MNE-PYTHON

AN INTRODUCTORY GUIDE

Laura Gwilliams

(Last updated 02/07/2014)

If you have any problems with this guide, please email laura.gwilliams@nyu.edu

Contents

Scripted steps:

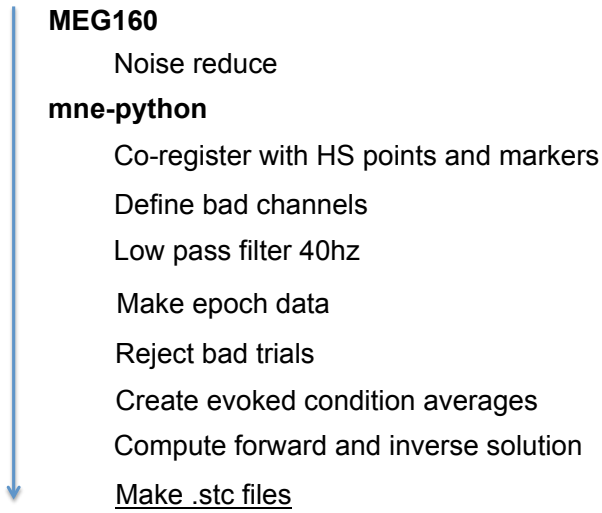
1. Introduction

Welcome to the introductory guide to mne-python!

Here you will find a step-by-step guide to preprocessing MEG data. This document is intended to be accessible for both beginners to mne-python and those who are more familiar with python programming. It is designed for individuals using a Unix Macintosh computer. MNE can also be used with Linux, but some of the installation processes will be different to those described here.

This guide has two parts. The document you are reading now, and the 'mne-python_guide' script. If you have not done so already, please save a local copy of this script and save it with your experiment name. Each step in the script has a corresponding explanation in this document. By executing each element in turn, the end product will be a script tailored to your experiment. You can then use this script to preprocess your data easily and efficiently.

The structure of this pipeline is detailed below. There is an initial noise reduction step in MEG160, and the rest of the workflow will be conducted using mne-python. As you can see, it will take you from raw files to source estimates. You can then use these source estimate files to conduct statistical tests on your data.



This guideline is based upon the official MNE tutorial:
http://martinos.org/mne/stable/index.html

Useful Links

To find definitions and uses of any mne-python command:
http://martinos.org/mne/stable/python_reference.html

For other tutorials: http://scipy-lectures.github.io/intro/intro.html

For information about python:
http://docs.python.org/2/using/mac.html#getting-and-installing-macpython

2. Preliminary Installation Instructions

2.1 Required Software

Before you can begin using MNE, you have to download quite a few different pieces of software:

- MNE
    - Command line
    - Graphic User Interface (GUI)
- Freesurfer
- Enthought Canopy
- Python
- Pip
- Eelbrain
- XCode
- TextWrangler (Optional)
- GitHub (Optional)

2.2 Folder visibility

In order to make trouble-shooting a little easier, it is a good idea to make all hidden files visible. This is very easy to do on a Mac.

If you are using Mac OS X 10.8 Mountain Lion, 10.7 Lion, or 10.6 Snow Leopard copy and enter the following into terminal:

```
$ defaults write com.apple.Finder AppleShowAllFiles TRUE
```

For OS X Mavericks 10.9 there is a very slight difference in casing:

```
$ defaults write com.apple.finder AppleShowAllFiles TRUE
```

Next, refresh all open finder windows by 'killing' the finder process, which is also done through the command line with the following string:

```
$ killall Finder
```

Again, hit Enter, and you'll discover the Finder quits and relaunches itself very quickly with the changes in effect.

2.3 Check Mac Version

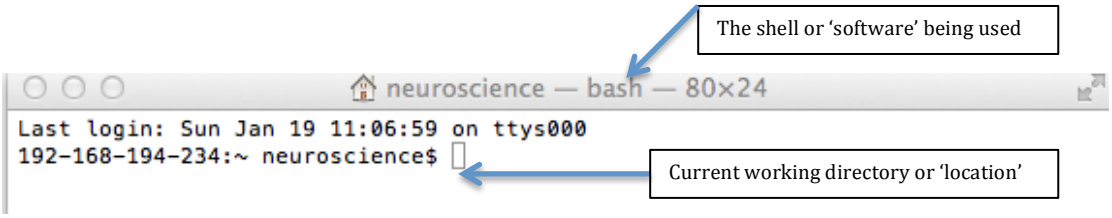
An important thing to do before we go further is to check the Mac version you are using. To find this information, go to the Desktop, click the 'Apple' symbol in the top left, and 'About this Mac'. Then, click 'More Info…' to pull up the processor and software currently installed:

From this information and referring to the table below, I know the current machine I'm working with is a 64-bit processor using OS X 10.9.1 software. Make a mental note of this information for your own computer.

Processor Name	32- or 64-bit
Intel Core Solo	32 bit
Intel Core Duo	32 bit
Intel Core 2 Duo	64 bit
Intel Quad-Core Xeon	64 bit
Dual-Core Intel Xeon	64 bit
Quad-Core Intel Xeon	64 bit
Core i3	64 bit
Core i5	64 bit
Core i7	64 bit

2.4 Using Terminal

Some of the downloading will need to be done using 'terminal'. This is the command-line based interface. To open the program, click the 'Spotlight' search in the top right and search for 'terminal'. Click on the Hit next to where is says 'Applications'. This will open a white command-line window. Displayed in the terminal are various pieces of information.

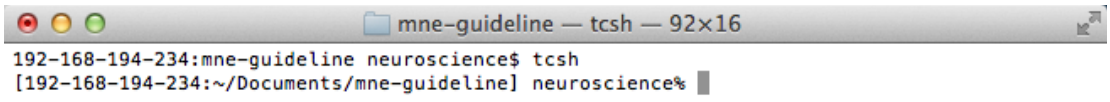


2.4.1 Changing Terminal Shell

The default shell or 'software' used by the terminal in Mac is bash, and this is what is mainly used in the installation. This is usually the default when you open the terminal.

In order to change the shell you are using, simply call it by name:

```
$ tcsh
```

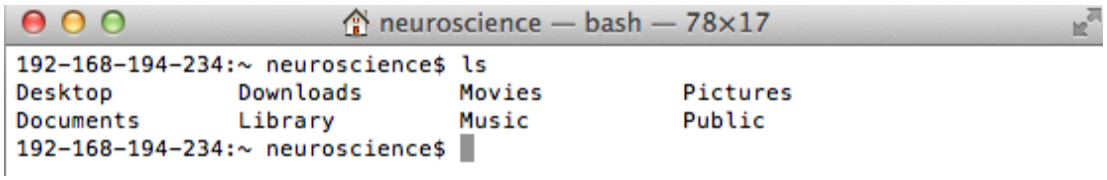


2.4.2 Changing the Working Directory

In order to see the files and folders in your current location, you can use the 'list' command by entering:

```
$ ls
```

The output should look something like this:



For some of the installations, you will need to change the directory you are working in. In order to do this, you need to type 'cd' ('change directory') plus the desired location.

To move up one level in your directory folders:

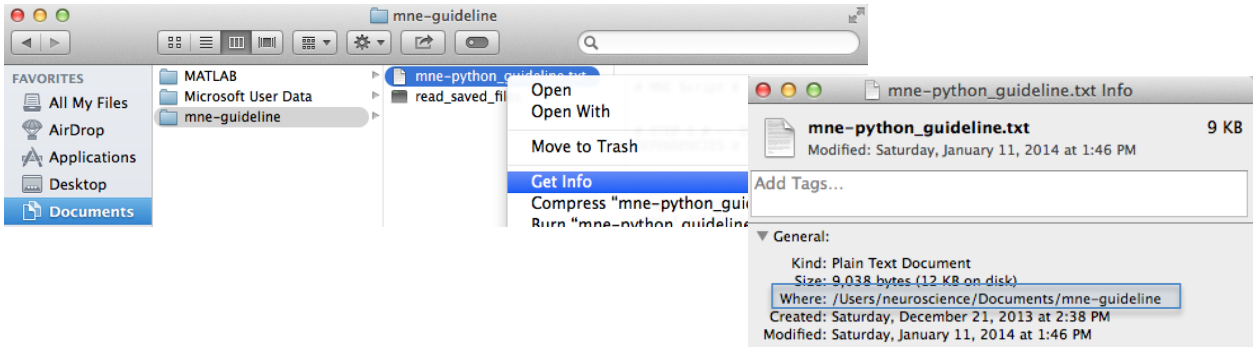
```
$ cd ..
```

To change directory into a folder contained in your current location, simply type cd and the name of the folder:

```
$ cd Downloads/
```

You can auto-complete the path with the tab button once the input does not correspond to any other document in that location.

To extract the exact path of your file in one step, open it in the finder and right click > 'Get Info':

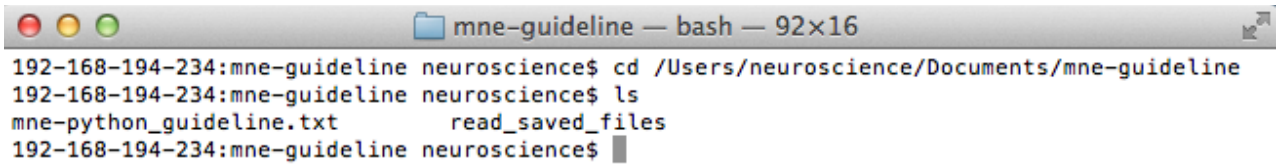


You can then copy the 'Where' information into your terminal window:

```
$ cd /Users/neuroscience/Documents/mne-guideline
```

```
$ ls
```

You should get an output similar to this:



3. Downloads and Installation

3.1 Download Command Line MNE

First, go to the MNE website at http://martinos.org/mne/ .

Here you have a little explanation about what MNE is, what you can do with it, and its association with Python and Matlab.

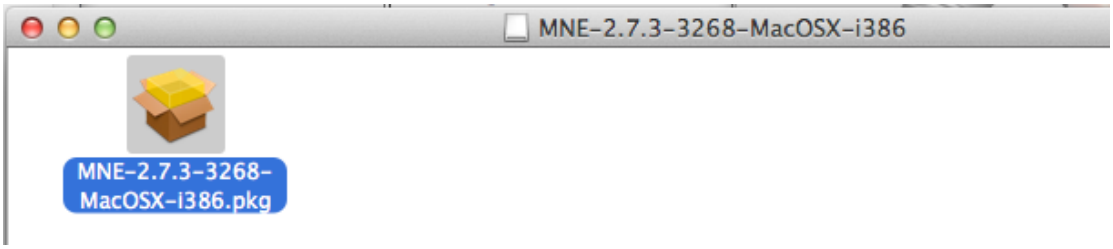
To download MNE you have to register on this page. It doesn't cost any money but you have to give information such as name, institution, and the operating system you're using.

Once this is done you get an email with your login and password, and then you can download the MNE software from the download page.

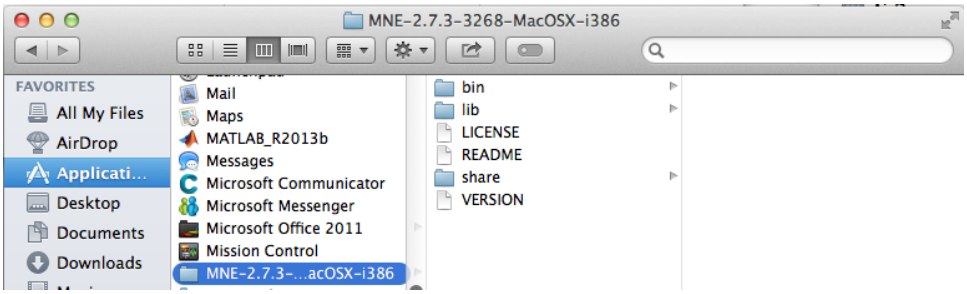
Scroll down to 'software distribution packages [the most recent development version]'. Pick the relevant option to your operating system and select the 'disk image' download. **Do not download the nightly version as this is unstable.**

MNE-2.7.3-3268-MacOSX-i386.dmg	28781K	MNE Version 2.7.3 (3268) MacOSX Leopard Intel (disk image)

Once is has finished downloading, open it up in your downloads folder and double click the disk image



This will open the *Install MNE* walkthrough. Follow the instructions on the screen. Once this is done, look in your Applications folder and there should be an MNE-<version> folder containing 'bin', 'lib', 'share' folders:

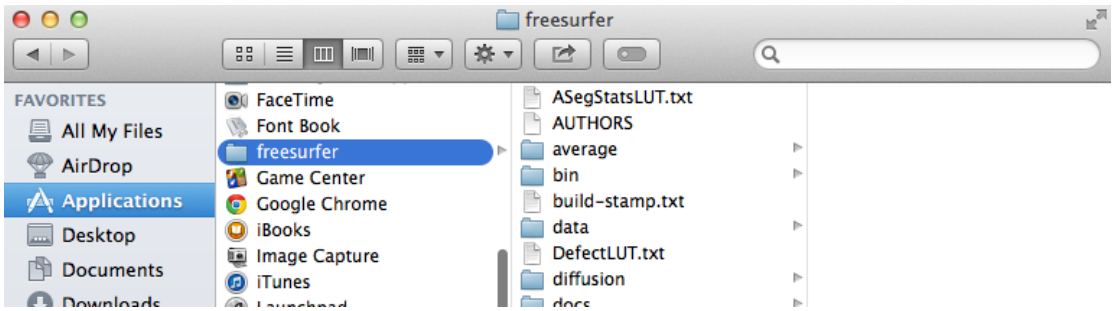


3.2 Download Freesurfer

Follow the link to download FREESURFER. Click option number three, which is 'download', and select the relevant disk image download corresponding to the operating system you are using:

Mac	Lion OX X 10.7 (64b Intel)	stable v5.3.0	15 May 2013	freesurfer-Darwin-lion-stable-pub-v5.3.0.dmg	3.5G
Mac	SnowLeopard OS X 10.6 (32b Intel)	stable v5.3.0	15 May 2013	freesurfer-Darwin-snowleopard-i686-stable-pub-v5.3.0.dmg	3.2G

This is quite a big file so don't be surprised if it takes an hour or so. Once complete, you should open the file in your downloads folder and follow the 'Install Freesurfer' wizard. This saves Freesurfer to your Applications folder:

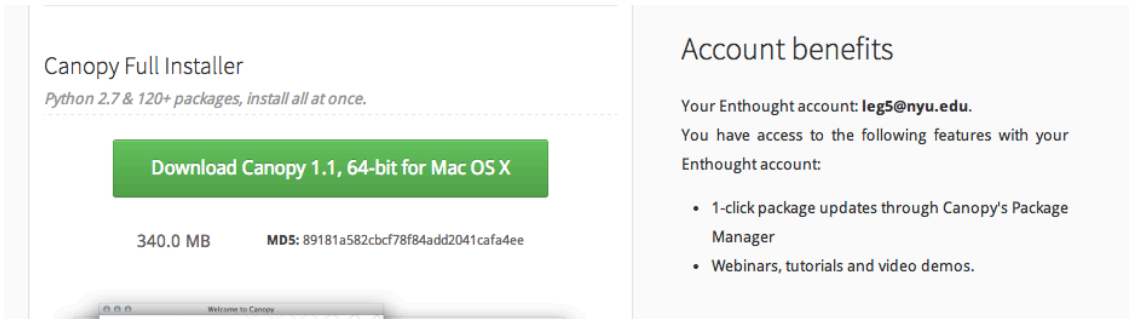


3.3 Download Enthought Canopy

This is an editor interface that is used to execute commands with mne-python. As the download also includes a Python distribution, we do not need to install Python separately. Canopy isn't your only choice, but it's the one that most people use and it is generally reliable.

