EEG Toolbox Tutorial

This is a walkthrough tutorial on how to use the eeg toolbox codes to analyze EEG data. It is an amalgamation of the old eeg toolbox documentation found in the eeg toolbox itself (doc.pdf), Josh Jacobs and Nicole Long's tutorials. It is up to date as of **January 14, 2015**. Several key points:

- This tutorial assumes data will be analyzed on the computing cluster rhino; however, with the exception of *Accessing Rhino*, *Getting the toolbox*, and *Parallel Processing for Group Analysis*, all sections can be applied to locally stored data.
- This tutorial uses pyFR data and TJ039 as the example experiment and subject, but these codes can be applied to any experiment and subject so long as they are formatted similarly.
- This tutorial assumes that data has already been pre-processed. This includes: anonymizing, uploading data, creating event structures, splitting, re-referencing, aligning and artifact detection. If you would like a tutorial on those, please see

  `https://memory.psych.upenn.edu/InternalWiki/InternalWiki/DataAcquisition/Hospital_Documentation`
- **Note:** for all of the functions in the EEG toolbox, you can get help by typing `help` *functionname* at the matlab prompt. This will show you what the function does and what the input and output should be.

Contents

Accessing Rhino

If using UNIX-based systems, the connection to the rhino file server can be initiated through the terminal using the following command:

```
ssh user@rhino.psych.upenn.edu -X
```

To use macfuse and macfusion on snow leopard to mount the rhino file systems on your local machine:

1. Download and install both Fuse and SHFS from https://osxfuse.hithub.io/
2. Run the following script, which will mount Rhino in the home directory: `mkdir -p ~/rhino`

   ```
   sudo sshfs [username]@rhino.psych.upenn.edu:/-o allow_other ~/rhino/
   ```

Headnode vs. Worker nodes on Rhino

When you initially log on to rhino, you will be using the processor known as the "headnode". It is critical that you do not test out analyses or run code by launching matlab through the head node. Instead, you should log into a worker node via using: `qlogin -l h_vmem = 8G` where XG is how many gigs of memory you need, where 8 is the maximum.

Note: QLogin jobs won't quit automatically. If you quit terminal or lose connection with rhino, your qlogin job will still exist and be taking up memory. You either need to type "exit" in the terminal window which is qlogin'ed or, if that window is no longer available, you need to manually kill the job by typing qdel JOBNUMBER where JOBNUMBER is the leftmost number shown when typing qstat

Getting the toolbox

The EEG toolbox is a file library used to manipulate data from each patient. It can be obtained from SVN.

Setting up SVN

To setup SVN for use on the rhino server directly, add these lines to your .bashrc file in your home directory:

```
export SVNROOT=file://localhost/home/svn
export SVN_EDITOR="emacs -nw"
```

To setup SVN for use on a different machine, add these lines to your .bash_profile file:

```
export SVNROOT=svn+ssh://USERNAME@rhino.psych.upenn.edu/home/svn/
export SVN_EDITOR="emacs -nw"
```

(replace USERNAME with your actual rhino username)

Note: When running SVN using this procedure to access rhino remotely, it will ask you for your rhino password. You can follow this procedure to have it automatically log in and eliminate the need for typing your password: http://www.gotidea.net/Linux/sshowto.htmlGrieb01a

Checking out eeg toolbox and iEEGDataBase

To get the eegtoolbox, you must check it out using the following command. `cd` to the directory where you would like the codes stored. Then run the checkout command.

```
svn checkout $SVNROOT/eeg/eeg_toolbox/trunk eeg_toolbox
svn checkout $SVNROOT/eeg/iEEGDataBase
```

Note: You are checking out the trunk folder because these codes are the most up to date. There are other folders within eeg toolbox besides trunk; however, it is *very important* to be sure you are using trunk as different versions of codes sometimes use the same parameters, but occasionally in different orders.

Finally, add the directory eeg_toolbox/trunk (and its subdirectories) and iEEGDataBase to your matlab path.To add folders to path, select File –> Set Path –> Add with subfolders and add both the eeg_toolbox and the iEEGDataBase. Save a local pathdef file when prompted to do so.

