Universität Stuttgart

Institute for Artificial Intelligence

Socially Intelligent Robotics Lab

Universitätsstraße 32
70569 Stuttgart

Masterarbeit

Implement the RIDE Algorithm Into the Unfold.jl Toolbox

Till Prölß

Studiengang:	Msc. Softwaretechnik
1. Prüfer:	Jun.-Prof. Dr. Benedikt Ehinger
2. Prüfer:	
Betreuer:	René Skukies
begonnen am:	11.09.2024
beendet am:	11.03.2025

Implement the RIDE Algorithm Into the Unfold Toolbox

Till Prölß
Supervised by: René Skukies and Jun.-Prof. Dr. Benedikt Ehinger

March 11, 2025

Contents

Abstract

Event-related potentials (ERPs) are commonly used in EEG research to study brain responses to specific stimuli. Traditional ERP averaging methods are affected by trial-to-trial latency variability, which can distort the extracted components. Residual Iteration Decomposition Estimate (RIDE) is an established method designed to correct for this variability by separating components based on their latency distributions. While effective, its original MATLAB implementation lacks flexibility and integration with modern EEG analysis frameworks.

This work presents a Julia-based implementation of RIDE and its integration into Unfold, a regression-based deconvolution toolbox for EEG analysis. Two versions were developed: Classic RIDE, which replicates MATLAB RIDE's behavior, and Unfold RIDE, which replaces iterative decomposition with regression-based deconvolution. Both implementations were evaluated using simulated datasets with known ground truth and a real-world EEG dataset to assess their accuracy and compare them with MATLAB RIDE.

The results show that Classic and Unfold RIDE successfully reconstruct ERP components, producing reasonable results in both datasets. However, differences exist when compared to MATLAB RIDE, likely due to missing features and variations in filtering, latency estimation, and iteration behavior. Despite these discrepancies, the findings confirm that Unfold's regression-based approach is a viable alternative to iterative decomposition, offering potential advantages such as improved modeling flexibility.

Future research should focus on implementing missing features and achieving equivalence with MATLAB RIDE. Additionally, randomized testing with UnfoldSim could provide systematic validation and help optimize the algorithm. This work lays the foundation for further improvements in EEG deconvolution by combining RIDE's latency estimation with Unfold's flexible regression framework.

The implemented algorithms are available as an open source project on GitHub: `https://github.com/unfoldtoolbox/UnfoldRIDE.jl`

Chapter 1

Introduction

Understanding the inner workings of the brain has long been a goal of neuroscience. One of the most widely used methods to study brain activity is Electroencephalography (EEG). By placing electrodes on a subject's scalp, EEG records electrical signals generated by the brain. This technique is widely used in both clinical and research settings, from diagnosing epilepsy to investigating how the brain processes information.

An influential method for analyzing EEG data is the concept of **event-related potentials (ERPs)**. ERPs are brain responses triggered by specific stimuli or events. The underlying principle is that if a certain brain response is systematically elicited by a stimulus, it should be detectable in the EEG recording. However, in practice, EEG signals contain a significant amount of unrelated neural activity, making it challenging to isolate individual responses. To extract ERPs, multiple trials are aligned to the **stimulus onset** and the average is calculated. This process reduces random background noise while enhancing stimulus-related activity.

Although averaging works well in many cases, it has major limitations. If a component does not always appear at the exact same time across trials, averaging will cause it to become blurred or *"smeared"*. Additionally, when multiple components overlap, simple averaging cannot separate them. This thesis addresses these challenges by looking into different algorithms with the ability to compensate for these phenomena.

1.1 Electroencephalography (EEG)

EEG is a non-invasive method for measuring electrical activity in the brain. By placing electrodes on the scalp, researchers can record voltage fluctuations caused by brain activity. EEG is widely used because of its excellent **temporal resolution**, allowing us to capture rapid neural responses in the millisecond range.

However, EEG data can be difficult to interpret. The recorded signals are a mixture of different neural processes, and external factors such as muscle movements or blinking can introduce unwanted noise. To extract meaningful information, researchers use specialized signal-processing techniques, such as the ERP—brain responses that are time-locked to specific events.

1.2 Event-Related Potentials (ERPs)

Event-related potentials (ERPs) are the brain's direct electrical responses to stimuli, such as a sound, image, or decision-making process.

1.2.1 ERP Formation Through Averaging

The fundamental assumption behind ERP analysis is that stimulus-related activity is **consistent** across multiple trials, while background noise is random. To construct an ERP, researchers segment the EEG into **epochs**—time windows that capture brain activity before and after the stimulus, which can be seen in figure 1.1. The individual epochs are shown at the top, all aligned to around the moment when the stimulus is presented to the experiment subject, also known as the **stimulus onset**. By computing the **average**, unrelated brain activity cancels out, leaving behind the stable ERP waveform near the bottom of the graph. The blue and red component now show up clearly and the the signal is visibly less noisy.

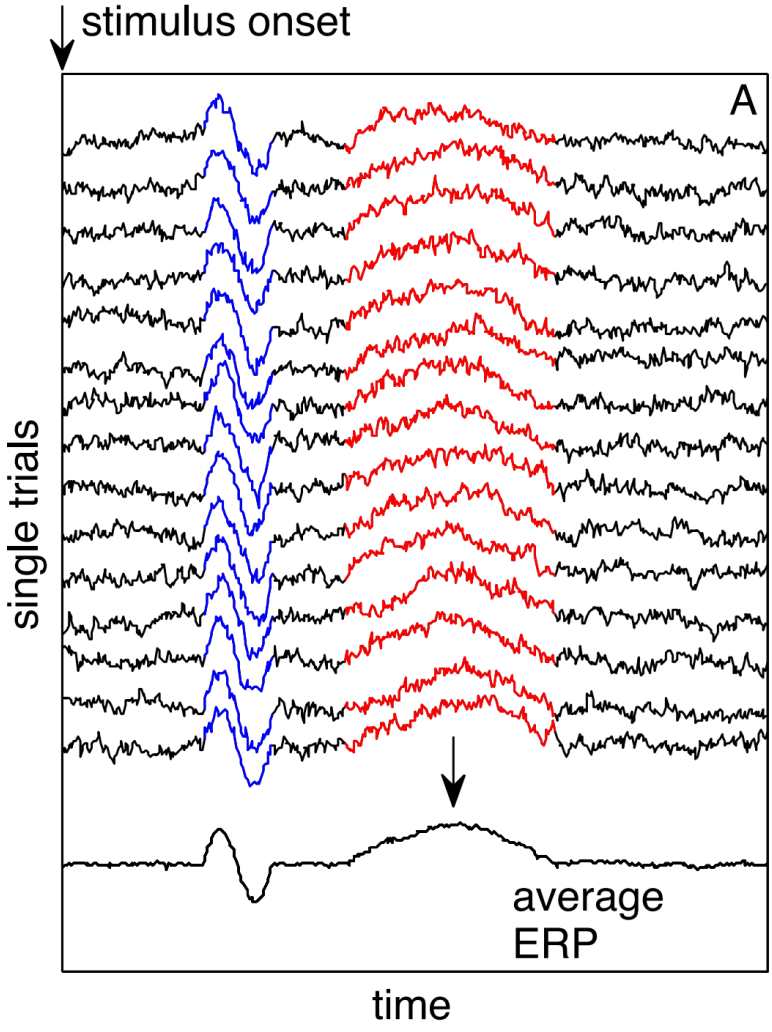

Figure 1.1: Construction of an ERP graph from multiple EEG trials. The top row shows single-trial EEG recordings, while the bottom row represents the final ERP after averaging. Reproduced from [19].

1.2.2 ERP Components

The ERP waveform consists of distinct **components**, which are specific peaks or troughs occurring at characteristic time points after the stimulus. Each component is thought to reflect some kind of underlying neural processing.

Common ERP Components and Their Interpretation

- **P1 (First Positive Peak, ∼100 ms)** – Related to early sensory processing. [25]
- **P300 (Positive Peak, ∼300 ms)** – Often linked to decision-making and stimulus. evaluation[15]
- **N400 (Negative Peak, ∼400 ms)** – Involved in language comprehension and semantic. processing[9]

Studying these components helps researches to gain insight into cognitive and perceptual processes.

1.3 Challenges of ERP Analysis

Event-related potentials (ERPs) are a useful tool for analyzing brain activity, but their interpretation comes with a number of challenges. Two major issues that affect ERP analysis are **overlapping components** and **variable latency components**. Both of these problems can distort the resulting ERP waveform, making it difficult to extract meaningful insights from the data.

1.3.1 Overlapping Components and Deconvolution

One issue with ERP averaging is that different neural responses often overlap in time. This happens when multiple stimuli are presented in quick succession or when different brain processes activate at the same time. Because the EEG is a mixture of electric fields, the resulting ERP is a mix of multiple overlapping components.

One solution to this issue is **deconvolution**. Deconvolution-based approaches, such as those implemented in the Unfold.jl [12] toolbox, aim to mathematically separate these overlapping responses. Instead of assuming that each trial contains only a single response type, deconvolution models the contribution of each component separately. This allows researchers to extract individual neural responses even when they occur close together in time.

1.3.2 Smearing due to Variable Latencies

Even when components do not overlap, ERP averaging faces another issue: **latency variability**[15]. In a typical experiment, we assume that a given ERP component appears at the same time in every trial, relative to stimulus onset. However, in reality, this is rarely the case.

