Manual

Contents

Preface

Before reading anything about RIDE, please watch this animation:

cns.hkbu.edu.hk/ride/animation.pptx

It highly concisely expresses the general idea of RIDE: Separating different ERP sub-components with different latency variability and reconstructing ERP.

RIDE is such a method for dealing with the limitation of conventional stimulus-locked averaged ERP that not all of the ERP sub-components are locked to stimulus-onsets, which, as a result, yields a blurred ERP.

In general, RIDE separates the ERP (with RT) to three component clusters: the stimulus-locked component cluster S, the response-locked component cluster R and the central, 'neither-nor' component cluster C. The separation paradigm can be extended to more components, say, two C components (if the pattern of ERP allows, for example, two clear humps in the late time window), or only S and R, or only S and C in the data without RT (from experiment without RT recorded). But the attempt to separate a larger number of C components is not suggested because it always complicates the problems and the reliability of separating large number of components will be compromised.

You are suggested to read the RIDE papers to get a deeper understanding of it. The basic idea of RIDE is firstly published on 2011 (Ouyang, et al.) which presented the initiation of RIDE framework and the first application of it. After two years of constantly upgrading, the algorithms of RIDE converged to a relatively robust version, all of the new algorithms are summarized in Ouyang et al., 2015. Briefly, the new RIDE employs several new modules such like L1-norm minimization to prevent serious distortion encountered by least-square-based algorithm. The RIDE algorithm was fixed since then.

About RIDE toolbox

RIDE toolbox directly operates data in the matlab environment. A basic knowledge about Matlab is required for working with RIDE. It is a package of matlab scripts. Once added in the Matlab paths RIDE functions can be called to process the EEG data that is also already loaded in the Matlab workplace.

The RIDE toolbox does not include any procedures about importing data from commercial EEG softwares (e.g., BrainVison, NeuroScan …) to Matlab. Actually the pipeline for importing data into Matlab is already very well developed in EEGLAB (http://sccn.ucsd.edu/eeglab/).

RIDE is only expected to apply on EEG data that is after the procedure of artifact rejection. The artifact rejection can be either done in commercial softwares or EEGLAB. In any case, before running RIDE, make sure that the data that was fed into RIDE is artifact-cleaned. One can simply have a glance on the time courses of EEG single trials for all electrodes to see whether the data is roughly clean. Some examples for data with clear unremoved artifacts are shown in Figure **1**.

Preparations

- Add the folder RIDE_call to the path of Matlab.

- Export the EEG data from the commercial softwares (e.g., BrainVision, NeuroScan, etc.) separately for EACH subject and EACH condition after strict ARTIFACT rejection; failure/improper of artifact rejection (Figure 1) would affect RIDE results to different extents. Export the data into segmented form, (e.g., from -200 ms to 1200ms after stimulus onset). Better export the data as epoched version, in this case there will be a 3-d matrix named ‘data’ under ‘EEG’ after you used EEGLAB to import it into Matlab (see below).

- Read the data by EEGLAB (e.g., file -> import data -> using EEGLAB functions and plugins -> From Neuroscan .eeg file). Note: EEGLAB may evolve with different interfaces in different versions. But one can always easily figure out how to import data exported by various commercial software to Matlab (download from http://sccn.ucsd.edu/eeglab/).

- If commercial softwares only export data separately for a single subject but combining all the conditions, one need to divide the data into different conditions with a little bit Matlab script and then feed the data for each condition into RIDE. If there is reaction time in each single trial, one also need to obtain the RT for each single trial (if one wants to separate R component). The extraction of data and RT for single conditions also can be done by Matlab scripting, which may be difficult for user with little Matlab background. There is an exemplary data and script for extracting data and RT for a single subject in the folder ‘../RIDE_call/examples/Snippet for data and rt extraction’.

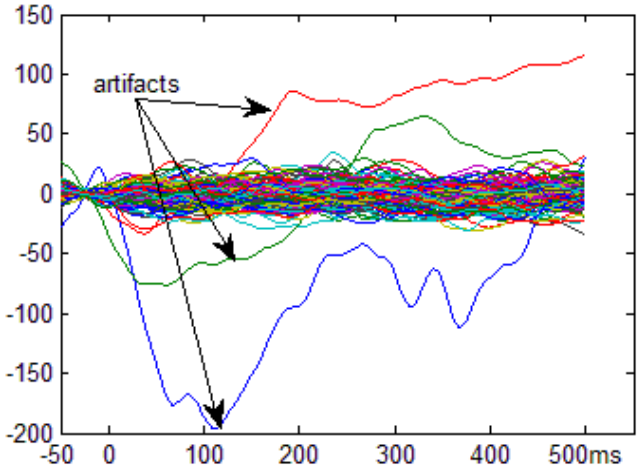
- Prepare the data in a 3 dimension matrix, e.g., data[600*38*82], of which the first dimension is the sampling points (600 time points), the second is the channels (total number 38), the third is the trials (here the trial number is 82). If the data from EEGLAB is not in this order, use the Matlab function ‘permute’.