Note If you would like to display graphics on your screen when running MATLAB on rhino, you will need to install XQuartz (http://xquartz.macosforge.org/landing/)

Subjects

Key codes
get_subs

The first step in analyzing EEG data is knowing which subjects to use and how to access a particular subjects' data. All intracranial data is stored on */data/eeg/* by subject number. The first two letters indicate the hospital (BW = Brigham and Women's, CH = Boston Children's Hospital, FR = Freiburg, TJ = Thomas Jefferson University, UP = Hospital of University of Pennsylvania). Within each subject's directory are subdirectories for each session of each experiment in which that subject ran. The data can easily be accessed by using the codes in this and the next section.

1. Load the subject structure for the experiment

```
subj = get_subs('pyFR');
```

2. Now there should be a 176x1 cell variable in your workspace called *subj* which lists all 176 patients who participated in pyFR.

```
whos
 Name        Size            Bytes  Class    Attributes

 subj      176x1             21534  cell
```

Event structures

Key codes
get_sub_events
filterStruct
inStruct
getStructField

The EEG toolbox is built around events structures, which are matlab structures that contain your behavioral data, and that can be used to retrieve the associated EEG data.

1. Load an event structure for one pyFR subject

```
[events] = get_sub_events('pyFR','TJ039');
```

2. This will load the variable *events* into the workspace

```
 whos
  Name        Size                Bytes  Class     Attributes

  events      1x3300            5805658  struct
```

3. The size of the events structure is equivalent to the total number of experimental events the participant completed. For TJ039, that was 3300 events. Therefore, every entry in the structure is a single event and all of the fields detail properties of that event.

4. As an example, the fifth event has these properties:

```
events(5)

ans =

      subject: 'TJ039'
      session: 0
         list: 1
    serialpos: 1
         type: 'WORD'
         item: 'KING'
       itemno: 145
     recalled: 1
        mstime: 1.3365e+12
     msoffset: 1
      rectime: 11342
    intrusion: -999
      eegfile: '/data7/eeg/TJ039/eeg.reref/TJ039_08May12_1744'
    eegoffset: 1420201
```

5. Information on all of these fields are described in detail on the lab wiki

   (see `https://memory.psych.upenn.edu/InternalWiki/Format_of_RAM_Events_Structures`). Briefly, this event is a study item, a word the participant saw on the screen and was instructed to remember for a later memory task. We can see from the "recalled" field that this item was indeed remembered, as 1 indicates recall and 0 indicates not recalled.

6. For electrophysiological analysis, the eegfile and eegoffset fields are critical as they point to the raw EEG file that contains this event and when in that file this event occurs.

7. To create a vector of data from a particular event structure field, you can either use `getStructField` or square/curly braces. To use `getStructField`, you pass the events structure and the desired field. The output variable will be a vector of that data, e.g. a vector of 1s and 0s for recalled/not recalled items.

```
recalls = getStructField(events,'recalled');
```

8. Alternatively, you can create a vector of data from a particular event structure field using square braces for integer values and curly braces for string values.

```
recalls = [events.recalled];
type = {events.type};
```

9. You can select subsets of events using the `filterStruct.m` function. filterStruct requires two inputs, an events structure which we've already loaded and a logical expression. As an example, to get all subsequently recalled events,

```
recEvents = filterStruct(events,'recalled == 1');
```

   To get all WORD events,

```
wordEvents = filterStruct(events,'strcmp(type,''WORD'')')
```

10. Instead of filtering to a subset of events, you may want a logical index of 1s and 0s across all events. To do this, you can use the function `inStruct.m`. The results from this function will tell you for a certain logical expression, e.g. 'recalled == 1' whether that is true for every event (so it yields a binary vector with the length of the events structure). Sometimes people get errors when using filterStruct, which could be caused by empty fields.

ERP

Key codes
gete_ms
GetRateAndFormat

ERP analyses, as well as spectral analyses, are primarily dependent on the function gete_ms (or its variant, gete, which takes parameters as samples instead of MilliSeconds), which calculates voltage for the events and parameters provided. It operates on the raw eegfile by breaking it into epochs based on the duration and offset information provided by the user.