When components shift in time from trial to trial, simple averaging causes them to become blurred or "smeared" in the final ERP waveform, as can be seen in figure 1.2. Instead of a sharp peak, we get a wider and lower-amplitude response, which does not

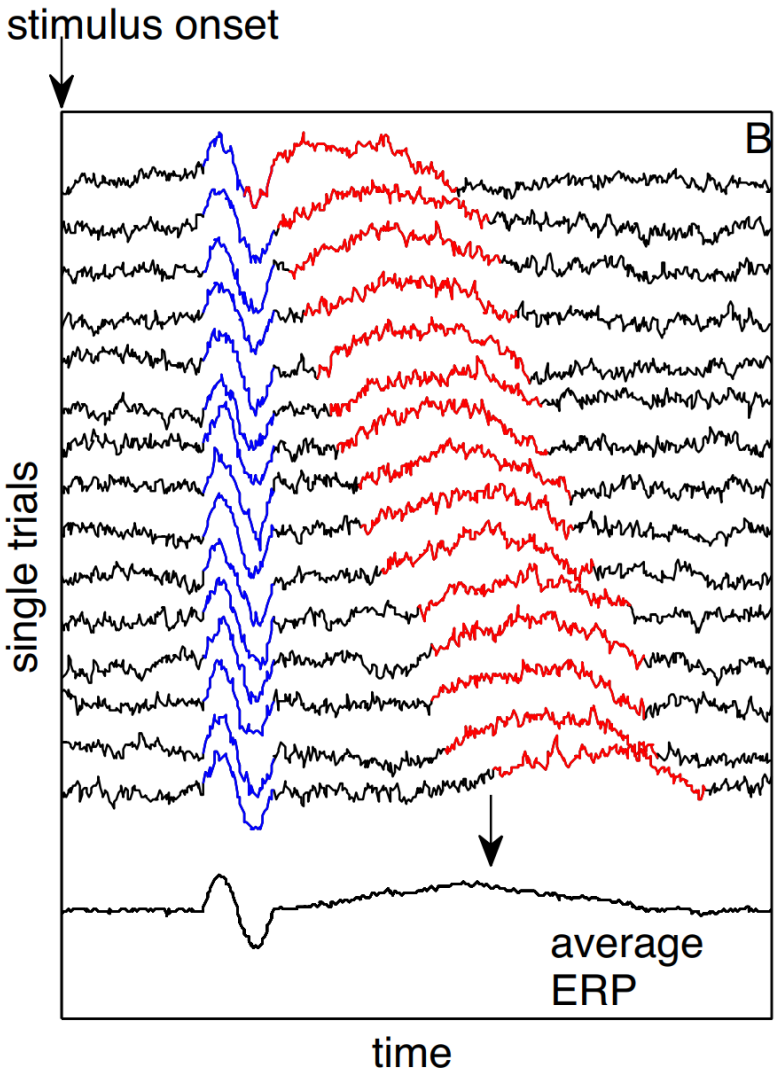

Figure 1.2: This figure shows an EEG graph with multiple trials and their computed average ERP at the bottom. The blue component cluster has a stable latency from the stimulus onset, while the red component cluster shows some variability. You can see the smearing caused by the inconsistency of the red component in the averaged ERP at the bottom, compared to the clear representation of the blue cluster. Reproduced from [19].

accurately reflect the underlying brain activity. The cause of this variability isn't certain, but it's usually attributed to a difference in processing time, reaction speed or differences in the applied stimulus [15]. Mostly later components, like the P300, are affected by this issues, which supports the hypothesis that a variable processing period is involved.

1.4 Residue Iteration Decomposition: The RIDE Algorithm

Residue Iteration Decomposition (RIDE)[20, 19] is a method designed to address the problem of latency variability in ERP analysis. Traditional stimulus-locked averaging assumes that all neural responses are time-locked to stimulus onset, but in reality, ERP components may exhibit trial-to-trial latency variability, leading to the above mentioned smearing. RIDE mitigates this by decomposing ERP signals into component clusters based on their latency characteristics, enabling a more precise reconstruction of neural activity.

1.4.1 RIDE Component Clusters

The core idea behind RIDE is that different ERP components exhibit different timing characteristics across trials. Instead of assuming a fixed timing relative to stimulus onset, RIDE classifies ERP activity into three distinct clusters:

- **S-cluster (Stimulus-Locked):** Components that consistently occur at a fixed time relative to the stimulus onset.
- **R-cluster (Response-Locked):** Components that are aligned with a participant's reaction time. The reaction time is measured during each trial and supplied to the algorithm.
- **C-cluster (Latency-Variable):** Components that do not consistently align with either stimulus onset or response execution.

1.4.2 RIDE Algorithm Steps

The RIDE algorithm follows an iterative approach consisting of several key steps, as illustrated in figure 1.3.

1. **Initial Latency Estimation**: The first step involves estimating trial-specific latencies using either Woody's algorithm or simple peak identification. These methods provide an initial approximation of C-latency values, which are later refined using cross-correlation.
2. **Initial S and R Component Extraction**: The S and R components are initially extracted through averaging, aligning each trial to its corresponding event marker. The S component is locked to stimulus onset, while the R component is locked to the response time. These averaged waveforms serve as initial estimates for the deconvolution step.

3. **Decomposition**: The purpose of deconvolution is to isolate and extract individual ERP components by systematically removing the influence of other components.

   The first step in this process is computing the residue. It is obtained by subtracting all other known components from the EEG signal, leaving only the remaining activity associated with the target component. For the first iteration, we want to extract the C component, so we subtract the S and R waveforms calculated in the previous step from the EEG signal.

   Once the residue has been computed, the relevant segment for each trial is extracted according to its estimated latency: stimulus onset for S, response time for R, and the estimated latency for C. These extracted segments are then averaged across all trials to obtain an updated estimate of the component.

   The deconvolution process follows a sequential order: first extracting C, then S, and finally R. The iteration must begin with the C because no prior estimate of C is available. Once all three components have been extracted, the process repeats, starting again with C, and continues iteratively until either a set iteration limit is reached or convergence is detected.

4. **Latency Estimation**: After deconvolution, the latency of the C component is refined using cross-correlation performed on the residue. The residue is obtained by subtracting the extracted S and R components from the original EEG data. The C component extracted in the previous deconvolution step serves as the template for this cross-correlation, allowing for a more precise estimation of its latency in the current iteration. This cross correlation process is displayed in figure 1.4. The updated latencies are then used in the next iteration of the deconvolution.

5. **Iteration and Convergence**: The deconvolution and latency estimation steps are repeated iteratively until convergence or reaching a set iteration limit.

1.4.3 Summary and Benefits

RIDE provides an advanced approach to handling latency variability and component deconvolution in EEG data. Unlike traditional ERP averaging techniques, which can cause temporal smearing of neural responses, RIDE allows for a more precise reconstruction of event-related brain activity by accounting for trial-to-trial latency variability.

RIDE has been successfully utilized across multiple domains to gain deeper insights into brain function. In functional neuroanatomical network analysis, RIDE has been used to investigate task-switching processes, specifically examining the variability between memory-based and cue-based task switching [26]. In the field of Brain-Computer Interfaces (BCI), RIDE has been employed to enhance EEG preprocessing, improving the quality of neural signals used for training machine learning models [6].

Additionally, RIDE has played a role in understanding brain-related conditions such as ADHD [5] and Tourette syndrome [14]. It has also contributed to cognitive neuroscience research by supporting investigations into the Theory of Event Codes [8, 24] and Flanker Task performance [16]. These applications demonstrate the algorithm's ability to refine EEG analysis, making it a valuable tool for both theoretical and applied research.

By improving EEG signal decomposition, RIDE enables more accurate interpretations of brain activity in both basic and clinical neuroscience.

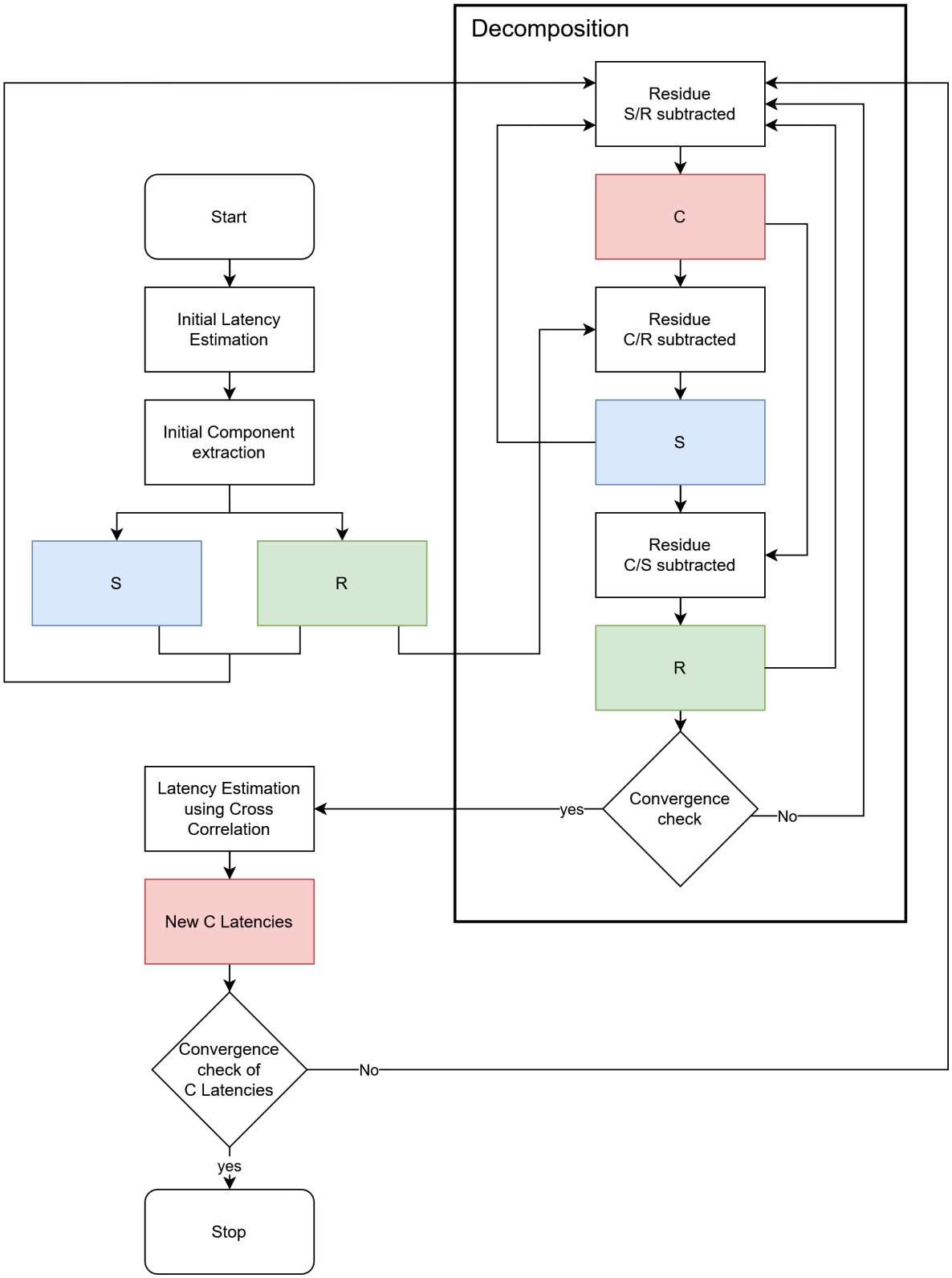

Figure 1.3: This graph shows the process of the RIDE algorithm and decomposition as it is implemented in MATLAB and Classic RIDE. The intermediate results of each step (S,R and C waveform) are shown as colored boxes.

