Tutorical:

Data Analysis for Multi-channel EEG Recordings during A Sustained-attention Driving Task

I. Overview

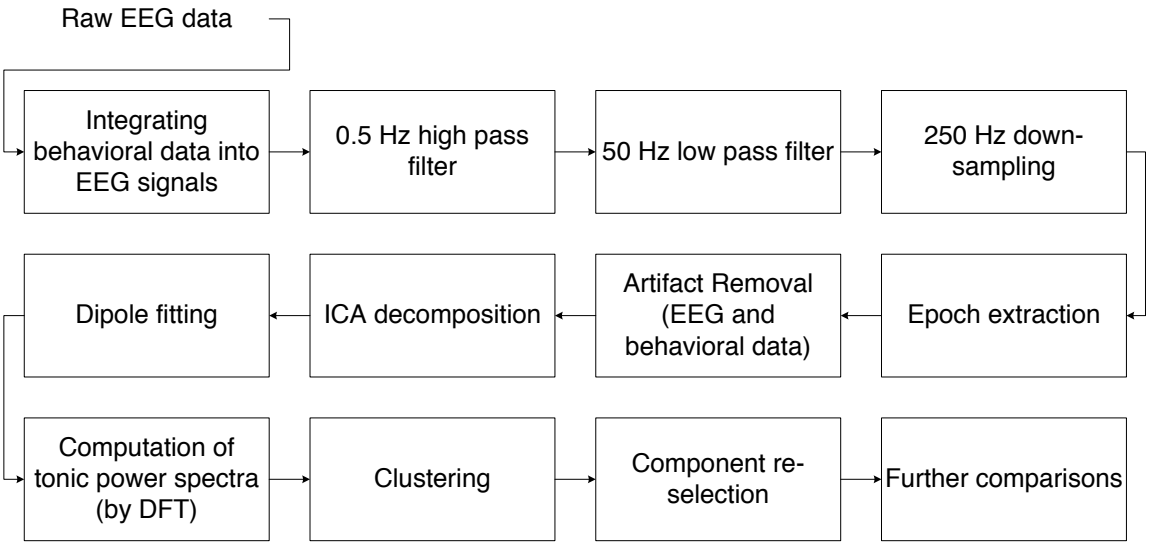
I.1. Software, toolbox, and environment:

- MATLAB R2007a
- EEGLAB v. 5.03
- Running under workstation (Linux): BRC server 140.113.34.3

I.2 Pre-requirements

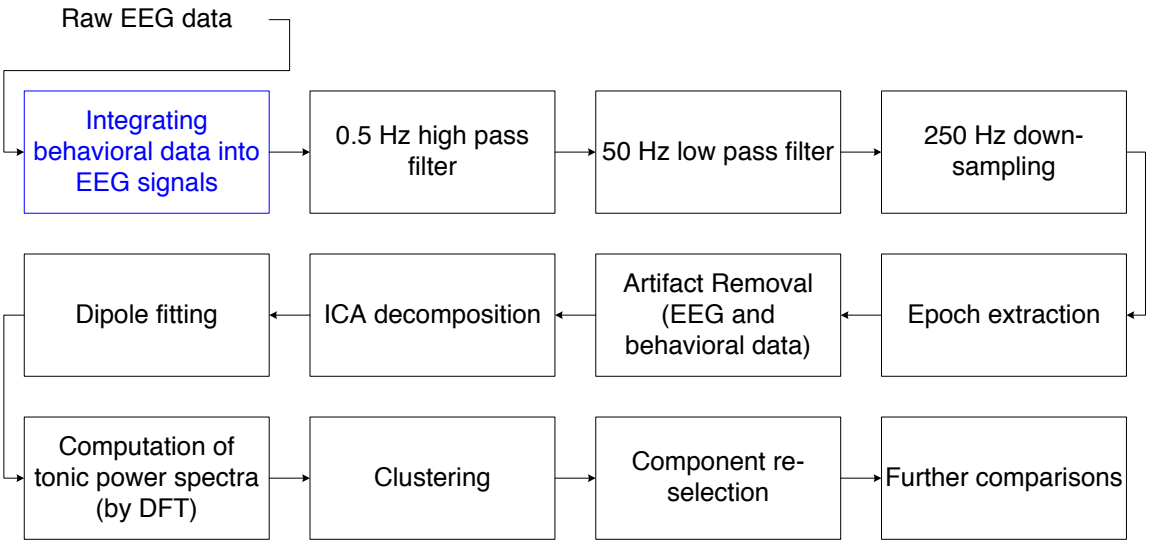
- Critical reasoning and thinking
- Being curious and skeptical about the (discovered) results
- Basic MATLAB script coding ability
- EEGLAB usage
- English communication skill
- C language programming (suggestion)

I.3 Flowchart

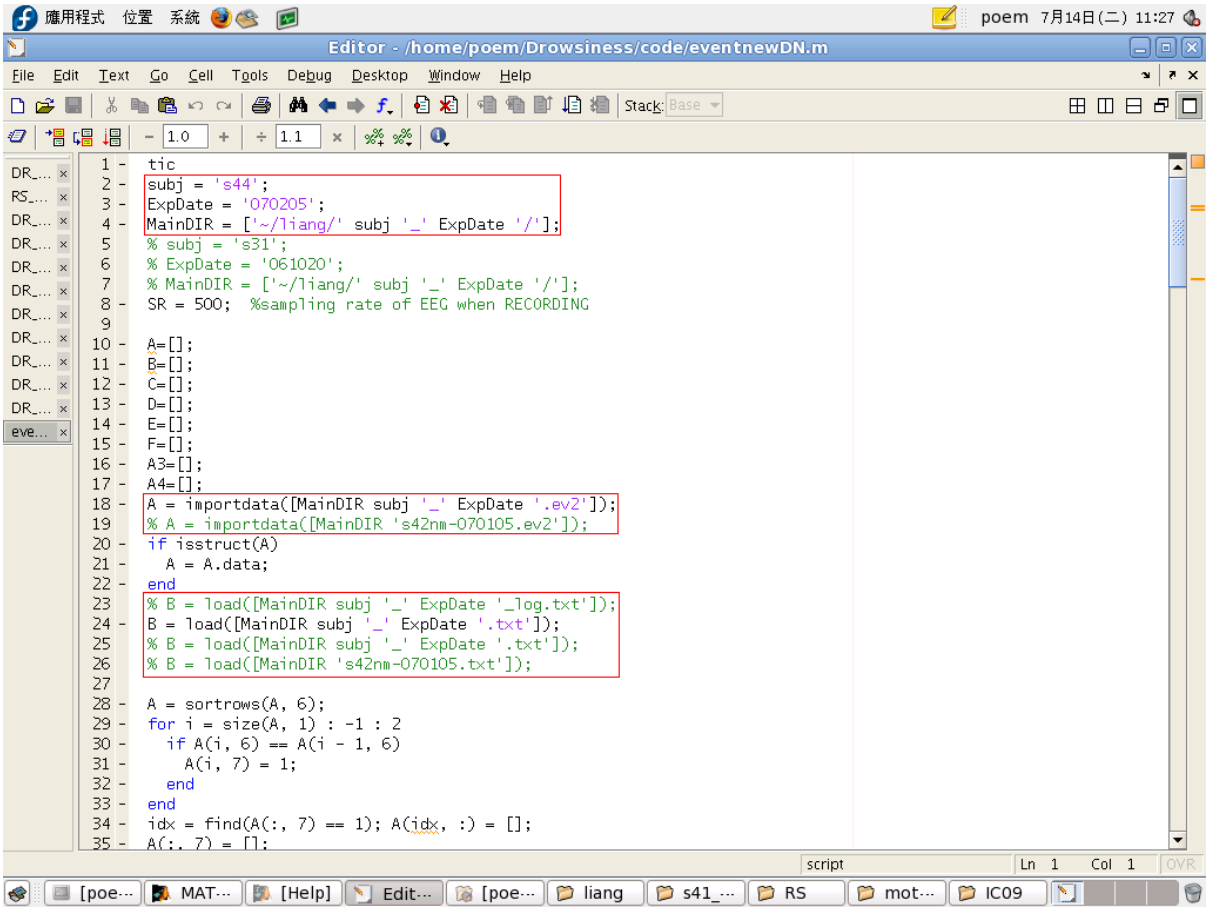


- Do NOT change the sequence of these procedures unless without previous notification.
  - If the procedure can be done later, it will be specified in this documentation.

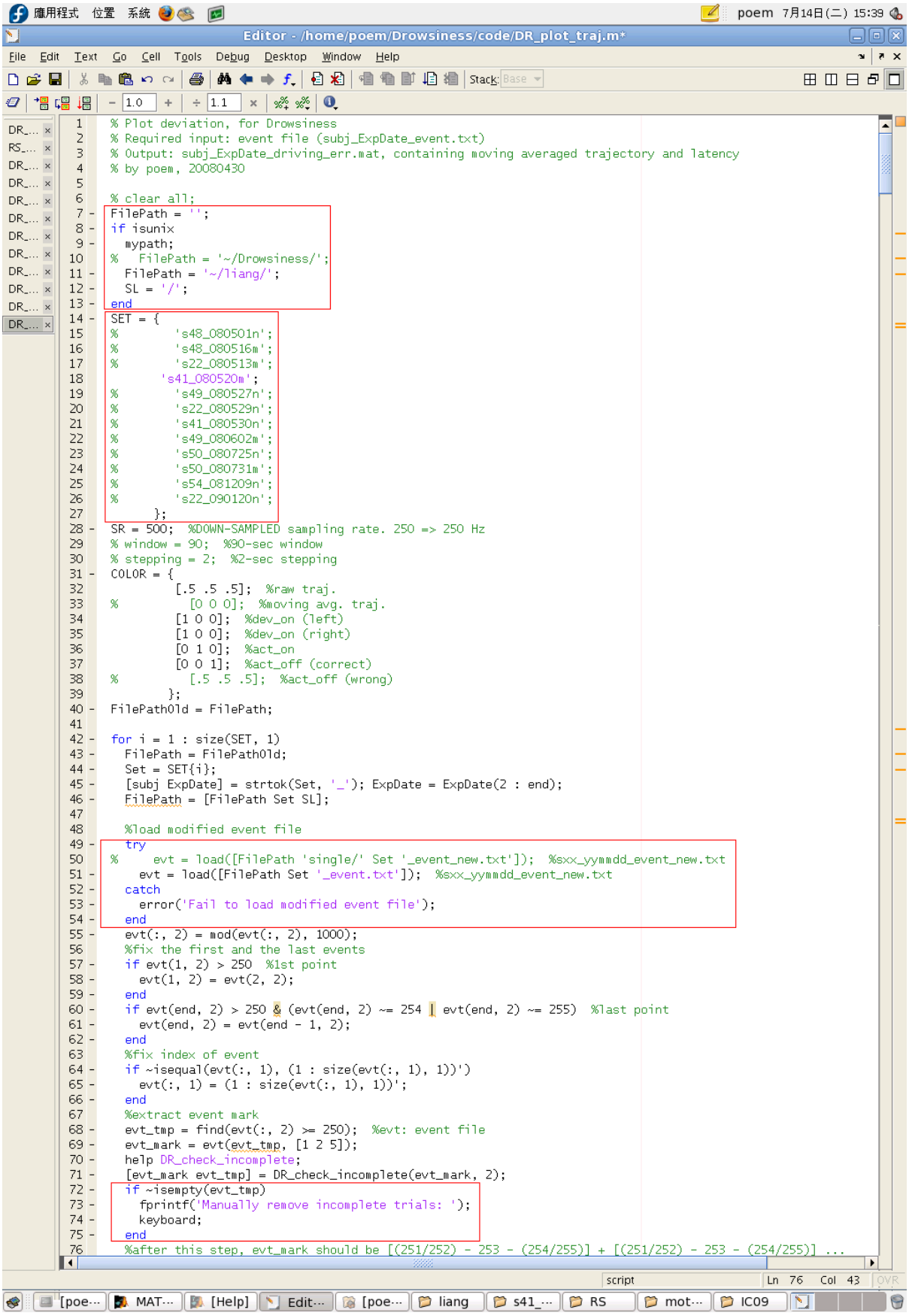
II. Integrating behavioral data into EEG signals



- MATLAB scripts
  - `eventnewDN.m`
  - `DR_plot_traj.m`
- Procedures
  - Integrate “log” file (from stimulus computer) and “ev2” (from Neuroscan) file ➔ “event” file (`sxx_yymmdd(MN)_event.txt`)
    - Use MATLAB script `eventnewDN.m` to integrate “log” file and “ev2” file



- Variables: checking before running
  - `subj`: subject code
  - `ExpDate`: date of experiment
  - `MainDIR`: directory of datasets
  - `A`: contents of ev2 file
  - `B`: contents of log file
- Fix response offset (by Darkflame 昂穎 or RichYe 人慈)
- Use MATLAB script `DR_plot_traj.m` to check if some events miss in the event file
  - Note: this code need `DR_check_incomplete.m` to check if incomplete trials (trials missing some of the events) exist



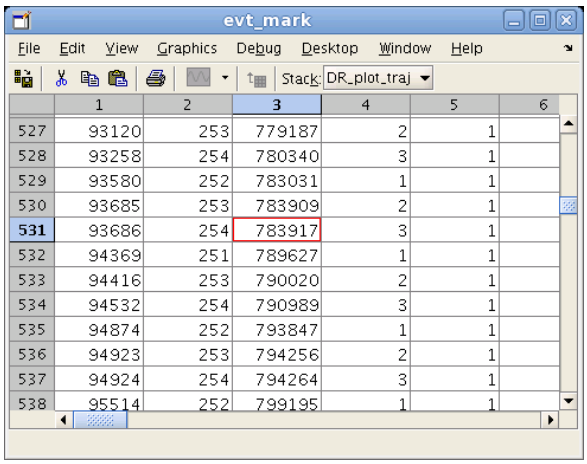
- Variables: check before running
  - `FilePath`: directory of datasets
  - `SET`: trajectories of datasets to be plot (multiple datasets allowed)

- `evt`: name of event file (notice the format)

◆ If the program does not stop in line 74 shown in the above figure (no missing events) ➔ begin pre-processing

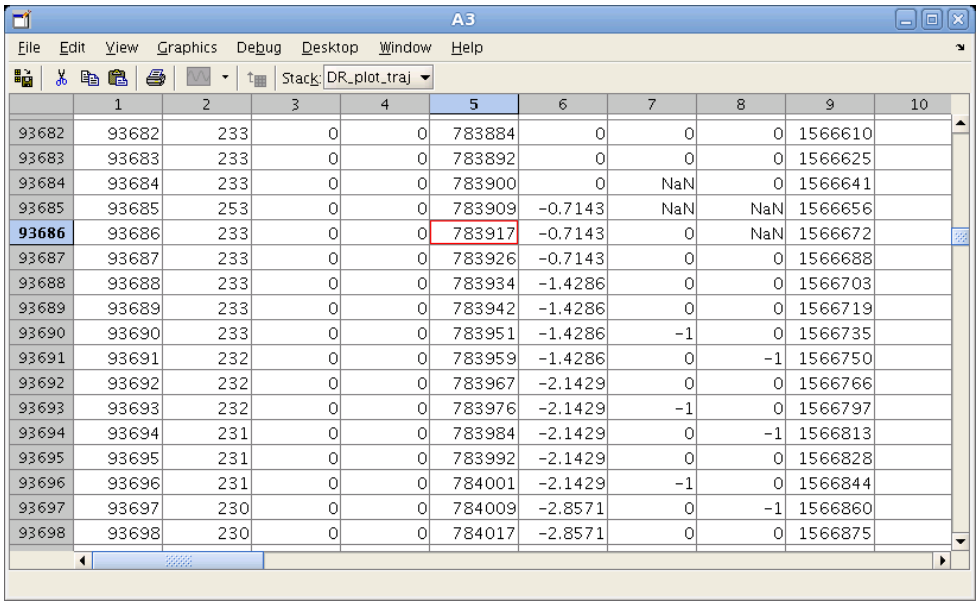
Else,

- Write down the values in the variable `evt_tmp` (see MATLAB workspace). These values are the indices of rows in the variable `evt_mark` with missing events



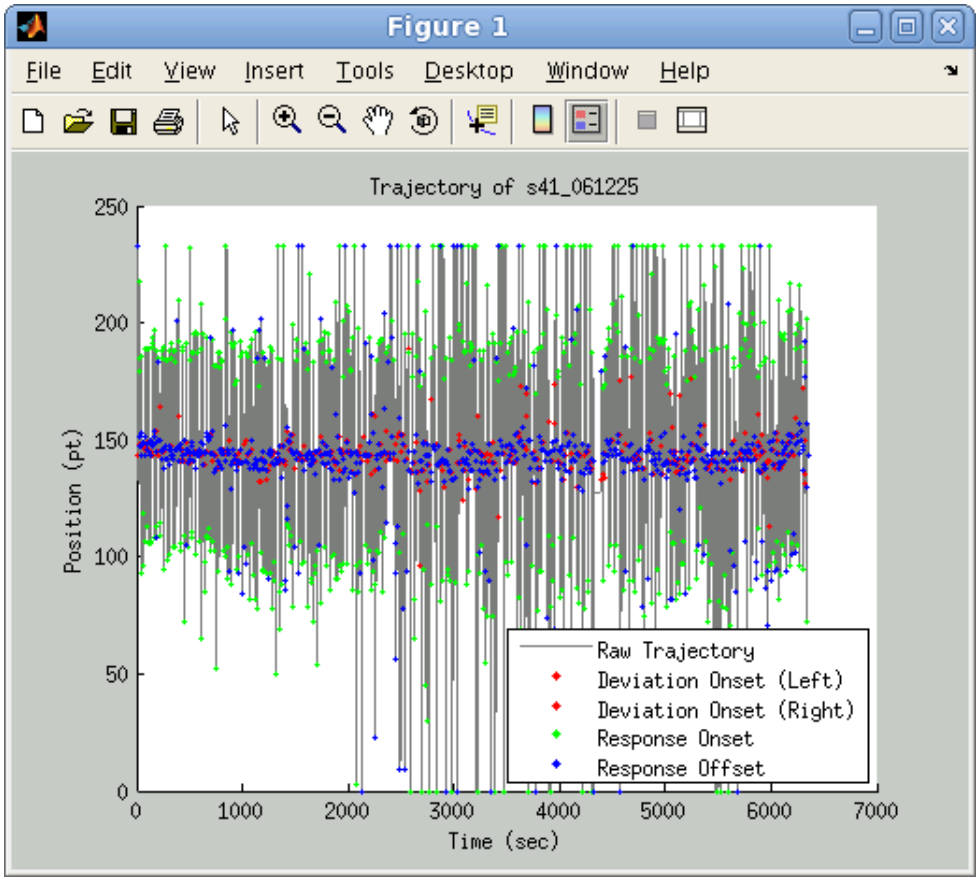
	1	2	3	4	5	6
527	93120	253	779187	2	1	
528	93258	254	780340	3	1	
529	93580	252	783031	1	1	
530	93685	253	783909	2	1	
531	93686	254	783917	3	1	
532	94369	251	789627	1	1	
533	94416	253	790020	2	1	
534	94532	254	790989	3	1	
535	94874	252	793847	1	1	
536	94923	253	794256	2	1	
537	94924	254	794264	3	1	
538	95514	252	799195	1	1	

- Run the remaining program
- Check the rows in `evt_mark` around the indices (in `evt_tmp`). The 3rd column is the latencies in the produced event file (the same as the 5th column in the variable `A3`)



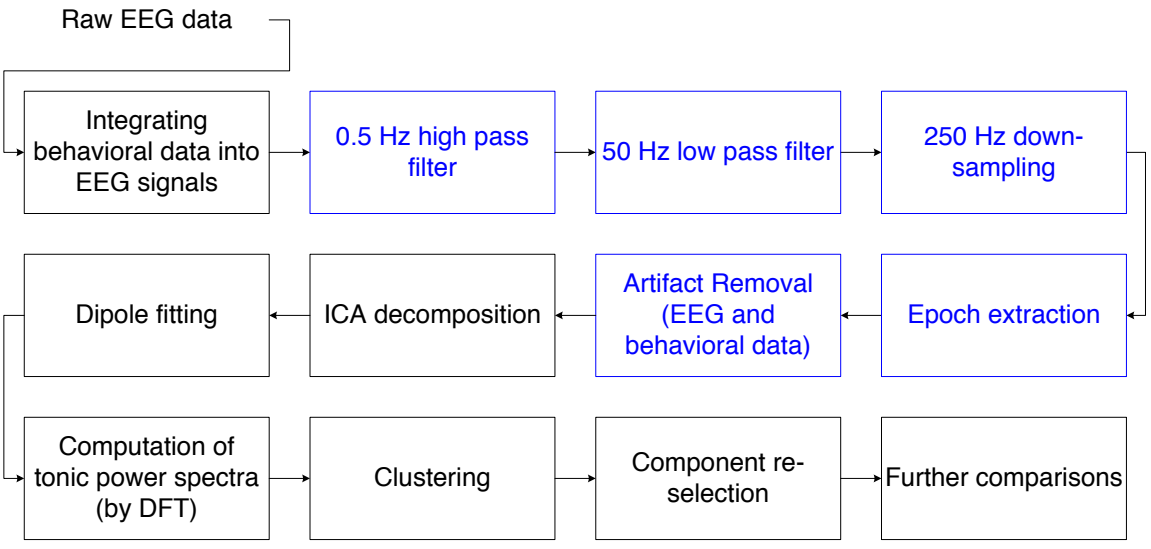
	1	2	3	4	5	6	7	8	9	10
93682	93682	233	0	0	783884	0	0	0	1566610	
93683	93683	233	0	0	783892	0	0	0	1566625	
93684	93684	233	0	0	783900	0	NaN	0	1566641	
93685	93685	253	0	0	783909	-0.7143	NaN	NaN	1566656	
93686	93686	233	0	0	783917	-0.7143	0	NaN	1566672	
93687	93687	233	0	0	783926	-0.7143	0	0	1566688	
93688	93688	233	0	0	783934	-1.4286	0	0	1566703	
93689	93689	233	0	0	783942	-1.4286	0	0	1566719	
93690	93690	233	0	0	783951	-1.4286	-1	0	1566735	
93691	93691	232	0	0	783959	-1.4286	0	-1	1566750	
93692	93692	232	0	0	783967	-2.1429	0	0	1566766	
93693	93693	232	0	0	783976	-2.1429	-1	0	1566797	
93694	93694	231	0	0	783984	-2.1429	0	-1	1566813	
93695	93695	231	0	0	783992	-2.1429	0	0	1566828	
93696	93696	231	0	0	784001	-2.1429	-1	0	1566844	
93697	93697	230	0	0	784009	-2.8571	0	-1	1566860	
93698	93698	230	0	0	784017	-2.8571	0	0	1566875	

- Fix the 2nd column (recover the missing events) in `A3` (the position of the car and behavioral events) in the designated row. Use the trajectory plot for help.