Generate a voltage trace

1. Load the events for a single subject

```
[events] = get_sub_events('pyFR','TJ039');
```

2. Gete_ms has many inputs which are integral to the analysis being performed:

```
[EEG resampleFreq] = gete_ms(channel,events,durationMS,offsetMS,...

bufferMS,filtfreq,filttype,filtorder,resampleFreq,RelativeMS)
```

   (a) *channel*: The electrode for which you would like to calculate the voltage.
   (b) *events*: The events you would like to analyze
   (c) *durationMS*: The time window, in ms, you would like to analyze around each event.
   (d) *offsetMS*: How much time you would like to analyze before stimulus onset.
   (e) *bufferMS*: How much time to use as a buffer. This deals with edge effects.
   (f) *filtfreq*: The frequency at which you want to filter (optional)
   (g) *filttype*: The type of filter (optional)
   (h) *filtorder*: The filter order (optional)
   (i) *resampleFreq*: The rate at which to resample the data (optional)
   (j) *RelativeMS*: For use with baseline subtraction (optional)

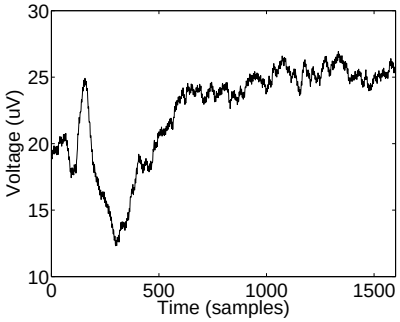
3. The values of these parameters will often change between experiments and depending on analyses. The examples provided here are typical values used for an encoding analysis (although we are getting the voltage for all events, including recalls).

```
[EEG] = gete_ms(30,events ,1600,0,1000, [58 62] , 'stop');
```

   (a) *channel*: 30 - in this example we are not using bipolar re-referencing, though see below
   (b) *events*: Single subject's events, loaded with get_sub_events
   (c) *durationMS*: 1600 - encoding duration for each pyFR event
   (d) *offsetMS*: 0 - no pre-stimulus information included
   (e) *bufferMS*: 1000 - standard buffer
   (f) *filtfreq*: [58 62] filters line noise - use a different value for data collected in Europe
   (g) *filttype*: 'stop'

4. This outputs a 3300 (events) x 1600 (samples) matrix of voltage values. The figure to the right shows the mean value of the EEG signal across all events and is plotted with the following commands

```
plot(nanmean(EEG);
ylabel('Voltage (uV)');
xlabel('Time (samples)');
```



Although it is not necessary to know the sample rate of the data to use gete_ms, you may want to know the sample rate for a particular subject/session. To do this, simply input events to the function `GetRateAndFormat.m`.

Power

Key codes
getphasepow
gethilbertphase
multiphasevec3

Similar to ERPs, our wavelet processing takes in a set of events and a channel in order to provide you with an events x frequencies x samples (time) array with power amplitude values (and if desired phase). The most useful function here is `getphasepow.m`, which requires very similar inputs as gete_ms. A lower level function, if you do not have events, but instead plain EEG, is `multiphasevec3`. Using `getphasepow` is convenient because it can easily deal with the buffer whereas directly using `multiphasevec3` requires some manual adjustment for the buffer.

Generate a time-frequency spectrogram

1. Load the events for a single subject

   ```
   [events] = get_sub_events('pyFR','TJ039');
   ```

2. The inputs to getphasepow are nearly identical to those for gete_ms with the exception of two additional optional parameters, *freqs* and *width*. *freqs* is the frequencies for which you would like to extract power. A common set of frequencies is typically 60 logarithmically spaced frequencies from 2 to 200 Hz.

   ```
   freqs = logspace(log10(2),log10(200),60);
   ```

   width is the Morlet wavelet number to use. A commonly used value is 6. Theoretical discussion of wave number can be found in "Analyzing Neural Time Series Data".

3. **A few notes**

   - The first output of getphasepow is **NOT** power, but phase. The second output is power.
   - By creating an eeganalparams.txt file in your home/eeg directory, you can skip inputting freqs and width to the getphasepow function.

4. Because calculating power is memory-intensive, we cannot extract spectral power for all 3300 of TJ039's events. First, we will filter to only events from the first session only.

