EEG-TOOLS USER GUIDE 2017

Instructions, rationale and example scripts to perform subject and group preprocessing of MEEG data with eeglab and Brainstorm software.

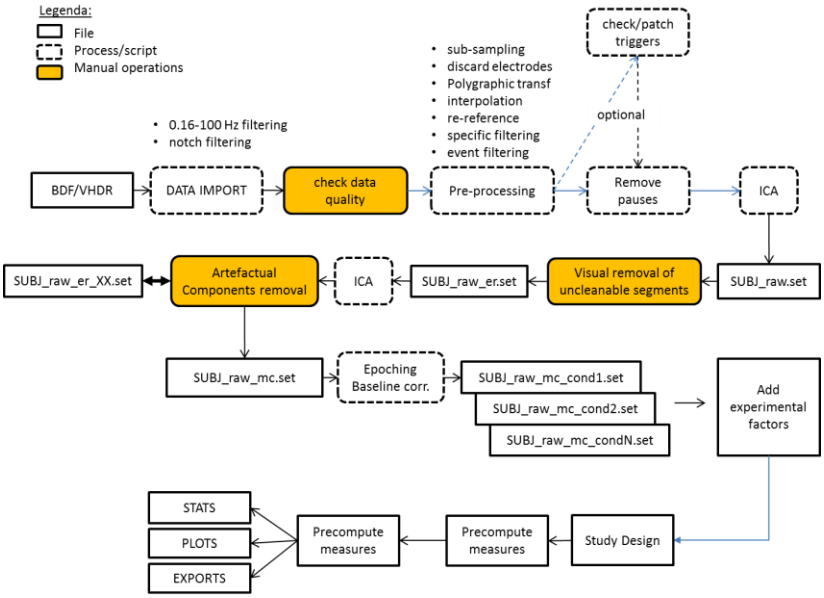
Rev. 0.1, September 2017.

Contents

Introduction

"EEG Tools" is a Matlab-based framework developed to automatize EEG analyses. It consists of a set of global scripts, few template project scripts and a set of naming conventions, mainly regarding the project file system structure, that must be accomplished for scripts usage.
Most of the scripts will call other software, in order to ease and automatize their usage. The current version analyzes EEG data both in sensors and sources space. Specifically, the electrode analyses are carried on with EEGLAB, source analysis reconstruction with BrainStorm and statistical analyses on sources with SPM8. A module analyzing functional connectivity is under development.

Analysis Pipeline



First-time Installation

These steps are required only the first time you setup this pipeline

1. Download and install EEGLab (https://sccn.ucsd.edu/eeglab/downloadtoolbox.php) and Brainstorm (http://neuroimage.usc.edu/bst/download.php) software. Rename them as "eeglab" and "brainstorm3", respectively.
2. Copy both software anywhere in your file system (referred as GLOBAL_SCRIPT_ROOT). For example: "Matlab Toolkit" folder
3. Create a root folder where you will put the data (eeg raw files, eeglab set/fdt files and your brainstorm database) of your EEG projects (referred as PROJECTS_DATA_ROOT). For example: "EEGData" folder.
4. Create a root folder where you will put all the matlab script of your EEG projects (referred as PROJECTS_SCRIPTS_ROOT). For instance, "All My Scripts" folder.

5. Finally, open Matlab and add the GLOBAL_SCRIPT_ROOT path. To do so, open the option "Seth Path" and select the folder you want to add. All the other paths will be added by the pipeline scripts.

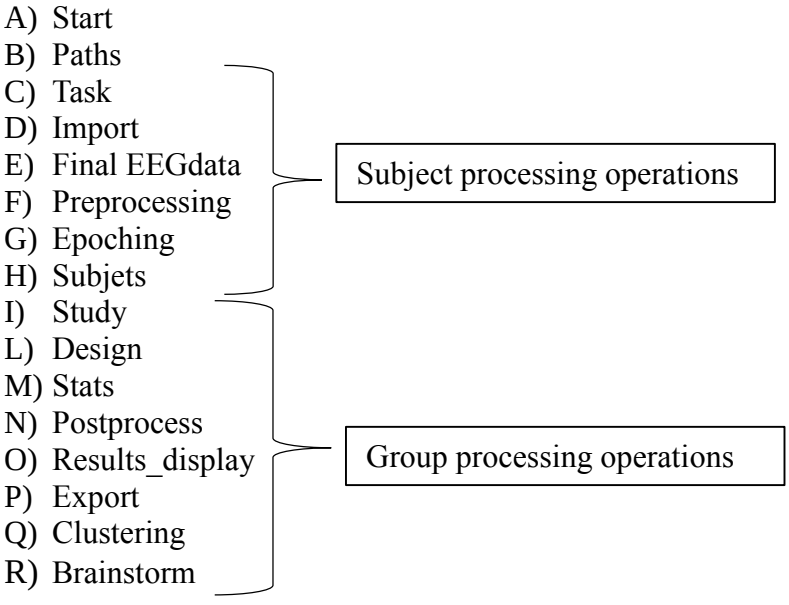
How to setup a new project

These steps are required each time you want to setup a new EEG project:

1. Create the project data folder: PROJECTS_DATA_ROOT/PROJ_NAME.
   e.g.: "EEGData/BLIND".
2. Create the project script folder: PROJECTS_SCRIPTS_ROOT/PROJ_NAME;
   e.g.: "All MyScripts /BLIND".
3. Copy the following three files from GLOBAL_SCRIPT_ROOT/eeg_tools/templates to your PROJECTS_SCRIPTS_ROOT/PROJ_NAME
   - template_project_structure.m
   - template_main_eeglab_subject_processing.m
   - template_main_eeglab_group_processing.m
4. Rename these files as you wish and edit them following the instruction explained in the documents titled "Project Structure", "Subject Processing", and "Group Processing".
5. Create the folder PROJECTS_DATA_ROOT/PROJ_NAME/original_data and copy all your raw eeg files.

Project Structure (project_structure.m)

The Project Structure Template contains all the specific information about our study and the analyses to be run. It let user specify how to perform all the operations, from the subject/individual level to the group level.
The first two parts are the same for both analyses, whereas the subject processing operations are defined in the parts C-H, and the group processing operations are specified in the I-R.
The explanation and the instructions to edit each section are described below.



A. Start

Similar to the first section of the Subject Processing Template, here we identify the project we are working on, as well as some general information about it.

project	
research_group	we write the name of the research group we are working with in this project.
research_subgroup	we write the name of the research subgroup involved in this study, or our name.
name	we name the project, but it must be the same as in the 'project.paths.local_projects_data'
study_suffix	label used to create a different study name, where the final file will be called: [project.name project.study_suffix '.study'])
analysis_name	Subfolder containing the epoching files of the current analysis type.
operations.do_source_analysis	
operations.do_emg_analysis	
operations.do_cluster_analysis	

For example:

```
project.research_group          = 'Experimental Psychology';
project.research_subgroup       = 'CDLab';
project.name                    = 'morality';
project.study_suffix            = 'toodlers';
project.analysis_name           = 'raw_data';
project.operations.do_source_analysis =
project.operations.do_emg_analysis =
project.operations.do_cluster_analysis =
```

B. Paths

Here are defined the paths of all the folders necessary to run the pipeline.

project.paths	
Set by:	What it contains:
main file	
projects_data_root	local projects root-folder
svn_scripts_root	local script root-folder
plugins_root	local MATLAB PLUGING root-folder
script.common_scripts	local projects root-folder
script.eeg_tools	
script.project	
define_paths_structure	
global_scripts	
global_spm_templates	
project	data, epochs, results etc...(not scripts)

original_data	EEG raw data (BDF, vhdr, eeg, etc...)
input_epochs	EEGLAB EEG input epochs set files
output_epochs	EEGLAB EEG output condition epochs set files
results	statistical results
emg_epochs	EEGLAB EMG epochs set files
emg_epochs_mat	EMG data strucuture
tf	
cluster_projection_erp	
batches	bash batches (usually for SPM analysis)
spmsources	sources images exported by brainstorm
spmstats	spm stat files
spm	spm toolbox
eeglab	eeglab toolbox
brainstorm	brainstorm toolbox

Subject Processing Operations

C. Task

In this part we specify the labels/triggers we introduced in our experiment when we designed the task, including starting/ending, pause, feedback, conditions, and baseline triggers.

project.task.events	
start_experiment_trigger_value	Label to inform that the task starts.
pause_trigger_value	Label to inform that a pause starts (pause, feedback and/or rest period).
resume_trigger_value	Label informing that a pause finishes (pause, feedback and/or rest period).
end_experiment_trigger_value	Label informing that the experiment is finished.
baseline_start_trigger_value	Label specifying when the baseline started.
baseline_end_trigger_value	Label specifying when the baseline finished.
trial_start_trigger_value	Label to specify when the trial started. If it is the same as the baseline, we just write "project.task.events.baseline_start_trigger_value".
trial_end_trigger_value	Label to specify when the trial finished. It might be substituted by the project.task.events.baseline_start_trigger_value" if it is the same.
mrkcode_cond	triggers defining conditions, where even if only one trigger is used for each condition, a cell matrix is used.
valid_marker	[project.task.events.mrkcode_cond {1:length(project.task.events.mrkcode_cond)}];
import_marker	Valid markers to be imported from the original files; for example "project.task.events.import_marker = [{'1' '2' '3' }]"

Note XX: baseline_start_trigger_value and trial_start_trigger_value MUST BE DIFFERENT. Thus, when they should coincide, put the former 2 samples after the latter. See the Advanced Epoching Section.

D: Import

Information about the MEEG recording device and the labeling scheme of the raw eeg data:

project.import	
acquisition_system	EEG hardware type: BIOSEMI | BRAINAMP
original_data_extension	original data file extension: bdf | vhdr
original_data_folder	original data file subfolder
original_data_suffix	Suffix or string after subject name in original EEG file name.
original_data_prefix	Prefix or string before subject name in original EEG file name.
output_folder	String appended to fullfile (project.paths.project,'epochs', ...) , determining where to write the imported files, by default coincides with the original_data_folder.
output_suffix	Suffix to add to the file name after importing original file.
emg_output_postfix	Suffix or string to add to the file name after importing EMG file.
reference_channels	List of electrodes to be used as reference; the options are: • [ ]: no referencing, • {'CAR'}: CAR ref, • {'el1', 'el2'}: use those electrodes.

For example:

```
project.import.acquisition_system          = 'BIOSEMI';
project.import.original_data_extension     = 'bdf';
project.import.original_data_folder        = 'ciechi';
project.import.original_data_suffix        = '';
project.import.original_data_prefix        = '';
project.import.output_folder               = project.import.original_data_folder;
project.import.output_suffix               = '_raw';
project.import.emg_output_postfix          = [];
```

Conventionally, EEGTOOL read raw data from
PROJ
    original_data
        original_data_folder

and writes set/fdt files in
PROJ
    epochs
        output_folder

By default, original_data_folder and output_folder coincides. Since user may test two different import procedures, he can specify a different folder for imported data.

priginal_data_prefix and original_data_suffix are used to help identifying the correct file name.