- Begin pre-processing

III. Pre-processing



- Reduce the amount of data
  - Take notes for preprocessing (Preprocessing log) till Part IV ends
    - Basic contents:
      - Original latency of datasets
      - # of trials before and after artifacts removal

- Components related to brain processes (with cluster ID)

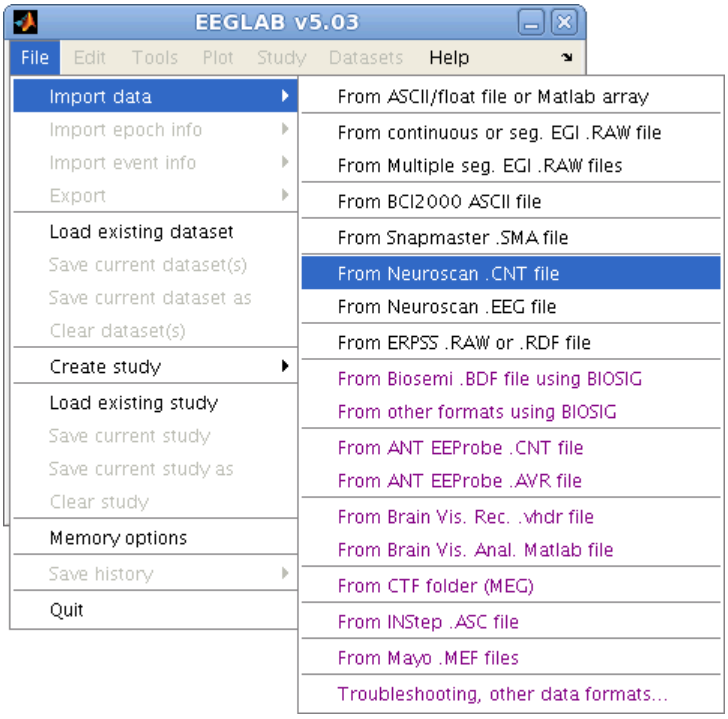
- Use EEGLAB toolbox (till Part IV ends)
  - Set the path in MATLAB:

    ```
    >> addpath(genpath('/home/eeglab5.03/'));
    ```

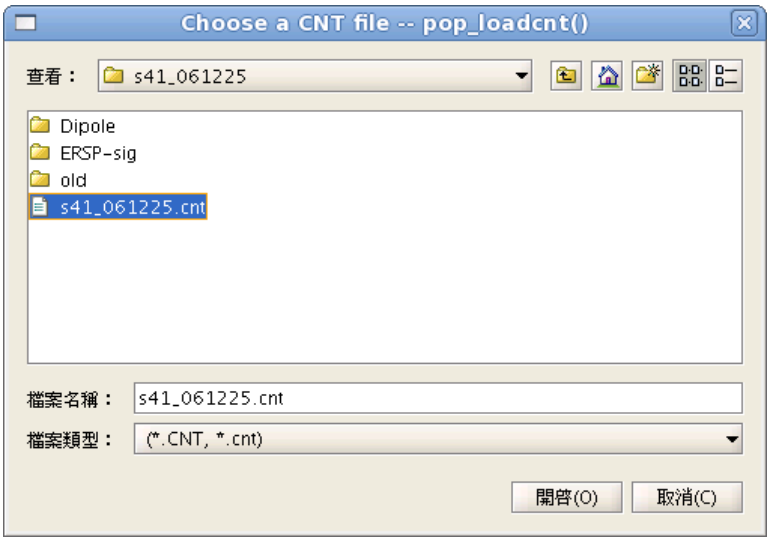
  - `>> eeglab`

III.1 Loading raw EEG data

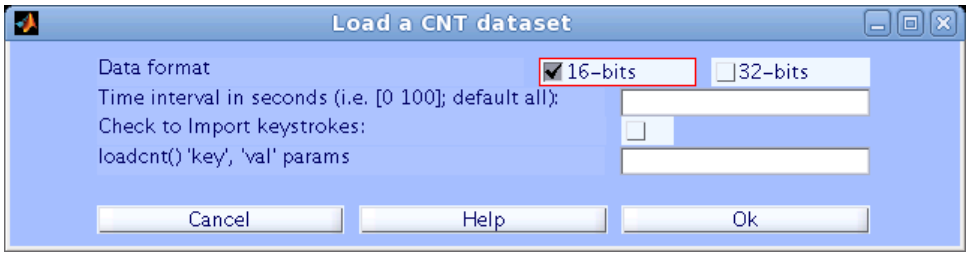
- MATLAB function: `pop_loadcnt()` / `DR_pop_loadcnt()`
  - `DR_pop_loadcnt()`: with the same function, but doesn't read event file (actually ev2 file) when loading EEG data; use with `DR_loadcnt()`
  - EEGLAB GUI interface:
    - File → Import data → From Neuroscan .CNT file



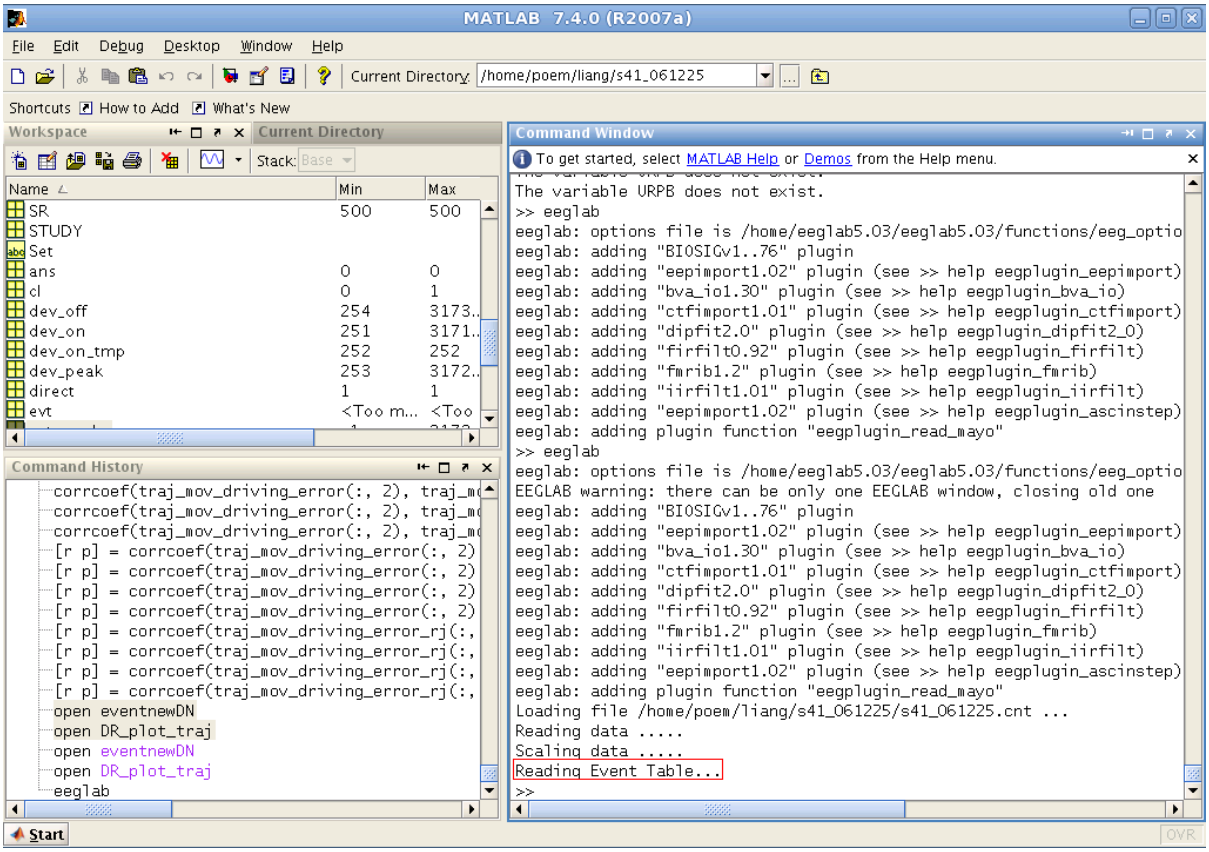
    - Switch to the desired directory and choose the CNT file (raw EEG data)



- Choose the proper resolution of raw data (currently 16-bit)

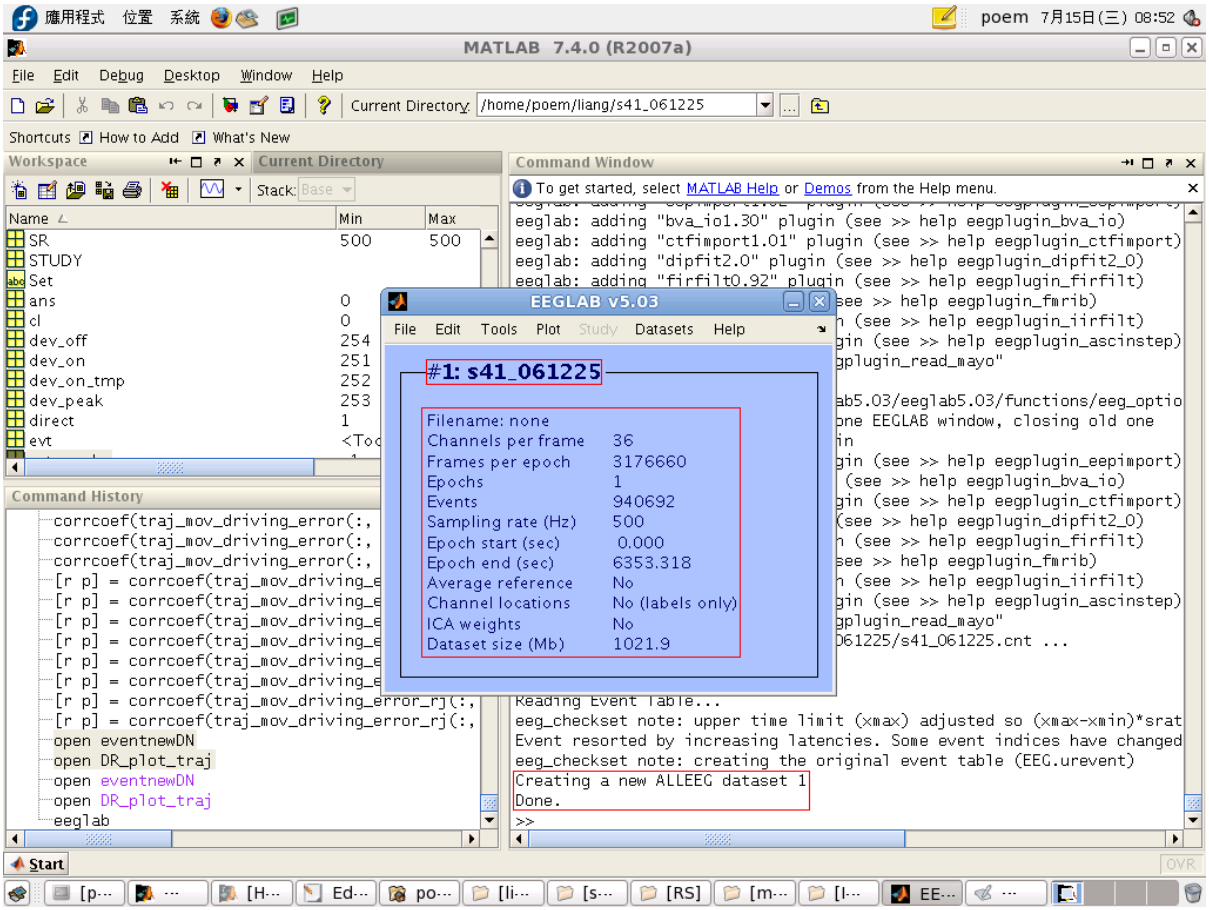


- Wait for finishing loading data…
- It takes to much time in reading the original events in raw EEG!!



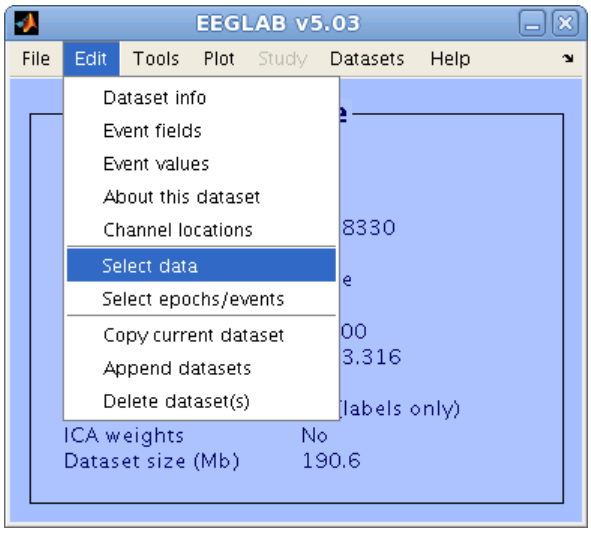
- Edit name to identify datasets (there may be more than one dataset in an EEGLAB window)

- Finally…

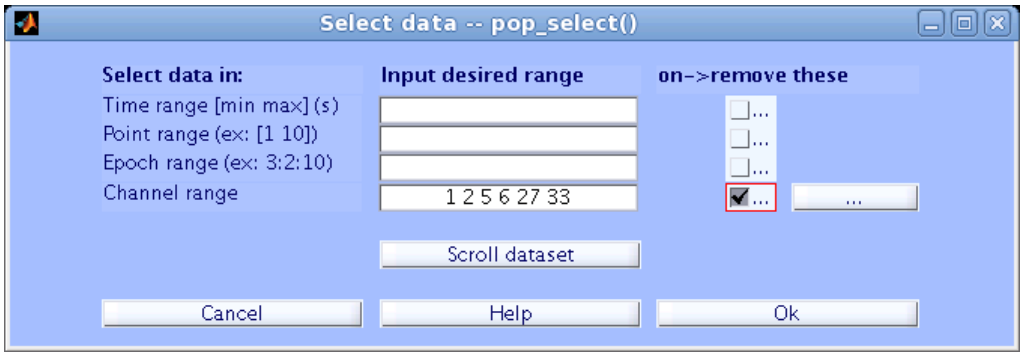


- In command line: show the index of dataset show
- In EEGLAB GUI interface: show the index, name, and other information of the raw data

- If using EEGLAB GUI interface or `loadcnt()` → `>> EEG.event = [];`
- Remove unnecessary channels in EEG data
  - Currently (30-channel data) ➔ remove channels with labels EKG1, EKG2, VEOU, VEOL, A1, and A2(channel indices: 1, 2, 5, 6, 27, and 33)
  - MATLAB function: `pop_select()`
  - EEGLAB GUI interface: Edit → Select data

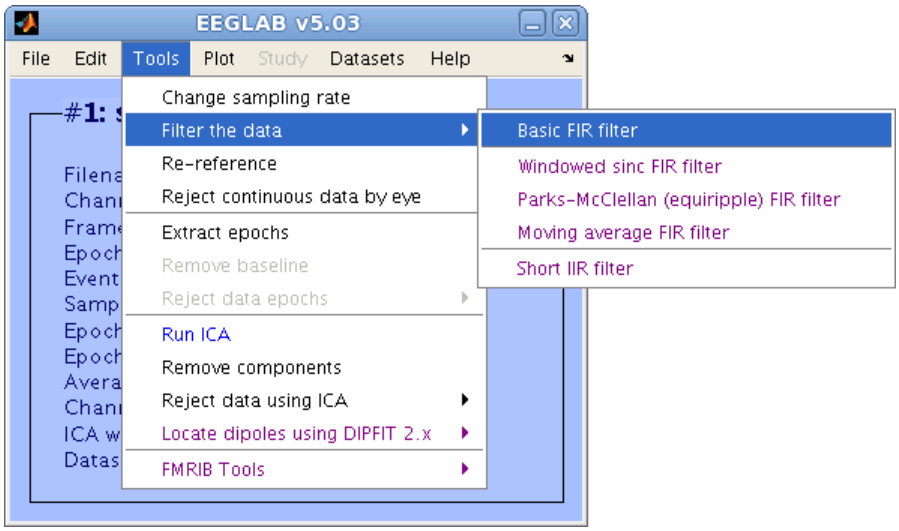


- Type indices of channels to be REJECTED in the “Channel range” editbox, and CHECK the checkbox; otherwise, type indices of channels to be KEPT, and UNCHECK the checkbox

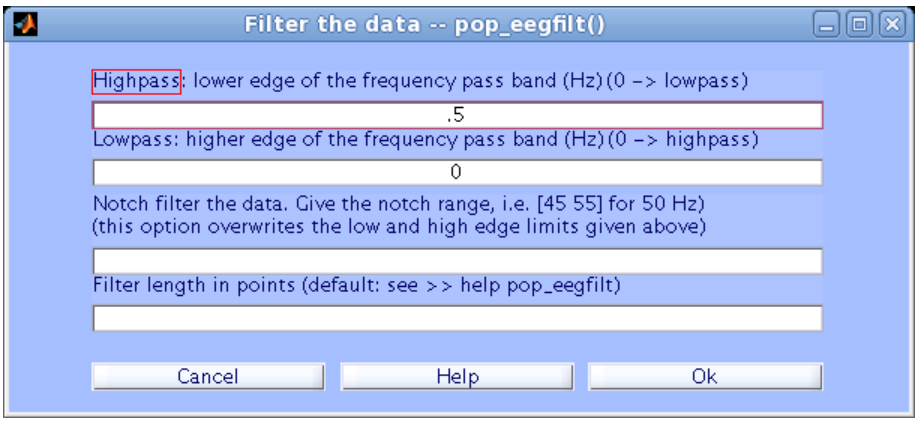


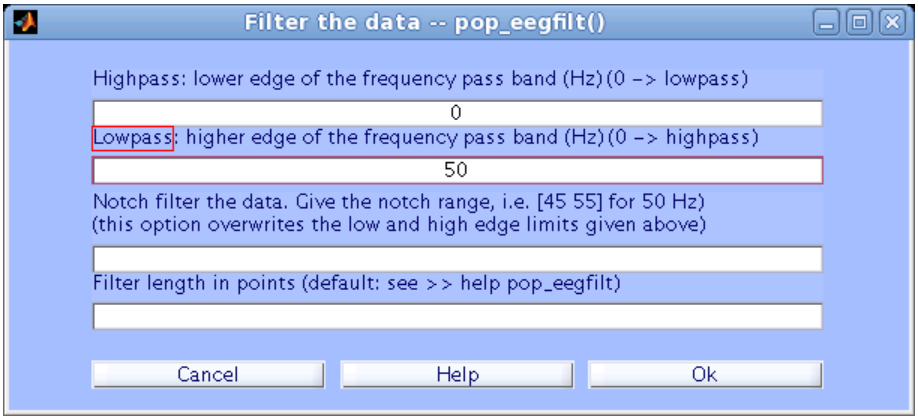
III.2 Filtering and down-sampling

- Filtering: 0.5-Hz high-pass filter (HPF) and 50-Hz low-pass filter (LPF)
  - MATLAB function: `pop_eegfilt()`
  - EEGLAB GUI interface: Tools → Filter the data → Basic FIR filter



  - Apply HPF and LPF SEPERATELY
    - Input the pass band (0.5) of the HPF
    - After applying the HPF, input the pass band (50) of the LPF

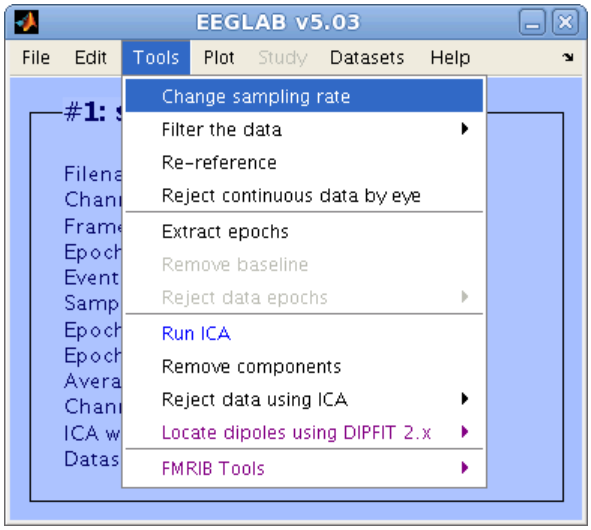


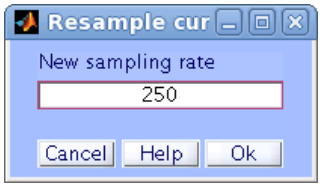


- Overwrite the dataset after applying each filter

If using the command-line mode, use `pop_newset()`

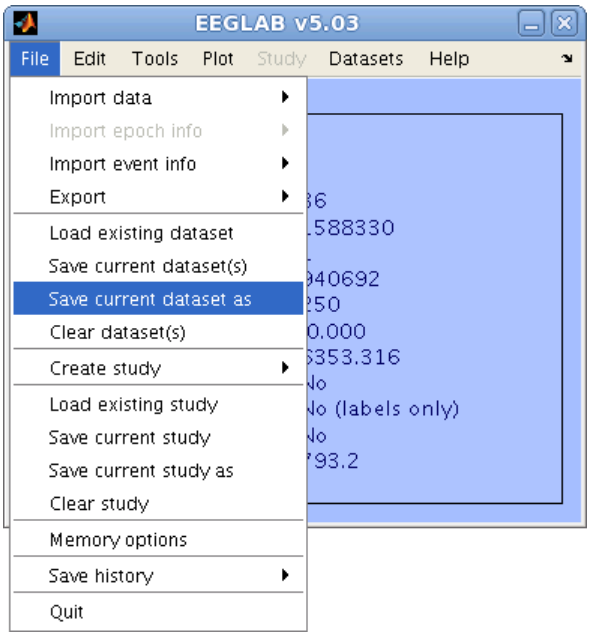
- Down-sampling: down-sample the sampling rate to 250 Hz
  - MATLAB function: `pop_resample()`
  - EEGLAB GUI interface:
    - Tools → Change sampling rate

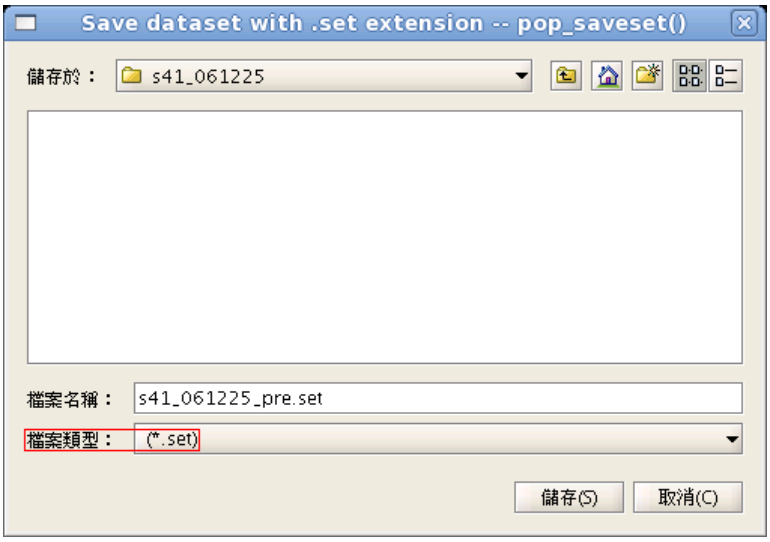




  - Overwrite the dataset in memory
- After this step, save the data as "`sxx_yymmdd_pre.set`"

- MATLAB function: `pop_saveset()`
- EEGLAB GUI interface:
  - File → Save current dataset as



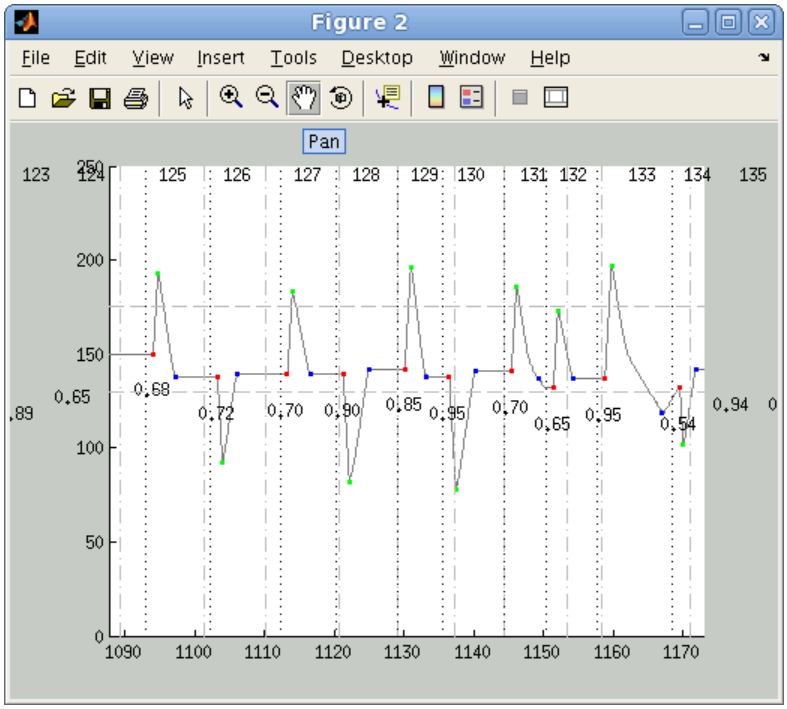


III.3 Integrating behavioral information into EEG data

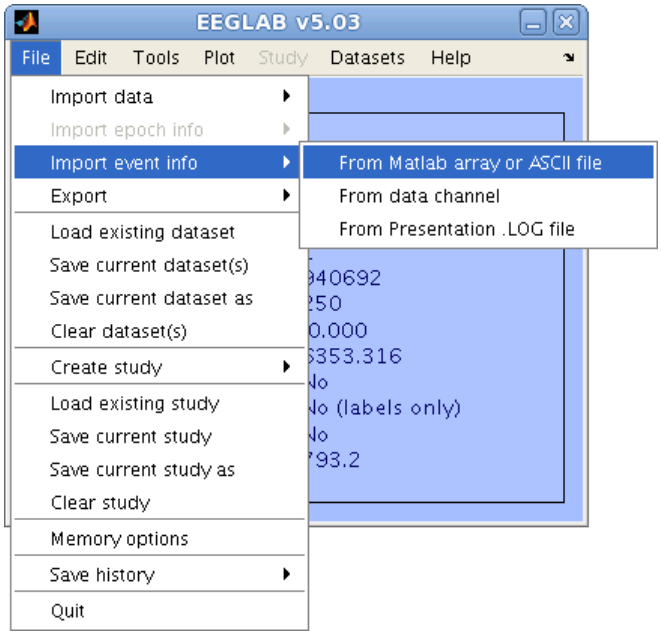
- Currently, no single MATLAB code (doesn't matter)
- Procedures:
  - Load event file: `sxx_yymmdd_event.txt`
  - Extract indices of event marks (1st column in event file), event marks and trajectory (2nd column), and the latency (5th column) from the imported event file into the array `evt`
    - I. e., columns of `evt`: 1st → indices, 2nd → event marks and trajectory, 3rd → latency
  - Extract event marks (deviation onset, response onset, and response offset)

