# Automated pre-processing explanatory document

Contents

[Automated pre-processing explanatory document 1](#_Toc80193845)

[1. Overview 2](#_Toc80193846)

[2. How do I use the pipeline? 2](#_Toc80193847)

[3. What software do I need to have installed? 7](#_Toc80193848)

[4. How should the files be set up? 7](#_Toc80193849)

[5. What pre-processing is performed? 8](#_Toc80193850)

[Pre-processing w/ PREP (Script\_Y1) 8](#_Toc80193851)

[Event correction (Script\_X1b) 10](#_Toc80193852)

[ICA preparation (Script\_X2) 10](#_Toc80193853)

[Run ICA (Script\_X3 and Script\_X3b) 10](#_Toc80193854)

[Manually select ICA components to remove (User input) 10](#_Toc80193855)

[Remove ICA (Script\_X4) 11](#_Toc80193856)

[Bin Epochs (Script\_X5) 11](#_Toc80193857)

[Artefact rejection (Script\_X6) 11](#_Toc80193858)

[X7\_Extracting and exporting epochs 13](#_Toc80193859)

1. Overview

Intention is to build a single, general purpose pipeline that performs high quality pre-processing for most of the use cases we have in the lab. It can be adjusted for specific instances, but the general goal is to make most common parameters “set and forget,” particularly for first time users. It integrates the processing steps of ERP CORE (Kappenman et al, 2021[[1]](#footnote-1)) and the PREP pipeline (Bigdely-Shamlo et al, 2015),[[2]](#footnote-2) using the ‘Cleanline” non-invFFT approach (<https://www.nitrc.org/projects/cleanline/>). The good thing is that you don’t need to know anything much about these packages – you just set the parameters you care about, and/or the properties of your data (e.g. device it was collected on) and just press go. This will involve making changes to a single “configuration” document (an excel spreadsheet in the current version). This can be a little daunting at first because you need to make all the decisions at once. But once you’ve used it a couple of times, you’ll notice that most of the values stay the same and that for whatever your use-case is, you will probably only interact with a few parameters.

This is because the pipeline takes as many decisions out of your hands as possible. That means it makes data-driven decisions about e.g. which channels need interpolation, which reference to use, and so forth. If you want more fine-grained control of this, the sub-functions are each thoroughly commented, and in most cases you could just add a field/value to the configuration file to allow extra control. Of the values that need to be entered, most have sensible default values so you are often safe leaving the file as is. In many cases, the processing stream allows subject-specific specification (because e.g. not all people blink with the same intensity). This will involve updating one more file. The pipeline stops at the point when processing would normally diverge (epoched and baseline-corrected data), and thereby allows you to be more experimental in your analysis of the pre-processed data. This approach also minimizes the number of values that need to be specified in advance.

## 2. How do I use the pipeline?

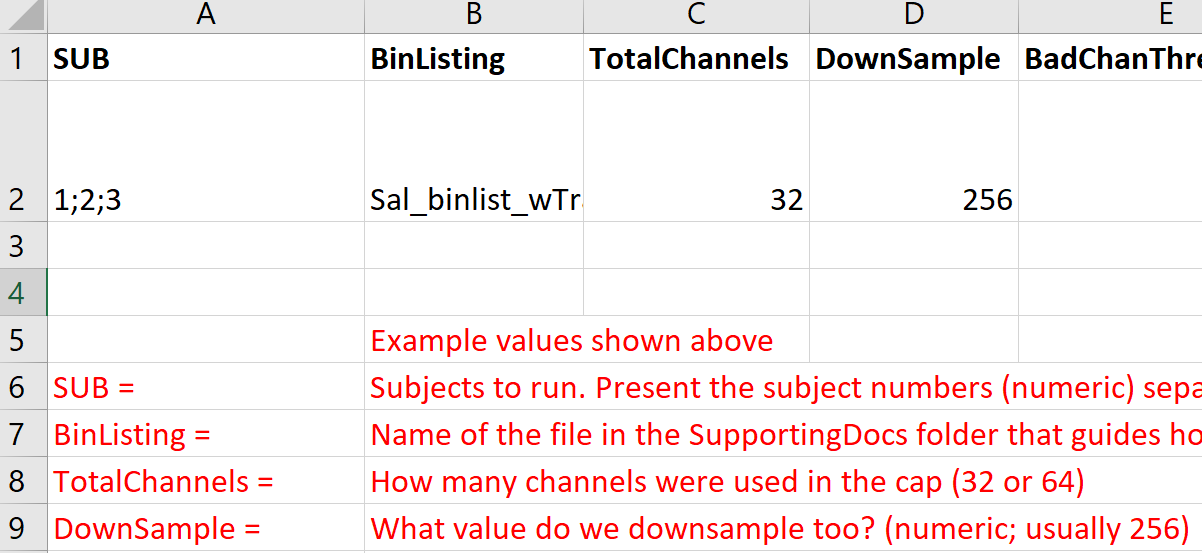
The general approach is to just set up your raw files in the expected file/folder format, then adjust a small number of values in two files (a configuration file and/or a “binlist” file), then press GO. This will generate a first pass process of all data files. The processing stream generates a number of images along the way to indicate what it has done to the data at key decision points. For example, it shows you a spectral plot to indicate effect of frequency filtering and it shows you a spatial map of the components obtained after performing ICA. You will almost always have to make manual decisions about ICA components, so the pipeline stops at ICA and weights for your input. Other data images can be checked to identify any outlying files (e.g. particularly noisy files) that require tinkering with their specific processing parameters. These can then be adjusted individually, and some/all files can re-submitted to the processing pipeline. Specific processing steps and decision points are detailed below.