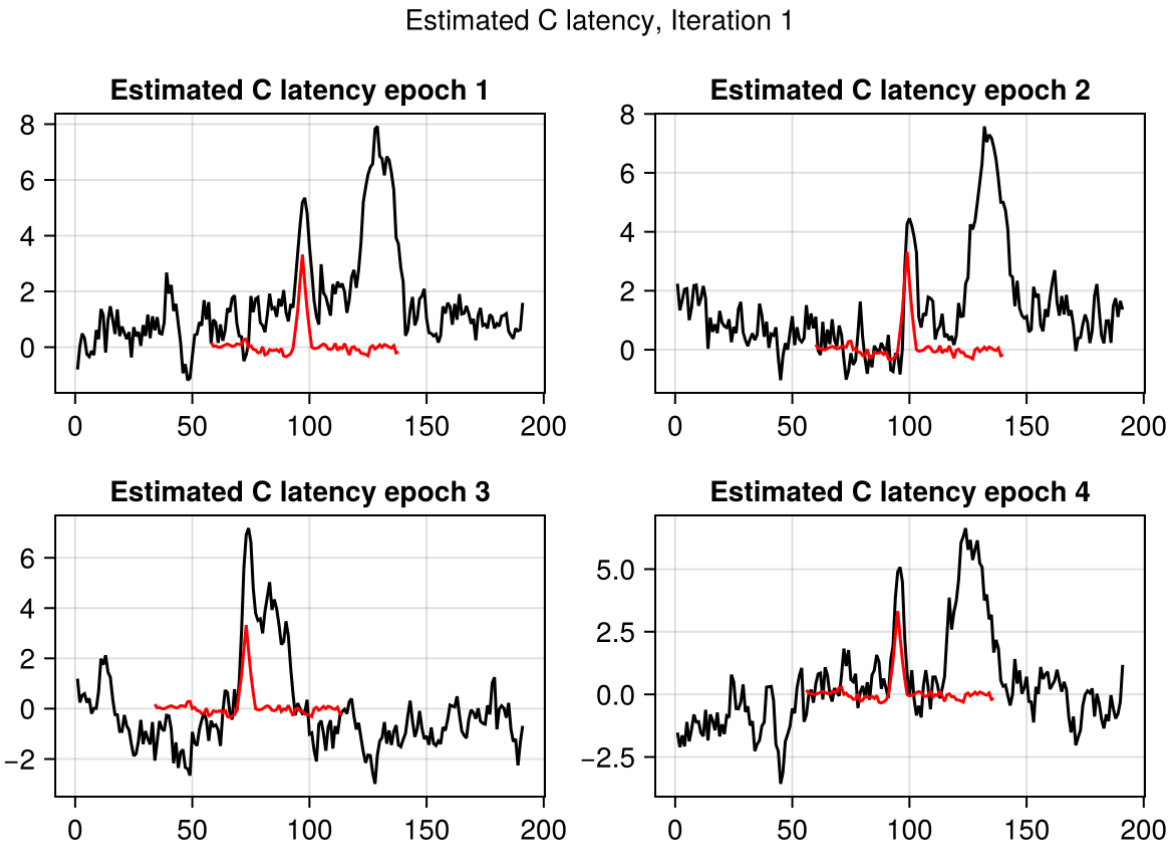

Figure 1.4: These graphs show one epoch each of the simulated dataset during the cross correlation process of Classic RIDE. This figure was used during the development process of the reimplementation to debug issues in the cross correlation. The black signal represents the raw data of the epoch. The red signal is the current waveform of the C component, painted in at the estimated latency of the last cross correlation.

1.5 Unfold

Unfold.jl [12] is a Julia-based toolbox that performs deconvolution using a regression-based approach. It was developed as a flexible alternative to conventional ERP averaging and provides tools for modeling EEG data with more control over analysis steps. Unfold doesn't contain any functionality to handle components with a variable latency and their associated smearing problem. A latency estimation like in RIDE is not implemented.

1.5.1 Differences Between Unfold and RIDE Deconvolution

The key difference between RIDE and Unfold lies in how they handle their deconvolution. RIDE separates components through an iterative decomposition process, refining the extracted waveform in each step until a stable solution is found, as described in section 1.4.2. Unfold, on the other hand, applies regression-based deconvolution, modeling ERP components directly using statistical estimation rather than repeated decomposition. The Unfold deconvolution process is displayed in 1.3 and described in detail in the associated paper [12].

While both approaches solve the same issue, Unfolds variant has several advantages:

- **Simultaneous multi-channel processing** RIDE processes EEG channels sequentially, whereas Unfold processes all EEG channels simultaneously, resulting in a considerable performance advantage.
- **Incorporation of covariates:** Unfold allows additional covariates such as room brightness to be included in the deconvolution model.
- **Ability to handle overlapping trials:** RIDE is unable to deal with overlapping epochs, while Unfolds regression based method is designed to handle them.
- **Advanced Deconvolution Methods:** Unfold provides more advanced functionality, like non-linear formulas and mixed models.

1.5.2 Usage of Unfold

Similar to RIDE, the Unfold toolbox [12] has been used in numerous studies to improve EEG data processing. One interesting avenue of study is the combination of EEG data with Eye tracking, which has been investigated by Dimigen and Ehinger et al. [10, 11]. They track the focus of the participant's eye to determine the moment when a particular image is noticed, which then corresponds to the stimulus onset for the EEG analysis. This approach enables them to do free-viewing experiments and seems to have been adopted by the research community for similar purposes [2, 1].

This free-viewing is unusual, as neural mechanisms are usually studied in strictly controlled experiments [2]. The deconvolution abilities of Unfold enable less stringent experiment setups, like matching a beat with a drum [27] or using a map to navigate a virtual city [4].

Another interesting avenue of study where Unfold has proven useful lies in analyzing the brain activity during reading activities [13, 18]. Some of these studies involved children with dyslexia [7, 17], in an attempt to better understand reading difficulties and help develop methods to improve reading ability.

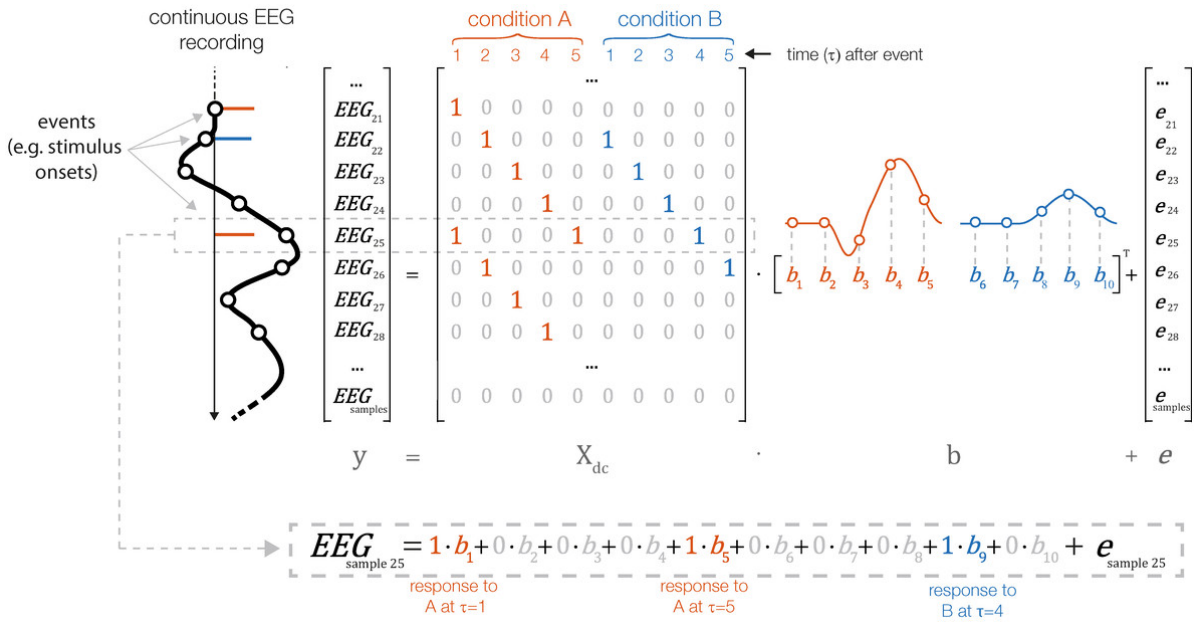

Figure 1.5: This graph shows the deconvolution performed by Unfold. The left part shows the continuous EEG signal, which is split into individual samples, labeled EEG21 through EEG28. During the actual deconvolution process, the matrix X in the center of the graph is created. Every time an event marker is encountered, like the stimulus onset or reaction time, the event is marked in the matrix with as seen in the graph (orange and blue 1s in the matrix). After the matrix is created for the entire EEG signal, the desired beta signals can be calculated for each sample as seen in the formula at the bottom. Reproduced from [12].