   ```
   [events] = filterStruct(events,'session == 1');
   ```

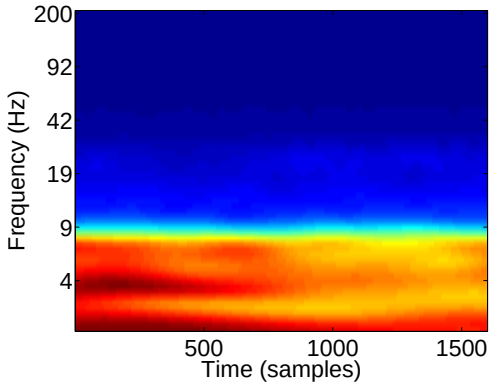
5. As with the ERP example, the parameters shown here are appropriate for pyFR encoding events.

   ```
   [~,pow] = getphasepow(30,events ,1600,0,1000,...
   'filtfreq', [58 62] ,'filttype', 'stop','freqs',freqs,'width',6);
   ```

6. This will output a matrix of 294 events x 60 frequencies 1600 samples. A spectrogram or time-frequency plot of these power values can then be plotted using the `imagesc.m` function.

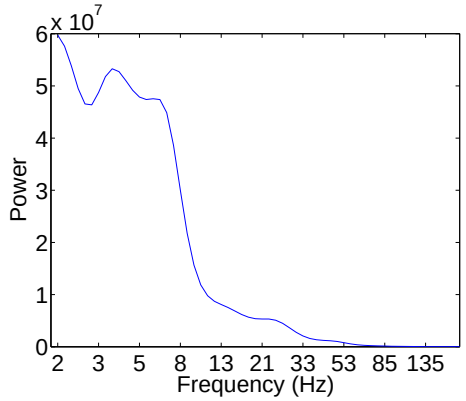
   ```
   imagesc(flipud(squeeze(nanmean(pow))));
   F = sort(round(freqs),'descend');
   set(gca,'ytick',1:10:60);
   set(gca,'yticklabel',F(1:10:60));
   set(gca,'xtick',[500 1000 1500]);
   set(gca,'xticklabel',[500 1000 1500]);
   set(gca,'Fontsize',25);
   ylabel('Frequency (Hz)');
   xlabel('Time (samples)');
   ```

7. Note: flipud inverts the matrix so that higher frequencies are plotted at the top of the figure.



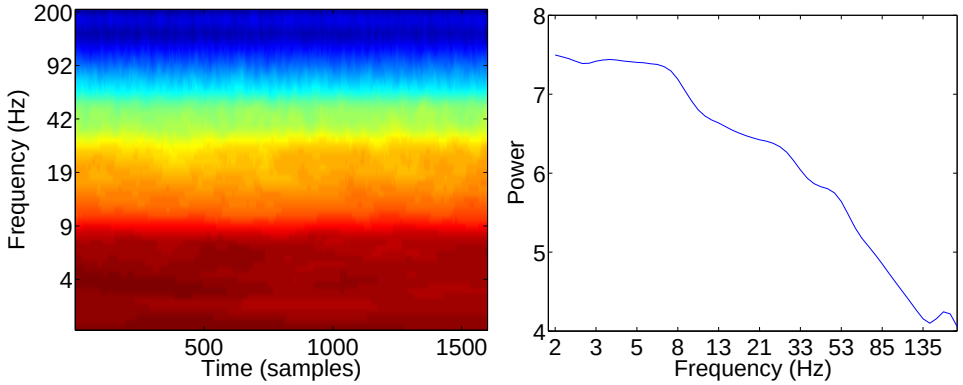
8. Or, you can average over the time dimension and plot power as a function of frequency. Both plots show the characteristic 1/f distribution that power values exhibit.

```
plot(squeeze(nanmean(nanmean(pow),3)));
F = sort(round(freqs),'descend');
set(gca,'xtick',1:10:60);
set(gca,'xticklabel',F(1:10:60));
set(gca,'Fontsize',25);
ylabel('Power');
xlabel('Frequency (Hz)');
```



9. Unlike voltage analyses, one additional step is required before continuing. Power values must be log transformed. There are several reasons to log transform power values, including comparing frequencies and running parametric analyses, see "Analyzing Neural Time Series Data" Chapter 18 for a discussion.

```
pow(pow==0) = eps(0);
pow = log10(pow);
```



Multitaper

For multitapers, we use very similar functions, where the main function is `mtphasepow.m`, which takes as input again the channel, events and desired durations. The optional parameters are slightly different: the bandwidth and windowsize of the multitapers. We have found that a bandwidth of 1 and windowsize of .3 work quite well, even though that uses only one taper. Note that you always have to adapt both windowsize and bandwidth together.