Ok, but how exactly do I start? Once the software (EEGlab and plug-ins; see Section 3) are installed and the raw data is set up in the appropriate folder structures (see Section 4), the pipeline is a 8-step process from start to finish. Here are the steps:

**1. Create a binlist file for your experiment.** The “binlist” document is how you connect (i) the abstract, numeric trigger codes that were sent to the EEG amplifier during the experiment, with (ii) the relevant parts or conditions of your experiment. For example, you might have an experiment in which there are trials in which a loud sound is played (and the code 37 is sent to the amplifier) and other trials in which no sound is played (and the code 45 is sent to the amplifier). In order to output the data separated into Sound/NoSound trials, you need to tell the pipeline which codes to look out for, and what each code means. You do with a binlister document. It has a standard syntax (see here for syntax [link](https://github.com/lucklab/erplab/wiki/Assigning-Events-to-Bins-with-BINLISTER)), but often can be made quite simply. Open up notepad and create text file (e.g. binlist.txt) and follow the guide below. Save this file in the “SupportingDocs” folder and enter the name of this file under the “BinListing” column in the Configuration document (which we will now write in Step 2).

|  |  |
| --- | --- |
| **Text in binlisting document.** | **Explanatory notes.** |
| Bin 1 | First line starts with “Bin” and numeric ID |
| SoundTrial | Text description of this bin |
| .{37} | Event locked to onset of code 37 is written by “period, open brace, code, close brace”. |
|  | Space between bins |
| Bin 2 | First line starts with “Bin” and numeric ID |
| NoSoundTrial | Text description of this bin |
| .{45} | Event locked to onset of code 45 is written by “period, open brace, code, close brace”. |
|  | Space between bins and so on… |

**2. Create a configuration file for your experiment.** This is the most complex step as it requires you to make all the decisions about your experiment in one go. But once it is written, everything becomes fast and easy. If you’re doing this for the first time, it’s best to start with the configuration template. Go to SupportingDocs and open up “Config\_TEMPLATE\_180821.xlsx.” You will see a header row (SUB, BinListing, TotalChannels, etc.) and then a row of data below that. Below those two rows there will be a heap of information in red text. It will look something like this:



The second row is the important row. You only need to change values in that second row. The text in red below the two critical rows is just there to give you some help in figuring out what to put in each cell in row 2. For example, you should type in the name of the binlisting file you made in step #1 into row 2 below the header “BinListing” (e.g. cell B2). When you have figured out what to put into each cell in row 2 (some may be left blank; see the notes in red), then you need to (i) delete all the text in red (there should only be two rows in the file when it is finished) and (ii) save the file in the SupportingDocs folder with a new name, e.g. “Experiment2\_configuration.xlsx”. Write that down because you’ll need to know the name of this file for the next step.

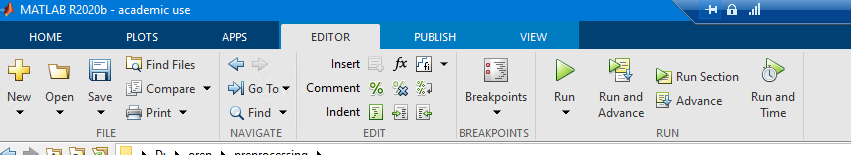
**3. Open the Master file** (“X0\_MasterFile\_XXXXXX.m” where XXXXXX refers to the date that file was updated). At the very top (around line 5) you will see a place to type in the name of your configuration file (just its name, not its full path). Replace the pink text with your configuration file name.

>> ConfigFileName = 'TheNameOfMyConfigFile';

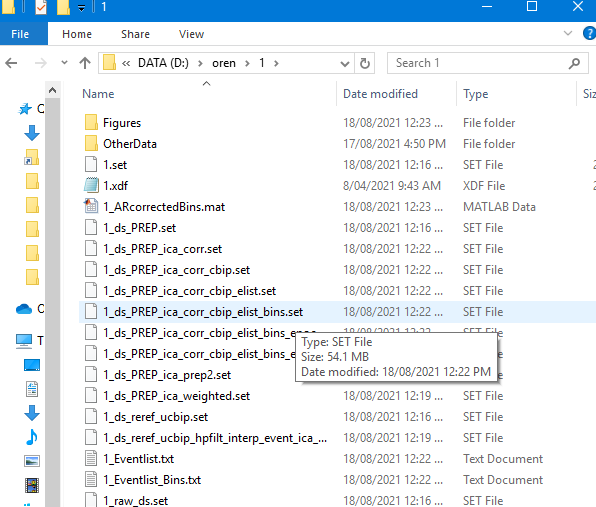
**4. Select which mode to perform** in a similar way by changing the pink text on line ~8. You have two choices: ‘PreICA’ or ‘PostICA’. If this is new data, start with PreICA.

>> ModeToPerform = 'PreICA';

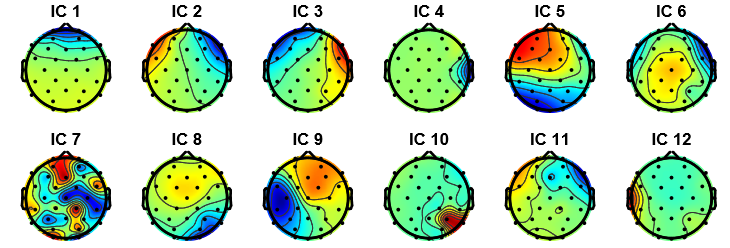
**5.** Once you’ve selected the correct config file and mode, **click on Run (green triangle)** and if it prompts you to change the working directory, then say yes.



**6. It will now run for a while and only stop when it has performed ICA decomposition**. It stops here because it needs you to manually interpret that output. Along the way it generates a bunch of new files, data and figures, most of which you will not need to review. It puts all the new files in each participants’ individual data directory as it goes. So for example, the “1” folder will start to fill up and look something like this:



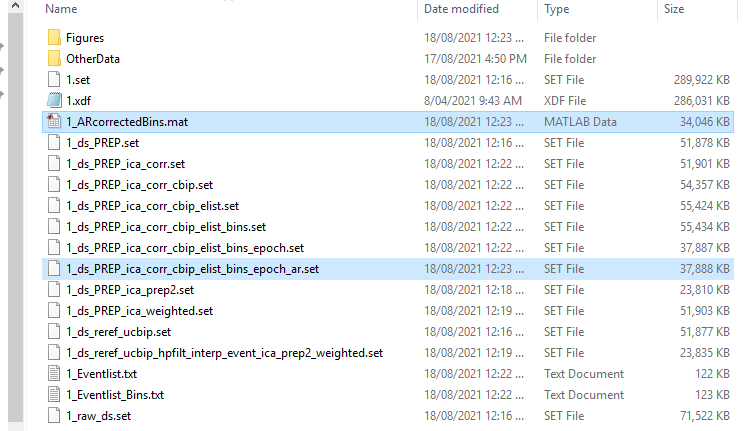
Each file represents the output of some part of the pre-processing stream. In general you don’t need to open these files, but they are nevertheless generated in case you need to check that processing has occurred as you expected. The file you need for ICA interpretation is in the “Figures” folder, and is called: “X3b\_XX\_ICA\_Weights.pdf” (where XX is the participant ID). Open that file and you will find topoplots of the components observed in the data. It will look something like this, but with anywhere from 10-64 heads. Your job is to identify the components you need to remove. Typically this involves removing 1-4 components associated with eye-blinks (like IC#1 below) or horizontal eye-movements (like IC#2 and IC#3 below). Ask your supervisor to help if you’re not sure how to identify these components.



When you have found the components you need to remove, you need to tell the pipeline which ones they are. You do this by navigating to the “SupportingDocs” folder (in the preprocessing section) and opening a spreadsheet called “ICA\_Components.xlsx”. That spreadsheet will have been created by the pipeline, but will be blank. You need to fill in the components you wish to remove per participants. So if that topographic plots from the last image were from Subject #1, we would look for the row with “1” in the left-hand column and enter the identifiers for IC#1-3 in that row. Specifically, we would enter “1” in the next cell to the right, “2” in the cell after that and “3” in the fourth cell. Once the components that need to be removed have been entered for every participant, save and close the file. And then return to matlab.

**7. Open the master file** (“X0\_MasterFile\_XXXXXX.m”), switch the mode to “PostICA” (see step 4 if unsure how), and press Run.

**8. Your preprocessing is now complete.** There pipeline outputs the data in two formats. First, it outputs the data in the EEGlab standard format. This is comprehensive, but can be a bit tricky to use if all you want is epoched data. The EEGlab format per person is located in their individual data directory, and is called something like “XX\_ds\_PREP\_ica\_corr\_cbip\_elist\_bins\_epoch\_ar.set” (where XX is the participant ID). The other output consists of raw data epoched into the separate bins that you need for your experiment (the definition of each “bin” is made in the “binlister” document in step 1). Specifically, if you requested 10 bins in your binlister document, then you would get a 10 cell array (called “GoodTrials”), with each cell containing the relevant EEG data in the standard [channels x samples x epochs] format. Only epochs that passed the artefact rejection criteria are included in each array. This is a much smaller file and can be easier to analyze later because it is already organized into bins. You can find this data in the .mat file called “XX\_ARcorrectedBins.mat” in each participant’s directory. The two output file are highlighted in the figure below. You can also look at the quality of all the epochs included in the outputted files by opening the “Figures” folder and looking for bin-specific .png images. That way you can check you’re actually getting clean output data. The names of the files you need will start with “X7” (as that is the function that generates them). That’s it. You’re all done now.

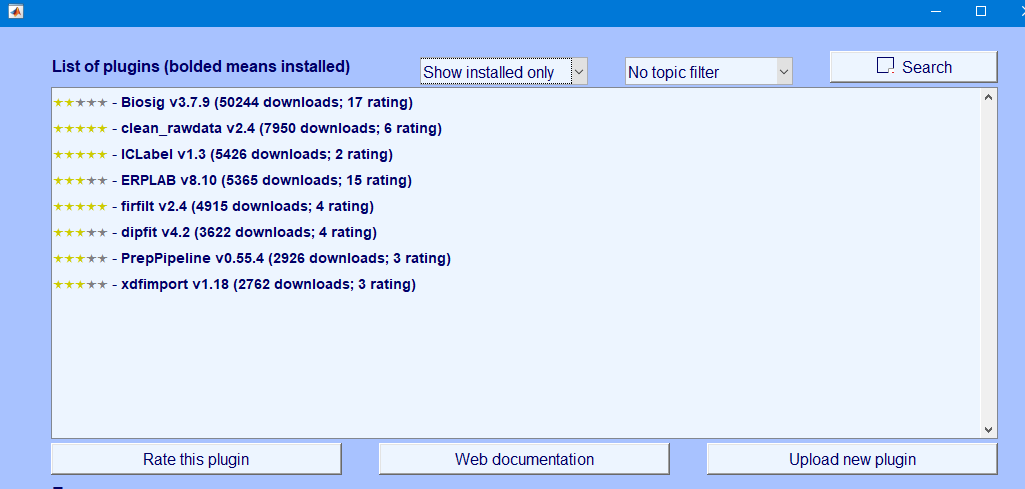


## 3. What software do I need to have installed?

The pipeline requires Matlab to be installed (IDS can help you install that if it is not already on your computer). It requires a package called “EEGlab” to be installed on top of that (full instruction here [[link](https://sccn.ucsd.edu/eeglab/downloadtoolbox.php/download.php)]). Finally, it needs several plug-ins/add-ons for the standard EEGlab package. To install these, you have to have EEGlab installed properly first. Once that is in place, open matlab and in the command window type “eeglab” and press enter:

>> eeglab

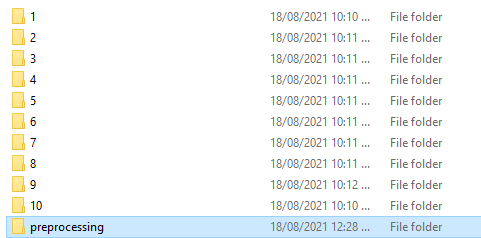
A blue window will pop-up. Navigate to “file/manage EEGlab extensions” and then install/add the extensions needed. The picture below shows all the extensions you will need.



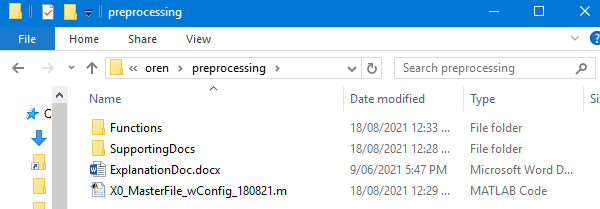
Once EEGlab and all the necessary extensions are in place, you’re good to go.

## 4. How should the files be set up?

You should create a base directory somewhere on your local hard drive (it doesn’t matter what that’s called). Everything runs much faster on your local computer. In that base directory, there should be a series of folders that each represent one dataset (i.e. one participant’s data). The name of the folder for each person should be the same as the name of the raw data file you wish to process. That is, in folder “1” there should be a raw data file called “1.bdf” (or “1.xdf”) which is the raw data you want to process. There should also be a folder called “preprocessing” that contains all the commands you’ll need to run the pipeline. You can just copy over the “preprocessing” folder from the R: (no need to adjust it). Overall it should look something like this.



To check that all is in order in the “preprocessing” folder, check that it looks like the picture below. There should be the top level .m file that you’ll open, make changes to and run. The name of this file will start with “X0\_”. There should also be an up-to-date version of this file (“ExplanationDoc”), and two folders full of supporting files and functions.



## 5. What pre-processing is performed?

### Pre-processing w/ PREP (Script\_Y1)[[3]](#footnote-3)

First the script, imports the data from raw format (.xdf or .bdf) into EEGlab (.set) format[[4]](#footnote-4). In the background, the import collects external and auxiliary channels, and offers a token reference for .bdf files to maintain SNR (you can easily change the reference channel later with no consequence). Then it downsamples the data to 256Hz (or whatever else is requested). Then it assigns channel labels (e.g. Fz, O1) and calculates electrooculogram (EOG and HEOG). It assigns spatial coordinates for all the scalp channels. This is also the place where unused external channels are discarded.

Then it commences the PREP toolbox. You can elect to use the whole pipeline (PREP == 1 in the configuration file) or you can just use a minimal version of the pipeline (PREP == 0 skips channel interpolation and robust referencing). In all cases, a DC detrend is performed first (default value: 0.5Hz), then the CleanLine protocol is used to remove line noise (50Hz and multiples). Then the processing diverges depending on whether you decided to apply PREP or not. If you apply the PREP package, “robust” referencing is performed. This is the magic step. Looks at each channel, one-at-a-time, and examines whether it is an outlier with respect to: global voltage, spectral properties, and the degree to which it is (un)predictable from its neighbours.[[5]](#footnote-5) Then calculates an average reference based on remaining channels and then applies that reference. It then checks to see whether there are additional outliers now and removes those. It repeats the process until no outliers are found. It’s important to do this iteratively because if an outlier is particularly large it will “move” the average reference towards its value, and thereby make that outlier less outlying (and other channels appear more noisy). Such effects should be reduced when that outlier is then removed in the second iteration, and then there’s more scope to find other, more subtle (but still problematic) electrodes that need to be removed.

At the end of this process, an average reference is calculated on the remaining (non-noisy) channels. (Actually, it uses median values by default, not average, so as to further limit any effects of outlying values; this is referred to as a “robust reference”). Removed channels are then spatially interpolated (spline), and the “common reference” (an average reference of correct channels) remains. Then applies a bandpass filter (Butterworth hardcoded, but [order, highpass, lowpass] specifiable in config file). There is scope in the configuration document to then switch the reference to something else – e.g. linked mastoids. In that case, the scalp data have been cleaned and interpolated, but the mastoid data (used as reference) will remain minimally changed from raw data (just detrended and line noise removed).

The PREP process generates a lot of output. In the “Figures” folder of each data directory, it will create a figure of the time series of the bad channels (ChansRemoved and ChansRemoved images), and one of the retained channels (GoodChannels image). It also creates figures of the averages of those good and bad channels (MeanOfBadChannels, Mean of Good Channels). Most usefully, it generates a topoplot (headmap) of the raw data and the post-interpolation data. This shows the mean activity across the whole time window, broken into four equal time windows (quartiles one to four), and importantly the upper and low plots are shown using the same scale so you can get a sense of whether the cleaning worked. This is in the CleanedTopos figure. It also generates data related to the removal of channels. This data is sent to the “OtherData” folder and includes information about how many iterations were needed, how many channels were removed, and which criterion each removed channel failed and so on. This information is only really needed if you are concerned that the algorithm is removing too many channels, or channels it should not have removed. Finally, the script displays the input and output of the bandpass filter applied (“BPFilter”) image.

**If PREP is set to inactive**, no interpolation or robust referencing is performed. By default an average reference of all scalp channels is applied as the baseline reference. This method still applies detrending, cleansing of line noise and a frequency filter (specified in the same way as noted above). The choice of an average reference can then be overridden by a specific reference choice (e.g. mastoids) if the config file requests it. This script produces much less output then the PREP version, but does display the input and output of the bandpass filter applied (“BPFilter”) image.

### Event correction (Script\_X1b)

This script does two things. It counts all the numeric trigger codes it finds in the raw data and outputs those counts as a .csv in the data directory. You can check here to see if the events you care about actually recorded in the raw data file (an absence of relevant event codes can cause errors later in e.g. the epoching steps). Sometimes biosemi records event codes as 256+N instead of 256 due to a bug in the matlab interface (still not sure why). To help figure out if this is a problem for your data, the script outputs a second tally with all values that are greater than 256 converted into the 0-255 range. Optionally this script does one more thing: it can move a subset of event codes by a fixed amount (measured in seconds). You should only do this if there is reason to believe that some (or all) of the important triggering events are off by a fixed time interval. If that is the case, you can nominate one or more numeric event code (usually 0-255) for the events that need to be shifted (look under “EventsToAdjust”) in the config file. If multiple codes, separate them by semi-colons (e.g. 122; 123). You can then shift those events (but not the others in the file) by a period measured in seconds (“TimeShift”) declared in config file. Currently no way to shift different codes by different amounts yet (but as a workaround you could run this function twice, with different codes nominated as the script only shifts the codes you specify).

### ICA preparation (Script\_X2)

This file is designed to set the data up for successful, speedy ICA decomposition. First it removes periods of inactivity by looking for periods of time that occur more than 2s away from a “relevant code” trigger. Currently needs the user to specify the relevant codes in the configuration file, or it will otherwise just use the whole data set. The “relevant codes” should be triggers that are sent in the course of doing the task so that there is good coverage of every trial (if the experiment includes many short trials). For long measurement windows, probably best to leave it blank. If no relevant code triggers are specified, the cleaning process just removes any period that doesn’t include a trigger for more than 2s. Then removes particularly noisy periods (indicative of e.g. participant movement or removing/adjusting cap) that will overshadow our efforts to find e.g. eyeblink artefacts. The algorithm also removes periods with abnormally large noise (indicating a break period). The parameters for this are automatically chosen (based on ERP CORE values) and are kept the same across all participants by default. You can have different values per participant. To do this, turn “CustomAR” on (i.e. set it to 1) in the configuration file, and manually create the “ICA\_Prep\_Values.xlsx” spreadsheet in the “SupportingDocs” folder.

### Run ICA (Script\_X3 and Script\_X3b)

Limits the file to only the scalp channels, and calculates the rank of the data using matlab analytical command (“rank”). Then ICA performed on the scalp data (slow) according to runica command (EEGlab selects all parameters), but limits the solution to the known rank of the data.[[6]](#footnote-6) Then script 3b draws a series of topographic plots of the components detected, ordered by rank (largest components first). You will need these for manual component rejection.

### Manually select ICA components to remove (User input)

Once steps X3 and X3b are complete, the script will pause. Then you will need to manually populate the file listing all the ICA components to remove. That file is automatically generated each time you run the overall script (unless you turn “CustomAR” off) and will appear in the “Supporting Documents” folder. The file is called “ICA\_components.xlsx”. It is structured with subject names/numbers as row headers (first column) and then all components to remove identified by a single number in each cell to the right of that header. If you don’t want to remove any components, just include the row header and no additional entries. To figure out which components to remove, look at the images created in step 3b (“ICA\_Weights”). Typically we seek to remove horizontal eye movements and blinks (ask about how to identify these if you’re unsure), but sometimes other components can be removed. Either way, this is where you will identify what to remove. Once the spreadsheet of components to remove is complete, then you can start the “PostICA” script and the remaining steps are performed.

### Remove ICA (Script\_X4)

Removes the components manually selected by the user. But does so on the complete continuous data set (i.e. puts back in the noisy bits that were removed, and returns all the original external channels). Additionally creates two new “raw” VEOG channels that were not subject to ICA (if you later want to search for and remove blinks in addition to ICA). Also generates a figure (based on channel Fp1) that looks at the efficacy of the ICA by comparing the blink activity pre-ICA with post-ICA (“RawVsCorrected\_VEOG”).

### Bin Epochs (Script\_X5)

Uses the information provided in config file about which event codes to create epochs (or “bins”) around, and the size of those epochs. The length of epochs are specified in ms, positioned relative to the triggering event (other possibilities will need to be manually coded in). For example, an epoch of 0.5s before an event and 2s after it would have values of “-500” and “2000” in EpochMin and EpochMax respectively. The baseline period is specified in the same way (BaselineMin and BaselineMax). Critically, the “bins” that the epochs are arranged into is specified by a custom binlister text file. You specify the name of that file under the value “BinListing” in the config file, and you need to place the binlisting file in the “SupportingDocs” folder, i.e. next to the configuration file. Generating a binlisting is a bit tedious, but simple enough once you get the hang of it (see here for syntax [link](https://github.com/lucklab/erplab/wiki/Assigning-Events-to-Bins-with-BINLISTER)).

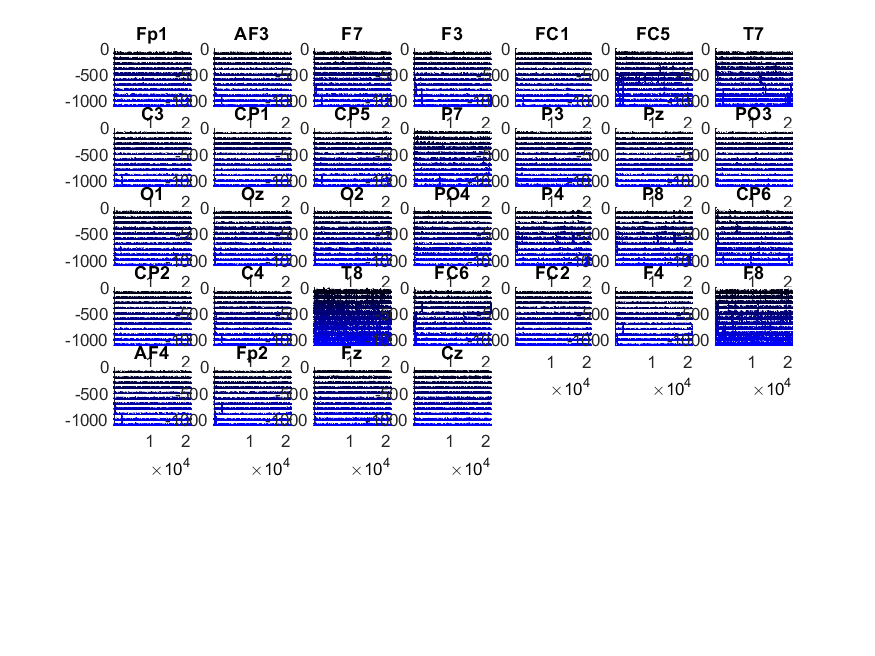
If non-numeric event codes are used (that is “CustomEpochs” are specified in the config file), these string entries will be found and converted into numeric codes (1001, 1002, 1003, and so on) and then the same routines are run. The assignment of numbers to string codes is based on the order of the strings specified in the config file. For example, [wordTrigger 1; wordTrigger2] would be assigned codes 1[001; 1002] and so on. Epochs are generated, meaning that continuous (channels by samples) data is changed into epoched (channels by samples by epochs) data. Simple subtractive baseline correction is performed over the resulting epochs, for the period specified in the configuration file.

### Artefact rejection (Script\_X6)

This step always applies a “simple voltage threshold” per epoch and can additionally apply the Luck et al step-based approach for finding eye-blinks. The latter step is only typically used for visual ERPs. There are a few key parameters in the configuration file for this step as you will likely need to adjust this between experiments, or even between participants within an experiment.

**Eyeblink removal:** If you want to include the eyeblink removal algorithm, then you need to set “Remove Blinks” to 0 in the configuration file. The algorithm will then look for blinks around the timelocking event (i.e. the trigger stimulus) using standard parameters taken from ERP CORE and applied to the vEOG channel (bipolar vertical electrooculogram channel, typically calculated as (EX5 – Fp1)). If you want to adjust these parameters, or if you want to apply different parameters for different participants, then it is possible to do so. You will need to turn “CustomAR” on in the configuration file (i.e. set it to 1). This will mean that the file used to guide this rejection process will need to be created manually. The file is called “AR\_Parameters\_for\_MW\_Blinks.xlsx.” The standard parameters from ERP CORE can be found in their files (with the same name) or you can look in the function that generates this file (“CreateIndividualARfiles.m”) to find the default values from which to adjust.

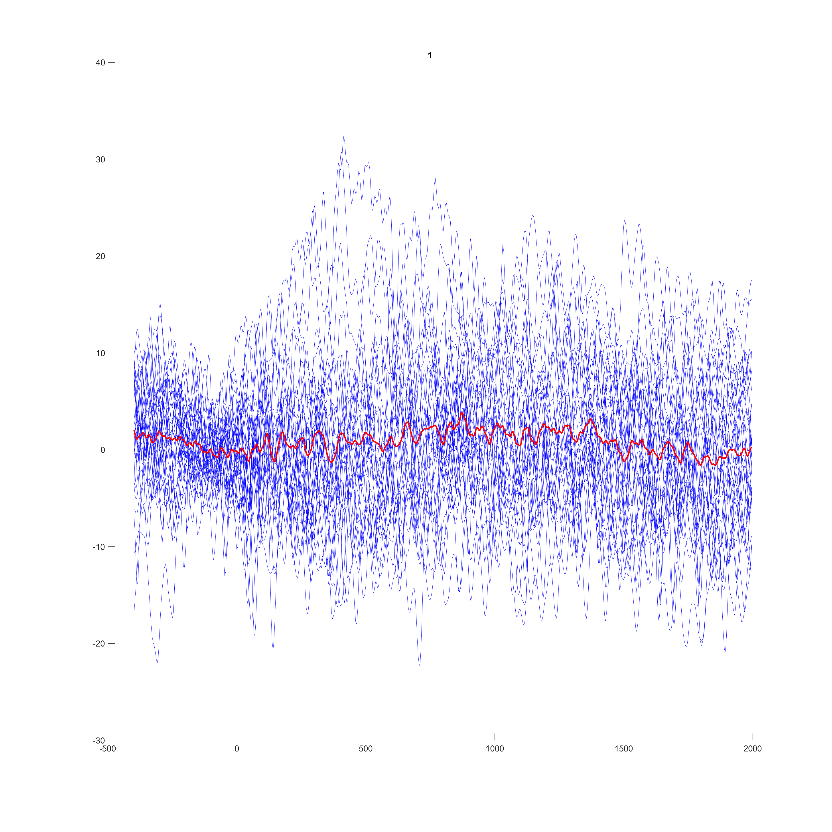
**Standard artefact rejection**: By default, the pipeline using a simple voltage threshold (SVT) to identify and reject epochs with excessive noise in them. The configuration file determines what value to use as the boundary threshold, under the field “BadChanThreshold.” For ERP experiments, this should be set to less than 200 microvolts but for other experiments (e.g. long window SSVEP tasks) it should be set so high as to be irrelevant. Once an above threshold period in one of the scalp channels is noted, the whole epoch is rejected. If you want to, you can manually include or exclude particular channels, or you can limit the checking to only part of each epoch (e.g. after stimulus onset). You can also change the threshold for some participants but not others. To do any of these things, turn “CustomAR” on in the configuration file and generate the “AR\_Parameters\_for\_SVT\_CRAP.xlsx” file in Supporting Docs manually. You can look in the function that generates this file (“CreateIndividualARfiles.m”) to find the default values from which to adjust.