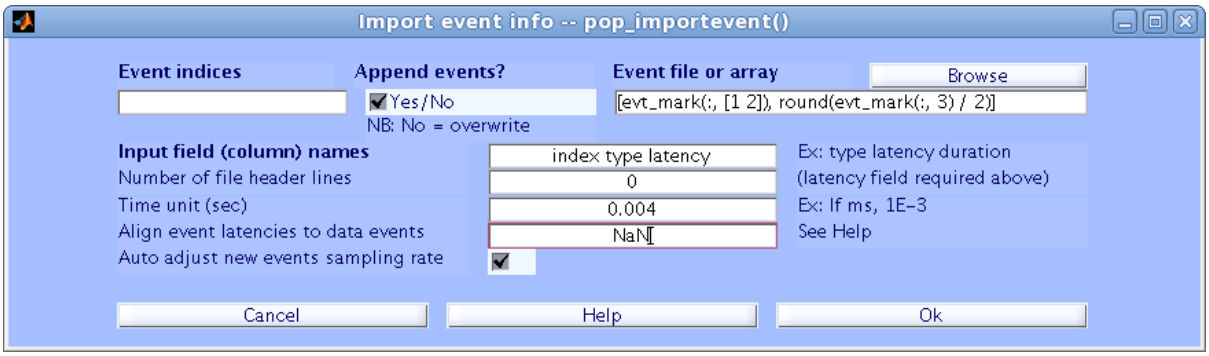
and the corresponding latencies from `evt` into the array `evt_mark`

- Save the latencies of events into `epoch_inf`. Columns:
  1: index of trial
  2~4: latencies of dev_on, act_on, and act_off (the PREVIOUS epoch)
  5~7: latencies of dev_on, act_on, and act_off (THIS epoch)
  8~10: latencies of dev_on, act_on, and act_off (the NEXT epoch)
  11: kept (1) or rejected (0); set this column to 1
  Note: latencies: the values in `evt`
- Save the RTs into `RT_original`. Columns:
  1: RT of this trial (sec)
  2: latency of the dev_on in this trial
- Modify the values in the 2nd column in the array `evt` (change the event marks into trajectories)
- Plot the trajectory of the car (with the event marks, indices of trials, and reaction time of each trial. Save the file into `sxx_yymmdd_raw_traj.fig`



- Integrate the event marks into EEG data
  - MATLAB function: `pop_importevent()`
  - EEGLAB GUI interface:
    - File → Import event info → From MATLAB array or ASCII file



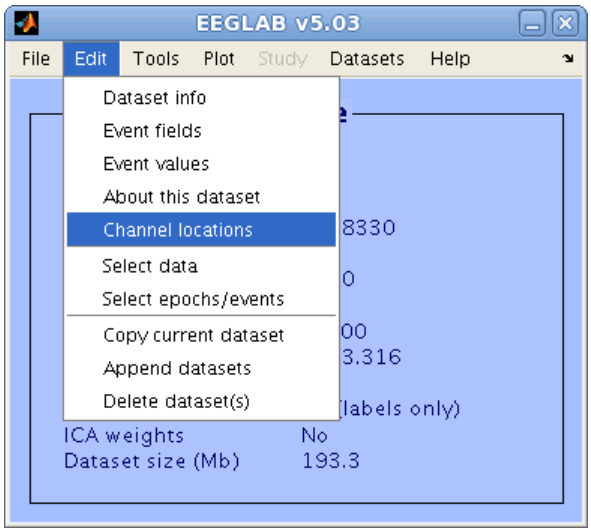


- Append events: (checked)
- Event file or array: `[evt_mark(:, [1 2]), round(evt_mark(:, 3) / 2)]`
- Input field (column) names: `index type latency`
- Number of file header lines: `0`
- Time unit (sec): `0.004`
- Align event latencies to data events: `NaN`
- Auto adjust new events sampling rate: (checked)
- Note that the `latency` column (`round(evt_mark(:, 3) / 2)`) means 0.004 sec (sampling rate: 250 Hz); if not, time unit cannot be set into 0.004

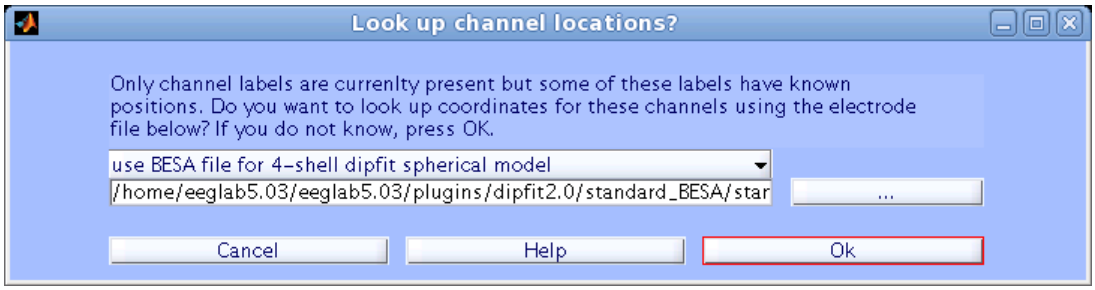
III.4 Importing channel locations

- This step can be done between removing unnecessary channels and extract epoch
- Procedures:
  - MATLAB function: `pop_chanedit()`
  - EEGLAB GUI interface:

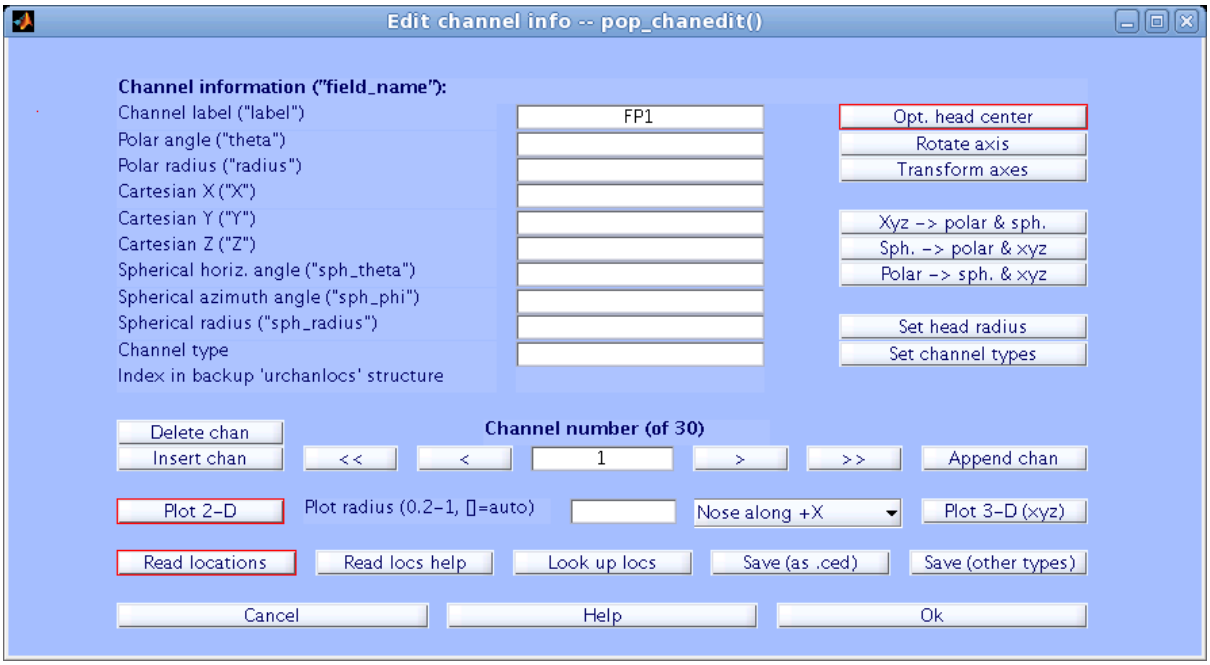
- Edit → Channel locations



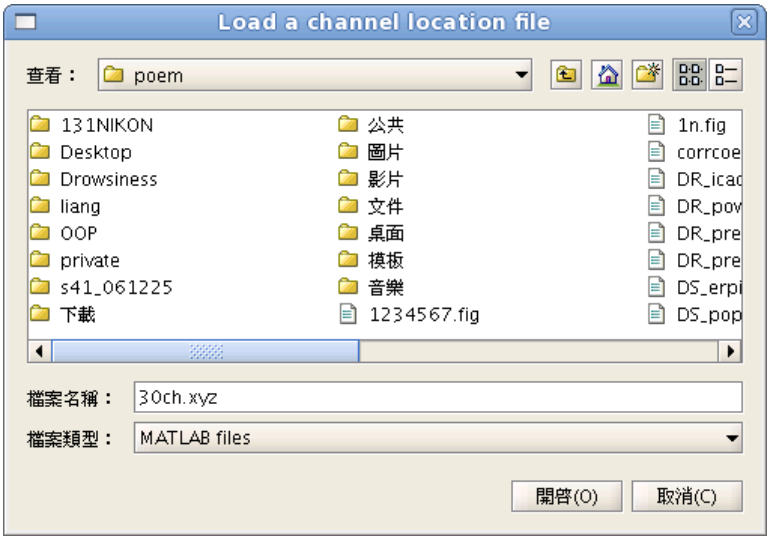
- Look up channel locations: use default settings (press “Ok”)



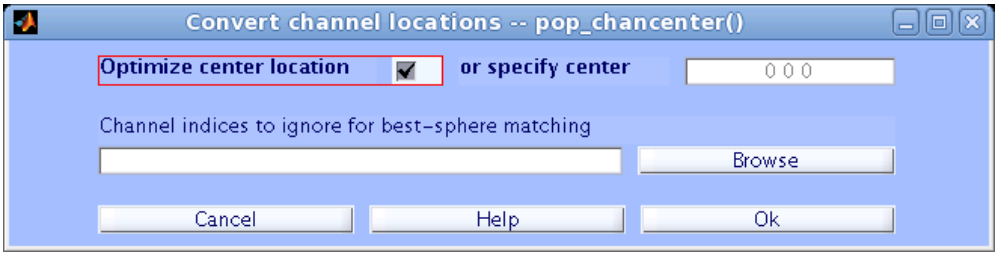
- Import channel locations (use “Read locations”)



- Choose the proper channel location file
  - Experiment done before 2008 or without digitized locations of electrodes: use `30ch.xyz` (standard channel locations)
  - Else, load `sxx_yymmdd.xyz` (digitized channel locations)



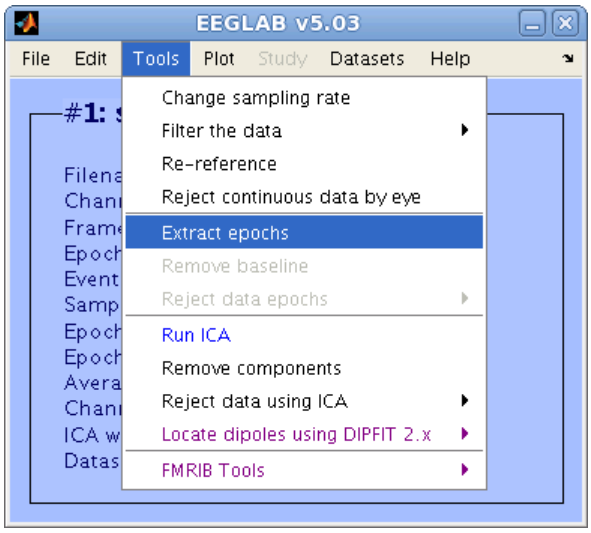
- Use “Opt. head center” to optimize head center (use default settings)



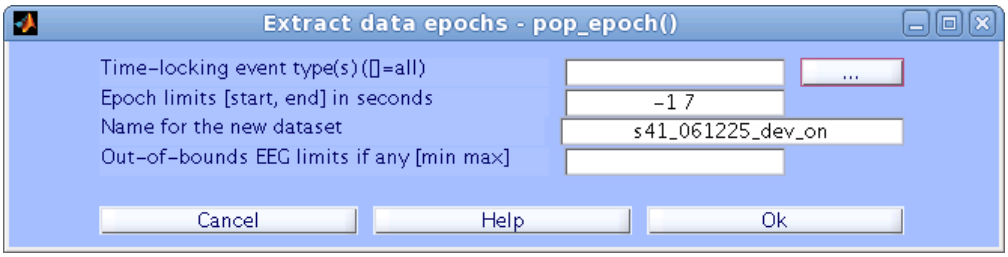
- Before pressing Ok in pop_chanedit() window…
  - Use “Plot 2-D” to check the loaded channel locations
  - Check if the number of channels in EEG data is the same as that in the imported channel locations

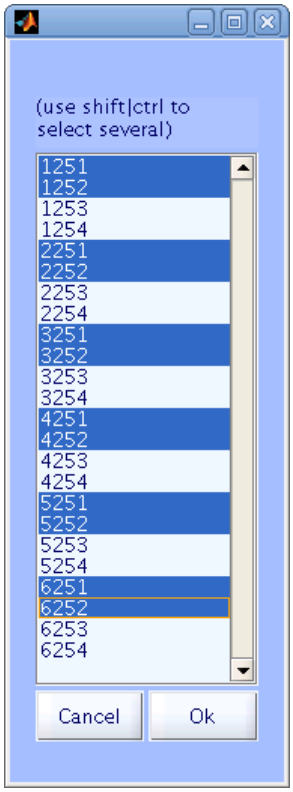
III.5 Extracting epochs

- Procedures:
  - Extract continuous EEG data into designated epochs (current: dev_on, act_on, and acf_off)
  - MATLAB function: `pop_epoch()`
  - EEGLAB GUI interface:
    - Tools → Extract epochs

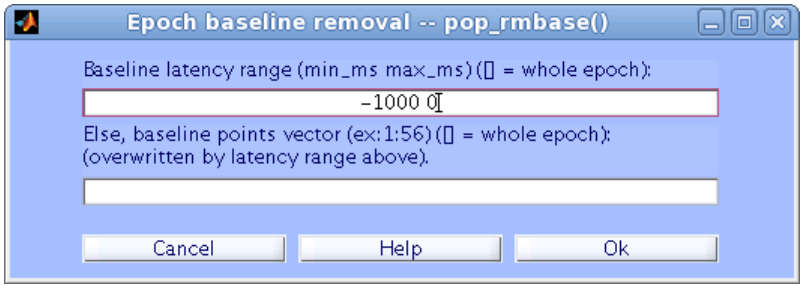


- Input event types into the text box directly or press the button in the right to select event types (dev_on: x251, x252; act_on: x253; act_off: x254)
- Input the limits in epochs (dev_on: [-1 7]; act_on: [-2 4]; act_off: [-2 4]) in seconds
- Rename the output dataset into `sxx_yymmdd_(epoch_type)`. Can be done when the "`pop_newset`" window shows





- When the "`pop_newset`" window shows, UNCHECK the "overwrite" option. Do NOT overwrite the original datasets (continuous EEG data) unless finishing extracting all kinds of epochs. (figure omitted)
- Remove baseline
  - MATLAB function: `pop_rmbase()`
  - EEGLAB GUI interface:
    - Pop-up automatically after extracting epoch, or
    - Use Tools → Remove baseline

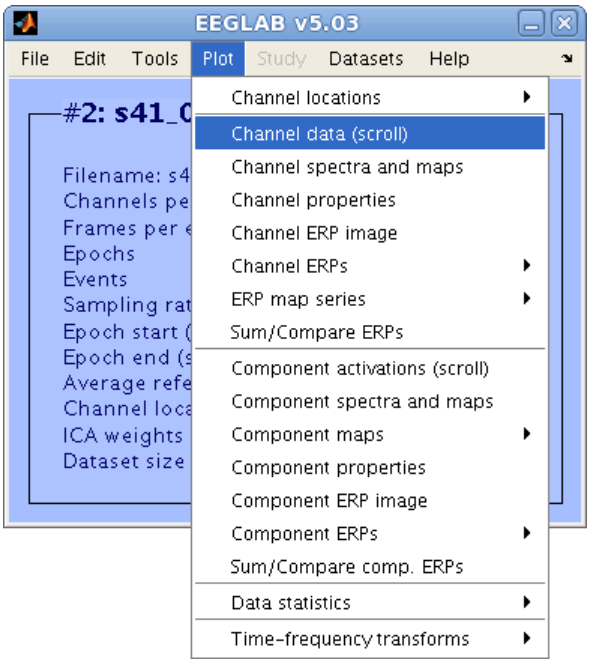


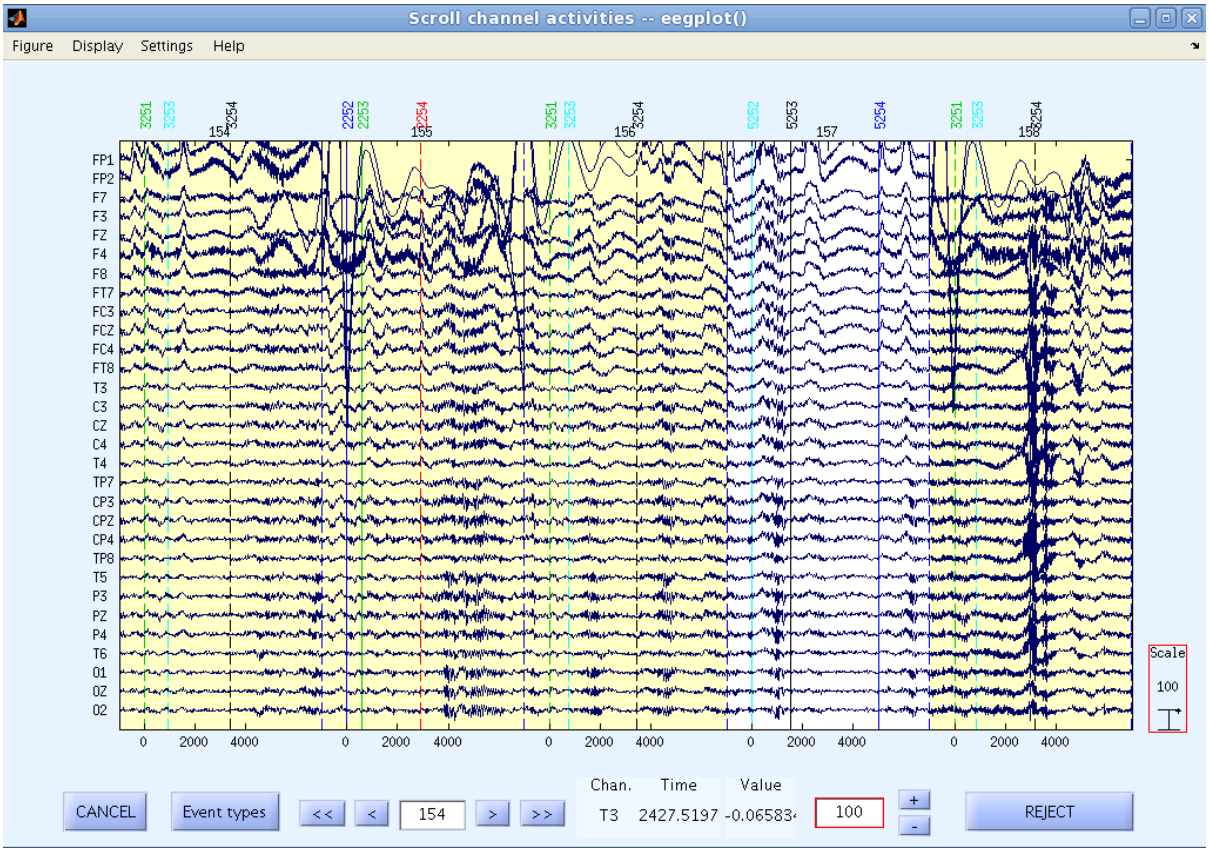
  - Range: from the beginning of each epoch (-1000 in this study) to the latency of designated events (0 in this study) in milliseconds
  - MANULLY save the dataset after finishing this step. Use the same name as the extracted dataset
- Save this dataset into "`sxx_yymmdd_(epoch_type).set`"

- Repeat the above steps until all types of epochs are extracted

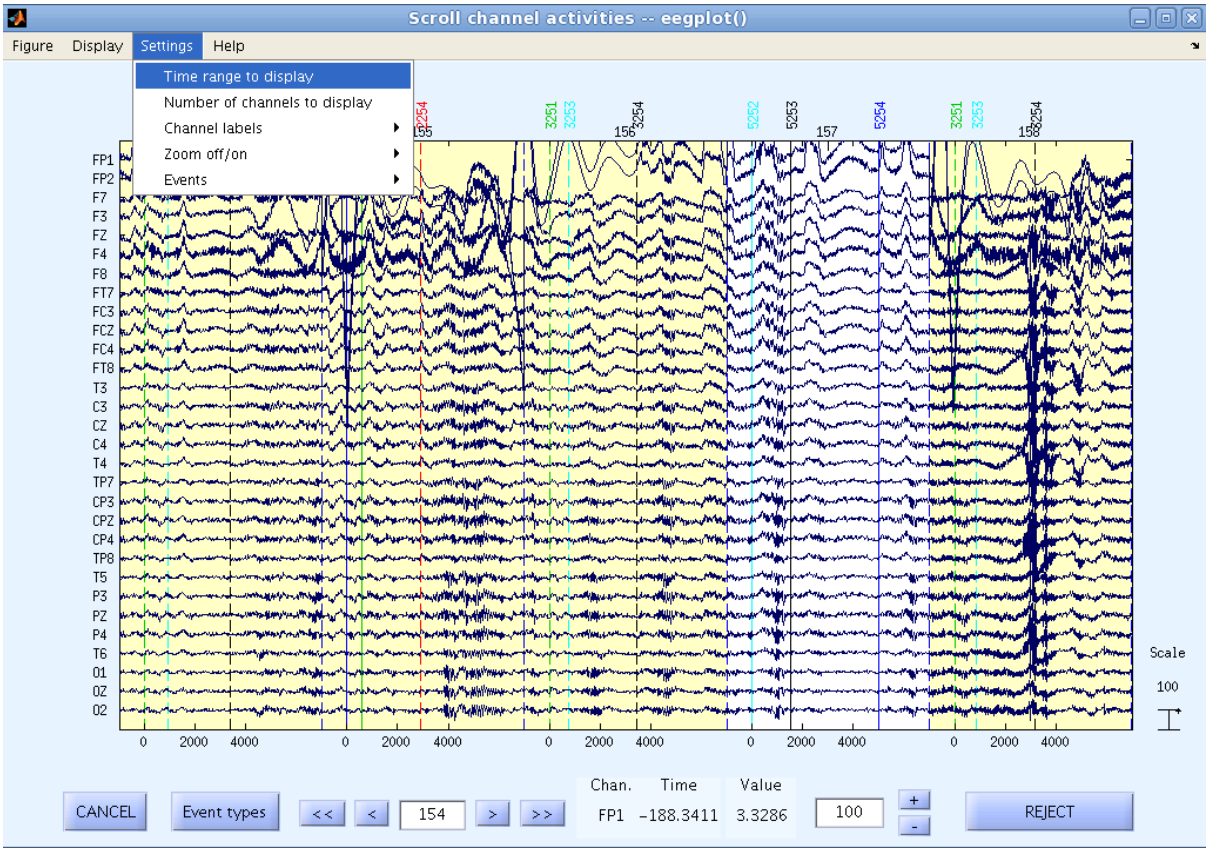
III.6 Artifact removal

- Label behavioral artifacts
  - Browse through the plotted trajectory (figure shown above)
  - Principles:
    - Reject trials with extremely short RT (RT < 0.3 sec)
    - Reject trials with "overshoot" or zigzag patterns (trajectories should be flat before dev_on and after act_off)
  - Save the indices of rejected trials to `rj_behave`
- Label EEG artifacts
  - Currently, only reject epochs time-lock to dev_on
  - Other types of epochs: reject the same epochs as those in dev_on
  - Browse through the signals of EEG data
    - MATLAB function: `eegplot()`
    - EEGLAB GUI interface: Plot → Channel data (scroll)