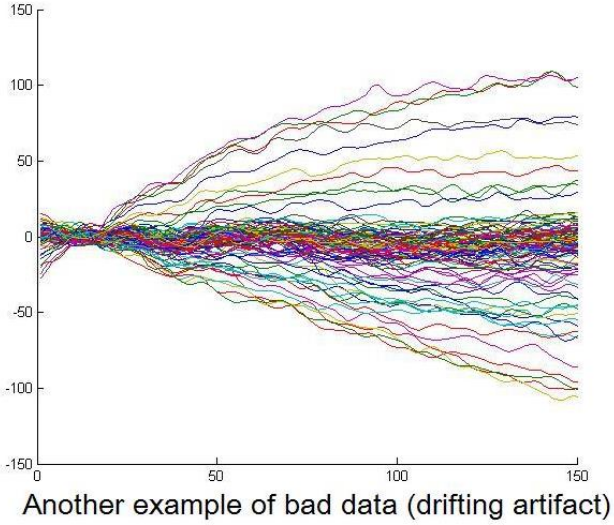
- Data has to be baselined.

- Discard the ‘non-brain’ channels, e.g., M1,M2,VEOU,VEOL,EOG,ECG etc.

- If the data is with reaction times, make a new vector containing the reaction times, e.g., rt [82*1], the length of this vector must be consistent with that of the third dimension of data. If the information of rt vector does not match the data trial by trial, one could not extract a corrected R component but probably a noisy pattern. To

simply check this, one can plot the pattern of R component when the application was done. The value in the vector of 'rt' must be in the unit of millisecond. If this experiment is without RT recording, then ignore this step.





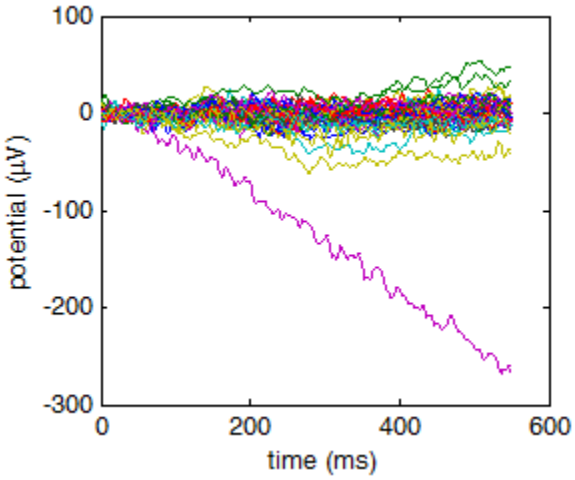


Figure 1 Failure of proper artifact rejection. These plots are the superimposed single trials from a single electrode.

Start Riding

- Once the data is ready in the workspace of Matlab, then start running the following script to run RIDE. For testing, one can load the example data from RIDE_call\example\samp_face.mat. The data is for one subject from a face recognition task with reaction times (Herzmann et al., 2010).

- The basic script for running RIDE is as follows:

```
cfg = [];%initialization
cfg.samp_interval = 2;
cfg.epoch_twd = [-100,1000];
cfg.comp.name = {'s','c','r'};
cfg.comp.twd = {[0,500],[100,900],[-300,300]};
cfg.comp.latency = {0,'unknown',rt};
cfg = RIDE_cfg(cfg);


results = RIDE_call(data,cfg);
```

The structure variable 'cfg' contains all of the parameters that will be fed to RIDE_call function applying on the 3-d matrix 'data'. The above script only shows the basic parameters that are compulsory for RIDE_call function. Other optional parameters will be summarized later.

As long as you have prepared the data in the Matlab workspace, you can run the above script in Matlab and get the results of RIDE outcome.

`cfg.samp_interval:` a single value – the time between each two consecutive dots of the data, in the unit of millisecond. For examples, if 2ms (500 sampling rate), set it as 2; if 4ms (250 sampling rate), set it as 4;

`cfg.epoch_twd:` 1*2 vector – time window of you data epoch (the first dimension of the 3-d data), e.g., if it is from –100ms to 1000ms, set it as [-100,1000]; (here is in the unit of millisecond). It's recommended to prepare data with longer time window than what the ERP is supposed to take place to reduce the boundary distortion of RIDE-components. For example, suppose the significant ERP waveform covers from about 100ms to about 900ms (by visual inspection, for example see Figure 2), than you can prepare the data from -100ms to 1100ms for example. Note: involving too long redundant data is of course also not necessary and will increase the computation time. cfg.epoch_twd(1) must ≤ 0.

