to resume or restart preprocessing, you will pass `preproc` the `--restart <stage>` option, along with the path to the old preprocessing output directory. `preproc` will then resume processing in this directory, starting from the given stage. For example:

```
preproc --restart 4 BSYN_S_01/preproc/bsyn_01.feat
```

It is also possible to re-run preprocessing with different options, without having to start back from the beginning. For example, to apply different motion spike detection, you could run:

```
preproc --restart 5 --fdrms 0.5 BSYN_S_01/preproc/bsyn_01.feat
```

3.5 Tour of the output directory

The `preproc` output directory contains most of the outputs of FEAT preprocessing, plus some script-specific outputs. Below is a short outline of the most important outputs. A full description of the FEAT-related outputs is provided in the FEAT user guide[5].

```
bsyn_01.feat/
  |- command.txt
    * Text file containing copy of preproc command that was run.
  |- design.fsf
    * FEAT design settings file used in first stage of analysis.
  |- example_func.nii.gz
    * Middle volume from raw functional data.
  |- mask.nii.gz
    * Binary brain mask.
  |- filtered_func_data.nii.gz
    * Functional data after FSL FEAT processing.

  |- surfreg/
    |- register.dat
      * Freesurfer registration file for mapping functional data to
        anatomical/surface space.

  |- art/
    |- fdrms.confound.txt
      * Spike regressor matrix for FDRMS metric.
    |- dvars.confound.txt
      * Spike regressor matrix for DVARS metric.
    |- wm.confound.txt
      * Mean-centered average white-matter signal.
    |- csf.confound.txt
      * Mean-centered average csf signal.

  |- filtered_func_data_clean.nii.gz
    * Functional data after denoising.

  |- lh.32k_fs_LR.mask.shape.gii
    * Binary brain mask after resampling to surface (left-hemisphere).
```

[5] http://fsl.fmrib.ox.ac.uk/fsl/fslwiki/FEAT/UserGuide#FEAT_Output

```
|- rh.32k_fs_LR.mask.shape.gii
  * Binary brain mask after resampling to surface (right-hemisphere).
|- lh.32k_fs_LR.surfed_data.func.gii
  * Clean functional data after resampling to surface (left-hemisphere).
|- rh.32k_fs_LR.surfed_data.func.gii
  * Clean functional data after resampling to surface (right-hemisphere).

|- report_QA.html
  * Html quality-assurance report.
|- QA_stats.csv
  * CSV file containing one row of quality summary statistics.
```

3.5.1 The surface-based data

The most important outputs are the surface-mapped functional data files:

```
lh.32k_fs_LR.surfed_data.func.gii
rh.32k_fs_LR.surfed_data.func.gii
```

You can view the surface-mapped data on the subject's cortical surface using the HCP's `wb_view`:

```
wb_view $SUBJECTS_DIR/BSYN_S_01/surf/lh.32k_fs_LR.hcp_inflated.surf.gii \
  lh.32k_fs_LR.surfed_data.func.gii
```

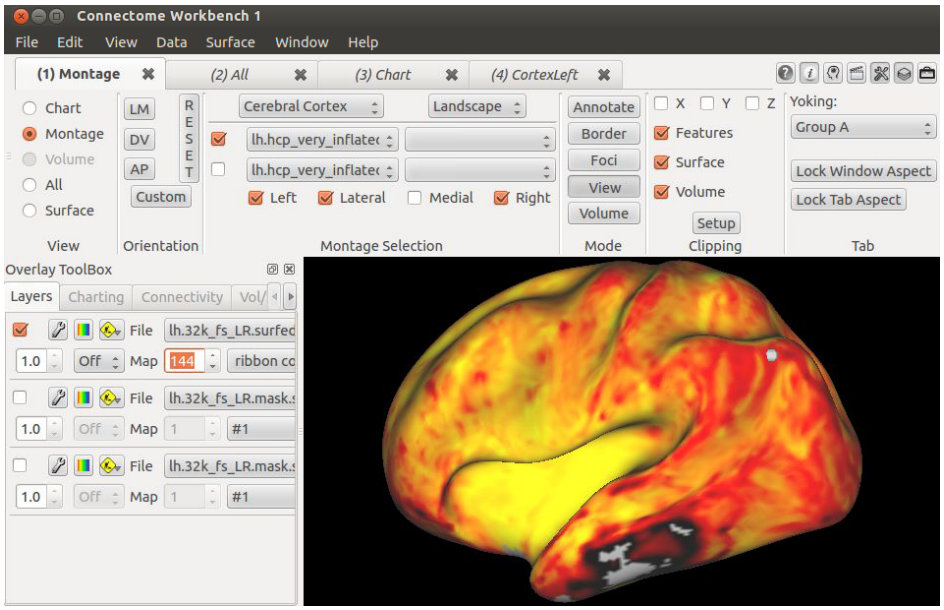

Figure 3.1: Surface-mapped functional data (one TR) displayed with `wb_view`. Brighter colors indicate higher values. Note the signal loss in the anterior temporal lobe due to susceptibility artifacts.

The mask files (`*h.32k_fs_LR.mask.shape.gii`) are also important to inspect, as they show which surface vertices contain data. You can view them using `wb_view` in the same way as above. However, since the masks have no temporal dimension, you can also look at them using Freesurfer's `tksurfer`:

```
tksurfer 32k_fs_LR lh hcp_inflated \
  --overlay lh.32k_fs_LR.mask.shape.gii \
  --fminmax 0.1 1.0
```

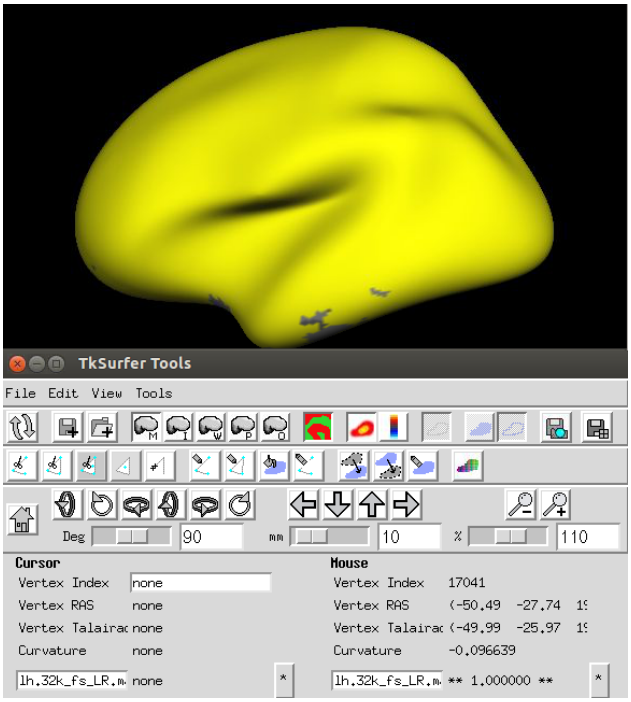

Figure 3.2: Surface-mapped data mask displayed with `tksurfer`. Yellow indicates presence of data on the surface. Note the data loss in the anterior temporal lobe.

Note the `32k_fs_LR` in the filenames and the `tksurfer` command line. This is included to indicate that the data are in 32k_fs_LR space. Recall from Chapter 2 that 32k_fs_LR is a Freesurfer *average* subject with a low-resolution mesh (~32k vertices). In contrast, the native subject surfaces generated by Freesurfer often have over 150k vertices. The average spacing between vertices on the 32k_fs_LR mesh is about 3.75mm, whereas native surfaces have sub-millimeter spacing. This makes the 32k_fs_LR mesh a more appropriate target when resampling functional data onto the surface. Vertices are the surface analogue of voxels. Hence the area covered by a single vertex should match the volume enclosed in a voxel.

3.5.2 The quality-assurance reports

The second most important outputs are the quality-assurance reports. These reports (`report_QA.html` and `QA_stats.csv`) should be inspected carefully every time `preproc` is run. The html report contains the following information:

Summary:

- A header showing the subject ID and run name that was analyzed.
- The `preproc` command line that was executed.
- Animated gifs of the raw and preprocessed data.

Global signal:

- The voxel-wise temporal standard deviation (tSD) of the motion-corrected data (before denoising). (Bright voxels are noisy and likely dominated by nuisance factors [Behzadi2007] [Marcus2013].)
- The voxel-wise temporal signal-to-noise ratio (tSNR) of the motion-corrected data. (Dark regions indicate significant signal dropout [Marcus2013].)
- The average time-series for: (a) the whole brain, (b) gray matter, (c) white matter, and (d) CSF.

- Global signal summary statistics: (a) average tSD within gray matter, (b) average tSD within white matter, (c) correlation of average gray matter and white matter signal, and (d) gray-matter over white matter signal-to-noise ratio (mean signal within gray-matter, divided by tSD in white matter).

Motion:

- MCFLIRT translation and rotation realignment parameters.
- Time-course of FDRMS values.
- Time-course of DVARS values
- FDRMS summary statistics: (a) mean, (b) 95th percentile, (c) number of time-points over various thresholds.

Surface mapping:

- Image of surface outline overlaid on the `example_func` volume after applying functional-to-anatomical transformation. (If the surface outline breaks from the gray matter/white matter intensity boundary, that's a sign the registration failed.)
- Amount of data loss across the lobes (due to FOV alignment or signal dropout). (Some data loss in frontal and temporal lobes is unavoidable. However there should *never* be data missing in the occipital lobe!)

In addition, the `QA_stats.csv` file contains all of the summary statistic information reported in `report_QA.html` in a single row. This makes for easy comparison of data quality across runs and participants.

CHAPTER

FOUR

SINGLE-SUBJECT STATS

4.1 Single-subject stats overview

The goal of statistical fMRI analysis is to find out what brain areas respond to your task. This question is usually addressed in a hierarchical fashion: first for the individual run (first-level), then for one subject (second-level), and finally for the entire group of subjects (third-level). This chapter describes the lab's procedure for single-subject stats, which includes both first-level and second-level analysis.

Before you can run any analyses however, you have to operationalize what *responding to the task* means. This is done by creating a GLM desgin using FSL's `Glm` utility. The design describes all of the conditions in the experiment, and the contrasts of interest.

Once you've defined the design, you can run the statistics. We use FSL to run our first-level analyses. The specific script we use is called `firstlevel`. This is a lab script that streamlines the process.

After first-level analysis, we use a second script, `fixedfx`, to perform a fixed-effects analysis for combining across runs, within a subject.

The following sections will cover how to create a design file, run `firstlevel` and `fixedfx`, as well as how to inspect the outputs..

4.2 Defining the design

4.2.1 General setup

Suppose you have a simple design with three conditions: syntactically complex sentences (sc), syntactically simple sentences (ss), and spoken nonword sequences (nw). You're interested in brain areas that respond to sentences > nonwords, as well as complex > simple sentences. Open `Glm` by running:

```
Glm &
```

Two windows will open. In the smaller window called "GLM Setup" you'll see fields for # timepoints, TR, and high-pass filter cutoff. You can fill these in with the correct values or leave them as is. The values are overridden by `firstlevel`, so they really only matter for documentation purposes and posterity.

4.2.2 Defining the explanatory variables

The larger window called "General Linear Model" is where you fill in information about your study design and desired contrasts. First, you enter the number of explanatory variables (EVs) in your study. This would be 3 in the current case (sc, ss, and nw), although we could also consider including a "response" EV to

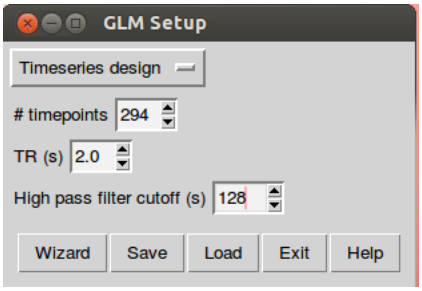

Figure 4.1: GLM Setup window.

model behavioral responses across conditions, or a "drops" EV to model trials on which the participant did not respond.

Once you enter the number of EVs, a tab for each EV will appear below. On each tab, you will fill in the EV name plus some settings for how you want to model the BOLD data. You will likely use the same settings every time you run an analysis.

Basic shape:

> This is where you specify the temporal shape of the EV. In most fMRI designs, trains of trials are best modeled as an on-off "boxcar" function. The easiest way to set this kind of model up is by ticking the "Custom (3 column format)" option. To use this setting, you will need a *timing* file for each condition. The timing file describes when each trial started, and how long it lasted for. More information on the timing file format is given below.

