Sayeed’s Preprocessing Pipeline.

# Section 1: Purpose

This document will help to explain the usage of the pipeline. The pipeline itself is a wrapper of two scripts, one which performs standard preprocessing functions on the data and saves the data, and the other which loads that saved data. The code in the wrapper serves to allow the user to change the crucial settings and perform preprocessing in different ways.

The purpose of designing an analysis script in this way is to accomplish two main goals. First, because most of the unchanging code is located inside the preprocessing script, it becomes much easier to see which settings have been used every time. Secondly, it makes explicit the order of processing steps, so that any user will be able to quickly understand what steps are being used on the data and in what order.

The hope of designing this script In this way is that it will help to reduce errors, and will allow a more efficient workflow. Data, once processed, is saved in a way that allows multiple datasets with different settings to coexist. Before running the preprocessing, the settings chosen are saved to the folder containing the pipeline. This settings file is named according to the name given to the dataset in the **exp.settings** variable. The data is also saved with this name in the title. In this way, you can process multiple sets of data overnight, and have all of them ready to be loaded when you’re ready to work on them.

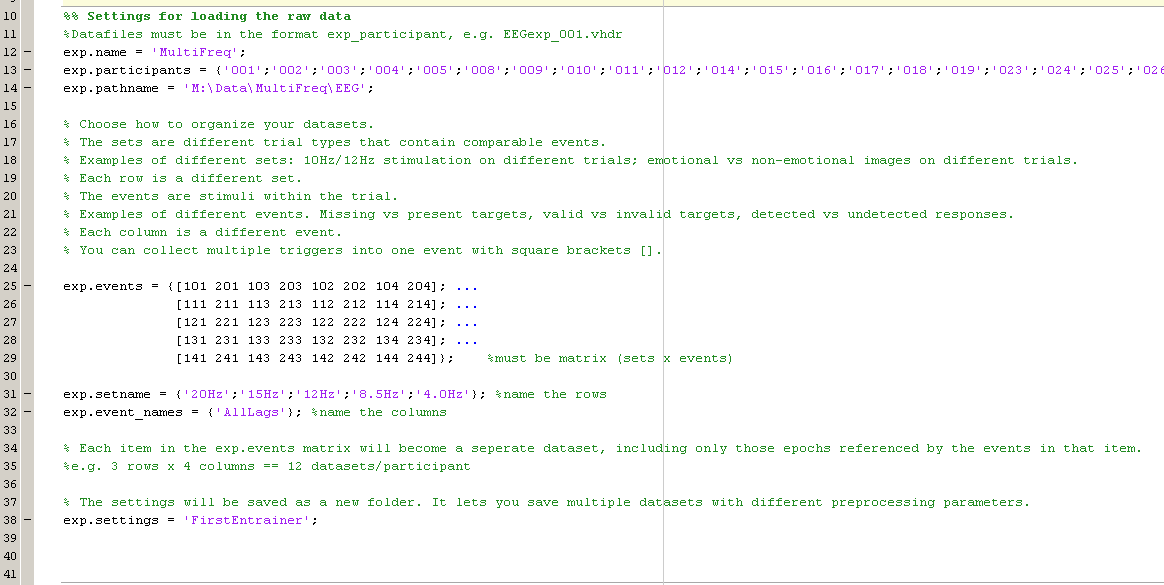
# Section 2: Usage of the pipeline

To open the settings file, type “edit Pipeline” into the command line.

This file is designed to present a clear set of settings which correspond to the preprocessing requirements of standard EEG experiments. In order, the steps in the pipeline are:

1. Load the data
2. Filter the data
3. Re-referencing the data
4. Epoching the data
5. Artifact Rejection
6. Blink Correction
7. Computing the time-frequency representation

Each section of the pipeline contains important choices to be made.

Step 1: Settings for loading and organizing data

To load your data, the script expects a certain filename in the format name\_number. For example, my experiment is called “MultiFreq”, so my data files for each participant are called “MultiFreq\_001”, ”MultiFreq\_002”, ” MultiFreq\_003”, etc. The script adds in the ‘.vhdr. extension.

The variable **ext.participants** must be a cell array of strings. After entering in the name and participant numbers, you need to enter in **exp.pathname**  the data files are located in.

The next step is to choose the organization of the processed datasets. This is the most complicated part of the pipeline, but will be easy to use once you understand the format.