Pepisode

Pepisode is a power analysis developed by Jeremy Caplan in our lab and basically it computes the fraction of time an oscillation exceeding the background activity is present in your events. Calling these functions is very similar to calling power functions. However, it requires a pre-processing step, in which for every piece of EEG data, it is determined whether there are significant oscillations present at every frequency (significant oscillations exceed a duration and an amplitude threshold). This pre-processing function is called `calcPepisode.m`, and takes as input arguments a set of events, a channel, and an output directory name in which the pre-processed data will be placed (matrices with zeros or ones for every point in time and every frequency, indicating whether there is a significant oscillation). The function `getuvec_ms` will then return this binary vector for your events of interest and for the specified duration. The variable bgThreshold indicates at what fraction of the mean oscillation amplitude over the whole EEG file the threshold should be set (usually set to 95%). You can take the mean of this binary vector over time in order to determine what fraction of your time interval of interest is taken up by this oscillation.

Z-scoring

Key codes
none

In order to account for across subject and across session differences in signal that are not the result of a condition of interest (e.g. a gradual increase in noise across a session), electrophysiological data must be z-scored on a session-by-session basis. There are several ways to z-score which are discussed both on the wiki (`https://memory.psych.upenn.edu/InternalWiki/EEG_analysis`) and in "Analyzing Neural Time Series Data" (Chapter 18). The method we will use here is to subtract and divide the mean and standard deviation power across all events of interest.

1. Load the events for a single subject

```
[events] = get_sub_events('pyFR','TJ039');
```

2. Filter to a single session

```
sess_events = filterStruct(events,'session == 1');
```

3. Get the voltage for those events

```
[EEG] = gete_ms(30,sess_events ,1600,0,1000, [58 62] , 'stop');
```

4. Get the mean voltage across both events and time

```
MU_EEG = nanmean(nanmean(EEG));
```

5. Subtract the mean voltage from the raw signal and clear the mean value variable (reduces memory usage when multiple electrodes are analyzed simultaneously)

```
zeeg = EEG - MU_EEG;
clear MU_EEG
```

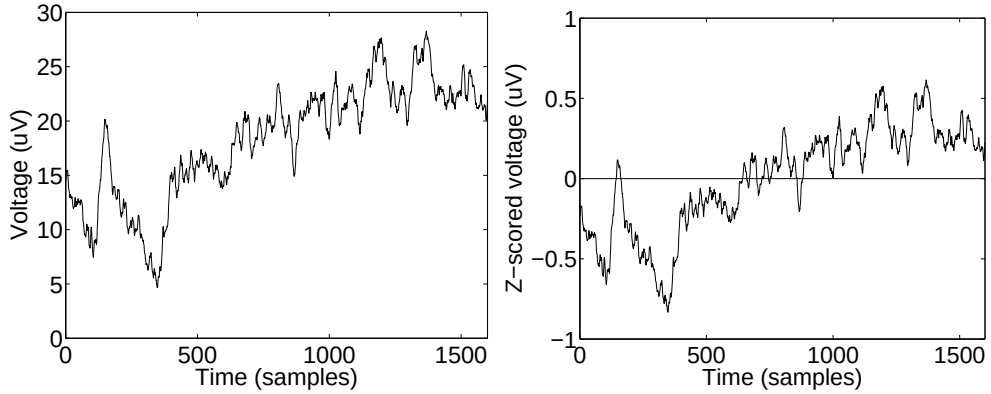
6. Get the standard deviation across events after getting the mean across time

```
STD_EEG = nanstd(nanmean(EEG,2));
```

7. Divide by the standard deviation and clear the standard deviation value variable to reduce memory usage

```
zeeg = zeeg./STD_EEG;
clear STD_EEG
```