Note: You do not *need* to include timing files when setting up the design. They will be filled in when you run `firstlevel`. However, if you include example timing files during the initial design set-up, you'll be able to generate a design.png figure. These can be useful visual reference later on.

Convolution:

> To generate the final BOLD predictor, the condition time-course is convolved with a hemodynamic response function (HRF). The "Convolution" field is where you specify what kind of HRF you want to use. "Double-Gamma HRF" is an uncontroversial choice.

Orthogonalise:

> Orthogonalizing *A* with respect to *B* means all shared variance between *A* and *B* gets assigned to *B*. There's almost **never** a good reason to do this.

Add temporal derivative:

> Including a temporal derivative is generally a good idea, especially if you didn't run slice-timing correction. It helps to correct for small differences between the predicted and actual start of the hemodynamic response.

Apply temporal filtering:

> You should always temporally filter your EVs in the same way as your data.

4.2.3 Contrasts

Once you have the EVs set up, you can define your contrasts. Contrasts are defined as vectors with one entry per EV. The number one rule is that contrasts between multiple conditions should sum to zero. Otherwise,

you can mostly follow your intuition. For example, you can define the sentence > nonword contrast (s>nw) as $[1.0, 1.0, -2.0]$, or the complex > simple sentence contrast (sc>ss) as $[1.0, -1.0, 0.0]$.

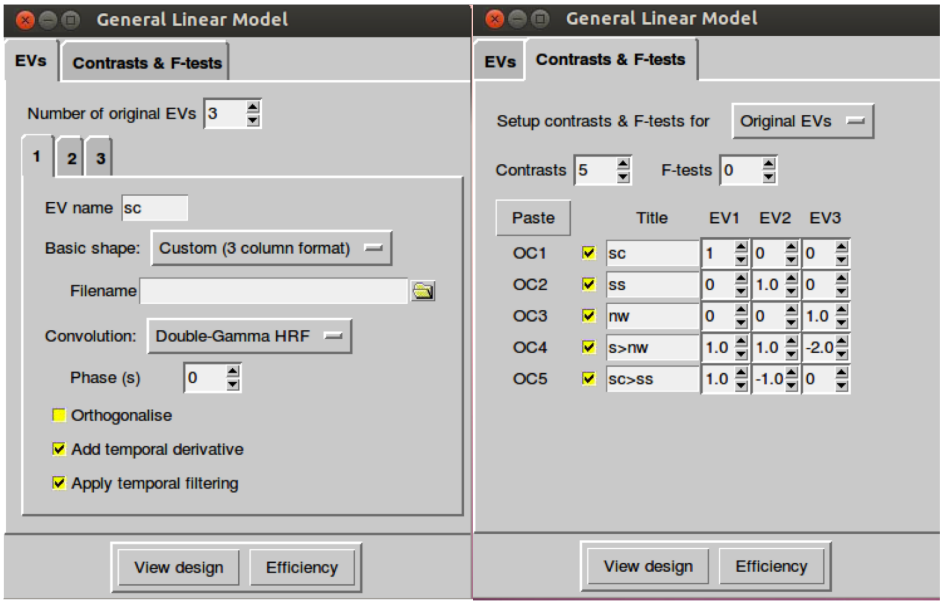

Figure 4.2: General Linear Model EV and contrast set-up.

4.2.4 Saving the design

After you finish defining contrasts you can save you design, e.g. as `design.fsf`.

Note: You might get this warning when you save: *Problem processing the model: Can't open for reading....* This is to be expected if you didn't include any timing files in the "Basic Shape" part of the dialog. As long as the `*.fsf` file is saved, you're fine to proceed.

For further information on GLM design set-up, see the FSL FEAT User Guide[1]. Also, Jeanette Mumford has a lot of useful tutorials[2] on GLM design concepts. You should also read chapters 5-7 and appendix A from [Poldrack2011] for general background.

4.3 Timing files

Timing files are used to represent a sequence of trials or events during a functional run. You will need one timing file for each subject, run, and condition in your experiment. There are two possible formats for the timing files. The most common in the 3-column format. Timing files in this format contain a 3-column matrix of numbers:

```
2.0 6.66 1
130.0 6.66 1
146.0 6.66 1
178.0 6.66 1
290.0 6.66 1
```

[1] http://fsl.fmrib.ox.ac.uk/fsl/fslwiki/FEAT/UserGuide#Stats_.28First-level.29
[2] http://mumfordbrainstats.tumblr.com/archive

```
354.0 6.66 1
386.0 6.66 1
434.0 6.66 1
562.0 6.66 1
```

Each column represents a single trial or event. The first column is the onset of the trial in seconds, relative to the start of the first scan TR. The second column is the duration of the trial in seconds. And the third column is the height of the condition regressor for the trial (usually 1).

The second timing file format is the single-column format. Timing files in this format contain a single column of numbers. The column must be the same length as the scan run. The *i*th entry is the height of the condition regressor for TR *i*.

Timing files must follow a specific naming convention. The filenames must end in `{condition}.txt`. Extra characters in the filename must be separated from the suffix by a `-` character. For example, the following are all legal timing file names:

```
sc.txt
bsyn_01-sc.txt
BSYN_S_01-bsyn_01-sc.txt
```

Critically, the condition names in your timing files must match those in your Glm `.fsf` file. The `firstlevel` script depends on this naming convention for parsing the design.

4.4 Basic `firstlevel` usage

The basic usage pattern for `firstlevel` is:

```
firstlevel <func> <surf> <design> <EVs> <outdir>
```

where `<func>` is a preprocessed, surface-mapped functional data series in Gifti format, typically one of `?h.32k_fs_LR.surfed_data.func.gii` produced during preprocessing.

`<surf>` is an anatomical Gifti surface in the same space as the functional data. Since surface-based functional data files do not contain any spatial structure, `firstlevel` relies on this anatomical surface for determining the spatial relations between vertices. Consequently, this surface should accurately represent the cortical anatomy for the subject being analyzed. In the standard use case, you should use the subject's 32k_fs_LR midthickness surface: `surf/?h.32k_fs_LR.midthickness.surf.gii`.

`<design>` is the GLM design `.fsf` file for your analysis.

`<EVs>` is a list of timing files for the given run and GLM design. The list should be delimited by spaces and enclosed in double-quotes. The quotes ensure that the list is passed to the script as a single argument.

`<outdir>` is the path to the desired output directory. The naming convention for output directories is `{run name}.?h.glm`.

For example, you might run something like:

```
firstlevel \
  BSYN_S_01/preproc/bsyn_01.feat/lh.32k_fs_LR.surfed_data.func.gii \
  $SUBJECTS_DIR/BSYN_S_01/surf/lh.32k_fs_LR.midthickness.surf.gii \
  Designs/bsyn_design.fsf \
  "BSYN_S_01/timing/bsyn_01-sc.txt BSYN_S_01/timing/bsyn_01-ss.txt BSYN_S_01/timing/bsyn_01-nw.txt" \
  BSYN_S_01/firstlevel/bsyn_01.lh.glm
```

4.5 Full `firstlevel` options

The three stages within the `firstlevel` script are: (1) pre-stats smoothing, (2) pre-whitening, and (3) GlM fitting, optionally including nuisance covariates. These stages can be modified with a set of command-line options:

```
--cfd <list>    List of text files containing confound regressors.
                Each file should have one line per TR, and one column
                per regressor (delimited by tabs). The list itself
                should be delimited by commas (E.g. --cfd cfd1.txt,cfd2.txt)
--tr <secs>     TR for <func-data> in seconds. [default: 2]
--hpf <secs>    High pass filter cutoff in seconds. This is the
                longest temporal period that will remain int the
                model. It should match the value used during
                preprocessing. [default: 128]
--fwhm <num>    Pre-stats smoothing kernel in mm. [default: 5.657]
--pw (0|1|2)    Prewhitening level. 0 is none, 1 is fast
                (no ac smoothing), 2 is full. [default: 2]
```

Note: Be careful when using the `--fwhm` option. If you have already applied some smoothing during preprocessing, you will need to use the formula $z = \sqrt{(x^2 - y^2)}$, where x is the desired smoothness, y is the current smoothness, and z is the additional smoothing needed. The default value of 5.657 assumes a current smoothness of 2mm FWHM, giving an overall smoothness of $\sqrt{(5.657^2 + 2^2)} \approx 6$.

(Example 1) First-level analysis with additional white-matter, CSF, and motion spike nuisance covariates:

```
wm_cfd=BSYN_S_01/preproc/bsyn_01.feat/art/wm.confound.txt
csf_cfd=BSYN_S_01/preproc/bsyn_01.feat/art/csf.confound.txt
mot_cfd=BSYN_S_01/preproc/bsyn_01.feat/art/fdrms.confound.txt

firstlevel \
  --cfd $wm_cfd,$csf_cfd,$mot_cfd \
  BSYN_S_01/preproc/bsyn_01.feat/lh.32k_fs_LR.surfed_data.func.gii \
  $SUBJECTS_DIR/BSYN_S_01/surf/lh.32k_fs_LR.midthickness.surf.gii \
  Designs/bsyn_design.fsf \
  "BSYN_S_01/timing/bsyn_01-sc.txt BSYN_S_01/timing/bsyn_01-ss.txt BSYN_S_01/timing/bsyn_01-nw.txt" \
  BSYN_S_01/firstlevel/bsyn_01.lh.glm
```

4.6 `firstlevel` outputs

The `firstlevel` output directory is organized as follows:

```
bsyn_01.lh.glm/
  |- commandline
    * Text file containing a copy of the firstlevel command that was run.
  |- design.fsf
    * GLM design file for this particular analysis (e.g. including the proper
      timing files instead of the place-holder values.
  |- design.mat
    * GLM design matrix.
  |- design.con
    * GLM contrast matrix.
  |- design.png
    * Visual representation of GLM design.
```

```
|- mask.shape.gii
  * Mask showing vertices that contain data.

|- timing_files/
  * Directory containing copies of the timing files.

|- stats/
  |- pe1.func.gii
    * parameter estimates (betas) for the first regressor.
  |- cope1.func.gii
    * contrast of parameter estimates for the first contrast.
  |- varcope1.func.gii
    * variance map associated with contrast 1.
  |- tstat1.func.gii
    * t-map for constrast 1.
  |- zstat1.func.gii
    * z-map for contrast 1.
```

`firstlevel` produces other kinds of outputs as well, which you can investigate on your own. Those listed above are the most important outputs.

After first-level analysis, you should load the data mask and z-map for each contrast of interest. You should look at the data mask to verify your surface coverage. The z-maps on the other hand will help you diagnose issues with your model. A surprisingly weak or inverted z-map for one run suggests that something is wrong with your model for this particular run. These kinds of issues can be very difficult to detect at later stages of analysis! Being careful and thorough here can save you hours of headaches later. For example:

```
wb_view $SUBJECTS_DIR/BSYN_S_01/surf/lh.32k_fs_LR.hcp_very_inflated.surf.gii \
  BSYN_S_01/firstlevel/bsyn_01.lh.glm/mask.shape.gii \
  BSYN_S_01/firstlevel/bsyn_01.lh.glm/stats/zstat*.func.gii
```

4.7 `fixedfx` usage

Second-level, fixed-effects analysis is comparatively simpler to first-level analysis. There is no GLM design set-up, and the command-line usage is relatively straightforward:

```
fixedfx <hemi> <out-dir> <stats-dir>...
```

where `<hemi>` is either "lh" or "rh", `<out-dir>` is the desired output directory, and `<stats-dir>` is a `firstlevel` output directory. The ellipsis in the usage pattern means that the `<stats-dir>` argument is repeatable: you can provide an arbitrary number of stats directories. In practice, you will give as many stats directories as you have functional runs. The naming convention for the output directory is `{task name}.?h.ffx`.

For example:

```
fixedfx lh BSYN_S_01/firstlevel/bsyn.lh.ffx BSYN_S_01/firstlevel/bsyn_??.lh.glm
```

Note: Note the "glob" pattern used in place of a literal list of `firstlevel` stats directories. You should become comfortable using these kinds of expressions as they're very time-saving. See this tutorial[3] for an introduction.