1.6 Issues with the Current RIDE Implementation

The original MATLAB RIDE implementation has been widely used in EEG research, but several issues hinder its maintainability, extensibility, and usability. Due to these limitations, further extending and maintaining the MATLAB RIDE implementation was deemed impractical.

The MATLAB version lacks automated unit testing, making it difficult to verify correctness and detect errors when modifying the algorithm. Any changes require extensive manual debugging, increasing the risk of introducing errors without realizing it.

Several aspects of the MATLAB implementation contribute to its poor maintainability:

- **Hardcoded Workflow:** The algorithm is implemented as a monolithic script with little modularity, making modifications complex and limiting extensibility.
- **Non-Descriptive Variable Names:** Variables such as `temp11`, `l1`, and `com_c` are frequently used without meaningful descriptions.
- **Lack of Inline Documentation:** The usage and entry point of the code is well documented with code comments and examples, but the inner workings of the algorithm are lacking in this aspect. Especially the deconvolution only contains sparse code comments and required extensive manual debugging to be understandable.
- **Use of `eval()`:** The implementation uses `eval()` to dynamically generate variable names, which makes debugging difficult and reduces code readability. This practice complicates maintenance, as it obscures how variables are created and referenced.

The MATLAB code does not appear to use any version control. There is no clear Git history, structured issue tracking, or continuous integration pipeline, making collaborative development and debugging more challenging. Without these tools, it is difficult to track changes over time or ensure that modifications do not introduce regressions. It would be possible to start using version control for future development, but the lack of an existing history is a notable downside.

1.7 RIDE Reimplementation

Both RIDE and Unfold show distinct advantages and limitations. A combination using parts from both projects could include RIDEs latency estimation abilities while maintaining Unfolds other benefits. For this reason, it was decided that a reimplementation of RIDE would be the best course of action.

With this project, there are now three different versions of the RIDE algorithm implemented:

- **MATLAB RIDE:** The original implementation in MATLAB by Ouyang et al. [22]
- **Classic RIDE:** A reimplementation of RIDE in Julia using the iterative decomposition method.
- **Unfold RIDE:** A modification of Classic RIDE using Unfold for the decomposition step and residue calculation.

A flowchart illustrating the differences between the different versions is provided in figure 2.1.

1.7.1 Integrating RIDE into Unfold

The new Classic and Unfold RIDE variants were integrated into the Unfold framework. This integration allows the easy use of RIDE on Unfold's standardized data structure, without the need for any additional data transformation steps. Additionally, integrating RIDE into Unfold also enables the use of other Unfold-related packages, such as UnfoldSim for simulating EEG data and UnfoldMakie for visualization. The addition to an actively maintained open-source project also improves long-term support and future compatibility, making RIDE more accessible and usable for future EEG analysis.

1.8 Choice of Implementation Language

The implementation language for this project was chosen based on multiple factors, including performance, maintainability, and long-term viability of relevant toolkits. These factors are particularly critical in EEG/ERP research, where large datasets, complex signal processing, and evolving analytical methodologies demand a language that ensures efficiency, reproducibility, and future-proofing of analytical tools. After evaluating different options, we selected Julia as the primary language for development.

1.8.1 Comparison: Julia vs. MATLAB

Two candidates were considered for the implementation language: MATLAB and Julia.

- **MATLAB:** The *Ride* algorithm is currently implemented in MATLAB, which initially made it an appealing choice for this project. Additionally, there is a MATLAB version of the *Unfold* toolbox available. However, this version is no longer actively developed, raising concerns about its long-term viability.
- **Julia:** The *Unfold* toolbox is also implemented in Julia, and unlike its MATLAB counterpart, the Julia version is actively maintained. This indicates that future developments and improvements in the toolbox will likely be centered around Julia, making it the more sustainable option for EEG/ERP research.

Given the stagnation of MATLAB-based *Unfold* and the increasing adoption of Julia in scientific computing, we opted for Julia to ensure long-term sustainability and compatibility. The following points were also considered as advantages of Julia compared to MATLAB:

- **Open Source:** Unlike MATLAB, which requires a paid license, Julia is completely open-source, making it more accessible for collaboration and reproducibility in the research community.
- **Performance:** Julia is designed for high-performance numerical computing and can often outperform MATLAB due to its Just-In-Time (JIT) compilation, allowing for faster execution of computationally intensive tasks.

- **Unit Testing and Software Engineering:** Julia has a more modern and flexible unit testing framework, which facilitates robust software development practices. Unlike MATLAB's built-in testing framework, which relies on separate scripts and lacks native test discovery, Julia's Test module allows inline, composable test cases with clear error reporting. Additionally, Julia's seamless integration with continuous integration (CI) tools makes automated testing workflows more efficient.
- **GitHub Integration:** Julia integrates seamlessly with GitHub, simplifying version control, package management, and collaborative development, which is particularly beneficial for research projects with multiple contributors.

Chapter 2

Methods/Process

The goal of the project was to translate the RIDE algorithm into the Julia language, then modify it to use the Unfold deconvolution and finally integrate it into the Unfold Julia toolbox. As a first step for this project, we decided to write a data generation algorithm to easily create a dataset that can be used to run and verify the RIDE algorithm.

2.1 Data Generation

Some dataset to run the algorithm was a necessity. The original MATLAB implementation does contain a real world dataset, but using this for the entire development process has several downsides. First of all, since this is a real world dataset, we don't actually know what underlying components are supposed to be contained within the data. This makes it difficult to determine whether the algorithm is actually working correctly. The only way to interpret the results would be a comparison to the MATLAB RIDE results. This assumes that the MATLAB RIDE results are actually accurate, which is an unnecessary compromise. With a generated dataset, we can calculate the expected output precisely and easily compare the gathered results to it. Secondly, it restricts us to testing with only one static scenario. Generating our own test data instead gives us the freedom to create multiple unique datasets with different challenges. The ability to run without any Noise has proven to be quite useful during various debugging sessions.

The data generation is largely based on the UnfoldSim.jl [23] toolbox. UnfoldSim already provides functionality to generate EEG data, but we were unable to find a method to fit our specific scenario. Mainly, we need to be able to define a sequence of components, each with a unique offset from a common stimulus. Creating a sequence of components is possibly through the use of a SequenceDesign, which was unreleased at the time of writing this paper. This SequenceDesign allowed us define multiple components with multiple offsets, but the offsets would always apply to the previous component instead of a common stimulus for the sequence. This issue was solved by overwriting the "UnfoldSim.simulate_onsets" function with our own implementation. We defined a default scenario for testing, which can be generated through a single function call. Some additional logic is included to also simulate each component individually without any variability in the onset and without any noise. These additional simulations are then used to calculate the underlying components, which in turn represent an optimal result for the RIDE algorithm.

Finally, a method to save the simulated data in the HDF5 format is included. This method also transforms the data into the format used by the MATLAB implementation

of the algorithm (Timesteps:Channels:Epochs). Importing the simulated data into MATLAB and running the MATLAB RIDE with this dataset was possible relatively early in the project and marked the first significant milestone.

We created a Julia Codebase to easily generate a simple dataset together with it's individual components and included functionality to export the data for use in MATLAB.

2.2 Translation from MATLAB into Julia

In line with the overall goals of the project, we next proceeded with a translation attempt of the MATLAB RIDE algorithm into Julia. The goal here was to create a functionally identical version, but with improved readability achieved through regular comments and refactoring. The new version should also be compatible with other Unfold Modules and the typical Unfold datasets.

Unfortunately, this translation attempt ended unsuccessful. This was due to two main reasons. First, the authors insufficient MATLAB and Julia skills. MATLAB uses a couple relatively unique concepts, like matrix notation, which can make it a bit more difficult to understand.

Unfortunately, this translation attempt ended unsuccessful. This was due to multiple reasons. First, the relatively poor code quality of the project, as illustrated by the following points:

- Usage of the algorithm is pretty well documented with a manual, examples and helpful comments, but the inner parts of the algorithm lack both code comments and other documentation
- Non descriptive Variable Names, like "temp" "temp0" "temp1" "temp11" "temp2" "tem" "tem0" are frequently used throughout the project, making it difficult to understand the data flow without running the code through the debugger.
- Long one line statements with high complexity.
- No automatic unit testing of individual methods.

Additionally, the code was modified to be used for micro saccade analysis, which was inserted into the main algorithm flow, instead of properly separated into other files or functions. This further complicates the overall structure of the code.

Another factor was the lack of MATLAB experiences on the side of the author. MATLAB introduces a couple foreign concepts compared to more common languages like java, like dot notation, matrix notation and ranges. These Differences were somewhat unexpected and further complicated the process.

The lack of MATLAB experience on side of the author combined with the poor code quality meant understanding even a small section of the code base consumed an excessive amount of time. This ultimately lead to the decision to abandon this course of action and focus on re-implementing the algorithm from scratch instead.