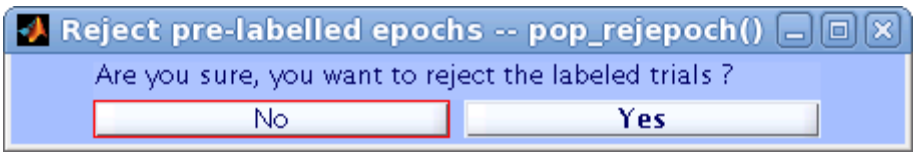




- Set the scale to 100
- Set the time range to 5 epochs in a window



- Principles:
  - Reject channels with extreme values throughout the entire or most parts of the session (the same procedure as that in “select channels”)
  - Reject epochs with severe fluctuations across most EEG channels (browsing through the data)
- Click the epoch to be rejected
- Press “REJECT” after finishing browsing and marking the epochs to be rejected

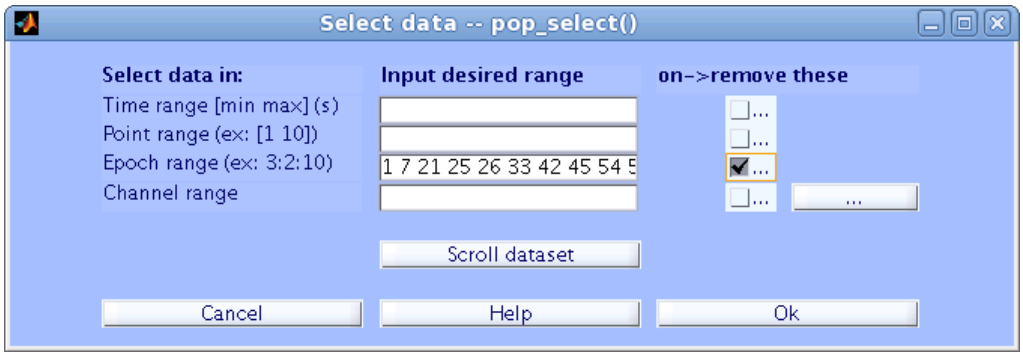


- Press NO when confirmation window pops up (do rejection manually in this study. If pressing “Yes,” EEGLAB will reject the labeled epochs)
- `>> rj_dev_on = find(EEG.reject.rejmanul);`

- Reject artifacts
  - Merge the above indices:

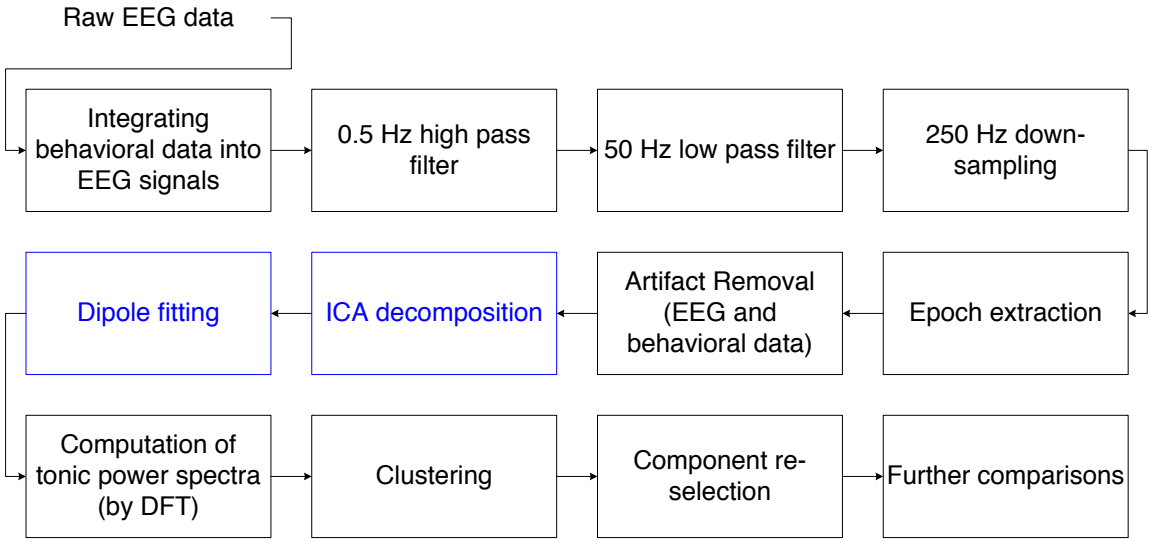
```
>> rj_latency2 = union(rj_behave, rj_dev_on);
```

- Modify the 11th column in `epoch_inf` (`1`: kept, `0`: rejected, i.e. the trials/epochs labeled above)
- Save the RT (in seconds) of the remained trial into the variable `RT` (format: `1-by-n_of_trials` array)
- Remove epochs
  - MATLAB function: `pop_select()`
  - EEGLAB GUI interface: Edit → Select data
    - Copy the contents of `rj_latency2` (using the array editor)
    - Paste the copied data into the "Epoch range" editbox and check the checkbox



- Name the new dataset "`sxx_yymmdd_(epoch_type)_rj`," and use the same name to save this dataset

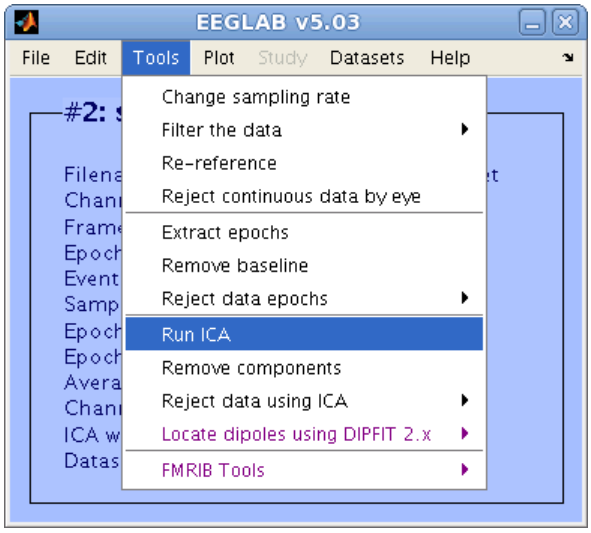
IV. ICA decomposition and dipole fitting



IV.1 ICA decomposition

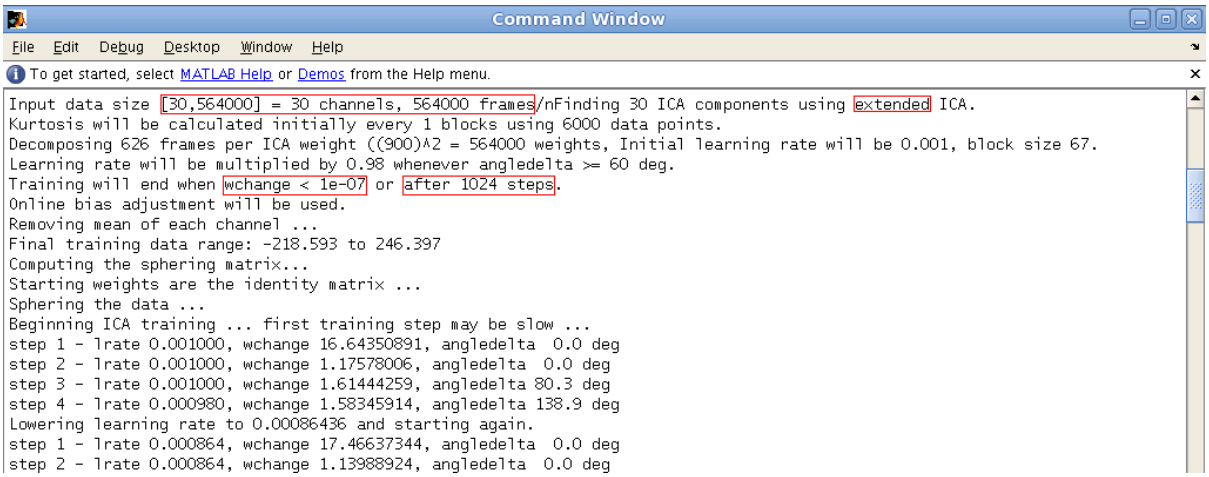
- Decompose EEG signals into independent signals
- Currently, only perform on epochs time-locking to dev_on

- MATLAB function: `pop_runica()`
- EEGLAB GUI interface: Tools → Run ICA



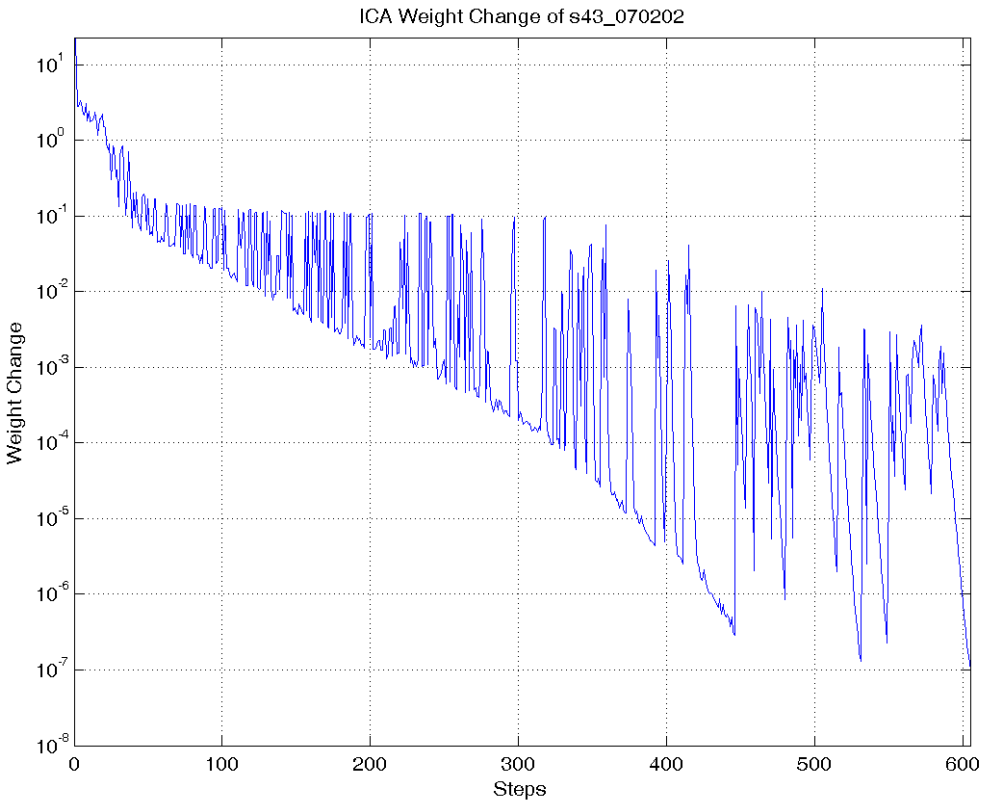
  - Settings:
    - Running extended ICA
    - Stop criteria: maximum steps = 1024 and weight change < 1E-7

    - The contents in "Command line options" editbox: (sequence doesn't matter): `'extended', 1, 'stop', 1024, 'maxsteps', 1E-7`
  - Run ICA
    - Check:

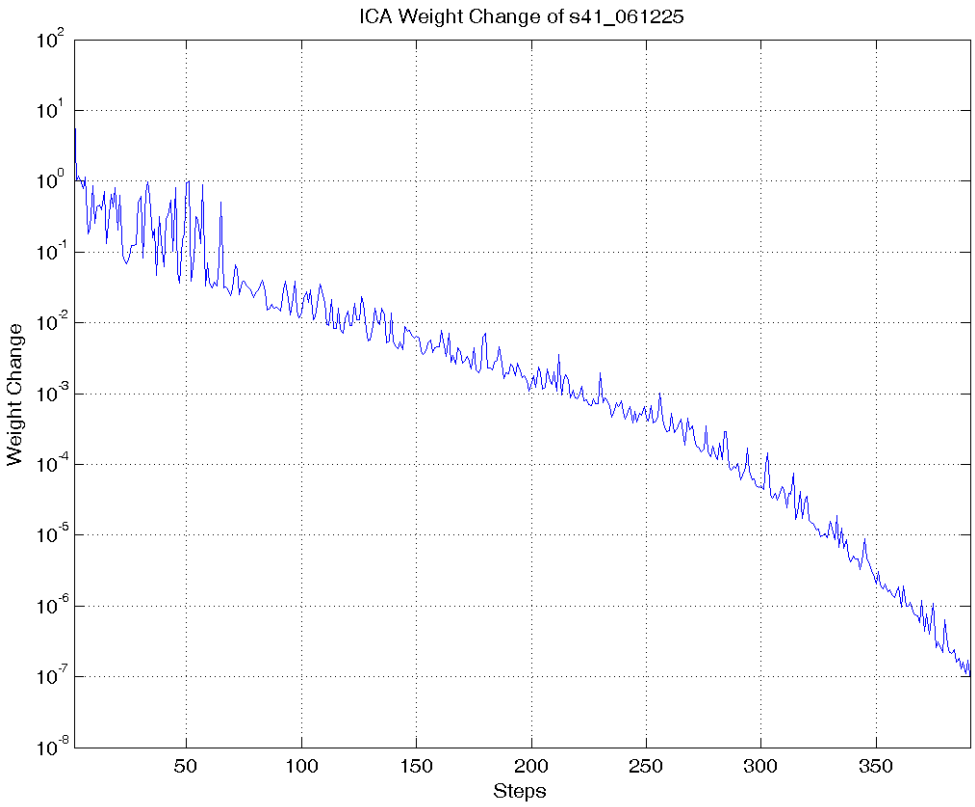


      - Frames = (# of epochs) × (length of epochs) × (sampling rate)
      - Using extended ICA
      - Stop criteria: weight change < 1e-7 or after 1024 steps
      - It is fine if ICA training start again
    - Wait for running ICA…

- After ICA training is finished…
  - Save the dataset into "`sxx_yymmdd_dev_on_rj`" (the same name)
  - Copy the whole process in ICA training and save it into `sxx_yymmdd_ICA_log.txt`
  - An alternative step:
    - Before ICA training, `>> diary('sxx_yymmdd_ICA_log.txt');`
    - After ICA training, `>> diary off`
  - Plot the step vs. weight change plot (no code, using log-scale in y axis) and save it into `sxx_yymmdd_ICA_log.png`. If the trace zigzags greatly, redo noise removal on the EEG data (the convergence of training possibly by chance)



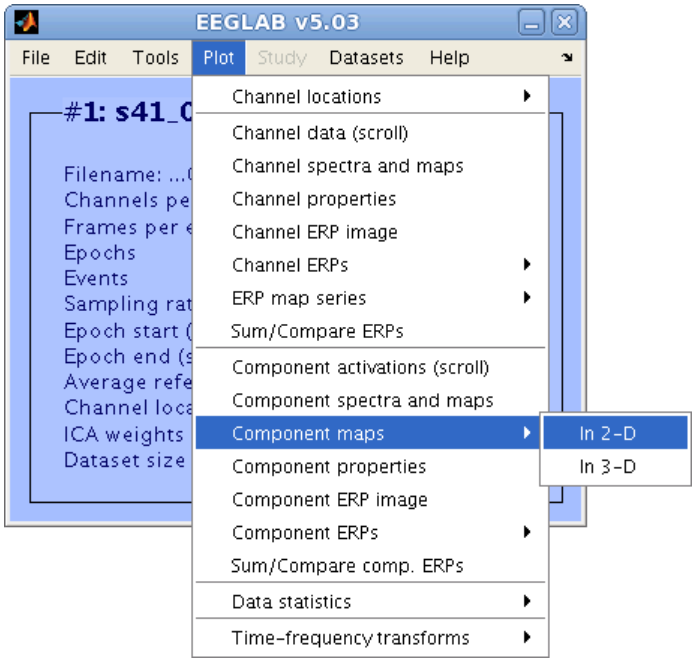
(bad ICA training)

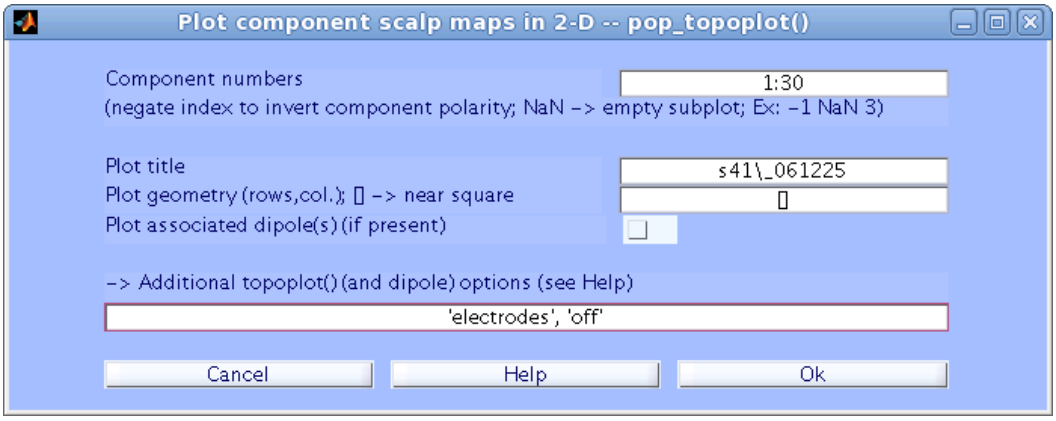


(good ICA training)

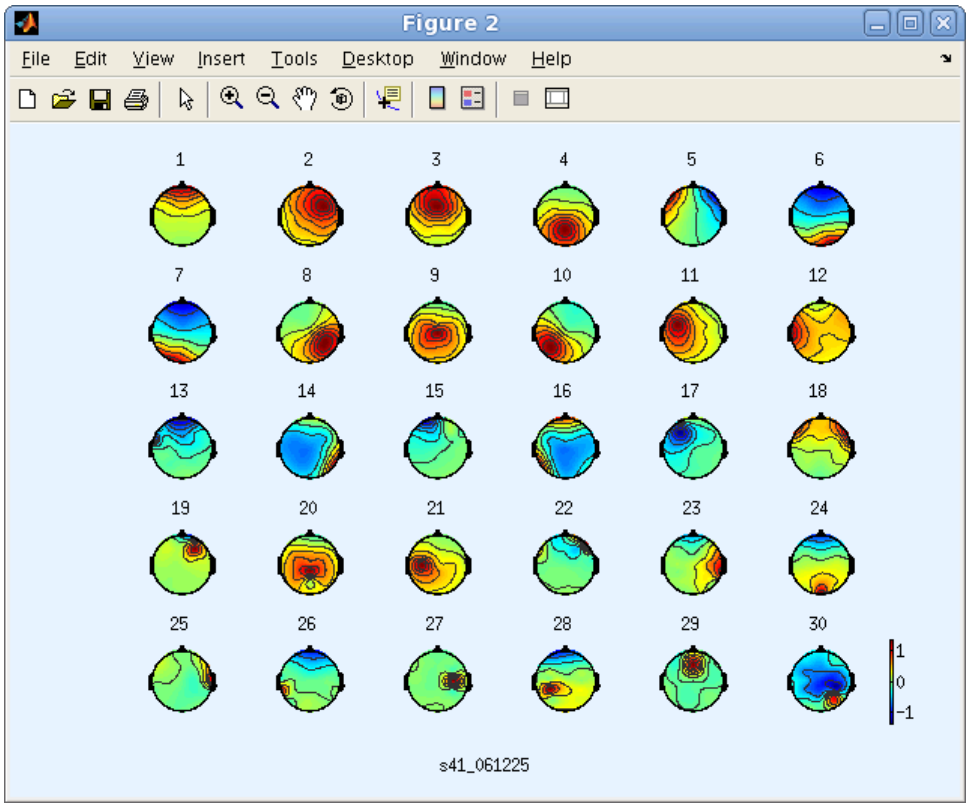
IV.2ICA scalp maps

- Show the activations on different cortical regions
- Plot the ICA scalp maps
  - MATLAB function: `pop_topoplot()`
  - EEGLAB GUI interface: Plot → Component maps → In 2-D





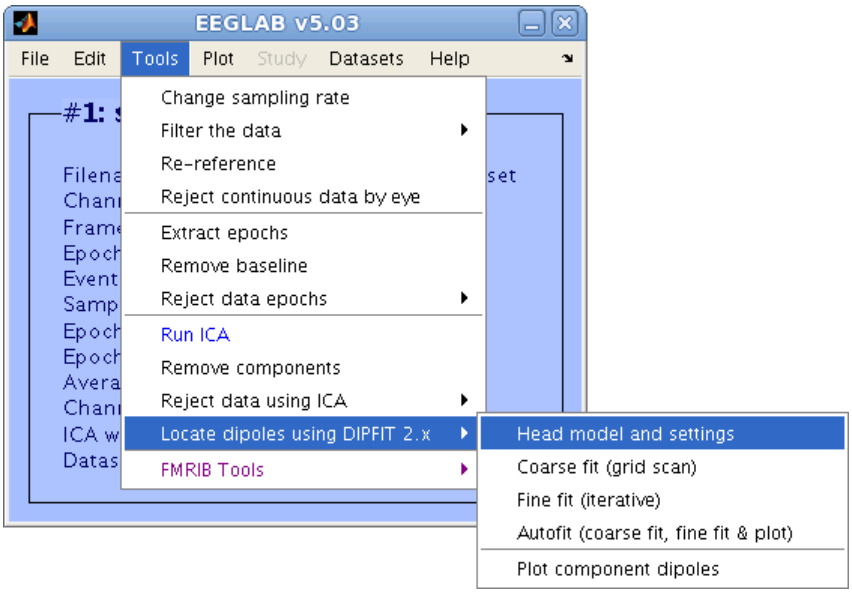
- Set plotting electrodes to off (as above) if using standard channel locations, or set to on if using digitized channel locations



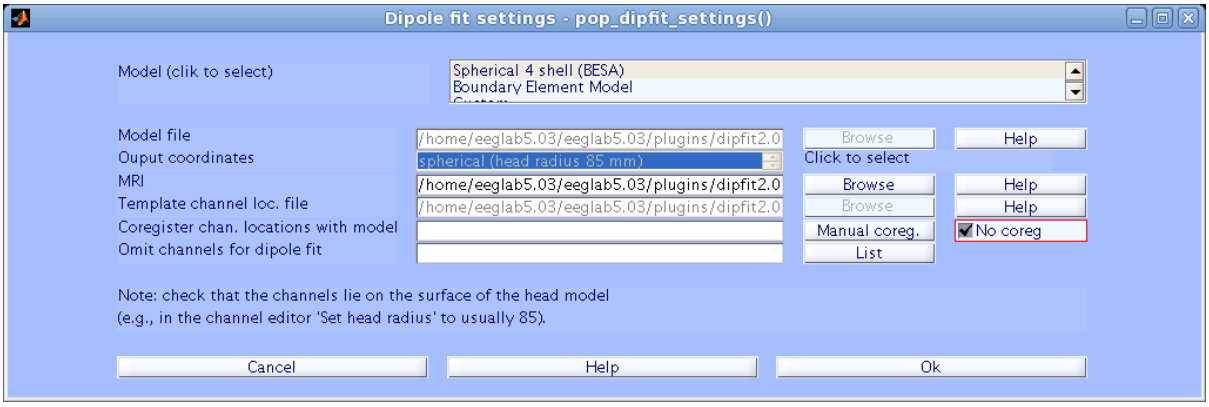
- Save the plot into "`sxx_yymmdd_component_map_intrial.png`"