In addition, there are a few command-line options you can pass to `fixedfx`:

[3] http://ryanstutorials.net/linuxtutorial/wildcards.php

```
--surf <surf>      Surface used to convert between Nifti and Gifti formats
                   [Default: 32k_fs_LR/`<hemi>`.hcp_very_inflated.surf.gii].
--empty-fix        Betas for vertices that have data in only some
                   are calculated based on those runs. Otherwise,
                   no stats are done for these vertices.
```

The `--surf` option allows you to give `fixedfx` an anatomical surface, which it uses to convert data files between various formats. Importantly however, fixed-effects analysis is totally naive of surface anatomy. Each vertex is analyzed independently. Consequently, the surface passed to `fixedfx` need not be anatomically accurate for the subject. It only needs to be in the same *space* as the functional data.

When you pass the `--empty-fix` flag, `fixedfx` will try to be smart about how it handles vertices that contain data only in some runs. In such cases, only runs containing data will contribute to the statistics.

4.8 `fixedfx` outputs

The `fixedfx` output directory contains the following files and folders:

```
bsyn.lh.glm/
  |- command.txt
    * Text file containing a copy of the fixedfx command that was run.
  |- design.mat
    * Fixed-effects design matrix.

  |- lh.hcp_very_inflated.surf.gii
    * Sym-linked copy of the anatomical surface used in the analysis.

  |- mask.shape.gii
    * Binary mask showing vertices that contributed data to the analysis.
  |- masks4d.shape.gii
    * Data masks from each of the analyzed runs, concatenated in time.
  |- coverage.shape.gii
    * Map showing the number of runs that contributed data to each vertex
      (only relevant when using --empty-fix option).

  |- zstat1.func.gii
    * Fixed-effects z-map for contrast 1.
  |- sigp1_fdr.func.gii
    * FDR-corrected p-values for contrast 1 (-log10(p) transformed).

  |- cope1/
    |- cope1.func.gii
      * Fixed-effects contrast of parameter estimates for contrast 1.
    |- varcope1.func.gii
      * Fixed-effects variance for contrast 1.
```

After running `fixedfx` you should again check the data coverage by viewing the `coverage.shape.gii` and `mask.shape.gii` files. You should also look at the z-maps to gauge the anatomical distribution of activation, and the strength of your contrast. For example, you can run:

```
wb_view lh.hcp_very_inflated.surf.gii mask.shape.gii coverage.shape.gii zstat*.func.gii
```

CHAPTER

FIVE

GROUP STATS

5.1 Group stats overview

The purpose of group statisticis is to make inferences on what brain regions are active across people. The inputs to group statistics are the lower-level contrast images from our single subject fixed-effects analyses. We then use linear regression to ask if e.g. the contrast is greater than 0 across subjects.

There are three main steps in group analysis:

1. Defining the group design and contrasts, which articulate our hypotheses about the activation pattern across subjects.
2. Evaluating the hypotheses by fitting a linear model. Critically, subject is treated as a random effect (as opposed to a fixed effect).
3. Correcting for multiple comparisons across the brain.

Additional background on group analysis can be found in chapters 6 and 7 of [Poldrack2011].

5.2 Defining the group-level design

Group-level analysis, like single-subject analysis, is carried out using a general linear model (GLM). At each voxel, we fit a model of the form:

$$\mathbf{y} = \mathbf{X}\beta + \epsilon$$

However here the rows in $\mathbf{y}$ and $\mathbf{X}$ correspond to subjects, rather than time-points. Each entry is the value of a lower-level contrast from one subject.

The first step in group analysis is to define the design matrix $\mathbf{X}$, as well as the group-level contrasts we're interested in. Group designs can be tricky to get a feel for at first. FSL provides a cookbook[1] for defining group designs and contrasts, which should help you get started.

Group designs can be defined using FSL's `Glm` tool, just as in first-level analysis. The only difference is you'll need to select "Higher-level/ non-timeseries design" in the GLM Setup window, rather than "Timeseries design".

Once you've completed defining the design in the Glm gui, you can save it as you did when setting up first-level analysis. Among the ouputs, there will be a `.mat` file, containing the design matrix, and a `.con` file, containing the group *t*-contrasts. You will need these outputs when you go to run the actual statistics.

[1] http://fsl.fmrib.ox.ac.uk/fsl/fslwiki/GLM

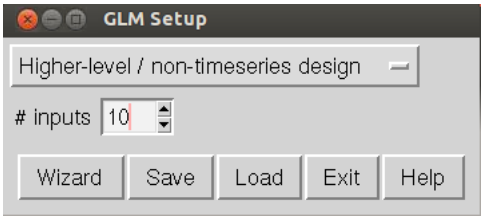

Figure 5.1: GLM Setup window for group designs.

However, it is not necessary to use `Glm` to define the design. Since group designs are much simpler than first-level designs (e.g. there are no timing files, no HRFs, and no convolutions), it is probably easier to just write out the design matrix and contrasts directly as plain text files.

Suppose you have an experiment with 10 subjects, 5 in group *A* and 5 in group *B*. You want to know where there is consistent activation in each group separately ("group average" contrasts), as well as where there is more activation in group *A* than in group *B* (group difference contrast).

Assume the lower-level contrasts have been put together with the *A*'s first, followed by the *B*'s to form the data series **y**. You would save the following matrix in a text file, to serve as your design matrix:

```
1 0
1 0
1 0
1 0
1 0
0 1
0 1
0 1
0 1
0 1
```

Here the first column models the mean activation in group *A*, while the second models the mean in group *B*. Your contrasts should be saved as row vectors, each in a separate file. Each contrast should have one entry per column in the design matrix. For example, the group average contrast for group *A* would be:

```
1 0
```

While the group difference contrast between *A* and *B* would be:

```
1 -1
```

In addition to simple *t*-contrasts, such as group averages and group differences, it is also possible to define *F*-contrasts. These contrasts are needed for complicated designs that require running an ANOVA at the group-level. Examples of these situations can be found in FSL's group design cookbook[2].

An *F*-contrast involves running an *F*-test over several *t*-contrasts. *F*-contrasts are represented as a text matrix with multiple rows. Each row indicates a separate *t*-contrast to be entered into the *F*-test.

Note: FSL's `Glm` saves all t-contrasts in a single `.con`, with each on a separate row. However, our script for group analysis expects each t-contrast to be in a separate file. Contrast files with multiple rows are assumed to represent *F*-contrasts. As a result you will need to modify `.con` file before running the stats. (The reason for this design choice on how to interpret contrast files with multiple rows will be discussed in the next section.)

---

[2] http://fsl.fmrib.ox.ac.uk/fsl/fslwiki/GLM

5.3 groupstats usage

5.3.1 Basic usage

Our script for running group analysis is called `groupstats`. It is built around Freesurfer's `mri_glmfit` tool for running surface-based group analysis.

The basic usage pattern for `groupstats` is:

```
groupstats [options] <hemi> <design> <con> <outdir> <ffx-dir>...
```

Here `<hemi>` indicates the hemisphere of the data (either `lh` or `rh`). The `<design>` and `<con>` arguments are for the design matrix and contrast files respectively. Either the FSL Glm format, or the plain text format described above is acceptable. (Except when there are multiple contrasts in an FSL `.con` file, as discussed in the note above.)

The `<outdir>` argument is a path to the desired output directory. It the output directory already exists, a "+" will be added to the end of the path. This prevents overwriting previous analyses. The conventional file extension for `groupstats` output directories is `.rfx`.

Finally, `<ffx-dir>...` is a repeatable argument used to pass in a list of lower-level fixed-effects (`.ffx`) analysis folders. There should be one folder for each subject. Each folder should contain the same contrasts, and contain results for the same hemisphere. The order of fixed-effects directories should also match the row order in the design matrix, since the fixed-effects cope images will be concatenated in the order given.

5.3.2 Default behavior and options

The default behavior of `groupstats` is to:

- Assume the input data are in the standard 32k_fs_LR surface space.
- Only analyze vertices containing data in every subject.
- Run a separate group analysis for every lower-level contrast found in the fixed-effects directories.
- Use weighted least-squares regression to fit the linear model.

However, this default behavior can be modified using a few options:

```
--s <subj>          Freesurfer subject to perform analysis on [default: 32k_fs_LR].
--copes <copes>     List of copes (contrasts) to analyze. Otherwise, will
                    analyze all copes for task. Must be comma separated
                    (E.g. --copes 1,2,3,7).
--ols               Do ordinary least squares regression. Otherwise, will
                    do weighted least squares.
--dil <dist>        Distance in mm to dilate data before performing stats.
```

5.4 groupstats procedure

The processing steps implemented in `groupstats` are as follows:

1. Read command-line arguments and validate inputs. In particular, load the design matrix and contrast file and verify that:
   (a) The design matrix is a valid numerical matrix.
   (b) The number of rows in the design matrix matches the number of fixed-effects directories.

(c) The design matrix and contrast files have the same number of columns.

(d) Warn the user if the contrast contains multiple rows. Make sure they know it will be interpreted as an *F*-contrast.

2. Retrieve and organize inputs to `mri_glmfit` from the fixed-effects directories. This includes for each contrast: finding the cope and varcope images for each subject, and concatenating them. These concatenated cope and varcope images become the key inputs to `mri_glmfit`.

3. Run `mri_glmfit` for each lower-level contrast. By default, weighted least-squares (WLS) regression is used. WLS extends ordinary least-squares (OLS) to the case where subjects have different lower-level variances. When you use WLS, noisy subjects are down-weighted relative to the cleaner subjects. Details on the WLS procedure can be found pages 102-104 and page 196 of [Poldrack2011].

   Regardless whether WLS or OLS is used, `mri_glmfit` always performs a random-effects analysis. This means that the between-subject variance is used to compute the final *t*-statistics, rather than the within-subject variances.

4. Perform false-discovery rate (FDR) correction using the lab script: `fdr_corr`. This produces an FDR-adjusted *p*-value image, as well the uncorrected threshold for $p < .05$ FDR correction. Importantly, this FDR correction is a one-tailed procedure–only the positive side of the contrast matters.

5. Save some instructions for doing permutation-based multiple comparison correction, using either Freesurfer's `mri_glmfit-sim` or FSL's `palm`. Permutation correction is not included in `groupstats` since it's time-consuming and relatively easy to run on its own, using these built-in Freesurfer and FSL tools. Permutation correction is discussed more below.

5.4.1 A note on multiple contrasts

There is an important point to make about why multiple group contrasts are not supported in `groupstats`. In `mri_glmfit` there is a bug that prevents accurate multiple comparison correction for analyses with multiple group contrasts. This bug is documented on the Freesurfer mailing list[3]. To keep people from making making this mistake and not realizing it, we just don't support multiple contrasts in the same analysis. The workaround is to run each group contrast as a separate `groupstats` job.