2.3 RIDE Classic Implementation in Julia

After the original MATLAB implementation proofed to be difficult to work with, we decided to re-implement the algorithm from scratch, using the MATLAB version as a

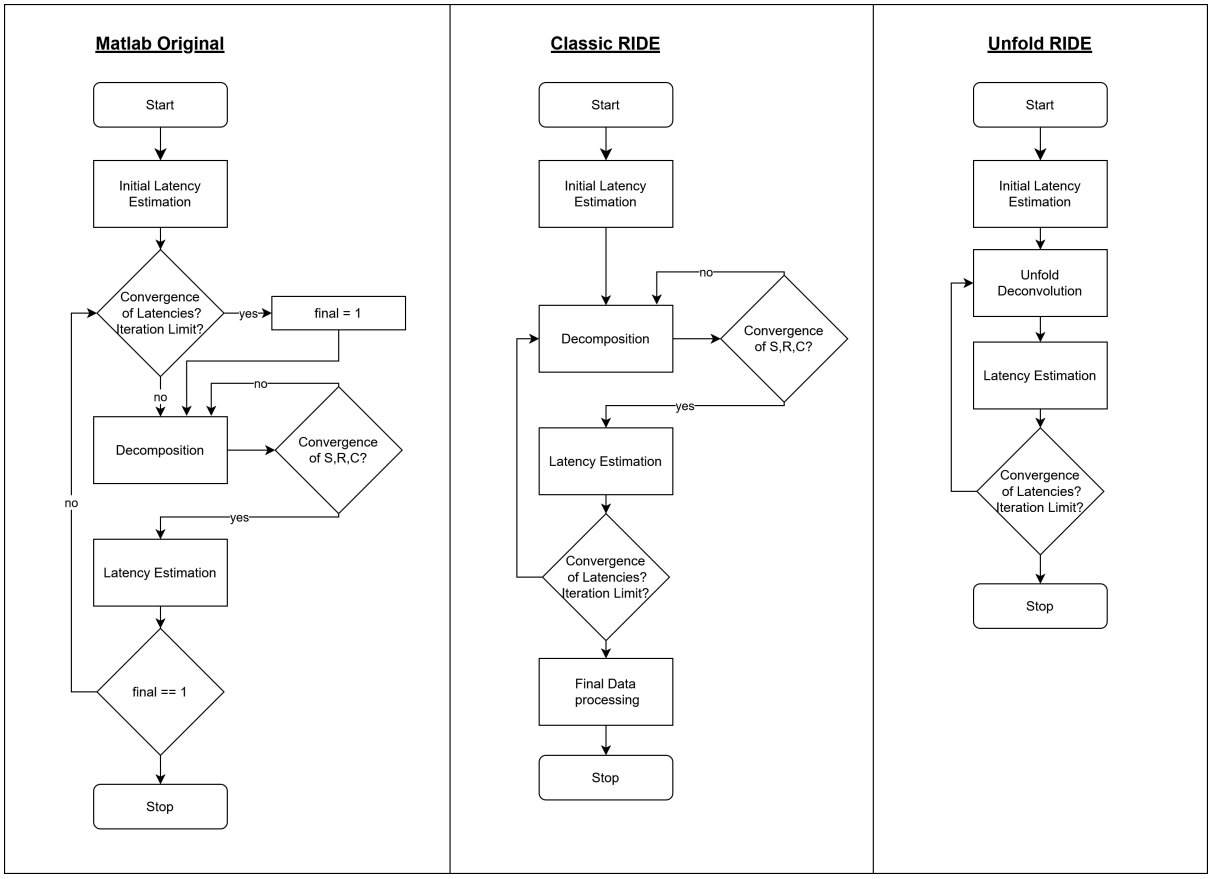

Figure 2.1: This flowchart shows the different algorithms. Noteable are the two loops present in MATLAB and Classic RIDE, one of which is removed in Unfold RIDE and replaced with the Unfold deconvolution.

reference. Additionally, there was already a previous implementation attempt by Prof. Ehinger, which was used as another point of reference. This previous implementation contained an unknown error, which we were unable to identify. Nevertheless, it proved to be a valuable tool for understanding the algorithm flow and how the Unfold deconvolution can be integrated into it.

RIDE in julia was implemented in two separate versions, the "Classic" and "Unfold" version. The "Classic" version uses the same deconvolution mechanism as described in the orginial paper and implemented in the MATLAB version. The "Unfold" version uses and Unfold model fit for the deconvolution and residue calculation. The differences between the different versions are also showcased in figure 2.1.

2.3.1 Heuristics and Data Processing

The MATLAB implementation of the algorithm includes several optimizations or heuristics to improve algorithm results. In an attempt to replicate the results of the original MATLAB version, these additional heuristics were also implemented in the Julia versions. These Heuristics can be enabled via the configuration input of the algorithm.

These heuristics consist of:

- Heuristic1: Only allow monoton latency changes. The C latencies should only move in one direction. When the latency change reverts, the latency is fixed at it's current

value for the remainder of the algorithm.

- Heuristic2: Randomize the latency when encountering a convex x-correlation curve. To check whether the x-correlation is convex, the curve is searched for peaks. If any peaks are found, the curve is not convex. If the curve is convex, the C latency is randomized using a standard deviation over the mean of all latencies from the previous iteration.

- Heuristic3: Consider competing peaks in the x-correlation of the latency estimation step. During the latency estimation, an x-correlation is calculated to determine where the C component fits best on the calculated residue. When there are multiple competing peaks in this x-correlation, the algorithm should choose the result closest to the previous latency instead of simply taking the biggest peak. By default, every peak within 10% of the maximum peak is considered as "competing". This cutoff can be customized in the configuration of the algorithm.

Chapter 3

Results

The three different versions of the algorithm have been executed on multiple datasets, results of which we will present in the following chapter. They were run on a simple simulated scenario, the real world dataset attached to the MATLAB implementation. Additionally, a performance benchmark has been conducted.

The associated code for the Classic and Unfold implementation can be found on `https://github.com/unfoldtoolbox/UnfoldRIDE.jl`.

3.1 Simulated Dataset

A representation of the simulated data can be seen in figure 1.4. The results of the different algorithm can be seen in figure 3.1.

3.1.1 Motivation for Using Simulated Data

Simulated datasets provide a controlled environment to evaluate the performance of different decomposition methods. Unlike real-world EEG data, simulated signals have a well-defined ground truth, making it possible to directly measure reconstruction accuracy. By using a simulation, we eliminate confounding factors such as physiological noise and experimental variability, allowing for a focused assessment of the algorithmic differences between MATLAB RIDE, Classic RIDE, and Unfold RIDE.

3.1.2 Simulation Setup

The simulated dataset was generated using `Unfold.Sim`, a module within the Unfold framework designed for EEG signal synthesis. The dataset consists of three ERP components (S, R, and C) as seen in the expected results of figure 3.1. Additionally, pink Gaussian noise was added using UnfoldSim[23] to simulate realistic signal conditions.

3.1.3 Visual Comparison of Results

Figure 3.1 shows the ERP decomposition results across methods.

The results from Unfold RIDE and Classic RIDE are nearly identical and closely match the expected ground truth across all three components. The amplitude, shape, and latency of the estimated signals seem accurate.

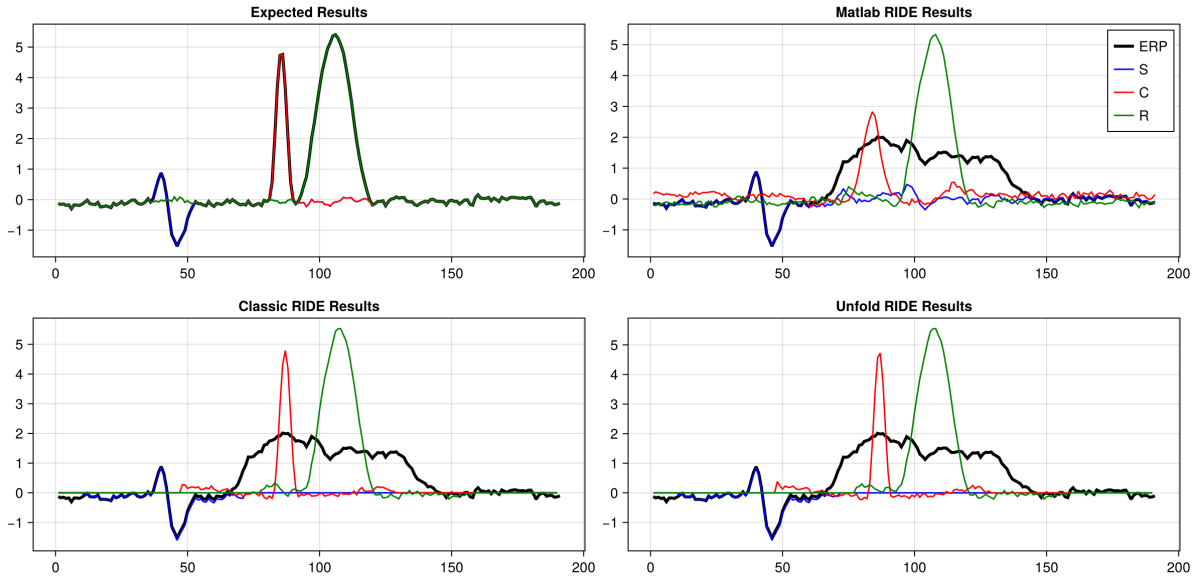

Figure 3.1: Comparison of ERP decomposition results across methods. The top-left plot shows the expected ground truth waveforms for components S (blue), C (red), and R (green). The remaining plots show the reconstructed ERPs from MATLAB RIDE (top-right), Classic RIDE (bottom-left), and Unfold RIDE (bottom-right).

In contrast, MATLAB RIDE underestimates the amplitude of the C component, which appears roughly 50% weaker than expected. Additionally, C seems slightly shifted to the left, indicating a potential latency estimation issue. The S and R components are accurately reconstructed in MATLAB RIDE, but the results contain more noise around the zero line, suggesting weaker decomposition or filtering compared to the Julia implementations.

3.1.4 Numerical Evaluation

To quantitatively assess the accuracy of each method, we computed the cross-correlation, and normalized root mean squared error (NRMSE) for each ERP component against the ground truth. Additionally, we computed the RMSE for the C component latencies to evaluate the estimation accuracy. The results are summarized in Tables 3.1 and 3.2.

Table 3.1: Numerical evaluation of ERP reconstruction accuracy for simulated data. Higher cross-correlation values indicate better waveform similarity, while lower NRMSE values indicate better amplitude accuracy.

Algorithm	**Cross-Correlation**			**NRMSE**		
	S	**R**	**C**	**S**	**R**	**C**
Classic RIDE	0.942	0.995	0.964	0.0238	0.0253	0.0479
Unfold RIDE	0.942	0.994	0.976	0.0242	0.0273	0.0354
MATLAB RIDE	0.912	0.993	0.840	0.0482	0.0265	0.0739

3.1.5 C Latency Estimation

To quantify the accuracy of C component latency estimation, we compared the detected latencies from each method to the known ground truth. Latencies were estimated using cross-correlation and computed for each individual epoch.