Polygraphic channels definition

Polygraphic channels, EMG, EOG, etc, may appear and the end of the EEG ones or be interleaved within them. In order to standardize recording montages, we decided to always append them at the

end of the EEG channels' list (possibly moving them from their original position). Thus at the end of the preprocessing, the first N channels are always the EEG ones and the other channels follow them. The project structure allows defining how the single NON-EEG channel must be treated. You can define their origin (EMG or EOG) and what to do with them. These are the possibilities:

List of electrodes to transform:
project.import.ch2transform(x) = struct('type', 'xxx' , 'ch1', 'yyy', 'ch2', 'zzz', 'new_label', 'www').

type	'eog', 'emg', 'other' : determines which kind of filtering they will undergo
ch1	the number of the channel to transform, if [] remove the channel
ch2	if [] is a monopolar channel, if NOT [] otherwise is a bipolar channel
new_label	if NOT [] label of the new bipolar channel obtained by subtracting : ch1 – ch2

For example, if we want to combine two EOG channels (acting as a bipolar channel) to have one EOG channel, we introduce the following script:

```
project.import.ch2transform(1) = struct('type', 'eog' , 'ch1', 65,'ch2', 66, 'new_label', 'dEOG');
```

However, if our aim is to discard the electrode 67 the script would be:

```
project.import.ch2transform(1) = struct('type', '' , 'ch1', 67,'ch2',[] , 'new_label', []);
```

project.import.valid_marker List of trigger marker to import. In general, it is the same as the markers specified in the previous section (project.task.events.import_marker). In that case:

```
project.import.valid_marker = project.task.events.import_marker;
```

Otherwise, we can import a cell array, or a string with these values: 'all', 'stimuli', 'responses'. For example:

```
project.import.valid_marker = {'S1' 'S2' 'S3' 'S4' 'S5' 'S 16' 'S 17' 'S 19' 'S 19' 'S 20' 'S 21' 'S 48' 'S 49' 'S 50' 'S 51' 'S 52' 'S 53' 'S 80' 'S 81' 'S 82' 'S 83' 'S 84' 'S 85' };
```

E. Final EEGData

project.eegdata	
nch	Final number of channels after removing electrodes and polygraphic transformation.
nch_eeg	EEG channels_number,
fs	Final sampling frequency in Hz; if the original is higher, then downsample it during pre-processing.
eeglab_channels_file_name	Universal channels file name containing the position of 385 channels.
eeglab_channels_file_path	Set later by define_paths
eeg_channels_list	List of EEG channels IDs

For example:

```
project.eegdata.nch = 64;
project.eegdata.nch_eeg = 64;
project.eegdata.fs = 256;
project.eegdata.eeglab_channels_file_name = 'standard-10-5-cap385.elp';
```

```
project.eegdata.eeglab_channels_file_path = ''
project.eegdata.eeg_channels_list = [1:project.eegdata.nch_eeg];
```

If you are working with the EMG and/or EOG channels, introduce the information about the channels and labels:
project.eegdata.emg_channels_list = [];
project.eegdata.emg_channels_list_labels = [];
project.eegdata.eog_channels_list = [];
project.eegdata.eog_channels_list_labels = [];

project.eegdata.no_eeg_channels_list → List of NO-EEG channels IDs; for instance:

```
project.eegdata.no_eeg_channels_list = [project.eegdata.emg_channels_list
project.eegdata.eog_channels_list];
```

F. Preprocessing

output_folder	string appended to fullfile (project.paths.project,'epochs', ...) , determining where to write imported file. If it is the same as the folder set in the Import section, complete as:
filter_algorithm	filter algorithm for all filters in the project. The options are:
	• 'pop_eegfiltnew_12' → it is the default filter of EEGLab, pop_eegfiltnew without the causal/non-causal option, and allows to set the band also for notch, so it's more flexible than pop_basicfilter of erplab. • 'pop_basicfilter' → erplab filters (version erplab_1.0.0.33: more recent presented many bugs) • 'causal_pop_iirfilt_12' → causal version of iirfilt • 'noncausal_pop_iirfilt_12' → noncausal version of iirfilt • 'causal_pop_eegfilt_12' → causal pop_eegfilt (old version of EEGLab filters) • 'noncausal_pop_eegfilt_12' → noncausal pop_eegfilt • 'causal_pop_eegfiltnew_13' → causal pop_eegfiltnew • 'noncausal_pop_eegfiltnew_13' → noncausal pop_eegfiltnew • The _12 suffix indicate filters of EEGLab 12; the _13 suffix indicate filters of EEGLab 13. • 'pop_eegfiltnew_12' → it is the default filter of EEGLab, pop_eegfiltnew without the causal/non-causal option, and allows to set the band also for notch, so it's more flexible than pop_basicfilter of erplab.
ff1_global	Lower frequency in Hz of the preliminary filtering applied during data import.
ff2_global	Higher frequency in Hz of the preliminary filtering applied during data import.
Notch filtering	
do_notch	Edit the script if it is applied the notch filter at 50 or 60 Hz: 0 (not to be done) / 1 (to be run).
notch_fcenter	Specify the frequency of the notch filter: 50 Hz or 60 Hz.
notch_fspan	Halved frequency range of the notch filters .

notch_remove_armonics	Remove all or only the first harmonic(s) : 'all' | 'first'
Specific filtering for EEG, EOG and EMG channels applied during preprocessing.	
ff1_eeg	Lower frequency in Hz of the EEG filtering
ff2_eeg	Higher frequency in Hz of the EEG filtering
ff1_eog	Lower frequency in Hz of the EOG filtering
ff2_eog	Higher frequency in Hz of the EEG filtering
ff1_emg	Lower frequency in Hz of the EMG filtering
ff2_emg	Higher frequency in Hz of the EMG filtering

For instance:

```
project.preproc.output_folder = project.import.output_folder;
project.preproc.ff1_global    = 0.16;
project.preproc.ff2_global    = 100;
project.preproc.do_notch  = 1;
project.preproc.notch_fcenter = 50;
project.preproc.notch_fspan   = 5
project.preproc.notch_remove_armonics = 'first';
project.preproc.ff1_eeg  = 0.16;
project.preproc.ff2_eeg = 45;
project.preproc.ff1_eog = 0.16;
project.preproc.ff2_eog = 8;
project.preproc.ff1_eog = 5;
project.preproc.ff2_eog = 100;
```

CALCULATE RT
project.preproc.rt.eve1_type →
project.preproc.rt.eve2_type →
project.preproc.rt.allowed_tw_ms.min →
project.preproc.rt.allowed_tw_ms.max →
project.preproc.rt.output_folder →

project.preproc.montage_list → we have this option if we need to unify different montages.
For example:

```
project.preproc.montage_list = {...
                        {'Fp1','Fp2','F7','F3','Fz','F4','F8','FC5','FC1','FC2','FC6','T7','C3','Cz',...
                        'C4','T8','TP9','CP5','CP1','CP2','CP6','TP10','P7','P3','Pz','P4','P8',...
                        'O1','Oz','O2','AF7','AF3','AF4','AF8','F5','F1','F2','F6','FT9','FT7',...
                        'FC3','FC4','FT8','FT10','C5','C1','C2','C6','TP7','CP3','CPz','CP4',...
                        'TP8','P5','P1','P2','P6','PO7','PO3','POz','PO4','PO8'};

                        {'Fp1','AF7','AF3','F1','F3','F5','F7','FT7','FC5','FC3','FC1','C1','C3','C5',...
                         'T7','TP7','CP5','CP3','CP1','P1','P3','P5','P7','P9','PO7','PO3','O1','Iz','Oz',...
                         'POz','Pz','CPz','Fpz','Fp2','AF8','AF4','AFz','Fz','F2','F4','F6','F8','FT8','FC6',...
                         'FC4','FC2','FCz','Cz','C2','C4','C6','T8','TP8','CP6','CP4','CP2','P2','P4','P6',...
                         'P8','P10','PO8','PO4','O2'}
};
```

project.preproc.insert_block.trials_per_block → Number of trials per block. Script to insert the block markers only if the project.preproc.insert_end_trial.end_trial_marker_type is not empty.

G. Epoching

Epoching consists in extracting, from the continuous data stream, blocks of time samples located around a certain trigger and save them to the file associated to such trigger. Our pipeline permits associating different triggers codes to the same experimental condition (e.g. you can observe 2 categories of videos and show repetitions of 4 different videos for each category). EEG-TOOLS allows managing the two most common case, you may in fact want to:

- extract a continuos block around the trigger
- extract two non-continuous blocks, one representing the baseline, the other is the signal

In the first case you do not need any additional triggers but the one identifying the experimental condition, in the latter case you 4 additional triggers.

- Start/end trial
- Start/end baseline

The best solution would be to insert all the necessary triggers during the experiment recordings, indicate them into the project structure and select the proper epoching method. When did was not done properly or the type of analysis does not permit it (e.g. you want to epoch across the user manual response that changes trial-by-trial), the framework lets you do it.

Pre-epoching processing

If we need to add new markers (not included into the original EEG data files), we have the following scripts:

In order to insert starting trial markers (only if both the target and the begin trial types are not empty):
Next scripts are aimed to insert the ending trial markers, only if both the target and the begin trial types are not empty.
The objective of the following lines is to insert a marker for the beginning of the baseline markers (project.epoching.baseline_replace.baseline_begin_marker):

project.preproc	
marker_type.begin_trial	
marker_type.end_trial	
marker_type.begin_baseline	
marker_type.end_baseline	
insert_begin_trial.target_event_types	string or cell array of strings denoting the type(s) (i.e. labels) of the target events used to set the begin trial markers.
insert_begin_trial.begin_trial_marker_type	
marker_type.begin_trial	string denoting the type (i.e. label) of the new begin trial marker.

insert_begin_trial.delay.s	time shift (in milliseconds) to anticipate (negative values) or to delay (positive values) the new begin trial markers with respect to the target events, if empty ([]) time shift = 0.
insert_end_trial.target_event_types	String or cell array of strings denoting the type(s) (i.e. labels) of the target events used to set the end trial markers.
insert_end_trial.end_trial_marker_type	
marker_type.end_trial	String denoting the type (i.e. label) of the new end trial marker.
insert_end_trial.delay.s	Time shift (in ms) to anticipate (negative values) or to postpone (positive values) the new end trial markers.

project.epoching	
baseline_mark. baseline_begin_target_marker	
baseline_mark. baseline_begin_target_marker_delay.s	a target event with certain delayed will be placed for the baseline markers. The delay (in seconds) between the target marker and the begin baseline marker to be placed:
	• >0 baseline FOLLOWS the target, • =0 baseline IS AT THE SAME TIME • <0 baseline ANTICIPATES the target. Important note: The latency information is displayed in seconds for continuous data or in milliseconds relative to the epoch's time-locking event for epoched data. As we will see in the event scripting section, the latency information is stored internally in data samples (points or EEGLAB 'pnts') relative to the beginning of the continuous data matrix (EEG.data). Similar to the previous scripts, in this case the lines are aimed to insert the end of the baseline markers (project.epoching.baseline_replace.baseline_end_marker)
baseline_mark. baseline_end_target_marker	{ '11','12','13','14','21','22','23','24','31'};

	A target event for placing the baseline markers: baseline begin marker will be placed at the target marker with a delay (in seconds). The options to set the delay between the target marker and the begin baseline marker to be placed are: • >0 baseline FOLLOWS the target, • =0 baseline IS AT THE SAME TIME • <0 baseline ANTICIPATES the target. Important note: The latency information is displayed in seconds for continuous data, or in milliseconds relative to the epoch's time-locking event for epoched data. As we will see in the event scripting section, the latency information is stored internally in data samples (points or EEGLAB 'pnts') relative to the beginning of the continuous data matrix (EEG.data).

For replacing baseline:
Consider the issues related to replace baseline. Replacing partially or totally the period before/after the experimental triggers has a problem associated: when epoching, generally it is required to do a baseline correction. However sometimes no part of the extracted epoch can be assumed as a good baseline.
The standard STUDY pipeline does NOT allow to consider smoothly external baselines.
One possibility is, for each trial, to replace part of the extracted epoch around each experimental event in the trial, by a segment (in the same trial or outside), that it's known to be a 'good' baseline.
The procedure has some requirements:

1) To have already marked in the recording events denoting begin/end of trial.
2) To have already marked in the recording events denoting begin/end of baseline.

Note that both requirements can be switched in the analysis if you that the 'good' baseline is at beginning of each trial (e.g. a pre-stimulus). In this case, you should mark the baselines (using as target events the stimuli) and then mark the trial, using as target for the beginning of the trial the new baseline begin marker. Or, you can to both mark baseline and trial use the same stimuli marker as target events, giving the right delays.

project.epoching.baseline_replace.mode → replace a baseline before/after events to be epoched and processed: 'trial' (employs a baseline within each trial), 'external' (employs a baseline obtained from a period of global baseline, not within the trials extracted from the current recording or from another file), 'none' (to not add a baseline -standard simple case-).
project.epoching.baseline_replace.baseline_originalposition → when replacing the new baseline: the baseline segments to be inserted are originally 'before' or 'after' the events to be epoched and processed.
project.epoching.baseline_replace.baseline_finalposition → when replacing the new baseline: the baseline segments are inserted 'before' or 'after' the events to be epoched and processed.
project.epoching.baseline_replace.trial_begin_marker →
It can be set as the project.preproc.marker_type.begin_trial;
project.epoching.baseline_replace.trial_end_marker →
It can be set as the project.preproc.marker_type.end_trial;
project.epoching.baseline_replace.baseline_begin_marker →
It can be set as the project.preproc.marker_type.begin_baseline;
project.epoching.baseline_replace.baseline_end_marker →

It can be set as the project.preproc.marker_type.end_baseline;
project.epoching.baseline_replace.replace →'all' or 'part' replace all the pre/post marker period with a replicated baseline or replace the baseline at the beginning (final position 'before') or at the end (final position 'after') of the reconstructed baseline.