`cfg.comp.name:` a cell variable containing the names of the components you want to decompose. For a typical data with RT, usually separate three components, namely, 's', 'c', 'r' as in the example script. For a data without RT, one can simply omit 'r'. One can also only separate S and R, i.e., just input {'s','r'}. For more sophisticated applications, one can for example, separate more than one C components: 's', 'c1','c2','r'. (One should always name R component as 'r' as it is relevant to RT retrieval in the RIDE internal programme).

`cfg.comp.twd:` The latest RIDE algorithm uses time window function to refine the waveform of each RIDE component. The use of time window function (refer to Ouyang et al., 2015) is to constrain searching and extraction of RIDE components from the time window that the components are supposed to occur. The determination of component time windows is mainly based on the pattern of ERP waveform. For example, in the case of Figure 2 the time window of S is set to be from 0 ms to 500 ms with the assumption that the stimulus-locked component cluster should be in the early time range. The time window of C is set to be from

100 ms to 900 ms based on the visual inspection of the data of the central component. The time window of R as -300 ms to 300 ms, which is supposed to be efficient to cover RT-locked component cluster. It should be specially explained here that unlike that for S and C, the values of time window input for R is not relative to stimulus onset, but to reaction time. That is why there is negative value.

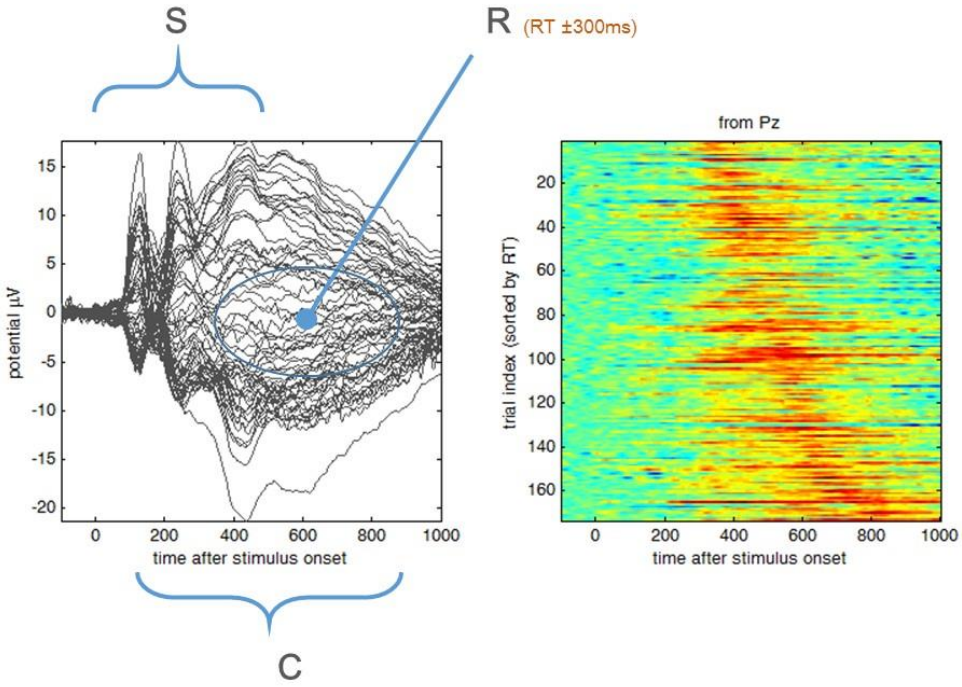


Figure 2 Determination of time window from the pattern of ERP course. For R component the time window is simply based on the assumption that there is a component locked to RT within a reasonable time window (e.g., RT±300ms).