IV.3 Dipole fitting

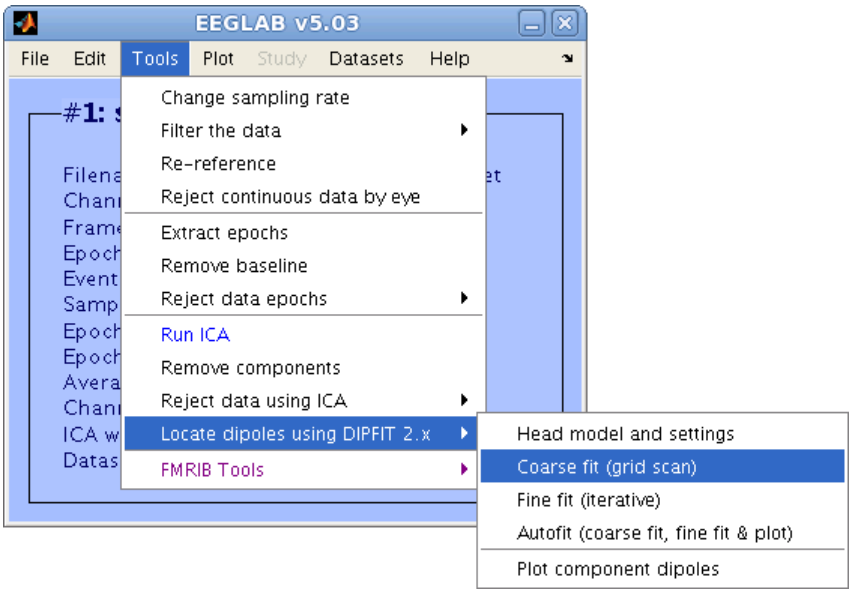
- Approximate the source of EEG signals in the brain
- Currently, only perform on epochs time-locking to dev_on
- Register channels
  - MATLAB function: `pop_dipfit_settings()`
  - EEGLAB GUI interface: Tools → Locate dipoles using DIPFIT 2.x → Head model and settings



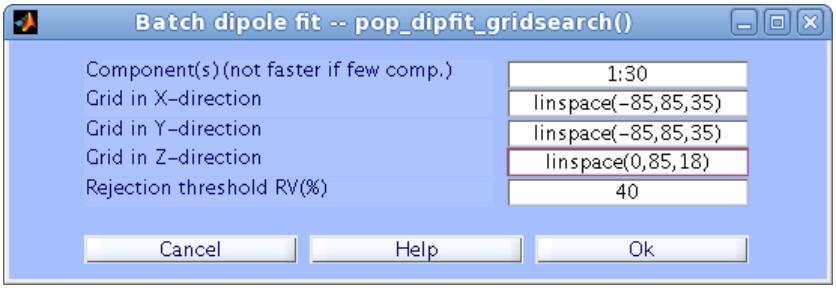
- Check “No coreg” (no coregistration)



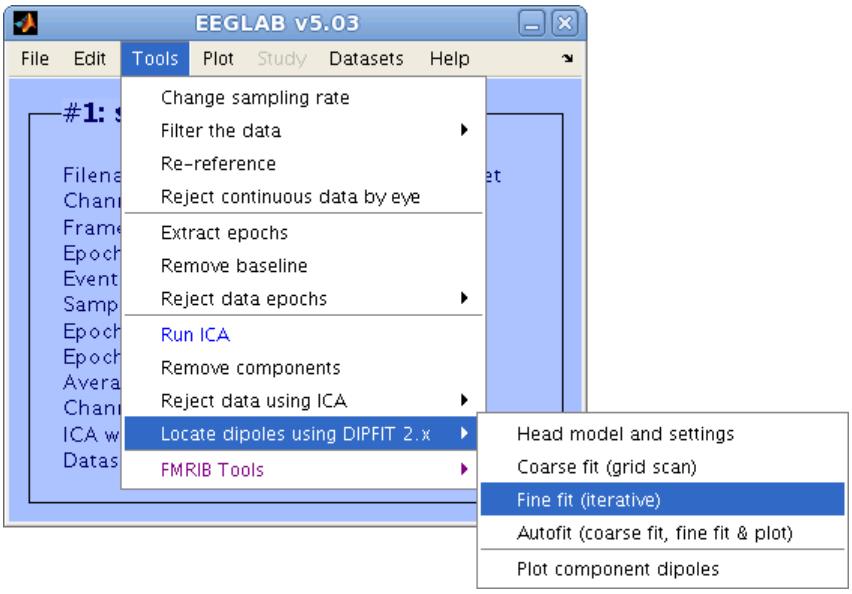
- Coarse fit: separates the brain into grids and locates dipole(s) onto appropriate grids
  - MATLAB function: `pop_dipfit_gridsearch()`
  - EEGLAB GUI interface: Tools → Locate dipoles using DIPFIT 2.x → Coarse fit (grid scan)



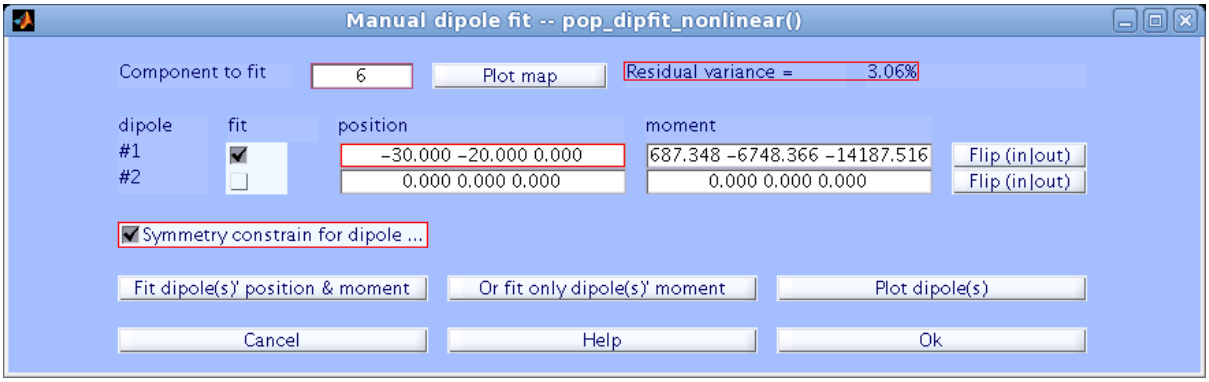
- Settings:
  - Component(s): all (the speed isn't faster if just fitting some components)
  - Grid in X-/Y- direction: `linspace(-85, 85, 35)` or `-85 : 5 : 85`
  - Grid in Z-direction: `linspace(0, 85, 18)` or `0 : 5 : 85` (That is, length of each side of a grid is 5 mm)
  - Rejection threshold RV (%): `40` (default)



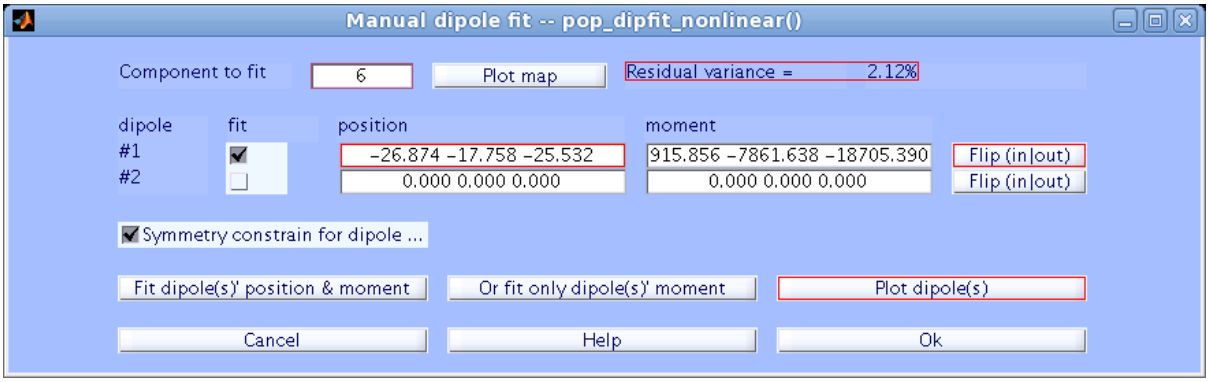
- Fine fit: locates dipoles into appropriate grids
  - Fit only the components related to brain processes (e.g., in the above component map, only fit components 2, 3, 4, 6, 7, 8, 9, 10, 11, and 21)
  - MATLAB function: `pop_dipfit_nonlinear()`
  - EEGLAB GUI interface: Tools → Locate dipoles using DIPFIT 2.x → Coarse fit (grid scan)



- Basic settings:
  - Component to fit: the desired component (one component/once)
  - Dipole (#1 or #2): the dipole to fit. Currently, only bilateral and tangential occipital components need to fit two dipoles; otherwise, fit only one dipole
  - Symmetry constrain for dipole...: check if fit two dipoles (if fitting only one dipole, it doesn't matter if this checkbox is checked)

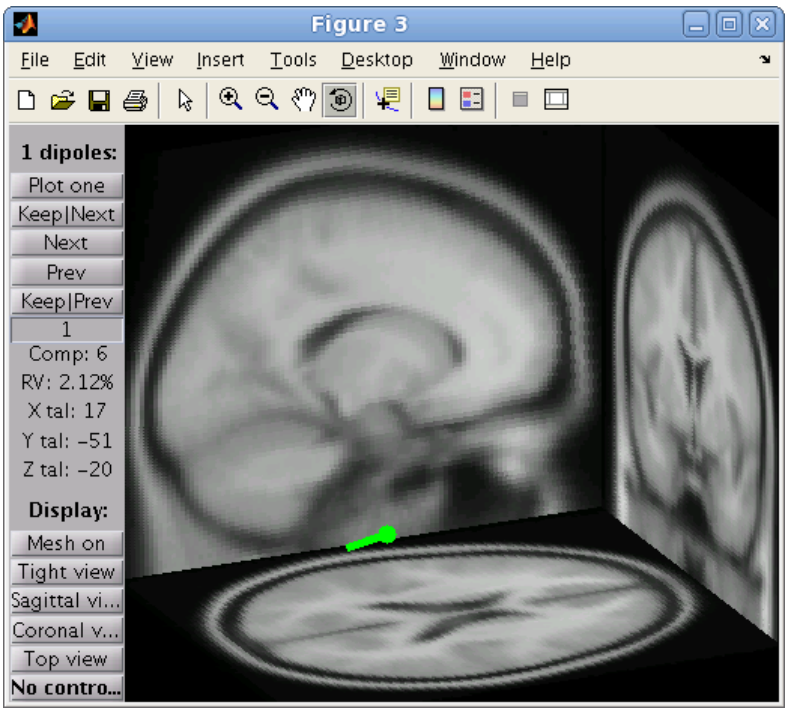


  - After fine fitting, notice the value in the "Residual variance" field should be less than the original and the value in the "position" field is usually fractions



- Use "Plot dipole(s)" to see if the dipole(s) is/are located reasonably. If

not, redo fine-fitting on this component



- Repeat the above steps unless finishing fine-fitting all the components related to brain processes

- Save the dataset into "`sxx_yymmdd_dev_on_rj`"

IV.4 Apply ICA weight matrices and dipole locations on other types of epochs

- Currently, only the dev_on epoch performs ICA decomposition and dipole fitting
- Other types of epochs: directly apply the results of those in the dev_on epoch
- ICA weight matrices:
  - `EEG.icaact`
  - `EEG.icawinv`
  - `EEG.icasphere`
  - `EEG.icaweights`
  - `EEG.icachansind`
- The dipole location matrix
  - `EEG.dipfit`
- Procedures
  - Load `sxx_yymmdd_dev_on_rj.set`. This dataset should have ICA weight matrices and dipole locations (if not, perform ICA decomposition and dipole fitting)
  - Use other variables to copy ICA weight matrices and dipole locations:

```
>> % icaact = EEG.icaact;  %no need to copy this
>> icawinv = EEG.icawinv;
>> icasphere = EEG.icasphere;
```

```
>> icaweights = EEG.icaweights;
>> icachansind = EEG.icachansind;
>> dipfit = EEG.dipfit;
```

- Load the dataset with other epoch: `sxx_yymmdd_(epoch_type)_rj.set`
- Copy the values in the above variables

```
>> % EEG.icaact = icaact;  %%%WRONG!!
>> EEG.icaact = icaweights * icasphere * ...
      reshape(EEG.data, size(EEG.data, 1), ...
      size(EEG.data, 2) * size(EEG.data, 3));
>> EEG.icaact = reshape(EEG.icaact, ...
      size(EEG.data, 1), size(EEG.data, 2), ...
      size(EEG.data, 3));
>> EEG.icawinv = icawinv;
>> EEG.icasphere = icasphere;
>> EEG.icaweights = icaweights;
>> EEG.icachansind = icachansind;
>> EEG.dipfit = dipfit;
```

- `>> eeglab redraw;  %upgrade the information in the dataset`
- Save the dataset into `sxx_yymmdd_(epoch_type)_rj.set`
- Repeat the above steps until all types of epochs have ICA weight matrices and dipole locations

V. Automatically preprocessing

V.1 Why using MATLAB scripts for pre-processing

- Drawbacks of using EEGLAB menu to pre-process EEG data
  - Time-consuming
    - Wait for finishing one step
    - Wait for user's response
  - Make mistakes easily
    - Type wrong numbers, press wrong key, ...
  - Arduous and inefficient
- Since you won't pre-process/analysis only one EEG dataset, why not using/writing MATLAB scripts?
- Currently, MATLAB script for pre-process is available

V.2 MATLAB scripts for pre-processing

- Script: `DR_preprocessing_compact.m` (ver 200809)
  - Pre-process EEG data semi-automatically

- With several breakpoints to prevent from overwriting data
- Some steps still need manual operations

- Can process multiple datasets
- Integrate pre-process of behavioral and EEG data in a whole (the parts III and IV in this document)
- Required toolbox/scripts:
  - EEGLAB ver. 5.03
  - `mypath.m` (set the path in MATLAB)
  - `DR_pop_loadcnt.m`
  - `DR_loadcnt().m`
  - Still have to make preprocessing log (contents described before)
- Check before you run:
  - The path is correct or not
  - Select or add dataset in the variable `SET`
  - Check if the output directory has other files. Files with the same names as the output files will be OVERWRITTEN!
- The following steps need manual operation: (so check the progress occasionally)
  - Artifacts removal (both behavioral and EEG data)
  - Plot ICA weight change
  - Fine-fitting dipoles
- Outputs
  - Datasets
    - `sxx_yymmdd_pre.set`: filtered and down-sampled EEG data
    - `sxx_yymmdd_(epoch_type).set`: epoched EEG data
    - `sxx_yymmdd_(epoch_type)_rj.set`: epoched and artifact-removed EEG data with ICA weights and dipole locations
  - `MAT` files
    - `sxx_yymmdd_epoch_inf.mat`
      `epoch_inf`: time frames of the events in a trial, the previous trial, and the next trial, and a flag to indicate if the trial is kept or not
    - `sxx_yymmdd_RT_of_trials.mat`
      `RT`: reaction time (sec) of the remained trials after artifact removal
      `RT_original`: reaction time (sec) of the original trials
    - `sxx_yymmdd_rejected_trials.mat`
      `rj_behave`: trials behavioral artifacts
      `rj_dev_on`: epochs with artifacts in EEG data
      `rj_latency2`: union of the above two variables
      `rj_latency1`: (obsolete, kept for future use)
  - Figures (shown in part III and IV)

- `sxx_yymmdd_raw_traj.fig`: raw trajectory in a session
- `sxx_yymmdd_ICA_log.png`: log file for ICA training [plot manually]
- `sxx_yymmdd_component_map_intrial.png`: component map after ICA decomposition

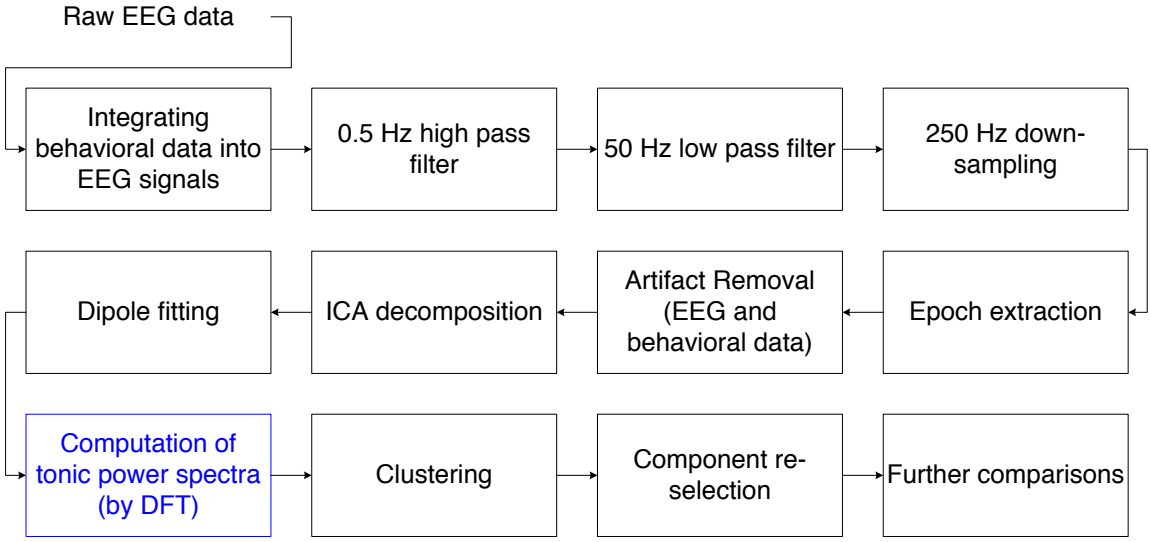
◆ Txt file

- `sxx_yymmdd_preprocess_log.txt`: preprocessing log file [create and write the contents manually]
- `sxx_yymmdd_ICA_log.txt`: log file for ICA training

■ Knowing bugs

◆ Bugs in applying ICA weight matrices and dipole locations on other types of epochs. The epochs may not have the desired ICA weight matrices and dipole location. Check after pre-process ends

VI. Computation of tonic power spectra



- Script: `DR_PBase_setup.m`

■ Save the information about components from all datasets and settings for plotting figures

◆ The “database” of subjects and components (for checking)

■ Add or remove datasets/components if needed

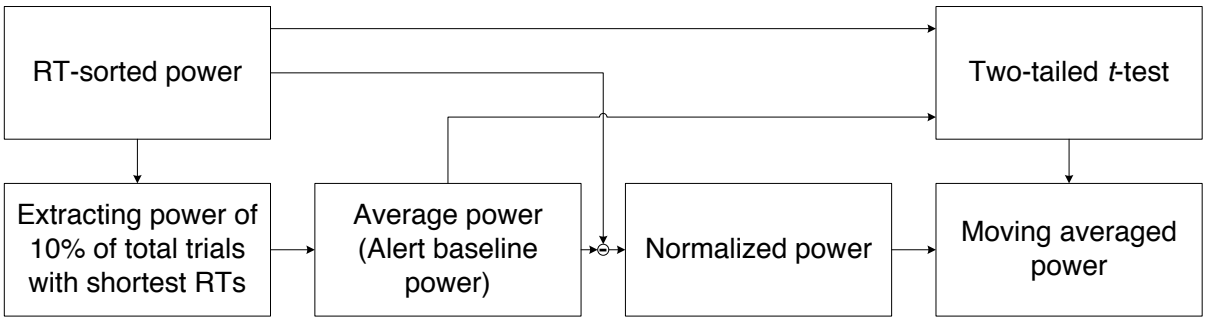
◆ Be cautious!

◆ Components and clusters #:

- Clusters #: use the settings in `DR_PBase_setup.m`
- Pre-classify when pre-process ends
- Modify after clustering (described in Section VII.2): make the cluster # negative (e.g.: 5 → -5)

- All processes below need this script

- Script: `RS_powerbase.m` (ver 200901)
  - Compute and plot power spectra for SINGLE-SUBJECT data
  - Two parts in this script:
    - Compute power spectra
      - Flowchart:



    - Plot power spectra
  - Required scripts
    - `mypath.m`
    - `DR_PBase_setup.m`
    - `timefreq2004.m`: performs time-frequency transform. The same as `timefreq.m` in UCSD, rename to prevent confusing in MATLAB
  - Required inputs
    - Components and identified clusters (in `DR_PBase_setup.m`)
    - ICA activation of the above components (in EEG dataset)
  - Check before you run:
    - Variables
      - `PLOT_SET`: the plotted datasets
      - `MN`: conditions. (`'motionless'` or `'motion'`)
      - `p_val`: significance level (currently `1E-4`)
      - `epoch_type`: types of power spectra
        `''` ➔ tonic spectra
        `'dev_on'` ➔ phasic spectra
        `'all'` ➔ mixed spectra
      - `rj`: `'_rj'` ➔ artifacts removed dataset
    - Directories
      - Tonic power spectra: `FilePath/MN/IC00~IC10`
      - Other power spectra: `FilePath/epoch_type/MN/IC00~IC10`
  - Detailed process of computing power spectra
    - Perform time-frequency transform using `timefreq2004.m`
      - `[PB, freqs, times] = timefreq2004(baseline, SR, 'wavelet', 0, 'freqs', [min_freq max_freq], 'winsize', 128, 'padratio', 2, 'ntimesout', 100);`

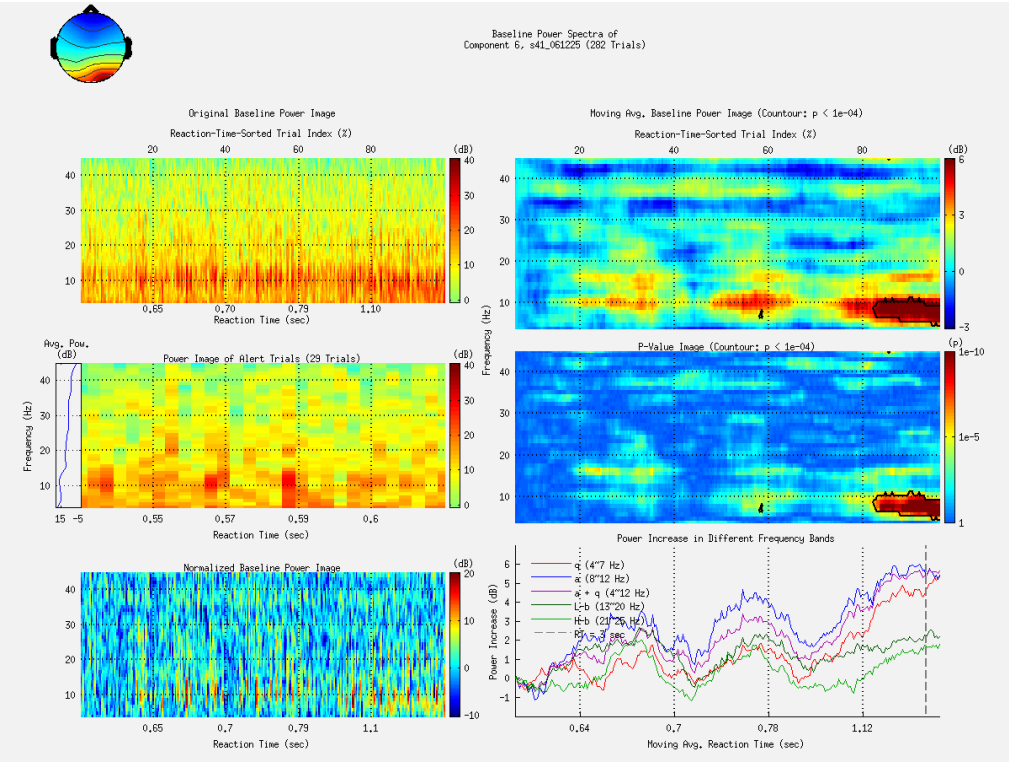
`baseline`: ICA activations in (part of) an epoch; not necessarily the activations before deviation

- Compute magnitude and convert into dB
  - `PB = 10 * log10(PB .* conj(PB));`
- Average the resulting spectra in different time windows and convert into a 2-D array
  - `PB_mean = trimmean(PB, 10, 2);`
  - `PB_mean = reshape(PB_mean, size(freqs, 2), n_of_trials);`
- Choose desired frequency range
  - `PB_mean = PB_mean(find(freqs >= 3 & freqs <= 45), valid_trial) %valid_trial: trials with desired RTs`
  - `freqs = freqs(find(freqs >= 3 & freqs <= 45));`
- Sort by RT
  - `PB_mean = PB_mean(:, ur_idx);`
- Compute alert baseline spectra (spectra from trials with shortest 10% RT)
  - `PB_alert = PB_mean(:, 1 : alert.trials);`
  - `PB_alert_mean = trimmean(PB_alert, 10, 2) * ones(1, size(valid_trial, 2));`
- Normalize
  - `PB_n = PB_mean - PB_alert_mean;`
- Moving average (RT and power spectra, using `for` loop)
  - Power spectra: `PB_mov = [PB_mov trimmean(PB_n(:, k : k + alert.trials - 1), 10, 2)];`
  - RT: `RT_s_mov = [RT_s_mov trimmean(RT_s(:, k : k + alert.trials - 1), 10, 2)];`