For the EEG epoching:
project.epoching.input_suffix → suffix to add at the end of the file name before epoching, by default is '_raw_mc'.
project.epoching.input_folder → input epoch folder, by default the preprocessing output folder
project.epoching.bc_type → type of baseline correction; the options are: 'global' (considers all the trials), 'condition' (by condition), and 'trial' (trial-by-trial).
project.epoching.epo_st.s → EEG epochs start latency
project.epoching.epo_end.s → EEG epochs end latency
project.epoching.bc_st.s → EEG baseline correction start latency
project.epoching.bc_end.s → EEG baseline correction end latency
project.epoching.baseline_duration.s →

For example:

```
project.epoching.epo_st.s = -1.012;
project.epoching.epo_end.s = 1.512;
project.epoching.bc_st.s = -1;
project.epoching.bc_end.s = -0.012;
project.epoching.baseline_duration.s = project.epoching.bc_end.s - project.epoching.bc_st.s ;
```

For the EEG point:
project.epoching.bc_st_point → EEG baseline correction start point.
project.epoching.bc_end_point → EEG baseline correction end point.

For instance:

```
project.epoching.bc_st_point → = round((project.epoching.bc_st.s-
project.epoching.epo_st.s)*project.eegdata.fs)+1;
project.epoching.bc_end_point = round((project.epoching.bc_end.s-
project.epoching.epo_st.s)*project.eegdata.fs)+1;
```

For the EMG epoching:
project.epoching.emg_epo_st.s → EMG epochs start latency
project.epoching.emg_epo_end.s → EMG epochs end latency
project.epoching.emg_bc_st.s → EMG baseline correction start latency
project.epoching.emg_bc_end.s → EMG baseline correction end latency

For the EMG point:
project.epoching.emg_bc_st_point → EMG baseline correction start point
project.epoching.emg_bc_end_point → EMG baseline correction end point

For example:

```
project.epoching.emg_bc_st_point = round((project.epoching.emg_bc_st.s-
project.epoching.emg_epo_st.s)*project.eegdata.fs)+1;
project.epoching.emg_bc_end_point = round((project.epoching.emg_bc_end.s-
project.epoching.emg_epo_st.s)*project.eegdata.fs)+1;
```

For the markers:
project.epoching.mrkcode_cond →

project.epoching.numcond → number of conditions.
project.epoching.valid_marker →
project.epoching.condition_names → labels for the conditions.

For example:

```
project.epoching.mrkcode_cond = project.task.events.mrkcode_cond;
project.epoching.numcond = length(project.epoching.mrkcode_cond);
project.epoching.valid_marker =
project.epoching.mrkcode_cond{1:length(project.epoching.mrkcode_cond)}];
project.epoching.condition_names = {'control' 'attention' 'movement'};
```

H. Subjects

In this section we fill in the information about our subjects and groups.

project.subjects.narrowband_file →
project.subjects.baseline_file →
project.subjects.baseline_file_interval_s →
project.subjects.narrowband_suffix_cell →

Then we introduce the information about each subject following the structure:

project.subjects.data(1) = struct('name', '_', 'group', '_', 'age', _, 'gender', '_', 'handedness', '_', 'bad_ch', [],'baseline_file',[],'baseline_file_interval_s',[],'frequency_bands_list',[]);

Where:

- Name → subject's name.
- Group → subject's group.
- Age → subject's age.
- Gender → subject's gender.
- Handedness → subject's handedness.
- Bad_ch → we will specify them later.
- Baseline_file →
- Baseline_file_interval_s →
- Frequency_bands_list →

So if the second subject of our study is called BPP, from the control group, she is 20 years old, female, and right-handed, the script will be:

```
% project.subjects.data(2) = struct('name', 'BPP', 'group', 'CG', 'age', 20, 'gender', 'f', 'handedness', 'r', 'bad_ch', [],'baseline_file',[],'baseline_file_interval_s',[],'frequency_bands_list',[]);
```

project.subjects.data(_).bad_ch → we introduce the bad channels for each participant. For example, if the subject number 3 has two bad channels (PO1 and PO3), we will edit as:

```
project.subjects.data(3).bad_ch = {'PO1', 'PO3'};
```

For example:

```
...project.subjects.data(6).frequency_bands_list = {[4,8];[6,10];[14,20];[20,32]};
```

project.subjects.list →
project.subjects.numsubj →

project.subjects.group_names → groups' name
project.subjects.groups → subjects' name by group

For instance:

```
project.subjects.group_names →      = {'CB',  'HC'};
project.subjects.groups   →         = {{'CB_01', 'CB_02','CB_03','CB_05',  'CB_07', 'CB_08',
'CB_09', 'CB_10', 'CB_11', 'CB_12','CB_14', 'CB_15'}; …
{'HC_01',  'HC_04', 'HC_05', 'HC_06', 'HC_07','HC_08',  'HC_10', 'HC_12', 'HC_13', 'HC_14',
'HC_15', 'HC_16', 'HC_17', 'HC_18','HC_19'}; ... };
```

Group Processing Operations

List of sections for the group processing:

I. Study
L. Design
M. Stats
} Steps containing the information to create the 'study' file
N. Postprocess
O. Results_display
P. Export
Q. Clustering
R. Brainstorm

I. Study

This section is aimed to add the information about factors' levels of each condition into every single file.

IMPORTANT NOTE: as additional factors are added to the EEG.event structure as new fields, you cannot call the new factor names using mathematical operators, which are wrongly intrepretated by matlab. eg, replace the name 'condition-group' with 'condition_group' or 'conditionGroup' OR 'conditionXgroup'

project.study.filename = [project.name project.study_suffix '.study']; →this will add the suffix 'study'

Following structure associates the conditions' file with (multiple) factor(s):

To be completed later

project.study.factors(1) = struct('factor', '__', 'file_match', [], 'level', '__');

Factor's name

Level's name

If we have a study with one factor (motion), and has two conditions (translating and centered) we will edit the script as:

```
project.study.factors(1) = struct('factor', 'motion', 'file_match', [], 'level', 'translating');
```

```
project.study.factors(2) = struct('factor', 'motion', 'file_match', [], 'level', 'centered');
```

If our study has more factors, two factors (motion and shape), and each one has two conditions (translating and centered, and walker and scrambled, respectively), our script will be:

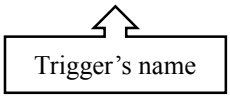
```
project.study.factors(1) = struct('factor', 'motion', 'file_match', [], 'level', 'translating');
project.study.factors(2) = struct('factor', 'motion', 'file_match', [], 'level', 'centered');
project.study.factors(3) = struct('factor', 'shape' , 'file_match', [], 'level', 'walker');
project.study.factors(4) = struct('factor', 'shape' , 'file_match', [], 'level', 'scrambled');
```

But apart from the levels of each factor, it is also a requirement to specify the triggers of each level. Following our example, there are two triggers for the each level:

- "Translating" level includes the triggers "translating walker" and "translating scrambled".
- "Centered" level includes the triggers "centered walker" and "centered scrambled".
- "Walker" level includes the triggers "translating walker" and "centered walker".
- "Scrambled" level includes the triggers "translating scrambled" and "centered scrambled".

Then we will reflect that information in the structure:

project.study.factors(1).file_match = {'__', '__'};



And the final scripts are:

```
project.study.factors(1).file_match = {'twalker', 'tscrambled'};
project.study.factors(2).file_match = {'cwalker', 'cscrambled'};
project.study.factors(3).file_match = {'twalker', 'cwalker'};
project.study.factors(4).file_match = {'tscrambled', 'cscrambled'};
```

To include the ERP information into the study we have the lines:
project.study.erp.tmin_analysis.s → when the ERP starts.
project.study.erp.tmax_analysis.s → when the ERP finishes.

We have the option to set both times as the times fixed for the epoching:

```
project.study.erp.tmin_analysis.s = project.epoching.epo_st.s;
project.study.erp.tmax_analysis.s = project.epoching.epo_end.s;
```

project.study.erp.ts_analysis.s →
project.study.erp.timeout_analysis_interval.s →

To include the ERSP information into the study we have the following scripts, with the same structure than in the ERP information:
project.study.ersp.tmin_analysis.s
project.study.ersp.tmax_analysis.s
project.study.ersp.ts_analysis.s
project.study.ersp.timeout_analysis_interval.s

And more specific information:
project.study.ersp.fmin_analysis → minimum frequency for the ERSP analysis.
project.study.ersp.fmax_analysis → maximum frequency for the ERSP analysis.
project.study.ersp.fs_analysis →
project.study.ersp.freqout_analysis_interval →
project.study.ersp.padratio →
project.study.ersp.cycles →

Operations to be compute:
project.study.precompute.recompute → 'on'/'off'
project.study.precompute.do_erp → 'on'/'off'
project.study.precompute.do_ersp → 'on'/'off'
project.study.precompute.do_erpim → 'on'/'off'
project.study.precompute.do_spec → 'on'/'off'

L. Design

The design section is to specify all the potential experimental design in our study with the next structure:

project.design(1) = struct('name', 'condition_x_group' , 'factor1_name', 'condition', 'factor1_levels', [] , 'factor1_pairing', 'on', 'factor2_name', 'group' , 'factor2_levels', [], 'factor2_pairing', 'off');

project.design(1).factor1_levels → Levels of the first factor.
project.design(1).factor2_levels → Levels of the second factor.

If, for example, the levels of the first factor are our conditions, and the levels of the second one are the groups we have already defined in the experiment, the script will be:

```
project.design(1).factor1_levels    = project.epoching.condition_names;
project.design(1).factor2_levels    = project.subjects.group_names;
```

I HAVE TO COMPLETE THIS PART WITH MORE EXAMPLES, AND MORE COMPLEX

N. Stats and Postprocess

1) ERP analyses:

During the first lines we introduce the statistical information, later we will edit the details for the analyses.
project.stats.erp.pvalue → level of significance applied in ERP statistical analysis.
project.stats.erp.num_permutations → number of permutations applied in ERP statistical analysis.
project.stats.erp.num_tails → number of tails for the analyses.
project.stats.eeglab.erp.method → method applied in ERP statistical analysis
project.stats.eeglab.erp.correction → multiple comparison correction applied in ERP statistical analysis: 'fdr';

Next structures address different types of analyses to be run with the ERP (see the document called "Plots and Analyses"):
project.postprocess.erp.mode.continous → continuous
project.postprocess.erp.mode.tw_group_noalign → aligned group
project.postprocess.erp.mode.tw_group_noalign → non-aligned group
project.postprocess.erp.mode.tw_individual_noalign → aligned individual
project.postprocess.erp.mode.tw_individual_align → non-aligned individual

project.postprocess.erp.sel_extrema → the options are: 'first_occurrence' or 'avg_occurrences'.

project.postprocess.erp.roi_list → we introduce the interesting ROI into brackets (one ROI per line) and the names.
project.postprocess.erp.roi_names → names of the interesting ROI.
project.postprocess.erp.numroi → number of interesting ROI, it might be completed with the information from the first script, in that case =length(project.postprocess.erp.roi_list);

For instance:

```
project.postprocess.erp.roi_list = {  ...
        {'PO7'};  ... left
        {'PO8'};  ... right
    ... {'C3'}; ... left M1 hand
      };
project.postprocess.erp.roi_names={'left-inf-PO','right-inf-PO'}; ...,'right-SM1'};
project.postprocess.erp.numroi=length(project.postprocess.erp.roi_list);
```

Following commands follow the same structure, but for EOG and EMG respectively:

project.postprocess.erp.eog.roi_list
project.postprocess.erp.eog.roi_names
project.postprocess.erp.eog.numroi

project.postprocess.erp.emg.roi_list
project.postprocess.erp.emg.roi_names
project.postprocess.erp.emg.numroi

Next lines are required to inform about the time windows with the structure:
project.postprocess.erp.design(1).group_time_windows(1)= struct('name','_','min',_, 'max',_);
project.postprocess.erp.design(1).subject_time_windows(1)= struct('min',__, 'max',__);

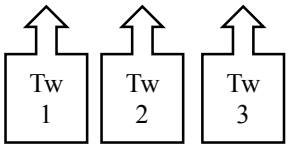
Where the "group time windows" is the wide time window you want to study, and the "subject time window" is the small time window around the peak. Note that, when we are fixing the subject individual window, it is better if the numbers are multiple of the "project.study.erp.ts_analysis.s"; otherwise the program interpolates it.
So if we are interested in the N2, with a global time windows between 200-280 milliseconds, and a subject time windows between -16 to 16 milliseconds, we will edit as:

```
project.postprocess.erp.design(1).group_time_windows(1) = -struct('name','N2','min',200,
'max',280);
project.postprocess.erp.design(1).subject_time_windows(1)   = struct('min',-16, 'max',16);
```