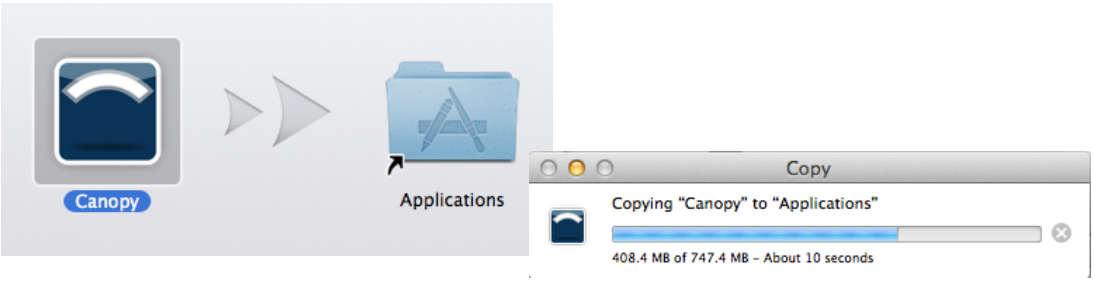
Go the Enthought Canopy website here. Click on 'Products' > 'Academic License' and scroll down to press the green 'Request your Academic License' button.

Here you need to sign up to Enthought using your academic email address. This gives you free access to the full version of canopy. From here, log in to your account and download the 'Canopy Full Installer'. Again, you need to pick your operating system, and whether you're using a 32bit or 64bit computer by selecting 'Pick your flavor'.

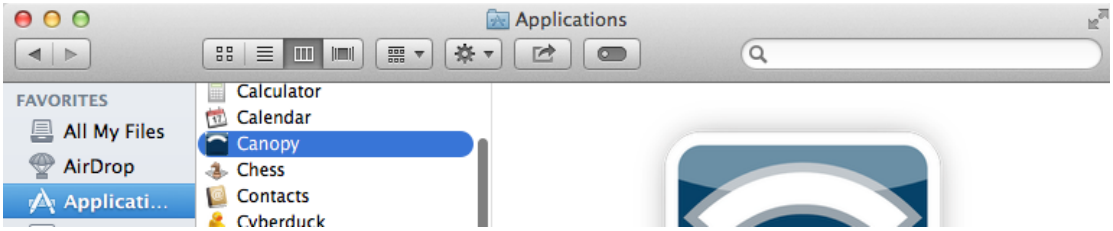


Note: Do not download 'Express Installer' otherwise you won't have all the add-ons you need.

Once the file appears in your 'Downloads', open it up and copy the Canopy application to your applications folder. Simply dragging the Canopy icon to the Applications icon can do this:



As before, you should now be able to see the Canopy program in your 'Applications' folder:

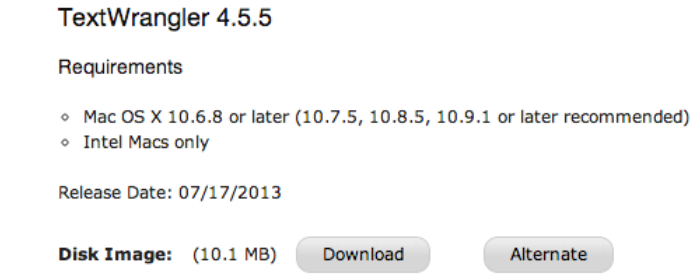


Make sure Canopy is your default python environment. This should pop up when running Canopy for the first time. If not, click Canopy > Preferences > Default Python environment.

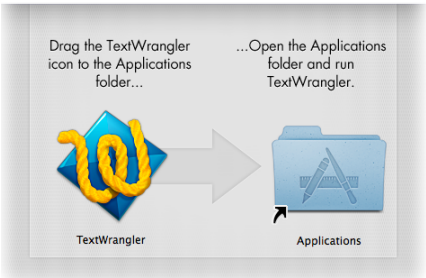
3.4 Download TextWrangler

If you do not have this installed already, it is good practice to have a functional plain text writer, which can handle and create different coding systems. I favour TextWranger which can be downloaded here:

As always, download the disk image.



Once it appears in your downloads folder, simply drag the TextWrangler icon to the Application folder icon:



3.5 Download XCode

When running command-line based installations it is crucial to have a code compiler which can read and understand the scripts being processed. There are many of these kinds of software freely available. You can download one called 'XCode' directly from the Apple App Store.

Open up the app store and search for 'xcode'. Click the button which says 'free' then 'install app' and you will be prompted with your Apple ID and password. Once this is entered, the download will begin automatically.

3.6 Download pip

Now we need to download the python interface program, or 'pip'. This runs from the command line and is crucial to downloading and updating software. Your installation of python must have been successful for this to work. Information about this program can be found here: http://www.pip-installer.org/en/latest/installing.html

3.6.1 Download with easy_install

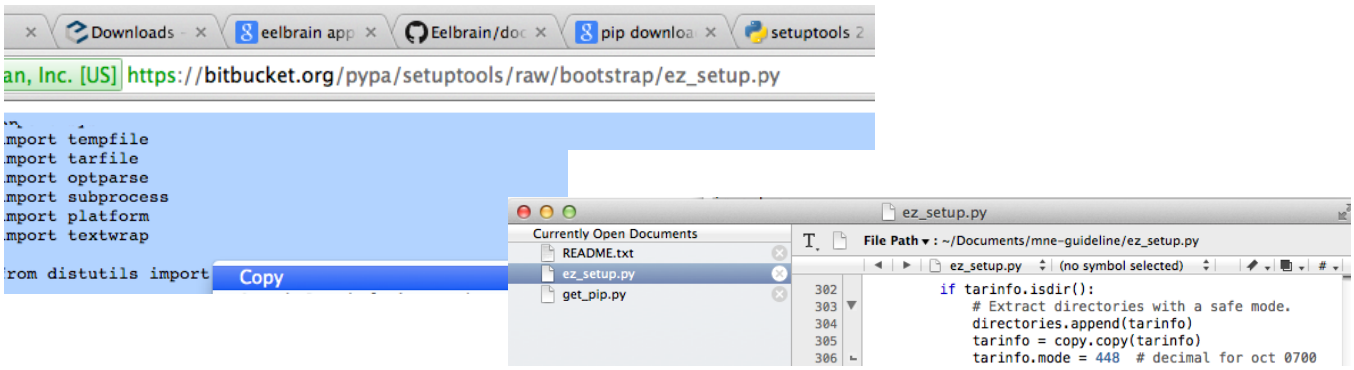
If you have 'easy_install' setup, then enter the following in the command line:

```
$ easy_install pip
```

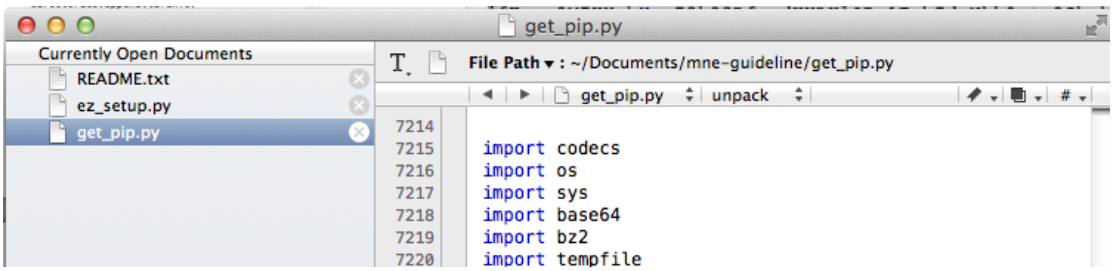
If not, (which can be quickly tested by trying this out), then follow these steps:

3.6.2 Download without easy_install

1. Copy and save a version of ez_setup.py. Find it here.



2. Now, do the same for the 'get-pip.py' script, which can be found here



3. Change your directory to the folder that contains these two scripts. To run a python script, we call the python shell and then the name of the script:

```
$ python ez_setup.py
```

If it was downloaded successfully, the final lines should look like this:

```
Installed /Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/site-packages/setuptools-2.1-py2.7.eg
Processing dependencies for setuptools==2.1
Finished processing dependencies for setuptools==2.1
192-168-194-234:mne-guideline neuroscience$
```

4. Now, get pip by running the script from the terminal

```
$ python get-pip.py
```

```
    Installing pip script to /Library/Frameworks/Python.framework/Versions/2.7/bin
    Installing pip2.7 script to /Library/Frameworks/Python.framework/Versions/2.7/bin
    Installing pip2 script to /Library/Frameworks/Python.framework/Versions/2.7/bin
Successfully installed pip
Cleaning up...
```

5. Install pip using the downloaded setup tools

```
$ pip install ––upgrade setuptools
```

```
----------------------------------------
  Rolling back uninstall of distribute
Cleaning up...
Command /Users/neuroscience/Library/Enthought/Canopy_64bit/User/bin/python -c "import setuptools;__
zffvr0000gn/T/pip_build_neuroscience/setuptools/setup.py';exec(compile(open(__file__).read().replace
rd /var/folders/c5/3jy2gnw50m1_ncy1lwbzffvr0000gn/T/pip-lLXAia-record/install-record.txt --single-ve
in /private/var/folders/c5/3jy2gnw50m1_ncy1lwbzffvr0000gn/T/pip_build_neuroscience/setuptools
Storing complete log in /var/folders/c5/3jy2gnw50m1_ncy1lwbzffvr0000gn/T/tmpRUnZOL
```

6. To test your pip installation, enter the following in the command line:

```
$ pip help
```

This will show you the available options which are now usable with pip. This includes installing, upgrading and uninstalling software, as well as gaining information about what software is already installed.

```
192-168-194-234:mne-guideline neuroscience$ pip help

Usage:
  pip <command> [options]

Commands:
  install                     Install packages.
  uninstall                   Uninstall packages.
  freeze                      Output installed packages in requirements format.
  list                        List installed packages.
```

3.7 Download mne-python

Open up the terminal window on your machine. We will download this using ‘pip’, so type

```
$ pip install mne
```

To check this worked, type:

```
$ pip show mne
```

In the future, in order to update your mne (which you should do every week or so) type:

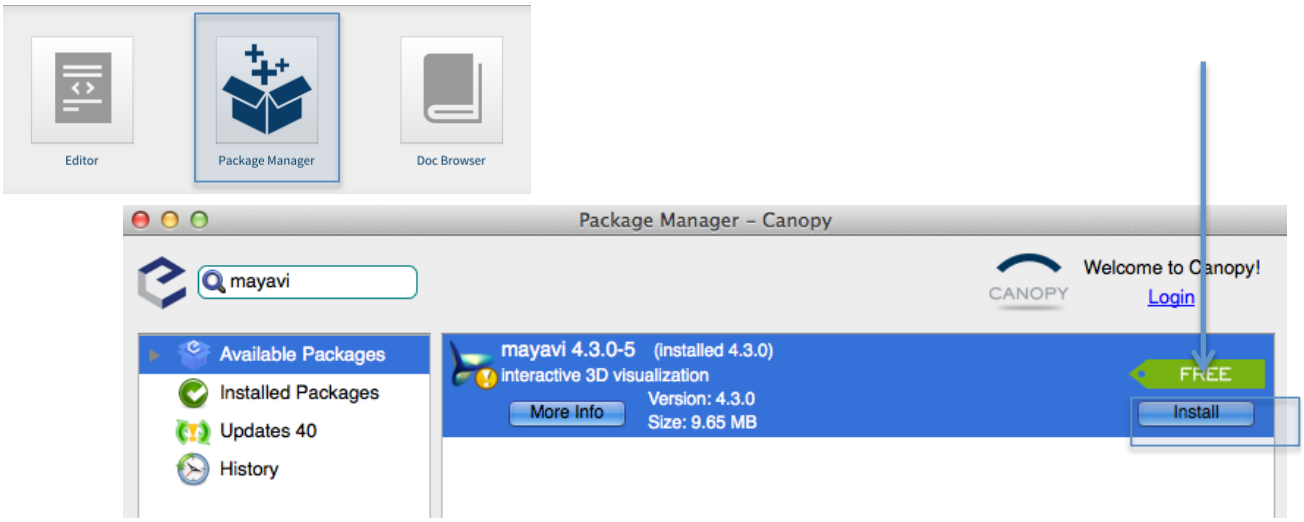
```
$ pip install mne -U
```

Note: The 'pip' application selects the correct directory for your installation automatically, so it doesn't matter *where* you run the command from in the terminal.

