EEG toolbox documentation

Computational Memory Lab

February 1, 2008

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.

1 Getting started

The EEG toolbox operates on data in a particular directory structure, where the data is saved in separate files for each channel, in a separate directory for each subject. There is a top-level directory often referred to as "basedir", which could be named after your experiment, for example "eeg/mms" (it is important that there is a directory named "eeg" at the very top). In that basedir, there is a directory for every subject, e.g., "MMS002". Every subject has various standard directories, most importantly a directory with events structures (will be explained later on). This directory can be named "events". It is important to place all your data under an "eeg" directory, which can hold files with information relevant to all data analyses such as specifications for wavelet and multitaper parameters.

```
~/eeg/
  eeganalparams.txt
  mms/
    FGS001/
      events/
      behavioral/
      eeg.reref/
      eeg.noreref/
        params.txt
      raw/
    FGS002/
    FGS003/
```

If you are using EGI data, the EEG toolbox contains a function to split the .raw files into single channel files (`egi_split.m`). A more generic function that reads binary files is `spliteeg.m`, otherwise, you can write your own function

along these lines. This will, importantly, create a file `params.txt`, with parameters like the sample rate. Another important file is `eeganalparams.txt` in the `~\eeg\` directory, which contains analysis methods parameters, e.g., wavelet width. Your `eeganalparams.txt` file can look for example like this:

```
freqs (2^(1/8)).^(8:56)
width 6
bandwidth 1
windowsize 0.3
```

In the above file, `freqs` is the vector of frequencies that will be analyzed in the oscillatory analysis scripts, and `width` is the width of the wavelets (for which we generally use 6 (Tallon-Baudry, Bertrand, Delpuech, & Permier, 1997; Caplan, Madsen, Raghavachari, & Kahana, 2001)).

In general if you want to look at a file with EEG data, you can use the function `look.m`, which takes as input argument an EEG filename and a channel, and gets basically the contents of the file as a vector and plots it on the screen.

2 Events

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. You will have to write your own matlab script to extract events from your behavioral logs, which can be modeled on the sample script `createFGSevents.m`. The crucial fields that every events structure should have in order to function in the EEG toolbox are a field containing the events time in the units of your behavioral data file (e.g., milliseconds). You can then select subsets of events using the `filterStruct.m` function, which takes in a logical expression, e.g., to get all CUE events we use

```
cueEvents = filterStruct(events,'strcmp(type,''CUE'')')
```

where 'events' is the name of the events structure we have loaded with

```
events=loadEvents('~/mms/data/FGS001/beh/events.mat')
```

(an alternative way of loading events is `events=loadSubjEvents`, which takes as input arguments a basedir (see above), a cell array with subjects (e.g., subj='FGS001','FGS002','FGS004') and the name of events files (e.g., 'beh/events.mat')). These cueEvents can then be used in subsequent EEG toolbox functions like `erp.m` (see below).

Another very useful function is `getStructField`, which gives you the contents of a particular field in your events structure in an array, e.g.,

```
isCorrect=getStructField(events,'correct')
```

gives you a binary vector that for every event tells you whether a subject got it right. If the field contains strings, the output of this function will be a cell array. Similarly, the `inStruct.m` function will tell you for a certain logical expression, e.g., 'iscorrect==0' whether that is true for every event (so it yields a binary vector with the length of the events structure).

3 Align

Once you have your events structures and EEG split out by channels, you are ready for aligning these events with the EEG data. This will create the fields "eegfile" and "eegoffset" in your events, which are crucial for the rest of the EEG toolbox to work. The exact functions to use for aligning are exemplified in the `prep_egi.m` function, and include `runalign.m` and `logalign.m`. This will basically take the extracted pulses from the EEG data and the pulse sending logs from your behavioral data and use regression to line those up. When you are dealing with iEEG data, you will have to extract the pulses from an analog channel, and this is best done using `alignTool.m`, which has a gui to extract pulses and also to put all the correct files together for the alignment. After you have aligned the data, you probably want to run an eyeblink detection script, which we have called `addArtifacts.m`. This basically looks for large deflections in the fast and slow moving averages of the EEG data. In our EGI data, you probably want to call it as follows:

```
addArtifacts(eventfile,{[26,127],[8,126]},100,0);
```

which basically uses the channels above and below the eye in bipolar to find these deflections.