For next lines, we need to say if we are interested on the minimum (-min- negative) or on the maximum (-max- positive) point of the curve, for each ROI (in the row) and time window (-tw- in the column). The structure is:
project.postprocess.erp.design(1).which_extrema_curve   = {

{'min';'max';'min';'min'}; → roi 1
{'min';'max';'min';'min'}; → roi 2
{'min';'max';'min';'min'}; → roi 3

};



The same structure for EOG:

```
project.postprocess.erp.eog.design(1).which_extrema_curve = {
                    {'max';'min';'min';'min'};
                    {'max';'min';'min';'min'};
                    {'max';'min';'min';'min'};
};
```

As well as for the EMG:

```
project.postprocess.erp.emg.design(1).which_extrema_curve = {
                    {'max';'min';'min';'min'};
                    {'max';'min';'min';'min'};
                    {'max';'min';'min';'min'};
};
```

Also, we might be interested on the deflections, assuming that the curve is a continuous. In such a case we have the next scripts following the same structure than curves:

```
project.postprocess.erp.design(1).deflection_polarity_list = {
                    {'positive';'positive';'positive';'positive';'positive';'negative'};
                    {'positive';'positive';'positive';'positive';'positive';'negative'};
                    };
```

For the EOG it will be:

```
project.postprocess.erp.eog.design(1).deflection_polarity_list = {
                    {'positive';'positive';'positive';'positive';'positive';'negative'}; ... roi 1
                    {'positive';'positive';'positive';'positive';'positive';'negative'}; ... roi 2
                    {'positive';'positive';'positive';'positive';'positive';'positive'}; ... roi 3
                    };
```

And for the EMG:

```
project.postprocess.erp.emg.design(1).deflection_polarity_list = {
                    {'positive';'positive';'positive';'positive';'positive';'negative'}; ... roi 1
                    {'positive';'positive';'positive';'positive';'positive';'negative'}; ... roi 2
                    {'positive';'positive';'positive';'positive';'positive';'positive'}; ... roi 3
                    };
```

If we use the deflections instead of the curves, we need to set the minimum duration (in milliseconds) to be considered as deflections, deflections shorter than our criteria will be removed. For doing that, we have the structure:
project.postprocess.erp.design(1).min_duration
project.postprocess.erp.eog.design(1).min_duration → EOG
project.postprocess.erp.emg.design(1).min_duration → EMG

2) ERSP analyses:

During the first lines we introduce the statistical information, later we will edit the details for the analyses.

project.postprocess.ersp.sel_extrema → the options are: 'first_occurrence' or 'avg_occurrences'.

As in the ERP section, with the ERSP there are also different types of analyses (see "Plots and Analyses" document) to be run:
project.postprocess.ersp.mode.continous → continuos
project.postprocess.ersp.mode.tw_group_noalign → aligned group
project.postprocess.ersp.mode.tw_group_noalign → non-aligned group
project.postprocess.ersp.mode.tw_individual_noalign → aligned individual
project.postprocess.ersp.mode.tw_individual_align → non-aligned individual

And we set the statistical information for the time-frequency analyses with:
project.stats.ersp.pvalue → level of significance applied in ERSP statistical analysis.
project.stats.ersp.num_permutations → number of permutations applied in ERP statistical analysis.
project.stats.ersp.num_tails → number of tails for the analysis.
project.stats.ersp.decimation_factor_times_tf →
project.stats.ersp.decimation_factor_freqs_tf →
project.stats.ersp.tf_resolution_mode → the options for the resolution mode are 'continuous'; 'decimate_times'; 'decimate_freqs'; 'decimate_times_freqs'; 'tw_fb'.
project.stats.ersp.measure → the options for the measure are 'dB' (decibel); 'Pfu' ((A-R)/R * 100 = (A/R-1) * 100 = (10^.(ERSP/10)-1)*100)
project.stats.eeglab.ersp.method → method applied in ERP statistical analysis.
project.stats.eeglab.ersp.correction → multiple comparison correction applied in ERP statistical analysis

For the frequency band analyses, we modify the structure:

project.postprocess.ersp.frequency_bands(1)=struct('name','__','min',__,'max',__,'dfmin',__,'dfmax',__,'ref_roi_list',{'__'}, 'ref_roi_name','__','ref_cond', '__', 'ref_tw_list', [___], 'ref_tw_name', '__', 'which_realign_measure','__');

Where:

- name → name of the frequency band of interest
- min → lower bound of the frequency band of interest (in Hertz)
- max →upper bound of the frequency band of interest (in Hertz)
- dfmin →

- dfmax →
- ref_roi_lis → roi/channel to be used as a reference
- ref_roi_name → name of the roi/channel to be used as a reference
- ref_cond → experimental condition to be used as a reference
- ref_tw_list → time window (lower and upper bound)
- ref_tw_name → name of the time window
- which_realign_measure → type of realign to be used. The options are: min |max |auc. For each band, select the frequency with the maximum or the minumum ersp or the largest area under the curve to realign the narrowband

For example, if we are interested in studying the mu frequency band (8-12 Hertz, plus/minus one), using the Cz and also the translating scrambled condition as references, with a time window ranging between 0-100, and the auc realign; the script will be:

```
project.postprocess.ersp.frequency_bands(2)=struct('name','mu','min',8,'max',12,'dfmin',1,'dfmax',1,'
ref_roi_list',{'Cz'}, 'ref_roi_name','Cz','ref_cond', 'tscrambled', 'ref_tw_list', [0 100], 'ref_tw_name',
'gigi', 'which_realign_measure','auc');
```

For the narrow band analysis:

project.stats.ersp.do_narrowband → the adjustment of spectral band for each subject; options are off|ref|auto; where off = no adjustment, the information is not used; ref adjustment is based on a reference condition; auto adjusts each condition separately.
project.stats.ersp.narrowband.group_tmin → lowest time of the time windows considered to select the narrow band. If empty, consider the start of the epoch.
project.stats.ersp.narrowband.group_tmax → highest time of the time windows considered to select the narrow band. If empty, consider the end of the epoch.
project.stats.ersp.narrowband.dfmin → low variation in Hz from the barycenter frequency.
project.stats.ersp.narrowband.dfmax → high variation in Hz from the barycenter frequency.

project.stats.ersp.narrowband.which_realign_measure → min |max |auc for each band. It is aimed to calculate the fnb. The parameters select the frequency with the maximum (max) or the minimum (min) ERSP or the largest area under the curve to realign the narrowband (auc). It has to be introduced with this format: {'max','min','min','min'}, where the first one is related to the first frequency band, the second is associated to the second one, and so on.
project.stats.ersp.narrowband.which_realign_param → fnb | cog_pos | cog_neg | cog_all. Set if re-align the narrowband to the peak (defined above) of to the center-of-gravity (cog) within the wide band. It has to be edited following this format: {'cog_pos','cog_neg','cog_neg','cog_neg'}. Even if it is usually applied the cog realignment, the fnb might be useful when you have different ages or pathologies.

As we have done with the ERP, here we have the scripts to be edited for ROI:

project.postprocess.ersp.roi_list → we introduce the interesting ROI into brackets (one ROI per line) and the names.
project.postprocess.ersp.roi_names → names of the interesting ROI.
project.postprocess.ersp.numroi → number of interesting ROI, it might be completed with the information from the first script, in that case =length(project.postprocess.ersp.roi_list);

For instance:

```
project.postprocess.ersp.roi_list = {  ...
```

```
        {'PO7'};  ... left
        {'PO8'};  ... right
    ... {'C3'}; ... left M1 hand
      };
project.postprocess.ersp.roi_names={'left-inf-PO','right-inf-PO'}; ...,'right-SM1'};
project.postprocess.ersp.numroi=length(project.postprocess.erp.roi_list);
```

Following commands follow the same structure, but for EOG and EMG respectively:

project.postprocess.ersp.eog.roi_list
project.postprocess.ersp.eog.roi_names
project.postprocess.ersp.eog.numroi

project.postprocess.ersp.emg.roi_list
project.postprocess.ersp.emg.roi_names
project.postprocess.ersp.emg.numroi

For the time windows:

Next lines are aimed to inform about the time windows with the structure:
project.postprocess.erp.design(1).group_time_windows(1)= struct('name','__','min',__, 'max',__);
project.postprocess.erp.design(1).subject_time_windows(1)= struct('min',__, 'max',__);

So if we are interested in the time windows between 350-500 milliseconds, and an individual time windows between -100 to 100 milliseconds, we will edit as:

```
project.postprocess.erp.design(1).group_time_windows(1) = -struct('name','350-500','min',350,
'max',500);
project.postprocess.erp.design(1).subject_time_windows(1)   = struct('min',-100, 'max',100);
```

Main Processing Files

Two class of analyses are present, one dealing with individual operations(from raw data import to experimental factors inclusion in each condition file), the other performing group-level operations (from model creation to results export). Two different main files

Subject Processing (main_eeglab_subject_processing.m)

Special variables

These variables are the only allowed to exist outside project. They are passed to each project function as varargin and their are used only if filled. They can thus be empty.

list_select_subjects

In order to select the subjects to be processed, several options are feasible:

Use all the subjects defined in PS

1) //list_select_subjects = []
2) list_select_subjects = []
3) list_select_subjects = project.subjects.list

Consider only some subjects

4) define their label in a cell array of strings
   list_select_subjects ={'ID01', 'ID08', 'ID12',}

custom_suffix

Use to add some suffix labels to the managed file. For instance, if we take the raw data (e.g., ID01_raw), and run the preprocessing steps, we would add the label '_pre' (custom_suffix = '_pre'), and save those data as different files (e.g., ID01_pre).

stat_threshold
electrode2inspect
save_figure
fig_output_path

File naming scheme

EEGTOOL will find files within the project folder according to the following file labelling conventions. A specific function

eeglab/subject/proj_eeglab_subject_get_filename(project, subj_name, analysis_step,varargin)

have been created in order to compose the correct subject's name according to the task (import, pre-processing, epoching etc..) performed.

analysis_step can have the following values:

'input_import_data'
'output_import_data'
'output_preprocessing'
'custom_pre_epochs'
'uniform_montage', 'input_epoching'
'output_epoching', 'add_factor', 'extract_narrowband'
'custom_step'

The function accept the following varargin variables pairs (label, value).

custom_suffix :
cond_name :
custom_input_folder :

It follows an excerpt of its code, stressing which project structure variables are used to define the subject's file name.

Note XX: project variable was omitted, that is paths.original_data is actually project.paths.original_data

case 'input_import_data'
paths.original_data [import.original_data_prefix subj_name import.original_data_suffix 'import.original_data_extension]

case 'output_import_data'
paths.output_import,[import.original_data_prefix subj_name import.original_data_suffix import.output_suffix custom_suffix]

case 'output_preprocessing'
paths.output_preprocessing,[import.original_data_prefix subj_name import.original_data_suffix import.output_suffix custom_suffix]

case 'custom_pre_epochs'
paths.output_preprocessing,[import.original_data_prefix subj_name import.original_data_suffix import.output_suffix custom_suffix]

case {'uniform_montage', 'input_epoching'}
paths.input_epochs [import.original_data_prefix subj_name import.original_data_suffix import.output_suffix epoching.input_suffix custom_suffix]

case {'output_epoching', 'add_factor', 'extract_narrowband'}
paths.output_epochs [import.original_data_prefix subj_name import.original_data_suffix import.output_suffix epoching.input_suffix custom_suffix '_' cond_name]

case {'custom_step'}
custom_input_folder, [import.original_data_prefix subj_name import.original_data_suffix import.output_suffix custom_suffix]

For conventional operations, user just need to modify the project structures once, for custom operations user can instead primarily manage the following one:

custom_suffix

this variable is defined in the to add some suffix labels to the output file. For instance, if we take the raw data (e.g., ID01_raw), and run the preprocessing steps, we would add the label '_pre' (custom_suffix = '_pre'), and save those data as different files (e.g., ID01_pre).

Com
that a

1) Local Paths

In the first section of the pipeline we specify the localization/path where the information required for running the analyses is located.
Depending on the system we are using (Linux or Windows), we have to complete the information in the first section (under "if), or the second one (under "else"). If we are working in Linux, we need to modify the scripts included in the first part, below the sentence starting with "if"; but if we are working on another system, we would do the same but below the part starting with "else".