Note: If you ever get an error message saying that a certain module is not found, then your first port of call should be to try and download the missing module through pip. Some functions have dependencies on other pieces of software, so you need to make sure your computer always has everything it needs.

3.8 Downloading Canopy Packages

If along the way you are getting errors of missing scripts, try to search for these in the 'Package Manager'. For example, you'll need to download a package called 'Mayavi' in order to visualize some of the plots and GUIs.



3.9 Download Eelbrain

In addition to mne-python, for one of the stages we will also be using an additional package called 'Eelbrain', developed by Christian Brodbeck. We will

only be using the blink rejection element from this software, but if you want to look at the other functionalities it holds, please look here.

To install eelbrain, use either easy_install or github.

If the previous steps were followed correctly, you should be able to use easy_install. A version of easy_install was saved to your computer with the Canopy installation as a Python package. Type and enter the following command. Again, note that it doesn't matter in which directory your terminal is located.

```
$ sudo easy_install eelbrain
```

You can update using:

```
$ sudo easy_install eelbrain -U
```

Alternatively, you can use Git to install Eelbrain. Instructions for this installation are detailed below.

3.10 Downloading GIT (Optional)

Git is a useful tool for downloading programs and collaborative projects. This is not immediately necessary for the installation of mne-python, but is likely to come in useful to you.

Go to the git-scm.com/download, which is here.

Select your operating system and the download should start automatically on the next page.

The download should then appear in your downloads folder. As with the MNE and Freesurfer downloads, double click on the git.pkg box image and this will begin the Install Git wizard. Follow the instructions on screen. You will not see it in your Applications folder – it will instead be located at usr/local/git.

Once this is done, set up GIT by opening the terminal. You need to configure by telling it your username and email address by typing:

```
$ git config --global user.name "John Doe"
```

```
$ git config --global user.email johndoe@example.com
```

3.10.1 Downloading Code Using Git

When downloading anything from GIT, you first need to change your directory to the place you want the items to be installed. This is done by typing 'cd' ('change directory') and then the path you want to be in (look at step iii above for more information on how to use the terminal).

Once you have changed directory to where you want to be, you'll need to type 'git', then 'clone' then the github web address of the code you want to download:

```
$ git clone https://github.com/LauraGwilliams/mne-python_guide.git
```

This will begin the download and you'll get a progress report. When it is finished, then you should see something like this as an output:

```
192-168-194-234:mne-guideline neuroscience$ git clone https://github.com/LauraGwilliams/mne-
python_guide.git
Cloning into 'mne-python_guide'...
remote: Counting objects: 9, done.
remote: Compressing objects: 100% (8/8), done.
remote: Total 9 (delta 2), reused 0 (delta 0)
Unpacking objects: 100% (9/9), done.
Checking connectivity... done
```

Change directory into the folder you just cloned from github, and you will be able to see the files within there:

```
$ cd mne-python_guide
```

```
192-168-194-234:mne-guideline neuroscience$ cd mne-python_guide
192-168-194-234:mne-python_guide neuroscience$ ls
README.md               load_saved_files        mne-python_main-script
192-168-194-234:mne-python_guide neuroscience$
```

In order to get the most up-to-date code, cd into the destination folder and enter

```
$ git status
```

```
$ git pull
```

This will first tell you whether any changes have been made since you last downloaded the latest code. If there have, 'pull' will update the code to reflect the latest versions.

3.10.2 Downloading Software Using Git (Eelbrain)

As explained above, you can also download Eelbrain using GitHub. In order to do this, first change your directory to where you want to save these files. This is not where the installation will take place, so it is not crucial where your directory is located.

```
$ cd /target/directory

$ git clone https://github.com/christianmbrodbeck/Eelbrain.git
```

The installation is completed by running the following command:

```
$ python setup.py develop
```

This copies all of the files into the correct location using your computer's directory structure.

To update Eelbrain, similar to updating code, change directory into the Eelbrain folder, pull and run the setup again:

```
$ cd /target/directory/Eelbrain
```

```
$ git pull
```

```
$ python setup.py develop
```

4. Updating

Updating using PIP

In order to ensure your packages are working coherently with one another, it is important to make sure you update each of these above modules frequently.

To update the code installed through pip (not the software installed through other means), simply use terminal and enter:

```
$ pip install <name_of_software> -U
```

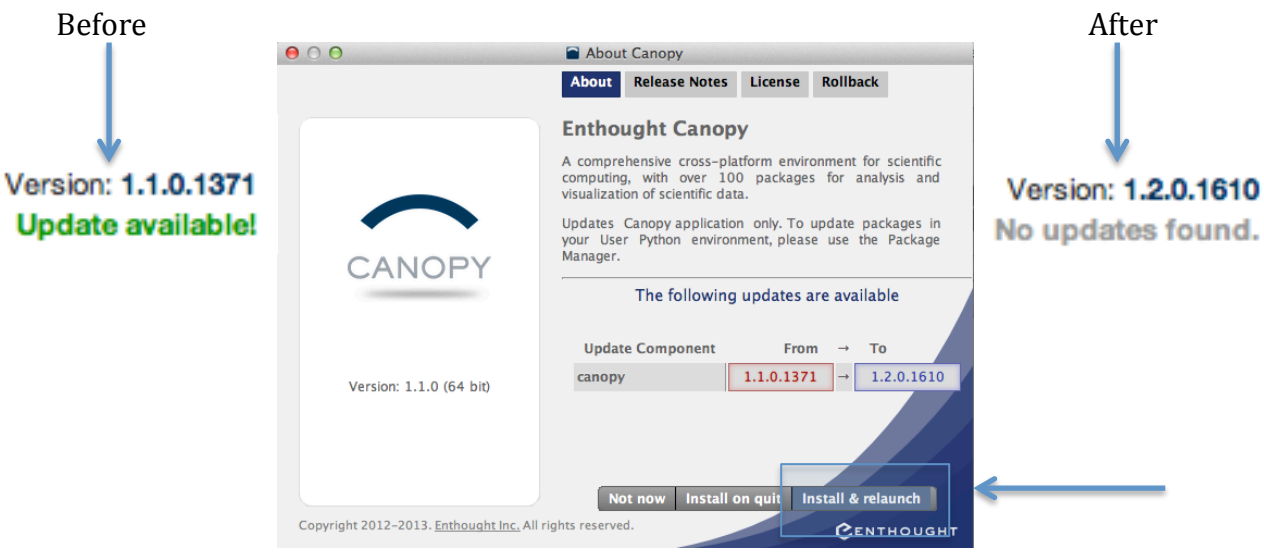
Here, the –U tag corresponds to 'update'. If everything is up to date, you should see a response such as this:

```
Lauras-MacBook-Pro-2:~ Laura$ pip install mne -U
Requirement already up-to-date: mne in ./Library/Enthought/Canopy_64bit/User/lib
/python2.7/site-packages
Cleaning up...
Lauras-MacBook-Pro-2:~ Laura$
```

Note: If you find that an element in your script does not work after the update, but it worked fine before the update, this may be a bug in the latest code. If this happens you can email the contributors list of whichever software you are having difficulties with, along with the error message, and this should be fixed in the following update.

Updating Canopy

Because Canopy comes along with its own python package distributions, you need to make sure it is regularly updated. Every time you open canopy, on the bottom right of the welcome screen you will see the update status of your current version. To update, simply click the green font 'Update available!'



4. General Setup

Open up your Enthought Canopy application. Click 'Editor' to pull up the command line interface.

Type and enter 'import mne'. If you don't get an error, everything from the previous steps has been installed correctly. Congratulations!

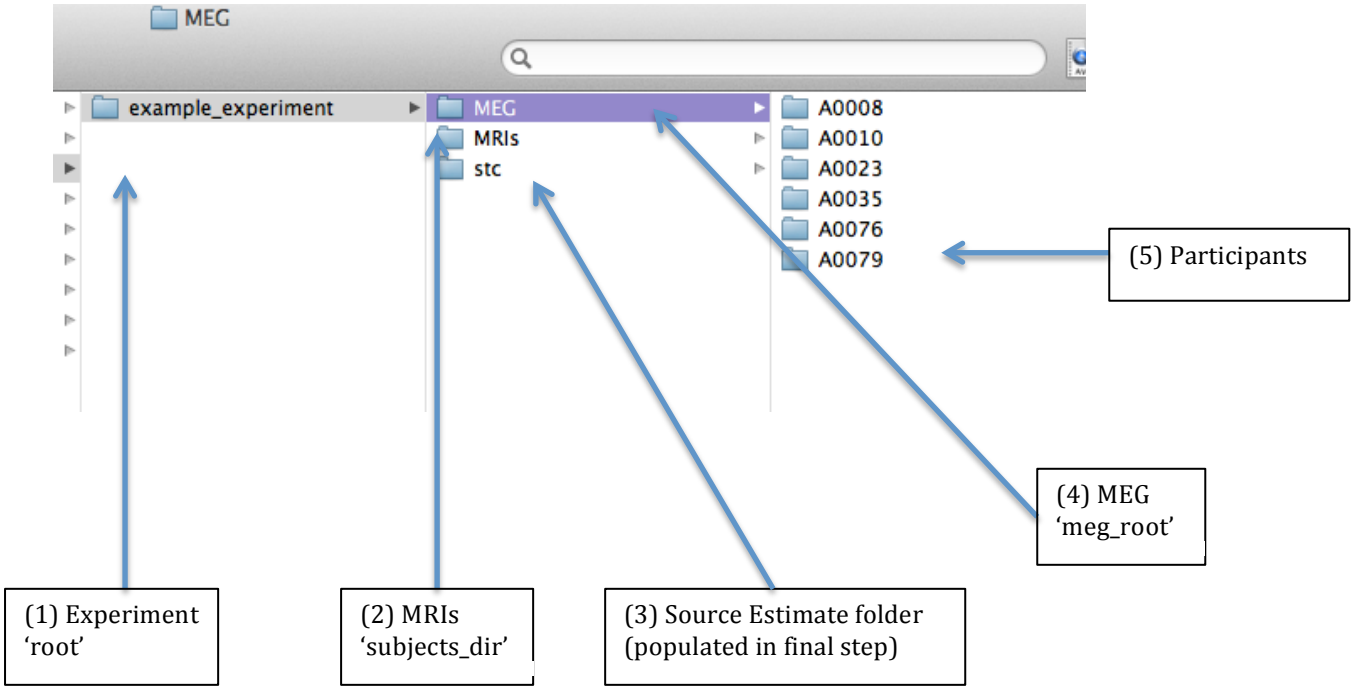
4.1 File organization.

A lot of the commands run in mne-python will assume a certain data structure. Apart from it being good practice to organize data in a systematic way, it will also help you avoid errors while working through the data.

Firstly, make sure you are not altering your original data files. If anything goes wrong, you want to make sure you have these original files safe an untouched.

Second, make sure the environment you're working from is not too general. I.e., you don't want to be working straight from the Desktop.

Third, follow this data structure in your folders:

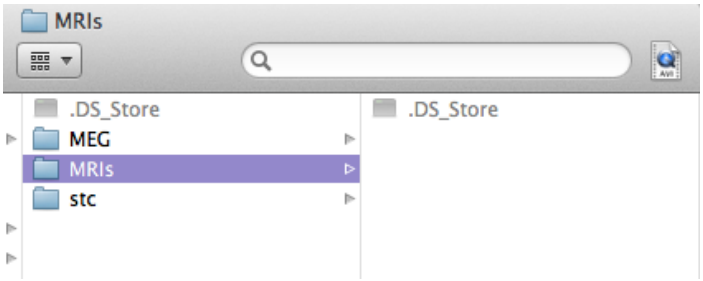


Note: Programming languages do not work well with 'spaces' in file and folder names. Mne-python is no exception. To avoid headaches, make sure that none of your files or folders contain spaces. You can instead use underscores ' _ ' or hyphens ' – '.

4.2 Folder Contents

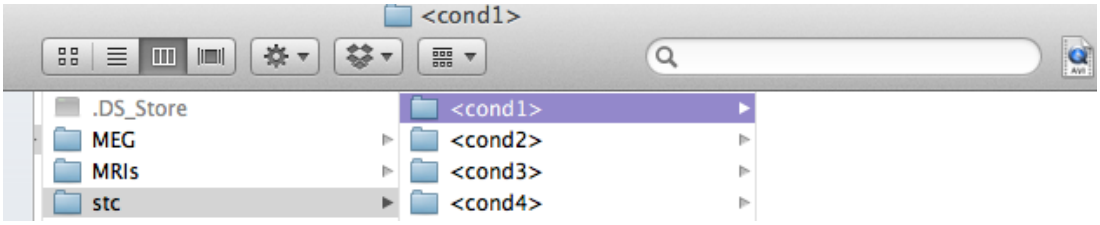
4.2.1 MRIs Folder

The contents of these participant folders will differ depending on whether you have actual MRI scans for the participants or not. If you do, the MRI files should be housed within a corresponding participant folder. If you do not have MRI scans, the MRIs folder is left empty – it will be populated at a later stage with the freesurfer average brain during coregistration.