5.5 `groupstats` outputs

The `groupstats` output directory is organized like this:

```
bsyn_Savg.lh.rfx/
  |- log
    * Local log file containing e.g. the command-line that was run, and
      information on running permutation correction.

  |- design.mat
  |- design.csv
  |- con1.mtx
  |- con1.csv
    * Design and contrast files in plain text format (tab-delimited):
      design.mat, con1.mtx; and csv format: design.csv, con1.csv. The csv
      format duplicates make it easier to use FSL's palm.

  |- cope1/
    * GLM output directory for first lower-level contrast.
```

[3] https://www.mail-archive.com/freesurfer%40nmr.mgh.harvard.edu/msg21068.html

```
|- mri_glmfit.log

|- cope1.func.gii
|- varcope1.func.gii
  * Concatenated cope and varcope images. One map per subject.
|- mask.mgh
|- mask.shape.gii
  * Data mask for the analysis. 1 on all vertices with valid (non-zero)
    data in each subject, and 0 elsewhere.

|- Xg.dat
|- X.mat
  * Group design matrix in plain text (.dat) and MATLAB format (.mat).
|- dof.dat
  * Group degrees of freedom (1 - # design matrix rows).
|- fwhm.dat
  * Measured fwhm in data. Usually slightly more than the smoothing
    kernel used during preprocessing (due to intrinsic smoothness in
    data).

|- beta.mgh
  * Concatenated beta parameter estimates. One map per design matrix
    column.
|- rvar.mgh
|- rstd.mgh
  * Residual error variance/stddev
|- wn.mgh
  * Concatenated normalized weights for WLS regression. One map per
    subject.

|- con1/
  * Statistical results folder for the included contrast.

  |- C.dat
    * Contrast vector in plain text format.
  |- F.mgh
    * F-statistic map. t^2 when just a t-contrast is used.
  |- sig.mgh
    * Uncorrected p-value map. Values are -log10(p), for easier
      visualization.

  |- sig_fdr.func.gii
    * FDR-adjusted one-tailed p-value map, also in -log10(p) format.
  |- fdr05_thresh.txt
    * Uncorrected -log10(p) threshold for p < .05 FDR correction.
```

Viewing group statistical results is easiest with `tksurfer`. For example, you could view all the key results by running the following command from inside the `con1` directory.

```
tksurfer 32k_fs_LR lh hcp_very_inflated \
  -overlay F.mgh \
  -overlay sig.mgh \
  -overlay sig_fdr.func.gii
```

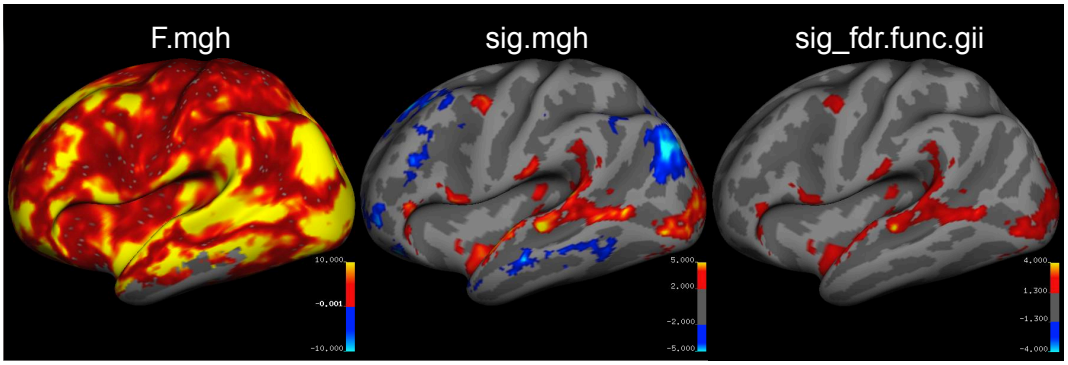

Figure 5.2: Example statistical maps viewed in `tksurfer` from a group analysis. Contrast shown is spoken verbs > spoken nouns, group average in a sample of 12 sighted adults.

5.6 Multiple comparison correction

5.6.1 FDR correction using `fdr_corr`

Only FDR correction is performed as part of `groupstats`. FDR correction is carried out using the lab script `fdr_corr`. The usage pattern for `fdr_corr` is:

```
fdr_corr --mask=<mask> --surf=<surf> --q=<q> <stat-type> <image> <out>
```

Descriptions of the input arguments are included below. (Flagged arguments are optional).

```
--mask=<mask>    Mask containing voxels to perform correction over.
                 By default will select voxels with non-zero values.
--surf=<surf>    Surface used to convert between Nifti and Gifti formats
                 [Default: 32k_fs_LR/lh.hcp_very_inflated.surf.gii].
--q=<q>          FDR threshold [Default: 0.05].

<stat-type>      "zstat" or "logp".
<image>          Input statistical image. Accepts .nii.gz, .gii, .mgh.
<out>            Output corrected image. .gii or .nii.gz permitted.
```

The primary output is an FDR-adjusted p-value map in $-\log_{10}(p)$ format. A nice discussion of FDR correction and FDR-adjusted p-values can be found on Anderson Winkler's Brainder blog[4].

`fdr_corr` also prints the uncorrected corrected threshold for $p < .05$ FDR correction, also in $-\log_{10}(p)$ format.

Note: There is a subtle difference an FDR-corrected and FDR-adjusted p-values. Be careful not to use them interchangeably.

5.6.2 Permutation (FWER) correction

A more conservative form of multiple-comparison correction can be achieved using non-parametric permutation testing [Nichols2002]. Permutation testing controls the family-wise error rate (FWER) rather than the false-discovery rate (FDR). The false-discovery rate is the proportion of null hypothesis rejections in a single study that are in fact false positives. By contrast, the family-wise error rate is the proportion of *studies* that

[4] https://brainder.org/2011/09/05/fdr-corrected-fdr-adjusted-p-values/

contain even a single false positive. $p < .05$ FDR correction ensures that 95% of what you report is accurate, while $p < .05$ FWER correction ensures a 95% chance that *everything* you report is accurate.

Intuitively, permutation testing involves running many repeated analyses of mixed-up (permuted) data. In thise permuted analyses, we expect the null hypothesis to be true everywhere. Then, if the stats in the unpermuted data are better than 95% of the permuted analyses, we pass the correction. More details can be found in [Nichols2002] and [Winkler2014].

There are two tools available for running permutation testing with surface-based data: Freesurfer's `mri_glmfit-sim`, and FSL's `palm`. `mri_glmfit-sim` is fast and easy to run. It takes a GLM directory generated by `mri_glmfit` as input, plus a few settings controlling the type of permutation test to perform. For example, you could run something like:

```
mri_glmfit-sim --glmdir bsyn_Savg.lh.rfx/cope1 \
  --sim perm 5000 2 perm_5000_01 --sim-sign pos
```

This would run 5000 permutations, using a cluster extent test-statistic, with $p = .01$ ($-\log_{10}(p) = 2$) cluster forming threshold. Ouptuts would be saved with the filename prefix "perm_5000_01". More details on how to use `mri_glmfit-sim` can be found on the Freesurfer mri_glmfit wiki[5], or by looking up the help message in the terminal: `mri_glmfit-sim --help`.

The one drawback to `mri_glmfit-sim` is its lack of extensive options. For example, it only supports the cluster extent or peak voxel test statistics. If, for example, you're interested in using the TFCE [Smith2009] or cluster mass test-statistics, you can use FSL's PALM[6]. PALM is a set of MATLAB functions with a command-line script interface: `palm`. It is considerably slower than `mri_glmfit-sim`, and has a more cumbersome usage usage, so you should only use it if you *need* one of its special options.

You can read more about using `palm` on the PALM wiki[7], or by accessing the help message: `palm -h`.

[5] https://surfer.nmr.mgh.harvard.edu/fswiki/mri_glmfit
[6] http://fsl.fmrib.ox.ac.uk/fsl/fslwiki/PALM/UserGuide
[7] http://fsl.fmrib.ox.ac.uk/fsl/fslwiki/PALM/UserGuide#Using_PALM

CHAPTER

SIX

ROI CREATION

6.1 Overview of ROI analysis and ROI creation

Region of interest (ROI) analyses of fMRI data offer several benefits over standard whole-brain analyses. First, they provide a hypothesis-driven way to reduce the number of independent tests performed. Second, averaging fMRI time-courses over an apriori homogenous ROI can boost the signal-to-noise ratio, much like spatial smoothing.

The first step of an ROI analysis is to define the voxels/vertices that make up your ROI. ROIs can be defined in many ways. ROIs can be based on anatomical data, such as sulcal and gyral landmarks, or underlying cytoarchitecture. However, we most often use functional ROIs, where vertices are picked based on their functional profiles.

Functional ROIs can be defined in many ways. Our tool for defining functional ROIs is called `make_roi`. In the rest of this section we will walk through how to use `make_roi` to define different kinds of ROIs.

6.2 Defining ROIs by vertex thresholding

6.2.1 Absolute thresholded ROIs

The simplest way to define a functional ROI is to choose all vertices within a *search space* that surpass some threshold for a contrast. For example, your ROI might consist of all vertices within the left inferior frontal gyrus (IFG) that have a z-value greater than 3.1 for a sentence > nonword contrast.

To define this kind of ROI, you will invoke `make_roi` in "vertex" mode and provide the search space mask (`--search`) and threshold (`-t`) as command-line arguments:

```
make_roi vertex --search IFG.lh.shape.gii -t 3.1 \
  zstat_S-NW.lh.func.gii \
  IFG_S-NW_thr3.1.lh.shape.gii
```

After the `-t` option, there are two positional arguments: the input statistic map, in this case a z-stat map (`zstat_S-NW.lh.func.gii`), and the output path for the generated ROI (`IFG_S-NW.lh.shape.gii`).

The general usage pattern is:

```
make_roi vertex [options] -t <thr> <stat> <roi>
```

Note: Recall the naming conventions for Gifti files: .func.gii is intended to represent functional data, while .shape.gii often indicates a binary cortical mask.

6.2.2 Relative thresholded ROIs

One problem with the above approach is that the size of your ROI will depend a lot on the overall amount of activation for the defining contrast. When there is a lot of activation, you'll get a very large ROI, but when there isn't much activation you might not find any supra-threshold vertices. This is a problem if you're trying to define equivalent ROIs across subjects.

To get around this, it's often better to set your threshold *relative* to the activation in the search space, as a percentile of the overall distribution. So, instead of using an absolute threshold of 3.1, you might use the top 10^{th} percentile as your cutoff. (Importantly, this is the top 10^{th} percentile *within* the search space, not across the entire activation map.) The advantage of this method is that you're guaranteed to get the same number of voxels every time you define an ROI within the given search space.

To define this kind of ROI, you will call `make_roi vertex` as above, only you'll pass the `-p <perc>` flag, rather than `-t <thr>`:

```
make_roi vertex --search IFG.lh.shape.gii -p 10 \
  zstat_S-NW.lh.func.gii \
  IFG_S-NW_perc10.lh.shape.gii
```

Note: The `-p` flag ranges from 0 to 100, **not** 0.0 to 1.0! Also, note that `-p 10` returns the top 10% of vertices, i.e. all vertices above the 90^{th} percentile.

6.2.3 Fixed-size thresholded ROIs

Using relative thresholding will still result in ROIs that vary in size across different search spaces. To address this potential issue, we can define fixed-size ROIs, which contain the top *N* most "active" vertices within the search space.

For this method, you will use the `-n <size>` flag:

```
make_roi vertex --search IFG.lh.shape.gii -n 200 \
  zstat_S-NW.lh.func.gii \
  IFG_S-NW_n200.lh.shape.gii
```

6.3 Cluster and watershed-based ROI selection

6.3.1 Defining contiguous ROI clusters

Cortical function is correlated with proximity: nearby regions often have similar functions. We can use this prior knowledge to define ROIs that are a priori more functionally homogenous, by enforcing spatial *compactness*.

One way to bias ROI selection towards more compact Regions is to require ROIs to be spatially contiguous.

You can define contiguous ROIs by calling `make_roi` in "cluster" mode. In addition, you will need to provide either an absolute (`-t <thr>`) or relative (`-p <perc>`) threshold. All vertices above the given threshold will be selected and categorized into separate contiguous clusters. By default, the "strongest" cluster is chosen as the ROI, where strength is measured as the sum of statistic values within the cluster. For example:

```
make_roi cluster --search IFG.lh.shape.gii -p 20 \
  zstat_S-NW.lh.func.gii \
  IFG_S-NW_perc20_cluster.lh.shape.gii
```

Alternate methods also exist for picking your ROI from the available clusters. You can change the evaluation metric from strength to one of:

- **dist**: Pick the cluster closest to a given "literature coordinate" (requires using the `--lit-coord <coord>` option, e.g. `--lit-coord -20,19,-5`).
- **strength / dist**: Combined strength and distance evaluation metric. (Note that by dividing strength by distance, relative changes in either factor are weighted equaly. E.g. being twice as close is just as good as being twice as strong.)
- **strength / spread**: "Spread" is directly a measure of how compact or spread out a cluster is. It is computed as the average distance from each vertex in the cluster to the cluster peak, weighted by the vertices' statistic values. This metric combines strength and spread, weighting both factors equally.

To set a different evaluation metric, you will pass the `--eval <metric>` option. For example:

```
# Cluster ROI with literature coordinate distance evaluation metric.
make_roi cluster --search IFG.lh.shape.gii --lit-coord -20,19,-5 --eval dist -p 20 \
  zstat_S-NW.lh.func.gii \
  IFG_S-NW_perc20_cluster_dist.lh.shape.gii

# Cluster ROI with strength / spread evaluation metric.
make_roi cluster --search IFG.lh.shape.gii --eval "strength / spread" -p 20 \
  zstat_S-NW.lh.func.gii \
  IFG_S-NW_perc20_cluster_strength_spread.lh.shape.gii
```

You also have the option to manually choose your ROI among the available clusters by changing the `--select <mode>` option. By default, the selection mode is "best" (as above). Alternatively, you can pass `--select pick` and `make_roi` will pop open a `tksurfer` window showing the numbered clusters overlaid on the statistical map. You can then enter the number of your desired cluster in the command prompt to override the "best" selection.