8. The z-scored voltage should be centered around 0. The code for these plots is below.



For the raw figure:

```
plot(nanmean(EEG),'black')
set(gca,'xlim',[0 1600]);
set(gca,'xtick',[0 500 1000 1500]);
set(gca,'xticklabel',[0 500 1000 1500]);
set(gca,'Fontsize',25);
label('Voltage (uV)');
xlabel('Time (samples)');
```

For the z-scored figure:

```
plot(nanmean(zeeg),'black');
set(gca,'xlim',[0 1600]);
set(gca,'xtick',[0 500 1000 1500]);
set(gca,'xticklabel',[0 500 1000 1500]);
set(gca,'Fontsize',25);
xlabel('Time (samples)');
ylabel('Z-scored voltage (uV)');
line([0 1600],[0 0],'Color','black');
```

Basic Analysis

Key codes
none

This section will walk through two basic analyses illustrating how to use the events structures in conjunction with EEG toolbox codes to analyze your data. All examples will be the subsequent memory effect (SME) which uses activity at encoding and compares items that will later be recalled to those that will later be forgotten. All examples in this section will be single subject, single electrode examples.

SME ERP Analysis

To carry out an SME analysis on the voltage data, you must 1. Load the desired events, 2. Filter to only encoding events, 3. Get the voltage for all encoding events, 4. Z-Score the voltage, 5. Get a logical index of recall status, 6. Plot the average subsequently recalled and subsequently forgotten voltage traces.

1. Load the desired events

```
[events] = get_sub_events('pyFR','TJ039');
```

2. Filter to just encoding events

```
events = filterStruct(events,'strcmp(type,''WORD'')');
```

3. Extract the voltage

```
[EEG] = gete_ms(30,events ,1600,0,1000, [58 62] , 'stop');
```

4. Z-score the voltage, by session

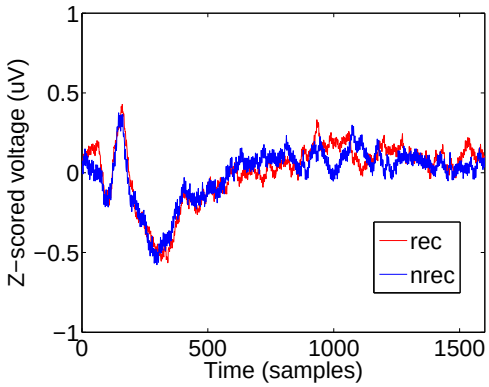
```
zeeg = [];
sessions = unique([events.session]);
for sess = 1:length(sessions)
sess_ev = inStruct(events,['session == ' mat2str(sessions(sess))]);
sess_pat = EEG(sess_ev,:);
MU_EEG = nanmean(nanmean(sess_pat));
zeeg_sess = sess_pat - MU_EEG;
clear MU_EEG
STD_EEG = nanstd(nanmean(sess_pat,2));
zeeg_sess = zeeg_sess./STD_EEG;
clear STD_EEG
zeeg = cat(1,zeeg,zeeg_sess);
end
```

5. Get index of recall status

```
recInd = inStruct(events,'recalled == 1');
```

6. Plot the average recalled and forgotten voltage traces.

```
plot(squeeze(nanmean(zeeg(recInd,:))),'red');hold on;
plot(squeeze(nanmean(zeeg(~recInd,:))));
set(gca,'ylim',[-1 1]);
set(gca,'ytick',[-1:.5:1]);
set(gca,'xlim',[0 1600]);
set(gca,'xtick',[0 500 1000 1500]);
set(gca,'xticklabel',[0 500 1000 1500]);
set(gca,'Fontsize',25);
ylabel('Z-scored voltage (uV)');
xlabel('Time (samples)');
legend({'rec','nrec'});
```



SME Power Analysis

1. Load the events for a single subject and filter to just encoding events. Set the freq parameter.

```
[events] = get_sub_events('pyFR','TJ039');
events = filterStruct(events,'strcmp(type,''WORD'')');
freqs = logspace(log10(2),log10(200),60);
```