4.2.2 Source Estimates folder

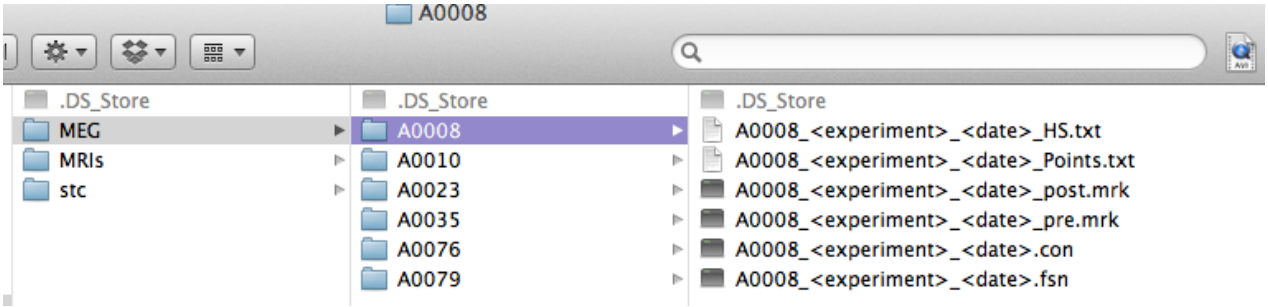
The stc folder should contain a folder for each of your conditions. Crucially, you must be consistent in the names of your conditions, as this will be used throughout the script.



The condition folders themselves can remain empty, you only need to create the folders themselves. These will be populated in the final step of the pipeline.

4.2.3 MEG folder

Here we house the MEG data for each of the participants. Each subject requires their own folder, which should contain (at least) the following:



4.3 Using the PDF guide and script

To make things easier, there is an 'mne-python_guideline' script available on the server. It is for use with this document so you can execute each command alongside. Each section is numbered accordingly.

If you haven't done so already, open this script now, and save a local copy as "mne-python_<your experiment>".

If you take a look, you will see that there are various places where the input is <cond1> <cond2> etc. This is where you will need to place your own condition names. Make the changes as you work through the script, rather than making them all now, as that will be more prone to unidentifiable errors. Remember if you have more conditions than those listed, just copy the relevant line and input the condition information. If you have less, feel free to delete them.

Remember to delete the arrows from the script e.g., '<cond1>' = 'high_freq'.

The idea is that once you have run your first participant through this pipeline, you can save this script as your own experiment script, and use this to process all subsequent participants. Work through this script alongside this word document, as the word document will give you more information about what's going on and what you're exactly doing with the data at each step.

You can also see a 'read_saved_files' part at the end of the script, which can be used to load objects back into the workspace from file. Make sure that you follow steps 1 and 2 before running the commands, and change your directory to the participant folder for whom you want to load the data. This means you do not have to begin from scratch if you did not follow the full pipeline from start to finish.

5. MEG160 Noise Reduction

As there is currently no standard method of noise reduction in mne-python, this is the only step to be conducted outside of this software.

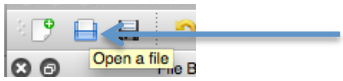
1. Open your raw .con or .sqd file in MEG160
2. Select 'Edit>Noise Reduction'
3. Select 'CALM' and the reference channels
    a. In Abu Dhabi these are 208:216
    b. In New York these are 160:167
4. Ensure parameters 'Moving width' is selected as 6001
5. Hit 'OK'

The noise reduction algorithm with then appear and display its progress. This usually takes around 3 minutes to complete.

When this is done, save the new file as '<participant>_<experiment>_nr'. **Do not overwrite the old file.** It's important to keep all raw files in original form.

Processing with mne-python

Open canopy by clicking the blue 'Canopy' symbol. A 'Welcome to Canopy' window will appear. Click 'Editor' to show the window where we will be employing the script from. Open the mne-python script by clicking the second button on the top left and searching for its location. The script should now appear in the top window.



Before we go further, make sure that the graphics backend. On the top left corner of your screen, click Canopy>Preferences>Python and ensure it is set to 'Interactive Qt4'. This is required for all of the "GUI" steps. You have to restart the kernel when changing the backend, which erases information in the computers working memory, so make sure that this is set beforehand.

1. Import MNE dependencies

Every time we use mne commands, we need to run functions that set up the working environment. At the top of the script you will see various 'import' functions. Most of them can be left untouched, you just need to alter the os.environ[] lines:

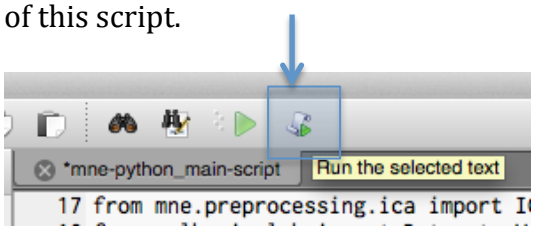
```
os.environ['MNE_ROOT'] = '/Applications/MNE-<your version of MNE here>'
os.environ['SUBJECTS_DIR'] = '/<path_to_mri_folder>'
os.environ['FREESURFER_HOME'] = '/Applications/freesurfer/'
```

First change the path after 'MNE_ROOT' to equal the version of MNE you are using, so the name of MNE as it appears in your Applications folder. Second, change the path after SUBJECTS_DIR to equal the path to your MRI folder. The name of 'freesurfer' should remain the same, but check this is how it appears in your Applications folder too, just to be safe.

Once you have modified these lines, they should look something like this:

```
os.environ['MNE_ROOT'] = '/Applications/MNE-2.7.3-3268-MacOSX-i386/'
os.environ['SUBJECTS_DIR'] = '/Volumes/data/Transfer/Experiments/lex_dec/MRIs/'
os.environ['FREESURFER_HOME'] = '/Applications/freesurfer/'
```

Highlight all of these functions in the top window and click 'run selected text', which is the last button on the top panel. This is how you need to run each step of this script.

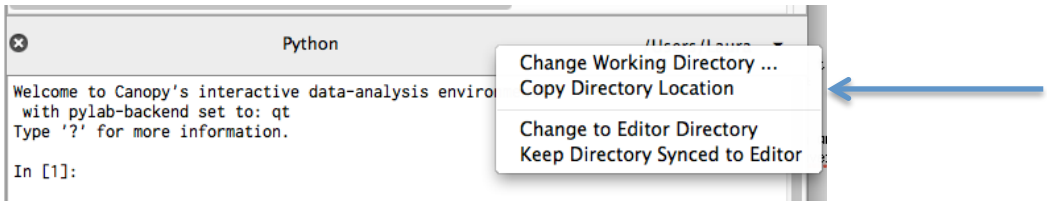


Note: If you are working with data from the server, take extra care that you have input the path correctly. Check that the path corresponds to the 'where:' once you click 'Get Info' on the file.

New York server: '/Volumes/server/…'
Abu Dhabi server: '/Volumes/data/Transfer/…'
Local files: '/Users/<user>/…'

2. Set participant and experiment names

This step is intended to save time and support uniformity, but requires a particular file structure. Please check that your files mimic the structure shown in step 4 above, in the section called 'General Setup'. When you're confident that it does, change the working directory to the MEG folder (3). Click the little black arrow on the top right, above where you can enter your commands, and search with the finder:



Now, replace the information in <>'s with the information of your experiment, surrounded by apostrophes. Remember to delete the arrows themselves too:

(5) Participant = participant number. i.e., 'A0068'.
(1) Experiment = name given to experiment folder
(1) Root = path to the folder which houses your MEG, MRI and stc folders

If your folders are named correctly, there should be no changes to following three, as they are made up of the experiment root, plus the folder name:

(4) Meg root = path to the folder which contains each of your participant's data
(2) Subjects_dir = path to MRI folder which will later house coregistration files
[3] stc_path = path to source estimates folder (final step of pipeline).

Once this is done, it should look something like this:

```
participant = 'A0068'
experiment = 'lex_dec'
root = '/Volumes/data/Transfer/Experiments/lex_dec'

meg_root = root + '/MEG'
subjects_dir = root + '/MRIs'
stc_path = root + '/stc'
```

Ensure this does not end in a '/' or the path will be incorrect

Run the lines in Step 2. You have now created six python 'objects'. You can call these objects at any point by simply typing the object name in the editor window and pressing enter. Do this now and check you have saved each of them correctly:

```
In [10]: participant
Out[10]: 'A0068'

In [11]: experiment
Out[11]: 'lex_dec'

In [12]: root
Out[12]: '/Volumes/data/Transfer/Experiments/lex_dec'

In [13]: meg_root
Out[13]: '/Volumes/data/Transfer/Experiments/lex_dec/MEG'

In [14]: subjects_dir
Out[14]: '/Volumes/data/Transfer/Experiments/lex_dec/MRIs'

In [15]: stc_path
Out[15]: '/Volumes/data/Transfer/Experiments/lex_dec/stc'
```

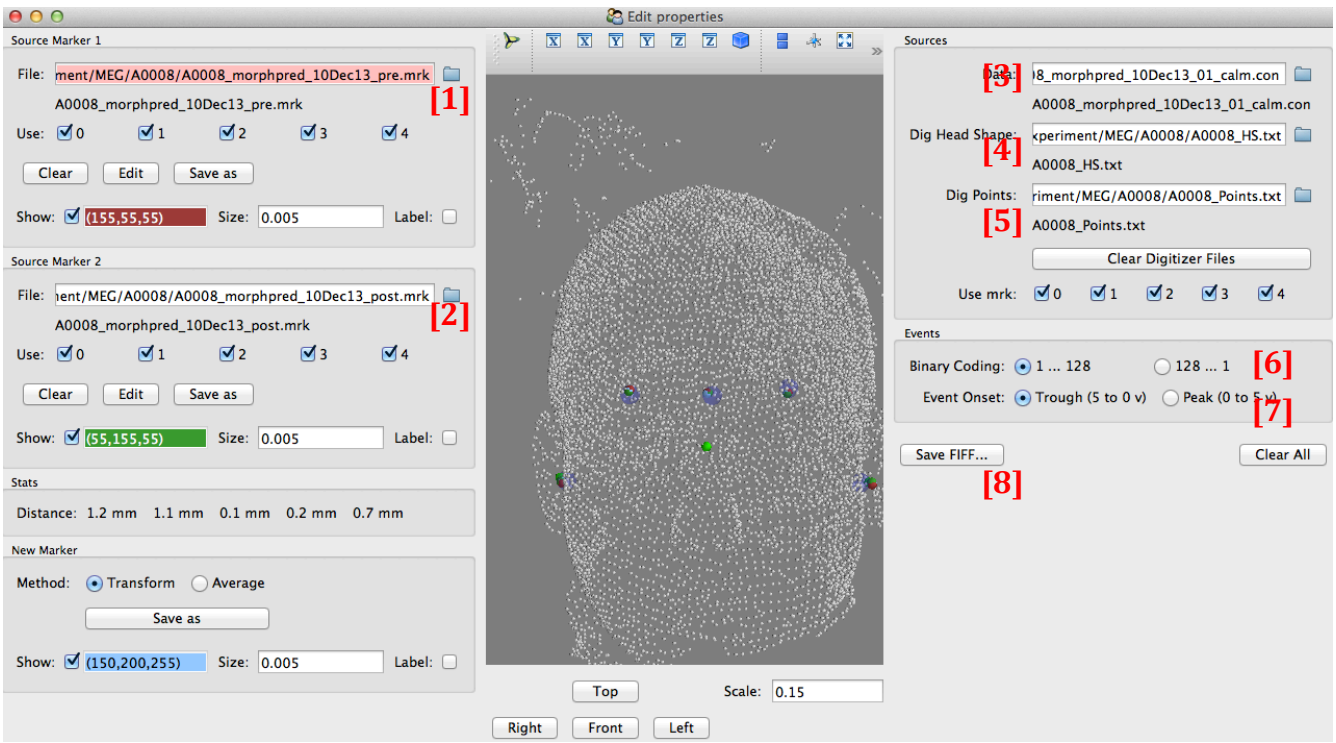
3. Convert files to .fif format

Now everything has been set up, we can begin loading up the data. In this step we use the marker points, head-shape, digital points and noise-reduced .con / .sqd file to create a single .fif object which contains all of this information.

This conversion is achieved with the 'kit2fiff' command.

In the editor window, highlight step 3:

```
>>> mne.gui.kit2fiff()
```



Each of the red numbers corresponds to a file we need to load, or a selection we need to make. Follow steps **a – f** below the pink box:

[1] **Source marker 1** = pre-experiment marker
[2] **Source marker 2** = post-experiment marker
[3] **Data** = Noise reduced .con file
[4] **Dig head shape** = sweeps HS.txt file
[5] **Dig points** = points .txt file
[6] **Binary coding** = low to high (order of channels to triggers)
[7] **Event Onset** = If trigger events are positive jumps on the channel pick peak, and if they're negative dips on the channel pick trough. For NYU, stimuli presented on a Mac are toughs and PC are peaks.
[8] **Save FIFF**… '<participant>_<experiment>-raw.fif'

a. First, load each of the five files by clicking on the little folder buttons.

b. Drag the mouse across the grey box to see the head shape. If you have not down-sampled this file yourself, kit2fiff will ask to do this conversion.

c. Make sure that the marker points from the headscan and the MEG markers are a reasonable distance apart. You can modify this distance by changing the weighting of the markers. This is done by checking/unchecking the 5 boxes on the right side 'use mrk'.

d. Click each of the direction buttons 'left', 'right', 'front', 'top' and make sure the head orientates as it should.

If the head does not rotate in the correct direction, the points may not have been recorded in the right order.

e. Click the appropriate binary coding and event onset [6], [7]

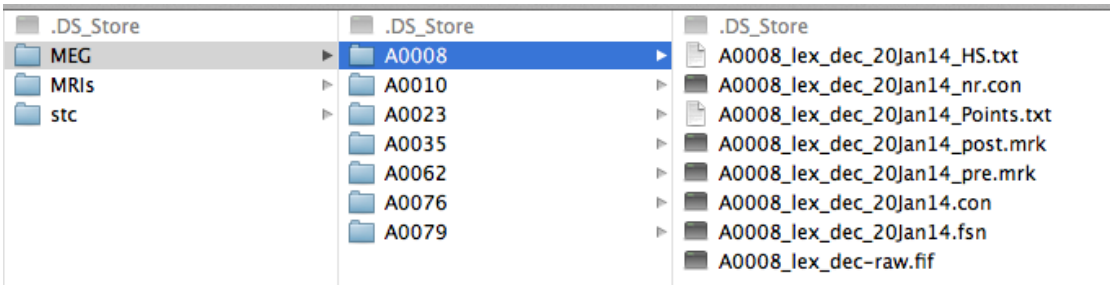
f. Save the file [8]

Note: The file name you give is very important, as this structure will be used later on to read the files again. This will mean deleting the date and 'nr' information from the file name, but that's okay. Make sure that you use exactly the same name and format for your participant and experiment throughout the script, as detailed from the very beginning (step 2).