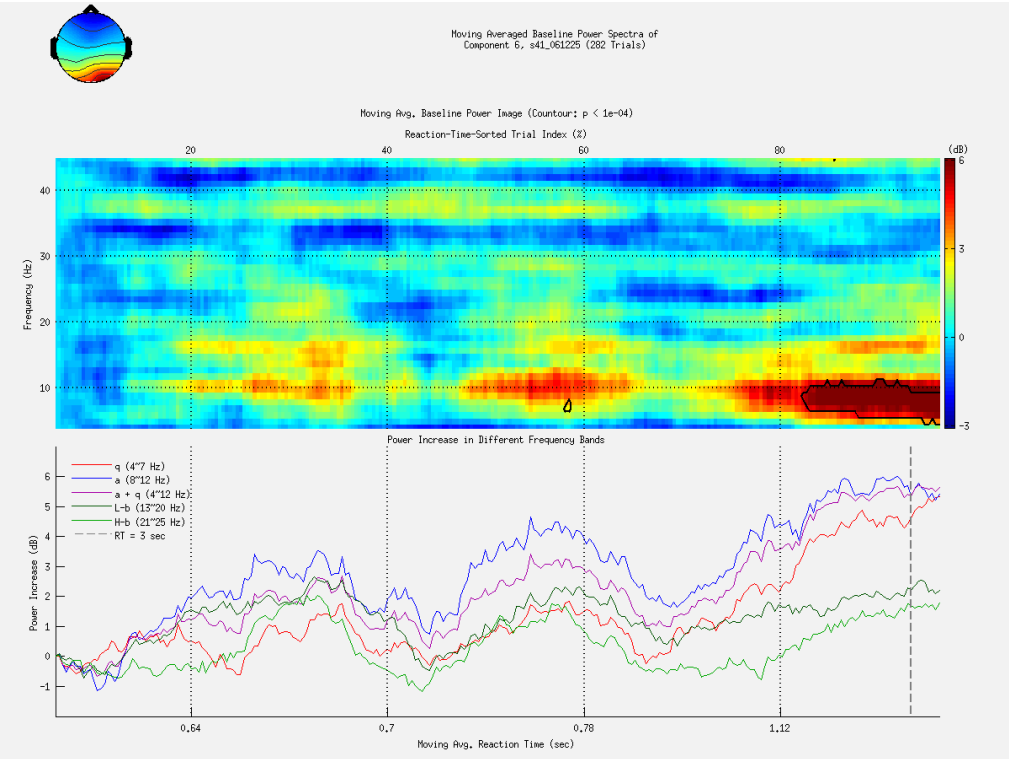
- Detailed process of applying two-sampled *t*-test (using `for` loop)
  - A moving window
    - `PB_tmp = PB_mean(:, sig_idx(k) : sig_idx(k) + alert.trials - 1);`
  - Estimate significance level
    - `[H, P] = ttest2(PB_tmp', PB_alert_mean(:, 1 : alert.trials)', p_val / 2 / size(freqs, 2), 'both');`
    - `PB_alert_mean(:, 1 : alert.trials)`: alert baseline spectra
    - `p_val / 2 / size(freqs, 2)`: *p* value corrected by degree of freedom
    - `H`: show if the power on this frequency bin is significant
    - `P`: *p* value on this frequency bin
  - Merge into the matrix (for adding contour on power image)

- `H_mask(:, k) = H';`
- `P_mask(:, k) = P';`

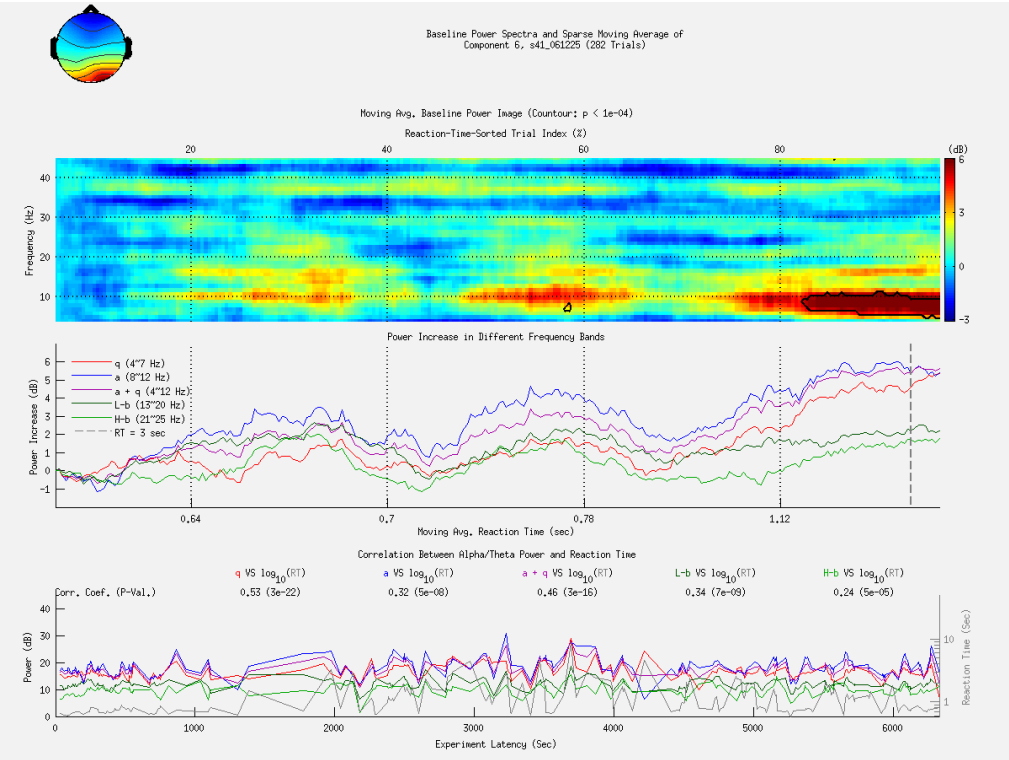
- Outputs
  - `MAT` file
    - `sxx_yymmdd_PBaseRT_(component #)(rj).mat`
      `icawinv`: ICA inverse weight matrix of this component
      `chanlocs`: channel locations of this component
      `PB`: original power spectra computed by `timefreq2004.m`
      `PB_mean`: power spectra (averaged across time windows)
      `PB_alert`: power spectra of alert trials
      `PB_alert_mean`: averaged power spectra of alert trials (alert baseline spectra)
      `PB_n`: normalized power spectra
      `PB_mov`: moving averaged power spectra (from `PB_n`)
      `freqs`: output frequency bins
      `RT`: reaction time of all trials
      `RT_s`: sorted reaction time
      `RT_s_mov`: moving averaged reaction time
      `FREQ_INC`: power spectra of the extracted frequency bands
      `H_mask`: significant region of power spectra
      `P_mask`: *p* value of individual frequency/trial
      `p_val`: the set *p* value (uncorrected)
      `URPB`: (sparse moving average)
      `URPB_mov`: (sparse moving average)
  - Figures: [if plotting tonic power spectra, this script plots in `fig` and `png` files; else, only plots in `png` format]
    - Power spectra after each step: `sxx_yymmdd_PBaseRT_(component #)(rj)_method.png/fig`



- Power image and trends of power increase on the frequency bands defined in `DR_PBase_setup.m`: `sxx_yymmdd_PBaseRT_(component #)(rj)_result.png/fig`

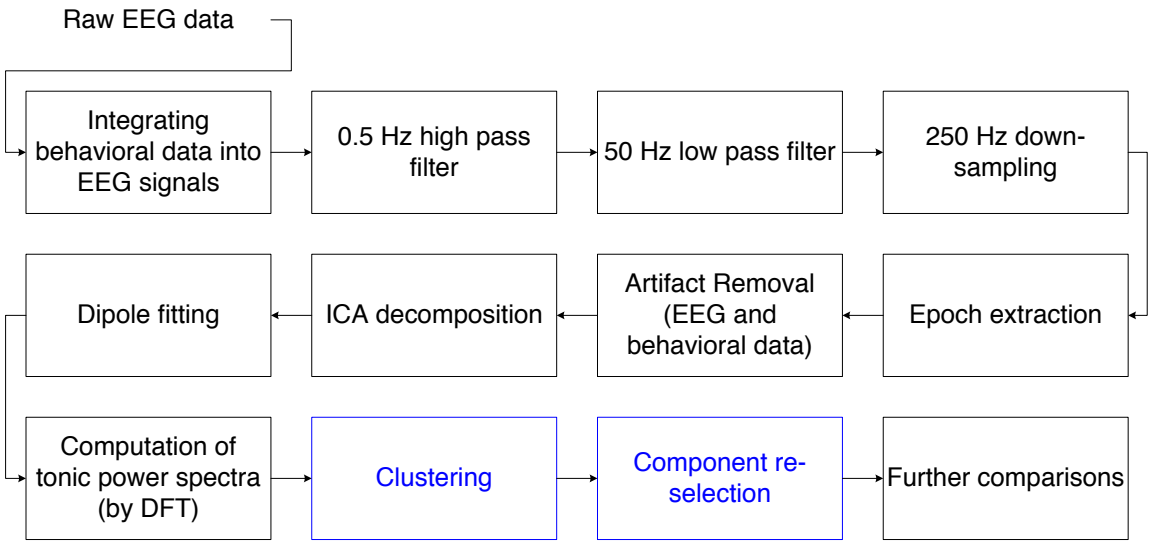


- The same plots in the above point and sparse moving average on the previous bands: `sxx_yymmdd_PBaseRT_(component #)(rj)_sparse.png/fig`



- Knowing bugs: (not reported)

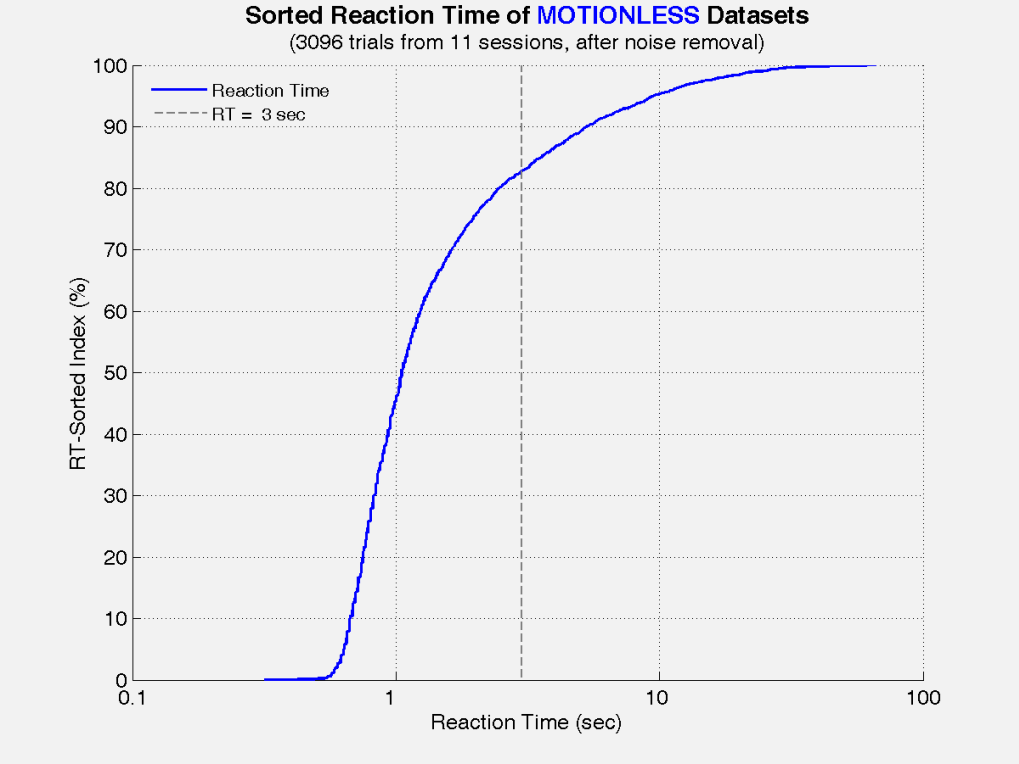
VII.Clustering and component re-selection



- Clustering: characterize the common pattern of spectral activities of similar ICs from different subjects and sessions
- Component re-selection: remove abnormal spectra/component scalp maps from IC clusters to refine the trend
- Re-compute cluster spectra after re-selecting components

VII.1 Clustered RTs

- Script: `DR_PlotClustRT.m` (ver 200903)
  - Plot CLUSTERED RTs under motionless or motion condition
  - Required scripts
    - `mypath.m`
    - `DR_PBase_setup.m`
  - Required inputs
    - RTs of trials after artifact removal
  - Check before you run:
    - Variables
      - `PLOT_SET`: the plotted datasets
      - `MN`: conditions. (`'motionless'` or `'motion'`)
      - `MN_COLOR`: color code used to indicate motionless or motion conditions
      - `rj`: `'_rj'` ➔ artifacts removed dataset
    - Directories: `FilePath/MN/`
  - Outputs
    - `MAT` file: `RT_Sorted_Trial_(MN).mat`
      - `urRT_s`: sorted clustered RTs (before artifacts removal)
      - `RT_s`: sorted clustered RTs (after artifacts removal)
      - `MN`: motionless or motion condition
      - `session_count`: # of sessions in this cluster
    - Figures: `RT_Sorted_Trial_(MN).fig`. Print the `png` file in PC



VII.2 Clustering ICA scalp maps, alert baseline power, and dipole lo-

cations

- Script: `DR_PlotClustCompPowerDipole.m` (ver 200903)
  - Compute and plot averaged scalp map, alert baseline power, and dipole locations in a cluster
  - Required scripts
    - `mypath.m`
    - `DR_PBase_setup.m`
  - Required inputs
    - ICA activation of the components in EEG datasets
    - Identified clusters with components from EEG datasets
    - Standard channel locations (for finding missing channels in datasets)
    - Power spectra of the components in the plotted cluster
    - Dipole locations of the components in the plotted cluster (optional)
  - Check before you run:
    - Variables
      - `PLOT_SET`: the plotted datasets
      - `MN`: conditions. (`'motionless'` or `'motion'`)
      - `MN_COLOR`: color code used to indicate motionless or motion conditions
      - `n_of_cls`: # of cluster; `undef_cls`: undefined cluster
      - `rj`: `'_rj'` ➔ artifacts removed dataset
      - `std_chanlocs`: file of standard channel locations (for checking missing channels)
      - `avg`: how to compute average. `'_trim'`: using 10% trimmed mean
    - Directories: `FilePath/MN/`
  - Detailed process of computing average scalp map in a cluster
    - Automatically negate the weights if needed to make all the clusters in this cluster with similar distribution in color: normalize each weight column vector according to it maximum absolute value
      - Determine if each scalp map in this cluster has missing channels

        ```
        [tmp_chanlabel chanlabel_diff] =
        setdiff(upper(std_chanlabel), upper(chanlabel));
        ```

      - If any dataset in this cluster has missing channels, fill the corresponding positions in ICA inverse weight vector with `NaN`, and then negate the weight if needed

        ```
        icawinv_tmp(find(~isnan(icawinv_tmp))) =
        EEG.icawinv(:, plot_comp);
        [abs_max max_idx] = max(abs(icawinv_tmp));
        icawinv = [icawinv icawinv_tmp /
        ```

```
icawinv_tmp(max_idx)];
```

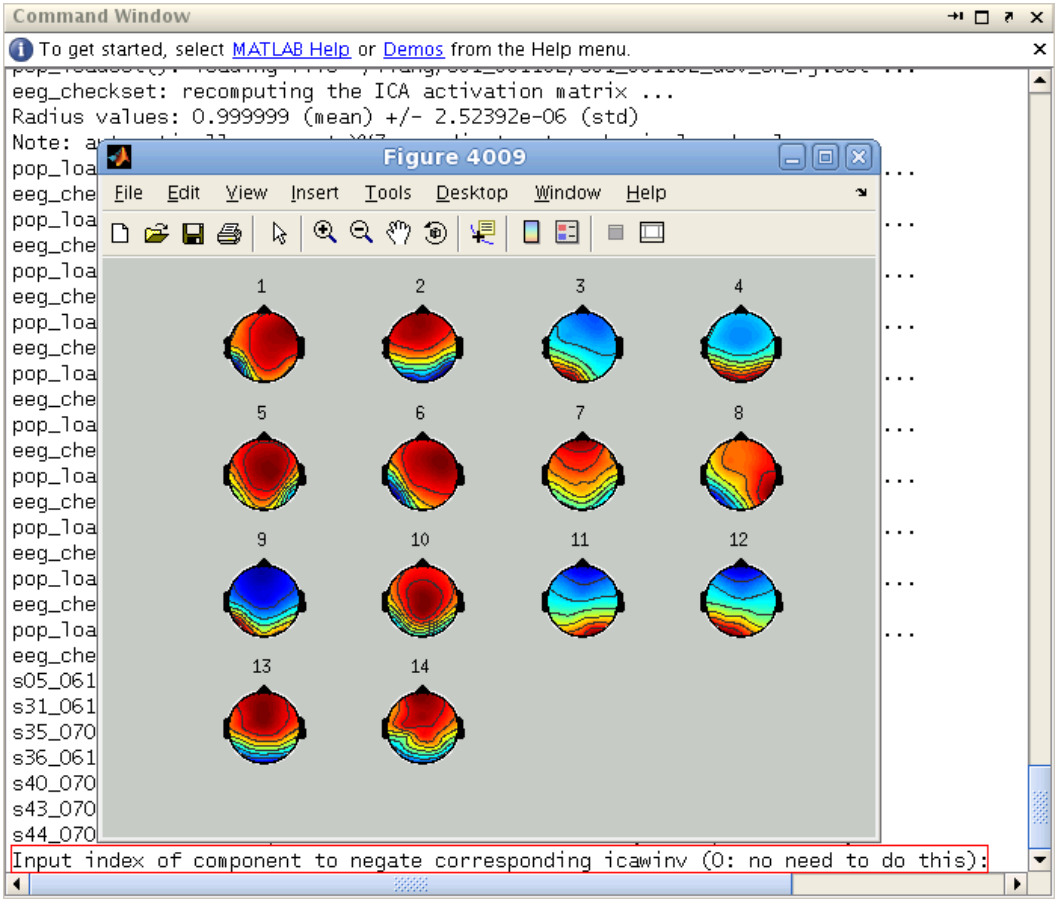
- If with no missing channels, negate the weights if needed

```
[abs_max max_idx] =
max(abs(EEG.icawinv(:, plot_comp)));
icawinv = [icawinv (EEG.icawinv(:, plot_comp)) /
(EEG.icawinv(max_idx, plot_comp))];
```

◆ Collect inverse weight matrix in a cluster

- `chan_all = [chan_all chanlocs];`

◆ Plot each scalp map in this cluster for manually negating the weights (enter the indices in the MATLAB command line in the matrix form)



◆ Compute average inverse weight vector in this cluster

- If any dataset in this cluster has missing channels, compute the average vector row by row

```
icawinv_tmp = find(~isnan(icawinv(k, :)));
icawinv_tmp = mean(icawinv(k, icawinv_tmp));
avg_icawinv = [avg_icawinv; icawinv_tmp];
```

- If with no missing channels, compute the average vector directly

```
avg_icawinv = mean(icawinv, 2);
```

◆ Compute average paired correlation in this cluster

- Compute paired correlation and take upper triangle matrix of the resulting matrix

```
comp_corr = triu(corrcoef(icawinv));
```

- Set the diagonal value to `0` (original `1`)

```
diag_matrix = zeros(size(comp_corr));
for k = 1 : size(comp_corr, 1)
  diag_matrix(k, k) = comp_corr(k, k);
end
```

- Compute average and SD of correlation value

```
tmp = find(comp_corr > 0);
avg_corr = mean(comp_corr(tmp));
sd_corr = std(comp_corr(tmp));
```

■ Detailed process of computing average dipole locations

◆ Fit dipoles location if the input dataset does not have dipole locations (doing when loading datasets; the same procedures as those in the pre-processing part)

- `if ~isfield(EEG, 'dipfit') || isempty(EEG.dipfit)`

```
  ......
end
```

◆ Using dipplot() to compute the Talairach coordinates of each dipole; save in `dip_tmp` for further use

- `eval(...`

```
['dip_tmp = dipplot(EEG.dipfit.model(plot_comp),
''mri'', mrifile, ''view'', view_angle, ''color'',
{[1 1 0]}, ' ...
'''dipolesize'', 20, ''dipolelength'', 0, '
[dip_option ...
', ''projimg'', ''off'', ''projlines'', ''on'',
''coordformat'', ''spherical'''] ');']);
```

◆ Update and save the original EEG dataset

- `EEG.dipfit.model(plot_comp).eleccoord = dip_tmp.eleccoord;`
- `EEG.dipfit.model(plot_comp).mnicoord = dip_tmp.mnicoord;`
- `EEG.dipfit.model(plot_comp).talcoord = dip_tmp.talcoord;`
- `EEG = pop_saveset(EEG, [FilePath plot_set SL plot_set EpochType{1} rj '.set']);`

◆ Collect dipole locations and compute total residual variance in a cluster

- `dipole_all = [dipole_all dip_tmp];`
- `rv_all = rv_all + dip_tmp.rv;`

◆ Collect Talairach coordinates and other information in a cluster [use

`talcoord_all` (Talairach coordinates of dipoles) for example]

- If the cluster is the occipital or tangential occipital cluster (possibly with two dipoles)

```
if dip_tmp.talcoord(k, 1) < 0  %left
  talcoord_all(j, 1:3) = dip_tmp.talcoord(k, :);
  ......
else  %right
  talcoord_all(j, 4:6) = dip_tmp.talcoord(k, :);
  ......
end
```

- If the cluster is one of the other clusters (only one dipole)

```
talcoord_all(j, :) = dip_tmp.talcoord(1, :);
```

◆ Compute mean and SD of residual variance and Talairach coordinates in a cluster [use `talcoord_all` for example]

- `rv_all = rv_all / cls_length;  %rv`
- `talcoord_all(j + 1, k) = mean(talcoord_all(find(~isnan(talcoord_all(1 : j, k))), k));  %average`
- `talcoord_all(j + 2, k) = std(talcoord_all(find(~isnan(talcoord_all(1 : j, k))), k));  %SD`
- `diffmap`: unknown function, keep in the script

◆ Knack for plotting dipole locations in the same axis

- The first dipole (or pair of dipoles)
  Plot dipoles on the designated axis directly
  Then, set this axis as the target (for pasting) (`target = gca;`)
- The other (pairs of) dipoles:
  Plot dipoles on ANOTHER axis (dummy figure)
  Then, find and copy the graphics objects with "`line`" properties in this axis/figure (`h2 = findobj(gcf,'Type','line');`)
  Next, paste these objects on the designated axis (`copyobj(h2, target);`)
  Finally, close the dummy figure

■ Detailed process of computing alert baseline spectra

◆ Take transposition of alert baseline spectra of each input dataset in this cluster to facilitate plotting spectra traces