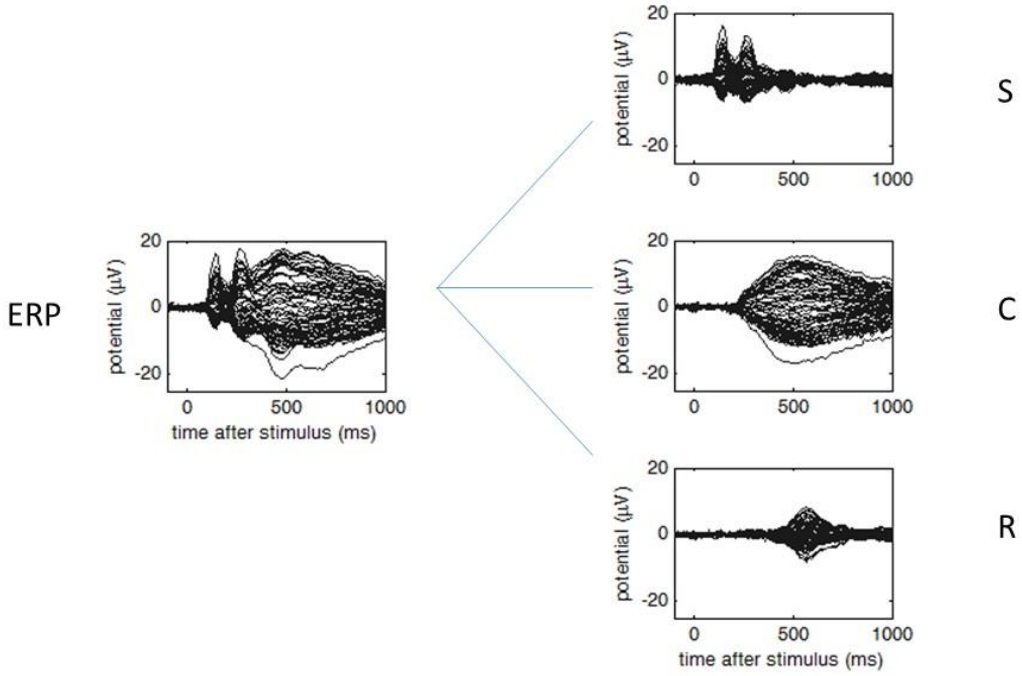


Figure 3 RIDE separation based on the specified time windows above.

cfg.comp.latency: The latency information for the component one wants to separate. In the example, the latency of S component is all-zero vector (with a length of trial number, or just a single 0) and the latency of R component is the vector of reaction times. The latency of C component should be set to be 'unknown' as it is to be estimated. For the data without RT, just omit the third cell.

So far, everything is ready. Run the script and all of the outcome of RIDE will be output to the structure variable 'results'. When RIDE was being run, there will be some information output to the command window showing the progress of the processing, e.g, how much iteration step for each channel by the RIDE definition of convergence. RIDE is only supposed to be applied to each single subject and single condition, separately. In principle, these parameters have to the consistent for all subject and conditions. After the application on all of the subjects and conditions, one can collect and results and do the subsequent analysis. All the above operations can be organized into a batch code for processing all of the subjects.

The above example is the most basic usage of RIDE. In the following are the additional and optinal parameters for more advanced usage.

Optional inputs:

cfg.re_samp: single value – re-sample the data to lower resolution in order to increase the computation speed, e.g., if sampling interval is 2, one can set it to be 4, 6, 10, etc. Usually it is not necessary to process the data in a high resolution (e.g, 1 or 2). But there is also a WARNING: re-sample to too-low resolution could lose essential information. The final outcome of RIDE components is recovered to original sampling rate by interpolation.

cfg.high_cutoff: single value. For estimating the latency of C, the single trial ERP and the cross-correlation curve for estimating the latency of C was low-passed filtered in order to diminish the bias from the high frequency oscillation in the background EEG noise. Note that the filtering is only applied on the data when the latency of C is being estimated by the cross-correlation method. Default: 4;

cfg.dur: cell. How much lags allows for searching C latency by template matching. Example: cfg.dur = {0, 300,0} allowing ±300 ms for searching C components. Note this corresponds to the separation scheme cfg.comp.name = {'s','c','r'}. The first the last '0' are nonsense because they are for S and R.

The RIDE results:

After running RIDE (typically less than 5mins for 1 subject with 60 channels and 100 trials), there is a variable call 'results' generated in the Matlab workspace, where all of the RIDE outcome is contained:

- erp: the original ERP with the size of [epoch length*channel number].

- s: S component separated by RIDE, with the size of [epoch length*channel numbers]. The epoch length covers the time window of the data you prepared, i.e., the same with time range of ERP. S component is baselined with the mean values of the segment of [0-200ms] being set to be the same with that of ERP.

- c: same as above. This C component is the latency-synchronized and averaged time course, which is different from the c_sl later (stimulus-locked averaged) in the following. Latency-synchronized means all the single trials of C components are synchronized to the mean latency and averaged, and the waveform is located at the most probable latency (here the median). C component is baselined on [0-200ms] based on the assumption that there should not be significant activity in the early time window.