Note: You can click 'clear all' and begin to process the next participant's .fif file. The files will queue up and process in the background. This is more time efficient if you have a number of participants' data to preprocess. When you're done creating .fif files for everyone, click the red close box on the top left.

Check File Status

If you now look at the participant folder in the MEG directory, it should have been populated with a .fif file. Your file structure should now look something like this:



4. Coregistration - Transformation matrix.

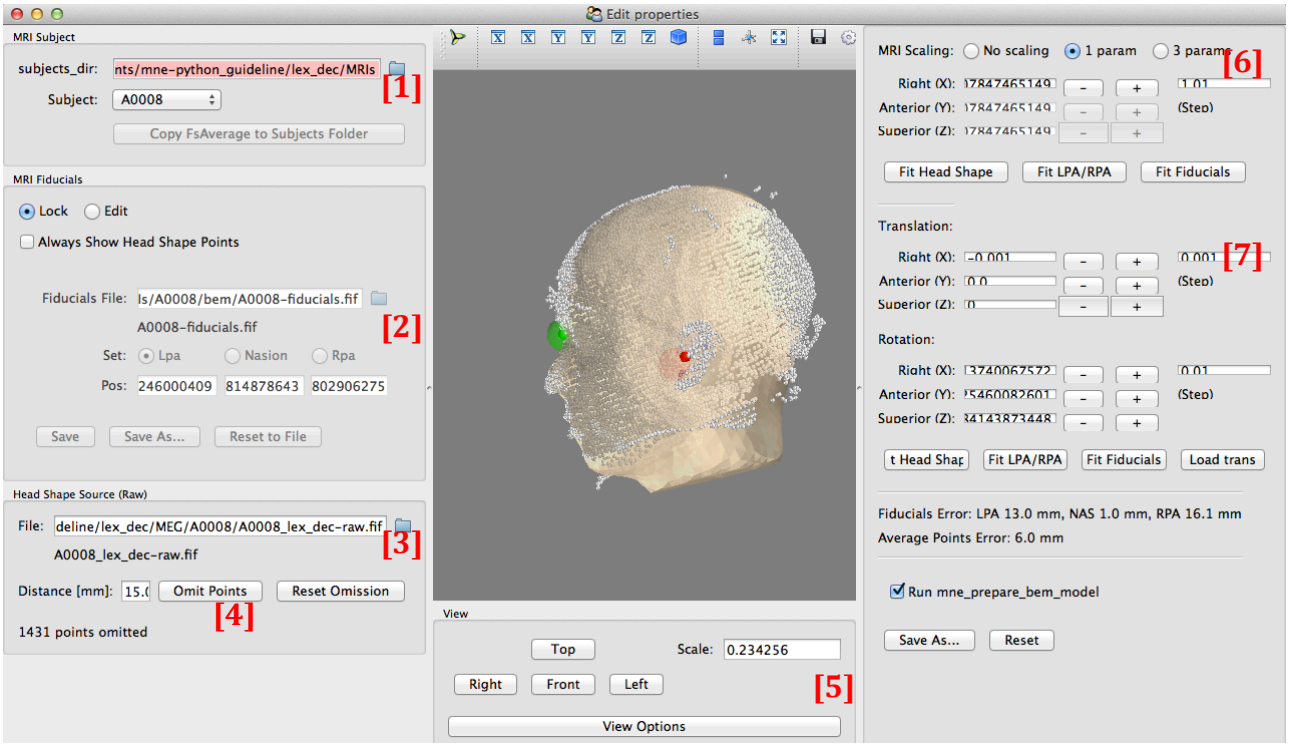
Now it's time for coregistration. In this step we create the FsAverage brain for the subjects if we do not have MRI data. This is then scaled to the headshape of the participant.

If you do have MRIs you're your participants, this is loaded at this point also.

Hightlight step 4 in the script:

```
>>> mne.gui.coregistration()
```

This will pull up a Graphic User Interface (GUI):



[1] **Subjects_dir** = the MRI folder for all of your participants
[2] **Fiducials file** = bem fiducicals created by freesurfer
[3] **Head Shape Source** = raw .fif file created in previous step
[4} **Omit** outlier head-shape points
[5] **View options** = Change opacity of the average head
[6] **Scaling** = Choice of scaling for fsaverage head
[7] **Translation** = rotation and movement of scan to fit fsaverage

Note: You may want to click the little green + button to maximize the window to see all of the gui screen.

a. First, ensure the MRI folder path is selected within the subjects_dir box on the top left [1]. If you correctly input this in Step 1, this should be the case.

b. Click 'Copy FsAverage to Subjects Folder'. This requires that freesurfer was also successfully downloaded and defined. This is only done once per experiment.

If you get an error regarding MNE_ROOT or freesurfer root then double check the path you specified to the MNE and freesurfer folders in applications in step 2.

c. Now, the fiducials should have been automatically loaded [2]

d. Load the .fif file [3] that we just created in the previous step. You should now see the overlay of the HS on the fsaverage head.

e. Remove outlaying points by entering a distance [mm] from the average head [4]. 15mm is usually a good threshold. If you find that any points were unintentionally deleted, click 'Reset Omission' and enter a larger number.

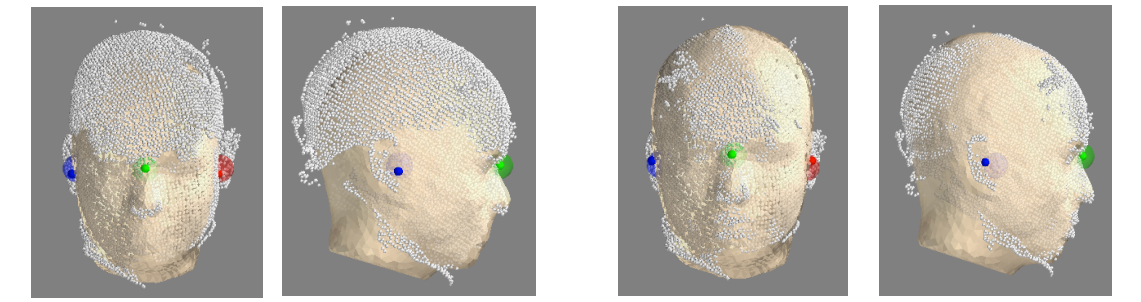
f. Make the average head more opaque by clicking 'view options' [5] and changing opacity to around 0.7. This will make it much easier to line up the head.

g. In MRI scaling [5] click '1 param'. This means that the fsaverage head will be scaled and rotated according to the best fit of the nasion. You can then make manual alteration on the basic of the left and right pointers. If you pick 3 parameters, it means that all points used to automatically fit the head shape. This is often less accurate than doing it manually, but if you chose this method be consistent for all subjects.

h. Click 'Fit head shape' both in the MRI scaling and the Translation/Rotation box

i. Adjust the size, translation and rotation of the scan by clicking the plus and minus buttons. You should be aiming to see the pointers and the fiducials line up. The head scan should also be placed on the fsaverage head as if it were the cap of the real participant.

Note: Remember that the person's head is smaller than the cap, and therefore smaller than the scan. So the headscan points should be over the head, not at the same point (unless they had no hair). It's good to keep a copy of the estimated distance between skull and cap – i.e., how much hair they have to be as accurate as possible in the coregistration.



Before **After**

In the boxes above, you can see that before the headshape is scaled and adjusted, the fiducials do not match up, and the grey points which make up the head scan do not accurately portray where the cap would have been relative to the person's head. Afterwards, the fiducials line up, and there is a little room towards the back to the head to allow for the person's hair under the cap.

i. Once everything looks okay, click 'save as'. Save the participant –trans.fif file in their MEG participant folder, in the same location as the –raw.fif file.

> **Note:** Again, you can do this for each person that you have a .fif file for and cycle through each in turn. Click 'reset' and then load up the next person's raw data. There is no need to copy FsAverage files – this is only done once. The previous participant will be processed in the background and a queue will be created. You need to wait until all mri's are being processed until you can close the window. If you have created a long queue, this may take a while.

4b. Set up source space

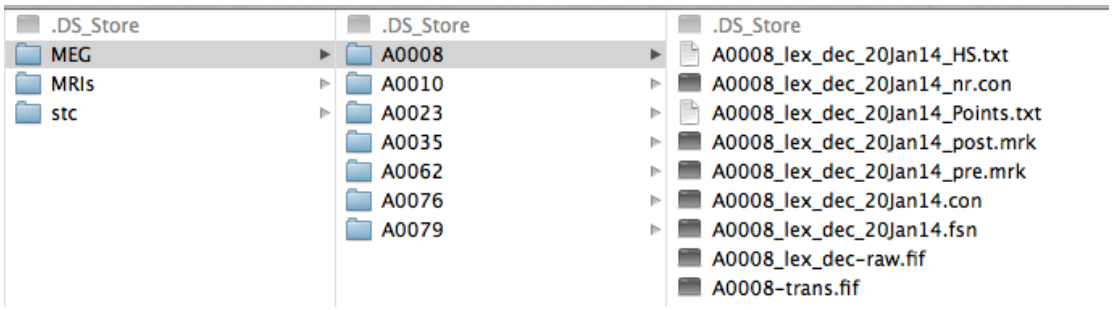
Once this is completed, we need to do a final step in the editor window. Run the setup_source_space command in order to create an ico4 source space file for the subject. Ensure you completed step 2 properly or this will not work.

```
    Write a source space...
    [done]
    2 source spaces written
Wrote /Volumes/My_Passport/Documents/Experiments/mne-python_guideline/lex_dec/MRIs/A0008/bem/A0008-ico-4-src.fif
You are now one step closer to computing the gain matrix
Out[9]: <SourceSpaces: [<'surf', n_vertices=163842, n_used=2562>, <'surf', n_vertices=163842, n_used=2562>]>
```

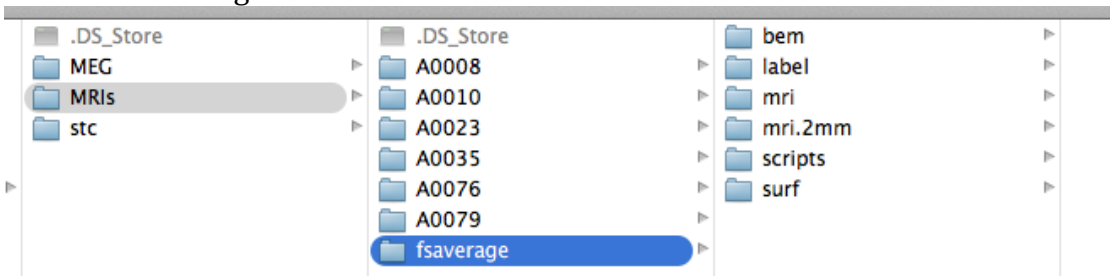
Check File Status

The coregistration step populates the participant folder in both the MEG and the MRI directory.

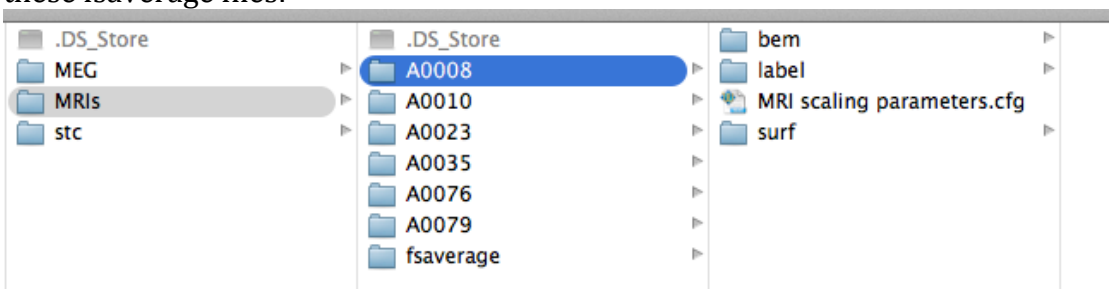
Your MEG folder should now contain a '-trans.fif' file:



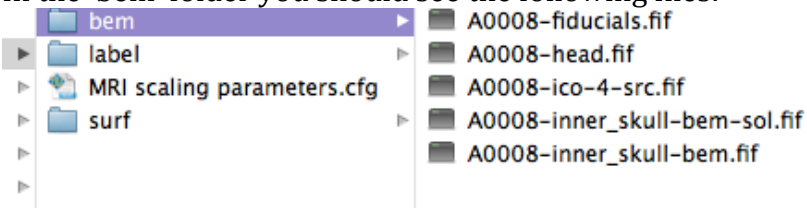
Your MRIs folder will now also contain an additional 'fsaverage' folder which houses the average bem and surface files:



The participant folder in the MRIs directory should now house scaled versions of these fsaverage files:



In the 'bem' folder you should see the following files:



If you do not see the ico-4-src.fif file, check you have completed the step setup_source_space command correctly. This is crucial for the inverse solution.

5. Read the data

Designate a name to the mne object which will be assigned to your raw data. In the script is has been designated the name 'raw'. This is an arbitrary choice but a standard in the pipeline.

```
>>> raw = mne.fiff.Raw('<path_to_raw_.fif_file>, preload = True')
```

We use the 'preload' argument as True in order to low pass filter the data later. This loads the raw file into the computer memory.

If the file cannot be found, check the following things:

1) Did you name the file correctly?
2) Is your current directory the MEG folder?
3) Have you loaded the 'participant' and 'experiment' variables?

You will see that the script has the path already partially designed for you. You need to make sure you are in the MEG directory for this path to work. Your .fif file needs to follow the structure '<participant>_<experiment>-raw.fif'.