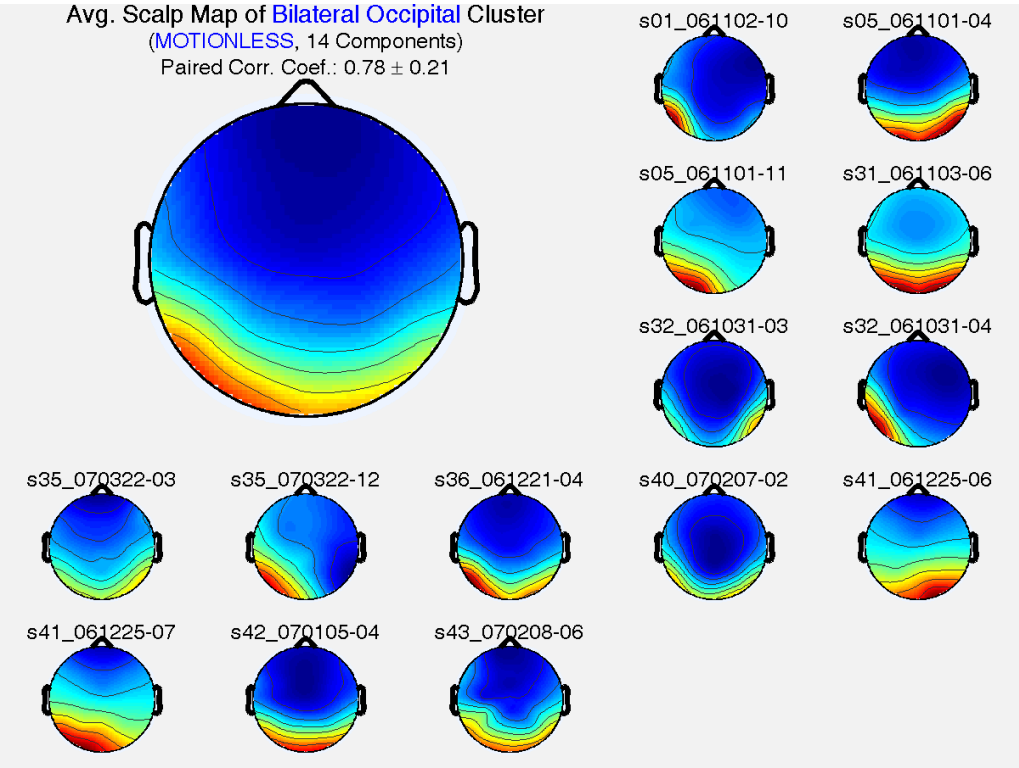
- `PB_alert_mean = PB_alert_mean(:, 1)';`

◆ Collect the spectra

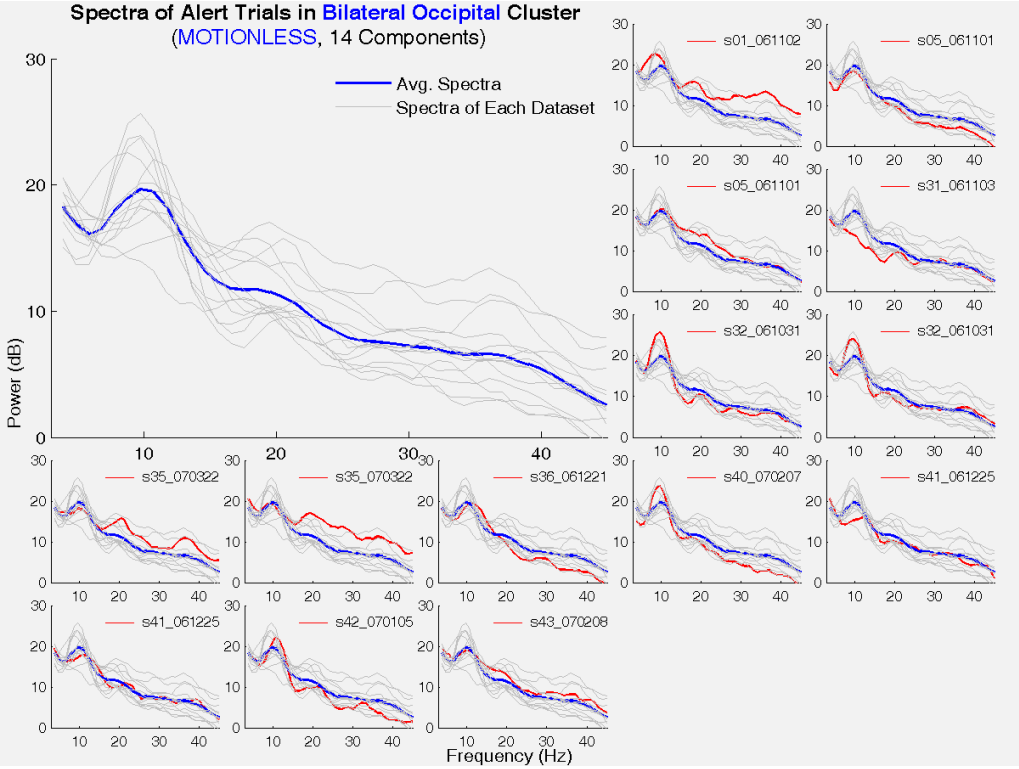
- `spec_all = [spec_all; PB_alert_mean];`

■ Outputs

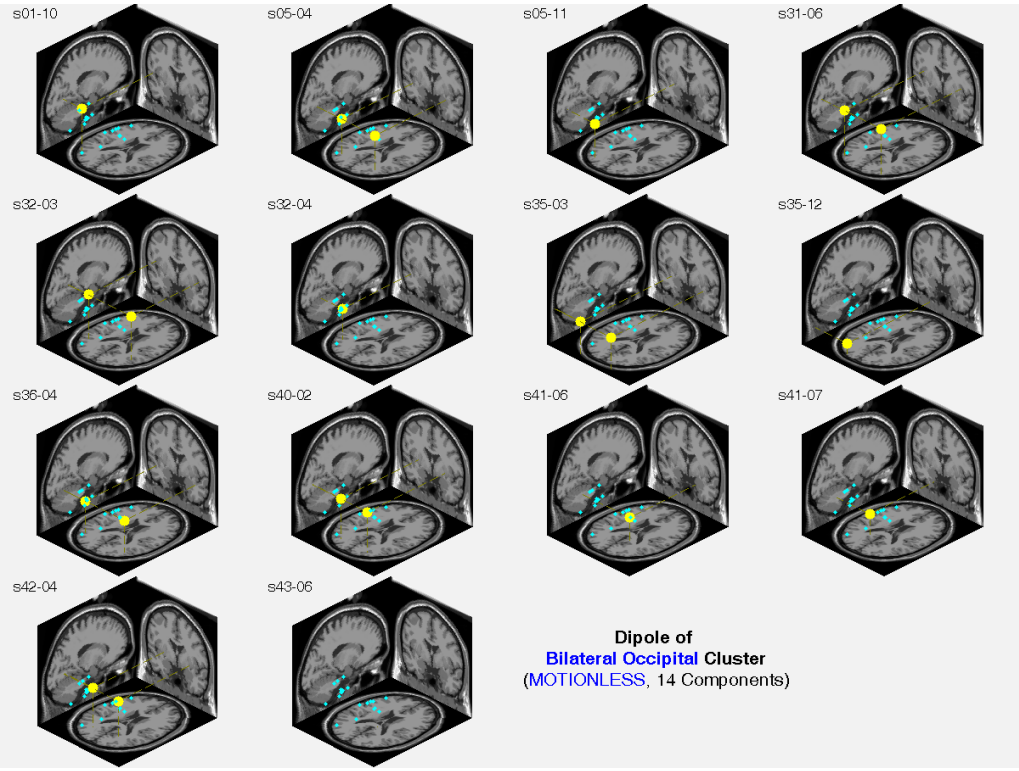
- MAT file: `ICnn_component_(cls_ver)_(MN).mat` (nn: code of this cluster)
  - `std_chanlocs`: standard channel locations
  - `chan_all`: channel locations of all datasets in this cluster
  - `icawinv`: ICA inverse weight matrix of all datasets in this cluster
  - `avg_icawinv`: average ICA inverse weight matrix of all datasets in this cluster
  - `comp_corr`: paired correlation of the components in this cluster
  - `avg_corr`: average correlation of the components in this cluster
  - `sd_corr`: SD of correlation of the components in this cluster
  - `spec_all`: alert baseline spectra from the datasets in this cluster
  - `freqs`: frequency bins
  - `dipole_all`: information of all dipoles in this cluster
  - `dipole_avg`: information of average dipoles in this cluster
- txt file: `ICnn_ClsDipole_(cls_ver)_(MN).txt`
  - Save the Talairach coordinates in this cluster in a printable file
- Figures:
  - Average scalp maps in this cluster:
    `ICnn_component_map_(cls_ver)_(MN).fig/png`



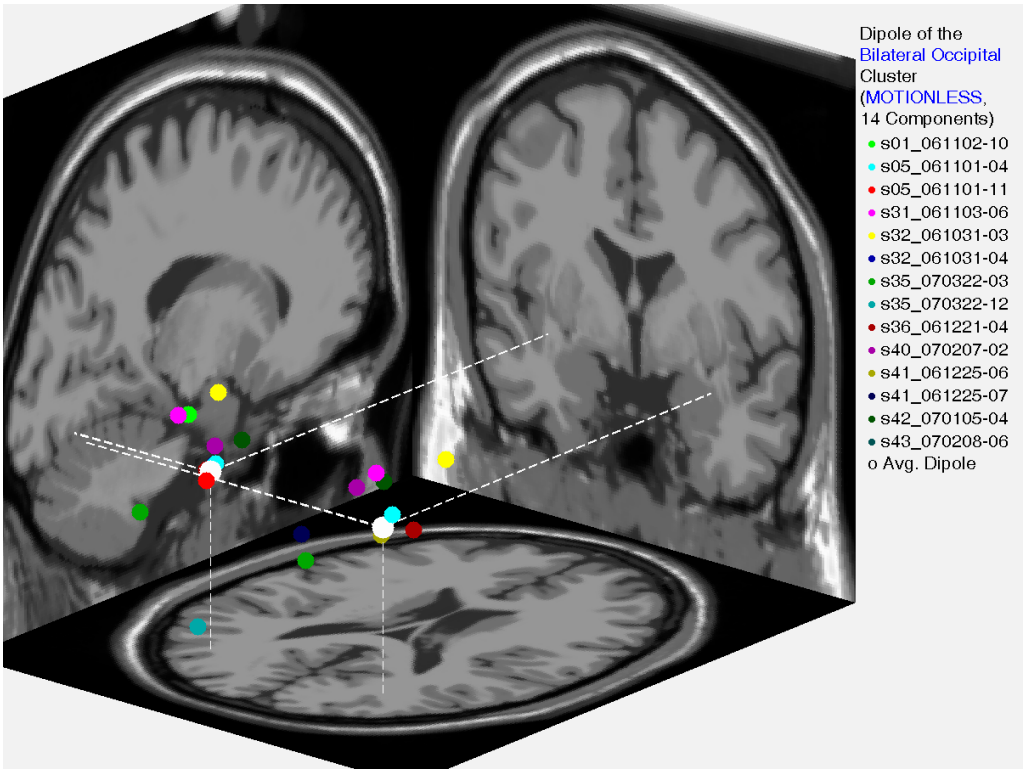
  - Alert baseline spectra in this cluster:
    `ICnn_Cluster_Spectra_(cls_ver)_(MN).fig/png`



- Dipole locations in this cluster: (in array of plots)
  `ICnn_DipoleMap_(cls_ver)_(MN).fig/png`

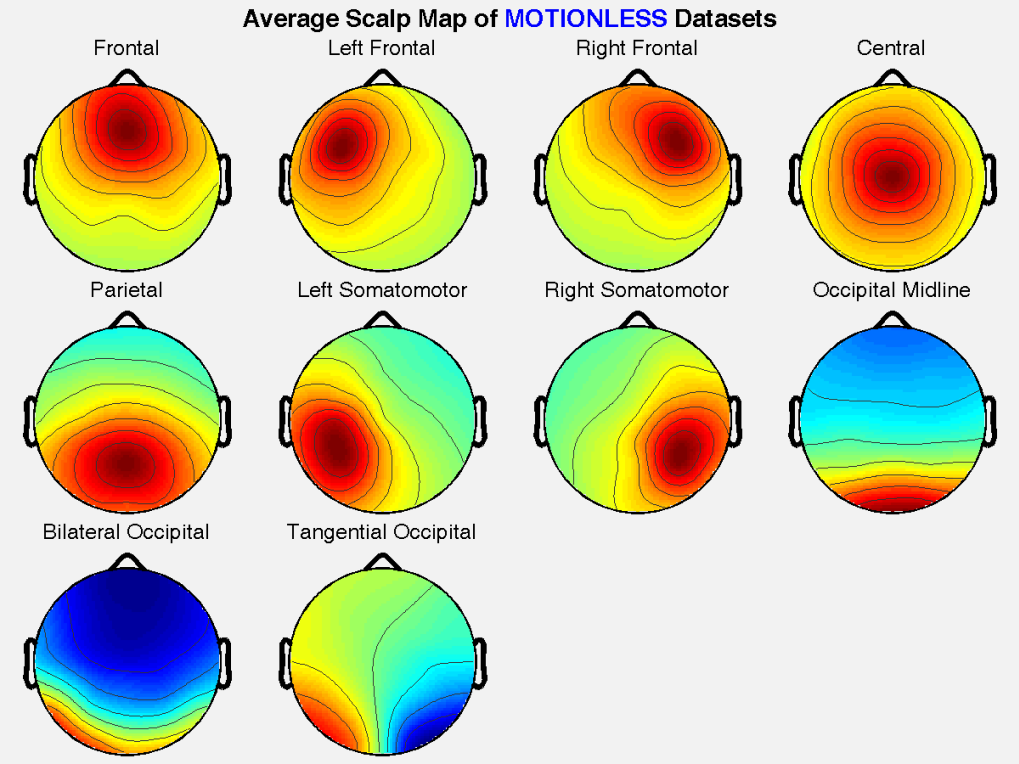


- Dipole locations in this cluster: (in a figure)
  `ICnn_DipoleMap_(cls_ver)_single_MN.fig/png`



- Knowing bugs:
  - Having errors when running this script in other accounts. Checking

- Script: `DR_PlotClustCompmap.m` (ver 200904)
  - Plot average component map in clusters
  - Read ICA inverse weight vectors and plot figures
  - Required scripts
    - `mypath.m`
    - `DR_PBase_setup.m`
  - Required inputs
    - ICA activation of the above components (in EEG dataset)
    - Components and identified clusters
  - Check before you run:
    - Variables
      - `MN`: conditions
      - `MN_COLOR`: color code used to indicate motionless or motion conditions
      - `p_val`: significance level (currently `1E-10`)
      - `rj`: `'_rj'` ➔ artifacts removed dataset
    - Directories
      - Tonic power spectra: `FilePath/MN/`
      - Other power spectra: `FilePath/epoch_type/MN/`
  - Outputs
    - Figure: `Avg_Scalp_Map_(cls_ver)_(MN).fig`. Print the `png` file in PC



- Knowing bugs: (not reported)

VII.3 Clustering power spectra

- Script: `DR_PlotClustPbase.m` (ver 200903)
  - Compute and plot power spectra for CLUSTERED data
  - The process and structure are similar to those in `RS_powerbase.m` except estimating the significance level
  - Required scripts
    - `mypath.m`
    - `DR_PBase_setup.m`
  - Required inputs
    - Components and identified clusters
    - Average ICA inverse weight vector in this cluster
    - Power spectra of the components in the plotted cluster
  - Check before you run:
    - Variables
      - `PLOT_SET`: the plotted datasets
      - `MN`: conditions. (`'motionless'` or `'motion'`)
      - `MN_COLOR`: color code used to indicate motionless or motion conditions
      - `p_val`: significance level (currently `1E-10`)
      - `epoch_type`
        `''` ➔ tonic spectra
        `'dev_on'` ➔ phasic spectra

'all' ➔ mixed spectra

- rj: '_rj' ➔ artifacts removed dataset
- SD:

  '' ➔ plot avg. of spectra

  '_SD' ➔ plot (avg. + SD)
- no_band: special purpose for thesis

  '' ➔ plot all the bands defined in DR_PBase_setup.m

  '1' ➔ plot all except the 3rd band defined in DR_PBase_setup.m
- nor: how to normalize spectra. Currently 1

◆ Directories

- Tonic power spectra: FilePath/MN/
- Other power spectra: FilePath/epoch_type/MN/

■ Detailed process of normalizing power spectra

◆ Collect alert baseline power and normalized power spectra from the datasets in this cluster

- PB_alert: padded to keep the same dimension as that of PB_n

```
PB_alert = [PB_alert tmp_set.PB_alert_mean];
PB_n = [PB_n tmp_set.PB_n];
```

- PB_alert_subj: one dataset, one spectrum

```
if ~strcmp(old_set, plot_set)
  PB_alert_subj = ...
    [PB_alert_subj tmp_set.PB_alert_mean(:, 1)];
end
```

◆ After finishing loading datasets, sort baseline power by RT

```
[RT_s ur_idx] = sort(RT_s, 2);
PB_mean = PB_mean(:, ur_idx);
PB_n = PB_n(:, ur_idx);
PB_alert = PB_alert(:, ur_idx);
subj_seq = subj_seq(:, ur_idx);
```

◆ Moving average (power spectra and RT)

- PB_mov = [PB_mov trimmean(PB_n(:, j : j + alert.trials - 1), 10, 2)];
- RT_s_mov = [RT_s_mov trimmean(RT_s(:, j : j + alert.trials - 1), 10, 2)];

■ Detailed process of applying two-sampled *t*-test (using for loop)

◆ A moving window

- PB_tmp = PB_mean(:, sig_idx(k) : sig_idx(k) + alert.trials - 1);
- subj_seq_id_tmp = subj_seq_id(:, sig_idx(k) : sig_idx(k) + alert.trials - 1);

- Compute weighted mean alert baseline power in this cluster (using `for` loop)
  - Find # of trials from each session in this cluster
    `n_of_trial = length(find(subj_seq_id_tmp == m));`
  - If there are trials in this component (session) in this cluster, compute the weighted sum
    `PB_alert_tmp = PB_alert_tmp + PB_alert_subj(:, m) * n_of_trial;`
  - Update # of subjects (if this dataset didn't appear)
    `n_of_subj_win = n_of_subj_win + 1;`
  - Compute weighted average of alert baseline spectra
    `PB_alert_tmp = PB_alert_tmp * ones(1, alert.trials) / alert.trials;`
- Estimate significance level
  - `[H, P] = ttest2(PB_tmp', PB_alert_tmp', p_val / 2 / size(freqs, 2) / n_of_subj_win, 'both');`
  - `p_val / 2 / size(freqs, 2) / n_of_subj_win`: *p* value corrected by degree of freedom
  - `H`: show if the power on this frequency bin is significant
  - `P`: *p* value on this frequency bin
- Merge into the matrix (for adding contour on power image)
  - `H_mask(:, k) = H';`
  - `P_mask(:, k) = P';`

- Outputs
  - `MAT` file:
    - Tonic spectra:
      `ICnn_PBase_(nor)_(MN).mat`
      `cls_length`: # of components in this cluster
      `session_count`: # of sessions in this cluster
      `icawinv`: averaged ICA inverse weight matrix of this cluster
      `chanlocs`: averaged channel locations of this component
      `freqs`: output frequency bins
      `RT_s`: sorted reaction time in this cluster
      `RT_s_mov`: moving averaged reaction time in this cluster
      `subj_seq`: sequence of subject (corresponding to each trial)
      `p_val`: the set *p* value (uncorrected)
      `PB_mean`: power spectra (averaged across time windows)
      `PB_alert`: averaged power spectra of alert trials
      `PB_n`: normalized power spectra
      `PB_alert_subj`: power spectra of alert trials (from the original

dataset of each subject)

`PB_mov`: moving averaged power spectra (from PB_n)

`FREQ_INC`: power spectra of the extracted frequency bands

`H_mask`: significant region of power spectra

`P_mask`: *p* value of individual frequency/trial

`IC(nn)`: datasets and components

- Other spectra:

  `ICnn_PBase_(nor)_(MN)_(epoch_type).mat`

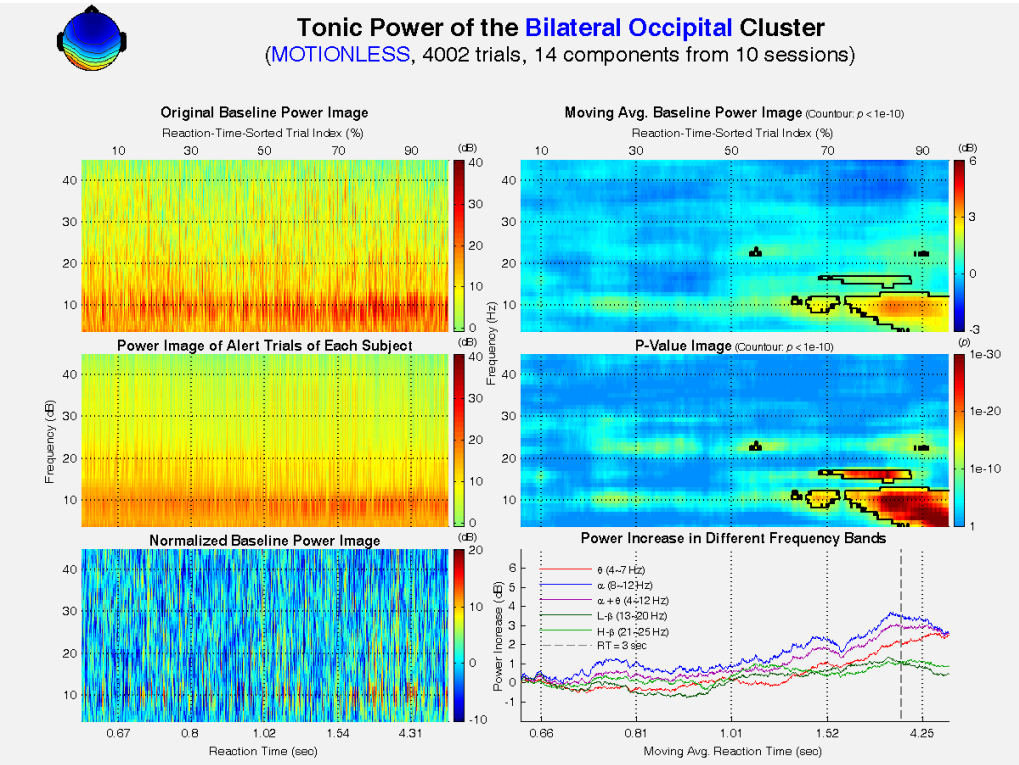
  (variables are the same as those in `ICnn_PBase_nor_MN.mat`)

◆ Figures: [if plotting tonic power spectra, this script plots in `fig` and `png` formats. (Note: print the `png` file again in PC); else, only plots in `png` format]

- Power spectra after each step:

  `ICnn_PBase_(nor)_(MN)_method.png/fig` for tonic spectra

  `ICnn_PBase_(nor)_(MN)_(epoch_type)_method.png/fig` for other types of spectra

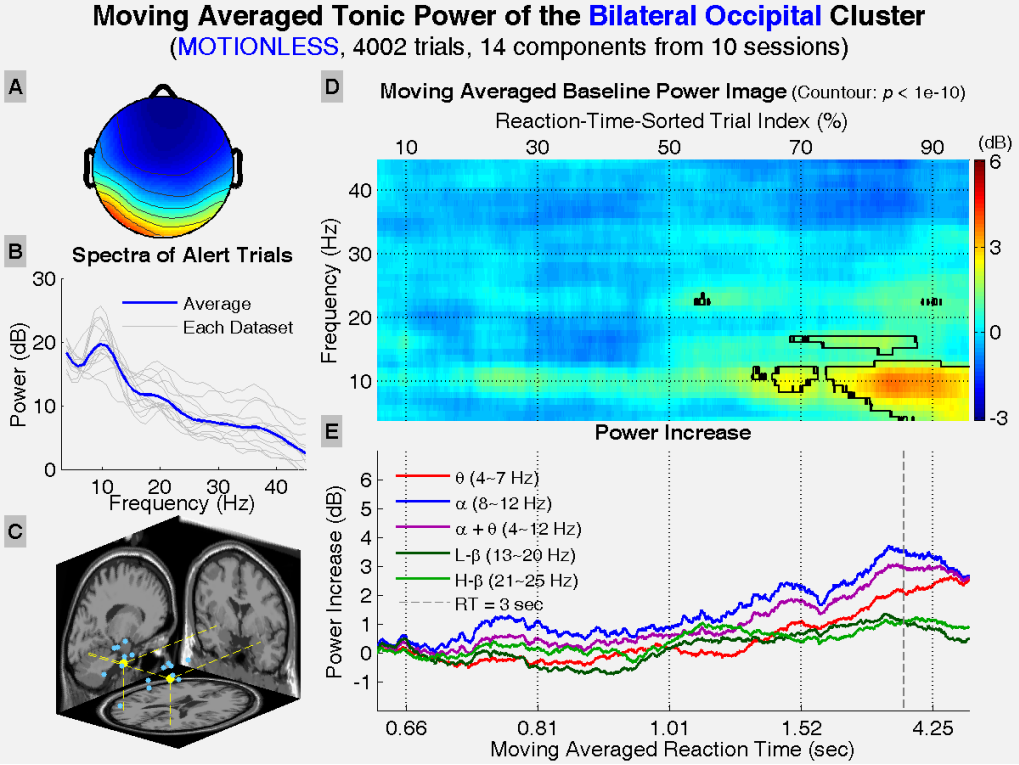


- `ICnn_PBase_(nor)_(MN)_result.png/fig` or `ICnn_PBase_(nor)_(MN)_(epoch_type)_result.png/fig`: (plotted by `DR_PlotClustPbase_merge.m`, described later)