- r: same as above. This R component is synchronized to the reaction time and is put at the most probable reaction time (here the median). So this 'r' represents the time course of R component in the trial where the reaction time is equal to the median RT.

Note: Since different subjects have different median reaction times, when one wants to get the grand average of R component across subjects, it's better to resynchronize the R components for individual subjects to the grand mean again. Otherwise, if one directly grand means this 'r' across different subjects, the grand average is again a convolution (blurred form) but not the proper waveform reflecting the time course of R (operations of re-synchronizing R can be easily conducted by Matlab scripting. One can also refer to the plotting script in 'Example datasets and plotting' folder. For each subject, shift the R with a distance based on the difference between the median RT of this subject and the median of median RTs for all subjects, so that the R waveform can be synchronized subject by subject. Moving edges can be compensated by zero-padding). R component is baselined on [0-200ms] based on the assumption that there should not be significant activity in the early time window.

- s_sl: the stimulus-locked average of S component (the same with s).

- c_sl: the stimulus-locked average of C component (the blurred version).

- r_sl: the stimulus-locked average of R component (the blurred version).

- latency_c: the latencies of C component for each single trial relative to the template of C component ('c' above) at the unit of millisecond. The median is zero. So, the negative latency of trials means that the C components in these trials occur earlier than the 'c', vice visa.

*The latency of C component for all the single trials reflects the latency variability of the central component (namely, C component cluster) across the single trials. It is an important indicator of individual's brain. One should pay attention to this information if one is studying the response variability of brains. In Matlab std(latency_c) reflect this variability. And since the latency_c here are relative values, it is meaningless to compare the mean latency across conditions. (only the std(latency_c) is comparable across conditions).

- latency_r: just the reaction times but after subtracting the median (at the unit of millisecond). To retrieve the original RT, one can refer to results.latency0, third cell.

- amp_s: the amplitude of S component for each single trial and each single channel estimated by covariance between the S component and single trial after removal of other RIDE component. So the size is [trial number * channel number]. The covariance is only calculated within the time window specified for each RIDE component, and it is calculated after the application of RIDE time window function (Ouyang et al, 2014).

- amp_c: same as above.

- amp_r: same as above.

- erp_new: the reconstructed ERP by summation of 's','c' and 'r' which are the RIDE components locating at the most probable latency.

Hints and Cautions

- Before apply RIDE, pay attention to the quality of your data (please refer to Figure **1**).

- Do not involve 'non-brain' electrodes. It is important that the 'non-brain' electrodes are excluded before running RIDE.

- Do involve sufficient length of EEG data to avoid boundary distortion on each side. For example, involve [-200ms, 1200ms] for an ERP data with waveform vanishing before 1000 ms.

- When specifying the time window for each RIDE component, do involve sufficient time window covering the component to be extracted. For example, Figure **4** left shows the single trial ERPs for a single channel sorted by RT. It is obvious that latency-variable component covers the length from about 300ms to about 900ms. So for this data it's proper to specify at least from 300 ms to 900 ms for the time window to extract C component. Involving a little bit longer (e.g., [200ms,1000ms]) could be better because the time window function for extracting component is Tukey time window which has intrinsically attenuated the boundary. Another hint of choosing time window for C is by choosing the boundary of the 'hump' (for example). On the other hand, do not extend the time window to extremely long (for example, [0,1500ms] for C) with the expectation to safely cover single trial C component because including too much irrelevant part of C would make the latency estimation un-precise (irrelevant segment would contain more noise of which the patterns may mimic C component which makes the latency estimation worse).

- If one is not sure about the significance of separating a C component cluster in his/her data, one can always simply start with separation of two components: S and R (only when there are reaction times). Separating S-R and reconstructing ERP can also effective correct the blurring.

- After the separation, all of the statistical testing can be re-done on the re-constructed ERP.

- In some ERP experiment (e.g., reading), the interval between each adjacent stimulus is always too short (e.g., 600ms), therefore, the ERP for the next stimulus is overlapping with the late part of the ERP for the present trial. Like Figure **4** right, it's obvious that the ERP for the next trial already emerges after about 800ms. If the ERP for the next trial is seriously overlap with the present trial, RIDE may have problem in extracting C.

- RIDE can separate a C component when it already 'visually' exists (i.e., clearly visible, for example, see Figure 3, Figure **4** left, Figure 5 right) but due to the latency variability its waveform is somewhat blurred and its amplitude is attenuated. Data with too weak late ERPs (Figure 5, left) are not suitable for applying RIDE. Because too weak late component could make C component undetectable. Or if the signal to noise ratio is extremely low, it will lead to more arbitrary generation of C component by arbitrarily capturing background noise.