If you instead pass `--select all`, all of the clusters will be saved out in a single metric Gifti file–each cluster with a different numerical label > 0.

6.3.2 Watershed-based ROIs

One issue with the cluster-based approach to ROI selection is that it offers little control over the size of the final ROI. Secondly, it depends on an arbitrary cluster-forming threshold (e.g. `-p 20`) which may not carve the search space into the right functional units.

The watershed-based ROI procedure improves on these issues, while still generating spatially compact ROIs. The watershed procedure operates in two stages. First, a watershed algorithm is used to parcellate the search space into contiguous subregions, corresponding to the distinct activation peaks in the statistic map. Next, separate ROIs (often clusters) are defined for each subregion, based on some vertex thresholding criteria (e.g. `-p 10` or `-n 200`).

To use the watershed procedure, you will invoke `make_roi` in "watershed" mode, and otherwise use the same options as above:

```
make_roi watershed --search IFG.lh.shape.gii -p 10 \
  zstat_S-NW.lh.func.gii \
  IFG_S-NW_perc20_watershed.lh.shape.gii
```

As in cluster mode, you will often have multiple candidate ROIs to choose from. All the same methods for ROI selection described above apply just as well in watershed mode.

The watershed algorithm is initialized with a collection of "seed" vertices–one set per activation peak. These seeds are enlarged as the watershed proceeds until they take up the entire search space. By default, seed

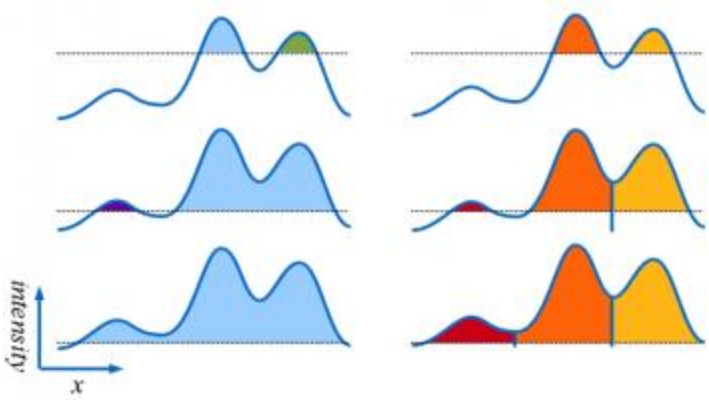

Figure 6.1: Illustration of the watershed algorithm. Colors represent separate watershed regions. A threshold is progressively lowered, starting with the max statistic over the search space. Regions are "colored" as the dropping threshold uncovers the tops of activation peaks. Region boundaries are drawn where the colors meet.

locations are computed from the statistic map automatically. However, you can also specify the seeds manually.

It is possible to save the seed vertex numbers in a text file, and pass the file to `make_roi` using the `-s <seeds>` option. For example:

```
make_roi watershed --search IFG.lh.shape.gii -p 10  -s seeds.txt \
  zstat_S-NW.lh.func.gii \
  IFG_S-NW_perc20_watershed.lh.shape.gii
```

In the seeds file, each row contains a list of vertices corresponding to a single seed:

```
1113 9460 12347 29001
2108 872 17030
7061 43 18380 5005
```

Another option is to pick the seeds manually during the execution of `make_roi` by passing the `--man-seeds` option, instead of `-s <seeds>`. When you run `make_roi watershed` with `--man-seeds` enabled, a `tksurfer` window displaying the statistic map will pop open. You'll also see a prompt in the command window, asking you to specify the vertices for the first seed.

1. Click around the `tksurfer` window to find the 1 to ∼5 vertices concentrated around the most prominent activation peak.
2. Enter the vertex numbers at the prompt and press `<Return>` when finished.
3. Continue in the same way for the remaining activation peaks in the search space.
4. Once you've entered seed vertices for all of the prominent peaks, press `<Return>` on a blank prompt to complete the seed definition.

6.4 Circular ROIs

It is also possible to define simple circular ROIs using `make_roi` in "circle" mode. Circular ROIs contain all vertices within some fixed distance of a given vertex: the circle center. The radius of the circle is given in mm, using the `-r <rad>` option. The center of the circular ROI can be given as either a vertex number, or an (X, Y, Z) coordinate in MNI 152 space, using the `-c <coord>` option. For example:

```
# 10mm circular ROI with vertex number center.
# Note that no statistical map is necessary.
make_roi circle --search IFG.lh.shape.gii -r 10 -c 4307 \
  IFG_circle1.lh.shape.gii

# Circular ROI with (X, Y, Z) coordinate center.
make_roi circle --search IFG.lh.shape.gii -r 10 -c 10,20.3,-19.5 \
  IFG_circle2.lh.shape.gii
```

Note: Circular

6.5 Full `make_roi` usage

The behavior of `make_roi` can be further modified by passing command-line options. A full list of available options, plus their descriptions are included below:

```
--fs-sub <subject>   Freesurfer subject in SUBJECTS_DIR [default: 32k_fs_LR].
--hemi <hemi>        Hemisphere (lh or rh) [default: lh].
--surf <surface>     Surface name [default: midthickness].
--search <mask>      Seach space mask in Nifti or metric Gifti format. Must
                     be in same space as stat map and surface. (E.g. --search
                     lh.IFG.shape.gii.) [default: cortex]
--lit-coord <coord>  Measure and report distance from ROI peak to literature
                     or group average coordinate. Coordinate can be X,Y,Z or
                     vertex index (E.g. --lit-coord=20.,19.,-5.).
--peak-thr <thr>     Threshold for finding distinct peaks of activation in
                     watershed mode. Each connected cluster above this
                     threshold will give rise to a distinct ROI. Can be
                     overridden by including a seed file or giving
                     the --man-seeds flag. [default: 3.0]
--eval <metric>      Metric to use when weighing ROI candidates. Options are
                     "strength" (sum of stat values), "dist" (weighted
                     average distance from literature coord to ROI),
                     "strength / dist" (strength / distance), or "strength /
                     spread" (strength / weighted average distance from ROI
                     peak to all other ROI vertices) [default: strength].
--select <mode>      ROI selection mode. Options are: "best" (the single best
                     ROI will be produced), "pick" (you will choose the ROI
                     from the available options interactively), or "all" (all
                     available ROIs will be produced) [default: best].
-h, --help           Display this message.
```

CHAPTER

SEVEN

ROI EXTRACTION

7.1 Overview of ROI extraction

This chapter describes the procedure for extracting percent signal change (PSC) from a region of interest (ROI). The purpose of ROI extraction is to enable researchers to ask whether a specific brain region responds to a task manipulation. A good overview of the motivations and methods behind ROI extraction is given on the MIT Mindhive[1] website, specifically here[2] and here[3]. In short, there are three steps to ROI extraction.

1. **Calculate the average time-series within an ROI, across a series of functional runs.** This average time-series is represented as an *N* dimensional vector, where *N* is the total number of time-points across all the runs.
2. **Analyze the average time-series as a function of experimental condition.** The result is a time-series showing the estimated hemodynamic response associated with each condition, in PSC units.
3. **Extract an estimate of the peak PSC response for each condition from the PSC time-series.** This is a single number representing the magnitude of the response to the condition in the ROI.

7.2 Step 1: Calculating the average ROI time-series

Given a set of functional runs and an ROI, the goal of Step 1 is to produce a single functional time-series representing the average activity across the ROI. You first compute an average time-series for each run separately, and then concatenate the individual run time-series' to form the "session average" time series.

There is one important issue to be careful of when computing the ROI time-series. Between runs, the average level of activation across time will vary. If not addressed, the between-run variability will corrupt your PSC estimates. Thus, you must scale each run's average time series to have the same mean (e.g. 1000) before concatenating. This is called *grand-mean scaling*.

Note: There is a question of order of operations here. Should you first compute the average time series (across the ROI) for each run, and then grand-mean scale? Or should you grand-mean scale every voxel's time-series in every run, and then compute the spatial average. I prefer the second option (and this is what's implemented in our script). With the first option, the average time series is biased to brighter (more active) voxels. Since the overall brightness of a voxel is not informative of brain activity, it is best to remove this bias.

There are a few other things you could do to deal with between-run variability better. For example, you could regress out a linear trend from each run's time-series, or high-pass filter to remove slow changes in

[1] http://mindhive.mit.edu/imaging
[2] http://mindhive.mit.edu/node/101
[3] http://mindhive.mit.edu/node/86

activity within a run. However, if you're data have been high-pass filtered during preprocessing, these extra steps are unnecessary.

Note: Some software packages (e.g. SPM's MarsBar) recommend extracting the "first eigenvariate" for an ROI, rather than the average time series. The first eigenvariate approach is described in [Saxe2006], on page 1094. It is intended to handle inhomogenous ROIs better (where there may be subgroups of voxels activating in different ways). It is unclear to me whether and how the first eigenvariate approach accomplishes this goal...

7.3 Step 2: Estimating the HRF for each condition

There are at least three ways you can go about step 2. The three options have different assumptions, and are appropriate in different circumstances. They also differ in their complexity.

7.3.1 Method 1: classic selective averaging

Selective averaging is the oldest and most straight-forward technique for extracting PSC. It was originally used in ERP analysis, and was first described for fMRI analysis in [Dale1997]. The goal is simply to compute the mean PSC time-series for each condition. There are four steps involved in selective averaging.

1. **Compute an estimate of the "baseline" signal across the time-series.** The baseline is a single number representing the average BOLD response during rest. In order to calculate an accurate baseline, your design must contain sufficient rest periods, uncontaminated by task responses. It is up to you to decide which time-points should count as rest. But you should be sure to take into account the slow response of the HRF. We usually assume that a stimulus' HRF takes off around 4 seconds after onset, and continues until 14 seconds after offset. Rest usually includes time-points outside this window.
2. **Scale the average BOLD time-series so that it's in PSC units relative to baseline.** I.e. so that a value of 2.0 reflects a 2% increase in signal over rest. The formula for this transformation is:

$$\text{PSC} = 100 * \frac{\text{BOLD} - \text{baseline}}{\text{baseline}}$$

3. **For each condition, splice the PSC time-series into several mini time-series'–one for each condition event.** Event time-series' should be time-locked to the onset of the event, and should all be the same length. Stack the time-locked time-series' into an *N* x *T* matrix, where *N* is the number of condition events across the experiment, and *T* is the number time-points in a single event.
4. **Compute the average hemodynamic response time-series for each condition**. You will average the matrix of event time-series' across the columns.

One of the main benefits of selective averaging is that it's very easy to compute. Second, it makes no assumptions about the shape of the hemodynamic response–it's just an average of the raw data. For these reasons, people say that selective averaging keeps you "close to the raw data". This is in contrast to standard whole-brain GLM methodology, where your analyses involve complicated statistical machinery.

There is one key problem associated with selective averaging. When adjacent trials are not well-separated by rest, their hemodynamic responses will overlap. The observed BOLD response at these overlaps will then be the sum of the two trial responses [4].

Ideally, our method for extracting PSC would handle this situation by dividing the summed response among the two trials. However, the selective averaging procedure does not do this. Instead, it double-counts the

[4] This is an important and somewhat debatable assumption in MRI. See [Boynton1996] and [Dale1997] for discussion.

overlap signal 100% for both trial types. This makes the response estimates for the beginnings and ends of trials (where overlaps occur) inaccurate. Usually the response will be over-estimated, since the true responses at the beginnings and ends of trials are often above baseline.

For example, the measured response estimate for the last time-point in condition *A* will equal the true average response for condition *A* at this time-point, **plus** the average response at the beginning of every trial *A* overlapped with.

Important: Selective averaging is thus not appropriate for designs with significant trial overlap.

7.3.2 Method 2: Finite-impulse response modeling

The finite-impulse response (FIR) model is an extension of selective averaging designed to deal with the problem of trial overlap. Importantly, when there is no overlap between trials, it will produce identical results as selective averaging. When there is overlap, it tries to attribute the PSC response to the overlapping conditions sensibly.

