Basic manual EEG preprocessing with EEGLAB and Matlab

Joram van Driel

July 6, 2015

Contents

1 Noise and signal

Raw EEG is a noisy signal. But how do you know what is noise, and what is your signal of interest? Preprocessing your data in the optimal way is a crucial step in getting clean data, thereby increasing your signal-to-noise ratio. Unfortunately, when your data were bad to begin with, preprocessing will not "change" these bad data into good data. Another problem with preprocessing is that there is no golden standard. This document describes a guideline to preprocessing based on combined personal experience.

This manual will assume an EEG equipment of BioSemi 64 electrodes, with the labels according to the 10-20 system. For the preprocessing steps, EEGLAB version 12 was used, but of most of these functions were/are preserved across versions, or are very easily adapted to meet version changes. The first steps described below are part of the `preICA.m` document that can be found at GitHub.

2 From raw, continuous EEG to filtered, epoched EEG

2.1 Epoch windows

Collected EEG data is continuous: even though your experiment (probably) consisted of trials, before the start of the experiment you pressed "record" and at the end "stop", which means you have one enormously long EEG "trial" of the same duration as your experiment. To be able to do EEG analyses, you need to cut the EEG into trials that correspond to the trials in your experiment. However, these *epochs* are not *exactly* the same as the trials from your experiment: they need to contain some extra time around the trial. If you are absolutely sure that your are only going to do ERP analysis, these "buffer zones" can be fairly small, but if you want to do time-frequency analyses, they should be longer, because of the issue of edge artifacts. Because you are always going to cut-off the long buffer zones in the end, whether you did ERP or time-frequency analyses, it is recommended to take long buffer zones. In fact, it also has an additional advantage, which will be clear below.

As a rule of thumb, add a full second to the earliest and latest time point in a trial, relative to the onset of the main stimulus of interest. For example, if you have a 100 ms stimulus, which requires an immediate response, and RT's are usually around 500 ms, and you want to leave in the option of doing RT-locked analysis in a later step, you can take an epoch window of -1.5 to 2.5 seconds, surrounding stimulus onset. The reasoning here is that you want to be able to analyze 500 ms pre-stimulus, up to about 1000 ms post-response. Note that already at this very early step, you need to make decisions that are neither trivial, nor do they have one right or wrong answer. Just think logical about your experiment, the timing of your stimuli, and the analysis that you want to do. The `epochtime` is defined on the top of `preICA.m`.

```
1 epochtime=[ -1.5 2.5];
```

2.2 Importing raw data

Biosemi raw data files have the `.bdf` extension. If you let Matlab make a list of all `.bdf` files in your folder that contains the raw data, you can loop over these files. EEGLAB has a default function that imports Biosemi data; if you direct Matlab into the folder that contains the raw data, your file-loop will look something like:

```
filz=dir('*.bdf'); % find all bdf files

%% big loop around raw data files

for filei=1:length(filz)

    %% load data with eeglab function
    EEG = pop_biosig(filz(filei).name);
    ...
```

Importing a `.bdf` file will give some warnings, which you can ignore. In the command window, you can type in `EEG` to see that it is a structure with different fields, which contain information about the file, the channels, and which contain the data. Note that at this point, `EEG.data` is a channel-by-time matrix.

2.3 Rereferencing

The next step is to rereference the data. The voltage measured at each electrode can only be interpreted if it has a reference: in this way, voltage reflects the *potential* of current to flow from the electrode to the reference location (it is slightly more complicated than this: voltage at an *active* electrode reflects the *difference* between the potential of current to go from the active electrode to the *ground*, and the potential of current to go from the reference to the ground; see chapter 3 in this book by Steve Luck; a highly recommended read!). What you choose as a reference, is again non-trivial, and requires a choice, which may differ per lab and EEG equipment. In general, low-density EEG nets (e.g. 64 or 32 systems) are rereferenced to two bilateral reference locations that are on the head (i.e. close to the active channels), but of which you can safely assume that they don't (or hardly) pick up brain signals; this way, referencing also removes noise that is common to both the reference and the active channels. Two such reference locations are common: the mastoids (the bones behind the ears), and the earlobes. The main argument for using earlobes is that it will less likely pick up brain signals; the main argument for using mastoids is that these are closer to the active channels. Both arguments are valid, although in my experience, mastoids give a slightly more noisy signal than earlobes. For high-density EEG, it is advisable to use an average reference: the average activity of all channels is used as a reference, so the voltage at a channel is always relative to this average. The EEGLAB code to do rereferencing is:

```
% re-reference to linked mastoids
EEG = pop_reref(EEG,...
    [find(strcmpi('EXG1',{EEG.chanlocs.labels})) ...
    find(strcmpi('EXG2',{EEG.chanlocs.labels}))],...
    'refstate','averef');
```

The labels `'EXG1'` and `'EXG2'` are arbitrary, and depend on which external electrodes you attached on the subjects earlobes/mastoids. Make sure you specify the correct ones in your code.

In the `preICA.m` file, there is also some other rereferencing, of external electrodes that measured other physiological signals: horizontal and vertical EOG (eye movements), and EMG (muscles). These signals were bipolar (e.g. for horizontal EOG, you placed an electrode on the left and right side of the left and right eye, respectively); by subtracting one from the other, you get the relative signal (e.g. did the subject make an eye movement to the left or the right). After all necessary rereferencing, the channels that contain redundant information (the mastoids/earlobes, and one of the two bipolar recordings) are removed from the data. The EOG/EMG is then also renamed for convenience.

2.4 High-pass filtering

The next step is to apply a filter to get rid of noise that is characterized by a particular dominant frequency. Raw EEG usually contains slow drifts; to remove this, we apply a *high-pass* filter (a filter that passes through high-frequencies and removes low-frequencies), usually with a cut-off of 0.5 Hz. Again, this is a point where you need to make a decision. For ERP analyses it is common to also apply a low-pass filter, because ERPs are characterized by frequencies below, say, at least 30 Hz. Why not apply this filter later? That is of course possible, but it is better to apply filters on continuous data, because of edge artifacts (see above). However, if you have set the epoch windows to be fairly long, you could do a low-pass filter afterwards, without much distortion around your time windows of interest. Note that using a high- and low-pass filter at once is the same as using a band-pass filter. Also note that if you want to leave in the option of doing time-frequency analyses, you should *not* do a low-pass filter at this step (or, if anything, use a high cut-off of say 100 Hz or higher). Applying a high-pass filter is done with the following EEGLAB function, where the first number is the high-pass cut-off, and the second number (here, 0) is the low-pass cut-off. A high-pass filter with this function takes quite some time:

```
% high-pass filter: removes < 0.5 Hz slow-wave drifts
EEG = pop_eegfilt(EEG,.5,0);
```

Of note, the high-pass filter is a good example of how in one situation something is considered "noise" while in another situation the same type of data is "signal". In EEG sleep research, slow waves, or even ultra-slow-waves have frequency characteristics of < 1 Hz. You would then obviously choose different settings at this step.

2.5 Epoching

Now you are ready to epoch the continuous EEG data into trials with buffer zones. For this, you need to know how you specified the triggers that were sent from you experiment software to the EEG recording. You probably sent multiple triggers in a trial, e.g. one for every stimulus, and one for the response. However, you want to center the epoch around one type of stimulus for every trial, such that after this step, time 0 in each EEG epoch corresponds to the onset of this stimulus. Look up in your experiment code how you handled the triggers, and to which trigger(s) you want to "lock" the epochs. Then, you simply do (below with example trigger 11):

```
1 EEG = pop_epoch(EEG,{'11'},[epochtime(1) epochtime(2)]);
```

Be careful: sometimes (depending on the settings in the lab), triggers can get weird, high values added to them; you then need to figure out how to get from these values to the values you specified in your experiment code.

Note that you have used your `epochtime` variable at this step. In the command window the output will show you how many epochs were generated. It is good to check this, to see if it matched the number of trials your experiment consisted of (which it should). On very rare occasions, the subject pressed a button *exactly* at the time the stimulus-trigger was sent; the button-press-trigger and stimulus-trigger may then add up, resulting in a weird trigger number that you haven't specified in the `pop_epoch` function. As a result the total number of epochs may sometimes be a bit lower. When you get a selected epoch number that is way too large or too small, then something went wrong (for example, with how the "raw" trigger values corresponded to your prespecified trigger numbers; see above).

2.6 Linear average baseline subtraction

After *or* before epoching, another step is required. By subtracting each channel's average activity over the entire epoch from every time point in the epoch (or the channel-specific average of the entire dataset from each time point in continuous EEG), the activity will fluctuate around 0 μV. This is another way to get rid of drifts (in addition to high-pass filtering) and other non-stationarities in the data, and to make all epochs comparable with respect to the scaling of the amplitudes. Note that this baseline subtraction is different from event-related baseline subtraction, for which you would use a pre-stimulus baseline (for example from -200 to 0 ms related to stimulus onset). A paper by Groppe, Makeig and Kutas (2009) argues that whole-epoch baseline correction improves Independent Component Analysis (ICA, see below) and should thus be preferred over pre-stimulus baseline correction. After all preprocessing steps, pre-stimulus baseline correction is still necessary in order to compute the ERP.