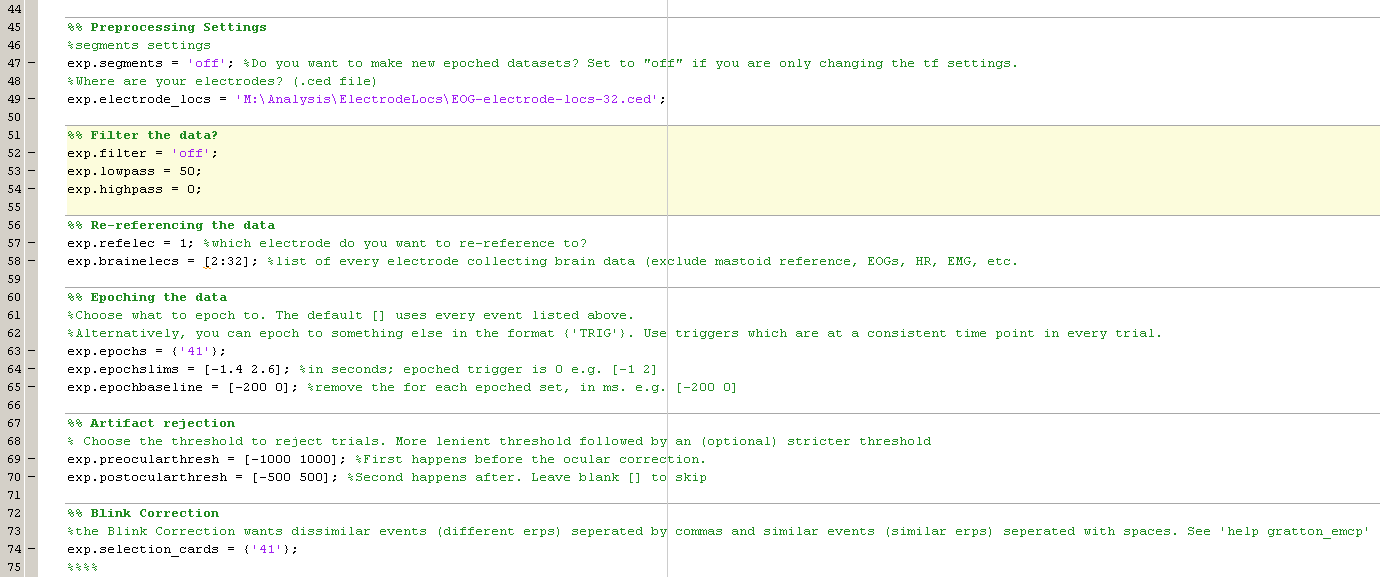
The script allows you to organize your processed data in terms of condition and trial type, as determined by different groupings of triggers. In the variable **exp.events**, you need to create a matrix of triggers, where the different conditions (called sets) are separated by rows, and the different trial types (called events) are separated by columns.

The sets and events represent two levels of nesting that you have available to you in this pipeline. To be more explicit, if you have different conditions you want to compare against each other in your experiment should be entered in as sets. However, if in addition to that you have different types of trials within a condition, then you have the option of one more level of nesting, which are called the events.

The script that segments your data will take each element of this matrix (either one trigger or an array of triggers) and create one EEGLAB dataset that contains only the epochs which contain at least one of the triggers in that element. In this method of organization, all of the information needed to separate your conditions and trial types must be contained in one trigger. For example, you cannot use triggers 10 and 20 at the start of your trial to indicate your conditions, and then the triggers 1 2 3 4 to indicate the different types of targets. You can, however, use triggers 11, 12, 13, 14, 21, 22, 23, 24 to indicate both the condition and trial type.

In the example below, I have five conditions where I was presenting entrainers at five different frequencies. The triggers in my experiment contain the frequency information at the tens place, so the triggers in each row (101, 102, 201…) correspond to one condition. The triggers also contain information about target location, as well as target SOA, but I’m not using that right now, so all of those are grouped together with square brackets.You can name each set and event in the variables **exp.setname** and **exp.event\_names**. These variables must be cell arrays of strings.

Lastly, you should give the set of settings in this script a particular name, so that you can create multiple fully processed datasets which you can use for different types of analysis

Step 2: Preprocessing Settings.

The next group of settings deals with the cleaning and segmenting of the raw EEG data. Short descriptions of each variable follow.

**exp.segments** – this variable determines whether to create new segments or not. The purpose of this variable is to allow you to make changes to the time-frequency settings without having to make new settings. If this is your first time making segments with the settings to follow, then set this to ‘on’

**exp.electrode\_locs** – this is the path for the electrode location file. It can be created from the EEGLAB gui. Look up EEGLAB .ced file.