Run the first line in step 5. You should have something like this as an output:

```
In [36]: raw = mne.fiff.Raw( participant + '/' + participant + '_' + experiment + '-raw.fif',
preload = True)
Opening raw data file /Volumes/My_Passport/Documents/Experiments/mne-
python_guideline/lex_dec/MEG/A0008/A0008_lex_dec-raw.fif...
Current compensation grade : 0
    Range : 0 ... 1324999 =      0.000 ...  1324.999 secs
Ready.
Reading 0 ... 1324999  =      0.000 ...  1324.999 secs...
[done]
```

In order to access information from any python object, simply type 'print' and then the name of the object itself:

```
>>> print raw
 <Raw  |  n_channels x n_times : 257 x 2630000>
```

You can now have a look at the different pieces of information your data set holds:

```
>>> raw.info
```

To look at one of these dicts, type the raw data name, .info followed by the field of interest in quotations within square brackets:

```
>>> raw.info['ch_names']
```

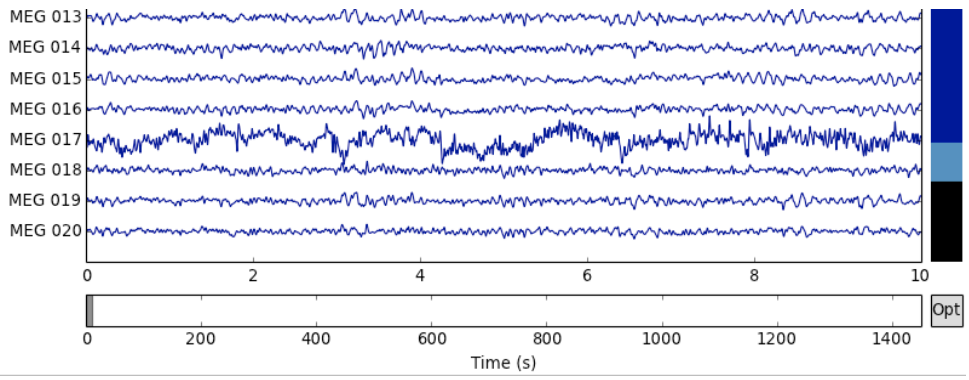
6. Visualise data

Take a look at the channels for your raw data. Highlight and run step 6:

```
>>> raw.plot()
```

This will produce a channel / time plot. You should use this information to check for any bad channels in the recording. Just press 'down' to look at all the channels in turn. If there are any noisy channels, make a note of these. In the plot below, we can see that channel 'MEG 017' [1] should be defined as 'bad'.

[1]



You may also notice that the channels begin to be counted at '1' in mne, so channel 16 on the sensor display during recording will appear as channel 17 on this plot. This is not a problem, just be aware of this fact if you have some consistently noisy channels the numbers displayed here will be one higher than you may have expected.

7. Define 'bad' channels:

```
>>> raw.info['bads'] += ['MEG 017', 'MEG 027']
```

From the notes you made with plotting the data, use this command to define the bad channels. Each channel should be separated by a comma and named the same as seen in the plot. You can check which channels have been defined as bad by entering 'raw.info['bads'] again:

```
In [40]: raw.info['bads']
Out[40]: ['MEG 017']
```

8. Low pass filter:

Next we need to low pass filter the data. Usually this is done at 40hz. Run the first line of step 8:

```
>>> raw.filter(0, 40, method = 'iir')
```

The method 'iir' corresponds to 'Infinite-impulse-response' filtering. This a forward-backward filtering technique rather than the default overlap-add FIR filtering. It has been shown to have better results than the default which is why it has been selected for this pipeline.

To pull up the information later about which filtering has been applied to the raw data, simply enter:

```
>>> raw.info['lowpass']
```

Next we need to save the filtered raw file to disk. You can see that the file structure is again provided in the script, so the low pass filtered .fif file will be saved to the participant's folder. Run the second line of step 8. On the screen you will see 'writing…. [done]' many times until it has finished saving.

```
In [45]: raw.save( participant + '/' + participant + '_' + experiment + '_lp-raw.fif', overwrite = True)

Writing ...
[done]
Writing ...
[done]
```

9. Load events

Now you need to tell mne to look at your raw file and extract the events – i.e., what triggers were sent and which timepoints. To do this, you need to provide the raw data and the stimulus channel:

```
>>> events = mne.find_events(raw, stim_channel='STI 014', min_duration
=0.002)
Reading 0 ... 41699  =      0.000 ...   277.709 secs... [done]
319 events found
Events id: [ 1  2  3  4  5 32]
```

Minimum duration is set to avoid 'false' triggers being detected. By default, the stim channel is 'STI 014'. If you are using a different system (i.e., a newer system that uses STI 101 by default) you will not see the triggers appear when you test it with:

```
>>> print events[:5]
[[6994    0    2]
[7086    0    3]
[7192    0    1]
[7304    0    4]
[7413    0    2]]
```

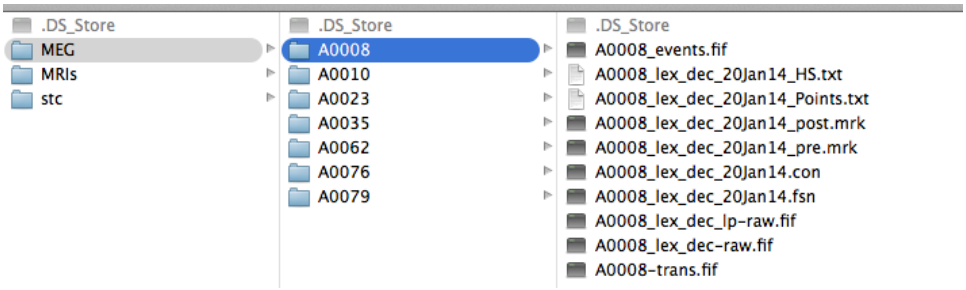
The first column is the time the trigger was sent, and the last column is the trigger number. If you see no trigger numbers, change the STI to 'STI 101'. You should look over this to make sure that the triggers you expect to appear to indeed appear. If you have triggers missing, the next step will be unsuccessful.

Finally, save the events to file to give the option of loading them from disk later.

```
In [12]: mne.write_events( participant + '/' + participant + '_events.fif',events)
```

Check File Status

The participant folder within the MEG directory should now contain the low-pass filtered .fif file, and an events file. It should look something like this:



10. Define events

Now you need to define which triggers correspond to which events. This is done by making a 'dict' list for each event of interest:

```
event_id = dict(<cond1>=1, <cond2>=2)
```

Make sure you name your conditions with a single simple name, as these will be used constantly throughout the script. Each name you give your conditions now should consistently replace all instances of <cond1> etc.

The name you use for the conditions here needs to correspond to your stc folder.

If you have more than one trigger per condition, don't worry. Just put the trigger-event information. So, for example, say you have two main conditions of interest, which are 'word-class' (noun or verb) and 'emotion' (positive, negative or neutral), you should define six event_id's:

```
In [14]: event_id = dict(noun_pos = 11, noun_neg = 13, noun_neut = 19, verb_pos = 21,
verb_neg = 64, verb_neut = 128)
```

You later have the option of collapsing trigger schemes such as this into simpler arrays, i.e., in order to compare all nouns to all verbs, or all negative words to all positive words.

Once this is done, check they have been input correctly by typing 'event_id':

```
In [15]: event_id
labelled correctly.

Out[15]:
{'noun_neg': 13,
 'noun_neut': 19,
 'noun_pos': 11,
 'verb_neg': 64,
 'verb_neut': 128,
 'verb_pos': 21}
```

11. Define epochs:

Now we need to make a time epoch around the trigger. So, think about how much time you want to be extracted before and after the trigger. The time before is 'tmin', the time after is 'tmax':

```
>>> tmin = -0.1  # start of each epoch (100ms before the trigger)
>>> tmax = 0.6  # end of each epoch (600ms after the trigger)
```

And we also need to define all of the 'good' channels – all of the channels apart from the ones we defined as 'bad' in step 7:

```
>>> picks = mne.fiff.pick_types(raw.info, meg=True, stim=False,
exclude='bads')
```

Next we define the baseline, which is taken from the tmin up to time 0:

```
>>> baseline = (None, 0)  # means from the first instant to t = 0
```

Now we are ready to create the epochs with the parameters we have defined:

```
    ...: epochs = mne.Epochs(raw, events, event_id,tmin, tmax,
proj = True, picks = picks, baseline=baseline, preload = True)

0 projection items activated
302 matching events found
Applying baseline correction ... (mode: mean)
Applying baseline correction ... (mode: mean)
Applying baseline correction ... (mode: mean)
Applying baseline correction ... (mode: mean)
0 bad epochs dropped

In [16]:
```

Print the epochs and make sure that the frequency of the conditions makes sense as to what you are expecting.

```
>>> print epochs <Epochs  |  n_events : 600 (good & bad), tmin : -0.1 (s),
tmax : 0.6 (s), baseline : (None, 0),  'noun_pos': 100, 'noun_neg': 100,
'noun_neut' : 100, 'verb_pos' : 100, 'verb_neg' : 100, 'verb_neut' : 100>
```

You will see that at the moment the events include both the good and the bad epochs. This is because we haven't done the rejections yet. At this point, the count of epochs should reflect what is in your experimental design and your log files.

If you don't have the correct number of events for your conditions, check your event_id description.

If you're happy and everything looks as it should, save the result to disk.

12. Blink rejection

Now we need to look at each epoch and reject any that look like they contain blinks or other unwanted artifacts.

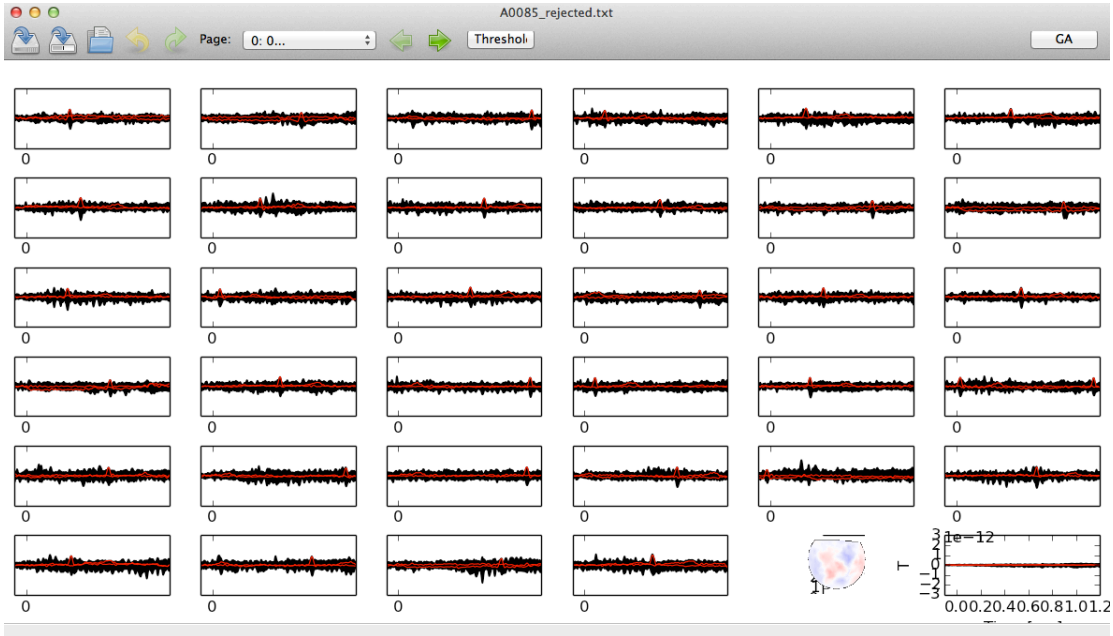
Make sure your backend is switched to interactive wx or this won't work.

The set_vlim is an automatic rejection which tries to figure out which of your trials are definitely no good. You can always choose to go against what this suggests, but it's usually accurate.

Run the first 4 lines.

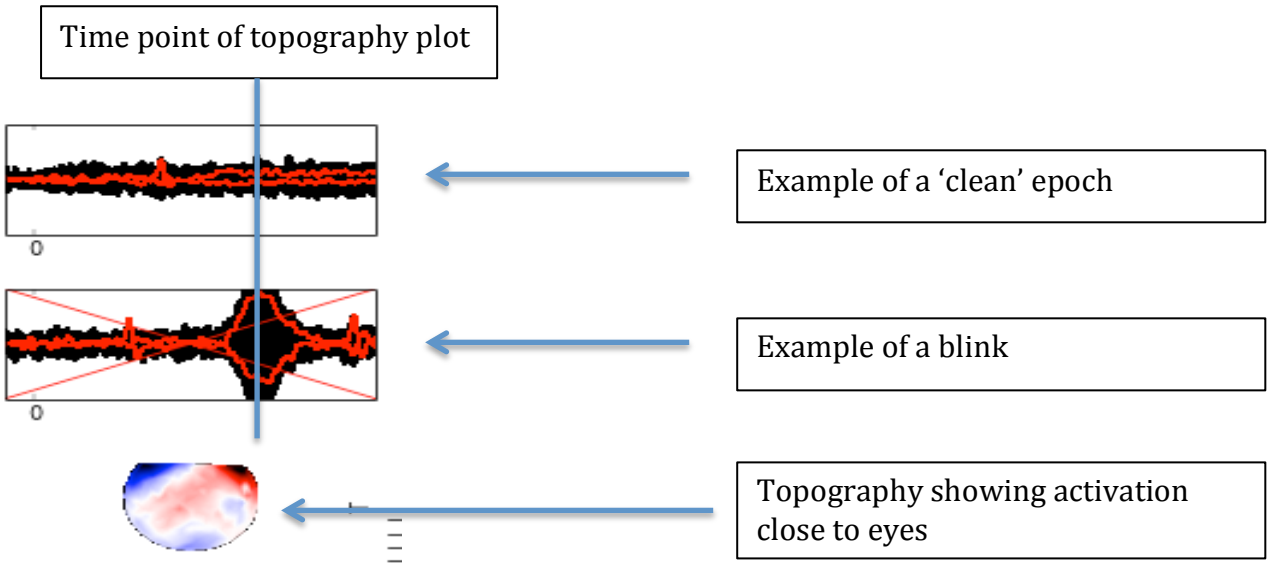
```
In [10]: ds = Dataset()
    ...: ds['epochs'] = epochs
    ...: ds['trigger'] = Var(np.ones(ds.n_cases))
    ...: g = gui.SelectEpochs(ds, data='epochs', path = root + '/' + experiment + '/' + participant + '/' +
participant + '_rejected.txt', mark = ['MEG 087', 'MEG 130']) # path to save rejections #
```

Click 'OK' about not finding eyelink data. You will see a gui interface with each of your epochs in a box. This may take a few seconds to load. The black shape represents all of the channels and the red lines show the two channels at each eye.