For the paths to be set up:

- **project.paths.projects_data_root**: we copy the path where our folder "PROJECTS_DATA_ROOT" is placed.
- **project.paths.projects_scripts_root**: we copy the path where our folder "PROJECTS_SCRIPT_ROOT" is placed.

- **project.paths.global_scripts_root**: we copy the path where our folder "GLOBAL_SCRIPT_ROOT" is placed.
- **project.paths.plugins_root**: we copy the path where our toolbox of Matlab is placed.

2) Project Data

Here we identify the project we are working on, as well as some general information about it.

- **project.research_group**: we write the name of the research group we are working with in this project.
- **project.research_subgroup**: we write the name of the research subgroup involved in this study, or our name.
- **project.name**: we name the project, but it must be the same as in the 'project.paths.local_projects_data' subfolder name
- **project.conf_file_name**: we specify the name of the project_structure file we are using in this study.

As an example:

```
project.research_group = 'Experimental Psychology';
project.research_subgroup = 'CDLab';
project.name = 'morality';
project.conf_file_name = 'Moral_project_structure';
```

3) Project Structure and File System Initialization

Here we do not need to edit or include any information; with the lines already included we will get our project and paths ready.

4) Design Specification

????

5) Override

This section is useful when you are testing some analyses or preprocessing steps, because you are able to identify just one subject or channel to run the operations.

custom_suffix → to add some suffix labels to the output file. For instance, if we take the raw data (e.g., ID01_raw), and run the preprocessing steps, we would add the label '_pre' (custom_suffix = '_pre'), and save those data as different files (e.g., ID01_pre).
stat_threshold →
electrode2inspect →
save_figure →
analysis_name →
list_select_subjects → In order to select the subjects to be processed, several options are feasible:

5) commented
6) set to [ ] → all the subject will be selected; e.g.: list_select_subjects = []
7) set to a cell array of strings → select specific subjects; e.g.: list_select_subjects ={'ID01'}, or list_select_subjects ={'ID01', 'ID08', 'ID12',}
8) set to project.subjects.list →????

fig_output_path →

6) Operations List

This is the last section of the subject processing script to be edited. Here we only need to choose which operation/analysis we want to run by writing 0 (not to be run) or 1 (to be run) in the specific command. For instance:

project.operations.do_import	= 1;	→ Importation will be run
project.operations.do_ica	= 0;	→ ICA will not be executed

If we choose two or more commands to be executed, MATLAB will run the operations in sequence. All the possible operations are:

project.operations.do_import
With this command we will import raw data and write set/fdt file into epochs subfolder, following the details given in the *project_structure* file.

project.operations.do_preproc
This script is to run the preprocessing of the imported file (sub-sampling, discard electrodes, channels transformation, interpolation, re-reference, specific filtering,..)

project.operations.do_emg_analysis
do emg extraction analysis.

project.operations.do_patch_triggers
This option must be run or by itself or, if we choose several commands, this one must to be set as the last operation to be run.
The current command allows us to edit our event triggers once the experiment has been finished and we realize we need to make some changes in the triggers. If we want to modify some of the triggers, the options appear at the end of this section, under the title “START PROCESSING”, and all of them will be clarify at the end of this section.

project.operations.do_auto_pauses_removal
If in the experiment we have introduced markers to label the beginning and the end of the pauses, here we are able to remove these pauses and undesired time intervals.

project.operations.do_testart
allow testing some semi-automatic artifact removal algorithms

project.operations.do_ica
Two ICAs have to be performed:
1) First ICA is run to identify whether the artifacts are presented in several channels but located at one specific time point → delete epochs.

After detecting and deleting the bad epochs, we save the EEG file (different folder, specific suffix "-.er"), and then we run the next ICA.

2) Second ICA is executed to examine whether the artifacts are specific or located and punctual components, like muscular tension or ocular movement → delete components.
   Once we have deleted these noisy component, we save the new EEG file (another folder, suffix "-.mc").

project.operations.do_uniform_montage
This operation is used to uniform montages when we have different polygraphs.

project.operations.do_mark_trial
If after running the experiment we realized that we needed more trial triggers, this option will insert the triggers into the data automatically.

project.operations.do_mark_baseline
This script is aimed to insert the baseline triggers into the data.

project.operations.do_check_mc
With this option EEGLab will check mc file status (triggers, number of epochs, errors, etc.) before doing the epoching.

project.operations.do_epochs
Once our EEG files contained all the required information (trial triggers and markers) and are clean (we have run both ICA), we are ready for epoching the data. The information used to run the epoching is in the *project_structure* file, where we have set the epoching start latency as well as the end latency.

project.operations.do_custom_epochs
Choose this operation if you need to custom the epoching, and not to run the epoching with the original information written in the *project_structure* file.

project.operations.do_handedness_epochs
Select this option to custom epoching that swap electrodes according to handedness.

project.operations.do_factors
This is the last step of the subject processing operation, after running this operation we are ready to perform the group processing analyses.
With this operation we add the factor and level information into the each EEG data file. This information has to be specified in the *project_structure* file, into the *Factor levels of each condition files*"section.

project.operations.do_singlesubjects_band_comparison
Choose this option to perform basic single-subject plotting and analysis, basically one condition spectral graphs, single epochs desynch, or two conditions comparisons.

project.operations.do_extract_narrowband
Select this operation to extract narrowband for each subject and selected condition and save separately each condition in a mat file.

Options for the project.operations.do_patch_triggers command:

0) Rename triggers
   eeglab_subject_events_rename(file_name, {'50','60','70'},{'6','7','8'});

subj_name = list_select_subjects{subj}; file_name = proj_eeglab_subject_get_filename(project, subj_name, 'custom_pre_epochs', 'custom_suffix', custom_suffix);

1) If not present, add start and/or end trigger 1 sec before the first trigger:
eeglab_subject_events_add_event_around_first_event(file_name, '',);
eeglab_subject_events_add_event_around_last_event(file_name, '',);
For example, if we want to introduce the start trigger (labeled as 1) two seconds before the first event, we will set the command as: eeglab_subject_events_add_event_around_first_event (file_name, '1', -2).
Whereas if we need to add the end trigger (labeled as 6) four seconds after the last event, the script would be set as:
eeglab_subject_events_add_event_around_last_event(file_name, '6', 4);

2) If you want to anticipate the pause-end trigger:
eeglab_subject_events_move_second_event_in_a_sequence(file_name, {project.task.events.pause_trigger_value}, {project.task.events.resume_trigger_value}, -1.5);

3) When only the question mark exists, and not the pause marks, add a pause trigger value at the same latency of a question_trigger_value and a resume trigger 1sec before the event following the question trigger:
eeglab_subject_events_add_event_at_code_and_around_next(file_name, project.task.events.question_trigger_value, project.task.events.pause_trigger_value, project.task.events.resume_trigger_value, (project.epoching.epo_st.s - 0.1));

4) When the question and the resume triggers are available, add a pause trigger value at the same latency of a question_trigger_value:
eeglab_subject_events_add_event_around_another_event(file_name, project.task.events.question_trigger_value, project.task.events.pause_trigger_value, 0.01);
eeglab_subject_events_add_eve1_pre_eve2(file_name {project.task.events.question_trigger_value},{project.task.events.resume_trigger_value},10000000000,project.task.events.pause_trigger_value); same function as above

5) Mark baseline start and end:
eeglab_subject_events_add_event_around_other_events(file_name, [project.epoching.mrkcode_cond{1:length(project.epoching.mrkcode_cond)}], project.task.events.baseline_start_trigger_value, project.epoching.bc_st.s);
eeglab_subject_events_add_event_around_other_events(file_name, [project.epoching.mrkcode_cond{1:length(project.epoching.mrkcode_cond)}], project.task.events.baseline_end_trigger_value, project.epoching.bc_end.s);
eeglab_subject_events_add_event_around_other_events(file_name, [project.epoching.mrkcode_cond{1:length(project.epoching.mrkcode_cond)}], project.task.events.trial_end_trigger_value , project.epoching.epo_end.s);

6) When end trial trigger does not exist:
eeglab_subject_events_add_event_around_other_events(file_name, [project.epoching.mrkcode_cond{1:length(project.epoching.mrkcode_cond)}], project.task.events.videoend_trigger_value, 3.4);

7) Calculate the distance between triggers, you need to create the following variables:
means1 = zeros(project.subjects.numsubj,1);
means2 = zeros(project.subjects.numsubj,1);

```
...subjects_data{subj} = eeglab_subject_events_distances_three_triggers_classes(file_name,
[project.epoching.mrkcode_cond{3:length(project.epoching.mrkcode_cond)}], {'7','8'}, {'5'});
...means1(subj)        = mean([subjects_data{subj}{:,2}]);
...if size(subjects_data{subj}, 2) > 2
...    means2(subj)        = mean([subjects_data{subj}{:,3}]);
...else
...    means2(subj)        = 0;
```

Group Processing (main_eeglab_subject_processing.m)

Special variables

These variables are the only allowed to exist outside project. They are passed to each project function as varargin and their are used only if filled. They can thus be empty.

list_select_subjects

In order to select the subjects to be processed, several options are feasible:

Use all the subjects defined in PS

1) //list_select_subjects = []
2) list_select_subjects = []
3) list_select_subjects = project.subjects.list

Consider only some subjects

4) define their label in a cell array of strings
   list_select_subjects ={'ID01', 'ID08', 'ID12',}

design_num_vec

Is a array of 1-based integer, indicating which element of the *project.design* struct array you want to process

```
design_num_vec = [];                => Process ALL designs
design_num_vec = [1, 3, 6];         => Process only these designs
```

stat_analysis_suffix

mask_coef

stat_freq_bands_list

1) Local Paths

In the first section of the pipeline we specify the localization/path where the information required for running the analyses is located.
Depending on the system we are using (Linux or Windows), we have to complete the information in the first section (under "if), or the second one (under "else"). If we are working in Linux, we need to modify the scripts included in the first part, below the sentence starting with "if"; but if we are working on another system, we would do the same but below the part starting with "else".

For the paths to be set up:

- **project.paths.projects_data_root** → we copy the path where our folder "PROJECTS_DATA_ROOT" is placed.
- **project.paths.projects_scripts_root** → we copy the path where our folder "PROJECTS_SCRIPT_ROOT" is placed.
- **project.paths.plugins_root** → we copy the path where our toolbox of Matlab is placed.

2) Project Data

Here we identify the project we are working on, as well as some general information about it.

- **project.research_group** → we write the name of the research group we are working with in this project.
- **project.research_subgroup** → we write the name of the research subgroup involved in this study, or our name.
- **project.name** → we name the project, but it must be the same as in the 'project.paths.local_projects_data' subfolder name
- **project.conf_file_name** → we specify the name of the project_structure file we are using in this study.

As an example:

```
project.research_group = 'Experimental Psychology';
project.research_subgroup = 'CDLab';
project.name = 'morality';
project.conf_file_name = 'Moral_project_structure';
```

3) Design Specification

4) Project Structure and File System Initialization

```
project.paths.script.eeg_tools_project = fullfile(project.paths.global_scripts_root, 'eeg_tools',
'project', '');
addpath(project.paths.script.eeg_tools_project);
project = project_init(project);
```

5) Override

Similarly to the Subject Processing document, in the Overrride section we are able to specify the subjects for running the analyses, or the frequency band of interest.
list_select_subjects → select the subjects to be processed. The options are: (1) commented; (2) set to []; (3) set to a cell array of strings; (4) set to project.subjects.list
For example:

```
list_select_subjects = project.subjects.list;
or
```

```
list_select_subjects = {'CC_01_vittoria', 'CC_02_fabio', 'CC_03_anna', 'CC_04_giacomo',
'CC_05_stefano', 'CC_06_giovanni', 'CC_07_davide', 'CC_08_jonathan', 'CC_09_antonella',
'CC_10_chiara', 'CP_01_riccardo', 'CP_02_ester', 'CP_03_sara', 'CP_04_matteo', 'CP_05_gregorio',
'CP_06_fernando', 'CP_07_roberta', 'CP_08_mattia', 'CP_09_alessia', 'CP_10_livia'};
```

Next two commands are aimed to select a specific band to focus statistics in ERSP time-frequency:
mask_coef →
stat_freq_bands_list →

6) Operation List

Here we have all the different operations we are able to run with the pipeline. Following the same structure than in the Subject Processing document, if we set the command to 0, the operation will not be executed; if we set the command to 1, the program will run that operation:
project.operations.do_study → create a study from preprocessed datasets.
project.operations.do_group_averages → create one merged epochs file for each condition and group in the study.
project.operations.do_design → create a set of experimental designs within a created study.
project.operations.do_study_compute_channels_measures → precompute channels measures for a set of designs: these measures are used in group statistics. It is mandatory if some dataset of the study is changed.