4 ERP

Plotting ERPs using the EEG toolbox is really simple: use the `erp.m` function. First select an interesting set of events as described above, e.g.,

```
corrEvs=filterStruct(events,'correct')
```

Then plug those into the ERP generating function, e.g.,

```
erp_chan10=erp(10,corrEvs,600,-100,500,[-100 0],[58 62],'stop',1,5)
```

A couple of notes are in order: the durations are specified in samples, so in the case of a samplerate of 500 Hz we often want to look at 1 second, with a 200-ms baseline (100 samples). The next variable is the buffer, for which we usually want a second, which is necessary to avoid filtering or convolution artifacts. The variable `[-100 0]` indicates that we use the 100 ms before the event as a baseline. Then really important is filtering out the line noise, which is why we use `[58 62]`, of a 'stop' (bandpass) filter, with one filtering iteration. The last parameter is the kurtosis threshold, which we use to throw out artifacts (Delorme

& Makeig, 2004). 5 is usually a pretty good threshold for scalp and iEEG in our experience. You can use the function `ga_erp.m` to compute a grand average ERP over subjects, which also gives you errorbars, and can compute the ERP both treating subjects independently as well as treating all data as one subject (see the help for more info).

In order to determine the significance of differences in ERP amplitude, you can use the `bootstrap.m` function. This function takes in as input arguments the number of bootstrap iterations (usually we use 1000), then either one or two vectors/matrices of data. The data can for example be the difference in ERP amplitude between condition 1 and condition 2 for every subject or every event at every point in time. The bootstrap function will create a distribution of results of your statistic (a one-sample t-test or signed rank test for one-sample data, or a two-sample t-test or ranksum test for two-sample data) for your data (in the one-sample case, you sign of the data is randomly swapped, in the two-sample case, the conditions of your data are swapped). You can then examine how far your observed test statistic falls outside the randomized distribution (the observed test statistic is the second output argument of the `bootstrap.m` function).

5 Power (wavelet and multitaper)

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 also phases). The most useful function here is probably `getphasepow.m`, which gives you basically that data. A lower level function, if you do not have events, but instead plain EEG, is `multienergyvec`. An example of calling `getphasepow.m` is:

```
[phase,pow] = getphasepow(10,corrEvs,1000,0,1000,'filtfreq',[58
62],'filttype','stop','kthresh',5)
```

which uses very similar parameters to the erp function, but now takes in durations in milliseconds. Here you do not do the baseline correction, so there is no variable for that (and probably most of the time you want your offset to be 0). Plotting a simple power spectrogram can be done by taking the mean over events (where you probably want to take a log10 of the power in order to be able to see anything, to counteract the 1/f falloff of the power spectrum). An example would be

```
imagesc(squeeze(mean(log10(pow))))
```

In order to make power comparable across frequencies and subjects, it is often useful to look at z-transformed power. The function that does this is `ztrans_pow.m`. The input arguments are very similar to `getphasepow.m`, except that they are inserted in a structure. Different is optional info you can give the function about pieces of data you'd like to give it to compute baseline power (e.g., times when the subject is staring at a fixation cross). If you do not give

separate baseline info, then the baseline is the mean and standard deviations over all events you give it as input.

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 (see Raghavachari et al. (2001); van Vugt, Sederberg, and Kahana (2007) for more info).

In order to asses significance of power results, we use mainly two randomization methods (you can also use the False Discovery rate procedure (Benjamini & Hochberg, 1995), with the function `fdr.m`). The first randomization method is called the "sum-z method" (Gibbons & Shanken, 1987; Sederberg et al., 2007b), where we compute a bootstrapped test statistic for every electrode, frequency and timepoint/timebin within every subject. We then combine across subjects by adding up the `norminv.m` of every p-value for the actual statistics as well as the bootstrapped distributions, and check where the actual statistic falls in that distribution, which is the final p-value across subjects for that frequency, timebin and electrode.