2. Loop through each session, extracting power and z-scoring. Z-scoring power is slightly different from voltage z-scoring as there is the added dimension of frequency.

```
zpow = [];
sessions = unique([events.session]);
for sess = 1:length(sessions)
sess_ev = filterStruct(events,['session == ' mat2str(sessions(sess))]);

[~,sess_pow] = getphasepow(30,sess_ev ,1600,0,1000,...
'filtfreq', [58 62] ,'filttype', 'stop','freqs',freqs,'width',6);

MU_POW = nanmean(nanmean(sess_pow,3));
STD_POW = nanstd(nanmean(sess_pow,3));

[n_ev,n_freq,n_time] = size(sess_pow);

pat = [];
for f = 1:n_freq
all_mu = repmat(MU_POW(f),n_ev,n_time);
all_std = repmat(STD_POW(f),n_ev,n_time);
pat(:,f,:) = (squeeze(sess_pow(:,f,:))-all_mu)./all_std;
end

zpow = cat(1,zpow,pat);
end
```

3. Get the recalls index.

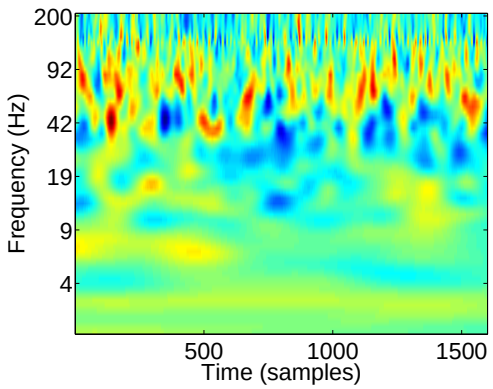
```
recInd = inStruct(events,'recalled == 1');
```

4. Subtract the mean not recalled power from the mean recalled power in order to generate an SME spectrogram.

```
SME_zpow = squeeze(nanmean(zpow(recInd,:,:))) - squeeze(nanmean(zpow(~recInd,:,:)));

imagesc(flipud(SME_zpow));
F = sort(round(freqs),'descend');
set(gca,'ytick',1:10:60);
```

```
set(gca,'yticklabel',F(1:10:60));
set(gca,'xtick',[500 1000 1500]);
set(gca,'xticklabel',[500 1000 1500]);
set(gca,'Fontsize',25);
ylabel('Frequency (Hz)');
xlabel('Time (samples)');
```



Parallel Processing for Group Analysis

Key codes
apply_to_subj
open_rhino_pool
rhino_pool_config

Ultimately, we would like to analyze data across a group of participants, rather than a single participant. We already have laid out all the tools needed to do this in a serial manner: simply generate an SME for every participant and perform across-participant statistics on those values. However, a major benefit of using a cluster such as rhino is the ability to run these analyses in parallel. Here we will describe a method for parallel distributed processing and how to integrate these functions with the EEG analysis functions above. For specifics on cluster usage, the lab wiki should be consulted (`https://memory.psych.upenn.edu/InternalWiki/Cluster`).

Parmgo/Matlab pool

Use of parmgo and matlab distributed pool functionality requires a wrapper function which includes a parfor loop around your main analysis code. It is important to perform as little computation as possible within this loop. Ideally, you will specify a set of parameters and a subject list outside of the loop, then only call the analysis function within the parfor loop, passing it the subject information and parameters.

1. Load the desired subject structure

```
subj = get_subs('pyFR');
```

2. Set some parameters - for EEG analysis this would be anything you would like to pass to gete_ms or getphasepow and anything needed for loading subject specific events.

```
params.exp = 'pyFR';
params.channel = 30;
params.durationMS = 1600;
params.offsetMS = 0;
params.bufferMS = 1000;
params.filtfreq = [58 62];
params.filttype = 'stop';
params.freqs = logspace(log10(2),log10(200),60);
```

3. In matlab, open a pool using `open_rhino_pool`, specifying the desired number of nodes and memory per node. Except for very large datasets (e.g. the PEERS study), you should request no more than 3G of memory.

```
num_nodes = open_rhino_pool(10,'3G');
```

4. Call the parfor loop and thus your main analysis code

```
parfor i=1:length(subj)
try
    pow_ana(subj(i),params);
        catch e
         fprintf('\n*****\nERROR!!!! : %s\n******\n',e.message);
        end
end
```

   - Where the function `pow_ana` runs all of the codes listed above for either the SME ERP or power analysis.
   - You need to include a `try` statement because in a parfor loop, if a single subject crashes, the entire loop will crash. The `catch e` line will print the error to the command window.