The procedure for doing FIR extraction is more complicated than selective averaging however. To understand it you need to be comfortable with multiple linear regression. If the terms "design matrix", "predictor", "beta", or "normal equations" are unfamiliar to you, you should review background on linear regression (E.g. Appendix A in [Poldrack2011]). With this in mind, the steps of FIR extraction as follows:

1. **Define a linear model that includes a bank of "finite-impulse" predictors for each condition.** For each condition, you will include as many predictors as there are time-points in a single condition event. Each predictor models a specific time-point in the PSC time-course. For example, the 9$^{\text{th}}$ predictor for condition *A* will estimate the response for the 9$^{\text{th}}$ time-point in condition *A*, across presentations. This predictor will be 1 on every 9$^{\text{th}}$ TR for condition *A*, and 0 elsewhere.

   The overall design matrix will have a shape: $N \times (D*T+1)$, where *N* is the total number of time-points in the data-series, *D* is the number of conditions in the experiment, and *T* is the number of time-points in one trial. The added 1 reflects the inclusion of an intercept predictor. Assume that the first predictor is the intercept, followed by the finite-impulse predictors for condition *A*, condition *B*, etc.

2. **Fit the linear model with ordinary least squares, by solving the normal equations:**

$$\mathbf{y} = \mathbf{X}\beta + \epsilon$$
$$\hat{\beta} = (\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \mathbf{y}$$

   Here $\mathbf{y}$ is the complete BOLD time-series, and $\mathbf{X}$ is the design matrix. The fitted beta $\hat{\beta}$ is a vector of length $D*T+1$, containing the parameter estimates that minimize the sum of squared errors ($\|\epsilon\|_2$). $\hat{\beta}[i]$ is the parameter estimate for the i^{th} predictor in the design matrix. Thus, $\hat{\beta}[0]$ is the intercept estimate, while $\hat{\beta}[d*T : (d+1)*T]$ contains the estimated finite-impulse responses for the d^{th} condition. Note that based on how we set up the design matrix, this is in fact the estimated BOLD time-series for the d^{th} condition.

3. **Tranform the estimated BOLD time-series' into PSC.** To do this we must determine the baseline, and scale the BOLD time-series' as in step 1 of the selective averaging procedure. However, we do not need to separately compute the baseline, as above. In linear regression, the intercept models the data average when all the other predictors are zero. Since the finite-impulse predictors are non-zero only during condition events, this means the intercept in our FIR model is actually the baseline.

   Furthermore, the BOLD time-series' derived from the beta vector are in units of $(\text{BOLD} - \text{intercept})$. This means we can use the following formula to transform the beta vector into PSC:

$$\hat{\beta}_{\text{PSC}}[1:] = 100 * \frac{\hat{\beta}[1:]}{\hat{\beta}[0]}$$

4. **Reorganize the scaled beta vector in separate PSC time-series' for each condition.** This just means splitting $\hat{\beta}_{\text{PSC}}$ into the condition PSC slices $\hat{\beta}_{\text{PSC}}[d * T : (d+1) * T]$.

The reason FIR extraction handles overlaps better than selective averaging is that in cases of trial overlap, both conditions' finite-impulse predictors will be "active". Thus they will "compete" with one another to explain the observed response. The linear regression framework enforces that the overlapping predictors *add up* to fit the observed data as close as possible. As a result, FIR extraction *disentangles* the observed response, assigning to each condition the amount of PSC owed to it.

Importantly, the FIR extraction method is just as flexible as the selective averaging method. It makes no assumptions about the shape of the HRF. It is free to capture any variation observed in the data. This is due to each time-point within a trial being modeled independently.

However, the FIR method is significantly more complicated than selective averaging. Thus, it is important to be careful with your implementation, and to check your results against those produced by selective averaging.

I argue that the added complexity is only a practical problem though. There is no theoretical reason to prefer the selective averaging method for its relative simplicity. Even though selective averaging looks very different from FIR extraction, it is really just a bad implementation of the same analysis. When you use it, you're not actually doing something theoretically simpler–you're just doing FIR extraction incorrectly.

To see why this is, note that if instead of carrying out FIR extraction as a multiple regression, you regressed each predictor separately, you would get *exactly* the same results as selective averaging. It is just a different way of framing the same computation. Fitting the finite-impulse predictor corresponding to the 9^{th} time-point for condition *A* gives you *exactly* the same value as taking the average of all the 9^{th} time-points. Framed in this way, we see that selective averaging amounts to doing repeated simple linear regressions, when really a multiple regression is needed. This is a common mistake in data analysis. It results in over-estimated effects when predictors are correlated (e.g. when trials overlap).

Important: You should therefore **always** use FIR extraction over selective averaging, assuming you are confident in the FIR implementation. When there is no trial overlap the two methods give you the same results. When there is overlap, selective averaging is *invalid*.

7.3.3 Method 3: HRF basis function modeling

We mentioned above that selective averaging and FIR regression are both *flexible* methods for estimating hemodynamic responses. They make no assumptions about what the true shape of the HRF should be. Hence they are able to detect any conceivable HRF with equal fidelity.

However, this flexibility also results in more variable estimates. With the FIR model, there is a free parameter for each trial time point. Hence, relatively few data points contribute to each parameter estimate. This makes for more variable within-subject HRFs.

Alternatively, if you shrink the number of parameters by imposing some constraints on shape of the HRF, you can reduce the variability of the estimates. You must be careful however, since the constraints you choose may bias you against detecting some valid HRF shapes.

This is called the *bias-variance* tradeoff. It is explained well in chapter 5 of [Poldrack2011]. The FIR method is at one extreme of the spectrum, while the canonical double-gamma HRF approach common to whole-brain analysis is at the other extreme.

An intermediate choice is to use a set of HRF basis functions to model an ROI's hemodynamic response. The hope is that the HRF basis functions you use *span the space* of possible HRFs [5]. When this is true, there will only be reduced flexibility for fitting noise.

[5] The FLOBS HRF basis functions used by FSL were generated with exactly this in mind. The three basis functions are in fact the first three principle components derived from a large dataset of empirical HRFs, obtained from different subjects and regions.

The procedure for using HRF basis functions during PSC extraction is as follows:

1. **Run a standard GLM analysis on the ROI time-series, using the HRF basis functions.** The HRF basis function approach to PSC extraction is built on the same GLM modeling framework used in whole-brain analysis. We model the hemodynamic response for a given condition as a weighted sum of HRF basis functions, after those basis functions have been convolved with a predicted neural time-course.

   The neural time-course vectors are generated from the same timing files used during single-subject whole-brain analysis. They have high temporal resolution (.05 sec) to match the HRF basis functions. Usually they represent simple on-off boxcar functions (1 during all condition events, and 0 otherwise).

   They weights are computed using standard linear regression. Before regression, the convolved predictors are down-sampled to match the temporal resolution of the ROI time-series (~2 sec). An intercept is also included in the linear model. This will enable us to compute the baseline activation, and transform to PSC units as in FIR extraction.

2. **Use the GLM weights to generate the estimated HRF time-series'** If we take the HRF basis predictors plus the parameter estimates for one condition and matrix-multiply them, we will get the modeled BOLD time-series for that condition, across the entire experiment. What we want instead is the BOLD time-series corresponding to a single trial, and time-locked to trial onset. To get this you can just "crop" the complete BOLD time-series so that only one trial is left. Since all trials have the same duration (a requirement for ROI analysis), it doesn't matter which one you pick.

3. **Use the model intercept to transform the HRFs to PSC.** Lastly, we can use the modeled intercept to transform the BOLD HRFs into PSC units:

$$\mathrm{HRF}_{\mathrm{PSC}} = 100 * \frac{\mathrm{HRF}_{\mathrm{BOLD}}}{\mathrm{Baseline}}$$

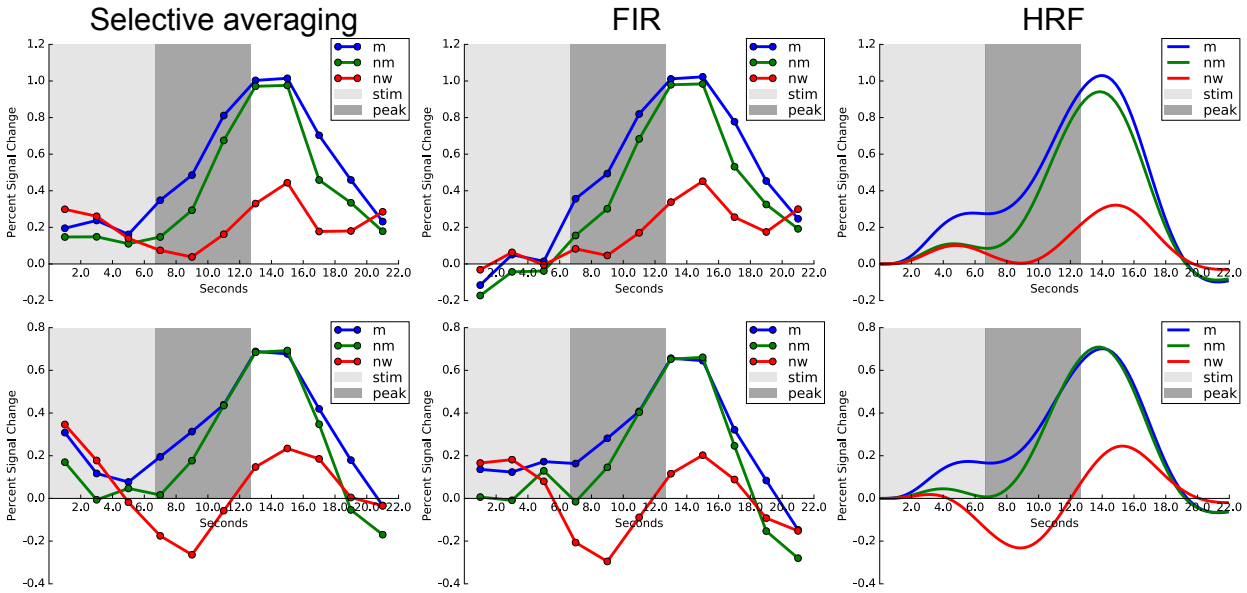

Figure 7.1: Comparison of selective averaging (left), FIR (middle), and HRF basis function (right) PSC extraction methods (right). Two subjects' PSC curves from an inferior frontal ROI are shown. The conditions are +MOVE sentences (m), –MOVE sentences (nm), and nonword sequences (nw). Note that the PSC values are consistently higher for the selective averaging curves near the start and finish. Also note how the HRF basis function PSC plots look like smoothed versions of the FIR PSC plots.

7.4 Step 3: Peak PSC estimation

The final step in PSC extraction is to calculate the peak PSC for each condition. This step is important because peak PSC is typically what we perform our statistics on.

Peak PSC is calculated by taking an average of the PSC time-series over a *peak window*–a time window during which the HRF is likely to peak. The tricky part is determining what the peak window should be. On the one hand, you could choose a single time-point as your peak. Alternatively, you could select the entire trial window as the "peak".

By choosing a smaller window, you are biasing yourself against detecting PSC effects outside the chosen window. We know that the shape of the HRF changes between regions, and across people. If you pick a very small peak window, you will likely miss most subjects' peaks. In addition, smaller peak windows result in more variability, since less data contributes to the measurement.

On the other hand, a larger peak window will result in peak estimates that are less noisy, and more robust to inter-subject differences. However, a larger window can also mean smaller effects, since PSC changes (e.g. differences between conditions), are often much smaller at the beginnings and ends of trials (when the response is near baseline).

Another choice to make is where to center the peak window (in addition to how wide it should be). One option is to base this choice on the canonical model of the HRF. The predicted peak time-point is the time at which the canonical HRF peaks (after convolving with the predicted neural time-course for your condition).

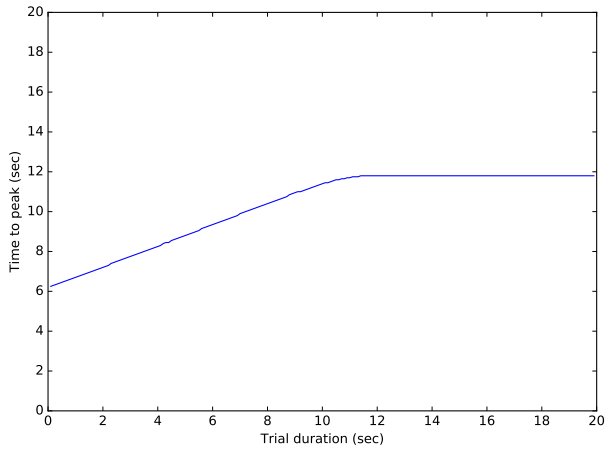

Figure 7.2: Relationship between trial durations and peak time-point (assuming the canonical HRF). Note that after ~11 sec, the peak time-point is constant. This curve is approximated as piece-wise linear function in `roi_extract` in order to decide the default peak window.