Table 3.2 presents the root mean squared error (RMSE) of the estimated C latencies across all epochs.

Table 3.2: RMSE for C latency estimation in simulated data. Lower values indicate better estimation accuracy.

Method	**C Latency RMSE (ms)**
Classic RIDE	0.78
Unfold RIDE	0.95
MATLAB RIDE	10.22

3.2 Real World Dataset

3.2.1 Dataset Description

The dataset used for this evaluation is provided with the MATLAB RIDE Toolbox and serves as an example dataset within the software. It consists of 65 EEG channels, each containing 550 data points per epoch, recorded over 173 epochs. Channel 44 was selected as the reference channel for evaluation, as it is the default channel used in the RIDE toolbox example.

3.2.2 Preprocessing and Data Handling

Unlike the MATLAB RIDE implementation, which processes epoched data directly, Unfold RIDE requires a continuous data format. Since the original continuous dataset was unavailable, the epochs were concatenated, and a padding of 500 zeroes was inserted between each epoch to maintain temporal separation while preventing artificial overlaps.

3.2.3 Visual Comparison of ERP Decomposition

Figure 3.2 presents the ERP decomposition results for Channel 44 across the three algorithms. The S and R components show similar shape in all three algorithms, while the C component contains more fluctuations in Classic and Unfold RIDE. The amplitude of the R component is about half in the MATLAB results compared to the other implementations. The S component of Classic and Unfold are offset downwards compared to the ERP and MATLAB results.

3.2.4 Latency Estimation Variability

To assess how consistently each method estimated C latencies across trials, figure 3.3 visualizes the detected latencies for Channel 44.

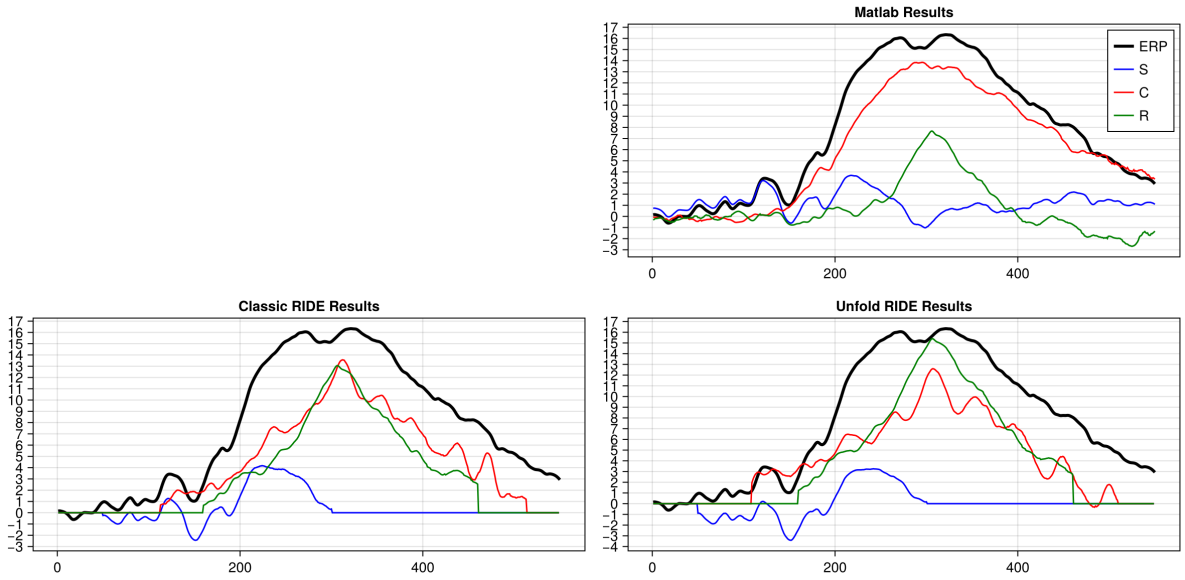

Figure 3.2: Comparison of ERP decomposition results for real EEG data (Channel 44). The top-right plot shows MATLAB RIDE results, while the bottom-left and bottom-right plots show Classic RIDE and Unfold RIDE, respectively.

The latency estimation results show noticeable differences in consistency between methods. Classic RIDE and Unfold RIDE exhibit high variability with frequent outliers, while MATLAB RIDE produces more tightly clustered latency estimates.

3.3 Performance Benchmark

3.3.1 Benchmarking Methodology

To evaluate the computational efficiency of each implementation, we measured execution time while running the simulated dataset on MATLAB RIDE, Classic RIDE, and Unfold RIDE. The benchmarking was conducted on a Lenovo laptop with a Ryzen 7 7840HS processor while connected to a power source. All tests were performed sequentially to ensure consistent system conditions.

Execution time was measured using BenchmarkTools.jl[3] in Julia with the `@benchmark` command, and the `timeit()` function in MATLAB. To mitigate initialization effects, the benchmark was run twice, with the first measurement discarded.

Due to implementation differences, namely the lack of convergence checks, Classic RIDE was tested in two variations. Once with the inner decomposition iterations limited to 10, to closely match the MATLAB execution, and once with 25 iterations as that is the default used during development. It was experimentally determined that the MATLAB execution runs with a mean of 10 inner iterations for this exact dataset.

3.3.2 Results

The execution times for each implementation are summarized in Table 3.3.

The results indicate that Classic RIDE with 10 iterations executes faster than MATLAB RIDE, while running 25 iterations significantly increases computation time. Unfold

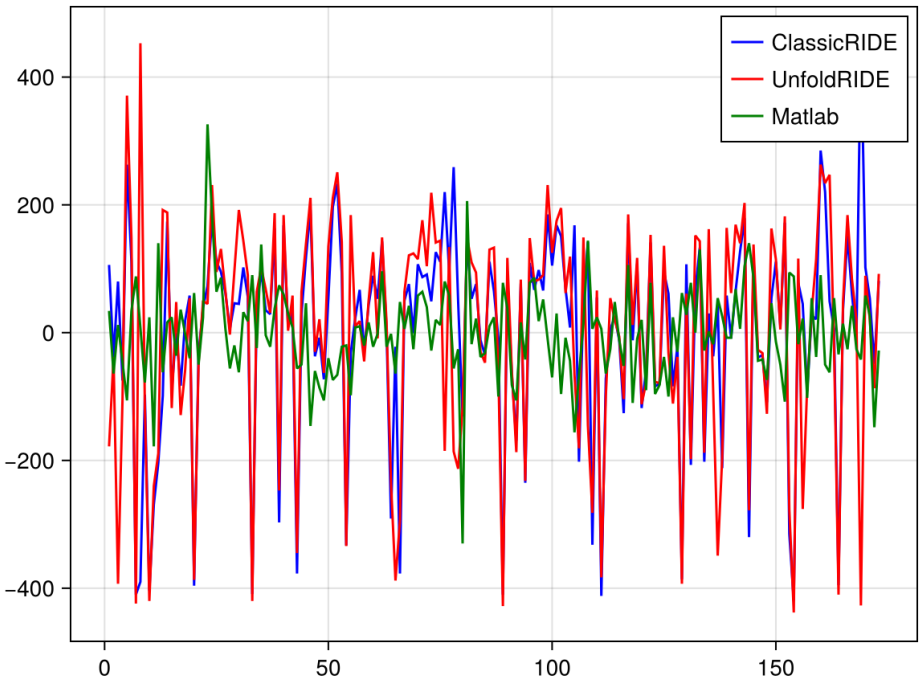

Figure 3.3: Estimated C latencies across epochs for Channel 44 of the real dataset. Classic RIDE and Unfold RIDE show considerable variability, with frequent outliers, while MATLAB RIDE estimates remain more clustered.

Table 3.3: Execution time for different implementations of RIDE.

Method	**Execution Time (ms)**
MATLAB RIDE	88.7
Classic RIDE (10 iterations)	71.27
Classic RIDE (25 iterations)	142.25
Unfold RIDE	85.15

RIDE performs comparably to MATLAB RIDE in terms of execution speed.

Chapter 4

Discussion

4.1 Interpretation of Results

This section provides an overview of the results from both the simulated and real-world datasets. The goal is to assess whether Classic RIDE, Unfold RIDE, and MATLAB RIDE produce similar outcomes and whether the new implementations perform differently from the MATLAB original.

4.1.1 Simulated Dataset Findings

The simulated dataset was designed to provide a clear ground truth for evaluation. Comparing the results of each algorithm to this known truth allows for a direct assessment of their accuracy.

Classic RIDE and Unfold RIDE produce nearly identical results, closely matching the expected ERP components in terms of both amplitude and latency. MATLAB RIDE, on the other hand, shows a noticeable reduction in the amplitude of the C component and a slight shift towards earlier latencies. This suggests that the iterative decomposition in MATLAB RIDE does not fully recover the expected signal in this specific case.

The numerical evaluations confirm these trends. Cross-correlation values indicate strong agreement between the reconstructed and expected ERP waveforms, but Classic RIDE and Unfold RIDE consistently achieve higher similarity scores than MATLAB RIDE. The normalized root mean squared error (NRMSE) also shows that MATLAB RIDE has larger deviations, especially for the C component. C latency RMSE values further reinforce these findings, showing that Classic and Unfold RIDE estimate latencies more accurately than MATLAB.

Since Classic and Unfold RIDE produce nearly identical results, it is likely that the differences observed in MATLAB RIDE stem from changes introduced in the reimplementation rather than the difference in decomposition strategies. One potential factor is the initial latency estimation method—MATLAB RIDE relies on Woody's method, whereas Classic and Unfold RIDE use a peak-picking approach. Another possible explanation lies in the cross-correlation process, where MATLAB RIDE may apply additional filtering or outlier rejection that is not present in the reimplemented versions. Finally, it is also possible that unknown implementation differences or unintended deviations from the original algorithm contribute to the observed discrepancies.