**exp.filter** – this chooses whether or not to filter the data. Usually you will only use this to create ERPs

**exp.lowpass** – frequencies higher than this number will be filtered out

**exp.highpass** – frequencies lower than this number will be filtered out

**exp.refelec -** if you have a secondary electrode to rereference to, indicate which electrode it is.

**exp.brainelecs** – you may have non-brain electrodes, such as mastoid electrodes, EOG electrodes, heart rate monitors, etc. Indicate which electrodes correspond to brain electrodes.

**exp.epochs** – this is the trigger that each epoch will lock to, such that 0ms is the latency of the trigger. If you leave this blank by typing [] then it will use all the triggers in exp.events. In the settings below, I am separating the trials based on the information stored in the target triggers (101, 102…) but I am epoching to trigger 41, which is at a consistent time point in every trial.

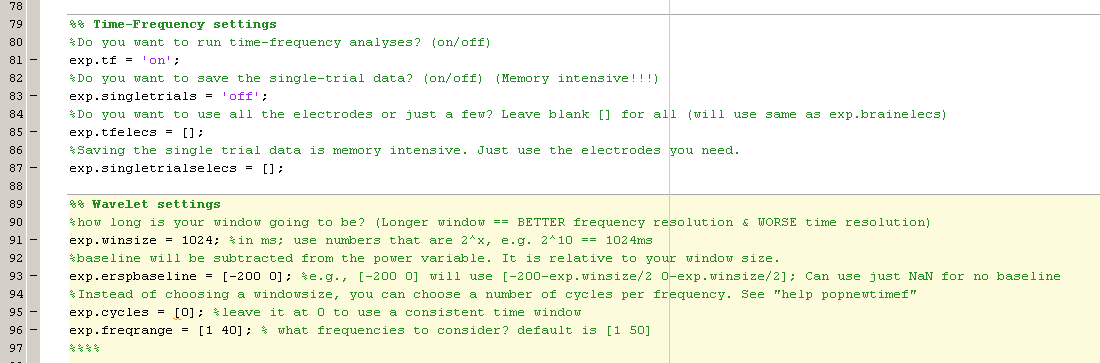
**exp.epochlims** – These are the time limits of the epoch, in seconds before and after the time-locked trigger. This should encompass your entire trial, and not overlap with other trials.

**exp.epochbaseline** – this is the baseline to take and subtract from the epoch data, usually a few hundred ms before the trial starts

**exp.preocularthresh** – This is the range in uV used to reject trials, e.g. if a trial has data that exceeds one of the two limits then it is removed. The first threshold is a more generous threshold (e.g. [-1000 1000])

**exp.postocularthresh** – The second threshold occurs after the blink correction, and should be more strict (e.g. [-500 500])

**exp.selectioncards** – This is used by the eye-movement correction procedure (emcp\_gratton.m) which we use to remove blinks from the data. You want to include all the events that you will be using to separate trials, but separate them on the basis of how the ERP is expected to look. Events that should have similar, if slightly larger or smaller, ERPs should be included in the same string, separated by a space. Events that represent significantly different ERPs should be in a different string, and separated with a comma. In the example below, I’m epoching to the a consisting trigger in every trial, so I will just use that value.

Step 3: Time-frequency Settings

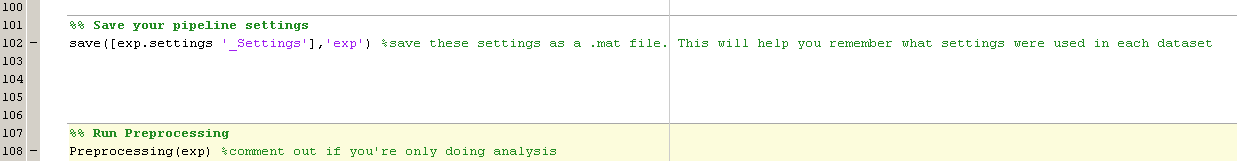
Time-frequency analyses are done using the newtimef function in EEGLAB, and there are two main settings to choose from.

**exp.tf** – This selects whether or not to do time-frequency processing at all. For example, if you just want an ERP, then it’s best to set this to off. This must be set to ‘on’ for any oscillation analyses to take place.

**exp.singletrials** – the output of newtimef consist of three variables, ersp, itc, and all\_ersp. The variables ersp and itc take an average across trials, and these two variables will be outputted if **exp.tf** is set to ‘on’. The all\_ersp variable outputs complex values for every trial in the segment, in which the real component corresponds to the power for that trial and the imaginary component corresponds to the phase information. This all\_ersp variable is very large, easily reaching hundreds of Gb across many subjects, and very time consuming to save and load. For this reason we suggest only using this variable if needed, and setting this variable to ‘off’ if it isn’t needed.