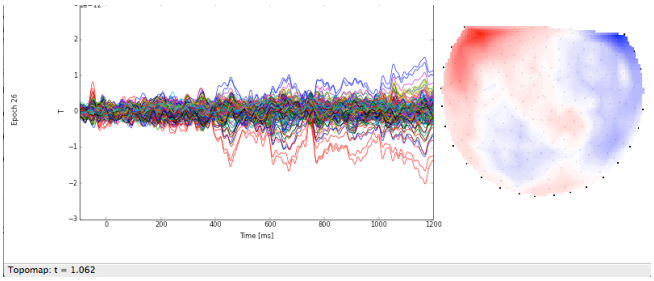
To reject a trial which contains a blink, simply click on the box that contains the divergent epoch. This box will then show a red cross through it. If you change your mind, simply click the box again and the cross will disappear.

Blinks will show themselves as divergence from one another, as well as activation close to the front of the head. Hover the mouse over a time point to see the topography of channels at the bottom.

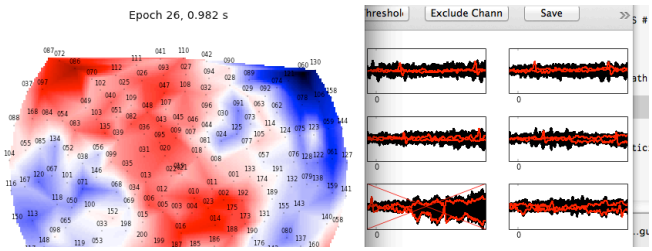


You can pull up a topography or butterfly plot by hovering the mouse of the time of interest and pressing 't' or 'b' respectively.

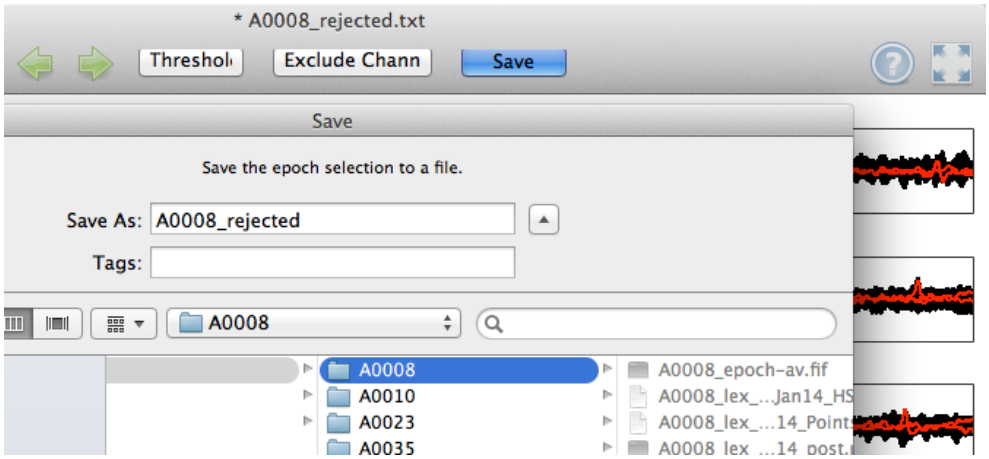
Topography plot – Press T



Butterfly plot – Press B



Click on each trial that you would like to reject, and press the arrow to move forward until you've looked at them all. Click 'save' and then exit with the red cross.



Now run the last 4 lines:

```
In [13]:
    ...: blink = load.tsv( participant + '/' + participant + '_rejected.txt')
    ...: idx = blink['accept'].x
    ...: epochs = epochs[idx]
```

This is combining the rejection list we just created with the epochs, so that we are now only using the 'clean' epoch data.

To see how many trials were rejected, compare the original frequencies compared to the output of typing 'print epochs'.

```
>>> print epochs <Epochs  |  n_events : 575 (all good), tmin : -0.1 (s),
tmax : 0.6 (s), baseline : (None, 0),  'noun_pos': 97, 'noun_neg': 98,
'noun_neut' : 91, 'verb_pos' : 99, 'verb_neg' : 94, 'verb_neut' : 96>
```

Now the count of epochs corresponds to only the 'good' remaining trials. Compare this result to the original output to see how many trials were rejected per condition. The details of this are saved in the <participant>_rejected.txt file.

In order to see which trials specifically were rejected, call a boolian index by typing 'idx':

```
In [15]: idx
Out[15]:
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True, False,
        True,  True,  True,  True,  True, False,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True, False,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True  True  True  True  True  True  True  True  True
```

13 Make epochs

13a. OPTIONAL -- not one-to-one trigger-condition correspondence:

If your conditions are made up of simultaneous triggers, you will require an additional step.

```
combine_event_ids(epochs, [‘cond1a’, ‘cond1b’, ‘cond1c’], {‘cond1’: 10},
copy = False)
```

Here, we are combining each subset of cond1 (a, b, c,) into a single trigger code (10).

For example, say we have two kinds of words (verbs and nouns) and three kind of connotative meaning (positive, negative or neutral), each will a different trigger code. If we want to make comparisons between all nouns and all verbs, or just the positive vs. negative words we need to combine these conditions, thus:

```
combine_event_ids(epochs, ['noun_pos', 'noun_neg', 'noun_neut'], {'noun': 10}, copy = False)
combine_event_ids(epochs, ['verb_pos', 'verb_neg', 'verb_neut'], {'verb': 20}, copy = False)
combine_event_ids(epochs, ['noun_pos', 'verb_pos'], {'pos': 30}, copy = False)
combine_event_ids(epochs, ['nouns_neg', 'verb_neg'], {'neg': 40}, copy = False)
```

Once this has been completed for all conditions of interest, you can move to step 13b.

You can include any combination of conditions as long as you created an event_id for them in step 10. Each condition can then be called with epochs[‘<cond>’]:

```
>>> epochs[‘noun’]
>>> <Epochs  |  n_events : 386 (all good), tmin : -0.1 (s), tmax : 0.6 (s),
baseline : (None, 0),  'noun': 386, ‘verb’: 0, ‘pos’ : 0, ‘neg’ : 0>
```

13b. Make epochs for each condition:

This part is the same regardless of whether you did step 13a or not.

To make epochs for each condition, we have to get the data for each condition and save it to its own epoch. For each condition defined with an event id, run the following:

```
>>> epochs['cond1'.get_data()
epochs_cond1 = epochs[‘cond1’]
```

Remember to save each epoch condition as ‘epoch_<yourconditionhere>’. Also make sure that you are putting in the correct condition for each of these entries.

The completed lines should look something like this:

```
epochs['noun'].get_data()
epochs_noun = epochs['noun']
epochs['verb'].get_data()
epochs_verb = epochs['verb']
epochs['pos'].get_data()
epochs_pos = epochs['pos']
epochs['neg'].get_data()
epochs_neg = epochs['neg']
```

14. Read epochs and make evoked

This step averages each epoch and creates evoked data. This is then saved to disk:

```
In [23]:
    ...: evoked = epochs.average()
    ...: evoked.save( participant + '/' + participant + '-evoked-av.fif')
    ...:
```

14b. Evoked for each condition:

Next, we average individually for each condition. Here we input the details of each condition we want to average.

```
>>> evoked_cond1 = epochs_cond1['cond1'].average()
```

For example:

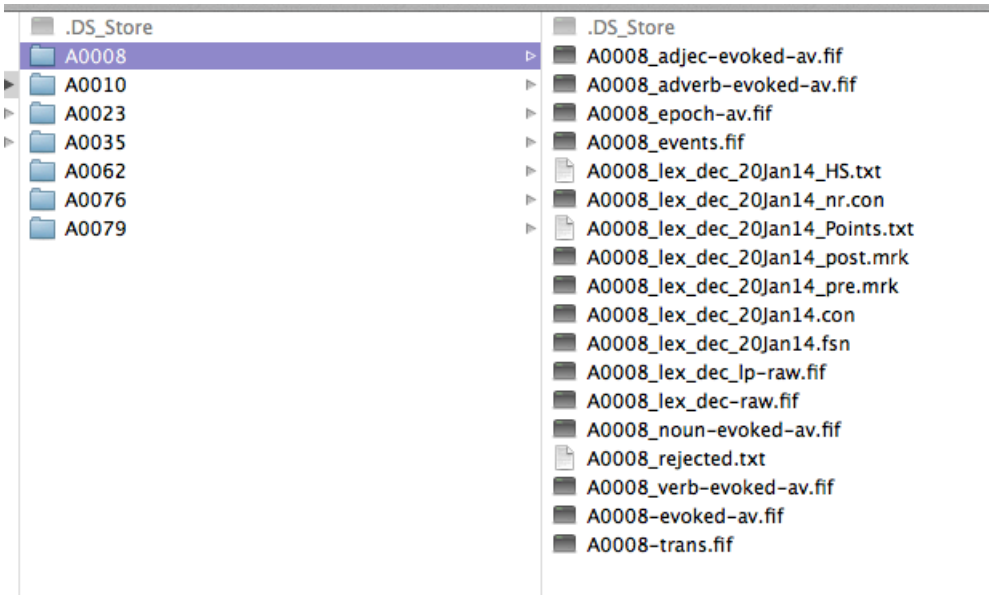
```
evoked_noun = epochs_noun['noun'].average()
evoked_noun.save( participant + '/' + participant + 'noun-evoked-av.fif')

evoked_verb = epochs_verb['verb'].average()
evoked_verb.save( participant + '/' + participant + 'verb-evoked-av.fif')
```

This requires some tedious script-changing, so make sure that you have named each condition correctly and consistently with what is detailed previously.

Check File Status

The are a number of files which have now populated the participant folder in the MEG directory:



We now have a file which holds the 'event's information, an averaged epoch and rejected 'bad' trials. You will also see that there is an averaged evoked response across all conditions, in addition to an evoked response per condition.

If any of these files are missing, check you have run the corresponding step correctly. Also ensure at this point that your files follow the right naming conventions..

15. Covariance Matrix

This step computes a noise-covariance matrix of all the MEG channels with themselves. This matrix is later used in the computation of the inverse solution. This can be done using continuous empty room data or epochs from experiment.

Note: the epochs used must be baseline-corrected, otherwise the computation will be incorrect.
In the mne editor, run the first two lines. You should get an output similar to this:

```
In [21]:
    ...: cov = mne.cov.compute_covariance(epochs)
    ...: cov = mne.cov.regularize(cov, evoked.info, mag=0.05, grad = 0.05, proj = True, exclude = 'bads')

Number of samples used : 348668
[done]
0 projection items activated
    EEG regularization : None
    MAG regularization : 0.05
    GRAD regularization : None
```

Here, we are using the baseline of the epochs (the pre-stimulus period) to compute the covariance. '0' refers to the end point of the baseline. If this is not provided as an argument, the whole length of the epoch will be used to compute the noise covariance.

The regularisation uses magnetometer and gradiometer output factors set to 0.05. The bad channels are not included in this computation.

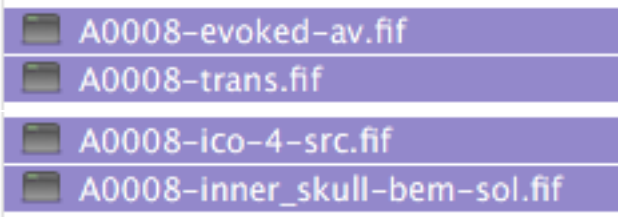
Finally, save the output to disk:

```
In [36]: mne.write_cov( participant + '/' + participant + '_cov.fif', cov)
```

16. Forward solution:

Now we compute the forward solution using the:

1. evoked average	(step 14)	= info
2. mri –trans file	(step 4)	= mri
3. bem ico file	(step 4)	= src
4. inner skull	(step 4)	= bem



The paths to each of these files has been entered for you. Please make sure that your working directory is still set to the MEG folder, and that you have correctly saved the participant and experiment objects.

Once you are confident this is the case, run all 6 lines:

```
# STEP 16 # -- FORWARD SOLUTION # PAGE 23 #

info = participant + '/' + participant + '-evoked-av.fif'
mri = participant + '/' + participant + '-trans.fif'
src = subjects_dir + '/' + participant + '/bem/' + participant + '-i
bem = subjects_dir + '/' + participant + '/' + '/bem/' + participant
fname = participant + '/' + participant + '_forward.fif'

fwd = mne.make_forward_solution(info = info, mri = mri, src = src, b
```

If you receive an error saying 'no such file or directory' check the following:

1) participant and subjects_dir variables named correctly
2) working directory is the 'MEG' folder
3) the files are named correctly

One this is completed, you should see a '<participant>_forward.fif' file in the MEG participant directory.

```
Composing the field computation matrix...
Computing MEG at 5124 source locations (free orientations)...

writing A0008/A0008_forward.fif...
Finished.
```

17. Inverse operator:

This requires a number of inputs which are loaded from memory rather than from disk:

1. evoked.info	(step 14)	= info	(number of sensors)
2. fwd	(step 16)	= forward	(forward solution)
3. cov	(step 15)	= noise_cov	(covariance matrix)

Loose, depth and fixed refers to the kind of dipole orientation and weighting you require. For this pipeline, we use free orientation, which created unsigned data. If you want signed data, use “fixed” orientation, which signs the directionality with respect to the cortex.