- In principle, the trial number for RIDE should be equal or larger than the component number to be adequate for separation. However, it is preferred that the trial number should be large in order to diminish the effect of noise. In general, 40 trials (for each subject and each condition) is about the minimum standard to have to proper outcome (empirical suggestion).

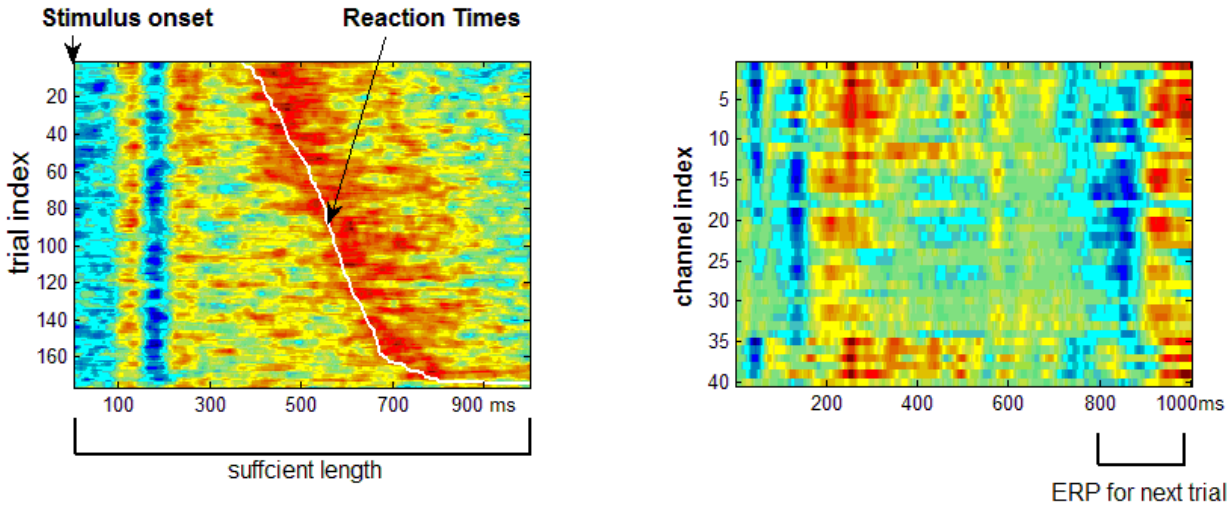


Figure 4 Example data. Left: Smoothed single trials ERP from Pz channel sorted by RT. Right: A single trial of ERP data from reading task.

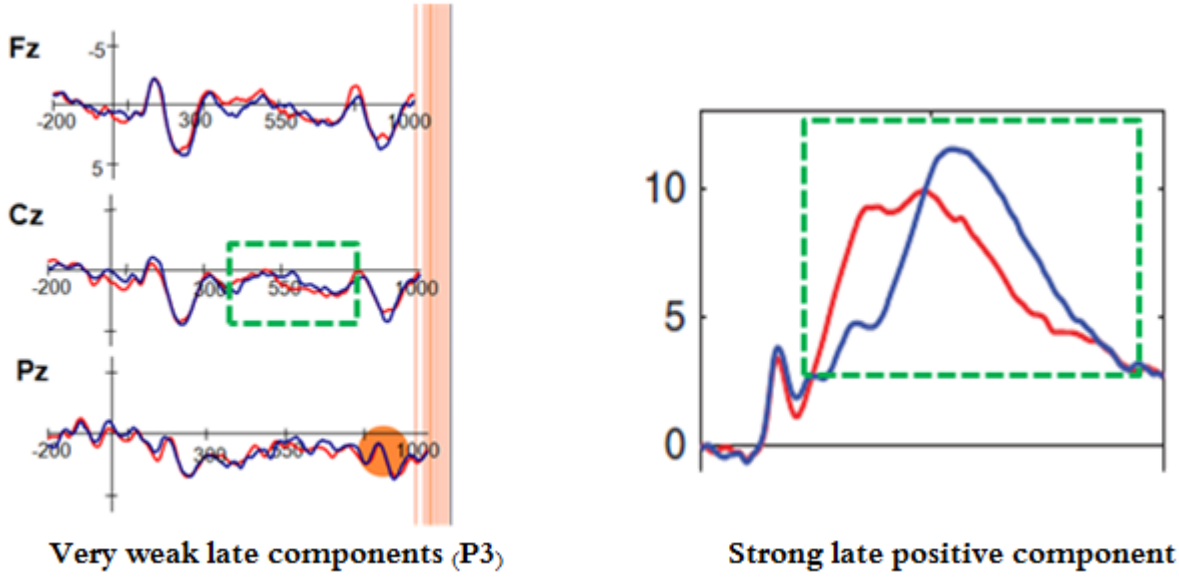


Figure 5 Comparison of weak ERP (left, not good for RIDE) and strong ERP (right).