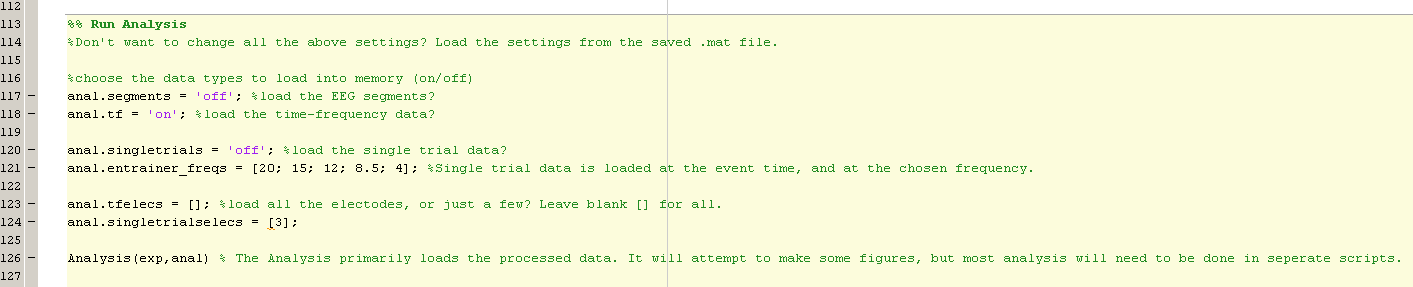
**exp.tfelecs –** this variable selects a subset of electrodes to use when generating the time-frequency representation. Restricting the number of electrodes used will save time and storage space. Alternatively, set this variable to [] to use all the electrodes.

**exp.tfelecs –** this variable selects a subset of electrodes to use when generating the single-trial time-frequency representation. It is very strongly recommended to restrict the number of electrodes to only those that will be used in the final analysis. Alternatively, set this variable to [] to use all the electrodes. (will create extremely large datasets)

Step 4: Saving your Settings and processing the data

The last function performed by the script is to save your above settings to a .mat file so that you can remember which settings were used, and easily load the data later on. This is done by saving the structure “exp” to the workspace in which this script resides. The structure “exp” will also be present when you load the data to make figures.

The line “Preprocessing(exp)” contains all the code to implement the preprocessing steps above.

Step 5: Loading the data and performing preliminary Analysis

The Analysis function has the primary purpose of loading the data, to be used in the creation of different figures. For this reason, you may load a dataset only for the purpose of using the time-frequency data, for example, without the need for the EEGLAB datasets used to make erps.

**anal.segments** – this variable will load the EEGLAB segments data, needed to make ERPS

**anal.tf** – this variable will load the time frequency data, needed to make spectras and scalp maps

**anal.singletrials** – this variable will load the single trial time-frequency, as well as the EEGLAB segments data, which is needed to find event latencies.

**anal.entrainer\_freqs** - If single trial information is being loaded, the full variable will often be so large that Matlab will run out of memory. For this reason, we only load the single trial information for a subset of time and frequency points. The time points are taken from the latencies of the events in **exp.events.** The variable **anal.entrainer\_freqs** is used to indicate which frequency should be used. Either one frequency can be selected for the whole experiment, or a different frequency can be chosen for each set and event in **exp.events**.

It may be useful to do the time-frequency processing for multiple electrodes, but only load one or two at a time. The variables **anal.tfelecs** and **anal.singletrialselecs** will indicate which electrodes to load from memory.

Once these settings are set, then all that is needed is to load the settings.mat file which will be saved after running the preprocessing. This will place the **exp** structure variable which will contain the name and location of the data to load.

Step 6: Plotting the data

This Pipeline creates a clear and simple way to process and save data, as well as load that data into the workspace.

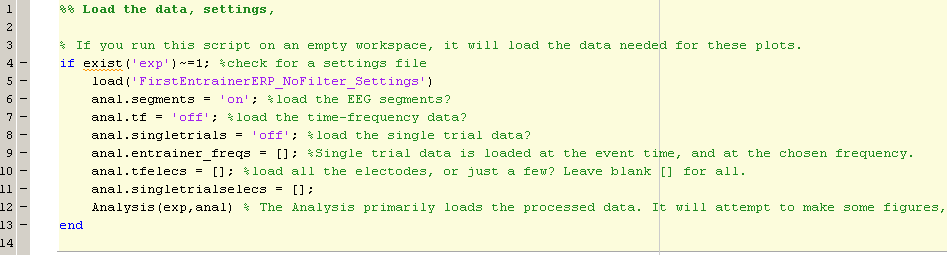
To make figures, the recommended process is to create a script that makes one figure or set of figures only, which loads it’s data based on a given settings file. In this manner, you can ensure that the figures you make will always be using the same data, processed in the same way, and you should be able to make identical figures easily.