5. Be sure to include a line at the end of your wrapper to close the matlab pool, otherwise the nodes will not be returned to the pool for use by others.

```
matlabpool('close')
```

Electrode locations

Key codes
getBipolarSubjElecs
get_mtl_leads

As electrode locations change for each hospital patient, you will need to be able to extract information about which electrodes a participant has and where those electrodes are located. Specific details on how electrodes are localized and how the various fields in the electrode structure are built are not described here, but can be found on the wiki (`https://memory.psych.upenn.edu/InternalWiki/InternalWiki/DataAcquisition/Hospital_Documentation/Electrode_Localization_Pipeline`). Currently, the lab uses a bipolar re-referencing scheme, so all electrodes analyzed from this point forward will be bipolar pairs. Information on why to use bipolar re-referencing can be found in Nunez & Srinivasan, 2006. This tutorial will show you how to get a list of electrodes for a subject and how to know where in the brain those electrodes are located.

Note: Although we are using bipolar re-referencing, the data is originally from a monopolar recording. As of January 14, 2015, in no part of the automatic processing does the data become re-referenced to the bipolar montage, **the user must implement the bipolar re-referencing during analysis**. Raw, non re-referenced data is available in each subject's eeg.noreref directory. EEG data that has been re-referenced *to the weighted-average reference* is available in each subject's eeg.reref directory. The event structures currently point to this weighted-average re-referenced data, e.g.

`eegfile: `/data7/eeg/TJ039/` **eeg.reref** `/TJ039_08May12_1744'`

This weighted-average reference is generated automatically through the code `reref.m`.

1. Load a subjects' electrodes using `getBipolarSubjElecs.m`.

```
[elecs] = getBipolarSubjElecs('TJ039');
```

2. This will return an electrode-length (number of bipolar pairs) structure array with several fields containing location information for that electrode

3. As an example, here is TJ039's first electrode:

```
      subject: 'TJ039'
      channel: [1 2]
      tagName: 'RFA1-RFA2'
      grpName: 'RFA'
            x: 4.5000
            y: 67
            z: -13
         Loc1: 'Right Cerebrum'
         Loc2: 'Frontal Lobe'
         Loc3: 'Superior Frontal Gyrus'
         Loc4: 'Gray Matter'
         Loc5: 'Brodmann area 11'
         Loc6: []
      Montage: 'rsag'
       eNames: '1-2'
        eType: 'S'
  bpDistance: 10.2470
     avgSurf: [1x1 struct]
   indivSurf: [1x1 struct]
```

4. The key fields of interest are listed below.

   - *channel* - these channel numbers correspond to the raw eeg files and will be needed when calling functions like gete_ms and getphasepow
   - *Loc1* - hemisphere
   - *Loc2* - lobe

- *Loc3* - gyrus
- *Loc5* - Brodmann area

5. getBipolarSubjElecs will provide information for all cortical electrodes; however, it will not have clear information about subcortical electrodes as those electrodes (specifically in medial temporal lobe) are localized using a different method (see `https://memory.psych.upenn.edu/InternalWiki/InternalWiki/DataAcquisition/Hospital_Documentation/Electrode_Localization_Pipeline`). To identify which electrodes in the structure are from the hippocampus, you must use the function `get_mtl_leads.m`.

   ```
   [hipp_elecs] = get_mtl_leads('TJ039','hipp');
   ```

6. This will return a vector of electrode numbers which are located in the hippocampus. If you need more information about location (e.g. left/right hemisphere), simply cross-reference this list with the output from getBipolarSubjElecs.

   ```
   hipp_elecs =
             41
             42
             43
             44
             45
   ```

7. **Note:** You'll notice that the output of `get_mtl_leads.m` is a vector of single numbers, not bipolar pairs. To use bipolar pairs across your analysis, grab any bipolar pairs from getBipolarSubjElecs that contain at least one of these numbers.