ERP analysis

There are different types of ERP analyses (see Plots and Analyses document) run by this pipeline. Here we show the scripts for analyses based on curve, evaluating and representing standard

ERP CURVE

EEGLab statistics on the curve of ERP, plot together levels of design factors:
project.operations.do_study_plot_roi_erp_curve_continous → analyzes and plots of ERP curve for all time points.
project.operations.do_study_plot_roi_erp_curve_tw_group_noalign → performs (and saves) statistics based on grand-mean of subjects within group time windows.
project.operations.do_study_plot_roi_erp_curve_tw_group_align → adjusts the group time windows to time windows which are re-aligned to the latencies of time window extrema.
project.operations.do_study_plot_roi_erp_curve_tw_individual_noalign → performs (and saves) statistics based on individual subjects within time windows.
project.operations.do_study_plot_roi_erp_curve_tw_individual_align → adjusts the group time windows to time windows which are re-aligned to the latencies of time window extrema.

For EOG, we have the same operations:
project.operations.do_study_plot_roi_erpeog_curve_continous
project.operations.do_study_plot_roi_erpeog_curve_tw_group_noalign
project.operations.do_study_plot_roi_erpeog_curve_tw_group_align
project.operations.do_study_plot_roi_erpeog_curve_tw_individual_noalign
project.operations.do_study_plot_roi_erpeog_curve_tw_individual_align

For EMG, we have the same operations:
project.operations.do_study_plot_roi_erpemg_curve_continous
project.operations.do_study_plot_roi_erpemg_curve_tw_group_noalign
project.operations.do_study_plot_roi_erpemg_curve_tw_group_align
project.operations.do_study_plot_roi_erpemg_curve_tw_individual_noalign
project.operations.do_study_plot_roi_erpemg_curve_tw_individual_align

Next scripts are for the analyses based on standard topographical time windows, analyzing and plotting of ERP topographical maps for the selected bands in the time windows of the selected design.
project.operations.do_study_plot_erp_topo_tw_group_noalign → performs (and saves) additional statistics based on grand-mean of subjects within time windows,
project.operations.do_study_plot_erp_topo_tw_group_align → adjusts the group time windows to time windows which are re-aligned to the latencies of time window extrema.
project.operations.do_study_plot_erp_topo_tw_individual_noalign → performs (and saves) additional statistics based on individual subjects within time windows.
project.operations.do_study_plot_erp_topo_tw_individual_align → adjusts the group time windows to time windows which are re-aligned to the latencies of time window extrema.
Now there are the same scripts but for the compact topographical analyses:
project.operations.do_study_plot_erp_topo_compact_tw_group_noalign → performs (and saves) additional statistics based on grand-mean of subjects within time windows,
project.operations.do_study_plot_erp_topo_compact_tw_group_align → adjusts the group time windows to time windows which are re-aligned to the latencies of time window extrema.
project.operations.do_study_plot_erp_topo_compact_tw_individual_noalign → performs (and saves) additional statistics based on individual subjects within time windows.
project.operations.do_study_plot_erp_topo_tw_compact_individual_align → adjusts the group time windows to time windows which are re-aligned to the latencies of time.

project.proj_eeglab_study_plot_allch_erp_time → evaluates and represents ERP of all channels as a function of time and compare different conditions in a time x channels space (TANOVA).
project.operations.do_study_plot_allch_erp_cc_time → evaluates and represents cross correlation of ERP of all channels between levels of one factor as a function of time and compares different levels of the other factor in a time x channels space (TANOVA).

ERSP analysis

For time frequency analysis, which evaluate and represent standard EEGLab statistics on the time-frequency distribution of ERSP, there are several options:
project.operations.do_study_plot_roi_ersp_tf_continuous
project.operations.do_study_plot_roi_ersp_tf_decimate_times → decimates times.
project.operations.do_study_plot_roi_ersp_tf_decimate_freqs → decimates frequencies.
project.operations.do_study_plot_roi_ersp_tf_decimate_times_freqs → decimates time and frequencies.
project.operations.do_study_plot_roi_ersp_tf_tw_fb → time-frequency distribution freely binned in the frequency and/or in the time domain.

The curve ERPS analysis, which evaluate and represent standard EEGLab statistics on the curve of ERSP in a selected frequency, and plots together levels of design factors, the operations are the same as for the curve ERP:

project.operations.do_study_plot_roi_ersp_curve_continous_standard
project.operations.do_study_plot_roi_ersp_curve_tw_group_noalign_standard
project.operations.do_study_plot_roi_ersp_curve_tw_group_align_standard

project.operations.do_study_plot_roi_ersp_curve_tw_individual_noalign_standard
project.operations.do_study_plot_roi_ersp_curve_tw_individual_align_standard

And the compact version of the operations:
project.operations.do_study_plot_roi_ersp_curve_continous_compact
project.operations.do_study_plot_roi_ersp_curve_tw_group_noalign_compact
project.operations.do_study_plot_roi_ersp_curve_tw_group_align_compact
project.operations.do_study_plot_roi_ersp_curve_tw_individual_noalign_compact
project.operations.do_study_plot_roi_ersp_curve_tw_individual_align_compact

We have the same analyses for EOG:
project.operations.do_study_plot_roi_erspeog_curve_continous
project.operations.do_study_plot_roi_erspeog_curve_tw_group_noalign
project.operations.do_study_plot_roi_erspeog_curve_tw_group_align
project.operations.do_study_plot_roi_erspeog_curve_tw_individual_noalign

project.operations.do_study_plot_roi_erspeog_curve_tw_individual_align

As well as for the EMG:
project.operations.do_study_plot_roi_erspemg_curve_continous
project.operations.do_study_plot_roi_erspemg_curve_tw_group_noalign
project.operations.do_study_plot_roi_erspemg_curve_tw_group_align
project.operations.do_study_plot_roi_erspemg_curve_tw_individual_noalign
project.operations.do_study_plot_roi_erspemg_curve_tw_individual_align

Following the same structure than the ERP analyses, next four scripts are for the standard topographical analyses, and the last four are for the compact topographical analyses:
project.operations.do_study_plot_ersp_topo_tw_fb_group_noalign
project.operations.do_study_plot_ersp_topo_tw_fb_group_align
project.operations.do_study_plot_ersp_topo_tw_fb_individual_noalign
project.operations.do_study_plot_ersp_topo_tw_fb_individual_align

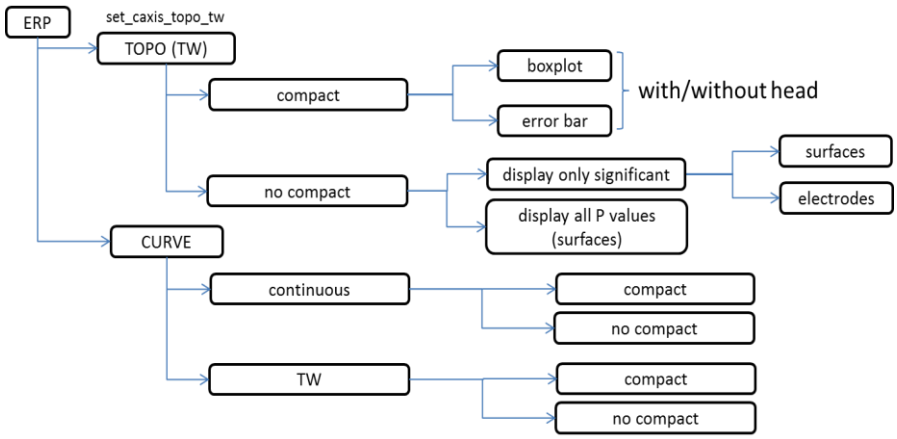
project.operations.do_study_plot_ersp_topo_tw_fb_group_noalign_compact
project.operations.do_study_plot_ersp_topo_tw_fb_group_align_compact
project.operations.do_study_plot_ersp_topo_tw_fb_individual_noalign_compact
project.operations.do_study_plot_ersp_topo_tw_fb_individual_align_compact

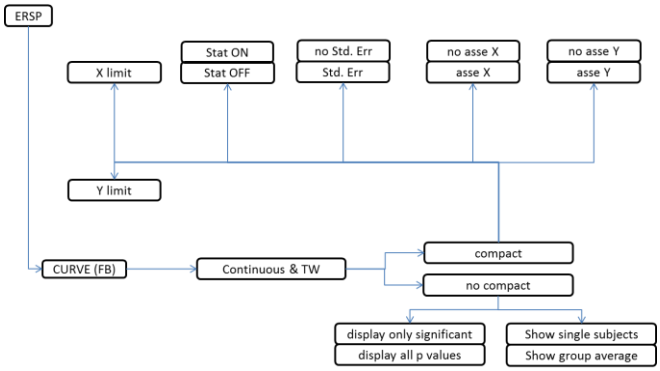
project.operations.do_eeglab_study_export_ersp_tf_r →

Plots and Groups Analyses

ERP

1. Curve
   1.1. Continuous
   1.2. Curve TW
      1.2.1. Group, no-align.
      1.2.2. Group, align.
      1.2.3. Individual, no-align.
      1.2.4. Individual, align.
2. Topographical plot
   2.1. Standard
      2.1.1. Group, no-align
      2.1.2. Group, align.
      2.1.3. Individual, no-align.
      2.1.4. Individual, no-align.
   2.2. Compact
      2.2.1. Group, no-align.
      2.2.2. Group, alignment.
      2.2.3. Individual, no-align.
      2.2.4. Individual, no-align.





Curve

1. **Continuous**: analyzes and plots ERP curve for all the time points. The pipeline gives you a plot for each ROI and condition comparing your groups, and a plot for each group comparing the different conditions.

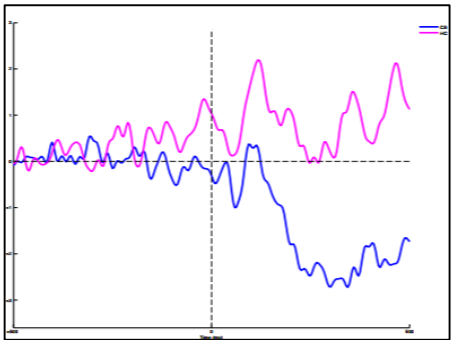


Fig 1. Example of a continuous curve ERP

2. **Curve time window (TW):** analyzes and plots ERP based on time windows of the selected design. For the design, there are four options depending on the alignment versus non-alignment and individual versus group. As in the curve plots, the pipeline provides you with a plot for each ROI and condition comparing your groups, and a plot for each group comparing the different conditions.
   - **2.1.** TW group and non-alignment: perform and save additional statistics based on grand-mean of subjects within TW.
   - **2.2.** TW group and alignment: perform and save additional statistics based on grand-mean of subjects within TW, adjusting the group time windows to time windows which are re-aligned to the latencies of the time window extrema.
   - **2.3.** TW individual and non-alignment: perform and save additional statistics based on individual subjects within TW.
   - **2.4.** TW individual and non-alignment: perform and save additional statistics based on grand-mean of subjects within TW, adjusting the individual time windows to time windows which are re-aligned to the latencies of the time window extrema.

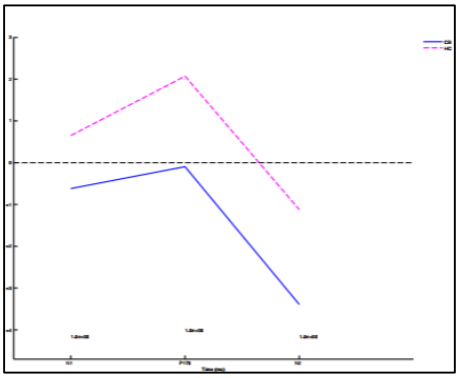


Fig 2. Example of a TW individual aligned ERP plot.

Topographical Plot

2.5. Standard:

analyzes and plots the ERP topographical maps for the selected bands in the time windows of the selected design. The pipeline creates two folders: one with the plots of the channels, another for the ROIs. In all the plots, the last row and column are aimed to indicate the significant results. As previously, there are four options for the design:

- **2.5.1. TW group and non-alignment**: perform and save additional statistics based on grand-mean of subjects within TW.
- **2.5.2. TW group and alignment**: perform and save additional statistics based on grand-mean of subjects within TW, adjusting the group time windows to time windows which are re-aligned to the latencies of the time window extrema.
- 2.5.3. **TW individual and non-alignment**: perform and save additional statistics based on individual subjects within TW.
- 2.5.4. **TW individual and non-alignment**: perform and save additional statistics based on grand-mean of subjects within TW, adjusting the individual time windows to time windows which are re-aligned to the latencies of the time window extrema.