■ Knowing bugs: (not reported)

- Script: `DR_PlotClustPbase_merge.m` (ver 200903)
  - ■ Plot power spectra for CLUSTERED data with dipole locations
  - ■ Run after finishing plotting clustered dipole locations (using script `DR_PlotClustCompPowerDipole.m`, described in Section VII.2)

- Required scripts
  - `mypath.m`
  - `DR_PBase_setup.m`
- Required inputs
  - Average ICA inverse weight vector in this cluster
  - Clustered power spectra of this cluster
  - Dipole locations and alert baseline spectra of this cluster
- Check before you run:
  - Variables
    - `PLOT_SET`: the plotted datasets
    - `MN`: conditions. (`'motionless'` or `'motion'`)
    - `MN_COLOR`: color code used to indicate motionless or motion conditions
    - `p_val`: significance level (currently `1E-10`)
    - `epoch_type`: types of power spectra
      `''` ➔ tonic spectra
      `'dev_on'` ➔ phasic spectra
      `'all'` ➔ mixed spectra
    - `rj`: `'_rj'` ➔ artifacts removed dataset
    - `SD`: `''` ➔ plot avg. of spectra; `'_SD'` ➔ plot (avg. + SD)
    - `no_band`: special purpose for thesis
      `''` ➔ plot all the bands defined in `DR_PBase_setup.m`
      `'1'` ➔ plot all except the 3rd band defined in `DR_PBase_setup.m`
    - `nor`: how to normalize spectra. Currently `1`
  - Directories
    - Tonic power spectra: `FilePath/MN/`
    - Other power spectra: `FilePath/epoch_type/MN/`
- Outputs
  - Figure: [if plotting tonic power spectra, this script plots in `fig` and `png` files (Note: print the `png` file again in PC); else, only plots in `png` format]
    - Power spectra for the results
      `ICnn_PBase_(nor)_(MN)_method.png/fig` for tonic spectra [and then print the `png` file in PC], or
      `ICnn_PBase_(nor)_(MN)_(epoch_type)_result.png/fig` for other types of spectra

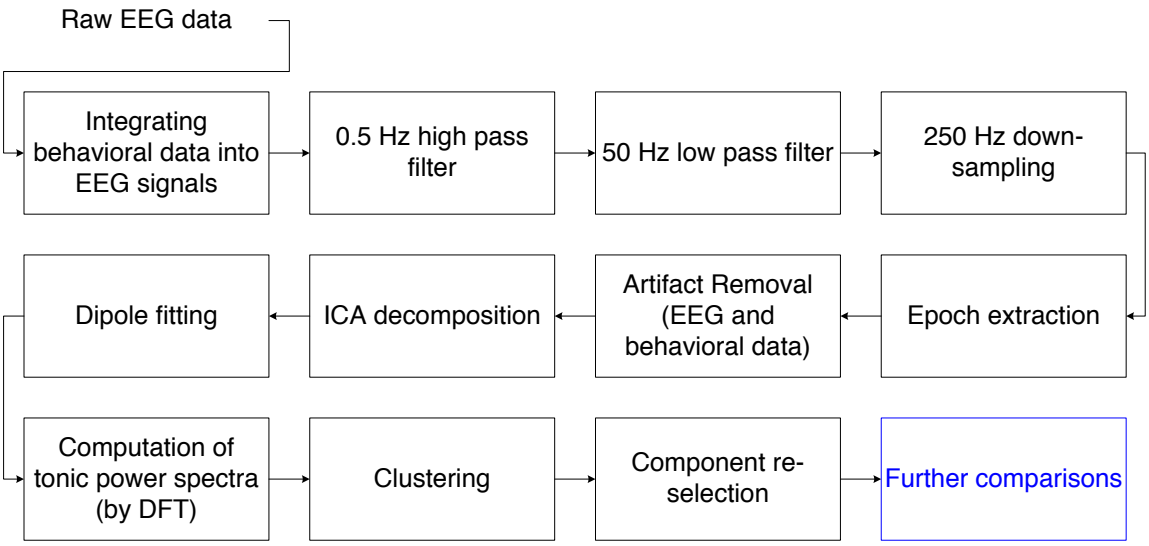


- Knowing bugs: (not reported)

VII.4 Component re-selection

- Criteria
  - ICA scalp maps in this cluster
  - Alert baseline spectra in this cluster
  - Dipole locations of independent components in this cluster (not necessary)
- Modify the cluster # negative in `DR_PBase_setup.m` (e.g.: 5 ➔ -5)
- Re-evaluate clustered RT, power spectra, alert baseline spectra, average scalp map, and dipole locations

VIII. Further comparisons



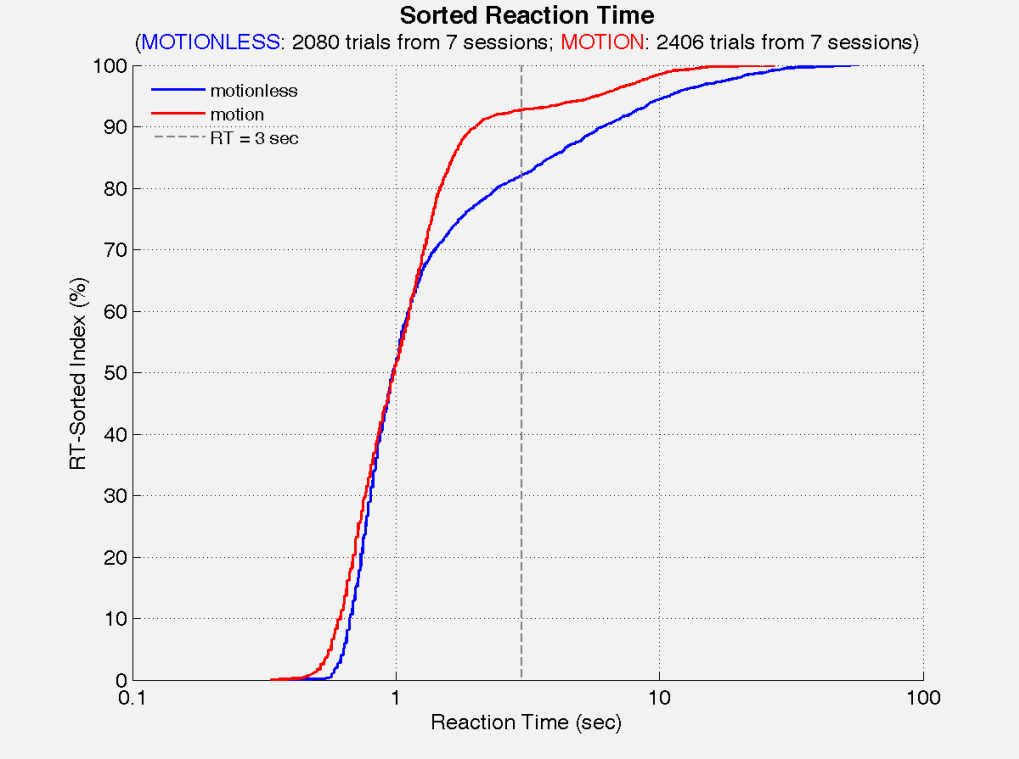
- Further comparisons in EEG (compare different conditions, etc.)
- The logic of scripts in this part mostly base on that of the scripts in the previous part, or just read the `MAT` files generated in the previous section and plot figures

VIII.1 Compare clustered RTs between motionless and motion conditions

- Script: `DR_PlotClustRT_motionless_motion.m` (ver 200903)
  - Compare CLUSTERED RTs under motionless and motion condition
  - Required scripts
    - `mypath.m`
    - `DR_PBase_setup.m`
  - Required inputs
    - RTs of trials after artifact removal
  - Check before you run:
    - Variables
      - `PLOT_SET`: the plotted datasets
      - `MN`: conditions (currently `MN = {'motionless', 'motion'};`)
      - `rj`: `'_rj'` ➔ artifacts removed dataset
      - `p_val`: significance level (currently `1E-4`; however, this is meaningless due to the appropriate method for estimating the significance level is yet to be found)
    - Directories: `FilePath/MN/`
  - Outputs
    - `MAT` file: `RT_Sorted_Trial_motionless+motion.mat`
      - `PLOT_SET1`: list of motionless datasets
      - `PLOT_SET2`: list of motion datasets
      - `urRT_s1`: sorted clustered RTs of motionless datasets (before artifact removal)

- `urRT_s2`: sorted clustered RTs of motion datasets (before artifact removal)
- `RT_s1`: sorted clustered RTs of motionless datasets (after artifact removal)
- `RT_s2`: sorted clustered RTs of motion datasets (after artifact removal)
- `RT_s`: clustered sorted reaction time (after artifacts removal)
- `RT_s_mov`: moving averaged reaction time in this cluster
- `MN`: motionless or motion condition
- `p_val`: the set *p* value (currently `1E-4`) [meaningless]
- `norm_distr`: check if the clustered datasets are with normal distribution (the 1st element: motionless datasets; 2nd: motion datasets)
- `stat_diff`: `struct` for storing actual *p* value and flags to show if two conditions are statistically significant [meaningless]
  `H`: flags to show if two conditions are statistically significant
  `P`: actual *p* value

◆ Figures: `RT_Sorted_Trial_motionless+motion.fig`. Print the `png` file in PC



VIII.2 Compare power spectra across motionless and motion conditions

- Script: `DR_PlotClustPbase_motionless_motion.m` (ver 200904)
  - Plot power spectra for CLUSTERED data across motionless and motion conditions

- The logic is similar to that in `DR_PlotClustPbase.m`
- Since some of the subjects may participated only motionless or motion session, it is necessary to exclude such datasets and re-cluster the data
- Required scripts
  - `mypath.m`
  - `DR_PBase_setup.m`
- Required inputs
  - Components and identified clusters
  - Average ICA inverse weight vector in this cluster [WRONG!! Since subjects may be different, it is necessary to re-compute average ICA inverse weight matrix]
  - Power spectra of the components in the plotted cluster
- Check before you run:
  - Variables
    - `PLOT_SET`: the plotted datasets
    - `MN`: conditions (currently `MN = {'motionless'}, {'motion'};`)
    - `MN_COLOR`: color code used to indicate motionless or motion conditions
    - `p_val`: significance level (currently `1E-10`)
    - `epoch_type`: types of power spectra
      `''` ➔ tonic spectra
      `'dev_on'` ➔ phasic spectra
      `'all'` ➔ mixed spectra
    - `rj`: `'_rj'` ➔ artifacts removed dataset
    - `SD`: `''` ➔ plot avg. of spectra; `'_SD'` ➔ plot (avg. + SD)
    - `no_band`: special purpose for thesis
      `''` ➔ plot all the bands defined in `DR_PBase_setup.m`
      `'1'` ➔ plot all except the 3rd band defined in `DR_PBase_setup.m`
    - `nor`: how to normalize spectra. Currently `1`
  - Directories
    - Tonic power spectra: `FilePath/MN/`
    - Other power spectra: `FilePath/epoch_type/MN/`
- Outputs
  - `MAT` file:
    - Tonic spectra: (nn: code of this cluster)
      `ICnn_PBase_nor_motionless+motion.mat`
      `motionless`: `struct` for storing necessary variables (motionless)
        `IC(nn)`: datasets and components
        `session_count`: # of sessions in this cluster
        `cls_length`: # of components in this cluster

`icawinv`: averaged ICA inverse weight matrix of this cluster

`chanlocs`: averaged channel locations of this cluster

`freqs`: output frequency bins

`RT_s`: sorted reaction time in this cluster

`RT_s_mov`: moving averaged reaction time in this cluster

`RT_s_overlapidx`: indices of trials with ovrlapped RT

`RT_s_mov_overlapidx`: indices of trials with overlapped moving averaged RT

`subj_seq`: sequence of subject (corresponding to each trial)

`PB_mean`: power spectra (averaged across time windows)

`PB_alert`: averaged power spectra of alert trials

`PB_alert_subj`: power spectra of alert trials (from the original dataset of each subject)

`PB_n`: normalized power spectra

`PB_mov`: moving averaged power spectra (from PB_n)

`FREQ_INC`: power spectra of the extracted frequency bands

`H_mask`: significant region of power spectra

`P_mask`: *p* value of individual frequency/trial

`motion`: (the same function as that in the above one)

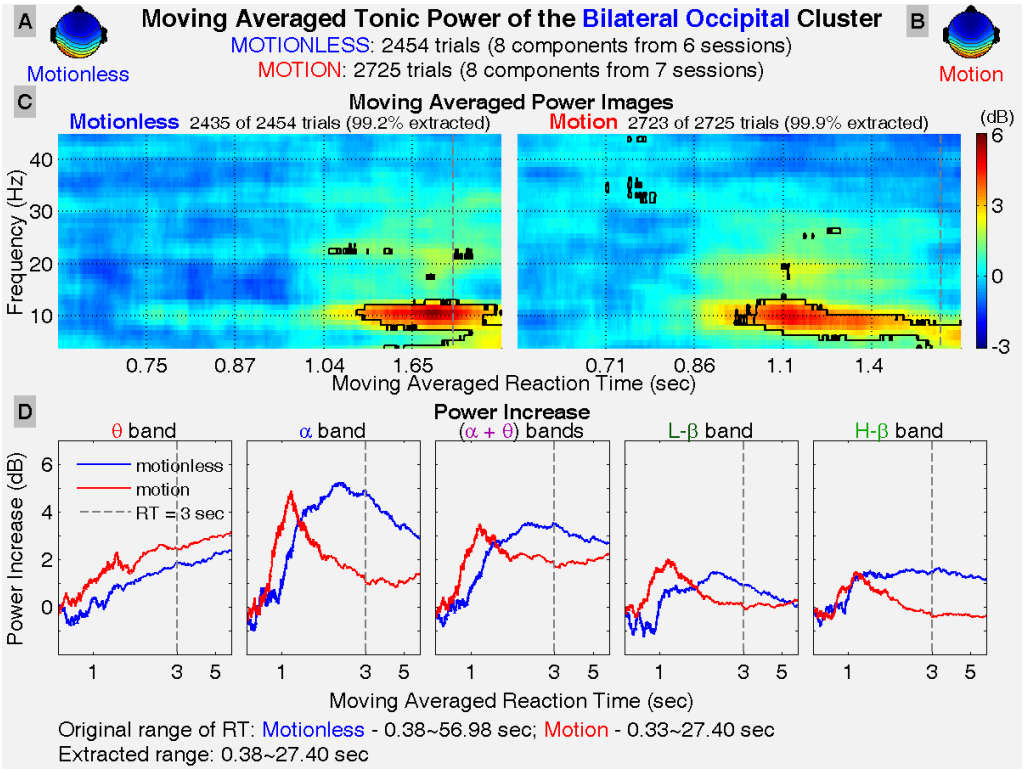
`RT_s_overlap`: overlapped RT in both conditions

`RT_s_mov_overlap`: moving averaged overlapped RT in both conditions

- Other spectra:
  `ICnn_PBase_(nor)_motionless+motion_(epoch_type).mat`
  (variables are the same as those in `ICnn_PBase_nor_MN.mat`)

◆ Figures: if plotting tonic power spectra, this script plots in `fig` and `png` files [Note: print the `png` file again in PC]; else, only plots in `png` format

- Power spectra after each step:
  `ICnn_PBase_(nor)_motionless+motion.png/fig` (tonic),
  `ICnn_PBase_(nor)_motionless+motion_(epoch_type)_method.png/fig` (other)



- Knowing bugs: (not reported)

VIII.3 Compare tonic and mixed power spectra

- Script: `DR_PlotClustPBase_compare.m` (ver 200905)
  - Compare power spectra between Two of the states: tonic, mixed, and phasic
  - Read the power spectra and plot figures
  - Required scripts
    - `mypath.m`
    - `DR_PBase_setup.m`
  - Required inputs
    - Components and identified clusters
    - Average ICA inverse weight vector in this cluster
    - Power spectra of the plotted cluster
  - Check before you run:
    - Variables
      - `PLOT_SET`: the plotted datasets
      - `MN`: conditions
      - `MN_COLOR`: color code used to indicate motionless or motion conditions
      - `p_val`: significance level (currently `1E-10`)
      - `EPOCH_TYPE`: types of power spectra (2 of the followings; currently `EPOCH_TYPE = {'', 'all'};`)
        `''` ➔ tonic spectra
        `'dev_on'` ➔ phasic spectra

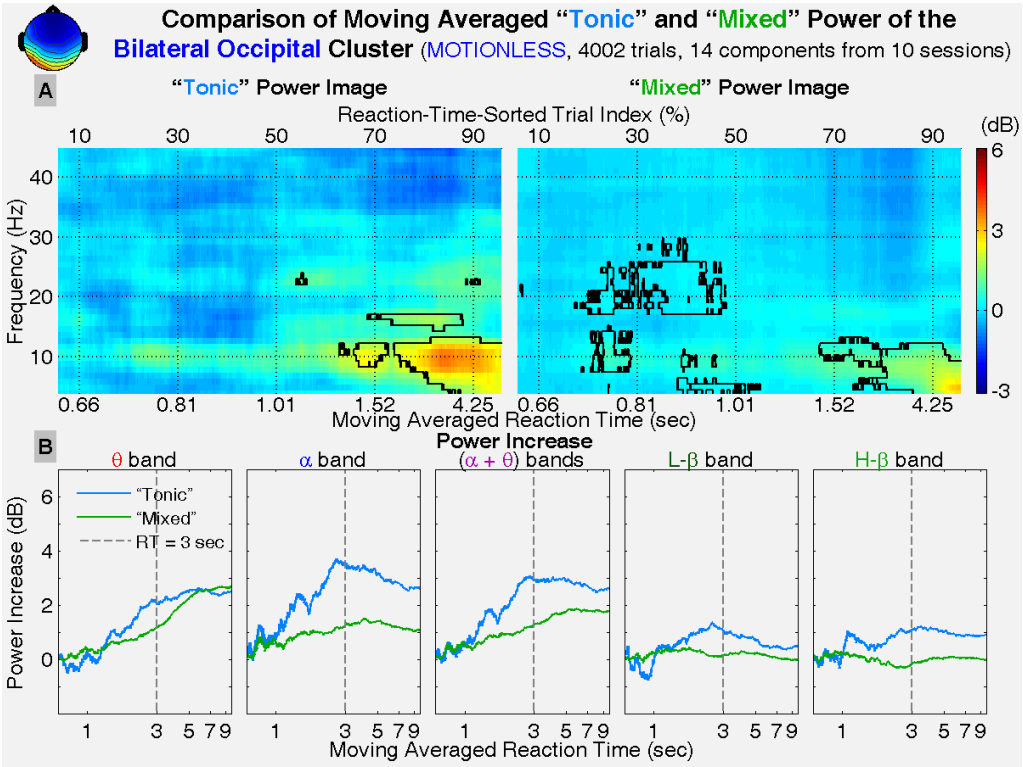
'all' ➔ mixed spectra

- EPOCH_COLOR: color code used to indicate different types of spectra
- rj: '_rj' ➔ artifacts removed dataset
- SD: '' ➔ plot avg. of spectra; '_SD' ➔ plot (avg. + SD)
- no_band: special purpose for thesis
  '' ➔ plot all the bands defined in DR_PBase_setup.m
  '1' ➔ plot all except the 3rd band defined in DR_PBase_setup.m
- nor: how to normalize spectra. Currently 1

◆ Directories
- Tonic power spectra: FilePath/MN/
- Other power spectra: FilePath/epoch_type/MN/

■ Outputs

◆ Figure:
ICnn_compare_(epoch_type1)_(epoch_type2)_(MN)(nor)(rj)(SD)(noband).fig. Print the png file in PC



■ Knowing bugs: (not reported)

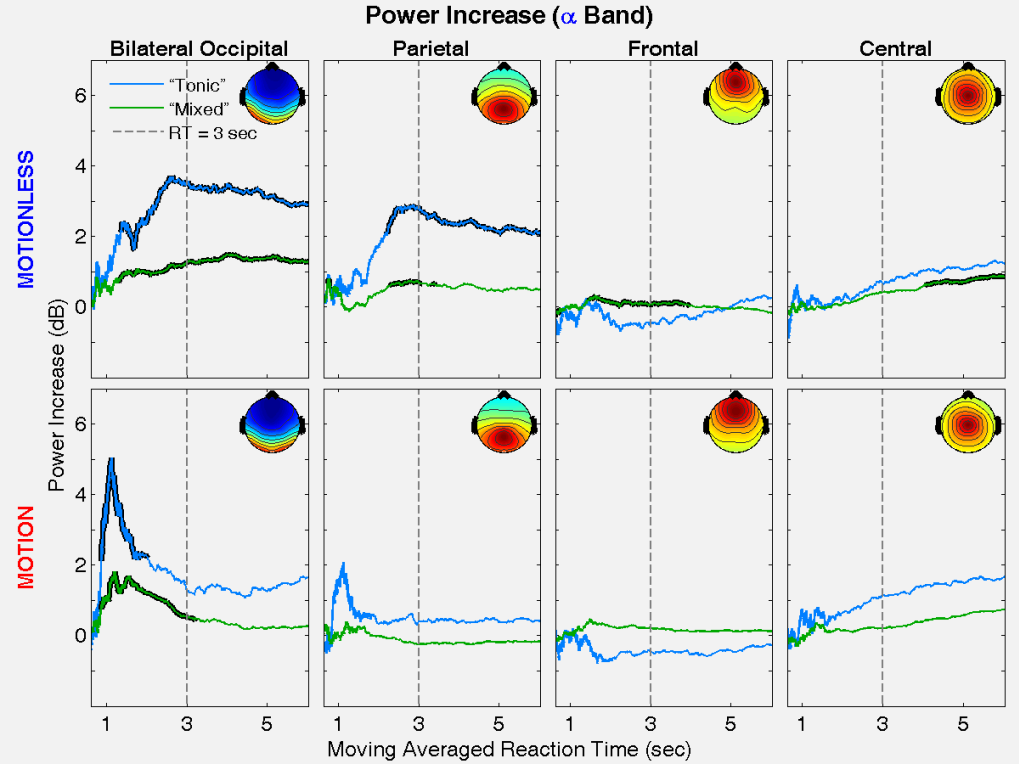
- Script: DR_PlotClustPBase_powinc.m (ver 200906) [only use in defense slide]

■ Compare power increase between

◆ Two of the following states: tonic, mixed, and phasic,
◆ Motionless and Motion conditions, and
◆ Different clusters

■ Plot power on a frequency band in a figure

- Read the power spectra and plot figures
- Required scripts
  - `mypath.m`
  - `DR_PBase_setup.m`
- Required inputs
  - Components and identified clusters
  - Average ICA inverse weight vector in this cluster
  - Power spectra of the plotted cluster
- Check before you run:
  - Variables
    - `PLOT_SET`: the plotted datasets
    - `MN`: conditions. (currently `MN = {'motionless', 'motion'};`)
    - `p_val`: significance level (currently `1E-10`)
    - `EPOCH_TYPE`: types of power spectra (2 of the followings; currently `EPOCH_TYPE = {'', 'all'};`)
      `''` ➔ tonic spectra
      `'dev_on'` ➔ phasic spectra
      `'all'` ➔ mixed spectra
    - `rj`: `'_rj'` ➔ artifacts removed dataset
    - `SD`: `''` ➔ plot avg. of spectra; `'_SD'` ➔ plot (avg. + SD)
    - `no_band`: special purpose for thesis (not important in this script since each figure only has power in a frequency band)
      `''` ➔ plot all the bands defined in `DR_PBase_setup.m`
      `'1'` ➔ plot all except the 3rd band defined in `DR_PBase_setup.m`
    - `nor`: how to normalize spectra. Currently `1`
    - `plot_clust`: the cluster (defined in `DR_PBase_setup.m`) to be potted (in matrix form; sequence matter)
  - Directories: `FilePath/`
- Outputs
  - Figure:
    `ICs(plot_clust)_(powinc)_(freq_name)(nor)(rj)(SD)`
    `.fig`. Print the `png` file in PC
    - Note: `plot_clust` is hexadecimal


Power Increase (α Band)
Bilateral Occipital
Parietal
Frontal
Central
"Tonic"
"Mixed"
RT = 3 sec
MOTIONLESS
MOTION
Power Increase (dB)
Moving Averaged Reaction Time (sec)