6 Pepisode

Pepisode is a power analysis developed by Jeremy Caplan in our lab (Caplan et al., 2001), 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.

7 Plotting

A useful function for plotting is `publishfig.m`, which makes the figure such that it has larger axis labels etc, which is useful for publishing purposes. Note

that it removes the title, so apply the title after running this function.

For plotting the topography of effects for scalp EEG, we use EEGLAB (Delorme & Makeig, 2004), `http://XXXXX`. The function that we use that is `headplot.m`, which requires as input a specification of where the electrodes are located (for our EGI system that is a file named 'GSN129.sfp') and an array with a value to be plotted for every electrode. For plotting iEEG data, we use `tal3d.m`, for which more documentation is available in the lab wiki (`tal3d.m` makes use of the electrode database stored in `/data/eeg/electrodesXXXX.mat`, which contains a record for every electrode, which has been generated using the talairach daemon).

There are also other plotting scripts in the `talk` directory of the EEG toolbox, which are older and not as accurate in general.

8 Simulation

If you're interested in testing new EEG analysis tools, you can generate fake EEG of a specified number of events, duration, samplerate and amplitude using `getfakeEEG.m`, to which you can add sinusoidal signals of specified amplitude, phase, duration and time using `addSignal.m` or `addSignalProp.m` (where the second version adds a signal with an amplitude that is a proportion of the background EEG and the first one specifies an absolute amplitude). These scripts were used in (van Vugt et al., 2007).

9 Running scripts on the cluster

To run Matlab scripts on the cluster, there are a few useful eeg toolbox functions. Basically you need to write a script that loops over some process that is independent of others, such as doing a computation for every electrode (looping over them) and storing the results of each in a separate file. You would probably use code similar to this:

```
for c = 1:128
 chanFile = sprintf('~/mms/data/datFiles/myAnalysis%.03d.mat',c);
 if ~exist(chanFile)
    lockFile(chanFile);
    << do computation>>
    << save data in chanFile >>
    releaseFile(chanFile);
 end
end
```

References

Benjamini, Y., & Hochberg, Y. (1995). Controlling the False Discovery Rate: a practical and powerful approach to multiple testing. *Journal of Royal Statistical Society, Series B*, *57*, 289-300.

Caplan, J. B., Madsen, J. R., Raghavachari, S., & Kahana, M. J. (2001). Distinct patterns of brain oscillations underlie two basic parameters of human maze learning. *Journal of Neurophysiology*, *86*, 368–380.

Delorme, A., & Makeig, S. (2004). EEGLAB: an open source toolbox for analysis of single-trial EEG dynamics. *Journal of Neuroscience Methods*, *134*, 9-21.

Gibbons, M. R., & Shanken, J. (1987). Subperiod aggregation and the power of multivariate tests of portfolio efficiency. *Journal of Financial Economics*, *19*, 389-394.

Raghavachari, S., Rizzuto, D. S., Caplan, J. B., Kirschen, M. P., Bourgeois, B., Madsen, J. R., et al. (2001). Gating of human theta oscillations by a working memory task. *Journal of Neuroscience*, *21*, 3175-3183.

Sederberg, P. B., Schulze-Bonhage, A., Madsen, J. R., Bromfield, E. B., Litt, B., Brandt, A., et al. (2007b). Gamma oscillations distinguish true from false memories. *Psychological Science*, *18*(11), 927–932.

Tallon-Baudry, C., Bertrand, O., Delpuech, C., & Permier, J. (1997). Oscillatory gamma-band (30-70 hz) activity induced by a visual search task in humans. *Journal of Neuroscience*, *17*(2), 722–34.

van Vugt, M. K., Sederberg, P. B., & Kahana, M. J. (2007). Comparison of spectral analysis methods for characterizing brain oscillations. *Journal of Neuroscience Methods*, *162*(1-2), 49–63.