If you require a different inverse solution, then consult this table and select the necessary forward parameters when computing the forward solution:

Inverse desired	Forward parameters allowed				
	loose	**depth**	**fixed**	**force_fixed**	**surf_ori**
Loose constraint, Depth weighted	0.2	0.8	False	False	True
Loose constraint	0.2	None	False	False	True
Free orientation, Depth weighted	None	0.8	False	False	True
Free orientation	None	None	False	False	True | False
Fixed constraint, Depth weighted	None	0.8	True	False	True
Fixed constraint	None	None	True	True	True

Run the command and save the result:

```
In [42]: inv = mne.minimum_norm.make_inverse_operator(evoked.info, fwd, cov, loose = None, depth = None,
fixed = False)

Computing inverse operator with 207 channels.
Setting small MEG eigenvalues to zero.
Not doing PCA for MEG.
Total rank is 207
Computing inverse operator with 207 channels.
Creating the source covariance matrix
Whitening the forward solution.
Adjusting source covariance matrix.
Computing SVD of whitened and weighted lead field matrix.
    largest singular value = 3.99585
    scaling factor to adjust the trace = 1.31459e+20
```

```
In [43]: mne.minimum_norm.write_inverse_operator( participant + '/' + participant + '_inv.fif', inv)

Write inverse operator decomposition in AD020/AD020_inv.fif...
```

You should now see a '<participant>_inv.fif' file in your MEG participant folder.

18. Create source time estimates

To create stc's for each condition, again we need to input various files and parameters. If you are familiar with BESA, this is the equivalent to batching to create the .dat files.

Signal to noise ratio (SNR)

The SNR refers to how much usable signal is obtained relative to the degree of background noise. It has been defaulted at 2 for the current purposes, but please check if you are unsure what ratio is used in your lab.

Lambda Regularisation

The lambda argument is used to help fit the inverse solution relative to the SNR. The standard way of computing this parameter is 1 divided by the signal to ratio, squared.

```
snr = 2.0
lambda2 = 1.0 / snr ** 2.0
```

Method

There are a number of 'methods' that can be used to apply the inverse solution. The default method is dSPM (dynamical Statistical Parametric Mapping) which is a noise-normalised estimate based on the MNE solution. Another option is the 'MNE' (minimum norm estimate), which is a depth-weighted linear algorithm. This method is selected used in the BESA computation.

```
method = 'dSPM'
```

Inverse Solution

The inverse solution as saved in memory is also included in this computation. It should have been named 'inv'.

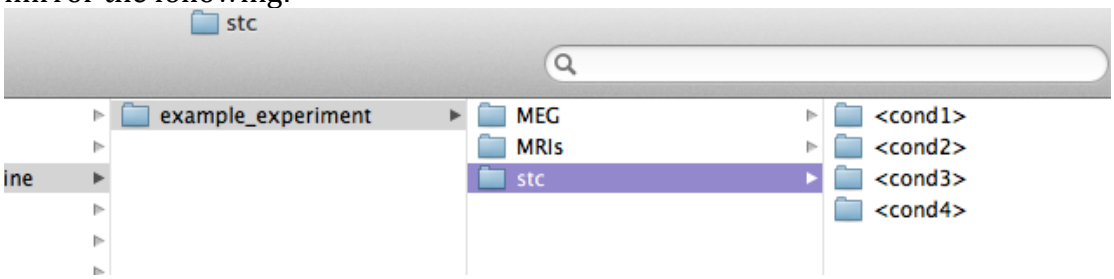
Run the command

Once you are happy with each of these parameters, first run the three lines which set up the parameters:

```
In [44]: # STEP 16 # -- APPLY INVERSE TO GET SOURCE TIME ESTIMATES # PAGE 24 #
    ...:
    ...: snr = 2.0                                          # signal to noise ratio #
    ...: lambda2 = 1.0 / snr ** 2.0
    ...: method = 'MNE'                                     # how do you want to apply the inverse
solution? #
    ...:
```

Set up the stc folder

First, if you haven't done so already, make a folder for each of your conditions, housed within the 'stc' folder. This is where the condition stc files will be saved for each of your subjects. Your folder structure within the stc folder should mirror the following:



It is important that all of the condition folders have the exact same name as you input in the script.

Save the stc files

Modify the following lines to include your specific condition information. Remember you can type 'epochs' at any point to pull up the specific condition names you have used.

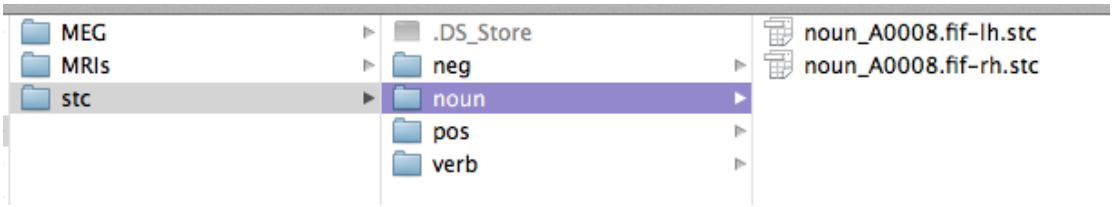
```
stc_noun = mne.minimum_norm.apply_inverse(evoked_noun, inv, method = method, lam
stc_verb = mne.minimum_norm.apply_inverse(evoked_verb, inv, method = method, lam
stc_neg = mne.minimum_norm.apply_inverse(evoked_neg, inv, method = method, lambd
stc_pos = mne.minimum_norm.apply_inverse(evoked_pos, inv, method = method, lambd
```

Once the stc objects have been created, we can then save them to disk to the 'stc' folder. This will save two files in each condition folder - one for each hemisphere. Once this is complete, the lines should look like this:

```
stc_noun.save( stc_path + '/noun' + '/noun_' + participant + '.fif')
stc_verb.save( stc_path + '/verb' + '/verb_' + participant + '.fif')
stc_neg.save( stc_path + '/neg' + '/neg_' + participant + '.fif')
stc_pos.save( stc_path + '/pos' + '/pos' + participant + '.fif')
```

Check File Status

If this has been completed correctly, you should now see the two files in each condition folder within the stc directory:



The two files correspond to the left and right hemispheres. As you process each individual, all of the stc files will get saved to the same folder, but will be distinguishable based on the participant code.

This is the final step of the pipeline!

The participant folder within the MEG directory should now be fully populated with the following files:

The majority of these files are able to be loaded back into memory from disk for future use. This may be required if you begin to process an individual's data and need to go back to it later, or simply want to look at one file without going through the whole process again unnecessarily.

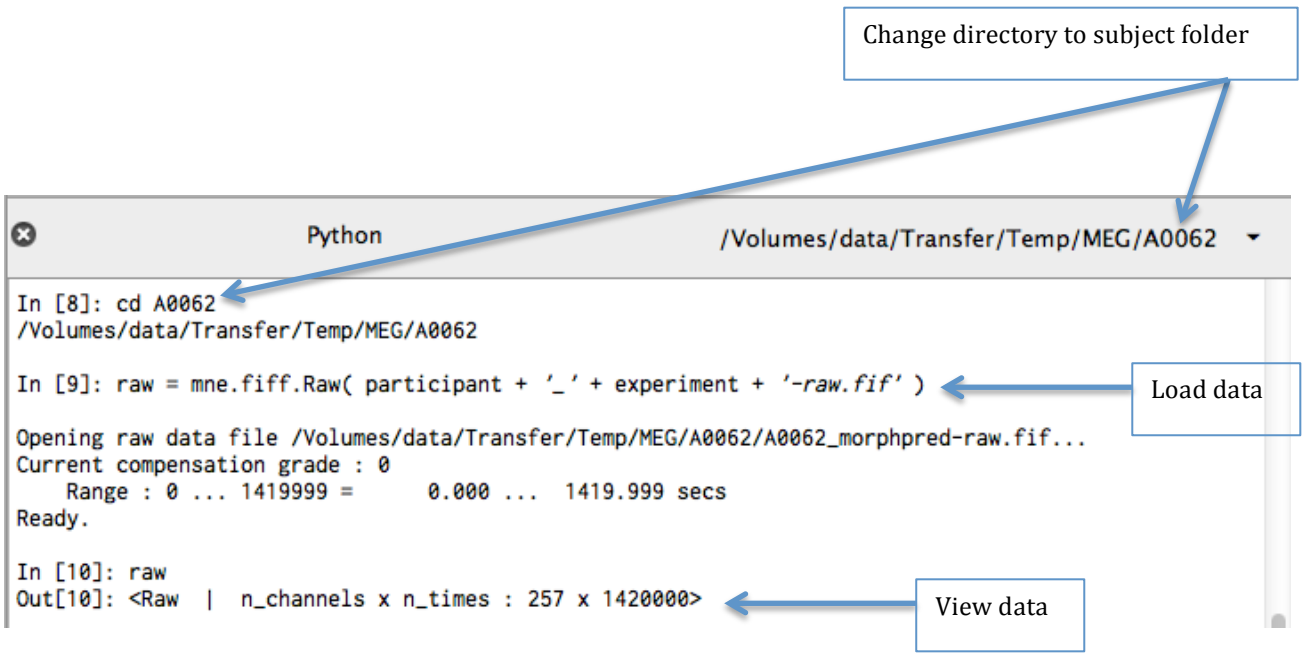
Loading saved files

At the bottom of the script you will see code to load your saved files back into the computer memory.

First, change your directory to the participant's folder within the MEG folder. In the screen shot above, this would be folder 'A0008'.

Then, ensure that steps one and two have been completed so that the 'participant' and 'experiment' objects have been created, in addition to the mne-python dependencies.

Next, run the line of code which corresponds to the object you which to load back into memory. You can then continue processing as if the object were 'newly' created.



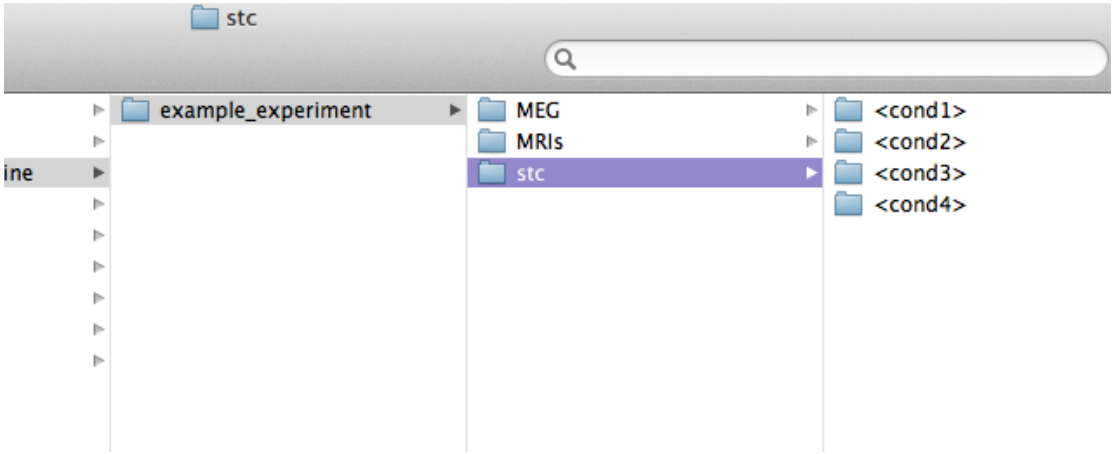
Note: If you continue processing from the main script, ensure that you have changed directory back into the MEG folder.

Statistical Analysis

Statistical Analysis using MRAT for MNE.

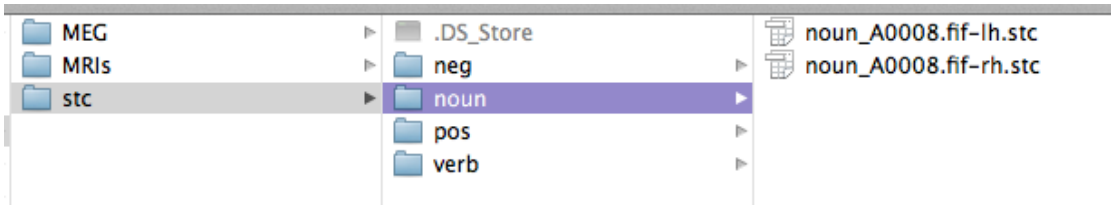
Set up the stc folder

Make a folder for each of your conditions, housed within the 'stc' folder. This is where the condition stc files will be saved for each of your subjects. Your folder structure within the stc folder should mirror the following:



Within each of the condition folders should be two stc files per subject – one for the left hemisphere and one for the right hemisphere.

For example, say my conditions of interest were 'neg', 'pos', 'noun', 'verb'. My *stc* folder should look something like this for the first participant:



Once all of the participants' stc files are in the correct condition folders, you now need to make sure that you include a 'labels' folder in your stc directory. This can be found on the server in the MRAT documentation on the path:

nellab_scripts/MNERegionViewer/ico4_labels_18012014

Simply copy this folder into your stc folder.

Now your stc folder should contain:

- Folders for each of your conditions
- Labels folder **called "labels"**

This is all you need to begin the MRAT analysis

MRAT Analysis in MATLAB

1) Open MATLAB and set the current working directory to your stc folder.

2) Set the path so that it includes (with subfolders) the MultiRegionAnalysis tool.

3) Enter 'MulitRegionAnalysis' in the editor. It should welcome you to the tool.

Note: If you get an error at this point, it means that the path has not been specified correctly

4) Follow the on-screen instructions.

5) When it asks for the data type, select 'mne' **not** 'text' or 'mat'.

6) Make sure that you specify the levels and factors of your experiment correctly.

Statistical Analysis Using mne-python

Alternatively, the data can also be analysed using mne-python packages. For more information about these packages, have a glance at the following website: http://martinos.org/mne/stable/auto_examples/index.html