The simulated dataset used in this study was relatively simple, containing three non-overlapping ERP components with moderate noise. While this setup was useful for

validating the algorithm's correctness, it does not fully test the limits of each method. More complex simulation scenarios—such as overlapping components, highly variable latencies, or more realistic noise distributions—could highlight differences that are not visible in the current setup. Future work should explore these cases to determine whether Unfold RIDE and Classic RIDE maintain their performance advantages across a broader range of conditions.

4.1.2 Real-World Dataset Findings

The results from the real-world dataset reveal several differences between the three methods. While all three algorithms successfully extract ERP components, there are notable discrepancies in component amplitude, shape, and alignment.

The **S component** appears to be in a similar position across all algorithms and maintains a comparable shape. However, in Classic and Unfold RIDE, the S component diverges downward from the raw ERP, whereas in MATLAB RIDE, it closely follows the raw ERP signal. This suggests that the MATLAB implementation retains more similarity to the original ERP waveform in this component.

The **R component** is also positioned similarly across all algorithms and exhibits a consistent shape. However, its amplitude is significantly higher in Classic and Unfold RIDE—nearly twice as large as in MATLAB RIDE. This could indicate differences in amplitude scaling between implementations or the way each method handles residual signal contributions.

The **C component** shows the most noticeable differences. While the general shape remains somewhat comparable across algorithms, MATLAB RIDE produces a much smoother C component, whereas Classic and Unfold RIDE exhibit a distinct peak. One possible explanation is that MATLAB's smoother C component results from lower C latency variability, leading to a more stable waveform. Additionally, MATLAB's C component appears to closely match the raw ERP signal, further suggesting that the reduced variability in C latencies influences the final decomposition.

Overall, the real-world dataset results confirm that Classic and Unfold RIDE behave similarly to each other, while MATLAB RIDE systematically differs in certain aspects—particularly in C component smoothness and R component amplitude. These differences could be due to variations in preprocessing, residual handling, or constraints applied during decomposition. Further analysis would be required to determine whether these discrepancies affect the interpretability or accuracy of the extracted ERP components.

4.2 Limitations

While the Unfold RIDE implementation successfully reproduces key aspects of the original MATLAB RIDE, certain differences and limitations exist. This section outlines the constraints in multi-component decomposition, discrepancies between the MATLAB and Classic implementations, and the absence of multi-channel support.

4.2.1 Multi-Component Decomposition

The MATLAB RIDE implementation allows for multiple R components and multiple C components in the decomposition process, as long as they do not share identical estima-

tion windows. This enables configurations such as S/C1/C2/R1/R2 decomposition. The current reimplementation in Classic and Unfold RIDE does not support multiple C or multiple R components and is restricted to a single S, C, and R component. Extending the model to allow for additional components would require modifying the current algorithm structure.

4.2.2 Differences Between MATLAB and Classic Implementation

Classic RIDE was originally intended as a direct replication of MATLAB RIDE but failed to produce identical results. The differences between the implementations arise from several factors.

First, the overall algorithm workflow differs in the Julia implementation, as seen in 2.1. Some steps were reorganized, and certain features, such as MATLAB's convergence checks, were not implemented. As a result, Classic RIDE always runs a fixed number of iterations, potentially affecting the final decomposition.

Second, the filtering method used in Classic RIDE differs from the one in MATLAB RIDE. This could contribute to discrepancies in the extracted ERP components.

Additionally, minor implementation differences could have led to further divergence between MATLAB and Classic RIDE. Identifying the exact causes would require a step-by-step comparison of intermediate results between both implementations. This process was omitted due to time constraints, as the poor MATLAB code quality made debugging difficult. Furthermore, such a comparison would likely have been inconclusive without first aligning known differences, such as filtering and iteration behavior.

4.2.3 Multi-Channel Mode

The Unfold framework is designed to handle multi-channel EEG data, but this feature was not tested in the current implementation of Unfold RIDE. While implementing multi-channel support should be straightforward, validation would be required to confirm that the algorithm scales effectively.

4.3 Future Work

While the current implementation successfully integrates RIDE into Unfold, there are several areas for improvement. This section outlines potential feature expansions, numerical validation strategies, and further testing to strengthen the algorithm.

4.3.1 Addressing Identified Limitations

Several limitations of the current implementation were identified in Section 4.2. Future work should address these issues to further improve the algorithm:

- **Multi-component decomposition:** The current implementation does not support multiple C and R components as in MATLAB RIDE. Expanding this functionality would allow for greater configurability.

- **Convergence checking:** Classic RIDE currently runs a fixed number of iterations, unlike MATLAB RIDE, which stops dynamically. Implementing a convergence criterion would improve performance.
- **Step-by-step comparison with MATLAB:** Differences in filtering, iteration behavior, and workflow prevent a direct match between MATLAB and Classic RIDE. Future work should replicate MATLAB features as optional settings to facilitate debugging.
- **Multi-channel support:** Unfold RIDE has the potential to process multi-channel EEG data, but this feature was not implemented. Future work should enable and validate this functionality.

4.3.2 Expanding Numerical Validation

Another important step is testing the algorithm on more complex simulated datasets. The current simulations are relatively simple, consisting of non-overlapping ERP components with moderate noise. Future evaluations should include overlapping ERP components, highly variable latencies, and varying noise levels to better assess the robustness of the decomposition process.

Automated testing can further improve stability and reproducibility. The UnfoldSim.jl toolbox provides the capability to generate a large number of structured EEG simulations. Expanding the test suite with randomized simulations would allow for a broader evaluation of the algorithm under different conditions. Additionally, establishing an automated benchmark based on expected outputs could help track future improvements in accuracy and performance.

4.3.3 Testing on Additional Real-World Datasets

Simulated datasets provide a controlled evaluation but may fail to capture the complexity of real EEG recordings. Testing on additional real-world datasets could help identify issues arising from chaotic neural activity or unexpected artifacts, which are not well represented in simulations.

Another promising application is analyzing datasets where variable-latency components are expected, such as tasks involving cognitive processing delays. Evaluating the algorithm's ability to estimate these latencies in real EEG recordings would provide insight into its real-world effectiveness.

4.4 Conclusion

This work aimed to integrate RIDE[20, 21] into Unfold[12], transitioning from MATLAB's iterative decomposition to a regression-based deconvolution approach. To achieve this, two implementations were developed: Classic RIDE, which attempted to replicate MATLAB's behavior, and Unfold RIDE, which leverages Unfold's modeling framework. The algorithms were evaluated on both simulated and real-world datasets to compare their performance against the original MATLAB implementation.

The results indicate that while Classic and Unfold RIDE behave similarly to each other, they diverge from MATLAB RIDE in some aspects. The reasons for these discrepancies require further investigation.

By integrating RIDE into Unfold, this project combines the strengths of both approaches. RIDE provides a framework for latency estimation in EEG decomposition, while Unfold enables regression-based deconvolution, covariate modeling, and a more structured software environment. This combination allows for greater flexibility in handling variable-latency components while preserving the benefits of the original RIDE approach.

The implementation, while functional, is still lacking in some aspects and does not fully replicate MATLAB RIDE. Further debugging is necessary to determine whether observed differences are due to intentional reimplementation choices or unintentional deviations. Beyond this, additional automated testing would improve the algorithm's reliability. Using structured simulations and randomized test cases, the performance of the algorithm could be systematically evaluated. A standardized benchmarking approach could also prove valuable in tracking future improvements.

This integration makes RIDE more accessible by providing a modern, open-source implementation within the Unfold ecosystem. Additionally, it enhances Unfold by introducing a method for handling variable-latency ERP components, expanding its applicability in EEG research. While further refinement is required, this work lays a foundation for future developments in ERP decomposition.

Chapter 5

German summary

Ereigniskorrelierte Potentiale (ERPs) werden häufig in der EEG-Forschung verwendet, um die neuronale Reaktion auf spezifische Stimuli zu untersuchen. Traditionelle ERP-Mittelungsverfahren sind anfällig für Versuch-zu-Versuch-Variabilität in den Latenzen, was zu Verzerrungen in den extrahierten Komponenten führen kann. Residual Iteration Decomposition Estimate (RIDE) ist eine etablierte Methode, die diese Variabilität korrigiert, indem sie Komponenten basierend auf ihren Latenzverteilungen trennt. Trotz ihrer Wirksamkeit mangelt es der ursprünglichen MATLAB-Implementierung an Flexibilität und Integration in moderne EEG-Analysewerkzeuge.

Diese Arbeit stellt eine Julia-basierte Implementierung von RIDE sowie deren Integration in Unfold, eine auf Regressionsmodellen basierende Dekonvolutions-Toolbox für die EEG-Analyse, vor. Es wurden zwei Versionen entwickelt: Classic RIDE, das das Verhalten von MATLAB RIDE repliziert, und Unfold RIDE, das die iterative Dekomposition durch eine regressionsbasierte Dekonvolution ersetzt. Beide Implementierungen wurden mit simulierten Datensätzen mit bekannter Ground Truth sowie mit einem realen EEG-Datensatz getestet, um ihre Genauigkeit zu bewerten und sie mit MATLAB RIDE zu vergleichen.

Die Ergebnisse zeigen, dass Classic und Unfold RIDE ERP-Komponenten erfolgreich rekonstruieren und in beiden Datensätzen sinnvolle Ergebnisse liefern. Allerdings bestehen Unterschiede im Vergleich zu MATLAB RIDE, die vermutlich auf fehlende Funktionen sowie Variationen in Filterung, Latenzschätzung und Iterationsverhalten zurückzuführen sind. Trotz dieser Abweichungen bestätigen die Ergebnisse, dass Unfolds regressionsbasierter Ansatz eine praktikable Alternative zur iterativen Dekomposition darstellt und potenzielle Vorteile wie eine höhere Modellierungsflexibilität bietet.