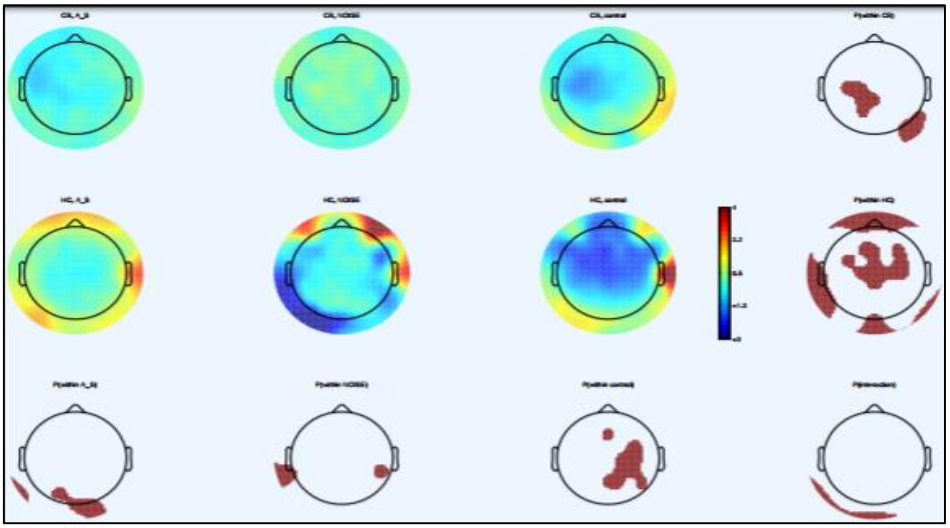


Fig 3. Example of a TW individual alignment standard topographical ERP plot.

2.5.5. Compact: performs and saves statistics based on grand-mean of subjects within time windows.

2.5.5.1. TW group and non-alignment: perform and save additional statistics based on grand-mean of subjects within TW.

2.5.5.2. TW group and alignment: perform and save additional statistics based on grand-mean of subjects within TW, adjusting the group time windows to time windows which are re-aligned to the latencies of the time window extrema.

2.5.5.3. TW individual and non-alignment: perform and save additional statistics based on individual subjects within TW.

2.5.5.4. TW individual and non-alignment: perform and save additional statistics based on grand-mean of subjects within TW, adjusting the individual time windows to time windows which are re-aligned to the latencies of the time window extrema.

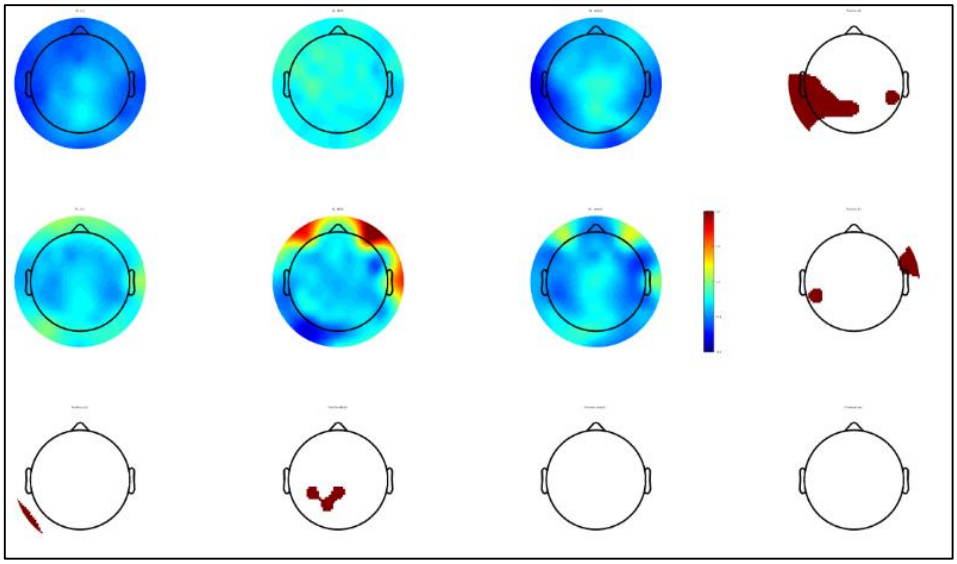
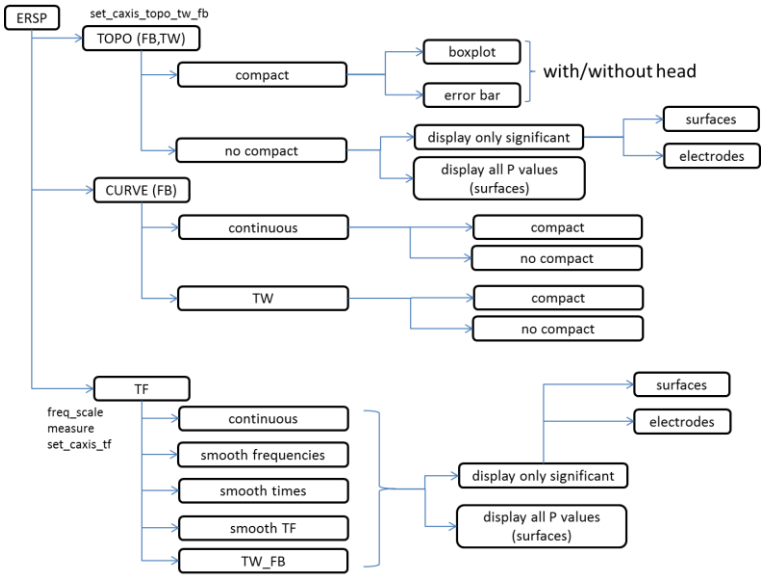


Fig 4. Example of a TW individual alignment compact topographical ERP plot.

ERSP

1. Time-frequency (TF)
   1.1. Continuous.
   1.2. Dec. T.
   1.3. Dec. F.
   1.4. Dec. TF.
   1.5. Dec. TW-FB
2. Curve
   2.1. Continuous
   2.2. Curve TW-FB
      2.2.1. Group, no-align.
      2.2.2. Group, align.
      2.2.3. Individual, no-align.
      2.2.4. Individual, align.
3. Topographical plot
   3.1. Standard
      3.1.1. Group, no-align
      3.1.2. Group, align.
      3.1.3. Individual, no-align.
      3.1.4. Individual, no-align.
   3.2. Compact
      3.2.1. Group, no-align.
      3.2.2. Group, alignment.
      3.2.3. Individual, no-align.
      3.2.4. Individual, no-align.



1. Time-frequency: standard ERSP time-frequency (TF), which evaluates and represents standard EEGLab statistics on the time-frequency distribution of ERSP. By using the time-frequency, there are five analyses/plots depending if you treat the ERSP as a continuous or you prefer to decimate the time, frequency, or both.

1.1. Continuous

1.2. Dec. time: decimates the time.

1.3. Dec. frequency: decimates the frequency.

1.4. Dec. time-frequency: decimates the time and frequency.

1.5. Dec. TW frequency band: time-frequency distribution freely binned in the frequency and/or in the time domain.

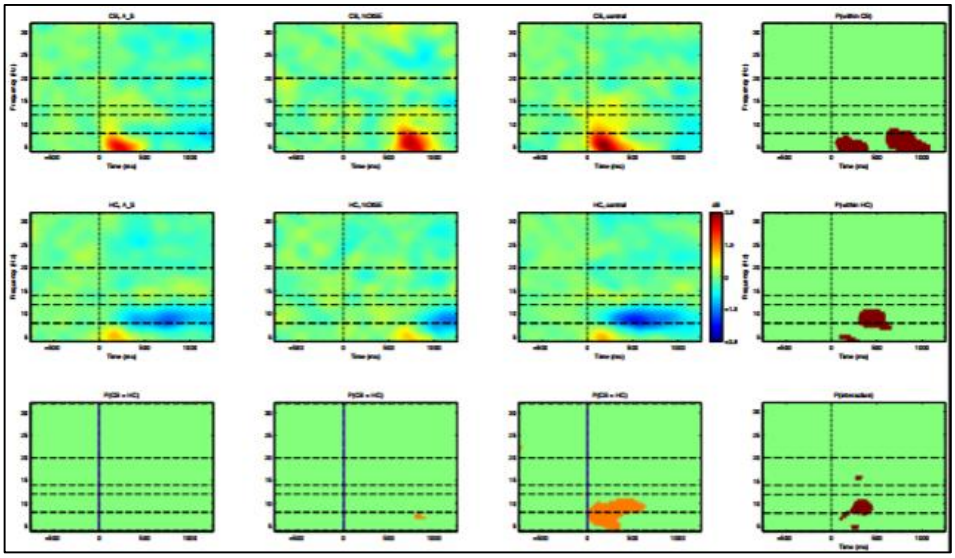


Fig 5. Example of a continuous TF ERPS plot.

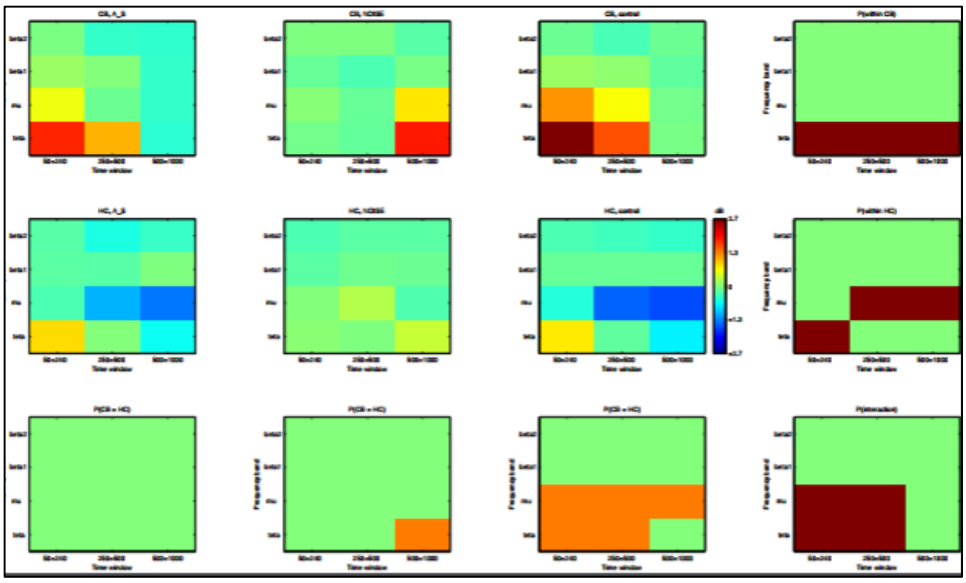


Fig 6. Example of a TW-FB TF ERSP plot.

2. Curve: evaluates and represents standard EEGLab statistics on the curve of the ERSP in a selected frequency, plot together levels of design factors.

2.1. Standard

2.1.1. Continuous

2.1.2. TW group and non-alignment: perform and save additional statistics based on grand-mean of subjects within TW.

2.1.3. TW group and alignment: perform and save additional statistics based on grand-mean of subjects within TW, adjusting the group time windows to time windows which are re-aligned to the latencies of the time window extrema.

2.1.4. TW individual and non-alignment: perform and save additional statistics based on individual subjects within TW.

2.1.5. **TW individual and non-alignment**: perform and save additional statistics based on grand-mean of subjects within TW, adjusting the individual time windows to time windows which are re-aligned to the latencies of the time window extrema.

2.2. Compact

2.2.1. **Continuous**

2.2.2. **TW group and non-alignment**: perform and save additional statistics based on grand-mean of subjects within TW.

2.2.3. **TW group and alignment**: perform and save additional statistics based on grand-mean of subjects within TW, adjusting the group time windows to time windows which are re-aligned to the latencies of the time window extrema.

2.2.4. **TW individual and non-alignment**: perform and save additional statistics based on individual subjects within TW.

2.2.5. **TW individual and non-alignment**: perform and save additional statistics based on grand-mean of subjects within TW, adjusting the individual time windows to time windows which are re-aligned to the latencies of the time window extrema.