The following example will create an ERP. It would have to be changed for the individual project, but once made to create a specific plot, it should be able to make that exact plot regardless of any future progress in analysis. The only exception to this is if the settings file is overwritten with different data. For this reason, we strongly recommend never using the same settings name for different analysis settings, only for adding additional subjects, or processing additional electrodes.

The first section of any figure script should be this section, which for the most part is identical to the settings used in pipeline.m to load the data.

The first step for this block of code is to check if the variable “exp” exists in the bae workspace. If it does, then it will load the data anew. If it does not, then it will simply move on to the plotting functions. The reason for this is you may have multiple figure scripts using the same data, and you wouldn’t want to load the data anew everytime. Alternatively, if you have been working on a different set of data, it is easy to switch to the data needed to make a given figure by simply clearing the base workspace and pressing run. The second line loads the settings file. I recommend that this settings file be hard-coded into the script as it is now. This will ensure that only the data indexed by this particular settings file (since the name of the settings file is included in the name of the data file)

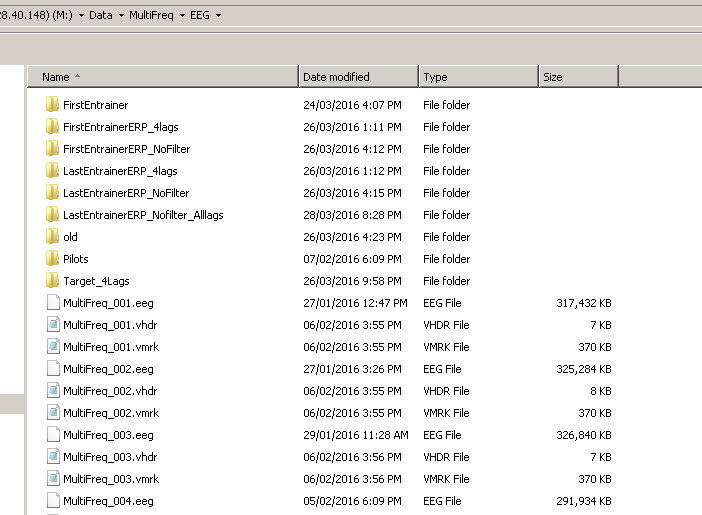
Lastly, you can choose whether to load more or less electrodes. In this case, I am just making an ERP, which will always load all the electrodes. The settings for time-frequency and single trial time-frequency are set to ‘off’, since there is no code in this section to use that data.

The section on the next page contains the code to actually create the ERP. It consists of two main sections, one to get the data from the EEGLAB segments, and the second to plot the data, including multiple additions of lines, titles, and axes limits. The code to make each figure will be different, and the prerequisite knowledge of MATLAB required to understand the code below will be needed for each user, in order to make new figures. However, the code below does demonstrate how parsimonious the creation of one figure can become using this format, only requiring the user to deal with a single page’s worth of code.

# 

# Section 3: File structure and saving the data

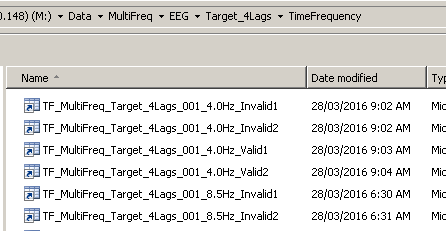
The structure used to save the datafiles will now be briefly explained. As mentioned earlier, the pipeline expects a particular location and format for the raw data files. The way this looks for our lab is shown below.

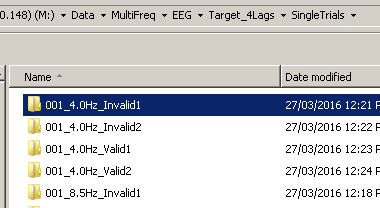


Once the preprocessing has been completed, it will create a folder in that same location with the name of the Settings file. This will allow you to see at the first level how many kinds of data processing you have. We will now step into the Target\_4Lags folder, because it has data for all of the processed datatypes: EEGLAB segments, time-frequency data, and single-trial time-frequency data.



The next level of analysis is the three datatypes. For the TimeFrequency and Segments, inside these folders are the raw datafiles. The datafiles are named according to the format datatype\_experiment\_settingsname\_participantnumber\_setname\_eventname. This should allow the datafile’s name to include all the relevant information about what it contains.



For the single trial data, first there is a folder which shows the participant number, setname and eventname, and within each of those folders is the data for each electrode from that subject and condition and trialtype. These datafiles named as before.