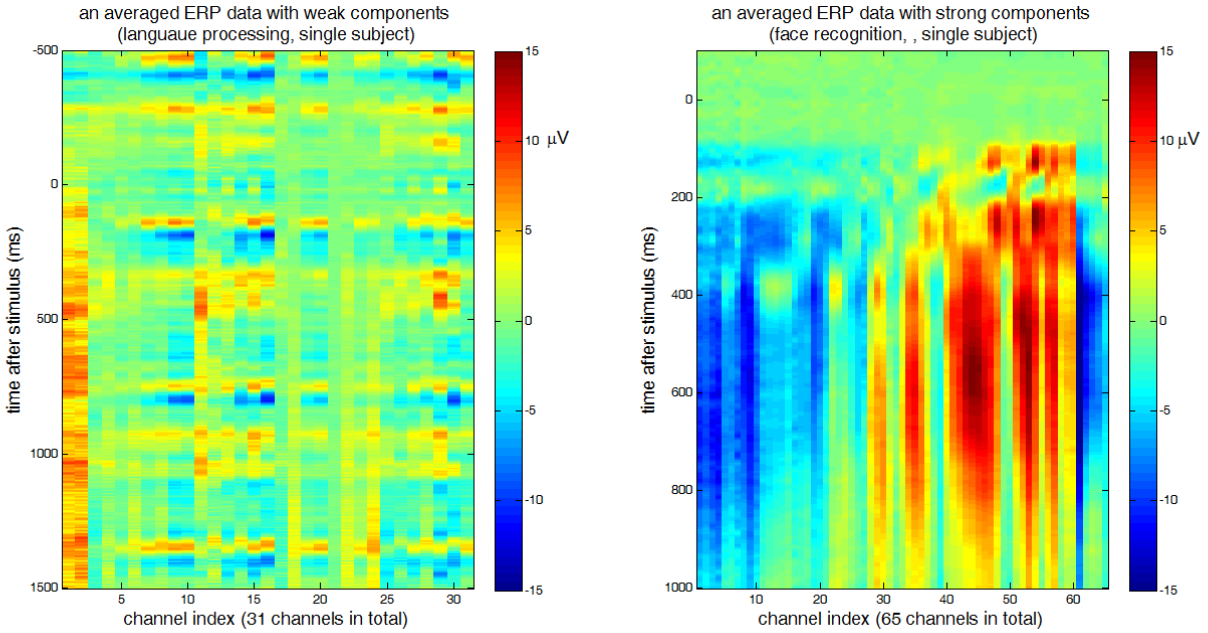


Figure 6 Another example. Comparison of weak ERP (left, not good for RIDE) and strong ERP (right).

- It is not recommended to separate 'lots of' C components. In one of our work, we attempted to separate two C components respectively capturing N400 and P600 from semantic processing task (we specified [200ms,600ms] and [400ms,800ms] as the time window for extracting them. However, when one tries to separate more C components, the time window for extracting each C component would be shorten. Therefore the information of waveform of each C component for latency estimation would be more limited for precisely capturing the corresponding sub C component from noisy EEG activity. Moreover, different C component might also tightly mingle with each other which additionally raise problems. So, generally, it is not recommended to separate 'many' sub C component. Separating more C components additionally required the sub C component to be distinct from each other in the waveform so that they do not affect each other when the latency is estimated by cross-correlation.

- Figure 7 shows a good example of separating two C components.