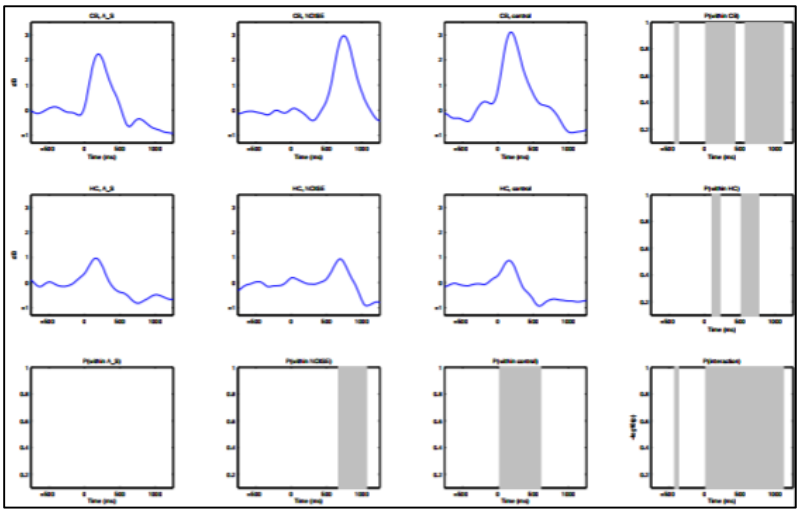


Fig 7. Example of a continuous standard curve ERSP plot.

2.3. Topographical TW FB: evaluates and represents standard EEGLab statistics on the curve of ERSP in a selected frequency, plot together levels of design factors.

2.3.1. **Standard**

2.3.1.1. **TW group and non-alignment**: perform and save additional statistics based on grand-mean of subjects within TW.

2.3.1.2. **TW group and alignment**: perform and save additional statistics based on grand-mean of subjects within TW, adjusting the group time windows to time windows which are re-aligned to the latencies of the time window extrema.

2.3.1.3. **TW individual and non-alignment**: perform and save additional statistics based on individual subjects within TW.

2.3.1.4. **TW individual and non-alignment**: perform and save additional statistics based on grand-mean of subjects within TW, adjusting the individual time windows to time windows which are re-aligned to the latencies of the time window extrema.

2.3.2. **Compact**

2.3.2.1. **TW group and non-alignment**: perform and save additional statistics based on grand-mean of subjects within TW.

2.3.2.2. **TW group and alignment**: perform and save additional statistics based on grand-mean of subjects within TW, adjusting the group time windows to time windows which are re-aligned to the latencies of the time window extrema.

2.3.2.3. **TW individual and non-alignment**: perform and save additional statistics based on individual subjects within TW.

2.3.2.4. **TW individual and non-alignment**: perform and save additional statistics based on grand-mean of subjects within TW, adjusting the individual time windows to time windows which are re-aligned to the latencies of the time window extrema.

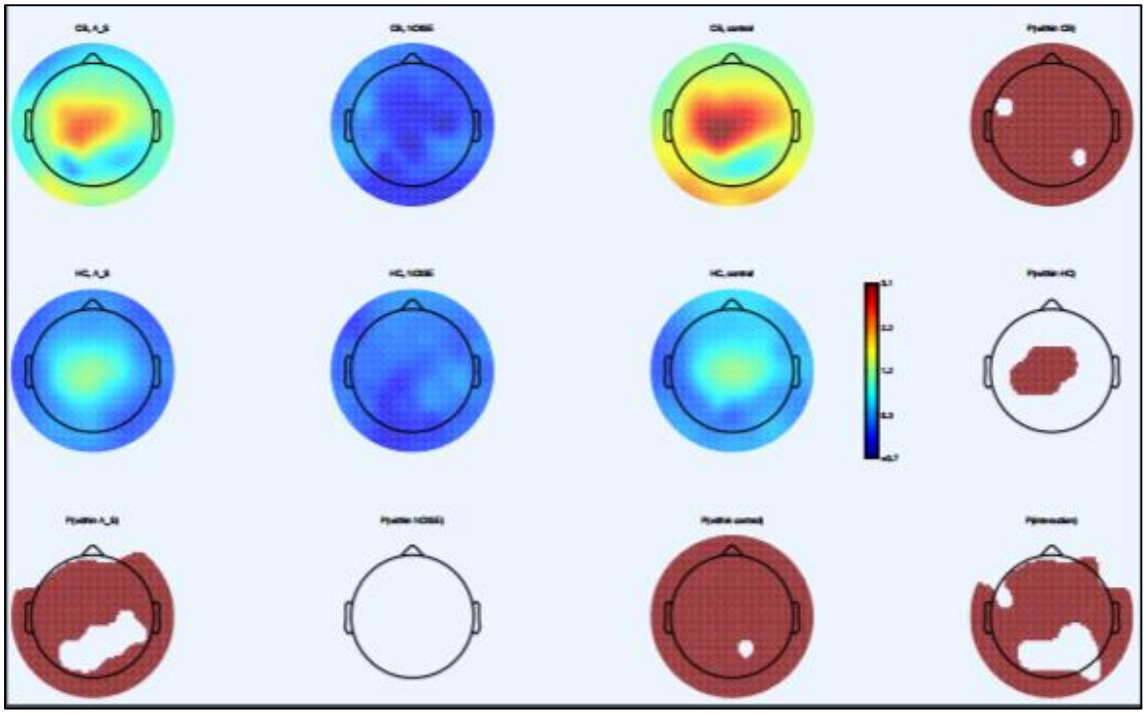
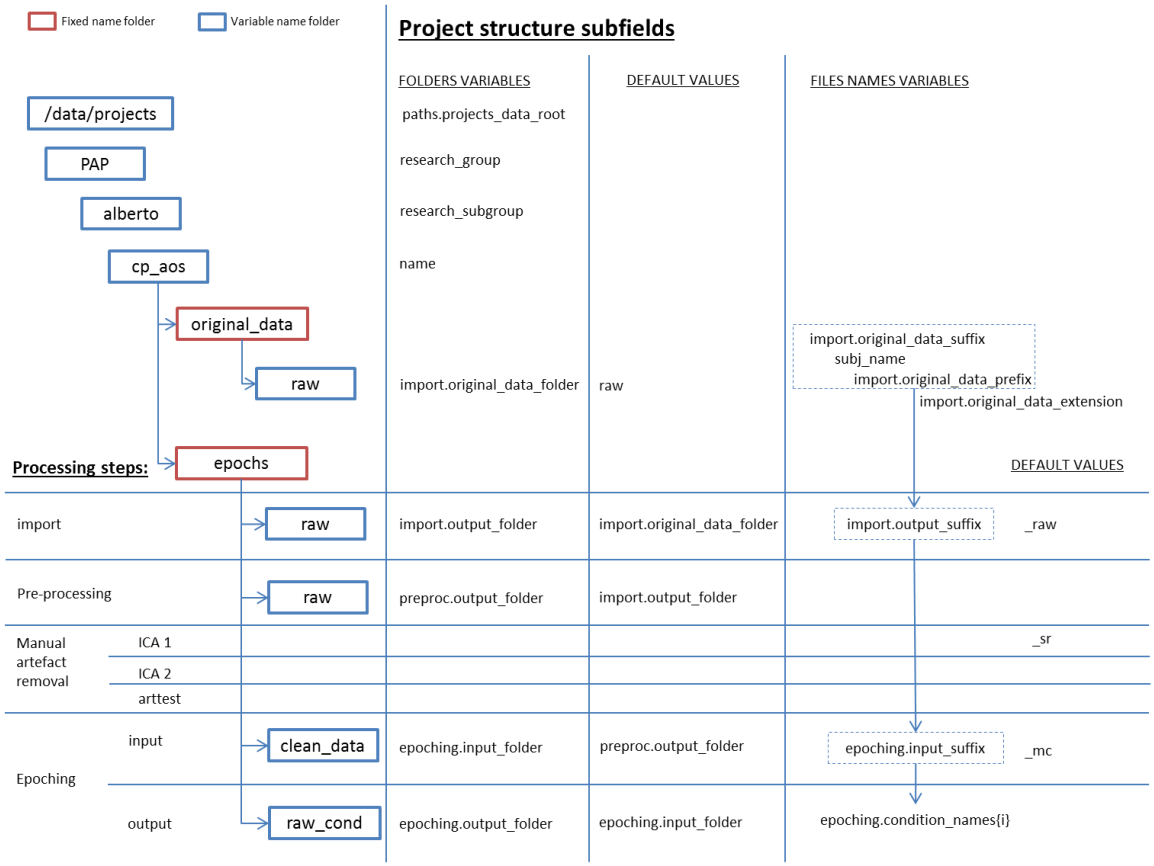


Fig 8. Example of a TW individual alignment standard topographical ERSP plot.

Appendix

File system



Results structure

eeglab_study_plot_find_extrema_avg

datatw.curve{nf1, nf2}	nsubj	MEAN per ogni soggetto nella finestra
datatw.curve_tw{nf1, nf2}	nsubj x ntp	serie temporale per ogni soggetto nella finestra
datatw.extr_avg{nf1, nf2}	1	EXTR calcolato sulla serie temporale media di gruppo
datatw.extr_lat_avg{nf1, nf2}	1	LATENCY of EXTR calcolato sulla serie temporale media di gruppo
datatw.extr_lat_avg_vec{nf1, nf2}	N	latencies in cui raggiungi l'EXTR

INDIVIDUAL NO ALIGN (eeglab_study_plot_find_extrema_gru)

datatw.curve{nf1,nf2}	nsubj	MEAN per ogni soggetto nella finestra
datatw.curve_tw{nf1,nf2}	nsubj x ntp	serie temporale per ogni soggetto nella finestra

datatw.extr_gru{nf1,nf2}	nsubj	EXTR per ogni soggetto nella finestra
datatw.extr_gru_mean{nf1,nf2}	1	MEAN di ampiezza nella finestra
% datatw.extr_gru_sd{nf1,nf2}	1	SD di ampiezza nella finestra
% datatw.extr_gru_median{nf1,nf2}	1	MEDIAN di ampiezza nella finestra
% datatw.extr_gru_range{nf1,nf2}	2	MAX e MIN delle ampiezze tra tutti i soggetti
% datatw.extr_lat_gru{nf1,nf2}	nsubj	EXTR LATENCY (in funzione pero del criterio di occorrenza definito nel project config....e.g la prima occorrenza o la media delle occorrenze...)
% datatw.extr_lat_gru_mean{nf1,nf2}	1	MEAN di latenza nella finestra
% datatw.extr_lat_gru_sd{nf1,nf2}	1	SD di latenza nella finestra
% datatw.extr_lat_gru_median{nf1,nf2}	1	MEDIAN di latenza nella finestra
% datatw.extr_lat_gru_range{nf1,nf2}	2	MAX e MIN delle latenza tra tutti i soggetti
% datatw.extr_lat_gru_vec{nf1,nf2}	nsubj x N	vettore con tutte le occorrenze dell'estremo per ogni soggetto

INDIVIDUAL ALIGN (eeglab_study_plot_find_extrema_single)

datatw.curve{nf1,nf2}	nsubj	MEAN per ogni soggetto nella finestra
datatw.curve_tw{nf1,nf2}	nsubj x ntp	serie temporale per ogni soggetto nella finestra
datatw.extr_single{nf1,nf2}	nsubj	EXTR per ogni soggetto nella finestra
datatw.extr_single_mean{nf1,nf2}	1	MEAN di ampiezza nella finestra
datatw.extr_single_sd{nf1,nf2}	1	SD di ampiezza nella finestra
datatw.extr_single_median{nf1,nf2}	1	MEDIAN di ampiezza nella finestra
datatw.extr_single_range{nf1,nf2}	2	MAX e MIN delle ampiezze tra tutti i soggetti
datatw.extr_lat_single{nf1,nf2}	nsubj	EXTR LATENCY (in funzione pero del criterio di occorrenza definito nel project config....e.g la prima occorrenza o la media delle occorrenze...)
datatw.extr_lat_single_mean{nf1,nf2}	1	MEAN di latenza nella finestra
datatw.extr_lat_single_sd{nf1,nf2}	1	SD di latenza nella finestra

datatw.extr_lat_single_median{nf1,nf2}	1	MEDIAN di latenza nella finestra
datatw.extr_lat_single_range{nf1,nf2}	2	MAX e MIN delle latenza tra tutti i soggetti
datatw.extr_lat_single_vec{nf1,nf2}	nsubj x N	vettore con tutte le occorrenze dell'estremo per ogni soggetto
datatw.extr_pattern_subject_single{nf1,nf2}	nsubj x ntp	inizializzata a NaN, per ogni soggetto contiene i valori della serie temporale nelle posizioni riallineate
datatw.extr_pattern_lat_range_single{nf1,nf2}	nsubj x 2	inizio e fine della finestra riallineata