7.5 ROI extraction with `roi_extract`

Our tool for performing ROI extraction is a custom Python script called `roi_extract`. The tool is used for extracting PSC for a single subject and ROI. The basic usage pattern for `roi_extract` is:

```
roi_extract [options] --roi=<roi> --runs=<runs> --cond=<timing>... --out=<outdir>
```

The key arguments include:

```
--roi=<roi>         Binary mask image of ROI (.gii or .nii.gz)
--runs=<runs>       List of functional runs in same space as ROI mask (.gii or
                    .nii.gz format). The list must be delimited by spaces and
--cond=<timing>     List of timing files for a given condition, delimited by
                    spaces and enclosed in quotes. The condition label must
                    precede the list, and be followed by a colon. There must
                    be a timing file for each run given, and the two lists
                    must be in corresponding order. This is a repeatable
                    argument; you should give a --cond argument for each
                    condition that you'd like to model. Timing files should be
                    in FSL's 3-column format.
--out=<outdir>      Output directory. Will be created if it doesn't exist, or
                    overwritten if it does.
```

For example, you could run something like this to extract PSC from an inferior frontal ROI, for a single subject:

```
roi_extract --roi=IFG.lh.shape.gii \
  --runs="bsyn_01.lh.func.gii bsyn_02.lh.func.gii bsyn_03.lh.func.gii bsyn_04.lh.func.gii" \
  --cond="m:bsyn_01-m.txt bsyn_02-m.txt bsyn_03-m.txt bsyn_04-m.txt" \
  --cond="nm:bsyn_01-nm.txt bsyn_02-nm.txt bsyn_03-nm.txt bsyn_04-nm.txt" \
  --cond="nw:bsyn_01-nw.txt bsyn_02-nw.txt bsyn_03-nw.txt bsyn_04-nw.txt" \
  --out=BSYN_S_01/roi/IFG_bsyn.lh
```

Of course, this example is only for illustration. Usually the paths for the functional data and timing files will be much longer, if you're following the lab conventions for analysis organization. For a more realistic example, you might use a little snippet like this to *construct* the `roi_extract` command-line (which is often very long).

```
studydir=/media/BednyDrobo/BSYN
cd $studydir

roi=ROI/IFG.lh.shape.gii

subj=BSYN_S_01
hemi=lh
task=bsyn
outdir=$subj/roi/IFG_bsyn.lh

# Glob list of preprocessed functional runs.
runs=$(echo $subj/preproc/${task}_??.feat/${hemi}.32k_fs_LR.surfed_data.func.gii)

# Build list of --cond arguments
conds="m nm nw"
condargs=
for cond in $conds; do
  # Glob list of timing files.
  timings=$(echo $subj/timing/${task}_??-${cond}.txt)
  # Add --cond argument to growing list.
  condargs="$condargs --cond=\"$cond:$timings\""
done

# Run roi_extract
roi_extract --roi=$roi \
  --runs="$runs" \
  $condargs \
  --out=$outdir
```

7.6 roi_extract outputs

The roi_extract output directory contains the following files:

```
|- log.txt
  * roi_extract log containing the command-line, warnings, errors, as well as
    extra information such as the number of data voxels found in the ROI.
|- roi_ts.txt
  * ROI time-series, after mean-scaling, averaging across the ROI, and
    concatenating across runs.

|- timing_info.csv
  * Descriptive information on the trial sequence, such as the number of
    trials per condition, and the average onset relative to TR start.

|- classic_tbt_results.csv
  * Trial-by-trial selective averaging results, arranged as a csv table.
|- classic_results.csv
  * Selective averaging results, after averaging across trials for each
    condition.
|- classic_psc.pdf
  * Plotted selective averaging results, with stimulus and peak windows
    highlighted in gray.

|- fir_design.txt
  * FIR design matrix in plain text format.
|- fir_beta.txt
  * FIR beta vector as a single column of numbers.
|- fir_results.csv
  * FIR extraction results including peak PSC and PSC time-course for each
    condition, arranged in a csv table.
|- fir_psc.pdf
  * Plotted FIR results.

|- hrf_design.txt
  * HRF design matrix in plain text format.
|- hrf_beta.txt
  * HRF beta vector as a single column of numbers.
|- hrf_results.csv
  * HRF extraction results including peak PSC, beta values, and PSC
    time-courses for each condition, arranged in a csv table.
|- hrf_psc.pdf
  * Plotted HRF results.
```

CHAPTER

EIGHT

RUNNING ANALYSES IN PARALLEL

8.1 Overview of parallel processing

One of the benefits of doing analyses in a shared server environment like Arwen is that we have access to much more processing power than any single personal computer can offer (See [Poldrack2011], appendix B.1). To leverage this extra computing power, we often submit batches of analysis jobs to run in parallel, rather than one at a time. For example, we might run preprocessing on all of a new subject's data at once.

We use GNU Parallel[1] to carry out our parallel processing. GNU Parallel is a well-developed tool with extensive options. You can read through the manual by running `man parallel` in the Arwen terminal. However, I've found that only the most basic usage is necessary.

First, you need to make a "job file": a plain text file containing each command you want to run on a separate line. For example, you could have a toy job called `job.txt` that looks like this:

```
echo 1
echo 2
echo 3
echo 4
echo 5
echo 6
echo 7
echo 8
echo 9
echo 10
```

To submit these jobs as a batch, you would run:

```
parallel --jobs 10 < job.txt
```

Here the `--jobs 10` flag sets how many jobs you want to run in parallel. 32 is the maximum you should ever use, since that is the number of processing cores on the server. In general though, submitting 32 jobs is bad practice, since it leaves no processing power left over for other users or general overhead. As a rule, 10 to 28 jobs are best, depending on the server load.

The expression `< job.txt` is known as "input redirection". It tells parallel to read the list of parallel jobs from the file `jobs.txt`.

A few other points about parallel jobs:

- If you ask for more jobs than are in your job file, all of your jobs will run at once.
- If you ask for fewer jobs than are in your job file, `parallel` will work through the job list in equal size batches.

[1] https://www.gnu.org/software/parallel/

- Jobs are not necessarily run in the order listed in the job file. So it's important not to have one job depend on the output of another one.
- You can use the Linux command `top` to estimate how many jobs you should run. `top` shows you in real time what is being run by other users. If you see that another user has already submitted a large batch of jobs, don't overwhelm the system by submitting a large batch yourself.

8.2 Using scripts to generate job files

Although it is possible to create job files by hand–by typing out each command you want to run one by one–it is often easier to create them using short shell scripts. When you run a parallel job, you're usually trying to do something like: "run preprocessing on every run from subjects 1-3". This kind of job can be expressed easily in a script using "for" loops. For example:

```
# Loop through the list of subjects.
for subject in BSYN_S_01 BSYN_S_02 BSYN_S_03; do

  # Loop through each run of data for the current subject
  # The expression bsyn_*.nii.gz matches e.g. bsyn_01.nii.gz, bsyn_02.nii.gz, bsyn_99.nii.gz
  for run in $subject/raw/bsyn_[0-9][0-9].nii.gz; do

    # Figure out the output directory, based on the name of the run.
    # E.g. BSYN_S_01/raw/bsyn_01.nii.gz --> BSYN_S_01/preproc/bsyn_01
    # The expression ${run%.nii.gz} strips off the trailing ".nii.gz" from the file name.
    # The function basename prints the last file/folder name in a path. E.g. A/B/c.txt --> c.txt
    # Finally, the $(...) expression stands for command-substitution. The command inside the
    # parantheses is replaced with its output.
    outdir=$subject/preproc/$(basename ${run%.nii.gz})

    # Print the preproc command to the job file.
    # The ">> job.txt" is called output redirection. The result is that the command is appended
    # to the job file.
    echo "preproc $subject $run $outdir" >> job.txt
  done
done
```

In the lab we call this kind of script a "make-job" script. In the NPDL scripts directory (`$NDPL_SCRIPT_DIR`), you'll find a set of template make-job scripts under `etc/template_scripts`. When you run parallel analyses, you can copy these templates to your study folder, and modify them for your specific purposes.

The standard practice is to have a Jobs folder in your study directory. The purpose of the Jobs folder is to organize all of the parallel jobs you run throughout your analyses. If you organize it well, it will serve as an enduring record of everything you did to your data. This is *very helpful* when you go to write up analysis methods. For example, one possible organization could be:

```
Jobs/
  |- preproc/
    * All preprocessing jobs.
    |- preproc_01_090115/
      * sub-folder for first preprocessing job.
      |- make_job.sh
      |- job.txt
      |- log.txt
  |- firstlevel/
    * All firstlevel jobs.
    |- firstlevel_01_090115/
      |- make_job.sh
```

```
     |- job.txt
     |- log.txt
|- roi/
  * All roi creation and extraction jobs
```

CHAPTER

NINE

ONE-OFF TIPS AND TRICKS

9.1 Transferring data to and from Arwen

There are two standard ways of transferring data to Arwen: (1) using the command-line tool `scp`, or (2) by mounting the Arwen filesystem as an SMB share.

The syntax for `scp` is like a combination of `ssh` and `cp`.The basic usage is:

```
scp file1 user@pbs-mb-arwen.pbs.jhu.edu:file2
```

Setting up Arwen as a SMB share is trickier at first, but probably more convenient in the long-run. SMB is a Windows protocol for mounting a remote filesystem. It is managed on the server-side using the `samba` package. Configuration settings on the server, which list the folders available for mounting, and the allowed users are stored in `/etc/samba/smb.conf`.

You do not need to worry much about the server-side however. All you need to do to get SMB working is to ask the system administrator to add you as a new SMB user. This involves running the following command on Arwen:

```
sudo smbpasswd -a user
```

Where `user` is your JHED ID. You will be prompted to enter your password–you should enter the same JHED password you use to login to Arwen. Next, the system admin will need to restart the SMB daemon:

```
sudo service smbd restart
```

You should now be able to open an SMB commection. On a Mac, you can use the "Connect so Server" function in the Finder (Command-K in a Finder window). For the address, you should enter `smb://user@pbs-mb-arwen.pbs.jhu.edu`. Enter your JHED ID and password, and choose which folder(s) to mount. If you're using a different OS, you'll need to look up platform-specific instructions.

Once you've opened the SMB connection, you should be able to move files back and forth between your computer and Arwen, as if it were part of your local file-system.

Note: Once you set your SMB password initially, it should auto-update whenever your system password changes. This means you don't have to re-enter it whenever you change your JHED password.

9.2 Automatically taking screenshots

We often need to take screenshots of brain images from `tksurfer` or `freeview` when making figures. Of course, you can always do this manually. However, there is also the option to take automatic screenshots

with `freeview`, using the lab function `takesnap`. This is especially helpful when you have a lot of screenshots to take.

`takesnap` is a Bash function defined in `$NPDL_SCRIPT_DIR/lib/npdl_funcs.sh`. Below is the help message, which you can also access by typing `takesnap -h` in the Arwen terminal:

```
Usage: takesnap [options] <subj> <hemi> <surf> <view> <out-png>

Take a snapshot of a surface & overlay using freeview.

Arguments:
  <subj>        Freesurfer subject in /media/BednyDrobo/Tools/NPDL-scripts/subjects
                (e.g. 32k_fs_LR).
  <hemi>        lh or rh.
  <surf>        Anatomical surface to load (e.g. hcp_very_inflated).
  <view>        lat, med, inf, sup, post, or front.
  <out-png>     Path to output png.

Options:
  --ov=<overlay>     Statistical overlay to load.
  --annot=<annot>    Annotation file to load.
  --thr=<min,max>    Min, max threshold for overlay. [default: 2.33,7.0]
  --size=<l,w>       Size of freeview window/image. [default: 800,575]
  --zoom=<frac>      Zoom fraction. [default: 1.75]
```

The usage pattern in intentionally similar to `tksurfer`, only with two added arguments: `<view>`, and `<out-png>`. There are also options for controlling the statistical overlay, plus the size and zoom of the image.

Note: `tksurfer` also has an option for taking automatic screenshots (`-snap <filename>`), however it doesn't seem to work right in our installation. This might be something to troubleshoot in the future.