Another thing to note is that because this is a simple, linear subtraction, it doesn't change the actual data, except for the fact now every epoch has a zero mean. This is different from, e.g., baseline correction after time-frequency analysis, where you do, e.g., a dB conversion of trial-averaged power relative to trial-average power in a baseline time window; such baseline correction changes the interpretation and representation (the unit) of the data.

As stated above, linear baseline subtraction can be done for every single trial (epoch), as described in Groppe et al. (2009). However, another option is to apply the correction on the continuous data, and *then* epoch your data. Based on the argument that it is best to apply ICA on data that is as close to the raw data as possible, this latter option could be preferred. The EEGLAB code to do whole-epoch/whole-data baseline subtraction is very simple:

```
% baseline-correct
EEG = pop_rmbase(EEG,[]);
```

The rest of the `preICA.m` script adds some channel and event information to the EEG structure. Because the high-pass filtering takes so long, and because all the steps described so far are fully automatic, while the next steps require manual choices and visual inspection, you could loop over all files until you reach this part and save the EEG files, with the suffix `_preReject.mat`. Then, move to the artifact rejection.

3 Artifacts: from rejected epochs to removed independent components

The filtered, epoched EEG data still contains unwanted sources of noise. The most common ones are: eye blinks, eye movements, muscle (EMG) artifacts, and noise in specific channels that weren't well attached. Here, preprocessing becomes very tricky; what follows below is only a recommendation. It might very well be that in a few years, I have changed my mind about some of the following steps. It would be good to keep the discussion going about pro's and con's of several options in artifact rejection, so feel free to criticize the below.

3.1 Semi-automatic rejection of EMG-contaminated epochs

Visual inspection of the epochs and the functionality of how you can mark epochs for rejection, is one of the strengths of EEGLAB. It is in this regard superior to other packages like Fieldtrip, and MNE (Python). As you will see later, the ICA functionality, and the inspection of components that capture eye blinks, eye movements, etc., is another advantage of EEGLAB over other toolboxes. However, inspecting every trial to manually mark contaminated epochs is tedious, and you run the risk of changing your personal threshold of accepting good trials and rejecting bad trials; this can happen across subjects, but also within subjects. One downside of EEGLAB is that its automatic trial rejection algorithms still take quite a long time, and result (in my opinion) in too many false positives and negatives. Here, Fieldtrip can be of help. Below, an approach is described that combines Fieldtrip with EEGLAB, to "semi-automatically" reject epochs, which (one could argue) has best of both worlds.

Fieldtrip has a tutorial website where they explain their philosophy behind the artifact rejection algorithms. It is highly recommended to read this webpage. The nice thing of Fieldtrip is that it reads in the raw data very efficiently, by defining the trigger-based trials and epoch-times beforehand, and reading in only these epoch windows from the raw data file; this way the detection algorithm works relatively fast. Another nice feature is that you can pre-specify some settings of the algorithm to have it be custom-tailored to the type

of artifact you are looking for. Muscle (EMG) artifacts are characterized by high-amplitude, high-frequency activity, cover a large number of electrodes, and are short-lived. These settings are operationalized in the `cfg.artfctdef.zvalue` settings, and the algorithm looks for those trials with "outlier" z-values within these settings.

But first, we need Fieldtrip to read in the raw data. This means we need to go back to the original `.bdf` files (assuming a Biosemi EEG system), and use the same trigger information that we used in EEGLAB to generate epochs. The code to do this looks like this:

```
cfg=[];
cfg.filename            = [readdir filz(filei).name];
cfg.headerfile          = cfg.filename;
cfg.trialdef.eventtype  = 'STATUS';
cfg.trialdef.eventvalue = 11;
cfg.trialdef.prestim    = 0; % latency in seconds
cfg.trialdef.poststim   = 1; % latency in seconds

cfg = ft_definetrial(cfg);
trl = cfg.trl;

filename        = cfg.filename;
cfg             = [];
cfg.trl         = trl;
cfg.datafile    = filename;
cfg.headerfile  = filename;
cfg.continuous  = 'yes';
```

Note that now we have not created buffer zones around stim onset, but instead we have specified that the epoch should go from 0 to 1 second. This is because Fieldtrip's artifact detection algorithm has buffer zone information specified separately. The Fieldtrip tutorial webpage explains how this works. The exact settings, again, depend on your experiment, and around what time windows you consider an artifact to be particularly harmful. For example, a burst of EMG around stimulus onset, or between stimulus onset and the response, should be rejected no matter what; but an EMG artifact in the inter-trial interval might not be necessary to remove, because you may not be analyzing this time window anyway. Again, play around with the settings and inspect the results carefully. Below is the code that does the detection automatically.

```
% channel selection, cutoff and padding
cfg.artfctdef.zvalue.channel = 1:64; % don't consider EOG!
cfg.artfctdef.zvalue.flexible = 'yes';
cfg.artfctdef.zvalue.cutoff      = -4;
cfg.artfctdef.zvalue.trlpadding  = 0.5;
cfg.artfctdef.zvalue.fltpadding  = 0.5;
cfg.artfctdef.zvalue.artpadding  = 0.2;

% algorithmic parameters
cfg.artfctdef.zvalue.bpfilter    = 'yes';
cfg.artfctdef.zvalue.bpfreq      = [110 140];
cfg.artfctdef.zvalue.bpfiltord   = 9;
cfg.artfctdef.zvalue.bpfilttype  = 'but';
cfg.artfctdef.zvalue.hilbert     = 'yes';
cfg.artfctdef.zvalue.boxcar      = 0.2;

% make the process interactive
cfg.artfctdef.zvalue.interactive = 'yes';

[cfg] = ft_artifact_zvalue_x(cfg);
```

A few things are important. First, EMG activity is here defined as 110-140 Hz activity. This is again a choice that the researcher makes, no law of nature. Second, there are some padding (buffer-zone) variables: `trlpadding` adds data around the epochs within which you still consider artifacts to be harmful (e.g. in the inter-trial interval, or in the pre-stimulus baseline) and thus determines the eventual epoch size; `artpadding` is the window around the peak of the detected artifact; `filtpadding` adds some additional data on top of the `trlpadding` to control for edge artifacts of the band-pass filtering; this padding is removed after the filtering and is not taken into account in the artifact detection.

Finally, the starting z-value cut-off is here set at 4, and the option 'fexible' set to 'yes'. We have manually adjusted the regular `ft_artifact_zvalue` function, to make the cut-off somewhat less arbitrary. The usual Fieldtrip approach is to specify above at which z-value threshold you want to reject a trial. Z-values represent deviance from the mean, are computed for each channel over time, and are then added together per trial, such that trials with strong artifacts on several channels receive higher z-scores than trials with a small artifact on only one channel. Still, setting the cut-off is arbitrary, as also described on the Fieldtrip website:

"*Depending on the variance of the artifacts versus the variance of your brain signal a higher or lower threshold has to be set. The lower this threshold the more conservative the artifact detection will behave, the higher the more liberal. Since these characteristics of the data might vary per experiment and even from one recording to another, care has to be taken to investigate the threshold that works for you*".

The pre-specified cut-off will completely determine whether a trial is marked for rejection or not. This is also where the automatic process becomes in fact quite similar to the manual rejection procedure. That is, very clear artifacts will have very high z-values, so whether you set the cut-off at 10 or 20 (this is roughly the "window" of sensible cut-offs), these will be marked nonetheless. However, these were probably trials that you would have easily selected manually/visually as well (although this obviously would have taken more time). On the

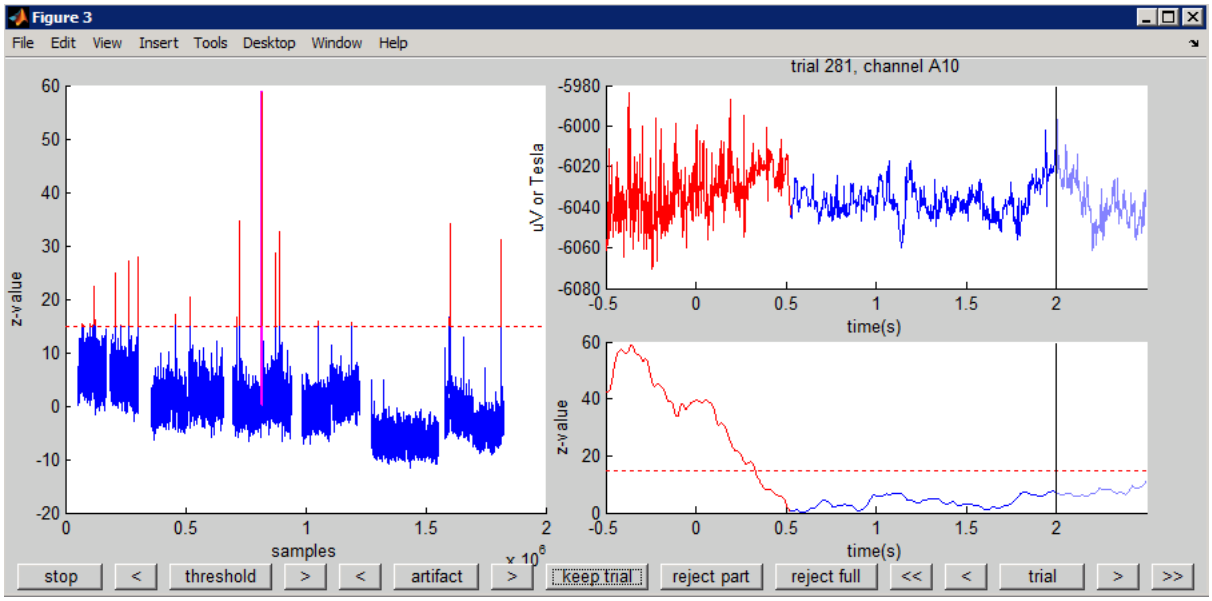

Figure 1: Output of Fieldtrip's EMG artifact detection algorithm.

other extreme, very clean trials have very low z-values and will probably never be marked for rejection. The choice of cut-off could be considered a numerical expression of the subjective doubt you sometimes have during the manual rejection: should I reject this trial or not? However, during manual rejection it is hard to verbalize how you came to your decision; with the z-value cutoff, you have a number, that you can for example report in a research paper, or that you can save so that somebody else can replicate your rejection procedure.

The default Fieldtrip algorithm has been adapted slightly, to let the cut-off change per subject. With the 'flexible' option set to 'yes', it computes the average "bandwidth" of z-values, and adds the value of the provided cut-off to the upper-limit of this bandwidth. For a regular Biosemi recording of good quality, this results in a cut-off of around 12-15, but can be lower or higher depending on the quality of the recording. You should check the eventual cut-off values, and report the range in your paper where you describe this method. The nice thing of this approach is that it's automatic while still leaving in room for variability in decision thresholds per subject; much like you would do with manual artifact rejection, except that now you have a quantification of this variability, and people can exactly replicate your rejection method. Of course, whether you add 3, 4 or 5 (or 80 for that matter) is still a subjective decision, but we would have to wait for artificial intelligence to provide us with an EEG-artifact-detection-robot to circumvent this. To have the same cut-off for every subject, set the 'flexible' option to 'no' and apply a default cut-off value of 12-15, depending on the quality of the data.

If you have set the `cfg.artfctdef.zvalue.interactive` to `'yes'` you get a pop-up window that shows which trials were detected, like the one shown in Figure 1. As you can see, some trials showed very high z-values. In this interactive window, you can "scroll" through the trials, or jump to each next detected artifact, with the arrow-buttons in the bottom of the figure. You can also see that the cut-off value of 15 is reasonable, as most trials fall below. You can also see that there are quite some trials that are just below or above the threshold. That's where it becomes tricky. You can adjust the threshold still at this step (again by the arrow buttons), which will simply mean that with a lower threshold more

trial will be added to the rejection-list, or with a higher threshold trials that were marked for rejection are now removed. This is also where one of the downsides of Fieldtrip kicks in again: poor visualization of raw EEG. Although this window is quite handy, the top right plot only shows one trial, and one channel that was affected the most. But you don't get a complete picture of what was going on, in terms of the surrounding trials and all other channels. What I advice to do is keep the z-value as is, except if you can see from the left subplot that it is really off; then, simply click "stop", which Fieldtrip will interpret as you having accepted these settings. The output in the command window will say something like "`detected 34 artifacts`", and Fieldtrip has generated a list of time points along the continuous EEG where the algorithm has found artifacts.

The next step is to get these detection values back into EEGLAB, and as such make use of the superior visualization of EEGLAB. This is a bit tricky, because Fieldtrip doesn't give you trial numbers, but time stamps in real-time. The below code first removes the detected trials from the Fieldtrip trial list, and then compares the old and the new trial list, yielding trial numbers that were rejected. Note that we haven't actually removed any EEG data, we only removed lines from a matrix that contained the trial information. The code then puts the to-be-removed trial indices back into an EEG structure that EEGLAB can read, and then calls the EEGLAB raw data inspection window:

```
[cfg] = ft_rejectartifact(cfg);

rejected_trials_ft = ismember(cfg.trlold(:,1),cfg.trl(:,1))==0;

EEG.reject.rejmanual = rejected_trials_ft;
EEG.reject.rejmanualE = zeros(EEG.nbchan,length(trl));

pop_eegplot(EEG,1,1,0);

disp(find(rejected_trials_ft)');
```

The last line prints the trial indices that were marked in the command window. The `pop_eegplot` function generates a window as shown in Figure 2. The arrow buttons allow you to scroll through the trials, but now you have a complete glance of the data. Now, one of the advantages of having wide epoch windows (as mentioned earlier) becomes apparent: the epochs overlap in time around the ITI, or even include some next-trial-stimulus or previous-trial-response trigger info. This allows you to better judge where an artifact happened, and whether it is worth rejecting. Scrolling through the trials, you will notice that some are marked yellow; these are the trials that Fieldtrip marked for rejection based on the automatic algorithm. Do you agree with the automatic detection? You could go through all trials for a few subjects the first time you do this. You need to develop a feeling of what the algorithm does, and whether your timing variables of artifact-sensitive windows were reasonable. The EEGLAB data viewer allows you to do this better than the Fieldtrip window. You may occasionally reject a trial that Fieldtrip didn't detect, or vice versa, de-mark one that Fieldtrip detected but you want to keep in. But, **keep this to a very minimum**, preferably in less than 1% of trials! If you notice you are doing this for $> 1\%$ of the trials and for every subject, you need to think whether the settings used in the Fieldtrip algorithm were optimal. You can also decide beforehand not to change the Fieldtrip-marked trials manually at all, and accept some false

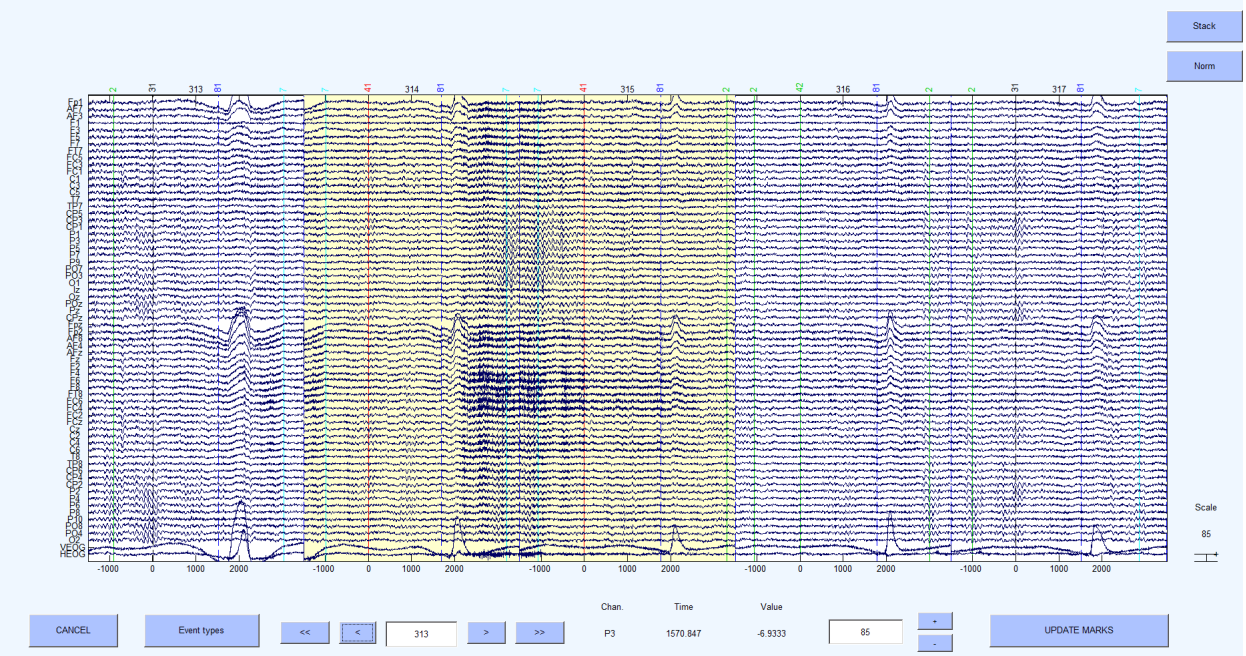

Figure 2: Output of EEGLAB's `pop_eegplot`*, with bad epochs marked by Fieldtrip's automatic algorithm.*

positives and negatives; this is arguably less subjective. It is however highly recommended to at least visually check the result in some or all subjects. This may give you a reason to change the cut-off from, e.g., 3 to 4 or 4 to 5; but it would be best to then apply this threshold to all subjects (and let the adapted Fieldtrip script set the eventual threshold to vary over subjects).

If you are satisfied with a cut-off, you can do three things:

1. Run everything without looking at any data; this is fine, but not if you haven't looked at *any* raw EEG data at this stage!

2. With the EEGLAB viewer, only do a random check of some clean trials in the EEGLAB window, and a random check of some rejected trials; you can use the command-line printed trial indices for this.

3. Go through all trials with the viewer, and only use the Fieldtrip detection as a starting-point to make this manual rejection procedure go faster.

With options 2) and 3), if you agree with the rejection, you hit "Update marks" in the right bottom of the EEGLAB window, and run the rest of the code. With option 1) you can skip this. All three options will give you in the end a vector of rejected trials (with zeros corresponding to clean trials and ones to trials that need to be rejected), which will be save, along with the cfg with the Fieldtrip settings. This way you can always look back to what you did, report the settings, and have somebody else replicate your semi-subjective artifact procedure. Actual rejection happens in a later step.

In sum, we believe that this combined approach of flexible, subject-dependent cut-offs (based on one number applied to all subjects), automatic rejection, and initial visual inspection of the settings, combines the best of both worlds of fully automatic on the one hand, and fully manual rejection on the other hand.

4 Removing eye-blinks with Independent Component Analysis

Muscle artifacts are momentary, and affect all or most channels. Eye-blinks are also unwanted signals, as they reflect a sudden change in the dipole generated by the eyeball, which, due to volume conduction, also affects neighboring (frontal) electrodes. Labs differ widely in how they deal with eye-blinks. Some try to instruct subjects to only blink during the inter-trial interval, and reject trials in which subjects failed to do this (i.e. they blinked during, or between the stimulus and response). Two disadvantages of such instructions are 1) eye-blink artifacts can sometimes become bigger, presumably because the blinks are more cognitively controlled in time and cost more effort; 2) this instruction induces an unrelated volitional, cognitive process on top of the experimental task. If you show subjects before actual data recording, but with their EEG already attached, how eye-blinks are expressed as artifacts, and if you tell a little bit about this, you make them well aware of eye blinks and this already reduces the impact of the artifact substantially *during* recording.

After recording, you can make use of one important characteristic of eye blinks: they look the same, happen consistently over time, and are quite different from brain signals. Independent component analysis (ICA) decomposes channel data into components that are independent from one another over time; these components consist of a weighted sum of all electrodes, such that some electrodes contribute more to this component than others. From this you can already guess that ICA is very suitable for capturing the eye-blinks from the channels that are affected by the blinks, by putting them into one component. If you remove this component, and recreate the channel data by "unweighting" the components, the eye-blinks artifacts are removed without having to remove any trial.

4.1 pop_runica

EEGLAB is famous for its ICA procedure (in fact, Fieldtrip uses EEGLAB's ICA function as a plug-in). The downside is that it takes quite some time (a few hours for a one hour recording). The `runICA.m` script that you can download from GitHub first removes the trials that were marked in the previous step, because artifacts can be harmful for the ICA estimation. In addition, this script contains a few lines of code to import channel indices of channels that were flat, broken, or contain some other clear source of noise such that you will interpolate these channels later (more about interpolation below). This is important, because a channel that is very different from every other channel in terms of its activity over time, will be fully captured by an independent component. If you know this already beforehand, it is best to not take into account this channel during ICA.

This particular code needs a `.txt` file with at every line the subject-file prefix (e.g. "pp01") followed by a tab, followed by channel names that are separated by spaces (e.g. "pp01 AFz T8"). Of course, change this code according to your own settings or likings, but just make sure that this code will eventually produce a variable `chanind` that contains channel indices over which you want to perform ICA. If a subject has clean channels, this should be all active scalp electrodes, *without* the EOG or other external electrodes. These can also badly influence ICA.

The ICA script is inside a loop over files/subjects, so you can wait until you have all independent components of all subjects. The EEGLAB code to remove trials and to run ICA

is:

```
% remove trials marked by variable rejected_trials
EEG = pop_select(EEG,'notrial',find(rejected_trials));

% import channel indices to be interpolated later
...

% run ICA
EEG = pop_runica(EEG,'icatype','runica','extended', 1,'chanind',chanind);
```

For high-density nets (128 or 256 electrodes), regular ICA takes even more time, because the number of components is equal to the number of channels. Given that only the first ∼20-30 components are the most relevant (because the order of components depends on the amount of explained variance), EEGLAB offers the possibility to first decompose the data into a number of *principal* components (with PCA), and then apply ICA on those components, yielding the same number of independent components. The code for this (with the data reduced to 30 principal components) is:

```
% run PCA-based ICA
EEG = pop_runica(EEG,'icatype','runica',1,...
    'options',{'pca' 30},'chanind',chanind)
```

In the example scripts, only the ICA loadings, or weights, are saved, not the EEG. This is because actual trial and component rejection is done in a later step; only saving the IC loadings will also reduce file size.

4.2 Component inspection

To inspect the components, import the `.mat` file into the EEGLAB GUI, and click on Tools > Reject data using ICA > Reject components by map (see Fig. 3). You can just plot the first 35 components, as the other components hardly contribute to the variance in the data. This will generate an interactive plot with numbered topomaps (see Fig. 4) that you can click on. From the example shown in Figure 4, you can already notice two things: the eye-blinks are probably captured by the first component (this is often, though not always, the case), and there was something wrong with one channel (which one?). Clicking on the first component opens up a pop-up window with information about this component: a more detailed scalp map, single-trial and trial-averaged ERPs, and the power spectrum. How can you see that IC1 represents blinks?

To get more insight into the components and what they reflect, you should open two more windows: the actual data (in the same viewer we used before; accessible in the GUI via Plot > Channel data (scroll)), and the component activations (Plot > Component activations (scroll)). This latter plot nicely illustrates how the explained variance decreases with increasing IC number. You should increase the scaling (in the bottom right; about a 10-fold increase), and plot only the first 30 components (Settings > Number of channels to display). Scroll

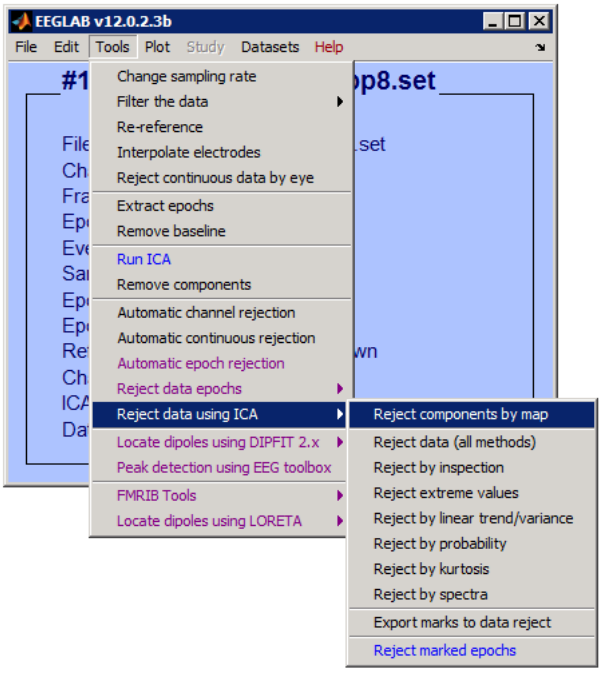

Figure 3: Reject ICs by map.

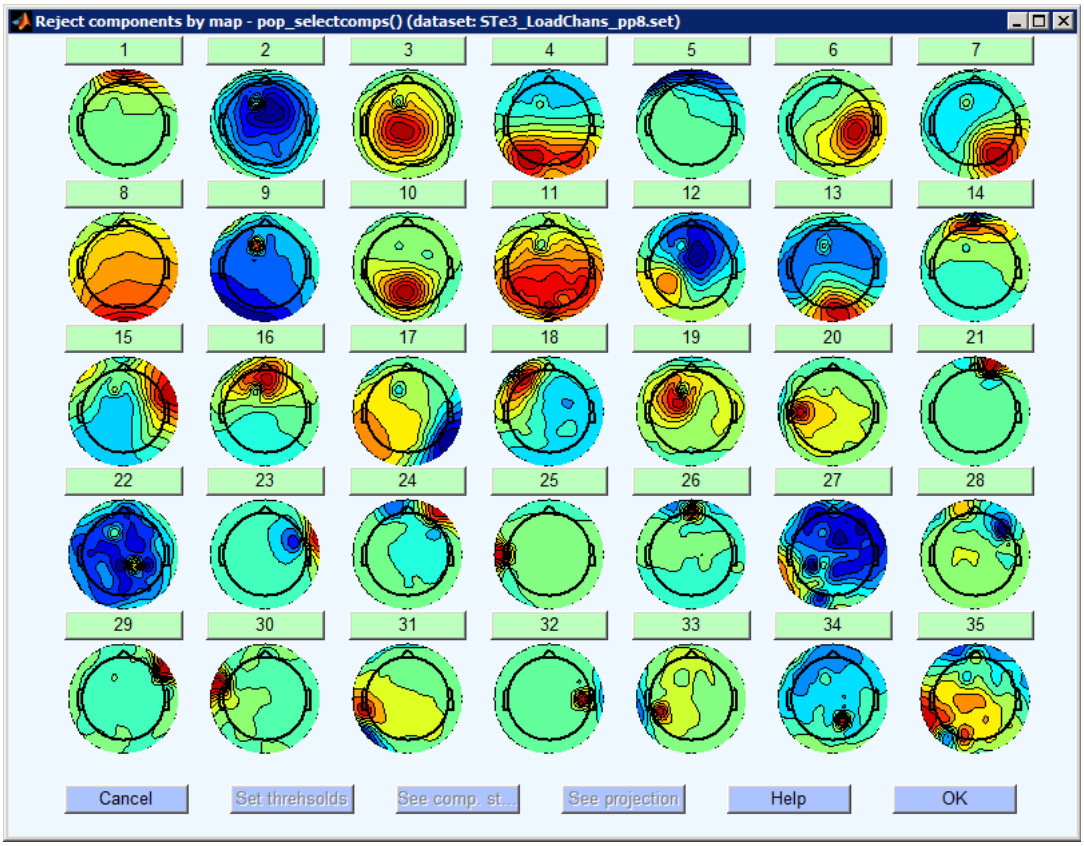

Figure 4: The first 35 components, scalp map view.

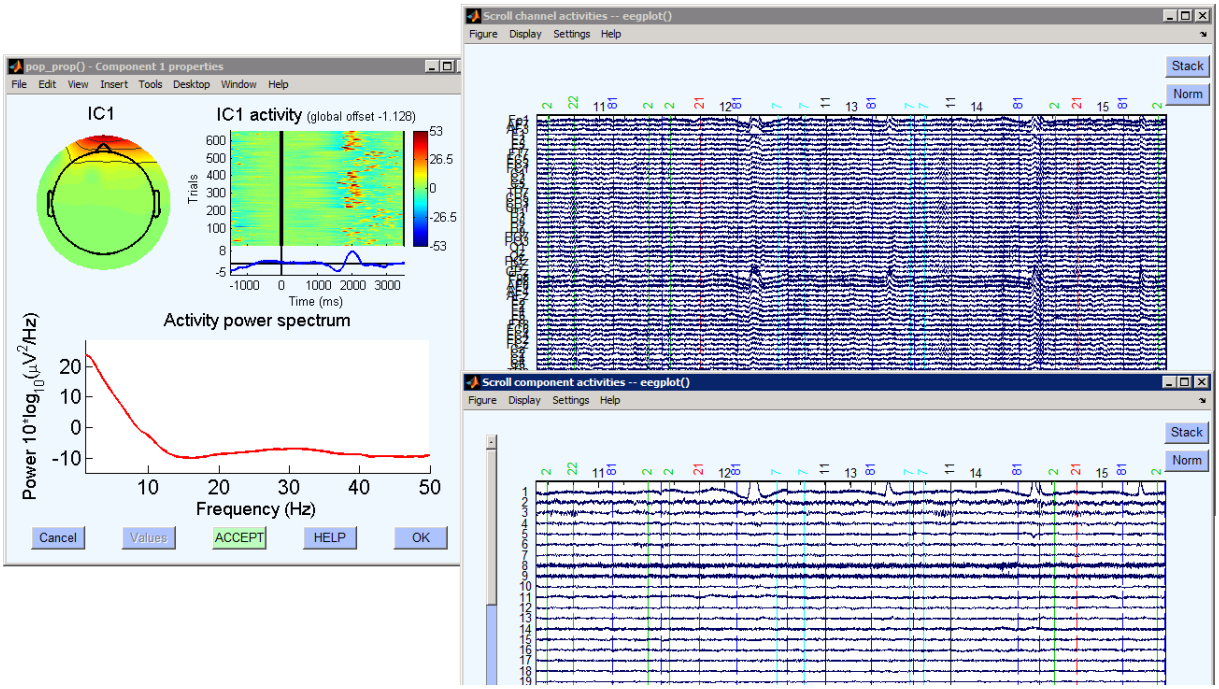

Figure 5: Converging evidence that component 1 reflects eye-blinks.

through the trials in both the component and real data; which component represents eye-blinks?

You could already remove this component at this moment. However, similar to trial rejection and channel interpolation, it is recommended to note down the components you want to remove, and do this in a subsequent, separate script, just before the next step in your analysis pipeline; this allows for more flexibility without having to redo everything from scratch. For example, use a simple `.txt` file in which you note the subject name followed by the components you want to remove; this text file is later imported and the actual components are removed (much like the channel interpolation, see above and below).

4.3 Eye-movements

Eye-movements are tricky. In comparison to blinks, they are arguably more likely to reflect a cognitive/perceptual process. However, similar to blinks, the eyeball movement generates dipole activity that contaminates nearby EEG electrodes, which is something you don't want. Fortunately, if eye-movements have contaminated the main EEG substantially, ICA will capture this, and you will probably be able to find the corresponding component quite easily. One could argue that you can remove this component, using the same argument as for removing eye-blink components. This way, if anything, you have cleaned your data from the oculomotor artifact. Whether you want to remove trials because a subject broke fixation, is a different issue and is explained later in this document, and depends on your experiment and your question.

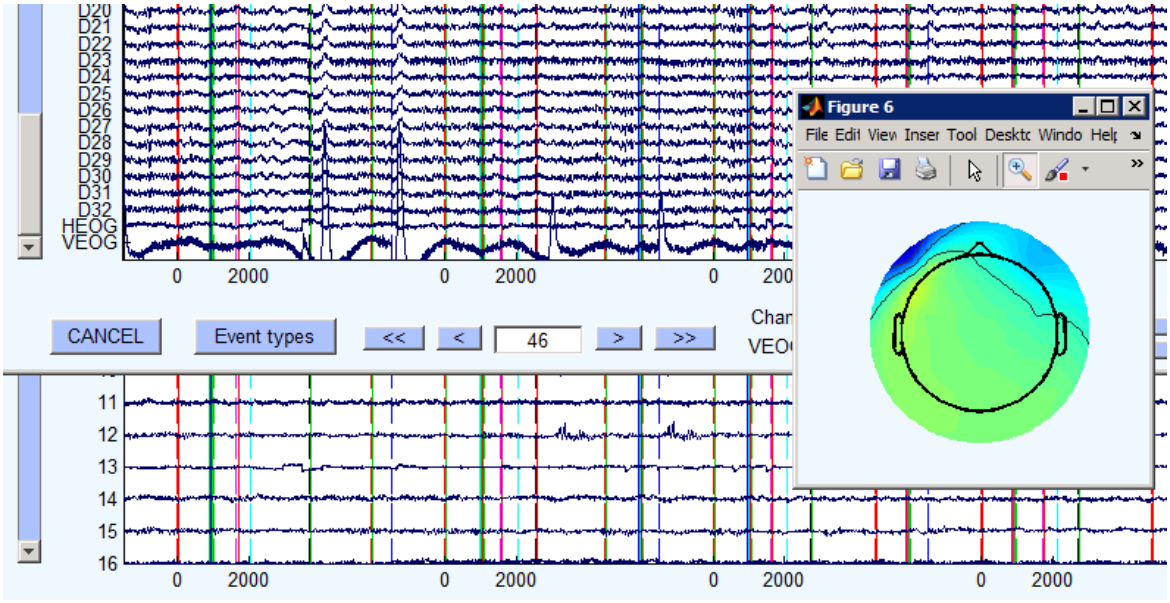

Figure 6: An example of an IC capturing eye-movements. Signal #13 in the lower line plot is the time course of the IC shown in the topomap; the box-car-like jumps are eye-movements, which can also be seen in the horizontal EOG channel (HEOG in upper line plot); notice that the other channels are also influenced by the eye-movement; removing the component will remove this oculomotor artifact.

4.4 Components that pick up non-EOG noise

From eye-balling the first 19 components in Figure 4, you can see that one electrode doesn't seem to contribute to the component activations, while the surrounding electrodes are. This is suspicious, and indeed, upon inspecting the raw data, this channel appears to be relatively flat (see Fig. 7). IC9 seems to load heavily on exactly this electrode, so in these circumstances you could try removing this component and see how it changes the data of this channel. However, remember that *every* component is *always* a weighted mixture of *every* channel. The color scaling of a topomap as shown in Figure 7 is in this sense misleading, because it looks like the other channels are unaffected. But removing this component will do something to *every* channel (be it to a very minor degree). When you have a flat channel (or a channel that shows a lot of noise that is not due to muscle artifacts, and which is only present in this channel), you should **interpolate** this channel, rather than removing the component that captures the "independent" noise (or flat signal) in this channel. As with the to-be-removed components, make a simple text file with the same list of subjects and the channels that need to be interpolated, and do the interpolation later. Remember from above that it is in fact better to perform ICA on data without such a to-be-interpolated channel. Sometimes you notice through ICA that a channel was flat or otherwise heavily contaminated, and you missed this by visual inspection of the raw data. It is recommended to then add this channel in your `.txt` file with the list of to-be-interpolated channels, and redo ICA for this subject.

It is not always clear cut, though. For example, you might have a channel that shows a weird artifact on only some parts of the experiment. You have to make a balanced decision. Consider the following options: 1) Was it an important electrode, considering the hypotheses that you want to test (e.g., T8)? If not, interpolation is fine even if the channel doesn't show noise all the time. 2) If there were very few moments of noise, then you might want to

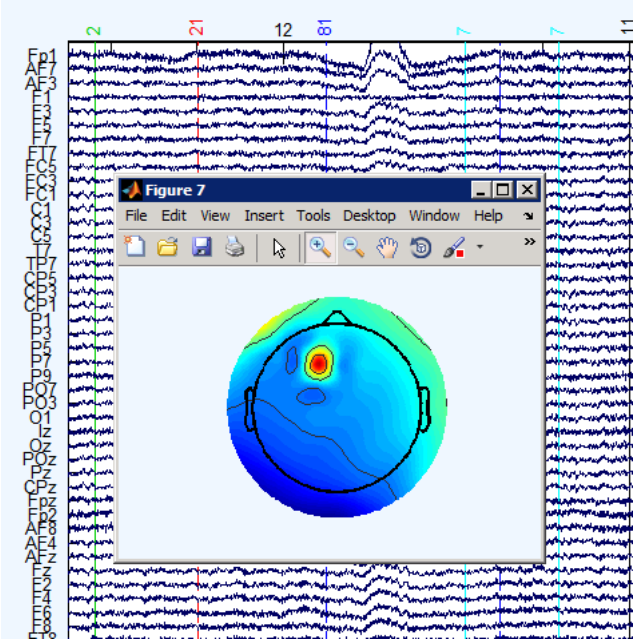

Figure 7: There was something wrong with channel F1.

consider removing the whole trial (I would only do this if the channel is very important for your research question). The downside is that you loose trials with good data in every other channel. 3) If there was noise in almost the entire experiment, you should probably interpolate no matter what. 4) If the situation is in between, and if the one-channel noise is well captured by an independent component, you could still consider removing the component. As ICA also takes into account the time dimension, this may thus be better than interpolation, because the periods of time without noise will be less likely to affect your data (the bad channel plus other channels). The downside is that in those time periods with noise in this one channel, the other channels will also be slightly affected by the component removal. In any case, should you decide to interpolate: mark the channel for interpolation, and redo ICA without this channel! This will improve the quality of the other components.

With high-density EEG nets, this final scenario can happen more often (see Fig. 8 for an example). As an example, in a study in which we used a 256 EGI net, we came up with the solution of doing single-trial interpolation: we interpolated the bad channel(s) only on the trials in which it/they showed an artifact. We did this because every solution described above failed (either too many trials rejected with lots of good channels, or too many electrodes interpolated that were clean most of the time).

Furthermore, you will notice that some components load on a few channels and in their time course show a thick, "caterpillar"-like signal of high-frequency noise. It is tempting to remove these components, because the channel data after removal looks cleaner. However, it is recommended to **not** remove these components. You don't know exactly what they reflect, and especially if you inspect the trial-average ERP in the component pop-up window, you are likely to observe event-related activity; you are going to remove this activity from a few channels, which is undesirable. As a general rule: try to remove as few components as possible. Some might even advice to **only** remove blink-components.

Finally, a flat channel is not always well-captured by one single IC. You might see in

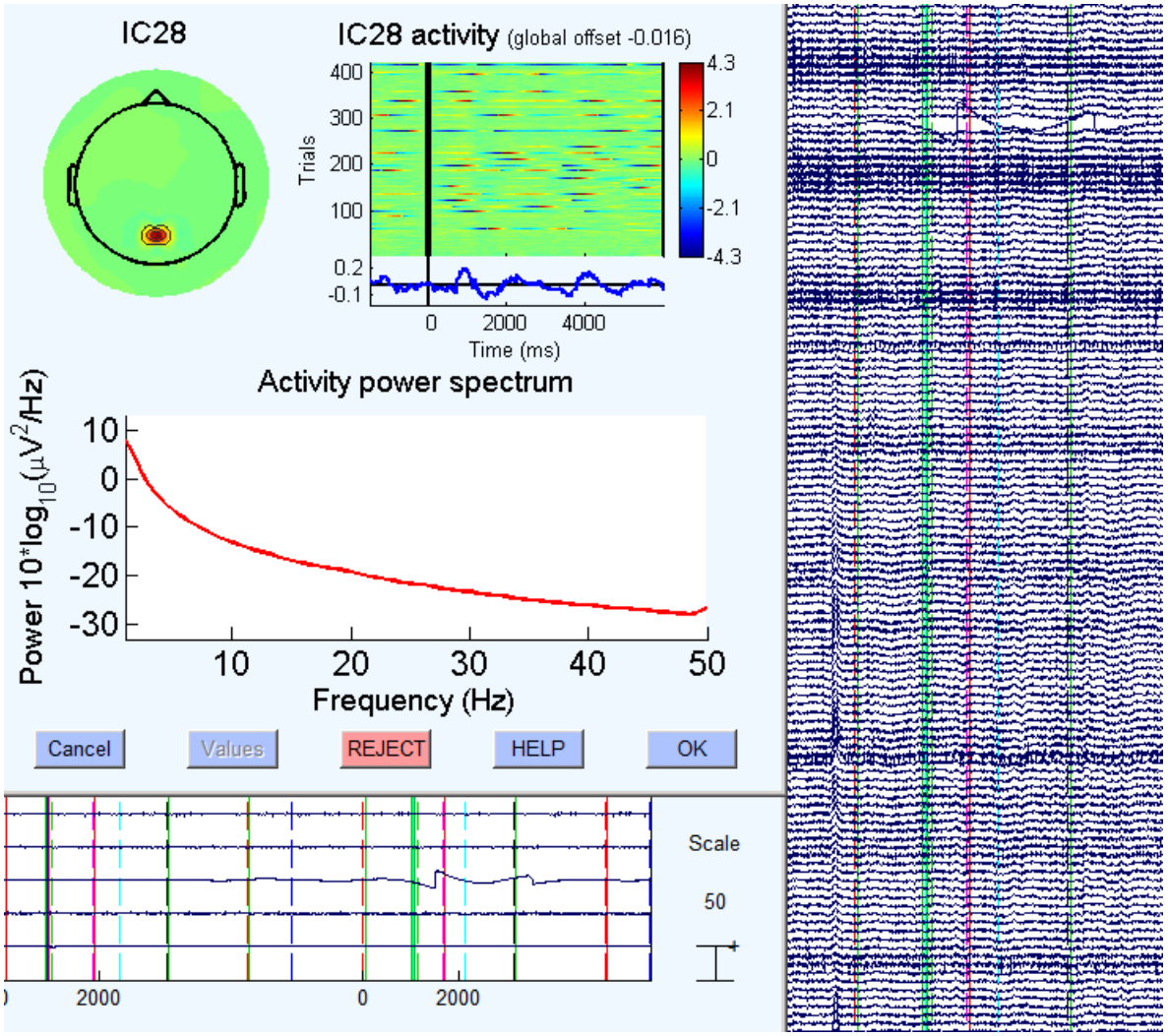

Figure 8: IC28 (after PCA-based ICA with 30 principal components of 128 channel data) showed non-EOG artifacts on roughly 25% of trials and loaded specifically on one channel that in the raw EEG indeed showed the same artifact. Here, you could consider removing the component, because this might be an important electrode (making interpolation less favorable) and the artifact happened in too many trials (making trial rejection not a good idea).

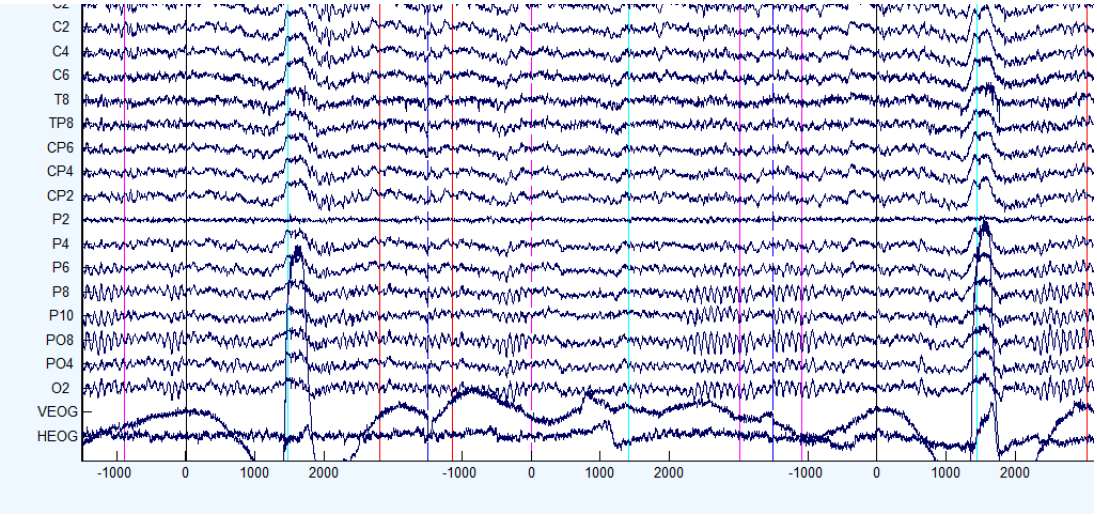

Figure 9: Another flat channel: P2

various component maps that one particular electrode loads less on this component than the surround electrodes; this should make you suspect that this channel might have been flat, and illustrates why it is better to not include this channel in the ICA computation. Careful inspection of the raw data can confirm this; here, eye-blinks can actually become handy: if a bunch of electrodes show a blink, while one in between doesn't, than this electrodes was probably not well-attached (see Fig. 9.) A way to get more confirmation is to make a rough ERP: just trial-average the data from this electrode and two surrounding ones. Figure 10 shows the example of the flat channel of Figure 9.

4.5 Inspecting the results

Usually, you will select only one or two eye-blink components to remove. By hitting the 'Accept' button in the pop-up window of the component, this turns into 'Reject', and is marked red. You can then check what component removal did to your data, by selecting in the EEGLAB GUI: Tools > Remove components. Delete the second box (because this is the reverse of removal: only *keep* the selected components; some researchers argue that particular components more accurately reflect the cognitive/perceptual process of interest, and that you should continue your analyses on the components instead of the data) and hit OK. In the next pop-up window, EEGLAB will ask you to confirm; **do not** hit Accept. Instead, you can inspect the result by clicking 'Plot single trials', which will produce the familiar data viewer, but now on top of the original signal, the data with the removed component is plotted in red. As you can see, blinks are strongly attenuated (see Fig. 11). If you are satisfied, click **Cancel** in the confirmation window.

4.6 Automatic artifact-IC detection

Yes, even this step can be done automatically! After some practice with looking at ICs, you may notice that especially the blink (and eye-movement components if clearly present) look all very similar and end up high in the IC-ranking (usually, the blink component is nr. 1!). There is an EEGLAB plugin named ADJUST, that looks at the typical features of artifact

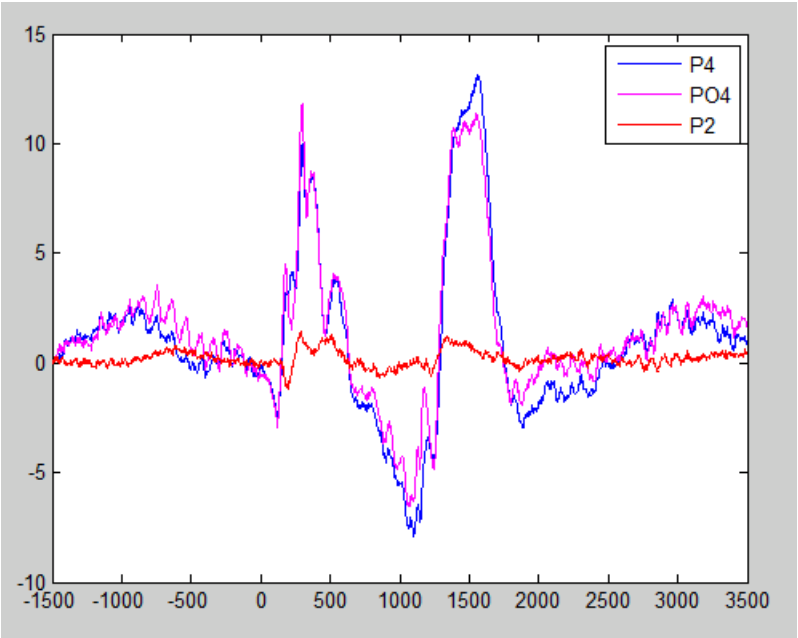

Figure 10: Three ERPs. You can see that P2 still contains a little bit of event-related activity, but it is much weaker than two surrounding electrodes.

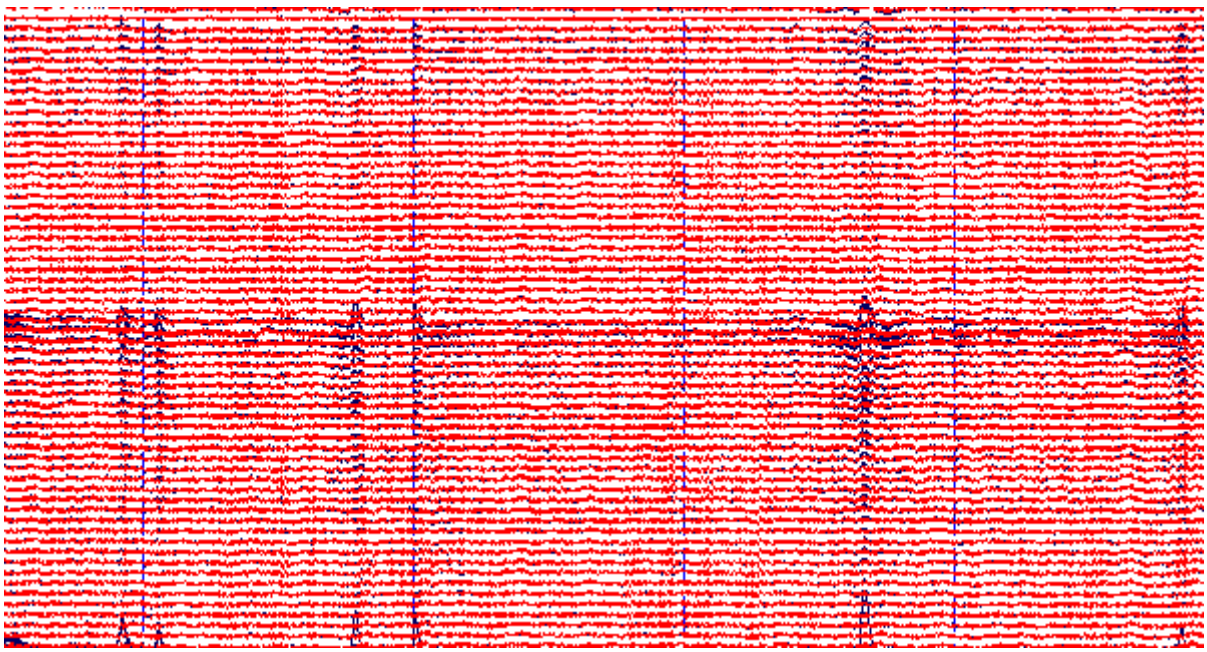

Figure 11: Before and after removal of one IC that captured eye-blinks.

components, and imports their numbers in a variable and in an output log file. The script to do this is very simple:

```
% perform ADJUST artifact-IC detection
[art, horiz, vert, blink, disc]=ADJUST(EEG,outputfilename,2);
```

The output variables will be simple vectors containing the numbers of the ICs that the ADJUST algorithm has detected to represent particular artifacts, respectively: components with all artifacts, horizontal eye movements, vertical eye movements, blinks, and other non-ocular artifacts. It is recommended to only remove the blink components, and inspect the other detected components carefully with the ICA visualization in EEGLAB. In addition, this function has again been adapted slightly, by adding a multiplier-factor to the input (here, '2'). The automatically generated threshold is usually too low, such that components with only very minor blink residuals will also be part of the blink component vector. By adding '2' to the input, you double the threshold, making the IC-removal criterion more stringent. In practice you may change this multiplier (to '1' if you trust ADJUST's threshold computation); again, make sure to use the same factor for every subject within a study.

5 Final preprocessing steps

5.1 Actually remove noise components and interpolate bad channels

In the next script (called `postICA.m`; again this script is available at this Github page), there are still quite a few additional preprocessing steps, all done in a soft-coded manner, mostly using the 'ingredients' of the previous steps (removing detected trials, channels and ICs that contain some kind of artifact). First, two `.txt` files are imported in which the components to be removed, and the channels to be interpolated, were written down manually. The script then finds the component and channel indices, and removes/interpolates them, respectively. The EEGLAB code for component removal and channel interpolation is:

```
% remove components
EEG = pop_subcomp(EEG,ICs);

% interpolate channels
EEG = eeg_interp(EEG,badchannels);
```

The order of these steps is important: channel interpolation basically means that the bad channel is removed from the data, and is "recreated" by a weighted average of the surrounding channels; you then want these channels to be as clean as possible, because you don't have independent component loadings of this bad channel (because you did not include it in ICA). This also requires an additional trick: IC loadings consist of a nchan-by-nchan matrix; to remove components, your EEG data need to contain a similar number of channels. In case you applied ICA on data without to-be-interpolated channels, the ICA matrix and the EEG data matrix will have a different number of channels. Thus, you need to 1) temporarily remove

the EOG channels and the to-be-interpolated bad channels, 2) remove blink components, 3) put the bad channels back in, and 4) interpolate these bad channels.

5.2 Peri-trial information

Usually, you average all trials of a particular condition, and at the group level compare conditions. However, depending on your experiment and question, you might want to have the option to look in a later stage at *sequence effects*. Examples of sequence effects are: correct trials after error trials are usually slower; brain dynamics related to cognitive control in situations of response conflict are attenuated when the previous trial was also a response conflict trial; the duration between a warning signal and an imperative stimulus on the previous trial influences motor preparation on the subsequent trial.

Even if you don't know a priori what kind of sequence effect may be worth studying, as soon as you rejected trials and continued your analyses, it gets harder to figure out what happened before or after a trial. Because of this, it can be handy to put peri-trial information in each epoch right *before* you remove artifact-contaminated trials. EEGLAB attaches to an EEG structure the trigger and latency information of each epoch; you can find this inside `EEG.epoch(n)`, where `n` stands for the n_{th} epoch. With a loop (see the `runICA.m` script) you can add an additional field to the EEG structure containing the epoch information from the previous and next trial. This way, you can always reconstruct, e.g., which button a subject pressed on the trial n-1, what the RT was of this previous trial, and whether this was correct or not. Note that EEGLAB's trial removal function only removes the data in `EEG.data`, not the epochs in `EEG.epoch`; you have to do this separately. Now, when you removed for example three trials in a row because these contained artifacts, the clean trial before this sequence still contains the information of what happened in the subsequent removed trial, in the EEG structure.

5.3 Horizontal eye movements

Apart from EMG (muscle) noise, trials can be 'contaminated' with (horizontal or vertical) eye movements. The apology quotes reflect the fact that here, it is not so much that the EEG contains noise (in case of very strong oculomotor noise due to eye movements, ICA could have taken care of this), but rather that the subject broke fixation which may be a bad thing given the task instruction (e.g. "focus at fixation while covertly attend to the left of the screen"). For some tasks this may not be a bad thing, or you may have even collected eye-tracking data, so you don't need the EOG for this. In short, you *can* detect large deflections due to eye movements with EOG, provided the signal is clean, which requires a careful setup. However, with poor EOG quality, and/or with subtle eye movements, this is a very tricky step. In general, if eye movements are very important, and you don't want any trial with even the smallest deviation from fixation, consider including eye-tracking measurements alongside EEG. This is way better than eye movement detection based on EOG.

Nonetheless, below is a snippet of code to automatically remove clear eye movements. An adapted version of the algorithm from the ERPLAB toolbox was used, and again you need to specify quite some settings which may be highly variable depending on data quality, experiment, etc. Check the results carefully (using the EEGLAB viewer) and save the trial

vector, much like you did with the EMG-artifact detection step. This step shouldn't remove too many trials (i.e. less than 10%), at least not for the majority of subjects.

```
rejected_trials_heog = pop_artstepX(EEG,'channel',66,'flag',1,...
    'threshold',25,'Twindow', [ -50 1050], ...
    'Windowsize',  100, 'Windowstep',  50 );
save([ prefilz(subno).name(1:end-13) 'rejected_trials_heog.mat'],...
    'rejected_trials_heog');
EEG = pop_select(EEG,'notrial',find(rejected_trials_heog));
```

Here, channel 66 contained the bipolar HEOG (left vs. right horizontal EOG). The 'threshold' parameter concerns the amount of microvolts above which you consider a deflection to be large enough to represent an eye movement. The 'window' parameters are also important: here, over the time window from -50 to 1050 ms around stimulus-onset, the average activity in short 100 ms windows was computed, in 50 ms steps (i.e. a sliding window approach); at each step, the average activity of the current window is compared to the previous one; when the difference exceeds the threshold, this is taken as a 'sudden' jump in activity, a charachteristic of an eye movement. You can easily guess that different window sizes can quite dramatically change the number of detected 'eye-movement trials'.

5.4 Divide into conditions and apply CSD

The EEG structure contains all data. To divide this into conditions, you need to make a script that searches for particular trigger information (e.g., what is the trigger value in a particular epoch with latency of 0 ms, i.e. the stimulus to which the epoch is locked?) and divides the data according to this information (e.g. put the epochs with trigger=11 sent at 0 ms in a separate structure, put the epochs with trigger=12 at 0 ms in a separate structure). Here, you could already incorporate peri-trial info, if you have specific ideas on what type of trial sequence you want to study. You could also make the trial selection be dependent on accuracy, RT, etc. This part requires some creativity. In the example script I left in settings from a specific study; note that these are for illustration purposes, and should be replaced with new code.

You can put the different conditions into a `ALLEEG` structure with different fields (e.g. condition 2 is in `ALLEEG(2)`), where you can give each ALLEEG a setname (`ALLEEG.setname`). This is handy because EEGLAB interprets ALLEEG structures as different datasets; this allows you to import them into the GUI, and do basic point-and-click (ERP) analyses.

Next, the very last preprocessing step before you can do time-frequency analyses, is to apply a current source density (CSD) transform, which is equivalent to the surface Laplacian. The problem with EEG is that any brain signal, at any location and time, projected its activity onto all electrodes (although nearby electrodes received more "weight" than distant electrodes) such that nearby electrodes measure a lot of overlapping activity (this is called *volume conduction*). For basic ERP analyses, this is not a big problem, because usually you don't make big claims about topographical specificity. However, especially if you want to do connectivity analyses in time-frequency space, volume conduction introduces a confound. Suppose two distant electrodes show theta-band (4-8 Hz) phase consistency over time and trials. This inter-site phase clustering (ISPC) effect is interpreted as interregional phase synchronization, which is

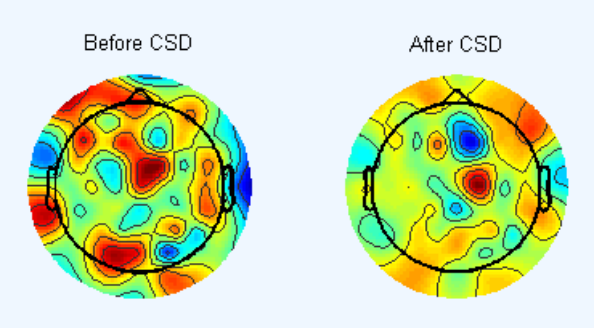

Figure 12: Before and after the surface Laplacian transform.

thought to reflect transient electrophysiological coupling of distant brain regions, so they can effectively communicate. However, the same finding could be a result of **one** brain region that projects activity to these two electrodes through volume conduction, resulting in "spurious" phase synchrony that is now a measurement artifact instead of true connectivity. There are several ways to circumvent this problem in the connectivity analysis stage. However, already during preprocessing you can heavily attenuate volume conduction.

The CSD or Laplacian is a spatial high-pass filter: topographical fluctuations of low spatial frequency are filtered out, thereby highlighting topographical specificity of high spatial frequency (i.e. fluctuations in activity that can show rapid transitions from electrode to electrode). An example is shown in Figure 12, of one subject, from 100 to 500 ms post-stimulus, averaged over 600 trials (different conditions). Although this is only one subject, you can see that the non-CSD activity is more scattered, while the CSD activity shows one clear dipole. Various methodological and empirical papers have demonstrated the strength and usefulness of the Laplacian. I believe it is less common in ERP research, but it will probably not harm ERP analyses, while leaving in the possibility to do phase-based connectivity analyses. At this stage, you could also fork your analysis pipelines, into non-CSD ERP analyses, and Laplacian-transformed time-frequency analyses. The EEGLAB function to apply the CSD is very simple:

```
load GHmatrices.mat
for ci=1:length(ALLEEG)
    ALLEEG(ci).data(1:64,:,:) = CSD(ALLEEG(ci).data(1:64,:,:),G,H);
end
```

A few things about this script: 1) it first imports a `.mat` file that contains two 64 channel-by-channel matrices; these are used by the CSD function for the spherical spline interpolation of surface potentials (G) or current source densities (H). These are fine tuned for BioSemi 64 EEG. At this website you can read more about the CSD and how you can make GH matrices for different EEG setups. It may be that not every EEGLAB version has the CSD function and the GH matrices included. 2) If your EEG data still contains external electrodes (EOG, EMG, ...) you need to specify specifically that you apply the CSD only on `ALLEEG(condition).data(1:64,:,:)`.

6 Overview of preprocessing steps

This document contained quite a lot of steps, example scripts and screenshots. Below we list each step briefly again, per script:

preICA.m

- Import raw data
- Re-reference to mastoids/earlobes/average reference
- High-pass filter (usually at 0.5 Hz)
- Epoch
- Baseline-correct (whole-trial baseline)
- Save the above
- Run Fieldtrip's automatic artifact detection algorithm
- Optionally: inspect detected artifacts with the EEGLAB viewer
- Save the `rejected_trials` vector and the Fieldtrip settings, do the rejection later!

runICA.m

- Import the EEG file created with the `preICA.m` script
- Import the `rejected_trials` vector and remove the trials (actual rejection still not saved, only for optimal ICA)
- Import the list of bad channels for each subject that you made manually and make vector of these channel indices
- Perform ICA on the EEG data; **prerequisites for ICA**: no EOG channels, no a priori detected bad (e.g. flat, broken) channels, no trials with clear artifacts, no pre-stimulus baseline-corrected trials
- Save the ICA weight matrices in a separate file
- Check the components with the EEGLAB GUI; you could note down the components that represent artifacts manually at this step, or use ADJUST in a later step

postICA.m

- Import the EEG file created with the `preICA.m`
- Import the file with the ICA weights matrices
- Import the `rejected_trials` vector and remove the trials

- Import the list of bad channels for each subject that you made manually and make vector of these channel indices
- Make a new EEG structure variable without these bad channels and without the EOG channels
- Put the ICA matrices into this temporary EEG structure
- If you have not detected artifact-ICs manually with the EEGLAB GUI, perform the ADJUST algorithm on this temporary EEG structure
- Remove the selected components (if you do not use ADJUST, import a text file with manually noted ICs2remove)
- Put these cleaned data back into the original EEG structure
- Interpolate the bad channels
- If necessary, do horizontal eye-movement detection, save the `rejected_trials_heog` vector into a separate file, and remove the trials
- If not used anymore, remove the EOG channels
- Split the EEG structure into several fields of an ALLEEG structure, corresponding to the experimental conditions
- If necessary, run the CSD (Laplacian) algorithm on the ALLEEG data
- Save the cleaned and condition-separated data