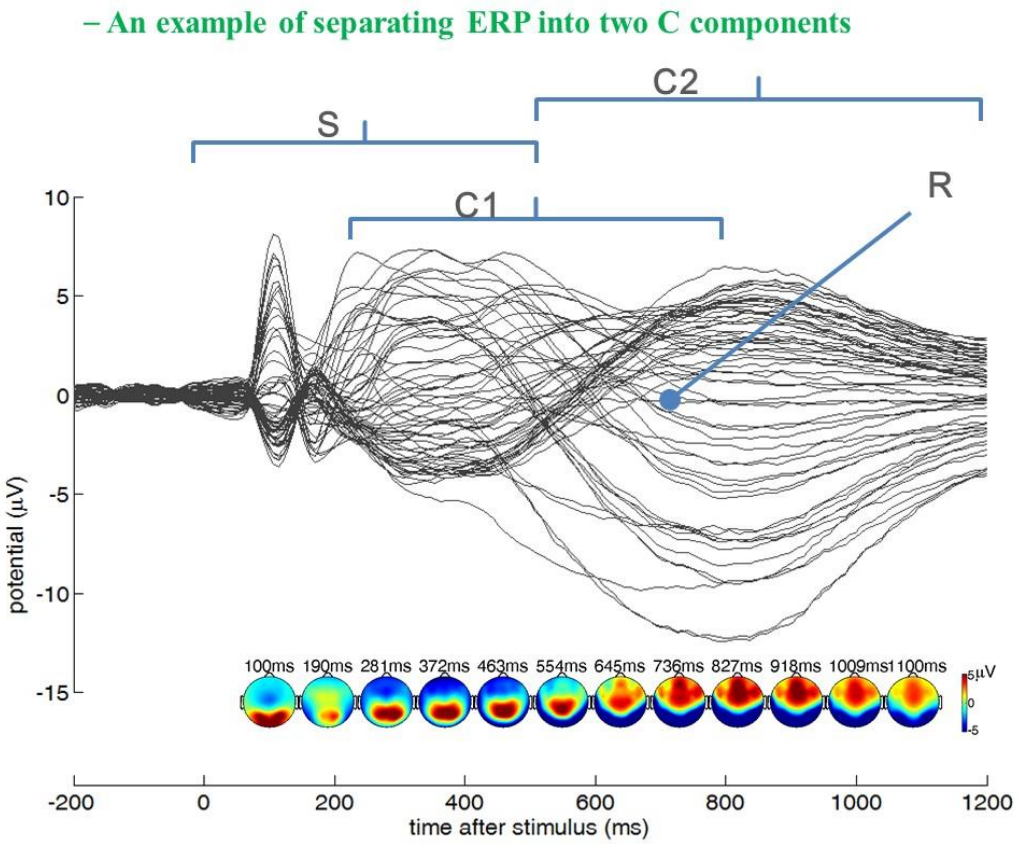


Figure 7 An example of separating two C components.

- C is aimed for capture conspicuous and highly variable component, like P3. Althogh, in principal, the early ERP component should be also variable in latency. But the variability may not allow its latency to be reliably estimated and not do the component to be isolated as a latency-variable component. For example, for P1-N1 complex, some might attempt to dissociate N1 from P1 by specifying a time window at the N1 range to extract a C component from it. However, as N1 is always tightly locked to P1 component, most likely N1 will be separated to S as well as P1 as a cluster. So, C component is always for capturing conspicuous component from relatively late time window (e.g., Figure **4** left).

- Whenever specifying time windows for any component, always extend it a little bit at both side because 1) the time window function used in RIDE is by Tukey time window by which the edges are shrank out; 2) there is subject-to-subject variability.

- Although each RIDE component is derived by specifying a certain kind of time window where it is supposed to occur, the final outcome of each RIDE component is still a continuous waveform covering the same time window as the epoch. Because in the algorithm of RIDE, the time window function is released at the final iteration (Ouyang et al., 2014).

Useful Plottings

Please find the demo from 'RIDE_inplementation' slide. It is in the last section 'Practice'.

References

Herzmann, G., & Sommer, W. (2010). Effects of previous experience and associated knowledge on retrieval processes of faces: An ERP investigation of newly learned faces. Brain research, 1356: 54-72.

Ouyang, G., Herzmann, G., Zhou, C., & Sommer, W. (2011). Residue iteration decomposition (ride): a new method to separate erp components on the basis of latency variability in single trials. Psychophysiology, 48(12), 1631-1647.

Ouyang, G., Schacht, A., Zhou, C., & Sommer, W. (2013). Overcoming limitations of the ERP method with Residue Iteration Decomposition (RIDE): A demonstration in go/no-go experiments. Psychophysiology, 50(3), 253-265.

Ouyang, G., Sommer, W., & Zhou, C. (2014). A toolbox for residue iteration decomposition (RIDE)—A method for the decomposition, reconstruction, and single trial analysis of event related potentials. *Journal of neuroscience methods*. In press.

Ouyang G, Sommer W, & Zhou C (2015) Updating and validating a new framework for restoring and analyzing latency-variable ERP components from single trials with residue iteration decomposition (RIDE). *Psychophysiology*. In press.

Stürmer, B., Ouyang, G., Zhou, C., Boldt, A., & Sommer, W. (2013). Separating stimulus-driven and response-related LRP components with Residue Iteration Decomposition (RIDE). Psychophysiology, 50(1), 70-73.