9.3 Automatically cropping images

After saving a screenshot, the image is rarely cropped tightly enough for use in a publication-quality figure. To avoid having to crop the images manually, we use a lab Bash function called `imgtrim`, defined in `npdl_funcs.sh`. `imgtrim` also has the ability to make the backgroud of an image transparent.

```
imgtrim [Options] <img> <out-img>

Trim background to fit image. Optionally make bg transparent.

Arguments:
  <img>         Path to input image.
  <out-img>     Path to output image. Use extension to specify format.

Options:
  --no-bg=<bg-color>    Make background transparent. Accepts the same
                        color fomats as ImageMagick (e.g. 'white',
                        '#00ff00', 'rgb(255,0,0)')
```

`imgtrim` relies on a useful image processing tool called ImageMagick[1].

[1] http://www.imagemagick.org/script/index.php

9.4 Make an animated Gif from a volumetric data series

It is sometimes useful and fun to display a functional data series as an animated Gif image. For example, in the `preproc` quality assurance report. There is a lab Bash function `make_data_gif` which accomplishes this.

```
make_data_gif <func-data> <out-gif>

Make an animated gif for one run of functional data.
```

`make_data_gif` will generate an animated Gif showing the middle axial, coronal, and sagittal slice over time. Volume numbers are also shown at the bottom of the image.

`make_data_gif` uses FSL's `slicer` tool to convert a Nifti data series into png files, and ImageMagick to string the png files into an animated Gif.

9.5 Getting information about an image

There are several tools available for looking up information about an MR image's parameters.

- FSL's `fslinfo` gives a brief synopsis of a Nifti image's header. E.g. The extent of the image in x, y, z, and t dimensions, and the voxel size.
- FSL's `fslhd` prints the entire header for a Nifti image.
- Freesurfer's `mri_info` is a general tool for retrieving information about nearly any imaging file format.
- The HCP's `wb_command -file-information` is useful for displaying detailed information about Gifti images.
- Finally, you can manually view the `.par` files associated with raw data in a standard text editor.

9.6 Converting between file formats

There are many tools available for converting between imagining file formats.

- Use Mricron's `dcm2nii` for convertion DICOM or PAR/REC images to Nifti format.
- Freesurfer's `mri_convert` can be used to convert nearly any volumetric data format into any other.
- Freesurfer's `mris_convert` can be used to convert between surface data formats. E.g. between .mgh and .gii.
- The HCP's `wb_command -metric-convert` can be used to convert between metric Gifti and Nifti format.
- Freesurfer's `mri_cor2label` can be used to convert metric Gifti or mgh files representing binary ROIs into Freesurfer labels.
- The lab function `label2mask` can be used to go from labels to metric Gifti ROI.

9.7 Concatenating images in time

To concatenate multiple 3D Nifti volumes in time (or along any other dimension), use FSL's `fslmerge`. You can also concatenate separate metric Gifti maps using the HCP's `wb_command -metric-merge`.

9.8 Image arithmetic and statistics

Performing arithmetic or statistics on images is one of the most common analysis operations.

- You can use FSL's `fslmaths` for doing arithmetic and higher mathematical operations on Nifi images. E.g. adding two images, subtracting images, thresholding, binarizing.
- Many of the same operations can be carried out on metric Gifti files using the HCP's `wb_command -metric-math`.
- Across-voxel statistics on Nifti images, e.g. mean, sum, max, can be performed with FSL's `fslstats`.
- Similar operations can be done on metric Gifti files using the HCP's `wb_command -metric-stats` and `wb_command -metric-reduce`.
- Lastly, you can use `wb_command -metric-weighted-stats` to compute metric statistics, weighted by surface area. One useful example is computing the surface area of a binary ROI.

All of the following jobs can be done with image arithmetic or statistics:

- Adding, multiplying, subtracting, dividing images (`fslmaths` or `wb_command -metric-math`)
- Scaling or shifting the values in an image (`fslmaths` or `wb_command -metric-math`).
- Converting between p values and $-log_{10}(p)$ (`fslmaths` or `wb_command -metric-math`).
- Converting between p and z values (`fslmaths` or `wb_command -metric-math`).
- Thresholding an image (`fslmaths` or `wb_command -metric-math`).
- Binarizing an image (`fslmaths` or `wb_command -metric-math`).
- Temporal filtering with a data series (`fslmaths`).
- Collapsing across the temporal dimension in a data series (e.g. by taking mean, max, min, etc.) (`fslmaths` or `wb_command -metric-reduce`).
- Computing summary statistics of an image across voxels, e.g. max, min, mean, stdev (`fslstats` or `wb_command -metric-stats`).
- Computing the surface area of a binary ROI (`wb_command -metric-weighted-stats`).

9.9 Loading data in Python or MATLAB

Sometimes, the mathematical functions provided by FSL or the HCP are not enough. In these cases, you may want to load Nifti or Gifti images into MATLAB or Python, for more flexible computation. There are several libraries available for doing this:

- FSL provides several useful MATLAB functions for working with Nifti files in `$FSLDIR/etc/matlab`. In particular, `read_avw.m` lets you read a Nifti file into matlab as an array.
- The MATLAB Gifti library[2] allows you to do the same thing with Gifti files.
- For Python, we can use the lab function `img_read`, found in the `npdl_utils` module. This function can load either Nifti or Gifti images. It is built around the more general Nibabel library[3].

[2] http://www.artefact.tk/software/matlab/gifti/
[3] http://nipy.org/nibabel/

9.10 Projecting data between volume and surface

We sometimes need to project data from the volume space to the surface space, or vice-versa. This is different than simply converting the file-format. In involves actually changing the structure of the data.

When thinking of projecting data between volume and surface, it helps to think of the midthickness surface outline overlaid on an anatomical MR image. When projecting from volume to surface, a surface vertex is filled in with an average of the volumetric data directly above and below it (along a cylinder in its "normal" direction). When going the opposite direction, the save voxels are filled in with the vertex's value.

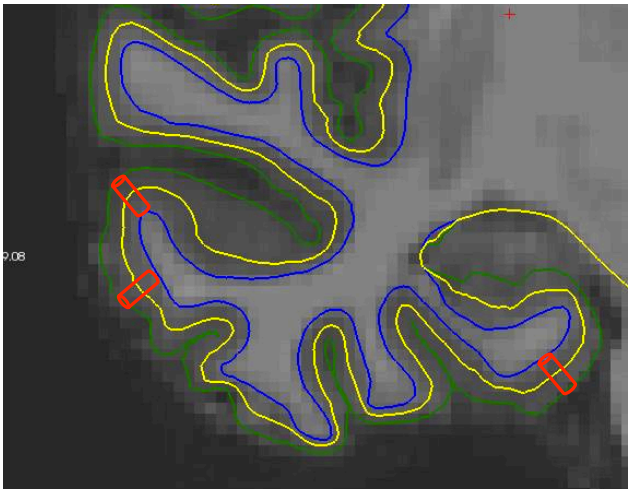

Figure 9.1: A region of left temporal cortex, overlaid with the white surface (blue outline), the midthickness surface (yellow outline), and the pial surface (green outline). At several vertices the "normal" cylinders are shown in red, which dictate how data is projected between the surface and volume.

You can use either Freesurfer's `mri_vol2surf`, or the HCP's `wb_command -volume-to-surface-mapping` for projecting from the volume to surface. For the opposite direction, you can use either Freesurfer's `mri_surf2vol` or the HCP's `wb_command -metric-to-volume-mapping`.

One special case is when you're trying to map a volumetric statistical map or ROI in MNI 152 space to the surface. For this, it's best to use `mri_vol2surf`, being sure to select the `--mni152reg` option. The reason is that Freesurfer knows the mapping between MNI 152 and fsaverage (MNI 305). If you use a different method, it's less clear how to align the volume and surface space before mapping.

Note: An important point is that the volume and surface data must be aligned prior to projection. I.e. when you overlay the source volue and target surface in Freeview, they must line up. Aligning a volume to surface space requires computing a volume transformation to the surface subject's `T1.mgz` or `brainmask.mgz` image. This is discussed more in another section.

9.11 Resample a surface or metric from one surface space to another

It is possible to convert an anatomical surface or metric from one surface space (mesh) to another. This can be useful for example if you have data in a subject's space, but want to resample it to 32k_fs_LR.

Performing this resampling only requires knowing how to associate the vertices in the source space with those in the target. You use this association to send data from a source vertex, to it's corresponding vertex in the target space.

This association is accomplished using registered spherical surfaces. During surface reconstruction, a spherical registration between subject space and fsaverage is computed. The transformation is then applied to the subject's sphere, yielding a registered sphere: `sphere.reg`. When the registered sphere is overlaid on the fsaverage sphere, two vertices being close together means they come from the same anatomical region on the cortex.

Similarly, we have registered spheres for all of the average subjects: 32k_fs_LR, 164k_fs_LR, and even fsaverage (although the registration for fsaverage is just the identity). The registered spheres are therefore a way-point for tranforming data from one space into any other. Data from vertex *i* in the source space is sent to the vertex *j* in the target space, such that vertex *j* is the closest vertex on the target registered sphere to vertex *i* on the source registered sphere [4].

You can use the HCP's `wb_command -surface-resample` for resampling anatomical surfaces. For resampling metric data, you can use either Freesurfer's `mri_surf2surf` or the HCP's `wb_command -metric-resample`.

[4] This is only roughly speaking of course. Actually, this is called nearest-neighbor resampling. There are also more complex, more accurate methods.

[Glasser2013] Glasser, M. F., Sotiropoulos, S. N., Wilson, J. A., Coalson, T. S., Fischl, B., Andersson, J. L., ... & Van Essen, D. C. (2013). The minimal preprocessing pipelines for the Human Connectome Project. Neuroimage, 80, 105-124.

[Jenkinson2002] Jenkinson, M., Bannister, P., Brady, M., & Smith, S. (2002). Improved optimization for the robust and accurate linear registration and motion correction of brain images. Neuroimage, 17(2), 825-841.

[Greve2009] Greve, D. N., & Fischl, B. (2009). Accurate and robust brain image alignment using boundary-based registration. Neuroimage, 48(1), 63-72.

[Power2012] Power, J. D., Barnes, K. A., Snyder, A. Z., Schlaggar, B. L., & Petersen, S. E. (2012). Spurious but systematic correlations in functional connectivity MRI networks arise from subject motion. Neuroimage, 59(3), 2142-2154.

[Jenkinson1999] Jenkinson, M. (1999). Measuring transformation error by RMS deviation. Studholme, C., Hill, DLG, Hawkes, DJ.

[Behzadi2007] Behzadi, Y., Restom, K., Liau, J., & Liu, T. T. (2007). A component based noise correction method (CompCor) for BOLD and perfusion based fMRI. Neuroimage, 37(1), 90-101.

[Marcus2013] Marcus, D. S., Harms, M. P., Snyder, A. Z., Jenkinson, M., Wilson, J. A., Glasser, M. F., ... & Hodge, M. (2013). Human Connectome Project informatics: quality control, database services, and data visualization. Neuroimage, 80, 202-219.

[Poldrack2011] Poldrack, R. A., Mumford, J. A., & Nichols, T. E. (2011). Handbook of functional MRI data analysis. Cambridge University Press.

[Nichols2002] Nichols, T. E., & Holmes, A. P. (2002). Nonparametric permutation tests for functional neuroimaging: a primer with examples. Human brain mapping, 15(1), 1-25.

[Winkler2014] Winkler, A. M., Ridgway, G. R., Webster, M. A., Smith, S. M., & Nichols, T. E. (2014). Permutation inference for the general linear model. Neuroimage, 92, 381-397.

[Smith2009] Smith, S. M., & Nichols, T. E. (2009). Threshold-free cluster enhancement: addressing problems of smoothing, threshold dependence and localisation in cluster inference. Neuroimage, 44(1), 83-98.

[Saxe2006] Saxe, R., Brett, M., & Kanwisher, N. (2006). Divide and conquer: A defense of functional localizers. NeuroImage, 30, 1088–1096.

[Boynton1996] Boynton, G. M., Engel, S. A., Glover, G. H., & Heeger, D. J. (1996). Linear systems analysis of functional magnetic resonance imaging in human V1. The journal of neuroscience, 16(13), 4207-4221.

[Dale1997] Dale, A. M., & Buckner, R. L. (1997). Selective averaging of rapidly presented individual trials using fMRI. Human Brain Mapping, 5(5), 329–340.