**Visual feedback on artefact rejection**: Sometimes the artefact rejection process can remove a lot of your data, and it is not always obvious how appropriate it was to remove that data. Perhaps just one channel is noisy and escaped interpolation in Stage 1. If so, you might be better off to exclude that channel from AR rejection and thereby retain X% more of your data. Or perhaps the threshold is too high or low for a particular person, thereby resulting in either keeping noisy data (too high) or removing clean data (too low). How can you tell what has occurred? After performing artefact rejection, the pipeline creates two figures per person. Both figures are located in the “Figures” folder in each participants’ data directory. One figure “X6\_XXXX\_ARperChannel.pdf” shows what proportion of epochs were rejected for each channel. (The XXXX here refers to the simple voltage threshold applied to the data). So if there is a single bad channel, there should be a single spike in this figure. The other figure “X6\_XXXX\_ARbyDecile.png” shows a more complex figures (shown above). Each channel is shown as a subplot. Within each subplot, the epochs shown represent the cleanest (black and at the top) to the noisiest (blue and at the bottom) epochs ordered by cleanliness (specifically they are organized by maximum value relative to the SVT). No matter how many epochs there are in your experiment, there will always be ten epochs shown. The ten epochs each represent a 10% change (a “decile”). For example, if you had 120 epochs in your experiment, the top line is the 12th cleanest trial (10% of 120 trials), the next line is the 24th cleanest trial (20% of 120 trials), and so on. Any lines shown in red represent epochs that have been rejected for exceeding the threshold in that channel. Typically the red lines will be at the bottom of the figure. You can use this figure to determine whether you need to ignore some channels in AR step for each person, or whether you need to adjust the rejection threshold.

### Extracting and exporting epochs (Script X7)

The final step is included to supplement the standard EEGlab output format. The EEGlab format is quite detailed, but does not organize the data according to bins. That is, it groups all the epochs together in its EEG.data structure (even the data from separate conditions) and only disaggregates in a fairly complex manner when it needs to e.g. draw a figure. That can be a bit of a pain when doing analyses either in regular Matlab or an external program (JASP, Excel, R, etc). So this step separates the raw data (that were not rejected due to having artefacts) into the experiment-related bins that you identified in the binlister file (e.g. separates data from Sound and No Sound trials in the example given in Step 1). It outputs the data in a cell array called GoodTrials.The size of the array depends on the number of bins you declared in the binlister file. Within each bin, there is a single matrix with 3 dimensions: channels, samples, epochs. This is the standard format for representing EEG data. This file is saved as “XX\_ARcorrectedBins.mat” in each person’s data directory (where XX is the participant ID). The file also creates a grand average for each of the bins and plots the grand average at three midline channels (Fz, Cz, Pz) so you can get a sense of what their average waveform will look like for the epochs (actually “bins” here) that you’ve chosen to analyse. This figure is called “X7\_XX\_BinGrandMeans.pdf” and is located in the Figures folder of the participant’s subdirectory (XX refers to participant ID). This one overall figures is supplemented with N additional ‘.png’ image files, where N is the number of bins you’ve requested (in your Binlister doc). For each of these N bins, it will draw a line figure of every epoch overlaid on each other (in blue) and the mean for that epoch (in red). It looks a bit like this. By default, it will show you this data from Cz, but you can specify the channel you care about by declaring “KeyChannel” in the DataConfig file. This will give you a sense of how noisy your data are epoch-to-epoch, and whether you need to change your artefact rejection criteria.



1. Kappenman, E. S., Farrens, J. L., Zhang, W., Stewart, A. X., & Luck, S. J. (2021). ERP CORE: An open resource for human event-related potential research. *NeuroImage*, *225*, 117465. [↑](#footnote-ref-1)
2. Bigdely-Shamlo, N., Mullen, T., Kothe, C., Su, K. M., & Robbins, K. A. (2015). The PREP pipeline: standardized preprocessing for large-scale EEG analysis. *Frontiers in neuroinformatics*, *9*, 16. [↑](#footnote-ref-2)
3. This is the only subfunction that must be completed for each file sequentially. Why? Because the PREP pipeline uses multithreading for robust reference calculations, and you can’t nest “parfors.” So just have to do this step slowly, sorry. [↑](#footnote-ref-3)
4. This is really just a matlab .mat file in disguise and can be opened manually if needed. [↑](#footnote-ref-4)
5. The default value in the PREP package is that if a channel is correlated less than r = 0.4 over a whole 1 second with its neighbours it is an outlier. This may not be suitable for e.g. 32 channel montages as there is more space between electrodes and thus lower correlations may still be “real” activity. Need to inspect the behaviour of this AR method when used. Other default values are: high frequency noise is triggered if an FFT finds a signal of z = +5 relative to that below 50Hz. [↑](#footnote-ref-5)
6. This process means that you often arrive at fewer components than you have input channels, and there is some risk of poorly fitting the data as a result of this… but that risk isn’t large, and if you don’t pre-specify the rank then the algorithm fails whenever there is less than perfect independence (which is annoyingly common). [↑](#footnote-ref-6)