Zukünftige Forschung sollte sich auf die Implementierung fehlender Funktionen und die Erreichung der Äquivalenz mit MATLAB RIDE konzentrieren. Darüber hinaus könnte ein randomisiertes Testverfahren mit UnfoldSim eine systematische Validierung ermöglichen und zur Optimierung des Algorithmus beitragen. Diese Arbeit legt die Grundlage für zukünftige Verbesserungen in der EEG-Dekonvolution, indem sie RIDE's Latenzschätzung mit Unfolds flexiblem Regressionsansatz kombiniert.

Die implementierten Algorithmen sind als Open-Source-Projekt auf GitHub verfügbar: `https://github.com/unfoldtoolbox/UnfoldRIDE.jl`

Bibliography

[1] Andrey R. Nikolaev et al. "Episodic memory formation in unrestricted viewing". In: 266 (Feb. 1, 2023). MAG ID: 4312083095, pp. 119821–119821. DOI: 10.1016/j.neuroimage.2022.119821.

[2] Anna L Gert et al. "WildLab: A naturalistic free viewing experiment reveals previously unknown EEG signatures of face processing". In: *European Journal of Neuroscience* (Sept. 16, 2022). MAG ID: 4296164644. DOI: 10.1111/ejn.15824.

[3] *BenchmarkTools*. URL: https://juliaci.github.io/BenchmarkTools.jl/stable/.

[4] Bingjie Cheng et al. "Using spontaneous eye blink-related brain activity to investigate cognitive load during mobile map-assisted navigation". In: 17 (Feb. 14, 2023). MAG ID: 4320723898. DOI: 10.3389/fnins.2023.1024583.

[5] Annet Bluschke et al. "Neuronal Intra-Individual Variability Masks Response Selection Differences between ADHD Subtypes—A Need to Change Perspectives". In: *Frontiers in Human Neuroscience* 11 (June 28, 2017). Publisher: Frontiers. ISSN: 1662-5161. DOI: 10.3389/fnhum.2017.00329. URL: https://www.frontiersin.org/journals/human-neuroscience/articles/10.3389/fnhum.2017.00329/full (visited on 08/27/2024).

[6] Chiara Botrugno. "Enhancement of Readiness Potentials' pre-processing chain in a Brain-Computer Interface for nonresponsive patients". laurea. Politecnico di Torino, July 27, 2022. 103 pp. URL: https://webthesis.biblio.polito.it/23765/ (visited on 03/11/2025).

[7] Catherine Manning et al. "Visual Motion and Decision-Making in Dyslexia: Reduced Accumulation of Sensory Evidence and Related Neural Dynamics." In: *The Journal of Neuroscience* 42.1 (Jan. 5, 2022). MAG ID: 4226134057, pp. 121–134. DOI: 10.1523/jneurosci.1232-21.2021.

[8] Witold X. Chmielewski, Moritz Mückschel, and Christian Beste. "Response selection codes in neurophysiological data predict conjoint effects of controlled and automatic processes during response inhibition". In: *Human Brain Mapping* 39.4 (2018). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/hbm.23974, pp. 1839–1849. ISSN: 1097-0193. DOI: 10.1002/hbm.23974. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/hbm.23974 (visited on 08/27/2024).

[9] Donna Coch, Jennifer Bares, and Allison Landers. "ERPs and morphological processing: the N400 and semantic composition". In: *Cognitive, Affective, & Behavioral Neuroscience* 13.2 (June 1, 2013), pp. 355–370. ISSN: 1531-135X. DOI: 10.3758/s13415-012-0145-3. URL: https://doi.org/10.3758/s13415-012-0145-3 (visited on 03/11/2025).

[10] Olaf Dimigen and Benedikt V. Ehinger. *Analyzing combined eye-tracking/EEG experiments with (non)linear deconvolution models*. Pages: 735530 Section: New Results. Mar. 26, 2020. DOI: 10.1101/735530. URL: https://www.biorxiv.org/content/10.1101/735530v2 (visited on 08/31/2024).

[11] Olaf Dimigen and Benedikt V. Ehinger. "Regression-based analysis of combined EEG and eye-tracking data: Theory and applications". In: *Journal of Vision*. 1st ser. 21.1 (Jan. 7, 2021), p. 3. ISSN: 1534-7362. DOI: 10.1167/jov.21.1.3. URL: https://doi.org/10.1167/jov.21.1.3 (visited on 08/27/2024).

[12] Benedikt V. Ehinger and Olaf Dimigen. "Unfold: an integrated toolbox for overlap correction, non-linear modeling, and regression-based EEG analysis". In: *PeerJ* 7 (Oct. 24, 2019). Publisher: PeerJ Inc., e7838. ISSN: 2167-8359. DOI: 10.7717/peerj.7838. URL: https://peerj.com/articles/7838 (visited on 08/27/2024).

[13] Joshua Snell et al. "Parallel word reading revealed by fixation-related brain potentials". In: 162 (May 1, 2023). MAG ID: 4322124135, pp. 1–11. DOI: 10.1016/j.cortex.2023.02.004.

[14] Maximilian Kleimaker et al. "Increased perception-action binding in Tourette syndrome". In: *Brain* 143.6 (June 1, 2020), pp. 1934–1945. ISSN: 0006-8950. DOI: 10.1093/brain/awaa111. URL: https://doi.org/10.1093/brain/awaa111 (visited on 08/27/2024).

[15] Marta Kutas, Gregory McCarthy, and Emanuel Donchin. "Augmenting Mental Chronometry: The P300 as a Measure of Stimulus Evaluation Time". In: *Science* 197.4305 (Aug. 19, 1977). Publisher: American Association for the Advancement of Science, pp. 792–795. DOI: 10.1126/science.887923. URL: https://www.science.org/doi/10.1126/science.887923 (visited on 03/04/2025).

[16] Moritz Mückschel et al. "The norepinephrine system shows information-content specific properties during cognitive control – Evidence from EEG and pupillary responses". In: *NeuroImage* 149 (Apr. 1, 2017), pp. 44–52. ISSN: 1053-8119. DOI: 10.1016/j.neuroimage.2017.01.036. URL: https://www.sciencedirect.com/science/article/pii/S1053811917300435 (visited on 08/27/2024).

[17] Najla Azaiez et al. "Brain Source Correlates of Speech Perception and Reading Processes in Children With and Without Reading Difficulties". In: *Frontiers in Neuroscience* 16 (July 19, 2022). MAG ID: 4285798973. DOI: 10.3389/fnins.2022.921977.

[18] Nora Hollenstein et al. "The ZuCo Benchmark on Cross-Subject Reading Task Classification with EEG and Eye-Tracking Data". In: (Mar. 8, 2022). MAG ID: 4221052551. DOI: 10.1101/2022.03.08.483414.

[19] Guang Ouyang, Werner Sommer, and Changsong Zhou. "A toolbox for residue iteration decomposition (RIDE)—A method for the decomposition, reconstruction, and single trial analysis of event related potentials". In: *Journal of Neuroscience Methods*. Cutting-edge EEG Methods 250 (July 30, 2015), pp. 7–21. ISSN: 0165-0270. DOI: 10.1016/j.jneumeth.2014.10.009. URL: https://www.sciencedirect.com/science/article/pii/S0165027014003690 (visited on 08/27/2024).

[20] Guang Ouyang et al. "Residue iteration decomposition (RIDE): A new method to separate ERP components on the basis of latency variability in single trials". In: *Psychophysiology* 48.12 (2011). eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1469-8986.2011.01269.x, pp. 1631–1647. ISSN: 1469-8986. DOI: 10.1111/j.1469-8986.2011.01269.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1469-8986.2011.01269.x (visited on 08/27/2024).

[21] Guang Ouyang et al. "Residue iteration decomposition (RIDE): A new method to separate ERP components on the basis of latency variability in single trials". In: *Psychophysiology* 48.12 (2011). eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1469-8986.2011.01269.x, pp. 1631–1647. ISSN: 1469-8986. DOI: 10.1111/j.1469-8986.2011.01269.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1469-8986.2011.01269.x (visited on 08/27/2024).

[22] *RIDE matlab toolbox*. URL: https://cns.hkbu.edu.hk/RIDE_files/Page308.htm (visited on 08/27/2024).

[23] Judith Schepers et al. *UnfoldSim.jl*. Version v0.3.2. Feb. 27, 2024. DOI: 10.5281/zenodo.10714844. URL: https://zenodo.org/records/10714844 (visited on 08/28/2024).

[24] Adam Takacs et al. "Connecting EEG signal decomposition and response selection processes using the theory of event coding framework". In: *Human Brain Mapping* 41.10 (2020). eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/hbm.24983, pp. 2862–2877. ISSN: 1097-0193. DOI: 10.1002/hbm.24983. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/hbm.24983 (visited on 08/27/2024).

[25] Sam Verschooren et al. "Early reduction of sensory processing within the visual cortex when switching from internal to external attention". In: *Biological Psychology* 163 (July 1, 2021), p. 108119. ISSN: 0301-0511. DOI: 10.1016/j.biopsycho.2021.108119. URL: https://www.sciencedirect.com/science/article/pii/S0301051121001125 (visited on 03/11/2025).

[26] Nicole Wolff, Moritz Mückschel, and Christian Beste. "Neural mechanisms and functional neuroanatomical networks during memory and cue-based task switching as revealed by residue iteration decomposition (RIDE) based source localization". In: *Brain Structure and Function* 222.8 (Nov. 1, 2017), pp. 3819–3831. ISSN: 1863-2661. DOI: 10.1007/s00429-017-1437-8. URL: https://doi.org/10.1007/s00429-017-1437-8 (visited on 03/11/2025).

[27] Yan Yan, Laurence T. Hunt, and Cameron D. Hassall. "Reward positivity biases interval production in a continuous timing task". In: (July 7, 2023). MAG ID: 4383499942. DOI: 10.1101/2023.07.06.548049.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Druck-Exemplaren überein.

Datum und Unterschrift:

Declaration

I hereby declare that the work presented in this thesis is entirely my own. I did not use any other sources and references that the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted hard copies.

Date and Signature: