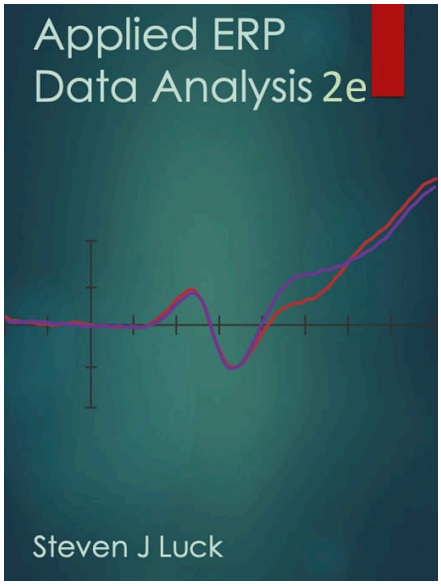
APPLIED EVENT-RELATED POTENTIAL DATA ANALYSIS 2E



Steven J Luck
University of California, Davis





Applied Event-Related Potential Data Analysis, Second Edition

Steven J Luck

University of California, Davis

This text is disseminated via the Open Education Resource (OER) LibreTexts Project (https://LibreTexts.org) and like the thousands of other texts available within this powerful platform, it is freely available for reading, printing, and "consuming."

The LibreTexts mission is to bring together students, faculty, and scholars in a collaborative effort to provide an accessible, and comprehensive platform that empowers our community to develop, curate, adapt, and adopt openly licensed resources and technologies; through these efforts we can reduce the financial burden borne from traditional educational resource costs, ensuring education is more accessible for students and communities worldwide.

Most, but not all, pages in the library have licenses that may allow individuals to make changes, save, and print this book. Carefully consult the applicable license(s) before pursuing such effects. Instructors can adopt existing LibreTexts texts or Remix them to quickly build course-specific resources to meet the needs of their students. Unlike traditional textbooks, LibreTexts' web based origins allow powerful integration of advanced features and new technologies to support learning.



LibreTexts is the adaptable, user-friendly non-profit open education resource platform that educators trust for creating, customizing, and sharing accessible, interactive textbooks, adaptive homework, and ancillary materials. We collaborate with individuals and organizations to champion open education initiatives, support institutional publishing programs, drive curriculum development projects, and more.

The LibreTexts libraries are Powered by NICE CXone Expert and were supported by the Department of Education Open Textbook Pilot Project, the California Education Learning Lab, the UC Davis Office of the Provost, the UC Davis Library, the California State University Affordable Learning Solutions Program, and Merlot. This material is based upon work supported by the National Science Foundation under Grant No. 1246120, 1525057, and 1413739.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation nor the US Department of Education.

Have questions or comments? For information about adoptions or adaptions contact info@LibreTexts.org or visit our main website at https://LibreTexts.org.

The LibreTexts name, logo, and book covers are trademarked by LibreTexts, Inc. (a 501(c)(3) not-for-profit organization) and protected against unauthorized use.

This text was compiled on 10/01/2026

TABLE OF CONTENTS

Licensing

Hardware and Software Requirements

Preface

Acknowledgments

How to Cite This Book

1: First Steps

1.1: Getting Started

1.2: Installing the Software and Downloading the Data

1.3: Exercise- Loading and Viewing an EEG File

1.4: Exercise - Viewing EEG Spectra

1.5: Exercise- Loading and Plotting ERPs

1.6: Understanding the Matlab Path

1.7: Key Takeaways and References

2: Processing the Data from One Participant in the ERP CORE N400 Experiment

2.1: Data For This Chapter

2.2: Design of the N400 Experiment

2.3: Exercise - Looking at the EEG and the Event Codes

2.4: Exercise- Filtering Out Low-Frequency Drifts from the EEG

2.5: Exercise- Creating an EventList

2.6: Exercise- Assigning Events to Bins with BINLISTER

2.7: Exercise- Epoching and Baseline Correction

2.8: Exercise- Artifact Detection

2.9: Exercise- Averaged ERPs

2.10: Exercise- Data Quality

2.11: Review of Processing Steps

2.12: A Simple Matlab Script

2.13: Key Takeaways and References

3: Processing Multiple Participants in the ERP CORE N400 Experiment

3.1: Data for This Chapter

3.2: Exercise- Organizing and Loading the Data from 10 Participants

3.3: Exercise- Examining the Single-Participant ERPsets

3.4: Exercise- “Bad” Data

3.5: Exercise- Making a Grand Average

3.6: Exercise- Low-Pass Filtering

3.7: Exercise - Scoring N400 Amplitude

3.8: Exercise- Simple Statistical Analysis of N400 Data

3.9: Exercise- A More Complex Analysis
3.10: Exercise- ERP Channel Operations
3.11: Exercise- ERP Bin Operations
3.12: Review of Processing Steps
3.13: Matlab Scripts For This Chapter
3.14: Key Takeaways and References

4: Filtering the EEG and ERPs

4.1: Data for this Chapter
4.2: Classes of Filters
4.3: Exercise- Assessing the Frequency Content of the Noise
4.4: Exercise- Filtering the 60 Hz Noise
4.5: Practical Advice About Filtering
4.6: Exercise- Filtering as a Linear Operation
4.7: Exercise- The Impulse Response Function
4.8: Exercise- Filtering a Series of Impulses (and a Sneaky Introduction to Convolution)
4.9: Predicting Waveform Distortion from the Impulse Response Function
4.10: Exercise- High-Pass Filtering
4.11: Background- How Should We Define the Optimal Filter?
4.12: Background- Quantifying Signal, Noise, and Waveform Distortion
4.13: Exercise- Creating artificial waveforms
4.14: Exercise- Quantifying Waveform Distortion in the Artificial Waveforms
4.15: Exercise- Quantifying Data Quality
4.16: Selecting the Optimal Filter
4.17: Matlab Scripts for this Chapter
4.18: Key Takeaways and References

5: Referencing and Other Channel Operations

5.1: Data for This Chapter
5.2: Background- Understanding Active, Reference, and Ground Electrodes
5.3: Exercise- Working with the Artificial Data
5.4: Exercise- Average Mastoids as the Reference
5.5: Exercise- Re-Referencing the N400 ERP CORE Data
5.6: Exercise- The Average Reference
5.7: What is the Best Reference Site?
5.8: Exercise- Current Density
5.9: Exercise- Global Field Power
5.10: Exercise- Referencing the EEG Data from the ERP CORE N400 Experiment
5.11: Exercise- Other Common Re-Referencing Scenarios
5.12: Matlab Script For This Chapter
5.13: Key Takeaways and References

6: Assigning Events to Bins, Averaging, Baseline Correction, and Assessing Data Quality

6.1: Data for This Chapter
6.2: Design of the ERP CORE Visual Oddball P3b Experiment
6.3: The Event Code Scheme
6.4: Overview of Bin Descriptor Files
6.5: Exercise - A Basic Assignment of Events to Bins
6.6: Exercise - Looking at the Averaged ERPs
6.7: Exercise - The Signal-to-Noise Ratio
6.8: Exercise - Response-Locked Averaging
6.9: Exercise - Comparing Correct and Error Trials to See the Error-Related Negativity
6.10: Exercise - Sequential Analysis of the P3b
6.11: Exercise - Combining Bins
6.12: Exercise - Overlap
6.13: Matlab Script For This Chapter
6.14: Key Takeaways and References

7: Inspecting the EEG and Interpolating Bad Channels

7.1: Data for This Chapter
7.2: Design of the Mismatch Negativity (MMN) Experiment
7.3: Video Demonstration- Performing an Initial Inspection of a Participant's EEG
7.4: Background- Interpolating Bad Channels
7.5: Exercise - Interpolating Bad Channels
7.6: Matlab Script For This Chapter
7.7: Key Takeaways and References

8: Artifact Detection and Rejection

8.1: Data for This Chapter
8.2: Overview
8.3: Background- Why Do We Reject Artifacts?
8.4: Background- The General Approach
8.5: Exercise- Simple Blink Detection
8.6: Exercise- Adjusting the Threshold
8.7: Exercise- Manually Marking and Unmarking Artifacts
8.8: An Iterative Approach to Setting Parameters
8.9: Exercise- Effects of Artifact Rejection on Data Quality
8.10: Exercise- Better Blink Detection
8.11: Exercise- Commonly Recorded Artifactual Potentials (C.R.A.P.)
8.12: Exercise- Detecting Eye Movements
8.13: Using Artifact Detection to Avoid Changes to Visual Inputs
8.14: The ERP CORE N2pc Experiment
8.15: Exercise- Visualizing the Eye Movements
8.16: Exercise- Using the Averaged HEOG to Visualize Consistent Eye Movements

8.17: Exercise- A Two-Stage Strategy for Eliminating Small But Consistent Eye Movements
8.18: Matlab Script For This Chapter
8.19: Key Takeaways and References

9: Artifact Correction with Independent Component Analysis

9.1: Data for this Chapter
9.2: Exercise- A First Pass at ICA-Based Blink Correction
9.3: Background- A Quick Conceptual Overview of ICA
9.4: Exercise- Making ICA Work Better
9.5: Exercise- Transferring the Weights, Identifying the ICs, Correcting the EEG, and Interpolating
9.6: Exercise- Deleting C.R.A.P. Prior to ICA
9.7: General Recommendations for ICA-Based Artifact Correction
9.8: Background- Evaluating the Effectiveness of Artifact Correction+Rejection
9.9: Exercise- Determining Whether Blinking is a Confound
9.10: Exercise- Was Blink Correction Effective?
9.11: Exercise- Does Artifact Correction+Rejection Improve Data Quality?
9.12: Matlab Scripts For This Chapter
9.13: Key Takeaways and References

10: Scoring and Statistical Analysis of ERP Amplitudes and Latencies

10.1: Data for This Chapter
10.2: Design of the Flankers Experiment
10.3: Exercise- Examining the Grand Averages
10.4: Exercise- A First Pass at Scoring and Statistical Analysis
10.5: Exercise- Simplifying the Statistical Analysis
10.6: Exercise- Peak Amplitude
10.7: Exercise- Peak Latency
10.8: Exercise- Fractional Area Latency
10.9: Exercise- Quantifying Onset Latency
10.10: Exercise- Collapsing Across Channels and Correlating Latencies with Response Times
10.11: Matlab Scripts For This Chapter
10.12: Key Takeaways and References

11: EEGLAB and ERPLAB Scripting

11.1: Data for This Chapter
11.2: Expected Background Knowledge
11.3: Bugs as an Opportunity for Growth
11.4: Design of the N170 Experiment
11.5: Exercise- The Matlab Command Line and the EEG Variable
11.6: Exercise- The ALLEEG Variable and Refreshing the ERPLAB Studio GUI
11.7: Exercise- EEG.history and eegh
11.8: Exercise- From the Command Line to a Script
11.9: Exercise- Using a Variable for the Path

11.10: Exercise- Loops
11.11: Exercise- Looping Through Data from Multiple Participants
11.12: Rapid Cycling Between Coding and Testing
11.13: Exercise- Using a Script to Reference EEGsets
11.14: Exercise- Improving the Script and Adding 3D Channel Locations
11.15: Exercise- Preprocessing the EEG and Using a Spreadsheet to Store Subject-Specific Information
11.16: Exercise- Building an Entire EEG Processing Pipeline
11.17: Exercise- Averaging, Quantifying Data Quality, and Plotting
11.18: Exercise- Scoring Amplitudes and Latencies and Performing Statistical Analyses
11.19: Key Takeaways and References

12: Multivariate Pattern Classification (Decoding)

12.1: Data for This Chapter
12.2: Example- Decoding Face Identity and Emotional Expression
12.3: The Essence of Decoding
12.4: Exercise- Binary Decoding of ID 1 versus ID 2
12.5: Exercise- Decoding Multiple BESTsets and Making a Grand Average
12.6: Exercise- 4-way Decoding of Face Identity and Emotional Expression
12.7: Exercise- Exporting the Data for Statistical Analysis
12.8: Exercise- Using Decoding in Place of Conventional Mean Amplitude Analyses
12.9: Exercise- Temporal Generalization
12.10: Demonstration- Decoding in the Frequency-Domain
12.11: Exercise- Artificially Inflated Decoding with Unbalanced Classes
12.12: Exercise- Methods for Appropriate Decoding with Unbalanced Classes
12.13: Exercise- Using Area Under the Curve (AUC) with the ERP CORE P3b Oddball Paradigm
12.14: Matlab Script For This Chapter
12.15: Key Takeaways and References

13: Appendix 1- A Very Brief Introduction to EEG and ERPs

14: Appendix 2- Troubleshooting Guide

14.1: Data for This Appendix
14.2: A2.1 The First Step
14.3: A2.2 Some Basic Solutions
14.4: A2.3 Taking a Scientific Approach
14.5: A2.4 Deciphering Matlab's Error Messages
14.6: A2.5 Debugging Scripts by Performing Experiments and Collecting Data
14.7: A2.6 Avoiding Bugs in Your Scripts with Good Programming Practices
14.8: A2.7 References

15: Appendix 3- Example Processing Pipeline
Index
Glossary
Detailed Licensing

This page titled Applied Event-Related Potential Data Analysis 2e is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck.

Licensing

A detailed breakdown of this resource's licensing can be found in ***Back Matter/Detailed Licensing***.

This page titled Licensing is shared under a Public Domain license and was authored, remixed, and/or curated by Steven J Luck.

Hardware and Software Requirements

The core of this book is a large set of data processing and analysis exercises. They are designed to run on your own computer using ERPLAB Studio, a free, open-source Matlab program that provides an easy-to-use user interface to two free, open-source toolboxes, EEGLAB and ERPLAB. Although these packages are free, you will need a Matlab license to run them. Check with your institution's Information Technology office to see if they provide free or reduced-cost Matlab licenses. It may also be possible for you to purchase an inexpensive student license. In particular, you will need:

- Matlab version 2023b or later
    - The Signal Processing Toolbox is required. If you don't have it, contact your institution's IT support department for assistance.
    - The Statistics and Machine Learning Toolbox is recommended (and required for Chapter 12).
    - To see what toolboxes are already installed in Matlab, type **ver** on the Matlab command line.
    - Some parts of the EEGLAB and ERPLAB code will run under Octave, a free Matlab simulator. However, Octave will not run the graphical user interface for ERPLAB Studio, which is necessary for most of the exercises in this book. You should therefore use Matlab rather than Octave.
    - EEGLAB version 2024.0 or higher (see Chapter 1 for installation instructions)
- ERPLAB version 13.1 or higher (see Chapter 1 for installation instructions)

EEGLAB and ERPLAB will run on any reasonably recent version of Mac OS, Windows, or Linux. EEGLAB and ERPLAB are designed for professional users, and they work best on a computer with a lot of RAM, a large hard drive, and a high-resolution external monitor. However, they will function adequately on a midrange laptop. Absolute minimum specifications are:

- 8 MB RAM (but more will work much better, especially for datasets with 64+ channels)
- A screen height of at least 1080 pixels (but more is better)

This page titled Hardware and Software Requirements is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- Hardware and Software Requirements by Steven J Luck is licensed CC BY 4.0.

Preface

Preface to the First Edition

When I was a new assistant professor, journal editors kept sending me ERP papers to review that were just awful. I think the editors didn't want to waste the time of more established researchers with these terrible manuscripts. The papers were generally written by researchers who were relatively new to the technique and had not "grown up" in an ERP lab. Because they had not received much ERP training, they made all kinds of errors in the design, the analysis, and the interpretation of the experiments.

That experience made me realize how fortunate I had been to be trained in Steve Hillyard's lab, which traced its roots to the very first published ERP experiment (Davis, 1939). I decided to write a book that distilled everything I had learned in the Hillyard lab so that everyone could benefit from this knowledge (Luck, 2005). I also started running in-person training workshops called ERP Boot Camps along with Emily Kappenman. Emily and I also put together an edited volume on ERP components (Luck & Kappenman, 2012). To make it easier for researchers to analyze their data using the methods we were promoting, I worked with Javier Lopez-Calderon to release ERPLAB Toolbox, a free ERP data processing package (Lopez-Calderon & Luck, 2014). And then I wrote a second edition of my ERP book (Luck, 2014), taking advantage of everything I had learned from the ERP Boot Camp about how to explain ERP methodology.

Although these efforts have reached thousands of researchers, I always felt that something was missing. The books and workshops provided a mix of theory and practical advice, but they were necessarily quite broad given the wide range of researchers who use ERPs. It can be very difficult to take these broad ideas and apply them to the analysis of actual experiments. So, what was missing from our books and our workshops was the opportunity to solve all the problems that arise when you're analyzing real data. We tried having data analysis tutorials in a few of the early ERP Boot Camps, but we found that we spent 98% of our time teaching the participants how to run the software. Also, we ended up teaching them how to analyze data from just a single experiment, which was often quite different from their own research interests.

The present book is designed to fill this gap. It includes tons of example data sets and exercises, all of which run on the free ERPLAB Toolbox package. These examples use data from the ERP CORE (Kappenman et al., 2021), which has data from six classic ERP paradigms. As a result, the examples in this book cover a broad range of paradigms and components with real data.

I've published this book on the free LibreTexts platform so that anyone in the world can learn about ERP data analysis for free. I was extremely fortunate to have great opportunities to learn about ERPs as a student, and I'm trying to "pay it forward" with this book. If you teach the ERP technique to other people (either in courses or in your lab), you should feel free to remix and reuse the book in any way that you find helpful.

Preface to the Second Edition

When we started working on the first version of ERPLAB Toolbox, we decided to make it an EEGLAB plugin so that we could take advantage of EEGLAB's routines for importing and preprocessing EEG and didn't need to "reinvent the wheel". That was a good decision, and the number of ERPLAB users has steadily grown over the years. I just checked Google Scholar, and it reports that over 4000 publications have used ERPLAB Toolbox. That's very gratifying.

However, I always found that it was cumbersome and inefficient to access ERPLAB through the EEGLAB graphical user interface (GUI). I therefore decided that it was time for ERPLAB to become a standalone program, which we christened *ERPLAB Studio*. ERPLAB Studio uses the same underlying EEGLAB and ERPLAB code that we've always used, but with a more modern and efficient user interface. ERPLAB Studio is especially beneficial to beginners, because it's much easier to see the results of each processing step.

The exercises in the first edition of this book used the EEGLAB plugin version of ERPLAB (which we now call *ERPLAB Classic*). As we were creating ERPLAB Studio, I knew that our new program would make the exercises much easier to run. I therefore created the second edition of this book, using ERPLAB Studio for all the exercises. I also made many small improvements to the book, and I made substantial changes to a few chapters (especially the Filtering chapter) and added a new chapter on multivariate pattern analysis (decoding).

I hope that ERPLAB Studio and this new edition of the book are helpful as the next generation of researchers learns how to

process and analyze EEG/ERP data, leading to important new scientific discoveries.

Funding

Preparation of this book was made possible by grant R01MH087450 from the National Institute of Mental Health.

References

Davis, P. A. (1939). Effects of acoustic stimuli on the waking human brain. *Journal of Neurophysiology*, *2*, 494–499.

Kappenman, E. S., Farrens, J. L., Zhang, W., Stewart, A. X., & Luck, S. J. (2021). ERP CORE: An Open Resource for Human Event-Related Potential Research. *NeuroImage*, *225*, 117465. https://doi.org/10.1016/j.neuroimage.2020.117465

Lopez-Calderon, J., & Luck, S. J. (2014). ERPLAB: An open-source toolbox for the analysis of event-related potentials. *Frontiers in Human Neuroscience*, *8*, 213. https://doi.org/10.3389/fnhum.2014.00213

Luck, S. J. (2005). *An Introduction to the Event-Related Potential Technique*. MIT Press.

Luck, S. J. (2014). *An Introduction to the Event-Related Potential Technique, Second Edition*. MIT Press.

Luck, S. J., & Kappenman, E. S. (2012). *The Oxford Handbook of Event-Related Potential Components*. Oxford University Press.

This page titled Preface is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- Preface by Steven J Luck is licensed CC BY 4.0.

Acknowledgments

This book simply would not have been possible if it had not been for the hard work and creativity of the wonderful people who have worked in my lab over the years.

I'd like to start with a shout-out to Emily Kappenman. Emily and I have worked together for almost 20 years on the ERP Boot Camp and many other efforts designed to promote rigorous and high-impact ERP research. Emily spearheaded ERP CORE, which is one of the two main pillars of this book. Jaclyn Farrens also put years of work into ERP CORE, and ERP CORE was pushed across the finish line with help from Wendy Zhang and Andrew Stewart.

The other pillar of this book is ERPLAB Toolbox. The first several releases of ERPLAB were written entirely by Javier Lopez-Calderon, and his fingerprints can still be seen throughout the software. After Javier moved on, Andrew Stewart took over programming duties, and he wrote the initial code for the data quality metrics. After Andrew's departure, my lab manager Aaron Simmons filled in for a while, and it's amazing how much he added while simultaneously running my lab. Guanghui Zhang then became the primary ERPLAB developer, and it's hard to believe how much progress he made over just a few years in my lab.

The second edition of this book relies heavily on ERPLAB Studio, our new interface into EEGLAB and ERPLAB. Andrew Stewart got the ERPLAB Studio project started (with a little help from my son, Carter Luck). Guanghui Zhang then took over and wrote most of the code. David Garrett helped with debugging and documentation. And the entire lab provided feedback during the development process.

I'd also like to thank the other members of my lab who provided feedback on drafts of the first edition of this book, including (alphabetically) Brett Bahle, Carlos Carrasco, John Kiat, Lara Krisst, Orestis Papaioannou, and Kurt Winsler. I received additional feedback from several members of the UCD EEG Research Group.

I'd like to thank Stella Fultz and Guanghui Zhang for help with proofreading the second edition.

In addition, I'd like to acknowledge the National Institute of Mental Health, which has funded ERPLAB for many years (grant R01MH087450) and made this book possible.

Finally, I'd like to thank Lisa and Ruthie, who kept me sane during the writing process.

This page titled Acknowledgments is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- Acknowledgments by Steven J Luck is licensed CC BY 4.0.

How to Cite This Book

Cite as: Luck, S. J. (2026). *Applied Event-Related Potential Data Analysis, Second Edition*. LibreTexts. https://doi.org/10.18115/D56G81

This page titled How to Cite This Book is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- Acknowledgments by Steven J Luck is licensed CC BY 4.0.

1: FIRST STEPS

Learning Objectives

In this chapter, you will learn to:

- Install the EEGLAB and ERPLAB toolboxes and download the data for the exercises
- Load EEG and ERP data
- Plot EEG waveforms and spectra
- Plot ERP waveforms and scalp maps
- Update the Matlab PATH when you install or update EEGLAB/ERPLAB

This chapter is designed to provide key background knowledge you'll need for the rest of the book. We'll begin by describing the goals of this book and the background knowledge you should have. Then we'll show you how to install the software and download some data, followed by a few exercises so that you can learn the basics of ERPLAB Studio and the underlying EEGLAB and ERPLAB functions.

1.1: Getting Started

1.2: Installing the Software and Downloading the Data

1.3: Exercise- Loading and Viewing an EEG File

1.4: Exercise - Viewing EEG Spectra

1.5: Exercise- Loading and Plotting ERPs

1.6: Understanding the Matlab Path

1.7: Key Takeaways and References

This page titled 1: First Steps is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 1: First Steps by Steven J Luck is licensed CC BY 4.0.

1.1: Getting Started

This section of Chapter 1 provides some important background information that you need to know to make effective use of this book. The whole book will make much more sense if you read this section first!

The Goal of This Book

The primary goal of this book is to help students and researchers learn how to process and analyze event-related potentials (ERPs). My first ERP book, *An Introduction to the Event-Related Potential Technique,* focused on providing a conceptual understanding of ERPs, and the present book focuses on applying those concepts to real data. Theory is important, but there is no substitute for loading up real data—with all its warts and complexities—and figuring out how to go from a gigabyte of raw EEG files to a few kilobytes of statistical analyses and figures that are ready for publication.

At its essence, this book is a set of data processing and analysis exercises that are wrapped in explanatory text. ERP analysis involves a million decisions, such as whether to filter before versus after artifact rejection and what measurement window to use for quantifying the amplitude of an ERP component. The exercises in this book are designed to give you experience making the choices that will lead to the most robust and valid conclusions. In theory, you could read the book without doing the exercises, but that would be like trying to learn painting from a textbook without ever picking up a paintbrush. So fire up Matlab and get ready to process some data!

I wanted to write this book for many years, but I waited because I needed two things: 1) Free software that anyone can use to do the exercises, and 2) a large public dataset with multiple different ERP paradigms. Both are now available. The free software consists of ERPLAB and its companion EEGLAB, and the large public dataset is the ERP CORE (Compendium of Open Resources and Experiments).

ERPLAB and EEGLAB

ERPLAB (Lopez-Calderon & Luck, 2014) is an open-source Matlab toolbox that my lab produces with support from an NIH grant. The original version of ERPLAB, now called **ERPLAB Classic**, is a plugin that operates within the **EEGLAB** package (Delorme & Makeig, 2004). **ERPLAB Studio** (or **eStudio** for short) is a standalone application that provides a more user-friendly interface to the same underlying ERPLAB and EEGLAB code. eStudio is included in the ERPLAB package, so you already have it if you've downloaded a recent version of ERPLAB.

I have some good news and some bad news: ERPLAB and EEGLAB are completely free, but Matlab is not free, and you will need it to run EEGLAB and ERPLAB. However, most institutions provide reduced-cost Matlab licenses, and the student version is even less expensive (or even free at some institutions). Matlab has become the *lingua franca* of cognitive neuroscience, and it's well worth the investment.

The ERP Core

The ERP CORE is a set of six classic ERP paradigms that have been optimized to isolate seven widely studied ERP components (N170, mismatch negativity, N400, P3b, N2pc, error-related negativity, and lateralized readiness potential). Emily Kappenman and I created the ERP CORE to provide a set of "reference" data that could be used by a large set of researchers for a wide range of purposes. The public resource includes the experimental control scripts, data from 40 neurotypical young adults, and EEGLAB/ERPLAB processing scripts for every stage of processing and analysis. If you'd like to know how to obtain a robust N400, you can download our N400 experimental control script to see how it's done. If you'd like to see how to professionally analyze the error-related negativity (ERN), you can download our ERN processing scripts. If you've just put together your own EEG recording system and you'd like to see if everything is working, you can run one or more of our paradigms and compare your data with our data (including quantitative metrics of data quality).

The ERP CORE is particularly useful for the present book because it provides data from many different paradigms, and yet the data are similarly formatted for each paradigm. That way you can see how to process many different types of data, but you won't have to deal with superficial differences between data sets (e.g., differences in file naming conventions). Although you can download the ERP CORE files directly from the ERP CORE site, for the exercises in this book you should instead download the data using

the links provided within each chapter, which provide a more streamlined set of files.

Read This Now or You'll Be Sorry!

The original heading for this section was “Troubleshooting,” but that sounds way too boring, and I figured many people would skip it. But don’t skip it! This is the most important section of this chapter.

Unless you already have years of experience with EEGLAB and ERPLAB, you’re bound to run into some bugs or other difficulties when you try to complete the exercises in this book. EEGLAB and ERPLAB are professional-strength software packages designed for state-of-the-art research, and the datasets used in the exercises are large and complex. As a result, you'll probably run into some problems when you run the exercises, including error messages, results that don’t match those shown in the book, etc.

When you encounter one of these problems, you’ll certainly be frustrated, and you might even be tempted to curse my ancestors. But these problems are actually an essential part of the learning process. When you’re analyzing your own data, you’ll run into many of the same problems. In fact, the problems will probably be worse with your own data, because the data used in the present exercises have been carefully chosen to avoid many common problems.

So, when you run into a problem, try to look at it as an opportunity for growth. Of course, you might let out a few expletives. And you may need to spend a minute doing deep breathing exercises before you remember that the error message on the screen is actually a gift in disguise. To reduce your blood pressure and help you learn the art of troubleshooting, we’ve provided a Troubleshooting Guide in Appendix 2. I recommend skimming the first two sections now and then returning to it later when you inevitably run into problems.

Here's a related but more positive piece of advice: Play! I learned the important of playing in science from one of my undergrad mentors, Allen Neuringer. If you simply follow the exercises in this book exactly as written, you’ll learn a lot. But if you want to really understand what you’re doing, you should spend considerable time playing around with things (e.g., the various options in the eStudio GUI). For example, there are many things that I say you shouldn’t do when analyzing ERP data, such as applying a high-pass filter to averaged ERP waveforms, but I actually recommend that you try them and see what happens. Just don't publish the results.

Expected Background Knowledge

This book assumes that you already have some very basic knowledge about ERPs. If you don’t, Appendix 1 provides a quick overview. Here are some things the book assumes you already know about:

- Generation of the EEG and ERPs from postsynaptic potentials in cortical pyramidal cells
- The 10/20 system for electrode locations
- Creating averaged ERPs from single-trial EEG epochs
- Artifact rejection and artifact correction
- ERP peaks and components

All of these issues are briefly covered in Appendix 1, and you should read about them if they are not already familiar. Or, better yet, you can take my free online course, Introduction to ERPs, which typically takes about 4 hours to complete. The first 2 “chapters” of the online course would be enough to get you started and should take you less than an hour. You could also read the chapter I wrote for the *APA Handbook of Research Methods, Second Edition* (Luck, 2024).

Much of the theory behind the analysis approaches described in this book is described in *An Introduction to the ERP Technique, Second Edition* (Luck, 2014). If you want to become an ERP researcher, you need to understand the reasons behind the processing steps, so I recommend going back and forth between that book and the present book. You’ll see lots of places in the present book where I point you to the relevant chapters in *An Introduction to the ERP Technique*. A third edition of *An Introduction to the ERP Technique* should be available in 2027, and it's even more directly linked with the present book.

You should also consult the online documentation for EEGLAB and ERPLAB if you want to understand some of the options and parameters in the software. There are millions of details about the operation of the software that aren't repeated in this book.

Scripting

ERPLAB Studio provides an easy-to-use graphical user interface (GUI) that allows you to process data by pointing and clicking. With the exception of the last chapter, all the exercises in this book use this GUI.

However, you can achieve a lot of additional power and flexibility by writing Matlab *scripts*, which are text files that specify each processing operation with a line of code. Scripts allow you to automate the EEG and ERP processing steps, which is a huge time-saver (especially when your mentor or a reviewer makes you reprocess all of your data). If you already know how to write Matlab scripts, then you'll find it straightforward to write scripts with the EEGLAB and ERPLAB routines. If you don't know Matlab but you have some significant experience in one or more other programming languages, you'll be able to pick up Matlab pretty quickly (although it has a few quirks that you'll need to learn).

If you don't have much programming experience, EEGLAB and ERPLAB provide an excellent starting point for you to learn. Every operation that you perform in the eStudio GUI corresponds to a line of code, and every time you perform an EEGLAB or ERPLAB operation in eStudio, that line of code is saved to a *history*. You can grab these lines of code from the history, paste them into a text file, and voila! You have a script!

To get real power and flexibility, however, you also need to learn a little bit about the Matlab programming language. Chapter 11 is devoted to teaching you how to write EEGLAB/ERPLAB scripts. It's designed for people at all levels of prior programming experience. However, it does assume that you know some basic programming concepts (e.g., variables, loops). If you want to learn scripting—which is an incredibly useful skill—I recommend taking a Matlab course and/or working through one or more Matlab books. I particularly recommend a book called *Matlab for Behavioral Scientists* (Rosenbaum et al., 2014) and the online Introduction to Programming with MATLAB course from Coursera.

I've also provided example scripts at the end of each chapter, showing you how to implement the GUI steps from that chapter in a script. I find that it's much easier to start with an example script and modify it than it is to write a script from scratch.

But you do not need to know any scripting to use this book. It's a good thing to know eventually, but it can wait until you've mastered processing data with the GUI.

This page titled 1.1: Getting Started is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 1.1: Getting Started by Steven J Luck is licensed CC BY 4.0.

1.2: Installing the Software and Downloading the Data

Minimum hardware and software requirements were listed in the front matter. However, keep in mind that those are only the minimum. You will find the exercises to be much more pleasant if you have at least 16 GB of RAM and at least a 1080p screen. You can run the software on a laptop, but everything will be easier if you run it on a larger screen (e.g., on a desktop computer or on a laptop with an external monitor).

Installing Matlab

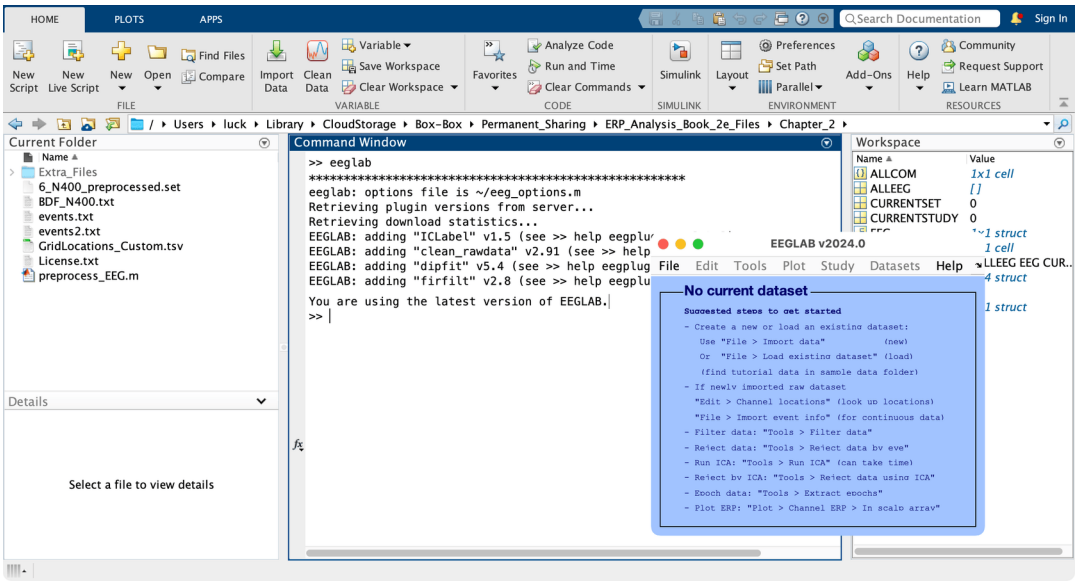
You will need Matlab version 2021a or later (2023b or later if you are using a Mac with Apple Silicon). Your Matlab installation must include the Signal Processing Toolbox. The Statistics and Machine Learning Toolbox is also required, but only for the decoding exercises in Chapter 12. Once you've installed Matlab, you can see what toolboxes are installed by typing **ver** on the Matlab command line. If you don't have the necessary toolboxes, contact your institution's IT support department for assistance.

Installing EEGLAB and ERPLAB

You will need EEGLAB 2025.1.0 or later and ERPLAB 13.1 or later to run all of the exercises in this book. New versions of Matlab, EEGLAB, and ERPLAB are released at least once per year, and these new versions can lead to changes in how things look (and occasionally changes in the results). To write this book, I mainly used Matlab 2024a, EEGLAB 2025.1.0, and ERPLAB 13.1 on a MacBook Pro running macOS 15.7. Everything should work fine if you use Windows or Linux, or if you use newer versions of EEGLAB or ERPLAB, but the screenshots will look a little different. You will almost certainly run into problems with older versions of Matlab, EEGLAB, and ERPLAB. See the Hardware and Software Requirements section for more details.

You must install EEGLAB before installing ERPLAB. You can find the documentation for EEGLAB, including instructions for downloading and installing, at https://eeglab.org. Make sure to install version 2025.1.0 or later. As noted in the EEGLAB installation instructions, you will need to update the Matlab PATH (whether you are installing EEGLAB for the first time or upgrading to a newer version). If you don't know how to do this, or you can't get EEGLAB to launch, you should read the description of how the PATH works in Section 1.6 of this chapter. If you're having trouble launching EEGLAB but the PATH isn't the problem, see the ERPLAB FAQ page, the EEGLAB documentation, or the Troubleshooting Guide in Appendix 2.

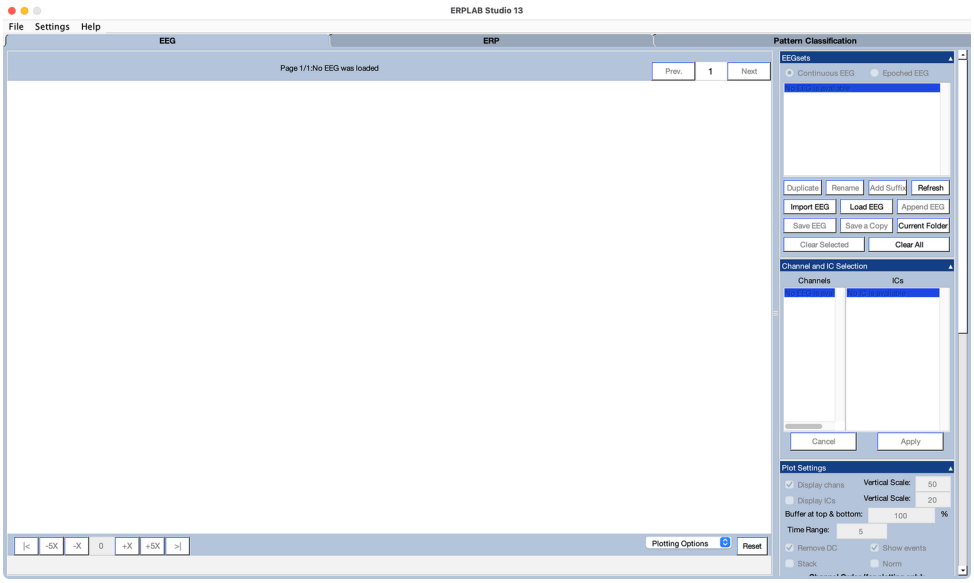
Once you've installed EEGLAB and updated the PATH, you should launch EEGLAB by typing **eeglab** in the Matlab command window. Screenshot 1.1 shows what it should look like when EEGLAB has been launched. Once you've verified that you can launch the EEGLAB GUI, you should quit it. Launching it once is important for getting some things set up, but you won't need the EEGLAB GUI for any of the exercises in this book (and you should not try to run it simultaneously with eStudio).



Screenshot 1.1

Once you have verified that EEGLAB is working, you can download and install ERPLAB (version 13.1 or later). The ERPLAB documentation, including installation instructions, can be found at https://github.com/lucklab/erplab/wiki. There are two ways to install ERPLAB. If you just want the latest major release, you can launch EEGLAB and use its built-in extensions manager by going to **File > Manage EEGLAB extensions** in the EEGLAB GUI. It will pop up a window with a list of extensions; select **ERPLAB** and click **install/update**. However, the EEGLAB extensions manager is sometimes out of date, and it might not contain the latest version of ERPLAB. To get the latest version (or an earlier version), you can manually download ERPLAB at https://github.com/ucdavis/erplab/releases. You can then follow the installation instructions at https://github.com/ucdavis/erplab/wiki/Installation. No matter which installation route you choose, make sure you download ERPLAB version 13.1 or later.

Once ERPLAB has been installed, you can launch it by typing **estudio** in the Matlab command window. It's a complex piece of software, so it takes some time to load, but you should see some progress information in the command window while it loads. Once it's done loading, you should see something like Screenshot 1.2. If it doesn't load, try running and then quitting EEGLAB again before launching eStudio.



Screenshot 1.2

ERPLAB Installation Problems

ERPLAB is installed inside the plugins folder within the EEGLAB folder (e.g., eeglab2025.1.0 > plugins > erplab). The most common installation problem is that ERPLAB is not located in this place. If it's installed correctly, you should see a folder named **studio_functions** directly inside the **erplab** folder (and not inside another embedded folder). ERPLAB has a Frequently Asked Questions page with information about solving such problems.

The other common problem is that the path isn't set correctly. To solve this, run EEGLAB and then quit it. Then click the Matlab **Set Path** button and save the path.

Downloading Data for the Exercises

The datasets for the exercises are stored in a cloud storage system, and the master folder can be accessed at https://doi.org/10.18115/D5G045. This master folder contains a set of subfolders with the datasets for each chapter. Unfortunately, the cloud storage system may be blocked in some countries, but I have faith that clever readers will find a workaround that allows them to access the data. The data have been released with a Creative Commons license that permits them to be copied, used, remixed, and reposted as long as the original source is cited (see the **license.txt** file in each subfolder). Note that some of the folders are quite large and may take quite a while to download.

Free Data!

Once you've finished using this book to learn how to analyze ERP data, you should feel free to download the original data from the ERP CORE. The CORE is a very rich dataset, and you could use it for novel analyses that might be publishable. Several papers have already been published that report secondary analyses of the ERP CORE data. Wouldn't it be nice to publish an ERP paper without having to collect the data yourself?

This page titled 1.2: Installing the Software and Downloading the Data is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

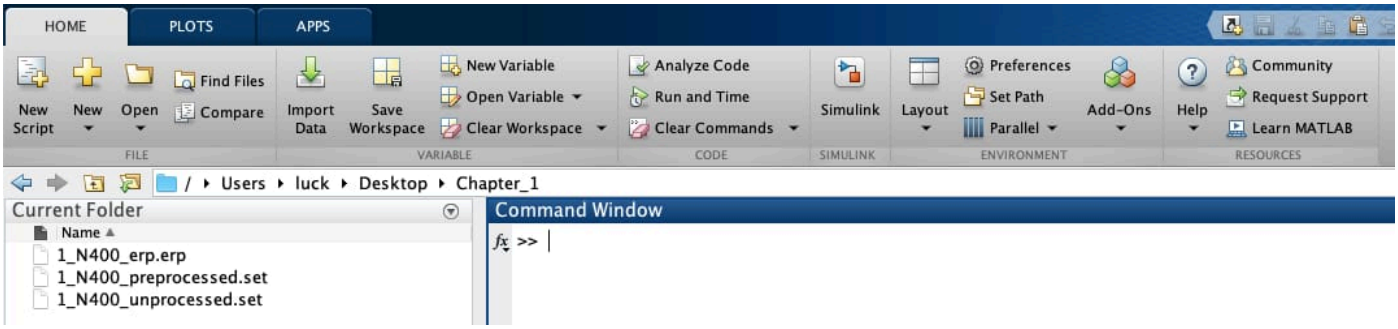
- 1.2: Installing the Software and Downloading the Data by Steven J Luck is licensed CC BY 4.0.

1.3: Exercise- Loading and Viewing an EEG File

We're now going to start a quick tour of ERPLAB Studio (also called eStudio). In the first exercise, you'll look at the EEG data and the averaged ERPs from one subject in an N400 experiment that will be described in more detail in the next chapter.

If Matlab is not already running, launch it. Then launch eStudio by typing **estudio** in the Matlab command window (if it is already running, quit it and then launch it again). It's usually a good idea to reset all of eStudio's settings at the beginning of a chapter. That way, eStudio will be in its default state, and what you see will better match the screenshots in this book. The best way to do this is to click the **Reset** button under the lower right corner of the plot region, wait for the **Reset** window to appear, and then click **Reset ERPLAB Studio**. This is like a "factory reset" that returns all settings to their default values and clears all loaded files. You can also reset settings and clear files for the individual **EEG**, **ERP**, and **Pattern Classification** tabs.

Now download the **Chapter_1** folder from the master folder at https://doi.org/10.18115/D5G045. Note that an underscore is used instead of a space in the folder name. Spaces can sometimes confuse Matlab, so it's best to use underscores or dashes instead of spaces in your folder names and file names. But don't use any other non-alphanumeric characters. Set Chapter_1 to be Matlab's *current folder*. You can do this either from the main Matlab window or by going to the **EEGsets** panel in eStudio and clicking **Current Folder**. The current folder is part of Matlab's path, and it's the first place Matlab will look for files and code. Screenshot 1.3 shows what the Matlab GUI should look like once you've set the current folder. You'll see the path to the folder and the contents of the folder.

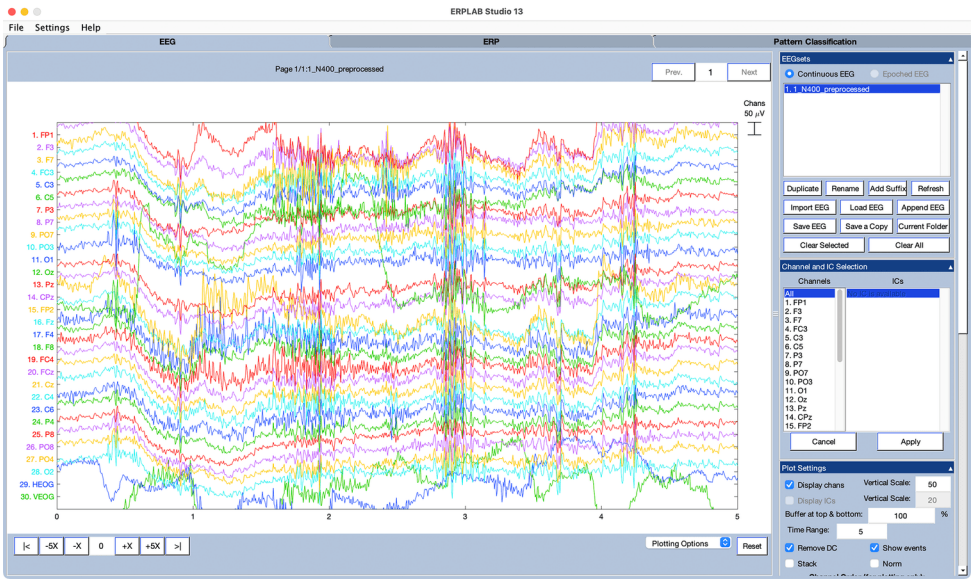


Screenshot 1.3

There are various ways of changing the current folder, including double-clicking on a folder name within the current folder. Play around with this a bit (googling it if necessary using a search phrase such as "matlab current folder GUI"). You'll be changing the current folder frequently, so you'll want to know how to do it efficiently.

Once you have the correct current folder, go to the **EEGsets** panel in eStudio and click **Load**. It will bring up a dialog box, which should show the contents of the Chapter_1 folder. Select and open the file named **1_N400_preprocessed.set**. This file is called an *EEGset*, which means that it contains the EEG data for one participant. [Note that EEGLAB uses the term *dataset* instead of *EEGset*, but ERPLAB uses a separate type of file called an *ERPset* to hold averaged ERPs, so *dataset* isn't a sufficiently precise term for a file that holds EEG data but not averaged ERPs.]

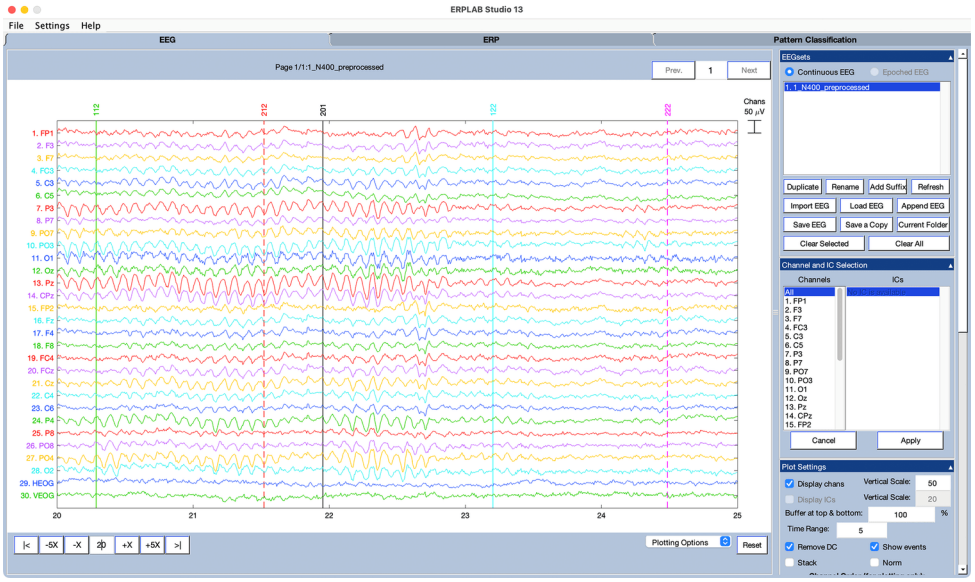
As shown in Screenshot 1.4, you should now see **1_N400_preprocessed.set** listed in the EEGsets panel and the EEG waveforms in the plot region.



Screenshot 1.4

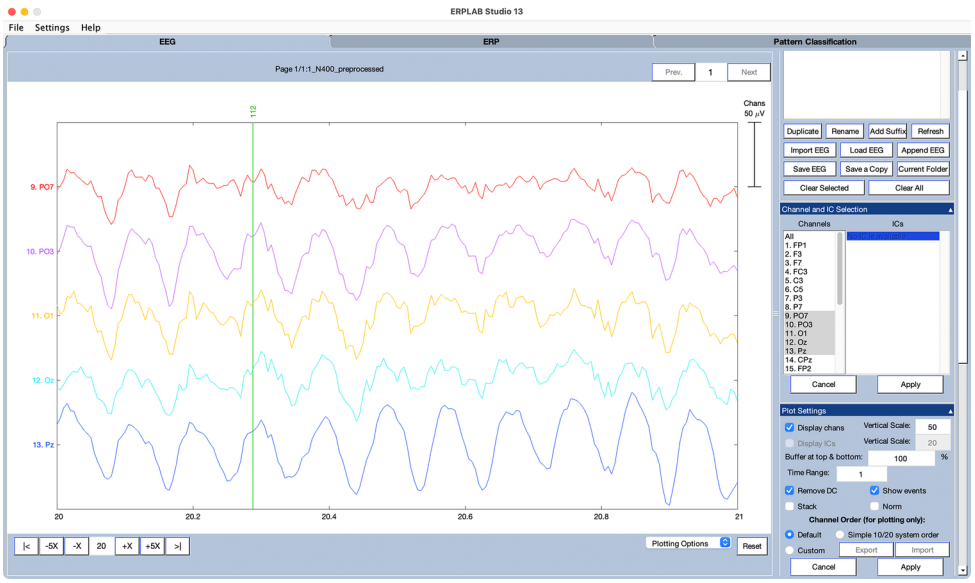
The plot region shows the EEG data from all of the electrode sites for the first five seconds of the recording. If you click the **+X** button near the bottom left corner of the window, the plot will shift rightward to show you the next 5-second period. The **+5X** button shifts 25 seconds rightward. If you were to change the **Time Range** in the **Plot Settings** panel to 10 (and click **Apply**), you would see 10 seconds of data per screen, **+X** would shift the plot rightward by 10 seconds, and **+5X** would shift by 50 seconds.

You can also go to a particular time point by typing a number into the box between the **-X** and **+X** buttons (and hitting Return/Enter). To see this, type **20** into that box right now (followed by Return/Enter) to show the time period beginning 20 seconds from the start of the recording. You should see something like Screenshot 1.5.



Screenshot 1.5

If you look at the occipital and parietal electrode sites (e.g., Pz in Channel 13), you can see an oscillating wave. Let's take a closer look by restricting the plot to a 1-second period and focusing on the posterior electrode sites. First, change the **Time Range** to **1** in the Plot Settings panel (and click **Apply**). Then go to the Channel & IC Selection panel and select Channels 9-13 (and click **Apply**). You should see something like Screenshot 1.6.



Screenshot 1.6

Now count the number of positive peaks between 20 and 21 seconds. You should see 11 peaks in this 1-second period. 11 peaks per second means that it's an 11 Hz oscillation. This is called the *alpha wave*, which was the very first feature of the human EEG to be described (Berger, 1929). Alpha waves are oscillations between approximately 8 and 12 Hz that are largest over posterior electrode sites and are particularly large when the participant is zoned out or has closed eyes. I like to think of alpha waves as indicating that the participant's attention is internally focused instead of externally focused. In most experiments, you want the subject to be alert and paying attention externally to the task, so you don't want to see a lot of alpha activity. However, most ERP experiments are pretty boring, so the participants are often a little sleepy (which makes them internally focused), and they generate quite a bit of alpha. When a participant in my lab produces a lot of alpha, we often give them a short break and/or offer a caffeinated beverage.

Now let's look at the vertical scale. If you look near the right edge of the plot region, you should see a vertical scale marker labeled 50 µV. This means that the vertical space indicated by the marker corresponds to 50 µV. The oscillations at the PO7 electrode site are about half as big as this marker, so they're about 25 µV (peak to peak). If we wanted to zoom in closer, we need to do something a little counterintuitive: We'd need to specify a smaller number of µV for the vertical scale. Let's give it a try. Type **25** in the Vertical Scale box in the **Plot Settings** panel (and click **Apply**). The scale marker is now the same size (in pixels), but it represents 25 µV instead of 50 µV. The oscillations at the PO7 electrode site are now about the same size as the scale marker.

The last thing we're going to do in this section is restore all the plotting parameters to their default values. That way, the EEG you see in eStudio will match the screenshots in the subsequent sections. To do this, click the **Reset** button under the lower right corner of the plot region. After a pause, you will see a small window that allows you to reset the parameters. Make sure that the box for **Restore default parameters for EEG tab** is checked. We haven't changed any of the ERP panels, so we don't need to reset those. And we don't need to clear any EEGsets or ERPsets. Just click **Continue** to restore the defaults for the EEG panels.

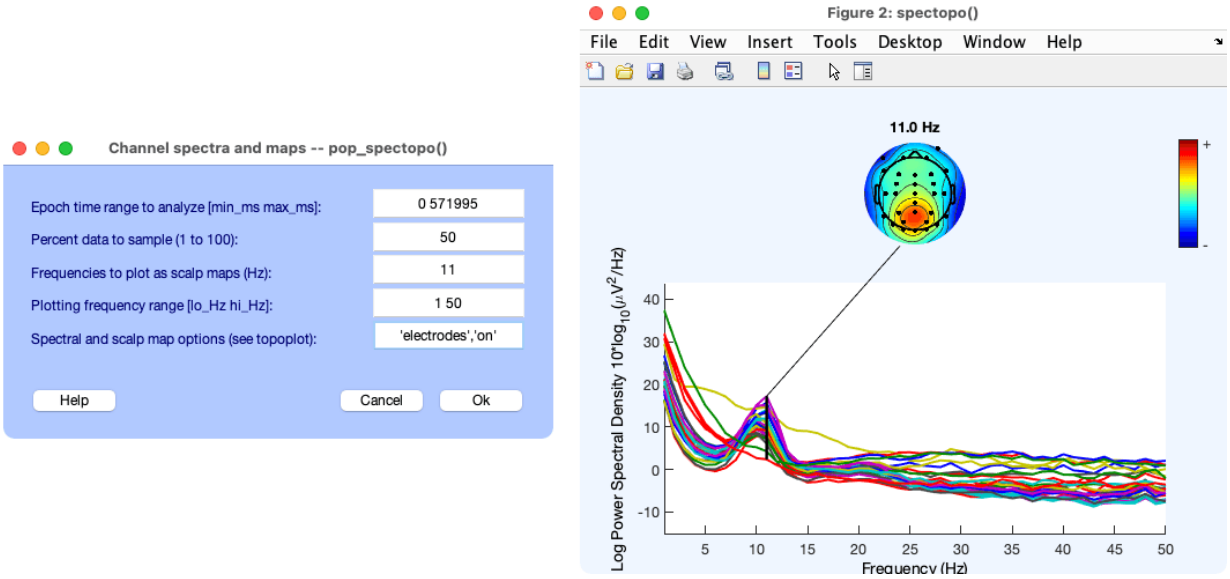
This page titled 1.3: Exercise- Loading and Viewing an EEG File is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 1.3: Exercise- Loading an EEG File by Steven J Luck is licensed CC BY 4.0.

1.4: Exercise - Viewing EEG Spectra

Now we're going to use an EEGLAB tool to take a closer look at the alpha wave. eStudio provides access to several useful EEGLAB routines through the **EEGLAB Tools** panel. Go to this panel and click the **Spectra & maps** button. A window will appear allowing you to specify some parameters; adjust the values in this window so that they match the values shown in the left portion of Screenshot 1.7 and then click **OK**. You should now see a window like that shown on the right side of Screenshot 1.7.

This routine performs a Fourier transform of the EEG, converting it from the time domain to the frequency domain (see Chapters 1 and 12 in Luck, 2014). The waveforms at the bottom of the window show the magnitude at each frequency in the EEG data, with one waveform for each channel. You can see that most of the channels have a peak around 11 Hz, which is the alpha wave you saw previously. The topographic map at the top shows how the strength at 11 Hz varies across the scalp (using interpolation to estimate the values between the electrode sites). You can see a big peak over the posterior electrode sites, where the alpha is largest.



Screenshot 1.7

Try playing around with the settings for this routine (e.g., getting scalp maps for different frequencies). When you're done, you can close this EEGLAB window.

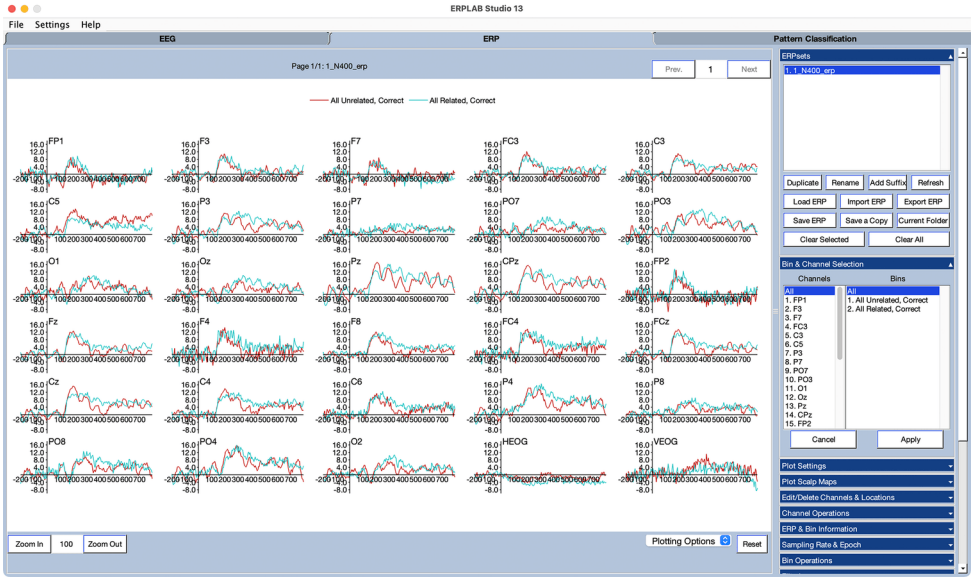
This page titled 1.4: Exercise - Viewing EEG Spectra is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 1.5: Exercise - Viewing EEG Spectra by Steven J Luck is licensed CC BY 4.0.

1.5: Exercise- Loading and Plotting ERPs

Now we're going to load some averaged ERP data and plot the waveforms. Click on the **ERP** tab near the top of the eStudio window, go to the **ERPsets** panel, click **Load**, and open the file named **1_N400_erp.erp**. (I previously created this ERPset by averaging together the trials in the EEG dataset that you looked at in the previous exercise. We'll see how this averaging process works later.)

You should see something like Screenshot 1.8. This ERPset contains ERPs for two different **bins** (two different experimental conditions), labeled "All Unrelated, Correct" and "All Related, Correct". There is a separate waveform for each electrode site for each of the two bins.



Screenshot 1.8

You can control which bins and channels to plot using the Bin & Channel Selection panel, and you can control a variety of plotting parameters using the Plot Settings panel. We will talk more about plotting averaged ERP waveforms in the next chapter. You can play around with the different settings in this panel to see how they impact the plot (but then click the **Reset** button right under the bottom right corner of the plotting region and reset the default parameters for the ERP tab).

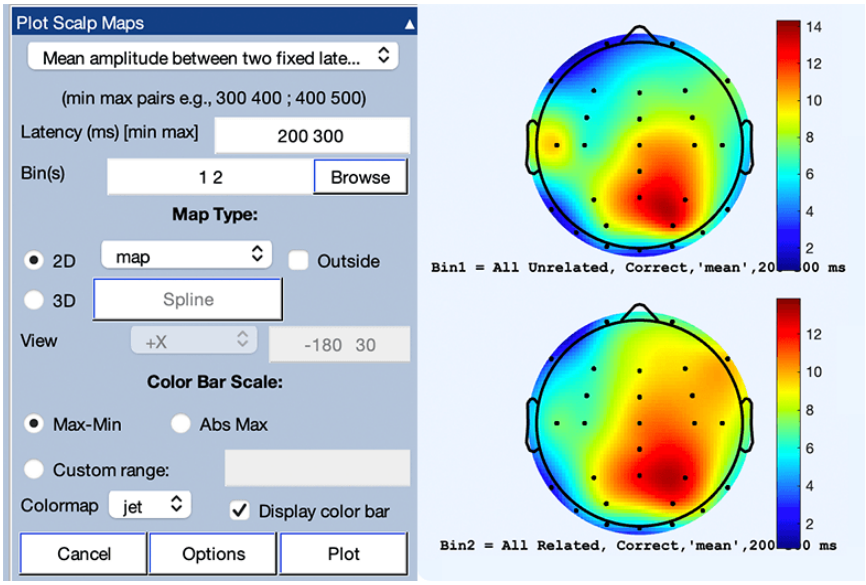
Making High-Quality Figures

The plots that appear automatically in the plot region are not designed to be publication-quality figures. To make nice figures for papers, posters, or presentations, you can use the Advanced Waveform Viewer, which you can access from the Plotting Options menu underneath the plot region. This tool gives you much more control over how the plots look, and it can often create final plots for publication. You can then save the plots in a vector format, such as PDF or SVG, which can then be inserted into your paper/poster or edited further in a graphics program. In my lab, we save the files in SVG format and then edit the images further in a graphics program called **Affinity Designer** (which is like Adobe Illustrator, but easier to use and much less expensive).

You can also save the plots in a variety of bitmap formats (e.g., .bmp, .jpg, .tif). However, I recommend against using bitmap formats, because they're difficult to edit and look terrible when blown up. If you don't know the difference between vector and bitmap formats, you should search for "vector versus bitmap" in your favorite search engine. It's an important distinction when you're saving plots of ERP waveforms.

Now let's plot topographic maps of the ERPs (aka *scalp maps*). Open the **Plot Scalp Maps** panel and set it up as shown on the left of Screenshot 1.9. Then click the **Plot** button, and you should see something like the scalp maps on the right side of the

screenshot. The coloring indicates the mean voltage between 200 and 300 ms at each point on the scalp (using interpolation to fill in the values between electrodes). The scale along the right side indicates what amplitude (in microvolts) is indicated by a particular color.



Screenshot 1.9

Once you're done looking at the scalp maps, you can close the scalp map window.

This page titled 1.5: Exercise- Loading and Plotting ERPs is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 1.6: Exercise- Loading ERPs and Plotting ERP Waveforms by Steven J Luck is licensed CC BY 4.0.

1.6: Understanding the Matlab Path

Now that you've had a chance to play around with ERPLAB Studio a bit, I want to discuss a vitally important concept: the *PATH*. Many of the problems that Matlab beginners encounter are a result of not understanding and appropriately setting the PATH. If you're already quite familiar with the Matlab PATH, you can skip this section.

Simply put, the Matlab's PATH defines where it will look for code and other files. For example, when you type **estudio** into the Matlab command window to launch ERPLAB Studio, the only reason that Matlab knows where to find the eStudio code on your computer is that you (or someone) set the PATH to include the location of the files containing the eStudio code.

As you know, your computer's file system is divided into a set of folders (also called *directories*). The folders are hierarchically organized: You have a main folder for each drive, and those folders contain sub-folders, which contain sub-sub-folders, etc. The location of a file can be expressed concisely as something like **D:\books\ERP_Analysis_Book\Exercises\Chapter_2\1_N400_erp.erp** (on a Windows system) or **/Users/luck/Documents/books/ERP_Analysis_Book/Exercises/Chapter_2/1_N400_erp.erp** (on a Mac or Linux system). To make life complicated, DOS and Windows use backslashes to separate the names of the folders, whereas Mac and Linux use forward slashes.

The location of a single file, when expressed this way, is the *path* to the file. But note that I'm using lowercase to refer to the path for a single file. I use uppercase to refer to Matlab's PATH, which consists of a list of multiple paths. This list of paths defines where Matlab will look for code and other files. But note: Matlab will first look in the current folder (indicated by the blue arrow in Figure 1.2) before looking in the folders defined by the PATH.

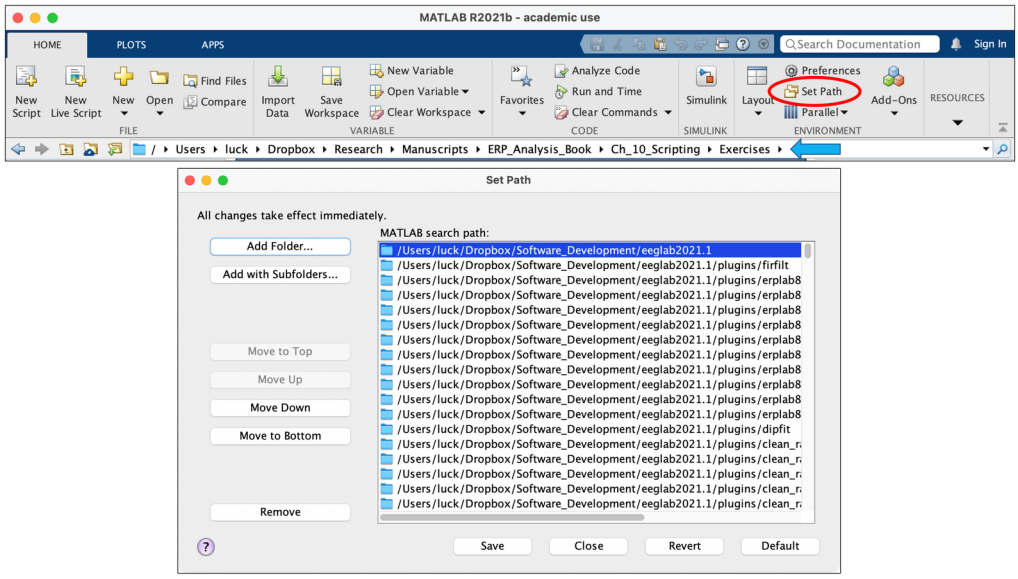


Figure 1.2. Setting the PATH in Matlab. To set the PATH, you click on the **Set Path** icon in the main Matlab window (circled in red here). This brings up the dialog box. Note that the current folder is indicated by the blue arrow. You can click on **Add Folder...** to add a new folder to the PATH. You can also remove items from the PATH using the **Remove** button. Make sure to click **Save** so that the PATH is saved for future Matlab session. Note that this might look different on your computer.

Let's take a look at the PATH on your computer. In the main Matlab window, click on the icon labeled **Set Path** (circled in red in Figure 1.2, but note that it might be somewhere else in your version of Matlab). This should cause a new window to appear, like the one in the middle of Figure 1.2.

You should see the main folder for EEGLAB in the PATH. If you recently installed EEGLAB, it should be at the top. If you're having trouble getting eStudio to launch, make sure you have the EEGLAB path in your PATH. To add the EEGLAB folder to your PATH, click on **Add with Subfolders...**, navigate to the location of the EEGLAB folder that you downloaded when you installed EEGLAB, and add it to the PATH. It's also possible that the path is incorrect (e.g., because the folder was moved after the PATH was set). If you already have EEGLAB in your PATH but you don't know how to verify that the path is correct, you can delete the existing EEGLAB path (including all the subfolders) using the **Remove** button and then add the path again using **Add with Subfolder...**

When you're done updating the PATH, click **Save** and then **Close**. If you click the **Close** button without saving, the updated PATH will work, but it will be forgotten when you quit from Matlab. You need to click **Save** before you click **Close** if you want Matlab to remember the path when you launch it again in the future.

Now let's talk in a little more detail about how the PATH works. A complicated package like EEGLAB or ERPLAB is divided into tons of individual files (often called **.m files** because the filename ends in **.m**). When you type **estudio** in the Matlab command window, Matlab searches the path for a file named **estudio.m**, and then it executes the code in this file. The code in the **estudio.m** file calls many other functions, such as **pop_basicfilter()**. The code for a given function is typically stored in a separate .m file. When Matlab needs to call **pop_basicfilter()**, it therefore looks for it in a file named **pop_basicfilter.m**. Where does Matlab look to find this file? In the folders defined by the PATH, of course. The **pop_basicfilter.m** file is actually located in a subfolder buried deep within the main EEGLAB folder. This is why there are actually many EEGLAB subfolders in your PATH.

When you add a new folder to the PATH, you will ordinarily click on **Add with Subfolders...** to make sure that all the subfolders within the new folder are in your path.

Here's the key takeaway from what we've discussed so far: Almost every command and function in Matlab is stored in a .m file, and when you try to execute a command or function, Matlab searches the PATH to find it. If Matlab tells you that it can't find a function or a .m file, this almost always means that you don't have the PATH set correctly.

Another problem that can arise is that there might be two different .m files with the same name in your path. For example, if you install a new version of EEGLAB on your computer, and you add the folder containing this version to your PATH without removing the old path, Matlab might execute the wrong version. So, make sure to clean out the old folder (and any subfolders) from the PATH using the **Remove** button when installing a new version.

It's also possible that you have another package (e.g., FieldTrip or PsychToolbox) in your PATH that defines a function with the same name as one of the EEGLAB functions. For example, both EEGLAB and FieldTrip could have a function named **PlotEEGWaveforms()**, which would be stored in a file named **PlotEEGWaveforms.m**. You would then have two identical files in your PATH. Which file will Matlab use when it tries to call the **PlotEEGWaveforms** function? The answer is simple: It searches the folder in the order that they're listed in the PATH, and it stops searching once it finds a match. But this might not be the function you want.

Yet another common problem is that the path to a folder may have one or more spaces, such as **D:\books\ERP Analysis Book\Exercises**. That's usually not a problem, but sometimes Matlab will interpret such a path as being three separate things (**D:\books\ERP** and **Analysis** and **Book\Exercises**). If this happens, you can usually solve the problem either by using underscores instead of spaces or by enclosing the path in single quotes (e.g., **'D:\books\ERP Analysis Book\Exercises'**).

If you want more information, Matlab's online documentation provides a detailed description of how the PATH works.

This page titled 1.6: Understanding the Matlab Path is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 1.9: Understanding the Matlab Path by Steven J Luck is licensed CC BY 4.0.

1.7: Key Takeaways and References

Key Takeaways

- This book focuses on applied ERP data analysis, using the ERPLAB Studio program as an interface into the open-source EEGLAB and ERPLAB analysis packages. The packages are free, but they require Matlab to run.
- Most of the analysis examples are from the ERP CORE, a set of 6 standardized ERP experiments.
- You will almost certainly run into technical problems, but a major goal of the book is for you to learn to solve these problems. Appendix 2 provides a Troubleshooting Guide that will teach you some useful principles for solving technical problems.
- Many problems and errors are a result of an incorrect PATH. If Matlab tells you that a function can't be found, you probably need to update the PATH.

References

Berger, H. (1929). Ueber das Elektrenkephalogramm des Menschen. *Archives Fur Psychiatrie Nervenkrankheiten*, *87*, 527–570.

Delorme, A., & Makeig, S. (2004). EEGLAB: an open source toolbox for analysis of single-trial EEG dynamics including independent component analysis. *Journal of Neuroscience Methods*, *134*, 9–21.

Lopez-Calderon, J., & Luck, S. J. (2014). ERPLAB: An open-source toolbox for the analysis of event-related potentials. *Frontiers in Human Neuroscience*, *8*, 213. https://doi.org/10.3389/fnhum.2014.00213

Luck, S. J. (2023). Event-related potentials. In H. Cooper, P. M. Camic, D. L. Long, A. T. Panter, D. Rindskopf, & K. J. Sher (Eds.), *APA Handbook of Research Methods in Psychology: Volume 1, Foundations, Planning, Measures, and Psychometrics* (2nd Edition, pp. 605–630). American Psychological Association. https://doi.org/10.1037/0000318-028

Luck, S. J. (2014). *An Introduction to the Event-Related Potential Technique, Second Edition*. MIT Press.

Rosenbaum, D. A., Vaughan, J., & Wyble, B. (2014). *MATLAB for Behavioral Scientists* (2nd Edition). Routledge.

This page titled 1.7: Key Takeaways and References is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 1.10: Key Takeaways and References by Steven J Luck is licensed CC BY 4.0.

2: PROCESSING THE DATA FROM ONE PARTICIPANT IN THE ERP CORE N400 EXPERIMENT

Learning Objectives

In this chapter, you will learn to:

- Load **EEG** data
- Create an **EventList**, which stores information about events such as stimuli and responses
- Assign events to bins with **BINLISTER**
  - A *bin* is a set of ERP waveforms, one for each channel, that were created by averaging together a set of trials
- Parse the continuous EEG into a series of discrete epochs (e.g., from -200 to +800 ms relative to stimulus onset) and perform baseline correction
- Perform simple artifact rejection
- Compute and plot averaged ERP waveforms
- Filter EEG and ERP data
- Make difference waves

This chapter goes through a simple example experiment so you can see the basic steps involved in processing the data from a single participant. The experiment is from the ERP CORE (Kappenman et al., 2021), and it is designed to isolate the N400 component that is elicited when a word is semantically unrelated to a previous word (e.g., SHOE when it's preceded by TREE instead of by SOCK).

In this chapter, we'll go through the most basic EEG processing steps (e.g., epoching, baseline correction, artifact detection) and create averaged ERPs for a single participant. In the next chapter, we'll process additional participants, measure the N400 amplitude, and do a simple statistical analysis. The details of these steps will be described in later chapters, along with important details about how Matlab and ERPLAB Studio work. The present chapter and the next chapter are designed for you to get the big picture of how the data are processed and learn the basics of using ERPLAB Studio. You can also find an overview of an entire EEG processing pipeline in Appendix 3.

If you're already experienced with ERPLAB Studio, you can just skim through this chapter. But if you don't have much ERPLAB Studio experience, you'll want to download the data and do each data processing exercise. Remember, you'll probably run into some error messages or other technical snags, but don't get discouraged. An important goal of this book is for you to learn how to troubleshoot technical problems. See the troubleshooting tips in Appendix 2 if you have problems.

2.1: Data For This Chapter

2.2: Design of the N400 Experiment

2.3: Exercise - Looking at the EEG and the Event Codes

2.4: Exercise- Filtering Out Low-Frequency Drifts from the EEG

2.5: Exercise- Creating an EventList

2.6: Exercise- Assigning Events to Bins with BINLISTER

2.7: Exercise- Epoching and Baseline Correction

2.8: Exercise- Artifact Detection

2.9: Exercise- Averaged ERPs

2.10: Exercise- Data Quality

2.11: Review of Processing Steps

2.12: A Simple Matlab Script

2.13: Key Takeaways and References

This page titled 2: Processing the Data from One Participant in the ERP CORE N400 Experiment is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 2: Processing the Data from One Participant in the ERP CORE N400 Experiment by Steven J Luck is licensed CC BY 4.0.

2.1: Data For This Chapter

This chapter, like every subsequent chapter in this book, is filled with exercises. Processing real data is the best way to learn ERP data analysis! Your first step in each chapter will therefore be to download the data for the exercises in that chapter.

The N400 data we'll be using for the exercises in this chapter can be found in the Chapter_2 folder in the master folder: https://doi.org/10.18115/D5G045. In this chapter, we'll be looking at the data from only one of the 40 participants (Subject #6). Go ahead and download this folder and all its contents (and make sure to name the folder **Chapter_2**).

I recommend organizing the data for the various chapters of the book on your computer just as they're organized online, with a separate folder for each chapter.

This page titled 2.1: Data For This Chapter is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 2.1: Data For This Chapter by Steven J Luck is licensed CC BY 4.0.

2.2: Design of the N400 Experiment

Before we look at the data, let's consider the design of the ERP CORE N400 experiment. As illustrated in Figure 2.1.A, the experiment involved a sequence of trials, each of which consisted of a *prime* word followed by a *target* word. The participants' task was to press one of two buttons on each trial to indicate whether the target was semantically related or semantically unrelated to the preceding prime word. For example, they would press the *related* button for CHAIR preceded by TABLE and the *unrelated* button for SPIDER preceded by RAKE. Additional methodological details can be found in Kappenman et al. (2021).

Hundreds of previous studies have shown that a word will elicit a larger N400 component if it is unrelated to a previously established semantic context than if it is related to that context (Kutas, 1997; Swaab et al., 2012), so we expected to see a larger N400 on unrelated trials than on related trials. Figure 2.1.B shows the ERP waveforms, recorded at the CPz electrode site (where the N400 is typically largest) and averaged over all 40 of the original participants (a *grand average*). In these waveforms, the N400 is a negative-going wave for the unrelated targets that is present from approximately 200-600 ms and is added onto the positive voltage that is ordinarily present during this time period.

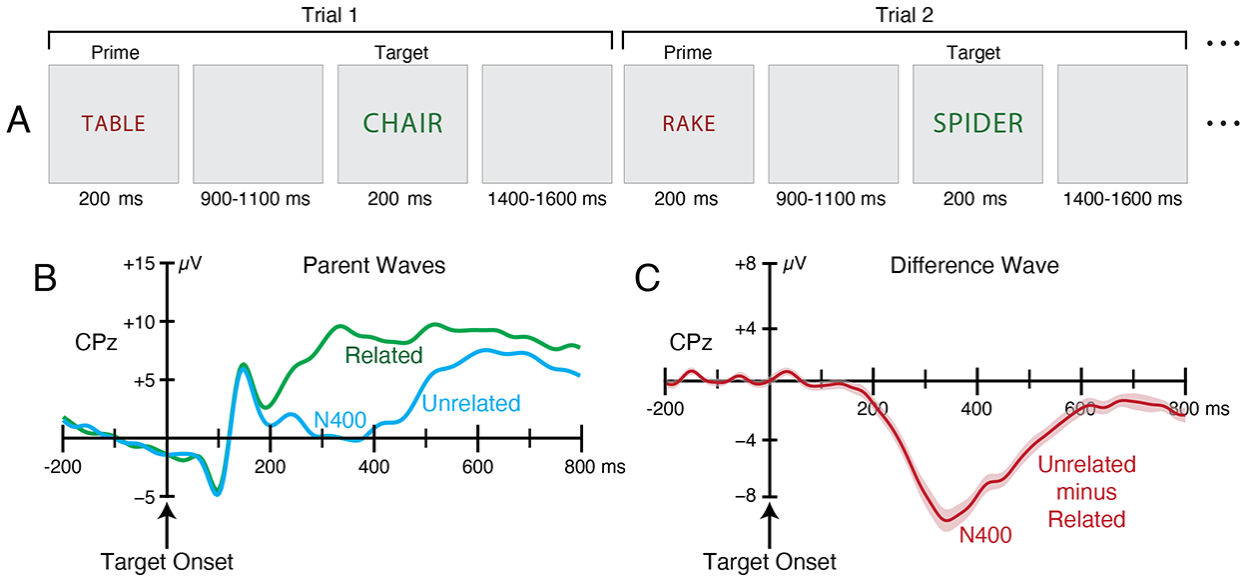


Figure 2.1. Experimental paradigm and results from the ERP CORE N400 experiment. (A) On each trial, subjects saw a prime word drawn in red and a target word drawn in green. The task was to press one of two buttons following each target word to indicate whether the target was semantically related or semantically unrelated to the preceding prime word. (B) ERP waveforms elicited by the target words, averaged over all 40 participants in the original study. (C) Difference wave created by subtracting the waveform on the related trials from the waveform on the unrelated trials (i.e., unrelated minus related).

It is often useful to subtract away everything that is in common to two conditions and focus on the difference in brain activity between conditions. To do this, we compute a *difference wave*, which is simply the difference in voltage between the two conditions at each time point. Figure 2.1.C shows the unrelated-minus-related difference wave for our N400 experiment. It allows us to see the brain's differential processing of unrelated versus related words. Because the difference deviates from zero at approximately 200 ms, we can conclude that the brain has determined whether the target is related or unrelated to the prime by this time.

This page titled 2.2: Design of the N400 Experiment is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 2.2: Design of the N400 Experiment by Steven J Luck is licensed CC BY 4.0.

2.3: Exercise - Looking at the EEG and the Event Codes

In our first exercise in this chapter, we'll take a look at the EEG and the event codes for one participant (Participant #6). You should always scan through a participant's EEG and event codes before starting to process the data. Many things can go wrong during an EEG recording, and you want to make sure that there aren't any problems that you need to address before you go further. I provide a detailed example of how to visually inspect a participant's EEG in the chapter on artifact rejection (Chapter 7).

Remember, event codes are stored in the EEG file during data collection and indicate what event happened (e.g., which particular stimulus or response) and the time of occurrence (see Figure A1.1 in Appendix 1). In the system we used for collecting the ERP CORE data, event codes were integers between 1 and 255 (see the box below for more information about event codes). Table 1 lists the event codes for the N400 experiment. An event code was produced for each prime word and each target word, and different codes were used depending on whether the target was semantically related or unrelated to the prime on that trial. Note that we used two different lists of words (for counterbalancing purposes), and that was also indicated by the event codes. Each subject saw a total of 120 trials, 60 on which the target word was related to the prime word and 60 in which the target and prime were unrelated. Thus, there were 30 occurrences of each stimulus event code (30 related and 30 unrelated for each of the two lists).

Table 2.1. Event codes for the ERP CORE N400 experiment.

	Word Type	Relatedness	Word List	Event Code	Occurrences
Stimuli	Prime	Related	List 1	111	30
	Prime	Related	List 2	112	30
	Prime	Unrelated	List 1	121	30
	Prime	Unrelated	List 2	122	30
	Target	Related	List 1	211	30
	Target	Related	List 2	212	30
	Target	Unrelated	List 1	221	30
	Target	Unrelated	List 2	222	30
Responses	Correct			201	Variable

	Word Type	Relatedness	Word List	Event Code	Occurrences
	Incorrect			202	Variable

Event Codes

In most EEG systems, one computer is used for presenting stimuli, and a different computer is used for recording the EEG. In addition, the experimenter often uses general-purpose stimulus presentation software that is not made by the manufacturer of the EEG recording system. To ensure compatibility, the method typically used to pass event codes from the stimulus presentation computer to the EEG recording computer is based on a communications protocol that has been around since 1970 (the Centronics-style parallel port). The protocol requires that the event codes be whole numbers between 1 and 255. That's plenty for some experiments, but it's woefully inadequate for others. In our N400 experiment, for example, it would have been nice for each event code to indicate the actual word that was presented. The ERP CORE online resource therefore provides text files (in CSV format) with the actual words used for each participant.

Some EEG systems come with their own stimulus presentation software, and they use custom protocols to allow for richer event codes (which might be text instead of 8-bit integers). However, this is not a very general solution.

Some stimulus presentation systems and EEG recording systems now allow you to use a newer standardized protocol called *Lab Streaming Layer*. This protocol makes it possible to use an Ethernet cable or even WiFi so that you can send much richer event codes. I hope this becomes the standard approach, because rich event information makes it easier to perform sophisticated ERP analyses. After all, "event" is the first word in "event-related potential".

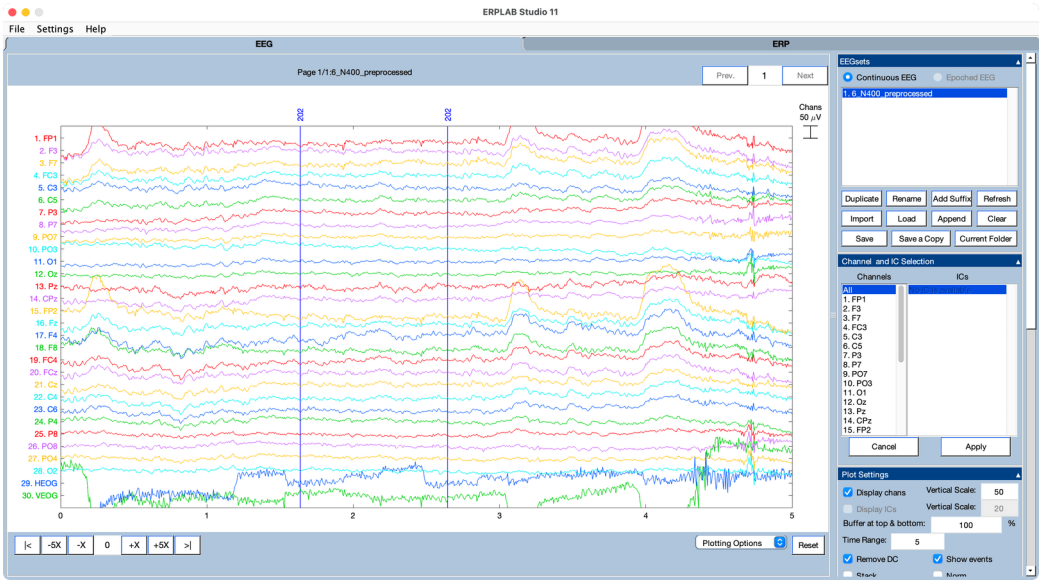
In the exercises for this chapter, we're going to look at the data from Subject 6, who has particularly nice data.

Go to the Matlab command window and start ERPLAB Studio (by typing **estudio** in the command window). To make sure that you get the same results that will be shown in the screenshots, reset eStudio's settings by clicking the **Reset** button under the lower right corner of the plotting region, check the boxes for restoring the default parameters for both the EEG panels and the ERP panels, and click **Continue**. It's a good idea to do this whenever eStudio is not behaving the way you expect.

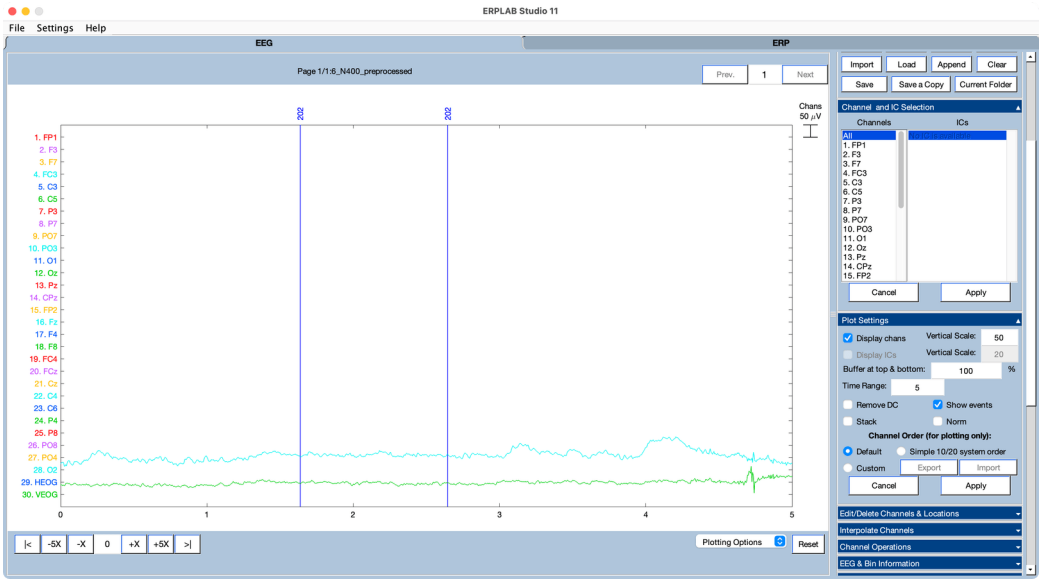
In Matlab, set the current folder to be the **Chapter_2** folder. On the left side of the Matlab window, you should see the contents of this folder, including a file named **6_N400_preprocessed.set**. That file contains the EEG from Subject 6. A few minor preprocessing steps have already been conducted to make the exercises in this chapter a little easier.

Make sure that the **EEG** tab is open in eStudio, click the **Load** button in the **EEGsets** panel, and select the file **6_N400_preprocessed.set**. The plotting region should look like Screenshot 2. 1.

Screenshot 2.1



In the **Plot Settings** panel, you can see that Remove DC is selected. Uncheck the box for this option and click the **Apply** button at the bottom of the panel. As shown in Screenshot 2.2, the waveforms are now visible for only two of the channels. Why do you think that is?



Screenshot 2.2

The answer is that the voltage in each channel contains a DC voltage offset that arises mainly from voltages generated by the skin (see Chapter 5 in Luck, 2014), and this shifts the voltages out of the range of the plotting display for most of the channels. That is, the voltage recorded in our EEG electrodes is the sum of the EEG plus any voltage offset, and the voltage offsets are often so large that they make the signal go beyond the range of values shown in the plot. The **Remove DC** option subtracts the mean voltage for each channel during the time period being plotted so you can easily see all the channels. You will almost always want to do this, so this option is on by default. Turn the option back on and click **Apply** so that the waveforms look like Screenshot 2.1 again.

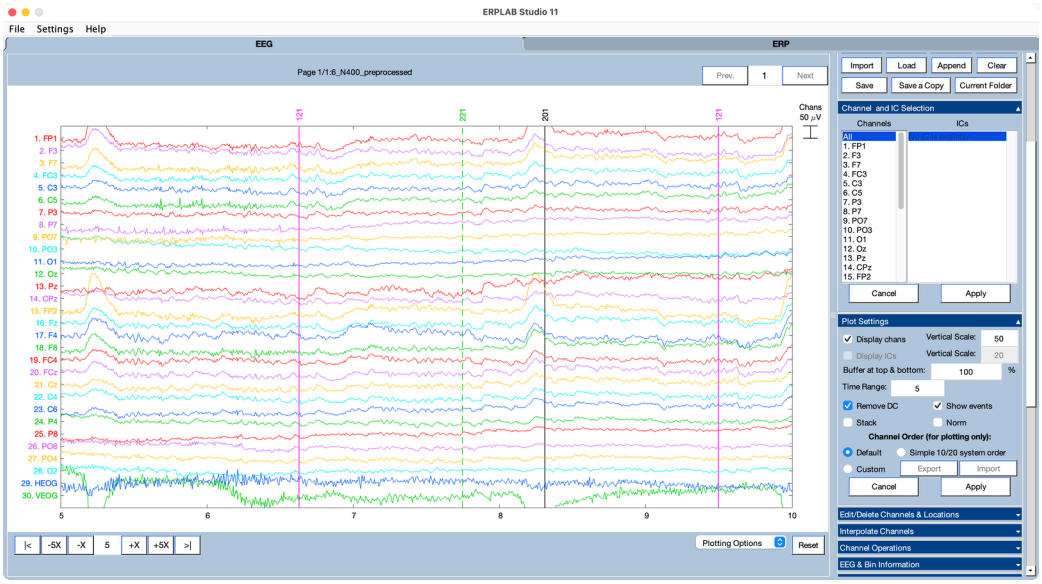
Now let's look at the event codes in the EEG plot. Do you see the two vertical lines with a label of **202** at the top of each line? Those are event codes. If you look at Table 2.1, you'll see that event code 202 corresponds to an incorrect behavioral response. In this task, there are some instruction screens at the beginning, and the participants are required to press a response button to go to the next screen. Those responses generated an event code 202.

Pro tip: Starting the recording several seconds before the first trial

You might expect that the EEG recording would begin only moments before the first trial. Why waste disk space with all that extra EEG? However, there is a technical reason why you should start the recording several seconds before the first trial. Specifically, this can help you avoid artifacts at the beginning and end of the waveform that are often produced when you filter the data. If you have some "extra" EEG at the beginning and end of the recording, the filter artifacts occur during these unused time periods rather than distorting the data during the first and last trials of the recording. See Chapter 7 in Luck (2014) for a more detailed explanation.

Click the **+X** button near the bottom of the plotting window to scroll forward 5 seconds in time. As shown in Screenshot 2.3, you should now see an event code 121 at approximately 6.6 seconds, an event code 221 at approximately 7.7 seconds, and event code 201 at approximately 8.3 seconds. Look up these event codes in Table 2.1. What happened at these three time points?

Event code 121 was a prime word that began approximately 6.6 seconds after the start of the recording. Event code 221 was a target word that was unrelated to the prime word, and it began approximately 1.1 seconds after the prime word. Event code 201 was a correct response, and it occurred approximately 0.6 seconds after the target word. So, this was the first trial in the experiment, and the subject correctly determined that the target was unrelated to the prime with a response time of approximately 600 ms. Later, we'll discuss how you can determine these times more precisely.



Screenshot 2.3

If you look near the right edge of the plotting region, you'll see the event code for the prime word on the second trial. You can type **9** (followed by Enter/Return) in the box between the **-X** and **+X** buttons so that the plot starts at 9 seconds, and then you'll be able to see the event codes for the prime, the target, and the response on this second trial. Here are some questions you should try to answer:

- Given the experimental design shown in Figure 2.1, what is the shortest amount of time you should ever see between the prime word and the target word? What is the longest time?
- Similarly, what are the longest and shortest times between the target word on one trial and the prime word on the next trial?
- What was the response time for the second trial (approximately)?

If possible, keep eStudio in the same state for the next exercise.

This page titled 2.3: Exercise - Looking at the EEG and the Event Codes is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

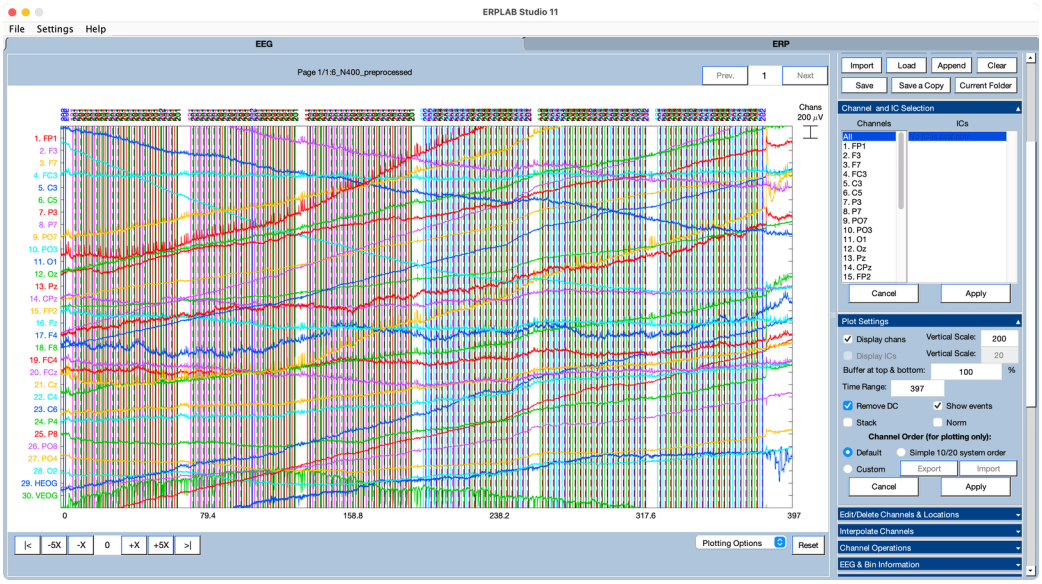
- 2.3: Exercise - Looking at the EEG and the Event Codes by Steven J Luck is licensed CC BY 4.0.

2.4: Exercise- Filtering Out Low-Frequency Drifts from the EEG

The skin is an electrical organ, and it produces slow drifts in voltage that are picked up by our EEG electrodes. These skin potentials can cause the voltage to gradually drift upward or downward by hundreds of microvolts over a period of a few minutes. To get a better look at these voltage drifts:

- Go to time zero by putting a zero in the text box between the **-X** and **+X** buttons under the plotting region (and hit Enter/Return)
- Set the Vertical Scale to **200** and the Time Range to **400** seconds in the **Plot Settings** panel (and click **Apply**)
    - Note that this EEGset is 397 seconds long, and eStudio will automatically adjust the Time Range from **400** to **397**

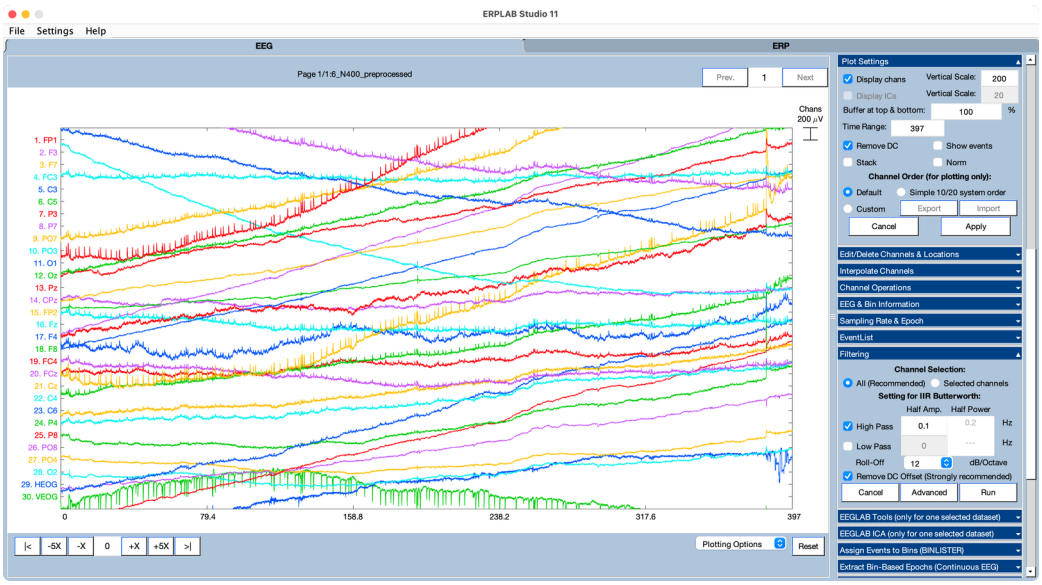
The result should look like Screenshot 2.4. Looks pretty weird, doesn't it? The first thing you should look at is the event codes (the vertical lines). The N400 experiment lasted about 6 minutes, and you're looking at the entire recording, so there are lots of event codes. Notice that there are 6 clusters of event codes, separated by gaps of approximately 7 seconds. Participants can maintain their attention better if the experiment is divided into a large number of short blocks, each followed by a brief break, so this experiment was broken up into short blocks that lasted only about a minute each. Specifically, the 120 trials in this experiment were divided into 6 blocks of 20 trials each, with a short rest break after each block.



Screenshot 2.4

The event codes make it hard to see the EEG on this time scale, so disable them by unchecking **Show events** in the **Plot Settings** panel (and click **Apply**).

Now take a look at the EEG waveforms. You can now see that the voltage is gradually drifting over time. It drifts upward in some channels and downward in others. Most of the channels change by well over 100 μV over the course of this 400-second period. These drifts are mainly caused by electrical potentials in the skin that are picked up by the EEG electrodes (see Chapter 5 in Luck, 2014 for more details).



Screenshot 2.5

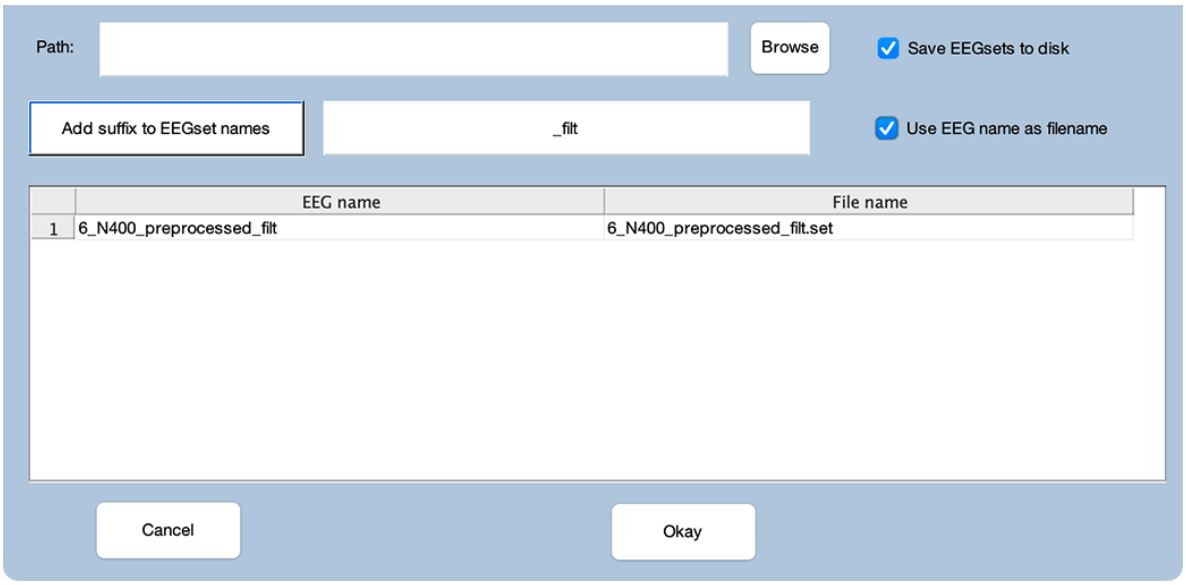
Filtering Out the Low-Frequency Drifts with a High-Pass Filter

These drifts can make it difficult to obtain reliable ERP differences across conditions, and it's usually a good idea to filter them out with a *high-pass filter*, which filters out low frequencies and passes high frequencies. To accomplish this, go to the **Filtering** panel and set it up as shown in Screenshot 2.5. This will apply a high-pass filter with a half-amplitude cutoff at 0.1 Hz and a roll-off of 12 dB/octave, which is a good general-purpose filter for most studies of cognitive and affective processes. If you don't know what these parameters mean, don't worry – we'll cover them in Chapter 4. You can also find a broad conceptual overview of filters in Chapter 7 of Luck (2014) and a more detailed mathematical treatment in Chapter 12 of Luck (2014). Zhang et al. (2024) provide a list of optimal filter settings for each of the 7 ERP components in the ERP CORE.

Click the **Run** button in the **Filtering** panel to run the filter and save the result in a new EEGset. You should see a new window like the one shown in Screenshot 2.6, which will allow you to name the new EEGset. In EEGLAB and ERPLAB Studio, most operations that modify an EEGset will actually create a new EEGset. That way, if you make a mistake or change your mind, you can easily go back to the previous EEGset. The EEGsets are stored in memory, where they're listed in the **EEGsets** panel. You can also save them to your hard drive if you want.

ERPLAB Studio will suggest a name for each new EEGset that you create, typically adding a suffix to the end of the previous EEGset name. As you can see in Screenshot 2.6, it suggests adding a suffix of **_filt** to the previous EEGset name, resulting in a new name of **6_N400_preprocessed_filt**. You can use any name you like, but it's quick and easy to use eStudio's suggested name, and that way you will automatically have a consistent naming scheme.

We're going to need the filtered EEGset for a later exercise, so you should save it to disk by checking the **Save EEGsets to disk** box. The name of the EEGset in memory doesn't have to be the same as the filename, but it can be confusing if the name in memory is different from the filename. You can automatically make the names identical by clicking the **Use EEG name as filename** box.



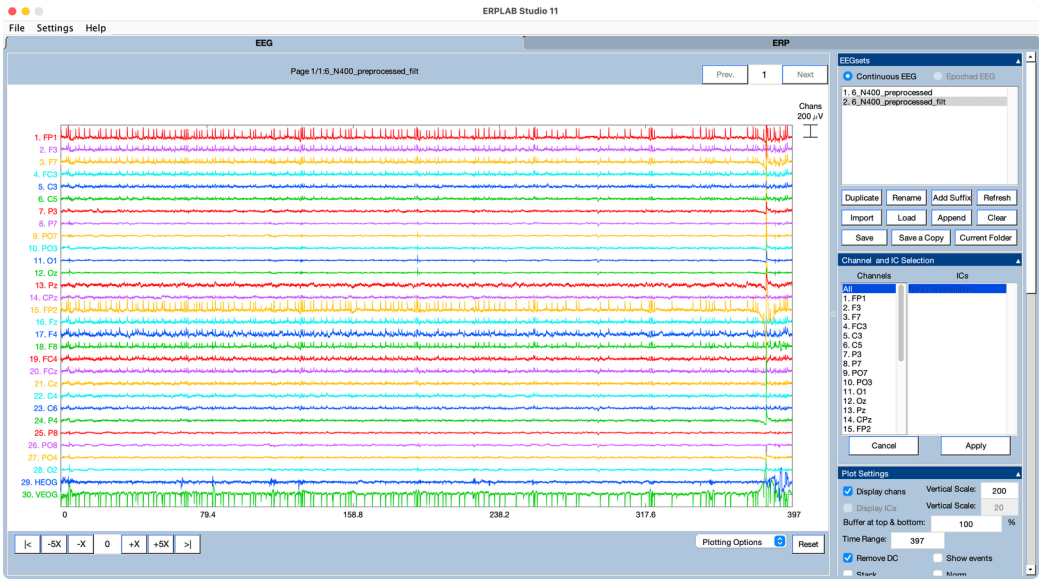
Screenshot 2.6

Once you have everything set up as shown in Screenshot 2.6, click **Okay**. You've now created a new EEGset with the filtered data and saved it to the current folder of your disk. If you look in the **EEGsets** panel, you should see both the original EEGset (**6_N400_preprocessed**) and the new EEGset (**6_N400_preprocessed_filt**), with the new one selected. And the plotting region will show the filtered EEGset, as shown in Screenshot 2.7.

Getting an Error Message?

Did you get an error message when you launched the filtering tool? If so, the message probably said that you're missing the Signal Processing Toolbox. This toolbox comes from the makers of Matlab and is required for certain ERPLAB processes, such as filtering. Depending on your institution's Matlab license, it may be free or it may require an extra fee.

You can see what toolboxes are installed by typing **ver** on the Matlab command line. If you don't have the Signal Processing Toolbox and you don't know how to get it and/or install it, contact your institution's IT support department for assistance.



Screenshot 2.7

Looking at the Filtered Data

As you can see in Screenshot 2.7, the slow drifts are now gone, and the data look much more orderly. We've now gotten rid of a major source of artifactual activity from the EEG, which improves our ability to obtain robust, reliable ERP effects.

Now change the time period to display to be 10 seconds instead of 400 seconds in the **Time Range** box in the **Plot Settings** panel. Because you're still removing the DC offset and the drifts are slow, the filtered and unfiltered data don't look as radically different with this 10-second time scale as they did with the 400-second time scale. But if you go back and forth between selecting **6_N400_preprocessed** and **6_N400_preprocessed_filt** in the **EEGsets** panel, you'll see that there is some drift in the unfiltered data that is absent in the filtered data (especially in Channel 10, which is the PO3 channel). You can also see that the faster deflections in the data are largely identical in the filtered and unfiltered EEGsets. So, the high-pass filter has largely eliminated the slow drifts but has had minimal effect on the other features of the EEG. We'll take a closer look at filters in Chapter 4.

You'll need the filtered dataset for the next exercise, so leave eStudio in its current state if possible. Also, you should reset the plotting parameters back to their defaults, which you can do via the **Reset** button under the lower right corner of the plotting regions.

This page titled 2.4: Exercise- Filtering Out Low-Frequency Drifts from the EEG is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 2.4: Exercise- Filtering Out Low-Frequency Drifts from the EEG by Steven J Luck is licensed CC BY 4.0.

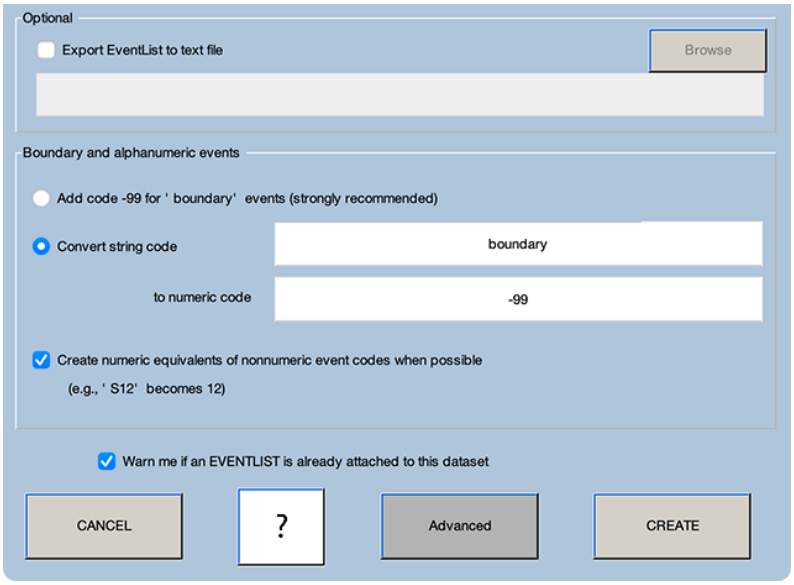
2.5: Exercise- Creating an EventList

Our next step is to add something called an *EventList* to the current EEGset. An EventList is a simple, easy-to-access list of all the event codes in a dataset. EEGLAB stores event codes in a Matlab data structure that's a little bit difficult for beginners to access, so we added the EventList structure when we created ERPLAB. As I mentioned earlier, "event" is the first part of the term "event-related potential", so we want to make it easy to see and manipulate the event codes.

If you don't still have the filtered EEGset from the previous exercise loaded in eStudio, go ahead and load it now (6_N400_preprocessed_filt). If you don't have the file, you can load it from the Extra_Files folder inside the Chapter_2 folder.

To create an EventList for this EEGset, go to the **EventList** panel in eStudio and click **Create**. Make sure that the window that appears is set up as shown in Screenshot 2.8, and click **CREATE**.

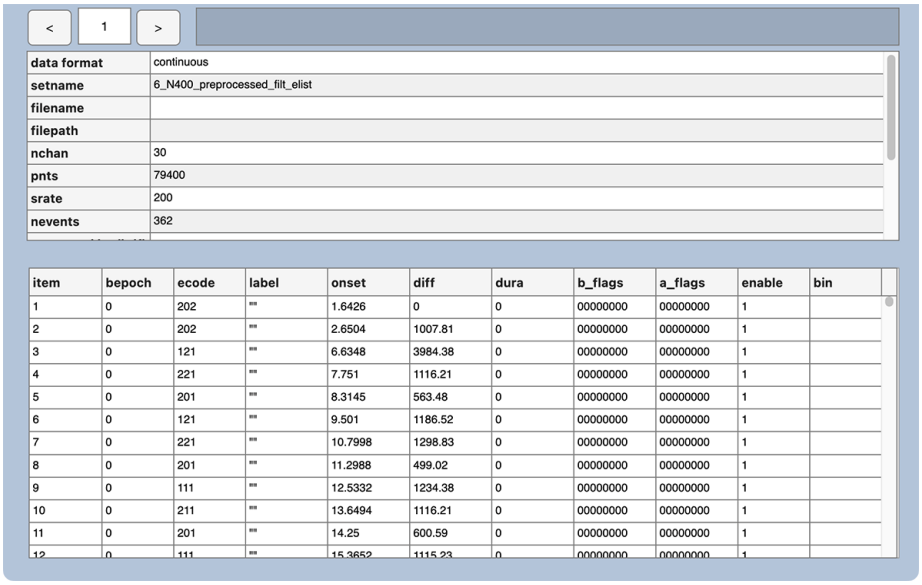
Screenshot 2.8



You'll then see a new window that allows you to name and save the new EEGset. The starting dataset was named **6_N400_preprocessed_filt**, and the new one should be named **6_N400_preprocessed_filt_elist**. You can save it to disk if you like. Click **Okay** to continue. You should now see **6_N400_preprocessed_filt_elist** in the **EEGsets** panel.

Now let's take a look at the EventList. Go to the **EventList** panel and click **View**. A new window will appear and should look like Screenshot 2.9.

Screenshot 2.9



At the top of the window, you'll see some general information about the EEG recording, such as the number of channels (**nchan**) and the sampling rate (**srate**). Below that, you'll see a table listing the event codes, with one per line. Each line contains several columns, which we will discuss later. For now, the key columns are **item** (which indicates the ordering of the event codes), **ecode** (which is the actual event code), **onset** (which is the time of the event code relative to the beginning of the recording), and **diff** (which is the amount of time between the current event code and the previous event code, in milliseconds rather than seconds). In principle, event codes can have a duration (listed in the **dura** column), but this feature is not used by ERPLAB, and event codes are typically considered to be instantaneous (i.e., a duration of 0).

The first two items are the button-press responses (code 202) that occurred at the beginning of the session. Items 3-5 are the event codes from the first trial, with a prime word (code 121), a target word (code 221), and a response (code 201). By looking at the difference in time between items 4 and 5 (the **diff**) column, you can see that the behavioral response on this trial was 563.48 ms after the preceding target word.

You can export the EventList as a text file (with the **Export .txt** button) or a spreadsheet (with the **Export .xls** button). You can then edit the file and import it back into the EEGset (with the **Import .txt** and **Import .xls** buttons). For example, imagine that the subject was confused about the task between items 120 and 160. You could set the **enable** column to 0 for these items, which would cause them to be ignored by all ERPLAB processes. Similarly, if you discovered that event code 21 was used instead of event code 121 in one participant's data, you could export the EventList as a .xls file, open the file with Excel and do a global search-and-replace to replace all occurrences of 21 with 121, and then import the .xls file. You can even add or delete event codes in this manner. [If you know how to program in Matlab, you could instead write a script that directly modifies the EventList structure rather than exporting, modifying, and importing the EventList.]

You can now close the window showing the EventList.

Our last step will be to verify that we have the current number of occurrences of each event code. As shown in Table 2.1, there should have been 30 occurrences of each of the stimulus event codes. The number of occurrences of each response event code varies across subjects depending on the number of errors. However, there were 120 trials, so there should be 120 total response event codes (plus the two at the beginning during the instruction phase).

You can see the number of occurrences of each event code in a table at the bottom of the **EventList** panel. Do we have the correct number of occurrences of each event code type? I recommend checking this for every participant you run. You can catch a lot of errors by making sure that you have exactly the right number of occurrences of each event code (see the box below).

Pro tip: Checking the number of event codes for every participant

In my experience, the most common error within ERP datasets is that there is some kind of problem with the event codes. If you are designing and running your own experiments, perhaps the single most important thing you can do to avoid trouble later is to make sure that your event codes are correct. A given experiment might have 1000+ event codes, so it is difficult to check every single one. However, if you set up your experiment properly, you can at least make sure that you have the right number of occurrences of each event code (for the stimuli).

The first step is to know exactly how many occurrences there should be for each code. And I do mean "exactly" rather than "approximately." You can have major problems with your event codes and end up with approximately the right number. Once you know exactly how many occurrences you should have, you can simply run your stimulus presentation program while recording the EEG but without a participant. You will then import your data into eStudio, add the EventList, and check the number of occurrences of each event code in the table at the bottom of the **EventList** panel. You should also "spot check" a few dozen randomly selected event codes to make sure that they are correct with respect to the stimulus that was actually presented.

You should also check the number of occurrences of each event code for every participant you run (immediately after the recording session). Lots of things can go wrong with event codes, and this will allow you to catch a problem before you've wasted weeks of time collecting data that turn out to be useless. Checking the number of each event code is easy, and **you absolutely must do it** to avoid problems.

This page titled 2.5: Exercise- Creating an EventList is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 2.5: Exercise- Creating an EventList by Steven J Luck is licensed CC BY 4.0.

2.6: Exercise- Assigning Events to Bins with BINLISTER

Bins and ERPsets

When we create averaged ERPs, we often want to combine trials with different event codes into the same average. For example, the ERP CORE N400 experiment used separate event codes to indicate which of the two word lists was used for a given trial, but we didn't need to make this distinction in our data analysis, so we just averaged together the trials for the different word lists. Similarly, some event codes are not used for creating averages. In the N400 experiment, for example, we created stimulus-locked averages (i.e., averages in which the prime or target stimulus is time zero) but not response-locked averages (i.e., averages in which the response is time zero), so we did not use the response event codes as time-locking events (although we did use them to exclude trials with incorrect responses from our stimulus-locked averages). Thus, we need a sophisticated way of indicating which events should be combined together when we make our averages.

We call this process *assigning events to bins*, and ERPLAB accomplishes this with a routine called *BINLISTER*. A *bin* is a set of averaged ERP waveforms—one for each electrode site—that were created by averaging together a specific set of trials. In our N400 experiment, for example, we will create four bins:

Bin 1: Primes followed by a related target

Bin 2: Primes followed by an unrelated target

Bin 3: Targets preceded by a related prime and followed by a correct response

Bin 4: Targets preceded by an unrelated prime and followed by a correct response

We recorded from 30 channels in the N400 experiment, so each bin had 30 averaged ERP waveforms, one for each channel. For all 30 channels in a given bin, we averaged together the same set of trials. The bins for a given participant are stored together, along with some general *header* information, in an *ERPset*. Figure 2.2 shows how the ERPset will be structured for the averages we will be creating in this chapter.

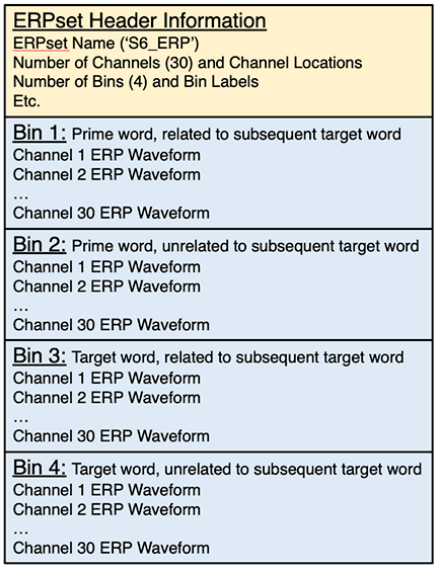


Figure 2.2. Structure of the ERPset that we will create in these exercises.

To create an ERPset, we need to assign events to the appropriate bins. This is done by updating the EventList. Each line of the EventList contains the information for one event code, and there is a **bin** field for each event code. Screenshot 2.11 below shows what we're trying to achieve, with a bin number listed in the **bin** field for each event (except the events that are not being assigned to bins, such as response events).

When you first create an EventList, the **bin** field is empty. In theory, you could manually enter the bin number for each event in a given participant's EventList, but that would take a very, very long time. You could also write a Matlab script that reads the event codes and figures out what bin each one should be assigned to. But in 99% of cases, you'll just use our BINLISTER tool to assign each relevant event to a bin.

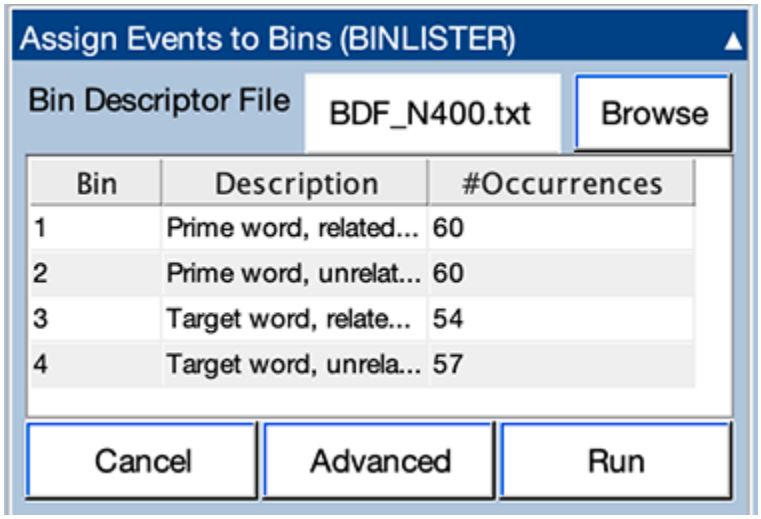
Running BINLISTER

Let's see how BINLISTER is used to assign events to bins for our example subject. We'll start with the dataset that you created in the previous exercise that has the EventList attached (**6_N400_preprocessed_filt_elist**). Make sure it is the active EEGset in the **EEGsets** panel. If it is not still listed in the **EEGsets** panel, you can load the version you saved to disk (or the version inside the Extra_Files folder).

Open the **Assign Events to Bins (BINLISTER)** panel and enter **BDF_N400.txt** in the Bin Descriptor File text box. **BDF_N400.txt** is a text file that tells BINLISTER how to assign events to bins for this particular experiment. We'll take a look at this file later.

Now click **RUN**, accept the default name for the new EEGset that will be created (6_N400_preprocessed_filt_elist_bins), and indicate that you want to save the result to disk.

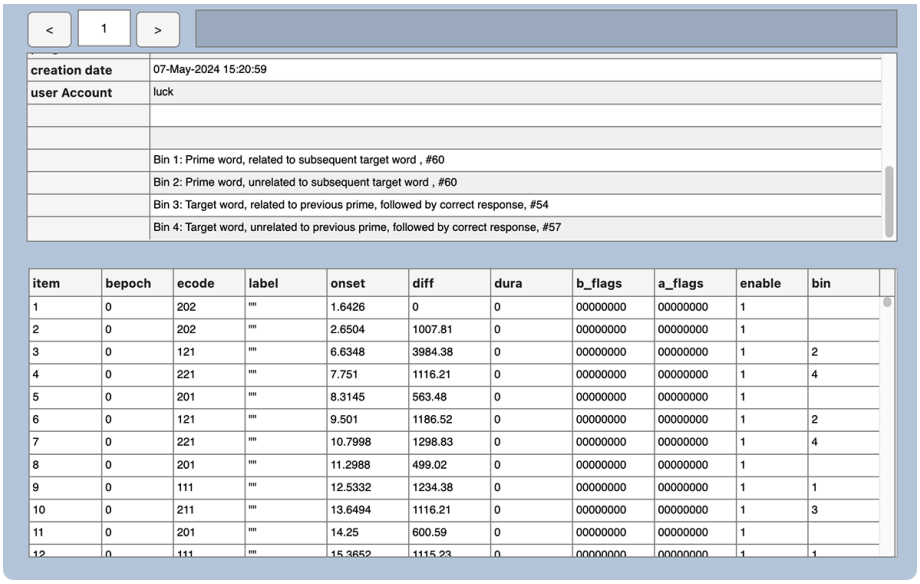
Once BINLISTER has finished running, you will see a table in the **Assign Events to Bins** panel showing the number of trials that were found for each bin (see Screenshot 2.10). For example, it shows that there were 60 trials each in Bins 1 and 2 (related and unrelated prime words). This makes sense because there were 60 related prime words and 60 unrelated prime words. There were only 54 trials in Bin 3 (related targets) and only 57 trials in Bin 4 (unrelated targets). This is because targets were included only if the behavioral response was correct, and the participant made 6 errors for related targets and 3 errors for unrelated targets. [There was no response to the prime, so those bins did not require a correct response.]



Screenshot 2.10

Looking at the Bin Assignments in the EventList

Now let's look at the bin assignments for the individual event codes, which are stored in the EventList. Go to the **EventList** panel and click **View**. If you scroll down to the bottom of the table at the top of the window, you'll see a list of the bins that were created (see Screenshot 2.11). For example, you can see the name of Bin 1 ("Prime word, related to subsequent target word) and the number of occurrences of this bin ("#60").



creation date	07-May-2024 15:20:59
user Account	luck
	Bin 1: Prime word, related to subsequent target word , #60
	Bin 2: Prime word, unrelated to subsequent target word , #60
	Bin 3: Target word, related to previous prime, followed by correct response, #54
	Bin 4: Target word, unrelated to previous prime, followed by correct response, #57

item	bepoch	ecode	label	onset	diff	dura	b_flags	a_flags	enable	bin
1	0	202	""	1.6426	0	0	00000000	00000000	1	
2	0	202	""	2.6504	1007.81	0	00000000	00000000	1	
3	0	121	""	6.6348	3984.38	0	00000000	00000000	1	2
4	0	221	""	7.751	1116.21	0	00000000	00000000	1	4
5	0	201	""	8.3145	563.48	0	00000000	00000000	1	
6	0	121	""	9.501	1186.52	0	00000000	00000000	1	2
7	0	221	""	10.7998	1298.83	0	00000000	00000000	1	4
8	0	201	""	11.2988	499.02	0	00000000	00000000	1	
9	0	111	""	12.5332	1234.38	0	00000000	00000000	1	1
10	0	211	""	13.6494	1116.21	0	00000000	00000000	1	3
11	0	201	""	14.25	600.59	0	00000000	00000000	1	
12	0	111	""	15.3652	1115.23	0	00000000	00000000	1	1

Screenshot 2.11

In the table of events at the bottom of the window, you can see that there are now numbers listed in the **bin** column for some items. These numbers indicate which bin (if any) a given event code has been assigned to. When an event is assigned to a bin, this means that the event is the *time-locking event* for that trial. In other words, the time of the event code will be time zero in our averaged ERP waveforms.

Consider, for example, item #3 in the EventList. It has an event code of 121, which means that it is a prime word that will be followed by an unrelated target word, with the word taken from the first word list (see the list of event code meanings in Table 2.1 of Section 2.3). This event was assigned to Bin 2 in the **bin** column. When we make an averaged ERP waveform for Bin 2, we'll include this event in that averaged ERP waveform, with this event code as time zero in the waveform. Now look at item #4 in the EventList. It has an event code of 221, and it's the target word that followed the prime word from item #3. It was assigned to Bin 4. The response event codes (e.g., items 1, 2, and 5) were not assigned to a bin because we are not making response-locked averages for this experiment.

How BINLISTER Works

How did BINLISTER know what bins to create and which events should be assigned to each bin? To accomplish this, BINLISTER used a set of abstract bin descriptions that are stored in the **BDF_N400.txt** file that you specified in the **Assign Events to Bins** panel (see Screenshot 2.10). The *BDF* in the filename stands for *bin descriptor file*, because it contains abstract descriptors for each bin. Here's what is in that file:

Bin 1
Prime word, related to subsequent target word
.{111;112}

Bin 2
Prime word, unrelated to subsequent target word
.{121;122}

Bin 3
Target word, related to previous prime, followed by correct response
.{211;212}{t<200-1500>201}

Bin 4
Target word, unrelated to previous prime, followed by correct response
.{221;222}{t<200-1500>201}

The details of the bin descriptor file syntax will be described in a later section, but some aspects of the syntax are fairly obvious here. Each bin is described with three lines. The first line just gives the bin number. The second number is a text string that describes the contents of the bin (and can be whatever you want). The third line is the actual *bin descriptor*. Each bin descriptor contains one or more event codes inside curly brackets. For example, Bin 2 contains **{121;122}**, indicating that event codes 121 or 122 should be used as the time-locking event for this bin. Bins 3 and 4 contain a second set of curly brackets that indicate what must follow the time-locking event code. The text **{t<200-1500>201}** means that the time-locking event code must be followed by an event code 201 with a delay between 200 and 1500 ms. For example, an event code of 221 will be assigned to Bin 4 if and only if it is followed by a correct response (event code 201) within 200 to 1500 ms. We specify this delay range because responses that are faster than 200 ms are probably bogus (because no one can respond that quickly), and responses that are later than 1500 ms probably indicate that the participant was zoning out.

BINLISTER reads the bin descriptor file and then goes through all of the event codes in the EventList. When it finds an event code that matches the bin descriptor for a given bin, it puts that bin number into the **bin** column for that event code. Note that a given event code can be assigned to more than one bin. For example, we could have one bin for target words that are followed by a correct response and a separate bin for target words irrespective of whether the response was correct.

Now that we've discussed how the four bins were defined in this experiment, you should go through the table of events in the EventList and make sure that you understand why each event was assigned to a given bin. In particular, you should find the target words that were followed by incorrect responses (event code 202) and verify that they were not assigned to a bin. Once you are done, you should close the window with the EventList table.

This page titled 2.6: Exercise- Assigning Events to Bins with BINLISTER is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 2.6: Exercise- Assigning Events to Bins with BINLISTER by Steven J Luck is licensed CC BY 4.0.

2.7: Exercise- Epoching and Baseline Correction

The next step in processing Subject 6's data is quite simple, and it involves two operations that are performed together in a single step. One of these operations is extracting a fixed-length *epoch* (time segment) for each event that has been assigned to a bin. If you're not quite sure what this means, see Figure A1.3 in Appendix 1. As shown in Figure 2.1 in Section 2.2, we will be looking at the data from 200 ms prior to stimulus onset through 800 ms after stimulus onset. Consequently, for each trial we need to extract an epoch that starts 200 ms before the time-locking event code and extends until 800 ms after the event code.

The second operation will be *baseline correction*, which is a way of dealing with the DC voltage offsets. As shown in Figure 2.3, this exercise uses the average voltage during the prestimulus period as an estimate of the offset, and the baseline correction process shifts the whole waveform upward or downward so that the average voltage during this period becomes zero.

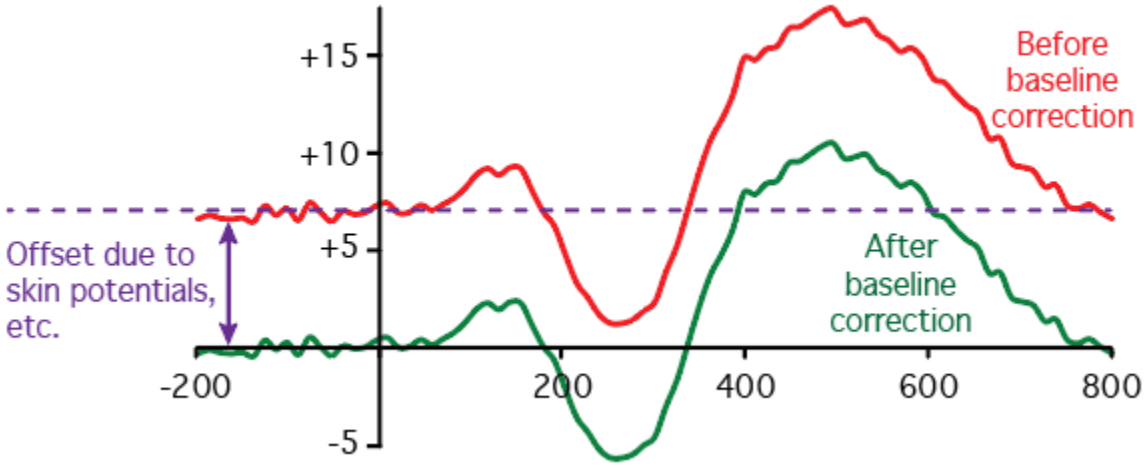
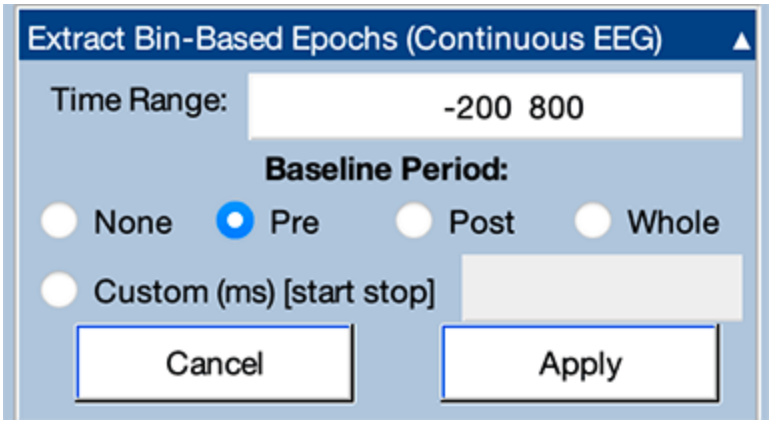


Figure 2.3. Baseline correction procedure. The mean voltage during the prestimulus period (from -200 to 0 ms in this example) is used as an estimate of the voltage offset. This value is simply subtracted from each point in the waveform to create the corrected waveform.

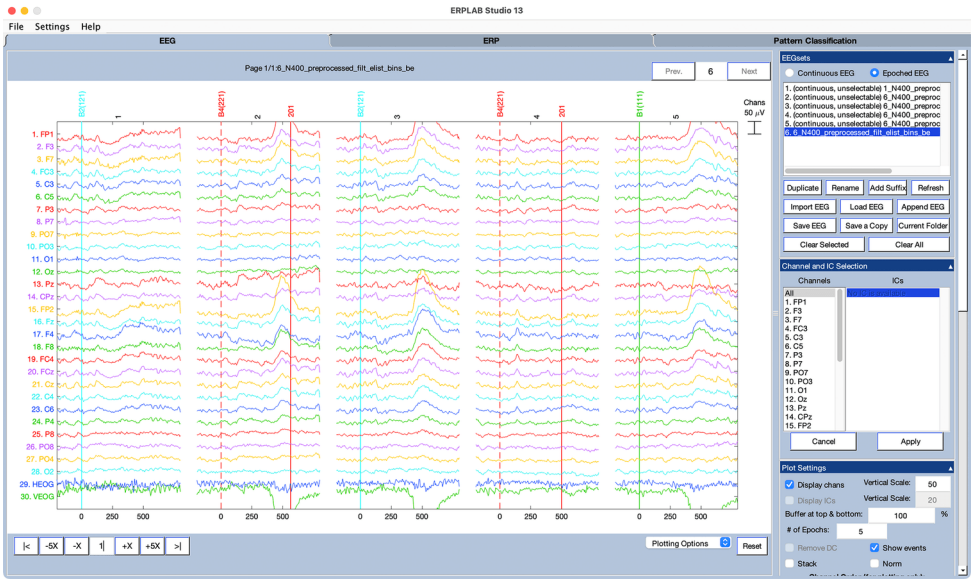
To perform these steps on our example data, make sure that the dataset created during the previous exercise (**6_N400_preprocessed_filt_elist_bins**) is loaded and selected in the **EEGsets** panel. Then go to the **Extract Bin-Based Epochs** panel, enter **-200 800** as the Time Range, and select **Pre** for baseline correction (as in Screenshot 2.12). The time window values tell the routine that you want the epochs to start 200 ms before stimulus onset and extend for 800 ms after stimulus onset, for a total epoch length of 1000 ms. The baseline period parameter tells the routine that you want to use the entire prestimulus interval (-200 to 0 ms) as the baseline period. Once you've set these parameters, click **Apply**. Accept the suggestion for the new EEGset name (**6_N400_preprocessed_filt_elist_bins_be**). If you're not going to do the next exercise right away, you should indicate that you want to save the new EEGset to disk.



Screenshot 2.12

Looking at the Epoched EEGset

The epoched EEGset should now be visible in eStudio's plotting region (and should look like Screenshot 2.13). Five epochs are shown per screen by default. You can see an event code at time zero in each epoch, with 200 ms of data prior to time zero and 800 ms of data afterward. Note that the time values on the X axis are now in milliseconds.



Screenshot 2.13

The event code at time zero is now labeled with the bin number, with the event code in parentheses. For example, the event code for the first epoch is labeled **B2(121)**, which means that this epoch was assigned to Bin 2 when you ran BINLISTER, and it had an event code of 121. When we create our averaged ERPs, all the epochs labeled B1 will be averaged together to form Bin 1 (whether the actual event code was 111 or 112), all the epochs labeled B2 will be averaged together to form Bin 2 (whether the actual event code was 121 or 122), etc.

Note that the second epoch contains an event code labeled **201** shortly after the 500 ms mark. This was the response to the target word. We're not using it as the time-locking point for any bins, so it does not have a bin number. However, the information about the event is still present, so you can see the event code in the plot.

Scroll through the epochs using the **+X** button at the bottom of the plotting window. Notice that there is usually a response event code following the target words (Bins 3 and 4) but not following the prime words (Bins 1 and 2). Sometimes the response event code is missing; this happens when the response was later than 800 ms and therefore fell outside of the time window of the

epochs. You might find it useful to view the EventList (by clicking **View** in the **EventList** panel) so you can compare the contents of the EventList to what you're seeing in the epoched EEG data.

You'll see a large positive voltage deflection in the frontal channels (especially Fp1 and Fp2) in many of the epochs, accompanied by a negative deflection in the VEOG channel. These voltage deflections were produced by eyeblinks—this participant blinked a lot! Most of the blinks occurred in the epochs for the prime words, not the target words. This is fortunate, because we mainly care about the ERPs elicited by the target words in this experiment.

Don't make this mistake!

EEGLAB also has a routine for extracting epochs from the continuous EEG (**EEGLAB > Tools > Extract epochs**), but do not use it!!! The EEGLAB routine knows nothing about bins, so the epochs won't contain the bin information that you'll need for the rest of the ERPLAB steps.

But what if you already have a dataset that has been epoched using the EEGLAB routine, followed by many other processing steps that you don't want to repeat? Or what if you've imported epoched data from another analysis system into EEGLAB and you'd like to process it in ERPLAB?

Fortunately, there is a trick for solving this problem. You can convert the epoched data back into continuous data by clicking the **Convert to Continuous EEG** button in the **EEG Utilities** panel (see this section of the ERPLAB Studio Tutorial for more details).

Note that the **Epoched EEG** radio button is now selected at the top of the **EEGsets** panel, and EEGsets 1–4 are labeled "(continuous, unselectable)". Several panels are also grayed out. You can have either epoched EEGsets or continuous EEGsets selected at a given time, but not both (because some panels operate on one type of data but not the other). If you want to look at or process one of the continuous EEGsets, you'll first need to select the **Continuous EEG** radio button.

This page titled 2.7: Exercise- Epoching and Baseline Correction is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

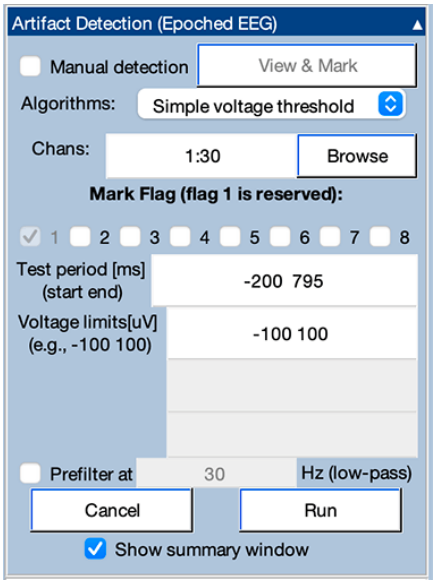
- 2.7: Exercise- Epoching and Baseline Correction by Steven J Luck is licensed CC BY 4.0.

2.8: Exercise- Artifact Detection

The eyeblink artifacts you saw in the previous exercise are huge relative to the EEG, and these artifacts can be a real problem. They can make it difficult to see the actual brain activity, and we need a way to deal with them. In the published version of the ERP CORE experiments, we used a method called *Independent Component Analysis* (ICA) to estimate and subtract the artifactual voltages, leaving behind the uncontaminated EEG. This is called *artifact correction*. ICA is a little complicated, so we won't use it in this chapter (but we'll cover it in detail in Chapter 9). Instead, we'll use a simpler approach called *artifact rejection*. In this approach, we use a simple algorithm to identify which epochs are contaminated by eyeblinks. We'll then mark these epochs by setting a *flag* in the EventList. Later, when we make the averaged ERPs, we'll simply exclude the flagged epochs from the averaging process. Note that, for most experiments, the optimal approach involves combining artifact correction with artifact rejection (Zhang et al., 2024).

There is a lot to know about artifacts and artifact rejection, and this topic will be covered in detail in Chapter 8. For now, we'll take a very simple approach in which we'll check every epoch to see if the voltage exceeds ±100 μV in any channel. If the voltage exceeds this range in a given epoch, we'll flag that epoch for rejection. We call this stage of the process *artifact detection* rather than *artifact rejection*, because we're simply marking the epochs with artifacts so that they will be excluded when we get to the averaging step.

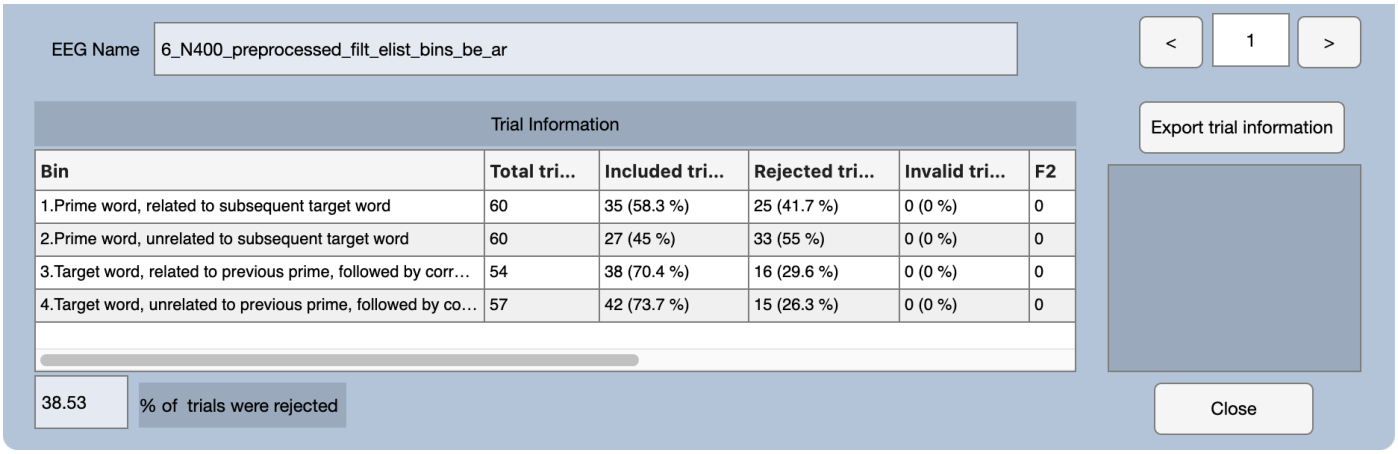
For this exercise, make sure that the dataset created during the previous exercise (**6_N400_preprocessed_filt_elist_bins_be**) is loaded and selected in the **EEGsets** panel. Then open the **Artifact Detection** panel and enter **-100 100** as the voltage limits so that the panel is set like Screenshot 2.14.



Screenshot 2.14

Click the **Run** button and accept the suggested name for the new EEGset (6_N400_preprocessed_filt_elist_bins_be_ar). A new window will then pop up, showing a summary of the number of epochs in which artifacts were detected (Screenshot 2.15). At the bottom of this window, you can see that 38.53% of trials were flagged for artifacts. You can also see the total number of trials per bin, the number and percentage that were included, and the number and percentage that were marked for rejection. For example, of the 60 total trials in Bin 1, 35 trials were not marked for rejection (e.g., will be included when you average the data) and 25 trials were marked for rejection. [Don't worry about invalid trials; they are rare.]

Ordinarily, my lab "throws out" any participant for whom more than 25% of trials were rejected (see Chapter 6 in Luck, 2014). However, this experiment was designed to be analyzed using artifact correction instead of artifact rejection (see Chapter 9), so we didn't actually need to exclude this participant.

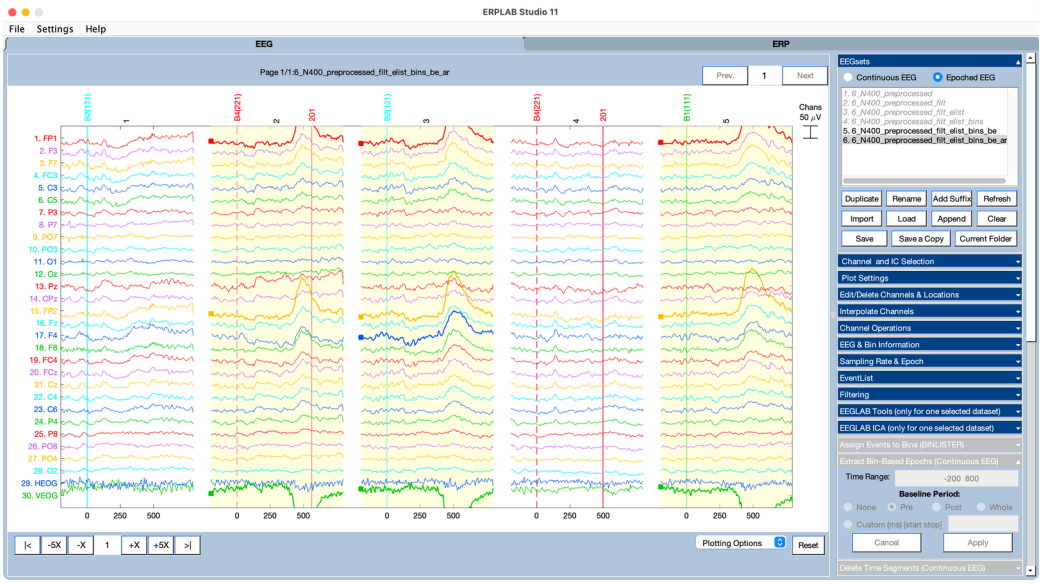


Screenshot 2.15

You can now close this window. If you want to see it again later, just select this EEGset from the **EEGsets** panel, go to the **Artifact Info & Tools** panel, and click **Artifact Summary**.

Verifying that the Epochs with Artifacts Have Been Flagged

Go ahead and take a look at the EEG in the plotting region. In the first screen of data, you can see that the second, third, and fifth epochs are marked as containing artifacts (see Screenshot 2.16). The individual channels that exceeded our ±100 µV limits are drawn with thicker lines and have a little square at the beginning of the waveform, and the epochs containing an artifact in one or more channels have a yellow background. We will exclude an entire epoch from our averages even if it contains an artifact in only one channel. The reason is that the artifact may not be easily visible in all channels in the raw EEG, but it might still be large enough to distort our data. Also, it would be a little weird if our averaged ERPs were based on different trials for different channels.



Screenshot 2.16

Now scroll through all of the data in the EEG plotting window and check to see if there are any epochs that: 1) contain large artifacts that are not marked for rejection, or 2) do not contain large artifacts and are nonetheless marked for rejection.

When I go through the data, it looks pretty good, but I did find a few epochs that contain smallish eyeblink artifacts but were not marked for rejection (e.g., epochs 9, 154, and 201). They all contain voltage deflections in the VEOG channel that have the

same basic shape as the eyeblinks that were flagged for rejection, along with an opposite-polarity deflection in the Fp1 and Fp2 electrodes. As will be discussed in Chapter 8, this pattern is characteristic of eyeblinks. So, the very simple approach that we've used to detect eyeblinks in this exercise is pretty good but not perfect. We'll talk about better approaches in Chapter 8.

After you mark epochs and check to see if the appropriate epochs were marked, you can decide whether it worked well enough. If it didn't, you can click **Clear** in the EEGsets menu to delete the EEGset you created and try again with different parameters (e.g., a different threshold. We'll talk more about how to decide whether it has worked well enough in Chapter 8.

Although the artifact detection we just performed wasn't good enough for publishing the data, it's good enough for the present demonstration. You should save the EEGset to disk by clicking the **Save** button in the **EEGsets** panel, because we'll need it later. This will save it in Matlab's current folder using the EEGset name as the filename.

When things don't work as expected...

At the end of Chapter 1, I noted that you will almost certainly run into problems while going through the exercises in this book, and you should treat this as an opportunity to hone your troubleshooting skills. I also noted that Appendix 2 provides detailed advice about troubleshooting.

When I first tried the artifact detection exercise in this section, the marked epochs were not shown with a yellow background (as in Screenshot 2.16). I looked at the troubleshooting advice in Section 2 of Appendix 2 (Some Basic Solutions), and the first suggestion was to try quitting and restarting eStudio. I tried it, and it solved the problem! So, if you run into problems, you might try this simple solution. And if that doesn't work, you can try the other suggestions in the appendix.

This page titled 2.8: Exercise- Artifact Detection is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

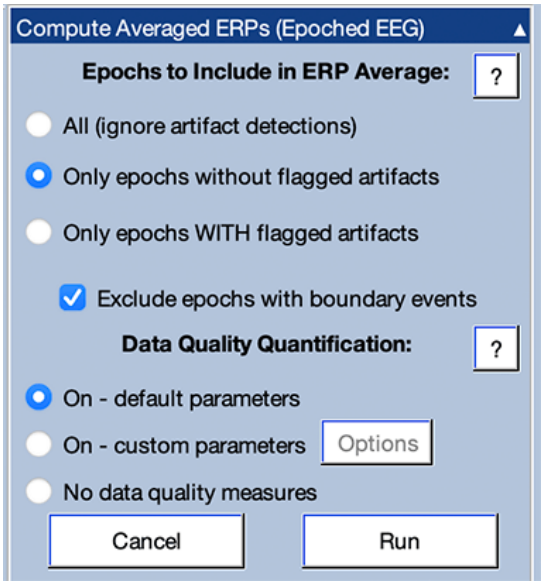
- 2.8: Exercise- Artifact Detection by Steven J Luck is licensed CC BY 4.0.

2.9: Exercise- Averaged ERPs

We're now finally ready to create the averaged ERPs for this participant. First make sure that the dataset from the previous exercise (**6_N400_preprocessed_filt_elist_bins_be_ar**) is loaded and selected in the **EEGsets** panel. Then go to the **Compute averaged ERPs** panel and make sure it is set up as shown in Screenshot 2.17. The selected radio button will exclude epochs flagged for artifacts during the artifact detection process, which you will almost always want to do. It's possible to ignore the artifact flags, which is sometimes useful. You can also average only the flagged epochs, which can be helpful for determining whether the artifacts vary systematically across bins. But you will almost always select the option for excluding marked epochs.

You should also make sure that there is a check mark in the box labeled **Exclude epochs that contain either "boundary" or invalid events**. A boundary is a special event code that indicates a discontinuity in the EEG. For example, imagine that the data collection was temporarily paused halfway through an experiment because the participant asked for a short break, and then it was restarted again a minute later. A boundary code would be inserted into the data at the time of the pause. If that boundary occurred midway through a trial (e.g., 300 ms after stimulus onset), we wouldn't want to include that trial, and so we always tell the averager to exclude trials with boundary events. Another possibility is that the **enable** flag in the EventList was set to zero for the time-locking event (e.g., because you realized that the participant had fallen asleep during the last part of the experiment and you therefore disabled the events during that period after the session was over). These trials should also be excluded.

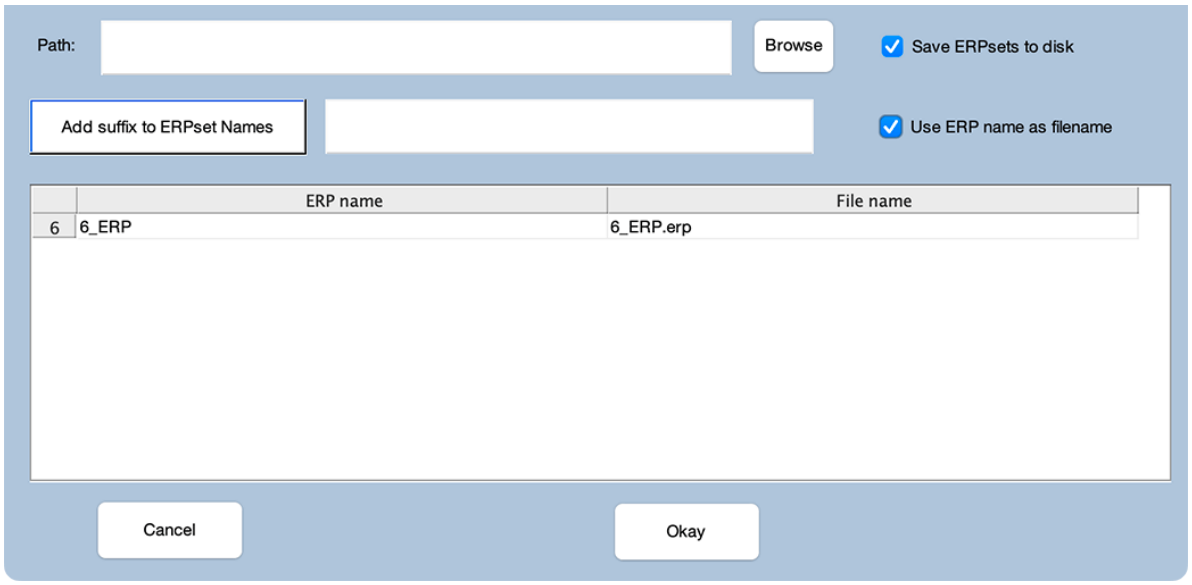
ERPLAB can calculate some measures of data quality during the averaging process, and the **Data Quality** section of the averaging GUI allows you to control this process. Just leave it set to **On – default parameters**.



Screenshot 2.17

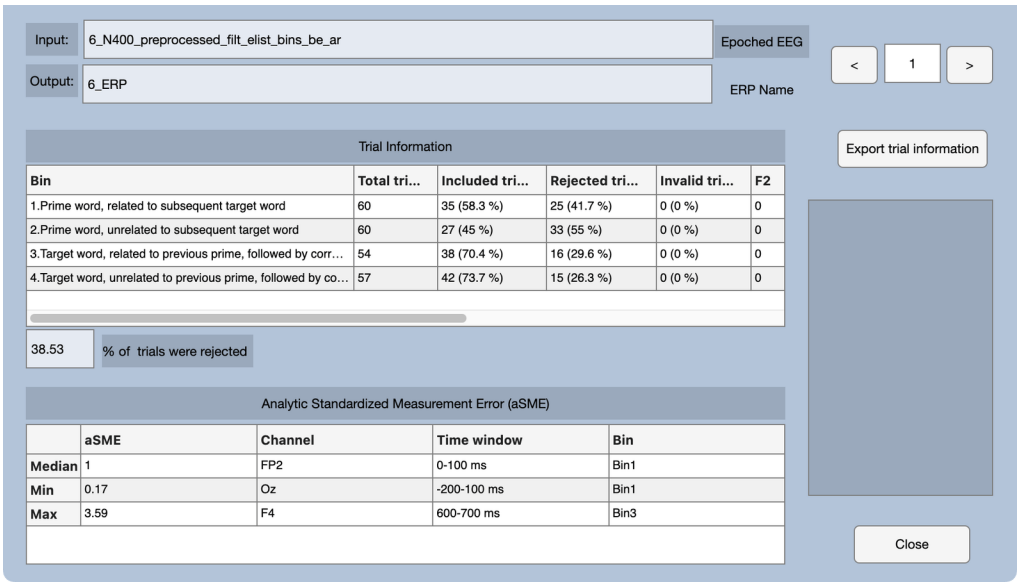
Once you have all the correct parameters set, click **Run** to compute the averaged ERPs.

You'll then see a window that allows you to save the averaged ERPs, which are stored in an ERPset (as illustrated earlier in Figure 2.2). As shown in Screenshot 2.18, you should overwrite the default name (which has gotten very long) and manually name the ERPset **6_ERP** (because this is the ERP data from Participant 6). That's the name that will show up in the **ERPsets** panel in the **ERP** tab. You should also save the ERPset to disk as a file. To do this, check the **Save ERPsets to disk** box. The name of the ERPset in memory does not need to be the same as the name of the file, but it's usually a good idea to use the same name for both. You can accomplish this by checking the **Use ERP name as filename** box, which will put **6_ERP** into the text box for the filename. Once you have everything set, you can click the **Okay** button.



Screenshot 2.18

After the averaging process is complete, you will see a window with summary information like the one shown in Screenshot 2.19. The **Trial Information** section allows you to confirm that the expected number of trials was included in each bin. The bottom section gives you some data quality information (which will be described in a later section). You should always take a look at this information to make sure that there aren't significant problems with the data.



Screenshot 2.19

When you are done looking at the summary window, click the **Close** button.

Dealing with multiple datasets for a given participant

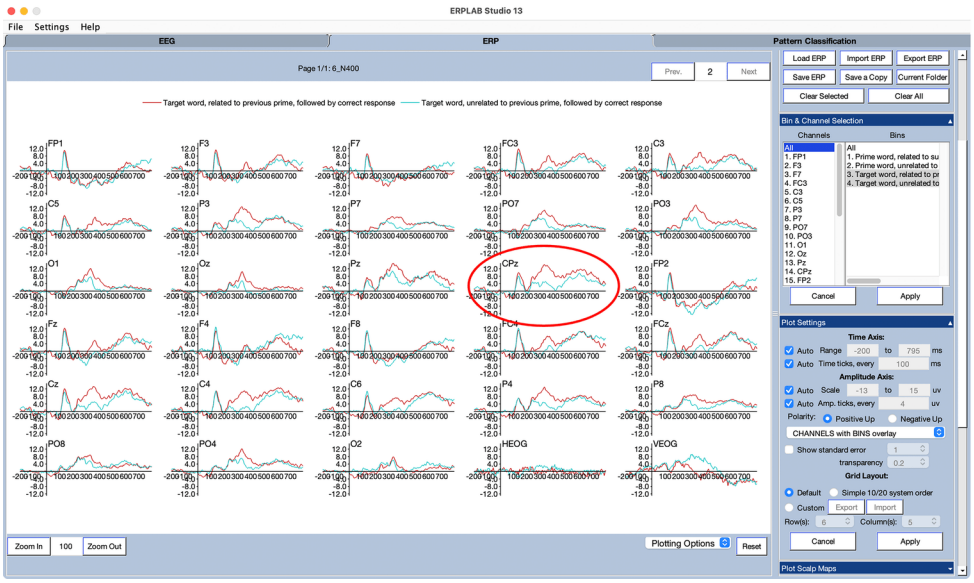
Imagine that you are running an EEG recording session and a fire alarm starts ringing halfway through the session. You stop the recording, disconnect the participant, and head to safety. But after 5 minutes, the alarm stops and you're allowed back into the lab. You reconnect the participant and resume the recording, but with the data in a new file. How do you average together the trials from the two files?

> Another possibility is that you run an experiment with 10 different trial blocks, and you create a different EEG recording file for each block. Again, you need to be able to average the data across multiple blocks.
>
> In most cases, the best solution is to combine the EEGsets into a single dataset by loading them into eStudio, selecting the EEGsets to be combined, and clicking the **Append** button in the **EEGsets** panel. This will concatenate all the selected EEGsets into a single EEGset. It's usually best to do this at an early stage of processing.

Viewing the Averaged ERP Waveforms

When the averaging processing completes, you will automatically be taken to the **ERP** tab, the new ERPset will be present in the **ERPsets** panel, and the averaged ERP waveforms will be plotted in the plotting region.

In this experiment, the main goal is to examine the difference between target words that were unrelated versus related to the preceding prime word. To focus on these waveforms, select Bins 3 and 4 from the **Bin & Channel Selection** panel and click **Apply**. You should see something that looks like Screenshot 2.20.



Screenshot 2.20

If you don't have much experience looking at ERP waveforms, the plot may look like a chimpanzee threw a plate of spaghetti on the wall, but once you gain some expertise it will be easy for you to comprehend what you're seeing. I always recommend starting by looking at the prestimulus baseline period. Note that it's relatively flat compared to the poststimulus period. In theory, the prestimulus period in the single-trial EEG epochs should contain only random noise, and if we average together enough epochs, the noise will "average out" to zero in the averaged ERP waveform. Of course, you'd need an infinite number of trials for the baseline to become perfectly flat. And even with an infinite number of trials, there may be an upward or downward "tilt" in the waveform during the prestimulus period as a result of overlapping ERP activity from the previous stimulus or anticipatory activity. However, you generally want to see that the prestimulus period in the averaged ERPs is relatively smooth, with deflections that are smaller than the experimental effects you see after time zero.

In this particular example, the prestimulus baseline period looks pretty good. The residual noise after averaging is relatively small compared to the ERPs in the poststimulus period, and there is no obvious tilt. I wish they always looked this good. I chose Participant 6 for this chapter because the EEG was very clean. In later exercises, you'll see participants with much noisier data.

Now take a look at the poststimulus period. Start by looking at the CPz channel, which is the channel where the N400 is typically largest. I circled this channel in Screenshot 2.20 because it can be hard to find. In fact, I had trouble finding it myself. It almost seems as if the channels are intentionally ordered in a manner that makes it difficult to find the channels you're looking for. This is because the order of channels in most systems reflects technical factors such as wire length minimization rather than reflecting neuroanatomy. In the next section, I'll explain how to obtain a more sensible spatial layout of the channels.

If you can't see the waveforms very well, try clicking the **Zoom In** button under the lower left portion of the plot area (or typing a new magnification factor into the text box next to the **Zoom In** button). You can then use the scroll bars to move around the plot.

Now that you're looking at the CPz channel, you can see that there is a large broad positive voltage for the targets that were related to the preceding prime word, extending from approximately 200 ms until the end of the epoch at 800 ms. For the targets that were unrelated to the preceding prime, the voltage is more negative (less positive) during much of this period. The existing evidence indicates that both the related and unrelated target words produce the broad positive voltage, but that the unrelated words also produce an additional negative voltage (the N400) that is added onto the broad positive voltage. So, even though the voltage remains above zero for the unrelated targets, the difference between the related and unrelated targets appears to be largely the result of the addition of an N400 component for the unrelated targets. This N400 activity appears to reflect the additional work the brain must do to process a word that is not related to the concepts that were already active when the word was presented (for a review of the N400 and other language-related ERP components, see Swaab et al., 2012).

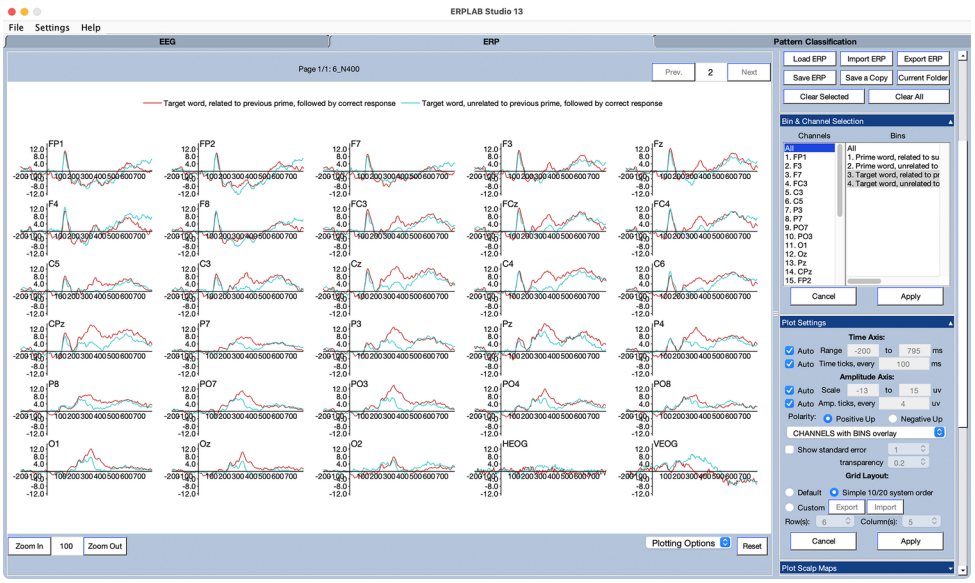
If this is the first time you've ever created averaged ERPs, I hope you have a real sense of accomplishment and awe. You are looking at voltages created by neurons in the brain of a living human being who viewed pairs of words and decided whether the second word in each pair (the target) was related or unrelated to the first word (the prime). A tremendous amount of brainpower and knowledge was needed for this participant to take the light emitted by the pixels on the computer screen, organize this light into letters and words, recognize the words, access their meanings, and compare them. And you are looking at the actual voltages created by the neurons as the brain carried out these processes. This is because the ERPs are the actual extracellular voltages produced by cortical pyramidal neurons as a result of neurotransmission, which (amazingly!) are able to pass through the brain, meninges, skull, and scalp to our recording electrodes. I've been recording and analyzing ERPs for over 40 years, and this still gives me chills!!!

Customizing the Channel Layout

A few years before we released ERPLAB Studio, I had lunch with a Psychology professor at Reed College named Michael Pitts. He told me that the main reason his lab used a commercial software package instead of ERPLAB Classic was that it was difficult to plot the channels in a rational layout with ERPLAB. I immediately understood that he was right; this was a major shortcoming of ERPLAB Classic that had bothered me for years. As soon as I got back to my lab, I told the ERPLAB Studio development team that we needed to make it easy to control the channel layout in ERPLAB Studio's plots. (To be honest, the "team" consisted of one person, Guanghui Zhang).

We came up with three solutions. One is the **Advanced Waveform Viewer**, which you can access from the **Plotting Options** menu underneath the plot region. This tool gives you a broader range of plotting options, with fine-grained control over the spatial layout of the channels. It also allows you to overlay different ERPsets so that you can, for example, overlay the grand average waveforms from two different groups of participants. In general, it is designed to help you make publication-quality plots. However, it takes a bit of time to set it up, and it seemed a little cumbersome for informal plotting.

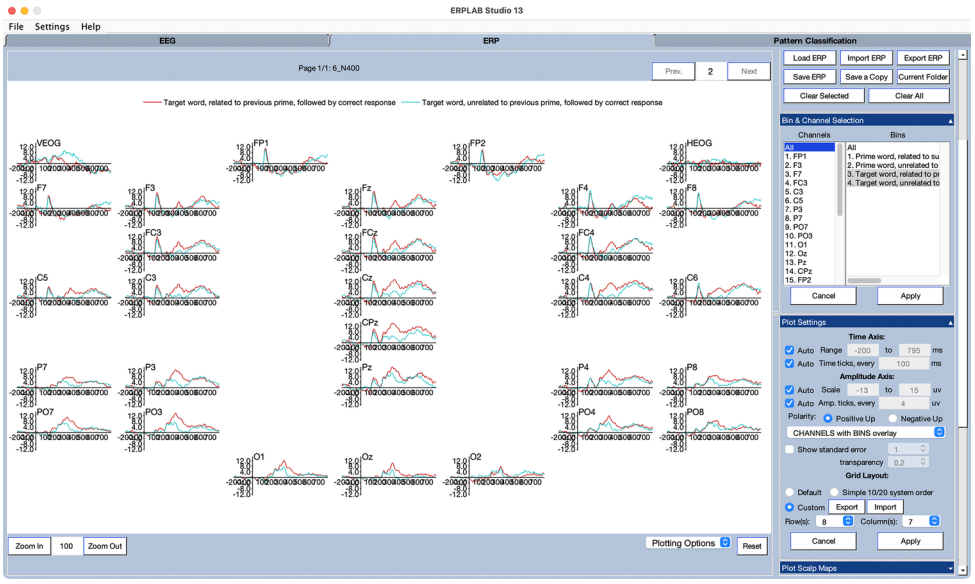
Our second solution was the **Simple 10/20 system order** option in the **Plot Settings** panel. Select this option now and then click **Apply**. You should see something like Screenshot 2.21. It puts the artifact channels first and then plots the frontal pole channels (FP1 and FP2), the frontal channels (F7, F3, Fz, F4, F8), and then keeps moving gradually toward the back of the head. In other words, the channel ordering goes systematically from left to right within a given anterior-posterior cluster, and then moves from the most anterior cluster (FP1 and FP2) to the most posterior cluster (O1 and O2).



Screenshot 2.21

That's an improvement, and it's very easy. But it's still not what Michael was looking for. What we really want is a way to organize the channels in a manner that more closely reflects their locations on the scalp.

To achieve that, we created a third solution: the **Custom** option in the **Plot Settings** panel. Select this option, click the Import button, and load the file named **GridLocations_Custom.tsv**. Then click **Apply,** and you will see a plot like the one shown in Screenshot 2.22.



Screenshot 2.22

Now we're cooking!

The **GridLocations_Custom.tsv** file is a tab-delimited text file that you can edit in any spreadsheet program. If you open it in a spreadsheet program like Excel, you will see that it specifies the layout in a very simple grid (see Screenshot 2.23). You can use your spreadsheet program to put the electrode names in the grid locations where you want them to appear. You can increase or decrease the number of rows and columns to be whatever you like.

	A	B	C	D	E	F	G	H
1		Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7
2	Row1	VEOG		FP1		FP2		HEOG
3	Row2	F7	F3		Fz		F4	F8
4	Row3		FC3		FCz		FC4	
5	Row4	C5	C3		Cz		C4	C6
6	Row5				CPz			
7	Row6	P7	P3		Pz		P4	P8
8	Row7	PO7	PO3				PO4	PO8
9	Row8			O1	Oz	O2		

Screenshot 2.23

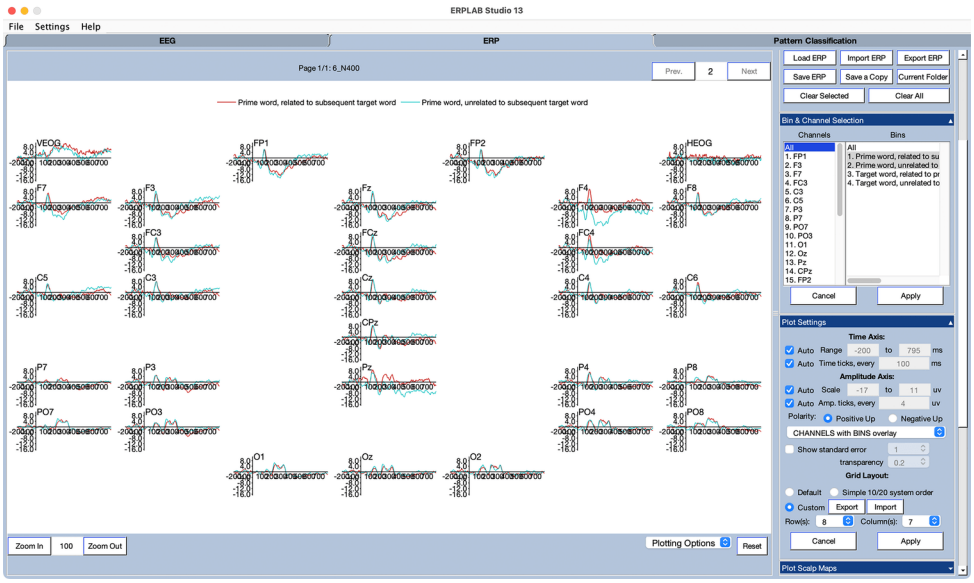
The easiest way to create a custom set of grid locations for your own data is to first use the **Default** layout and then use the **Export** button to create a locations file, edit that file in a spreadsheet program, and then use the **Import** button to import it. It will take you a couple minutes to make this file, but once you have it, you can use it for every experiment that uses that set of channels.

Note that this file must be in "tab-separated values" format. We prefer to use a **.tsv** filename extension for this file, but Excel does not like to save files with that extension, so you can just use a **.txt** filename extension. However, make sure that the channel labels are separated by tabs.

Viewing the ERPs from the Prime Words

Now let's look at the averaged ERP waveforms elicited by the prime words. Go to the **Bin & Channel Selection** panel, select Bins 1 and 2, and click **Apply**. Bin 1 is the ERP for prime words that are followed by a related target word, and Bin 2 is the ERP for prime words that are followed by an unrelated target word. Unless the participant had ESP, these ERPs should be equivalent except for random noise. How could the brain response to a given word vary according to the nature of the subsequent target word?

If you look at the Pz channel, you'll see that the waveforms for the two prime bins start deviating from each other after about 200 ms poststimulus (see Screenshot 2.24). After that point, the waveform becomes much more negative for primes followed by unrelated words than for primes followed by related words. Logically, this small difference must just be random noise in the data.



Screenshot 2.24

Now look at the F4 channel. You should see a large difference between Bins 1 and 2, beginning right around the onset of the prime word (0 ms). This absolutely must be noise, because it takes at least 50 ms for visual information to reach the cortex and generate an ERP. In the next section, we will discuss how to quantify data quality, and we will see that the poorest data quality is in the F4 and Pz channels, which our visual inspection here also flagged as being problematic.

Noise is an inevitable fact of life in ERP studies. After all, we're trying to measure voltages produced by tiny neurons in the cerebral cortex using electrodes placed on the skin, and there is a big thick skull between the neurons and the skin. Also, the brain signals are only a few millionths of a volt once they reach the scalp, where they're mixed with other signals such as skin potentials, muscle activity, and induced voltages from computers and other electrical devices in the recording environment. When you read published ERP papers, you don't usually get to see the single-participant data. Instead, you see grand averages, which have much less noise. Even without noise, the ERP waveforms from different people often look quite different from each other (probably due to individual differences in how the cortex is folded up in the brain). So, don't be surprised when the single-participant waveforms you see in this book or in your own studies look quite different from the grand average waveforms in published papers.

This page titled 2.9: Exercise- Averaged ERPs is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 2.9: Exercise- Averaged ERPs by Steven J Luck is licensed CC BY 4.0.

2.10: Exercise- Data Quality

Now that you've seen how noise can distort averaged ERP waveforms, let's look at how we can quantify the noise level (or, to put it in more positive terms, the *data quality*). When you averaged the data, recall that there was a **Data Quality Quantification** section in the averaging GUI, and you left it set to **On - default parameters**. And when the averaging routine finished, a window of summary information was provided that included data quality information at the bottom (Screenshot 2.19). Screenshot 2.25 shows a blown-up image of the data quality portion of that window.

Analytic Standardized Measurement Error (aSME)

	aSME	Channel	Time window	Bin
Median	1	FP2	0-100 ms	Bin1
Min	0.17	Oz	-200-100 ms	Bin1
Max	3.59	F4	600-700 ms	Bin3

Screenshot 2.25

This is a summary of a large set of data quality measures that are computed by default when you average, using a metric of data quality called the *standardized measurement error* (SME; the specific version of SME used here is the *analytic SME* or aSME). You can read a full description of the SME metric in Luck et al. (2021), and you can watch a short video overview and view a corresponding infographic.

Briefly, the SME values provided by default give you the *standard error of measurement* for the mean voltage within a set of time ranges (e.g., 0-100 ms, 100-200 ms, etc.). The larger the SME value, the less confidence we have that the voltage in that part of the waveform reflects the true voltage for that participant (i.e., the voltage that would be obtained if there were no noise or we could average over an infinite number of epochs).

SME values are interpreted in the same way as other standard errors. For example, imagine you were looking at a bar graph showing that the mean amplitude of an ERP component across a group of 20 participants was 6 μV, with error bars showing a standard error of ±1 μV. That would seem like pretty low error. Now imagine that the error bars showed a standard error of ±3 μV around the mean of 6 μV. That would seem like moderately large error. And if you saw a standard error of ±8 μV around the mean of 6 μV, that would seem like very large error. You would interpret a single participant's SME values in the same way. That is, if that participant's mean amplitude for a given ERP component was 6 μV, an SME of 1 μV would be very good, an SME of 3 μV would be moderately large, and an SME of 8 μV would be terrible.

The averaging routine computes a separate aSME value for each combination of bin, channel, and time period. Here we have 4 bins and 30 channels, and the averager computed the SME for 9 different time windows, yielding a total of 1080 different SME values. That's a lot of information about data quality! ERPLAB therefore summarizes the data for each participant by showing you the best (minimum), worst (maximum), and median aSME values for the current ERPset. You can see that the best aSME value was 0.17 μV at the Oz electrode site in Bin 1 during the first portion of the baseline period (-200 to -100 ms). The worst aSME value was 3.59 μV at the F4 electrode site in Bin 3 near the end of the epoch (from 600 to 700 ms). The median was 1.00 μV, so the worst value was more than three times as large as the median value.

Do you remember that we saw big differences between primes that were followed by related versus unrelated words starting at time zero in the F4 channel, which logically must have been noise? It's no accident that the aSME value was largest for the same channel. This channel was noisy, creating bogus differences between conditions and a large aSME value. The aSME value gives you a formal, quantitative way of determining the noise level so that you don't have to rely on subjective visual inspection. In addition, the SME values reflect both low-frequency and high-frequency noise, whereas you mainly see the high-frequency noise when you visually inspect averaged ERPs. In many cases, the low-frequency noise is a bigger problem than the high-frequency noise, so SME values can alert you to problems you wouldn't notice by visually inspecting the data.

Let's take a look at all the aSME values that were computed when the averaged ERPs were created. First, make sure that the ERPset from the previous exercise (6_ERP) is still loaded and selected in the **ERPsets** panel. Next, open the **View Data Quality Metrics** panel. This panel contains the same minimum, maximum, and median aSME values that are shown in Screenshot 2.25 above.

Now click the **Show in a table** button, which will open a new window containing a large table of aSME values. Each row is a different channel, and each column is a different 100-ms time range. You can select which bin is shown with a pop-up menu near the top of the window. We're currently looking at Bin 1.

A huge amount of information is shown in this table. To help you find the cases with the worst data quality (the largest values), select the **Color heatmap** option. Now the cells of the table are colored according to the magnitude of the aSME values (see Screenshot 2.26).



Screenshot 2.26

Notice that the values tend to get larger during later time windows. This is because the baseline correction procedure brings all the single-trial EEG epochs toward 0 µV during the prestimulus period, creating relatively low trial-to-trial variability during this period. Low-frequency noise will cause the voltage to drift randomly away from the baseline over a period of a few hundred milliseconds, and this causes more trial-to-trial variability later in the epoch. The more trial-to-trial variability is present, the harder it is to precisely measure the amplitude, so this drift causes larger aSME values.

You should also notice that the values tend to be largest in the F4 channel and second largest in the Pz channel. These are the same channels where we saw the greatest noise-related differences in voltage between Bins 1 and 2 in the averaged ERP waveforms in the previous section. The amplitude of the ERP at those time points (e.g., 500-600 ms) is about 6 µV at both F4 and Pz. Relative to this amplitude, the SME values of between 2 and 3 µV for these channels are not great, but they're not terrible.

You are currently looking at the aSME values for Bin 1. Look at the values for Bins 2-4 as well (using the **Selected BIN** popup menu). F4 and Pz are also noisy in those bins. If you go back and look at the EEG epochs, it's not obvious that these channels are noisier than the others. That's the value of having a quantitative metric of data quality: It's possible to objectively determine which channels, bins, and participants have unusually noisy data. In later chapters, we'll discuss what to do when a channel or participant is particularly noisy.

The data quality metrics are computed by default when you create averaged ERPs. Often, however, you want to assess the data quality prior to averaging (e.g., to determine whether a specific artifact rejection procedure will increase or decrease your data quality). To accomplish this, you go to the **EEG** tab, select the appropriate epoched EEGset in the **EEGsets** panel, go to the **Compute Data Quality Metrics** panel, and click **Run**.

This page titled 2.10: Exercise- Data Quality is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 2.10: Exercise- Data Quality by Steven J Luck is licensed CC BY 4.0.

2.11: Review of Processing Steps

You did a lot of exercises in this chapter, and they're probably beginning to blur together in your mind. So let's review the steps:

- Load the EEG data using the **EEGsets** panel in the **EEG** tab
  - Look at the dataset to verify that the EEG and event codes looked okay
- Filter out the low-frequency drifts in the EEG using the **Filtering** panel in the **EEG** tab
- Add an EventList to the EEGset using the **EventList** panel in the **EEG** tab
  - Verify that we had the correct number of occurrences of each event code
- Use BINLISTER to assign events to bins using the **Assign Events to Bins** panel in the **EEG** tab
  - Verify that we had the correct number of trials in each bin
- Extract fixed-length epochs from the continuous EEG and perform baseline correction using the **Extract Bin-Based Epochs** panel in the **EEG** tab
- Test each epoch for artifacts using a simple voltage threshold using the **Artifact Detection** panel in the **EEG** tab
  - Verify that the appropriate epochs were marked and see how many artifacts were detected
- Average together the artifact-free epochs, separately for each bin using the **Compute Averaged ERPs** panel in the **EEG** tab
  - In the summary window that appears, verify that we had the correct number of trials in each bin and view the data quality summary
- View the averaged ERP waveforms in the **ERP** tab
  - Create a custom channel layout using the **Plot Settings** panel in the **ERP** tab
  - Verify that each bin has the correct number of trials by looking in the **ERP & Bin Information** panel
- Examine the data quality of the averaged ERPs using the **View Data Quality Metrics** panel in the **ERP** tab

Although this may seem like a lot, it's the minimal set of steps for going from raw EEG to an averaged ERP in most studies. There are many additional steps that are often included in the *EEG preprocessing pipeline*, and there are also several steps after averaging. Moreover, each step has many options.

That's why I decided to write this book! I wanted to help you and other ERP researchers fully understand how to process EEG/ERP data using optimal methods that will lead to robust and important scientific discoveries.

This page titled 2.11: Review of Processing Steps is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 2.11: Review of Processing Steps by Steven J Luck is licensed CC BY 4.0.

2.12: A Simple Matlab Script

How long did it take you to run all of the steps in this chapter? Quite a while, I suspect. Now imagine that there are twice as many steps (as will typically be the case), and that one of them takes 20 minutes to run (which is typical for the artifact correction process that we typically use). And imagine that you've collected data from 40 participants and need to repeat the sequence of steps for each of them. Now imagine that a reviewer insists that you change one of the steps, so now you need to run them all again. Does this sound like fun? I don't think so. Also, when you do each step "by hand" using the ERPLAB Studio GUI, you're likely to make occasional errors.

To make your life easier and to reduce errors, you can automate the processing steps using a Matlab *script*. A script is basically a set of commands that you could type in the Matlab command window but that you instead save as a text file with a **.m** extension (sometimes called a *.m file*). To make things more efficient, a script can include more sophisticated programming elements like loops, if/then statements, and variables. If you already know how to program in a typical programming language, you'll find these aspects of Matlab scripting to be pretty straightforward. If you don't already have much programming background, this book will teach you the basics.

Fortunately, EEGLAB and ERPLAB have a *history* feature that makes scripting a lot easier. Every time you do something in the EEGLAB/ERPLAB GUI, the equivalent script command is saved in the history. For example, when you ran the artifact detection step, the following was saved in EEGLAB's history:

```
EEG  = pop_artextval( EEG , 'Channel',  1:30, 'Flag',  1, 'LowPass',  -1, 'Threshold', [ -100 100], 'Twindow', [ -200 795] );
```

This makes it really easy to see how to create a line of script that corresponds to a step you carried out in the GUI. In fact, when I write EEGLAB/ERPLAB scripts, I often do the steps first in the ERPLAB Studio GUI, look at the history, and then copy the relevant commands from the history into my .m file. In a later chapter, we'll discuss how to access the history and build your own scripts. For now, let's run an example script just so that you can see the basic idea.

In the folder for this chapter, there is a file named **preprocess_EEG.m**. You should be able to see it in the **Current Folder** pane of the Matlab GUI. Double-click the filename to open it in the Matlab text editor. This text editor recognizes certain aspects of the Matlab scripting syntax and uses different coloring for different parts of the script (even though the actual file is just plain text without any colors specified, as you can verify by opening it in some other text editor). Lines that begin with a percent symbol are comments, and by default they're colored in green. I added some comments at the top of the script that explain how to run it, and I also put a comment above each line of code to explain what that line does. For example, near the bottom you will see the **pop_artextval** command for doing artifact detection, and there is a comment above that line that explains what it does.

This script does all the basic EEG preprocessing steps that we covered in this chapter. Read through the comments in the script file to see the steps. The last step prior to averaging was artifact detection, and the dataset created by this step is saved to a file. The filename has **_from_script** appended to it.

To run the script, first quit eStudio to make sure that we don't have any conflicts. Then, run the script. There are several ways to do this, but the simplest is to click the **Run** button in the toolbar at the top of the Matlab text editor window for the **preprocess_EEG.m** file. While the script is running, you'll see a bunch of text appear in the Matlab command window, with information about each step that is running. When everything is done, you'll see a prompt in the command window.

If you see an error message or have some other kind of problem, then you're about to learn something useful! Start by reading the error message and trying to figure out how to solve the problem. If that doesn't work, see if there is someone around who has experience running Matlab scripts and can give you some help. And if you can't find someone to help, it's time to consult the troubleshooting tips in Appendix A2. You might also want to read the first half of Chapter 11 to learn some of the basic concepts of EEGLAB/ERPLAB scripting.

Once you've gotten the script to run, you should see that it has created a new dataset file named **6_N400_preprocessed_filt_elist_bins_be_ar_from_script** in the same folder with all the other files from this chapter. Launch eStudio and open this file using the **EEGsets** panel. The EEG in this file should look just like the EEG data that you saw after going through all the preprocessing steps with the GUI.

I hope you can now see how much faster it is to process a participant's data with a script than by doing it with the GUI. However,

this doesn't mean that you won't need the GUI once you've learned EEGLAB/ERPLAB scripting. You should use the GUI the first time you go through a participant's data so that you can check for problems and errors. You'll also need to use the GUI to determine the appropriate artifact detection parameters (and artifact correction parameters once you learn about that). But once you've processed the participant's data manually and determined the right parameters, you can run a script using those parameters to reprocess the participant's data. That way you can avoid any errors you might make when you process the data manually. And when Reviewer 2 asks you to change your preprocessing pipeline, you'll be able to reprocess all the data quite easily and won't be as grumpy about making the change.

This page titled 2.12: A Simple Matlab Script is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 2.12: A Simple Matlab Script by Steven J Luck is licensed CC BY 4.0.

2.13: Key Takeaways and References

Key Takeaways

- Several *EEG preprocessing* steps are necessary prior to creating the averaged ERPs for a given participant (and this chapter did not even cover all of them).
- It is important to look at the EEG/ERP waveforms after each step. It is also important to look at the EventList information after creating the EventList and after assigning events to bins with BINLISTER to verify that there are no problems or errors.
- You should also look at the data quality metrics that are computed during the averaging process to determine if there are problematic bins or channels (or a more general problem with the data quality for a given participant).
- In this experiment, the data analyses focus on the target words to see if the response to the target depends on whether the target word was related or unrelated to the preceding prime word. However, owing to noise, there were some differences in the ERP elicited by the prime words that were followed by either related or unrelated targets. The data quality metrics help you verify that the EEG was particularly noisy in the channels that showed these unexpected differences.

References

Kappenman, E. S., Farrens, J. L., Zhang, W., Stewart, A. X., & Luck, S. J. (2021). ERP CORE: An Open Resource for Human Event-Related Potential Research. *NeuroImage*, *225*, 117465. https://doi.org/10.1016/j.neuroimage.2020.117465

Kutas, M. (1997). Views on how the electrical activity that the brain generates reflects the functions of different language structures. *Psychophysiology*, *34*, 383–398. https://doi.org/10.1111/j.1469-8986.1997.tb02382.x

Luck, S. J. (2014). *An Introduction to the Event-Related Potential Technique, Second Edition*. MIT Press.

Luck, S. J., Stewart, A. X., Simmons, A. M., & Rhemtulla, M. (2021). Standardized measurement error: A universal metric of data quality for averaged event-related potentials. Psychophysiology, 58, e13793. https://doi.org/10.1111/psyp.13793

Swaab, T. Y., Ledoux, K., Camblin, C. C., & Boudewyn, M. (2012). Language-related ERP components. In S. J. Luck & E. S. Kappenman (Eds.), *The Oxford Handbook of Event-Related Potential Components* (pp. 397–439). Oxford University Press.

Zhang, G., Garrett, D. R., & Luck, S. J. (2024). Optimal filters for ERP research II: Recommended settings for seven common ERP components. *Psychophysiology, 61,* e14530. https://doi.org/10.1111/psyp.14530

Zhang, G., Garrett, D. R., Simmons, A. M., Kiat, J. E., & Luck, S. J. (2024). Evaluating the effectiveness of artifact correction and rejection in event-related potential research. *Psychophysiology, 61*, e14511. https://doi.org/10.1111/psyp.14511

This page titled 2.13: Key Takeaways and References is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 2.13: Key Takeaways and References by Steven J Luck is licensed CC BY 4.0.

3: PROCESSING MULTIPLE PARTICIPANTS IN THE ERP CORE N400 EXPERIMENT

Learning Objectives

In this chapter, you will learn to:

- Run a script for making averaged ERPs for multiple participants
- Create a grand average (an average of the single-participant averages)
- Visualize the grand average waveforms and noise metrics
- Apply low-pass filters to the grand average to minimize high-frequency noise
- Quantify N400 amplitude from single-participant averaged ERP waveforms
- Conduct a simple statistical analysis on the N400 amplitudes
- Conduct a more complex statistical analysis
- Run a script that automates the steps in this chapter

In the previous chapter, we processed the data from a single participant in the ERP CORE N400 experiment, up to the stage of creating and viewing the averaged ERP waveforms. In this chapter, we'll process nine more participants and do the steps necessary to get to the stage of quantifying N400 amplitude in each condition and conducting some statistical analyses.

Between the previous chapter and the current chapter, you'll be able to see most of the major steps required to go from raw data to a final conclusion. The details will be spelled out in subsequent chapters; my goal for Chapters 2 and 3 is for you to see the big picture before we dive into those details. To make things go quickly in this chapter, we'll be looking at only 10 of the 40 participants in the actual study. If you're already experienced with ERPLAB Studio, you can just skim this chapter.

Don't forget to consult the troubleshooting tips in Appendix 2 if you run into error messages or other problems.

3.1: Data for This Chapter

3.2: Exercise- Organizing and Loading the Data from 10 Participants

3.3: Exercise- Examining the Single-Participant ERPsets

3.4: Exercise- "Bad" Data

3.5: Exercise- Making a Grand Average

3.6: Exercise- Low-Pass Filtering

3.7: Exercise - Scoring N400 Amplitude

3.8: Exercise- Simple Statistical Analysis of N400 Data

3.9: Exercise- A More Complex Analysis

3.10: Exercise- ERP Channel Operations

3.11: Exercise- ERP Bin Operations

3.12: Review of Processing Steps

3.13: Matlab Scripts For This Chapter

3.14: Key Takeaways and References

This page titled 3: Processing Multiple Participants in the ERP CORE N400 Experiment is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 3: Processing Multiple Participants in the ERP CORE N400 Experiment by Steven J Luck is licensed CC BY 4.0.

3.1: Data for This Chapter

The N400 data we'll be using for the exercises in this chapter can be found in the Chapter_3 folder in the master folder: https://doi.org/10.18115/D5G045. Go ahead and download this folder and all its contents (and make sure to name the folder **Chapter_3**).

This page titled 3.1: Data for This Chapter is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 3.1: Data for This Chapter by Steven J Luck is licensed CC BY 4.0.

3.2: Exercise- Organizing and Loading the Data from 10 Participants

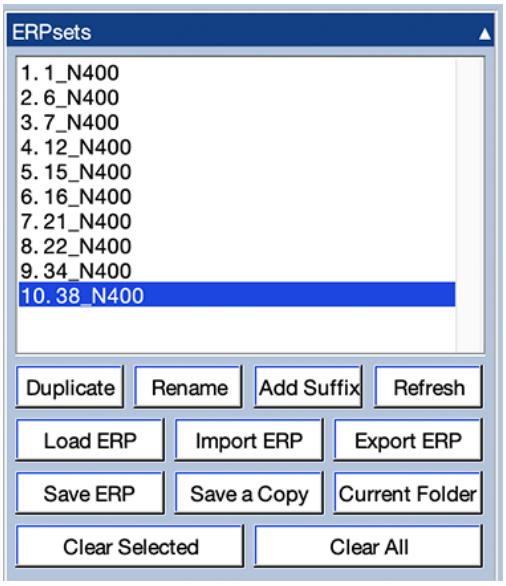
In this chapter, we'll be looking at the data from 10 of the 40 participants in the ERP CORE N400 experiment (Subjects 1, 6, 7, 12, 15, 16, 21, 22, 34, 38). Take a look at how the files are organized in the Chapter_3 folder. You'll see that there is a separate folder for each of the 10 participants, each containing an EEGset with the original EEG data for that participant, an EEGset with all of the preprocessing steps completed, and an ERPset with the averaged ERPs. This may seem like a cumbersome way to organize the data, because it requires moving from folder to folder as you process the data from the different participants. However, you will typically end up with 10-20 files per participant by the time you're done analyzing a given experiment, and it would be a bad idea to have all of these files in a single folder. I strongly recommend organizing your data as I have organized the data for this chapter (and the subsequent chapters).

I created the ERPsets for each participant by going through the same sequence of operations used for Subject 6 in the previous chapter. To save time, I implemented these operations in a script rather than by using the eStudio GUI. In case you'd like to look at this script (or even run it yourself), it's in the Chapter_3 folder and is named **preprocess_and_average.m**. It would be a useful exercise for you to do all of the preprocessing steps and the averaging step for several of these participants using the eStudio GUI. This would take a bit of time, but you'd learn the preprocessing steps really well by repeating them for each participant. If you don't want to spend the time, you can just use the EEGset and ERPsets I created using the script.

Before you go any further, it's a good idea to restore all the eStudio parameters to their default values. To do this, click the **Reset** button under the lower right corner of the plot region. After a pause, you will see a small window that allows you to reset the parameters. Click the button labeled **Reset ERPLAB Studio** to clear out all loaded files and reset all settings to their default values. You should also set **Chapter_3** to be the current folder (using either the Matlab GUI or the **Current Folder** button in the **EEGsets** or **ERPsets** panel).

Now let's load the ERPsets from the 10 participants. One way to accomplish this is to go to the **ERP** tab, then go to the **ERPsets** panel, click **Load**, navigate to the folder named **1**, and open **1_N400.erp**. You could then repeat this for each of the other 9 participants (6, 7, 12, 15, 16, 21, 22, 34, and 38). This can be a little tedious (especially in an experiment with 40 participants), so I wrote a little script named **load_erpsets.m** that loads all the ERPsets for you. If you don't want to load the ERPsets manually, you can run this script (by opening it and clicking the **Run** button in the Matlab editor window) and then click the **Refresh** button in the **ERPsets** panel (to let eStudio know that they have been loaded).

The ERPsets panel should now look like Screenshot 3.1.



Screenshot 3.1

This page titled 3.2: Exercise- Organizing and Loading the Data from 10 Participants is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

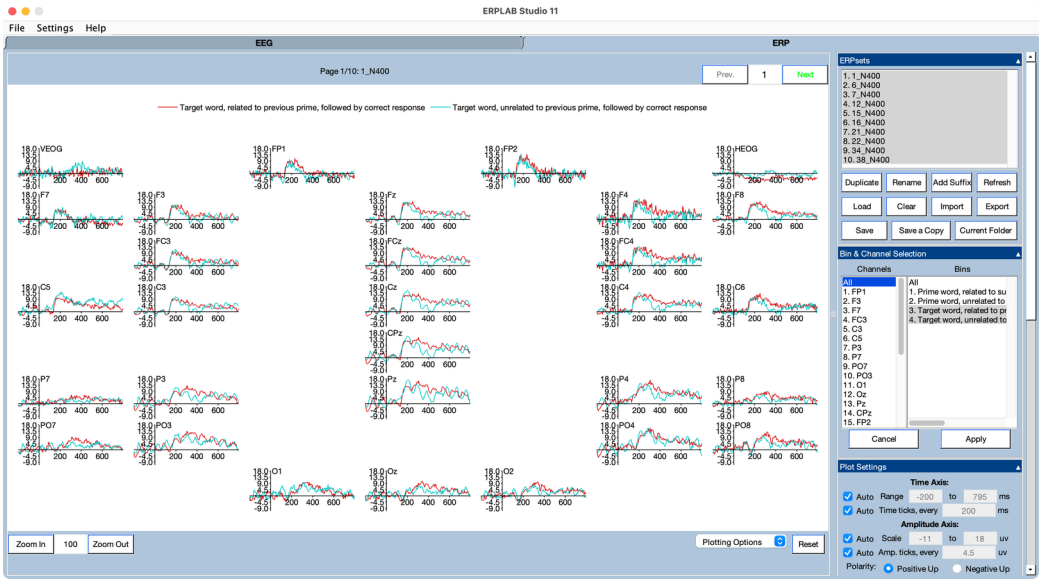
- 3.1: Data for This Chapter by Steven J Luck is licensed CC BY 4.0.
- 3.2: Exercise- Preprocessing and Averaging the Data from 10 Participants by Steven J Luck is licensed CC BY 4.0.

3.3: Exercise- Examining the Single-Participant ERPsets

Now let's take a look at the averaged ERP data from each participant. Select all 10 of the ERPsets in the **ERPsets** panel. Although all 10 are selected, only the first one should be displayed in the plotting region. You can choose which one is displayed using the **Prev.** and **Next** buttons (or the text box between them) in the upper right corner of the plotting region.

We mainly care about the ERPs elicited by the target words, so select Bins 3 and 4 in the **Bin & Channel Selection** panel (and click **Apply**). And let's use the nice channel layout that we used at the end of Chapter 2: Go to **Plot Settings**, select **Custom**, click **Import**, load **GridLocations_Custom.tsv**, and click **Apply**. It should look something like Screenshot 3.2.

Screenshot 3.2



If you look at the CPz electrode site, you should see that the unrelated targets elicited a more negative voltage around 400 ms than the related targets. This is the same effect we saw for Subject 6 in Chapter 2, but we're now looking at Subject 1's waveforms, which are a bit noisier. If you click on the **Next** button in the upper right corner of the plotting region, you can see the data from Subject 6. There is clearly less high-frequency noise in Subject 6 than in Subject 1.

It's also a good idea to look at the number of trials included in each average when you're viewing the ERP waveforms. To do this, open the **ERP & Bin Information** panel. The bottom of the panel contains a table showing the number of accepted and rejected trials in each bin. You can see that more trials were rejected for Subject 1 than for Subject 6. [You can get more detailed information by clicking the **Bin Info Table** button.]

Using the **Next** button, take a look at the rest of the ERPsets (both the waveforms and the number of accepted/rejected trials). Although all participants showed the typical N400 effect (more negative voltage for unrelated targets than for related targets at CPz and the surrounding electrodes), the waveforms differ quite a bit across participants. The noise also varies quite a bit. For example, Subject 22 has high-frequency noise at the most anterior electrode sites, and Subject 38 has clear alpha-band oscillations remaining in the waveforms at the posterior sites. The number of rejected trials also differs quite a bit across participants. This kind of variability is perfectly normal. In fact, these data are from a fairly homogeneous group of highly compliant young adults, so the data are cleaner and less variable than what is seen in many studies.

Viewing the Data Quality Metrics

Now let's look at the data quality by going to the **View Data Quality Metrics** panel (with all 10 subjects still selected in the **ERPsets** panel). Click the **Show in a table** button, which will open a new window letting you see the aSME values. By default, the values from the first ERPset will be shown. Select the **6_N400** ERPset using the **Selected ERPset** menu at the top of the window. The aSME values are shown for Bin 1, but you can change to a different bin using the **Selected Bin** menu. Turn on the

Color Heatmap option. The table should now look exactly like what you saw for this participant in Section 10 of Chapter 2, with the worst (largest) aSME values in the later time windows in the F4 channel.

Now select the **1_N400** ERPset using the **Selected ERPset** menu at the top of the window. The worst data quality for this participant is in the C5 channel. If you look at the other bins for this participant (using the **Selected Bin** menu), you'll see that C5 has the highest aSME values for all bins, and that it's worse for bins 3 and 4 (the targets) than for bins 1 and 2 (the primes).

Now look at the data quality for each of the remaining ERPsets. You should see that channel with the worst data quality varies quite a bit from participant to participant. This is normal; unless you have a hardware problem, the noisiest channel will be highly idiosyncratic (especially for the low-frequency noise that actually matters most for data quality when ERPs are quantified as the mean voltage over a time range). You'll also see that the highest values vary somewhat across participants, but none of the SME values are huge. I selected these 10 participants for this chapter because they all have nice, clean data.

ERPLAB automatically computes some additional data quality metrics. For example, if you select **Baseline Measure – SD** from the **Selected DQ Measure** menu, you can see a measure of noise during the prestimulus baseline period (the standard deviation of the voltage from point to point during this period). More details can be found in the ERPLAB manual page on data quality metrics. You may now close the data quality viewing window.

Viewing the ERP and ALLERP Data Structures

You can do quite a lot in ERPLAB Studio without knowing anything about Matlab programming. However, even if you don't care anything about programming, it's worth knowing a little bit about the data structures that are used to store EEGsets and ERPsets.

If you go to the main Matlab window, you should see a pane labeled Workspace (probably along the right side of the Matlab window). That pane shows all of the variables that are currently available to Matlab. The most important variables are

- EEG, which holds the currently selected EEGset
- ALLEEG, which holds all of the EEGsets that are currently loaded
- ERP, which holds the currently selected ERPset
- ALLERP, which holds all of the ERPsets that are currently loaded

Here we will take a look at the ERP variable. Go to the Matlab command window and type **ERP** (followed by Return or Enter). You should see something like the information shown in Screenshot 3.3.

Screenshot 3.3

```
>> ERP

ERP =

  struct with fields:

       erpname: '1_N400'
      filename: '1_N400.erp'
      filepath: '/Users/luck/Dropbox/Research/Manuscripts/ERP_Analysis_Book/Ch_3_Getting_Started_10_Participants/Exercises/1'
     workfiles: {'1_N400_preprocessed_filt_elist_bins_be_ar_from_script.set'}
       subject: ''
         nchan: 30
          nbin: 4
          pnts: 200
         srate: 200
          xmin: -0.2000
          xmax: 0.7950
         times: [1×200 double]
       bindata: [30×200×4 double]
      binerror: [30×200×4 double]
      datatype: 'ERP'
       ntrials: [1×1 struct]
     pexcluded: 29.6000
        isfilt: 0
      chanlocs: [1×30 struct]
           ref: 'common'
     bindescr: {1×4 cell}
        saved: 'yes'
      history: [3×2167 char]
      version: '8.12'
    splinefile: ''
     EVENTLIST: [1×1 struct]
   dataquality: [1×3 struct]
```

Typing the name of a variable in the command window tells Matlab to print out the contents of that variable. ERP is a complicated variable with many different fields, including the **erpname** field (the name of the ERPset), the **nchan** field (which stores the number of channels), the **chanlocs** field (which stores the names and 3-D locations of the electrodes), and the **bindata** field (which stores

the actual ERP data, in binary format).

You can also see the contents of a variable by double-clicking the name of the variable in the Workspace pane. Try this by double-clicking the ERP variable in the Workspace. This causes the variable to be shown in a separate pane. You can then double-click on the fields of the **ERP** variable to see those fields in more detail. For example, try double-clicking the **chanlocs** field to see what information it holds.

Now look at the **ERP.ntrials** field. This allows you to see the number of accepted trials, the number of rejected trials, and the number of invalid trials for each bin. You should see that there were 47 accepted trials and 13 rejected trials for Bin 1.

Looking at the ERP and EEG variables can be helpful if things aren't working correctly and you want to understand more precisely what is happening.

This page titled 3.3: Exercise- Examining the Single-Participant ERPsets is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 3.3: Exercise- Examining the Single-Participant ERPsets by Steven J Luck is licensed CC BY 4.0.

3.4: Exercise- “Bad” Data

So far, we’ve been looking at really clean data. However, the reality of ERP research (and most areas of human neuroscience) is that you often get some participants with really noisy data. And noisy data is the norm for most participants in some research areas (and large numbers of participants are needed to make up for it). For example, imagine trying to record the EEG from wiggly 2-year-olds. You’d get all kinds of movement artifacts, and they won’t sit through an hour of data collection the way a paid adult will. But you’ll also see some noisy data in studies of calm, compliant adults. So, no matter what kind of ERP research you’re interested in, you’ll need to deal with noisy data.

In this exercise, we’ll look at one of the 40 participants in the full N400 study whose data were problematic (Subject 30). This participant wasn’t horrible—all of our participants were college students who were pretty compliant with our instructions, and we know a lot of tricks for optimizing the data quality in EEG recordings (see Farrens et al., 2019 for a detailed description of our EEG recording protocol). However, the data from this participant were problematic in a way that we see in 5-10% of our college student population.

To see this participant’s data, set the current folder to the folder named **Bad_Subject** inside the **Chapter_3** folder. This folder contains the original EEG dataset file, the EEG dataset file after all preprocessing steps (including artifact detection), and the averaged ERPset file.

Start by loading the averaged ERP datafile (**30_N400.erp**) from this participant using the **ERPsets** panel and selecting Bins 3 and 4 in the **Bin & Channel Selection** panel. You should see a very noisy waveform for the related target words, but the waveform for the unrelated target words is missing. If you look at the aSME data quality metric in the **View Data Quality Metrics** panel, you’ll find an aSME value of 0 for every channel at every time point for Bin 3 (related targets), and a value of **NaN** for Bin 4 (unrelated targets). **NaN** is an abbreviation for *not a number*, and it’s what Matlab uses when something can’t be computed (e.g., when it requires dividing by zero).

Now plot the ERP waveforms for the prime words (Bins 1 and 2) and look at the aSME values for these words. The waveforms are noisy, and the aSME values are somewhat higher than those for the 10 participants you processed in the previous exercises. But at least it looks like there is valid data for these bins.

Your job now is to figure out what has gone wrong with Bins 3 and 4 for this participant. In Section 2.11 of Chapter 2, I made a point of describing several pieces of information you should verify while processing a participant’s data. Use those hints to figure out what went wrong with this participant. You might want to load the EEGset that was used for the averaging process to check this information (**30_N400_preprocessed_filt_elist_bins_be_ar_from_script.set**). Once you’ve figured out the problem, you can read the text box below to make sure your answer was correct (but no peaking until you’ve figured it out for yourself!).

I hope you’ve now figured out the problems with Subject 30. I included this example to drive home a point that I made in Chapter 2, namely that you really need to pay close attention when you’re initially processing each participant’s data. Don’t just run a script and hope for the best. Look at the number of event codes, the number of accepted and rejected trials, the continuous EEG, and the epochs that were marked by the artifact detection process. If you don’t, your data will be filled with C.R.A.P. (which is an acronym for *Commonly Recorded Artifactual Potentials*, but is also used to describe a wide variety of miscellaneous data problems, such as incorrect event codes). So, if you want your experiments to yield robust, statistically significant, and accurate results, make sure to closely examine the data from every participant at each major stage of processing.

Before going to the next step, change the Current Folder back to the main **Chapter_3** folder.

What’s wrong with Subject 30?

If you load the ERPset for Subject 30 and look at the table at the bottom of the **ERP & Bin Information** panel, you’ll see that there was only one accepted trial in Bin 3 and there were zero accepted trials in Bin 4. And if you load the EEGset (30_N400_preprocessed_filt_elist_bins_be_ar_from_script), you’ll see that this participant blinked a lot. In particular, the participant blinked right around the time of the buttonpress response (event code 201) on almost every trial. As a result, the ERP waveform for Bin 3 was based on an “average” of only one trial, and the aSME value was zero. Bin 4 had no trials, so no ERP waveform could be plotted for that bin, and the aSME value was not a number (NaN). Well over half the trials were also

rejected in Bins 1 and 2, and the EEG waveforms were just generally noisy for this participant. That's why the aSME values were higher than normal for Bins 1 and 2. You can also see this information about artifacts within the EEGset by going to the **Artifact Info & Tools** panel in the **EEG** tab and clicking the **Artifact Summary** button.

When you loaded the ERPset for Subject 30 into ERPLAB, the fact that there were no trials in Bin 4 led to a warning message that was printed in red text in the Matlab Command window (WARNING: bin #4 has flatlined ERPs). You can probably still see it if you scroll up in this window. You probably didn't notice it when it first happened, because the command window is probably obscured by the eStudio window. When you run into a problem (like a bin that doesn't appear to plot properly), you should look at the command window to see if any warning or error messages were printed. That can help you find problems like this.

So, what can we do about this subject? In the published version of the N400 ERP CORE experiment, we used *artifact correction* instead of artifact rejection to deal with blinks. That is, we used a procedure called *independent component analysis* to estimate and remove the part of the signal that was caused by blinking. We rejected trials with blinks only if the blinks happened near time zero, indicating that the eyes were closed when the word was presented (which was rare). Consequently, we were able to include almost all the trials from every participant in our averaged ERP waveforms.

This page titled 3.4: Exercise- "Bad" Data is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 3.4: Exercise- "Bad" Data by Steven J Luck is licensed CC BY 4.0.

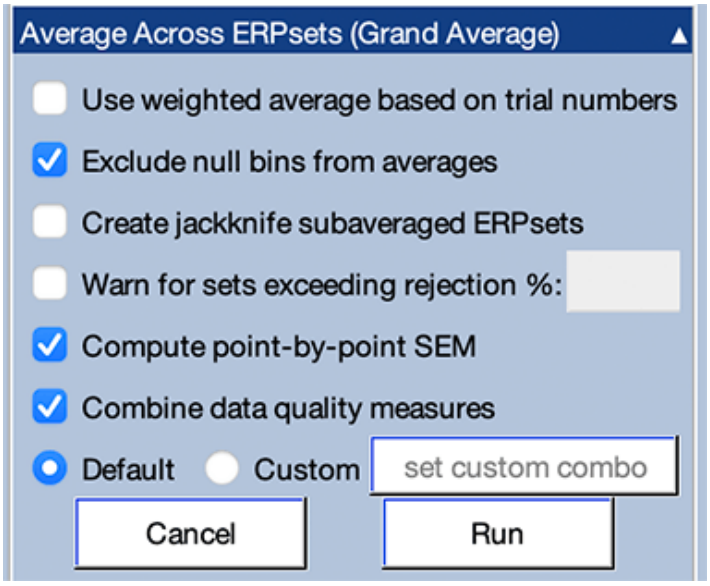
3.5: Exercise- Making a Grand Average

Although it's important to look at the single-participant ERP waveforms, it will be easier to see subtle effects by averaging the waveforms across participants to create *grand average* waveforms for each bin. To do this, first make sure that the ERPsets from all 10 participants from Chapter_3 are loaded (which you can verify by looking at the **ERPsets** panel). If you have an other ERPsets loaded (like 30_N400, the "bad" data from the previous section), select those ERPsets and clear them (by clicking **Clear** in the **ERPsets** panel).

Also, make sure that Chapter_3 is the current folder (by clicking **Current Folder** in the **ERPsets** panel) and select the 10 ERPsets in the **ERPsets** panel.

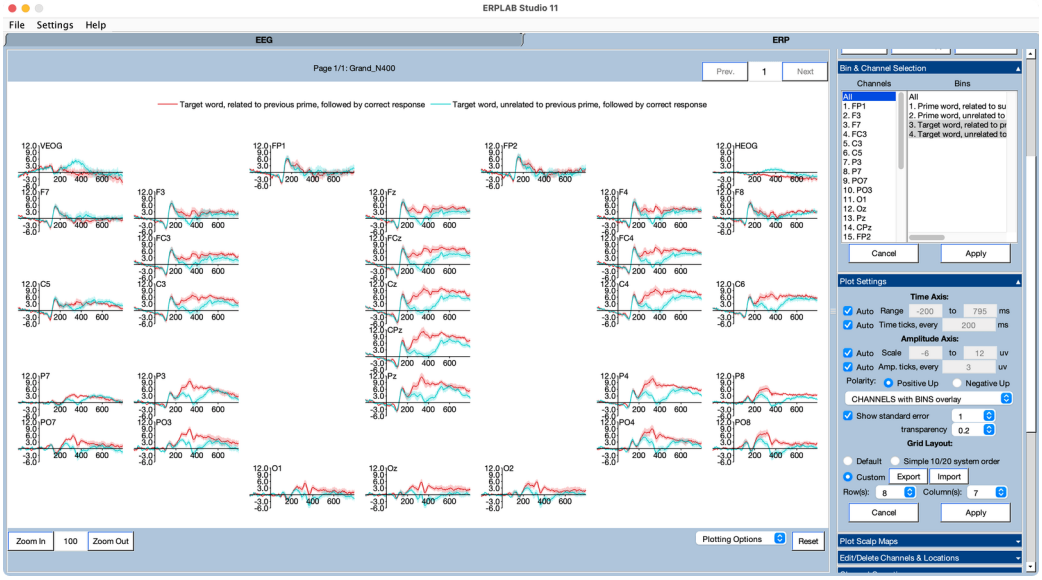
Now go to the **Average across ERPsets (Grand Average**) panel and make sure it is set up as shown in Screenshot 3.4. Select all 10 ERPsets in the **ERPsets** panel and then click **Run** in the **Average across ERPsets** panel to create a grand average of the 10 selected ERPsets. You'll then see the usual window for saving the new ERPset that you've created. Name it **Grand_N400** (and save it as a file named **Grand_N400.erp** so that you have it for the subsequent exercises). You should now see **Grand_N400** in the **ERPsets** panel, and it should be the only selected ERPset (so that you'll see the waveforms from this ERPset).

Screenshot 3.4



Select Bins 3 and 4 from the **Bin & Channel Selection** panel, and make sure that you're using the custom layout specified by **GridLocations_Custom.tsv**. And let's add something new: Tick the box labeled **Show standard error** in the **Plot Settings** panel (and make sure that the transparency level is set to 0.2). You should see something like Screenshot 3.5, with a more negative voltage for the unrelated targets than for the related targets at CPz and surrounding electrode sites. The light shading around the waveforms shows the standard error of the mean (SEM) at each time point (see the box below for more information).

Screenshot 3.5



Now plot Bins 1 and 2 (the prime words). The waveforms for these bins should be lying right on top of each other, with any differences being small relative to the SEM. Remember, these are the bins for primes that are followed by related versus unrelated targets, and unless the participants have ESP, they can't differ as a function of something that happens later. As a result, any differences between them are simply a result of noise.

Finally, take a look at the aSME data quality values for the grand average by going to the View Data Quality Metrics panel and clicking the **Show in a table** button. When you made the grand average, the default settings caused the aSME values from the individual participants to be combined together using something called the *root mean square* (RMS). This is like taking the average of the single-participant aSME values, except that the RMS of the single-participant values is more directly related to the impact of each participant's data quality on the effect size (see Luck et al., 2021).

These aSME values are like standard error values, but they're the standard error of the mean voltage over a 100-ms time window rather than the SEM at a single time point (see the box below). If you look at the table of aSME values, you'll see that the values in the N400 time range (300-400 and 400-500 ms) at the CPz electrode site are between 1.4 and 1.9 μV. That's reasonably small relative to the large difference in mean voltage between the unrelated and related targets. In other words, the data quality is quite good for our goal of detecting differences between these two types of targets.

Plotting the Standard Error

Plotting the standard error of the mean (SEM) at each time point in an ERP waveform, as in Screenshot 3.5, can be helpful in assessing whether the differences between conditions are reasonably large relative to the variability among participants. These standard error values are just like the error bars that you might see in a bar graph. At each time point, the grand average ERP waveform is simply the mean of the voltage values across participants at that time point. The SEM is just the standard deviation (SD) of the single-participant values divided by the square root of the number of participants (which is exactly how the SEM is usually calculated in other contexts).

You can also see the SEM when you plot a single participant's averaged ERP waveforms. In this case, the waveform shows the mean across trials rather than the mean across participants, and the SEM reflects the SD across trials divided by the square root of the number of trials (rather than the SD across participants divided by the square root of the number of participants).

Although the SEM can be useful, it has some downsides. First, imagine that the overall voltage at 400 ms was much more positive in some participants than others, leading to quite a bit of variability in the voltage for each condition. Because of this variability, the SEM for each waveform would be quite large at 400 ms. This would make it look like the difference in means between conditions was small relative to the SEM. However, imagine that the voltage at 400 ms was exactly 3 μV more negative for unrelated targets than for related targets in every participant (i.e., the experimental effect is extremely consistent across participants). Thus, the SEM is large even though the difference for each participant was extremely consistent. In

behavioral research, this problem is often addressed by using the within-subjects SEM (Cousineau, 2005; Morey, 2008). ERPLAB doesn't have this version of the SEM built in, but you can achieve the same result by making a difference wave between the conditions (e.g., unrelated targets minus related targets) and getting the SEM of the difference wave. This is exactly what was done in the grand averages from the full study (see Figure 2.1C in Chapter 2).

Another downside of the SEM is that it can be very large if there is a lot of high-frequency noise in the data, even though this noise has minimal impact when we quantify the N400 as the mean voltage between 300 and 500 ms (as we will do later in this chapter). The aSME value provided in our Data Quality measures does not have this downside, because it provides the standard error of the mean voltage over a time period rather than the standard error of the values at individual time points. See Luck et al. (2021) for a more detailed discussion.

One last thing: Open up the **ERP & Bin Information** panel. In the table at the bottom of the panel, you can see that there are hundreds of trials per bin. When a grand average is made, we sum together the trials across the ERPsets, so this table shows you the total number of trials that contributed to the grand average waveforms.

This page titled 3.5: Exercise- Making a Grand Average is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 3.5: Exercise- Making a Grand Average by Steven J Luck is licensed CC BY 4.0.

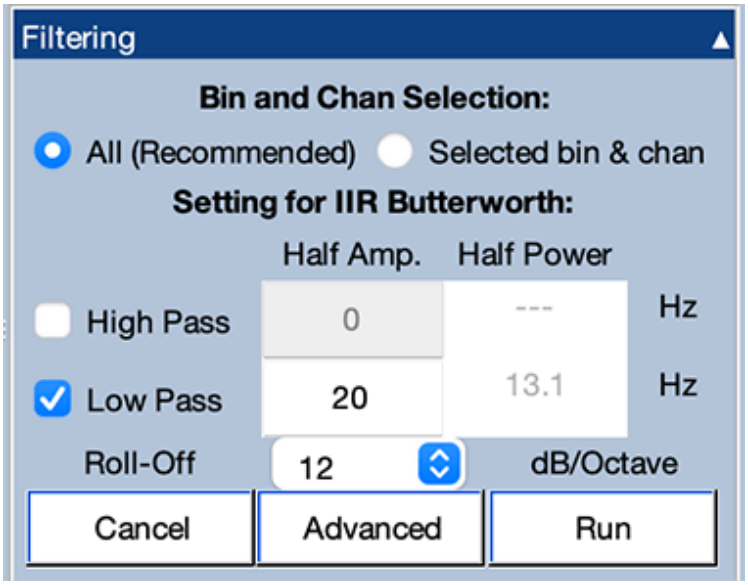
3.6: Exercise- Low-Pass Filtering

During the EEG preprocessing steps, the low-frequency voltage drifts in the continuous EEG data were reduced by applying a high-pass filter. However, there is also some high-frequency noise (small but rapid deflections in the waveforms). You can see this at the most anterior scalp sites in the grand average waveforms (especially if you specify a zoom factor of 200 using the zoom controls in the bottom left corner of the eStudio window). The waveforms just look a little fuzzy.

My lab takes great pains to reduce the major sources of high-frequency noise in the recording environment (e.g., induced voltages from electrical devices in the environment and muscle activity). As a result, there isn't a lot of high-frequency noise in the grand averages. But there's a little, and you'll see much more of it in other experiments (especially in the single-participant waveforms). The present exercise will show you how to filter out this high-frequency noise using a *low-pass filter*. We'll apply it to the grand average ERP waveform, but you could instead apply it to the single-subject ERPs, the epoched EEG data, or even the continuous EEG data (see Chapter 7 in Luck, 2014 for information about when different filters should be applied).

Make sure that the **Grand_N400** ERPset is still loaded and selected in the **ERPsets** panel, and then go to the **Filtering** panel. Set it up as shown in Screenshot 3.6, which should mainly involve setting the low-pass cutoff to 20 Hz. Then click **Run** to apply the filter. You can name the new ERPset **Grand_N400_filt**.

Screenshot 3.6



Take a look at the filtered data. The first thing you'll notice is that the standard error is no longer shown. And if you look in the **Plot Settings** panel, you'll see that the option for plotting the standard error is grayed out. When you filter the data, the original standard error values are no longer valid—they're the standard error of the unfiltered mean voltage at each time point, not the standard error of the filtered values. Similarly, the original data quality metrics are no longer valid, so they're no longer available in the **View Data Quality Metrics** panel. In a moment, we'll see how to get standard errors for filtered waveforms.

Now compare the filtered waveforms to the original unfiltered waveforms in **Grand_N400**. The filtered waveforms look a little smoother than the unfiltered waveforms (but the difference is small because we had such a small amount of high-frequency noise in the original data). In Chapter 4, we'll take a closer look at how filtering works (and how it can distort your data if you're not careful).

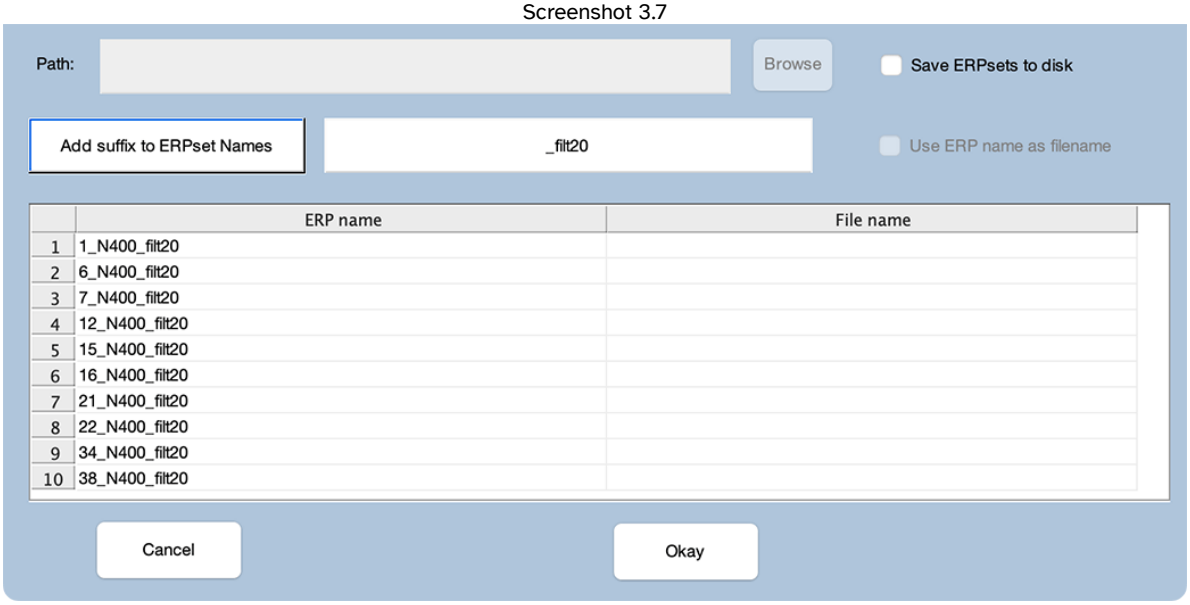
Filtering the Single-Participant ERPsets

To obtain valid standard error values for a filtered grand average, you need to filter the single-participant ERPsets before making the grand average. Fortunately, this is quick and easy in ERPLAB Studio.

Go to the **ERPsets** panel and select the ERPsets for the 10 individual participants (leaving out the grand average ERPsets). Now go to the Filtering panel (which should still be set up as shown in Screenshot 3.6) and click **Run**.

You're going to create a new filtered ERPset for each of the 10 selected ERPsets, so the window for naming the new ERPsets has 10 ERP names in it. ERPLAB has simply added a suffix of **_filt** onto each of the original ERPset names. Let's try changing this suffix by changing the suffix in the text box from **_filt** to **_filt20** and clicking the **Add suffix to ERPset Names** button (as shown in Screenshot 3.7). Click **Okay** to save the new ERPsets.

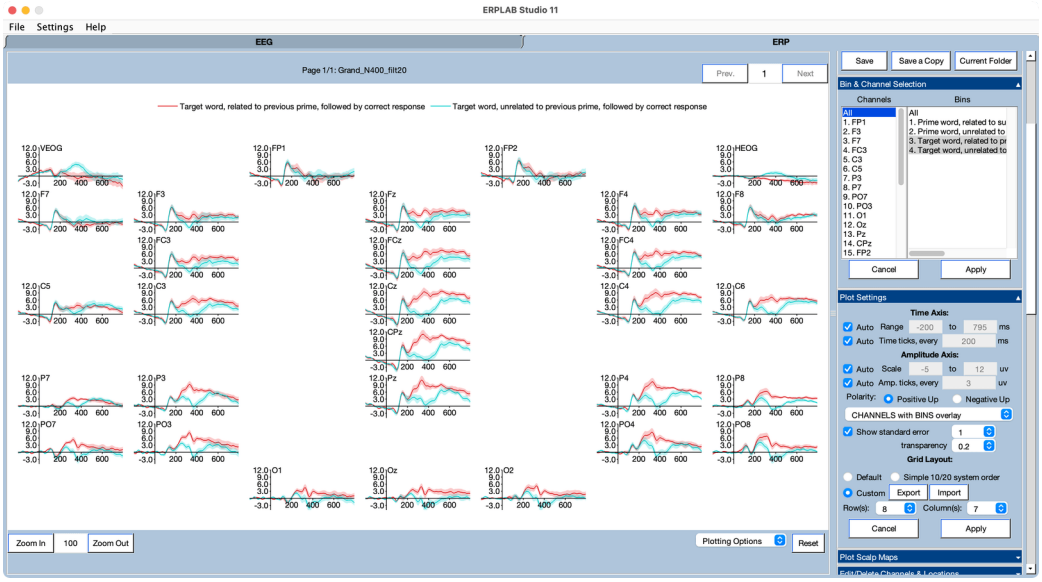
Screenshot 3.7



You should now see the 10 filtered ERPsets in the **ERPsets** panel, and they should all be selected. Now go back to the **Average Across ERPsets** panel and click **Run**. Name the new ERPset **Grand_N400_filt20**. Pretty efficient, isn't it?

Finally, select **Show standard error** in the Plot Settings panel (and click **Apply**). You should now see the standard error shading around the filtered grand average ERPs.

Screenshot 3.8



If you look in the **View Data Quality Metrics** panel, you'll see that we still don't have data quality metrics for Grand_N400_filt20. This is because filtering the single-participant ERPsets invalidated the data quality metrics for those ERPsets, and the data quality metrics in a grand average are created by combining the data quality metrics from the single-participant ERPsets. This isn't a

problem for the standard error values, because the standard error at a single time point in a grand average is not computed from the single-participant standard error values. Instead, it is computed as the standard deviation across the single-subject waveforms divided by the square root of the number of subjects.

To get the data quality metrics for filtered data, you would need to filter the single-trial EEG data for each participant prior to averaging. The data quality metrics would then be calculated when the single-participant averaged ERPs are computed. And they would then be available when the grand average is computed.

This page titled 3.6: Exercise- Low-Pass Filtering is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 3.6: Exercise- Low-Pass Filtering by Steven J Luck is licensed CC BY 4.0.

3.7: Exercise - Scoring N400 Amplitude

Our next step is to quantify the amplitude of the N400 component for the unrelated and related target words so that we can perform statistical analyses (which we will do in the next exercise). We need N400 amplitude values from each individual participant, so make sure that you still have the 10 single-participant ERPsets loaded in the **ERPsets** panel. We will use the unfiltered ERPs, so you can clear the filtered waveforms (and grand averages) if they're still present in the **ERPsets** panel.

We'll quantify the amplitude of the N400 as the mean voltage between 300 and 500 ms, which is the prototypical N400 measurement window. Let's be concrete about what this means. These data are sampled at 200 Hz, which means that there is one voltage value every 5 ms. This gives us 21 values between 300 and 500 ms (not 20, because we include the value at 300 ms and the value at 500 ms, plus the values between them). We calculate the mean voltage by summing together these 21 values and dividing by 21. It's that simple. If you want to know why we quantify the amplitude of an ERP component this way, see Chapter 9 in Luck (2014). We'll apply this procedure to the averaged ERP waveforms from each participant, once for the unrelated targets and once for the related targets. Initially, we'll just do this for the CPz electrode site.

"Scoring" Versus "Measuring" the Amplitude or Latency of an ERP Component

When researchers apply an algorithm to an ERP waveform to quantify the amplitude or latency of an ERP component, we often say that we're *measuring* the amplitude or latency. But I don't like the term "measuring" in this context. An ERP component is a hypothetical entity in the brain, and we're obtaining a value from a scalp signal that typically consists of a mixture of many components. As a result, it's not really a measurement of the component. Also, it seems weird to use the term "measuring" when we're taking values that were already measured (the EEG voltages) and recombining them in a new way. So, it seems perfectly natural to say that we're "measuring the EEG," but it seems odd to say that we're "measuring the amplitude of the N400 component."

I prefer to use the term *score*. That is, we're "scoring the amplitude of the N400 component," and we obtain "N400 amplitude scores." I started using this terminology just a couple years ago, so you'll see the term *measurement* rather than *score* in many of my writings and in ERPLAB.

Start by going to the **Measurement Tool** panel. You'll see many blank text boxes for entering the parameters telling the tool what you want to score.

We're going to set it up as shown in Screenshot 3.9. First, select **Mean Amp** from the **Type** menu at the top of the panel. This tells the tool that we want to quantify the mean amplitude across a time range (across a *measurement window*). As you can see, this menu allows you to select many other waveform features to score, such as positive and negative peak amplitudes and peak latencies.

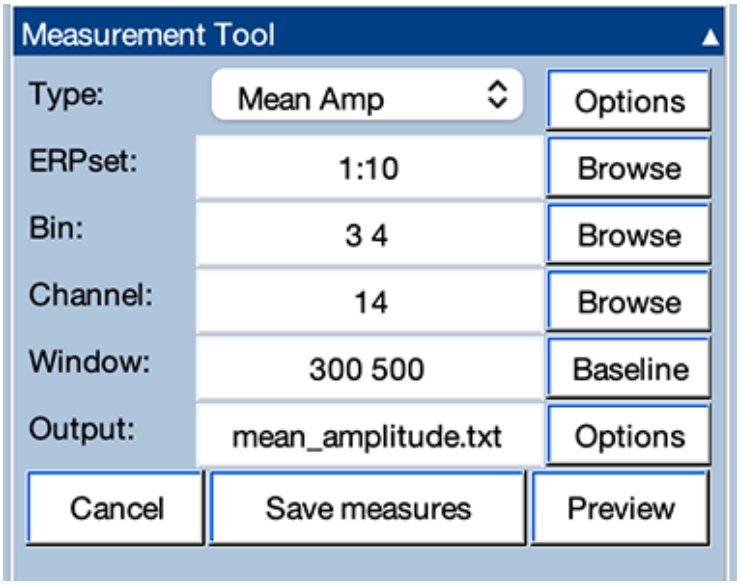
Next you should tell the tool that we want to obtain the scores from ERPsets 1-10 by typing **1:10** in the **ERPset** text box. The string **1:10** is Matlab's way of specifying that we want all values between 1 and 10. We could instead just put **1 2 3 4 5 6 7 8 9 10** into the text box.

We want to obtain the scores from Bins 3 and 4, so type **3 4** in the **Bin** text box. To indicate that we want to measure from CPz, click the **Browse** button on the **Channel** line, and select **CPz** from the list of channels that appears (followed by **OK**). You should now see **14** in the **Channel** text box (because CPz is Channel 14).

Next, type **300 500** (with a space between the two numbers) in the **Window** text box to indicate that we want to use a measurement window of 300–500 ms.

Finally, enter **mean_amplitude.txt** in the **Output** text box; this is the name of the file we will use to store the scores. Make sure that everything is set up as shown in the screenshot. Then click on the **Options** button next to the filename and make sure that the option labeled **Include bin label as well as bin index** is not set.

Screenshot 3.9

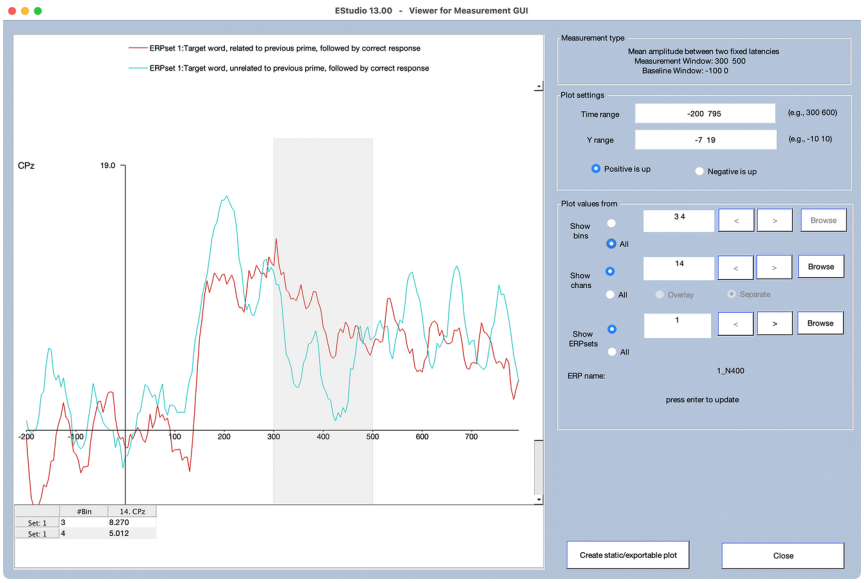


Most people would just click **Save measures** at this point to compute the mean amplitude scores and save them in the output file. But there is another very important step, which is to click the **Preview** button. This will allow you to see the scores for each ERP waveform, which is very important for making sure that the scoring procedure is working in a sensible manner. This is especially important for peak measures, because you may see that the peaks are outside of the measurement window for some of the waveforms.

The Preview window is shown in Screenshot 3.10. I've set it to show both Bin 3 and Bin 4. Note that it shows the measured values in the table at the bottom of the window.

You should scroll through the 10 ERPsets with the **<** and **>** buttons on the **Show ERPsets** line to see how the scoring is working for each of the participants. From my perspective, everything looks like it's working fine. Once you're done, close the Preview window and go back to the Measurement Tool panel. Then you can click **Save measures** to compute the amplitude scores and save them in the output file.

Screenshot 3.10



If you look in the Current Folder section of the Matlab GUI, you'll see that a file named **mean_amplitude.txt** has been created. Double-click on it to open it in the Matlab text editor. It should look something like this:

Screenshot 3.11

bin3_CPz	bin4_CPz	ERPset
8.270	5.012	1_N400
10.926	4.207	6_N400
9.757	0.174	7_N400
12.101	1.005	12_N400
14.017	10.293	15_N400
16.873	5.894	16_N400
2.293	-5.868	21_N400
7.389	0.719	22_N400
9.778	1.625	34_N400
13.207	0.738	38_N400

The first column has the amplitude scores from Bin 3 (related targets) and the second column has the scores from Bin 4 (unrelated targets). The third column is the name of the ERPset, which tells you which subject was measured on that line. You can see that every single participant has a more negative (less positive) voltage for the unrelated trials than for the related trials.

This page titled 3.7: Exercise - Scoring N400 Amplitude is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 3.7: Exercise - Scoring N400 Amplitude by Steven J Luck is licensed CC BY 4.0.

3.8: Exercise- Simple Statistical Analysis of N400 Data

Now we're going to perform a simple statistical analysis of the N400 amplitude scores that you obtained in the previous exercise. We have two amplitude scores for each participant, one for related targets and one for unrelated targets, and we want to know if the scores are significantly different for these two experimental conditions. The simplest way to do this is with a paired *t* test.

I used the free JASP statistical package to run the *t* test, but you can use whatever package you find comfortable. Make sure you specify a paired *t* test rather than an independent-samples *t* test. The results are shown in Screenshot 3.12. Before you look at the *t* and *p* values, you should always look at the descriptive statistics. Once we get to more complex analyses, it will be really easy to make mistakes in the statistical analysis. The most common mistake is to incorrectly specify which variable is in which column of the data file. For example, you might think that the unrelated and related targets are stored in the first and second columns, respectively, reversing the actual order. This kind of error becomes both more likely and more likely to lead to incorrect conclusions when your design has several factors, and each row of the data file has a dozen or more columns. By comparing the group means from the statistical analysis to the grand average waveforms, you can often detect these errors.

Screenshot 3.12

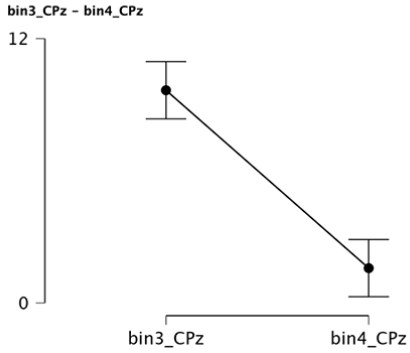
Paired Samples T-Test

Measure 1		Measure 2	t	df	p	Cohen's d	SE Cohen's d
bin3_CPz	-	bin4_CPz	9.946	9	< .001	3.145	0.509

Note. Student's t-test.

Descriptives

	N	Mean	SD	SE	Coefficient of variation
bin3_CPz	10	9.657	3.861	1.221	0.400
bin4_CPz	10	1.583	3.911	1.237	2.471



If you look at the group means in Screenshot 3.12, you'll see a mean of 9.657 µV for the related targets (Bin 3) and 1.583 µV for the unrelated targets (Bin 4). Those values at least approximately match what you can see for the CPz channel from 300-500 ms in the grand average waveforms shown in Screenshot 3.4 in Section 3.5.

Now that we've verified that the descriptive statistics look correct, we can look at the *t* and *p* values. The effect was significant with a p value of < .001, and the effect size (Cohen's *d*) of 3.145 was enormous. This effect size indicates that the difference between the group means for related and unrelated targets was 3.145 times as large as the standard deviation of the scores within a condition. You won't find effects this large in most experiments, but the N400 ERP CORE experiment was carefully designed to maximize the experimental effects. Also, I chose 10 participants with really clear effects for the exercises in this chapter; the effect size was "only" 2.33 in the full sample of 40 participants (but this was still a huge effect size).

Limits on Comparing Descriptive Statistics with Grand Average Waveforms

When we score the amplitude of an ERP component as the mean voltage in a measurement window, we can directly compare the group mean values from the statistical analysis with the grand average ERP waveforms. This is because this scoring method is a *linear operation* (for a definition and more information, see the Appendix in Luck, 2014). The order of operations

does not matter for linear operations. This means that we can obtain our mean amplitude score from the single-subject waveforms and then compute the mean of these scores, and we will get exactly the same value that we would obtain by scoring the mean amplitude from the grand average waveform.

Unfortunately, most other scoring methods are not linear. For example, the peak amplitude in a given time window is not linear. If we obtain the peak amplitude from the single-subject waveforms and then compute the mean of these scores, the result will not be the same as the peak amplitude of the grand average waveform. However, you should still compare the group means from your statistical analysis with the grand average waveforms. If there is a large mismatch, then you may have made an error in specifying the order of variables in your statistical analysis, or your grand average waveform may not adequately represent what is happening at the single-subject level. In either case, you want to know!

This page titled 3.8: Exercise- Simple Statistical Analysis of N400 Data is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 3.8: Exercise- Simple Statistical Analysis of N400 Data by Steven J Luck is licensed CC BY 4.0.

3.9: Exercise- A More Complex Analysis

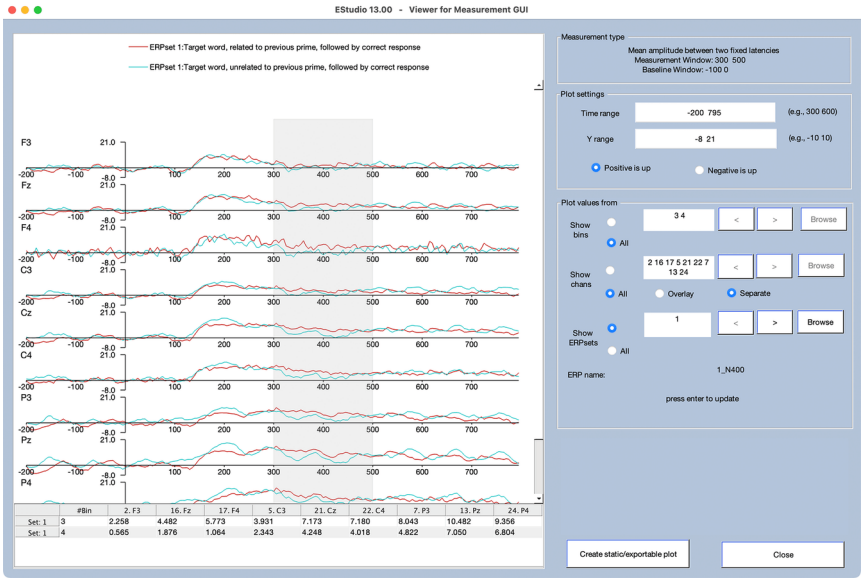
In this exercise, we're going to repeat the N400 analysis from the previous exercises, but we're going to make it more complex by measuring and analyzing the N400 at multiple electrode sites (F3, Fz, F4, C3, Cz, C4, P3, Pz, and P4). [We won't include CPz in these analyses because we don't have electrodes at CP3 and CP4 and we don't want an unbalanced design.]

We'll organize these 9 electrode sites as a 3x3 matrix with two factors: laterality (left hemisphere, midline, and right hemisphere) and anterior-posterior (frontal, central, and parietal). When we combine these two electrode factors with the relatedness factor, this will give us a factorial design with three total factors (related-unrelated, laterality, and anterior-posterior).

Make sure that the Measurement Tool is set it up exactly as in the previous exercise, except enter **2 16 17 5 21 22 7 13 24** in the **Channel** text box (these are the F3, Fz, F4, C3, Cz, C4, P3, Pz, and P4 channels, in the correct order; you can verify the channels by clicking the **browse** button on the **Channel** line). You should also change the name of the output file to **mean_amplitude_multiple_channels.txt**.

Click **Preview** to make sure that everything looks OK. I recommend setting it up to show all bins and all channels, as shown in Screenshot 3.13.

Screenshot 3.13



Once you've verified that everything looks fine, close the Preview window and click **Save measures** in the Measurement Tool.

Now open the **mean_amplitude_multiple_channels.txt** file in the Matlab text editor. The text editor doesn't deal with the tabs very well, so you might want to import the file into Excel instead. Now we have 19 columns: 9 channels x 2 bins plus the ERPset column. Unfortunately, the channels are in the order that they appear in the ERPsets, which is not very convenient. If you're not sure whether the amplitude scores are correct, you can launch the Measurement Tool again and use the Viewer to see the single-subject scores.

Once you've verified that the scores are correct, you can enter the data into a statistical analysis. You should use a 3-way repeated-measures ANOVA with factors of relatedness, laterality, and anterior-posterior. I ran this ANOVA in JASP, and the results are shown in Screenshot 3.14.

Screenshot 3.14

Within Subjects Effects ▼

Cases	Sum of Squares	df	Mean Square	F	p
Related vs Unrelated	1200.182	1	1200.182	118.149	< .001
Residuals	91.424	9	10.158		
Left–to–Right Electrode Site	46.522	2	23.261	12.398	< .001
Residuals	33.770	18	1.876		
Anterior–to–Posterior Electrode Site	342.871	2	171.435	21.131	< .001
Residuals	146.031	18	8.113		
Related vs Unrelated ✳ Left–to–Right Electrode Site	26.640	2	13.320	17.019	< .001
Residuals	14.088	18	0.783		
Related vs Unrelated ✳ Anterior–to–Posterior Electrode Site	123.650	2	61.825	30.925	< .001
Residuals	35.986	18	1.999		
Left–to–Right Electrode Site ✳ Anterior–to–Posterior Electrode Site	8.964	4	2.241	1.236	0.313
Residuals	65.261	36	1.813		
Related vs Unrelated ✳ Left–to–Right Electrode Site ✳ Anterior–to–Posterior Electrode Site	1.957	4	0.489	0.890	0.480
Residuals	19.796	36	0.550		

Note. Type III Sum of Squares

Between Subjects Effects

Cases	Sum of Squares	df	Mean Square	F	p
Residuals	1950.363	9	216.707		

Note. Type III Sum of Squares

Descriptives

Related vs Unrelated	Left–to–Right Electrode Site	Anterior–to–Posterior Electrode Site	N	Mean	SD	SE	Coefficient of Variation
Related	Left	Parietal	10	7.274	2.808	0.888	0.386
		Frontal	10	2.220	4.496	1.422	2.025
		Central	10	5.764	3.545	1.121	0.615
	Midline	Parietal	10	8.992	3.725	1.178	0.414
		Frontal	10	3.970	3.709	1.173	0.934
		Central	10	8.408	3.795	1.200	0.451
	Right	Parietal	10	8.387	3.328	1.053	0.397
		Frontal	10	3.669	3.682	1.164	1.004
		Central	10	8.329	4.367	1.381	0.524
Unrelated	Left	Parietal	10	2.059	3.378	1.068	1.641
		Frontal	10	0.271	4.662	1.474	17.188
		Central	10	0.656	3.894	1.231	5.940
	Midline	Parietal	10	2.435	4.186	1.324	1.719
		Frontal	10	0.472	3.768	1.192	7.986
		Central	10	0.895	3.643	1.152	4.071
	Right	Parietal	10	1.716	3.751	1.186	2.186
		Frontal	10	0.627	3.681	1.164	5.874
		Central	10	1.405	3.812	1.206	2.713

Again, start by looking at the descriptive statistics and make sure they match the grand average waveforms in Screenshot 3.4. For example, in both cases the amplitude for the related trials increases from the frontal to the central and parietal channels, and it tends to be larger for the midline and right-hemisphere channels than for the left-hemisphere channels. You can also see the basic N400 effect in both the grand average waveforms and the group means: the voltage is more negative (less positive) for the unrelated targets than for the related targets.

I actually messed things up the first time I set up the ANOVA, but I could tell that something was wrong from the means for each cell. I knew that the voltages for the related targets should be larger at Cz and Pz than at Fz (because of what I saw in the grand average waveforms), but this wasn't what the table of descriptive statistics showed. I then fixed the problem and ran the ANOVA again.

If you look at the *F* and *p* values in Screenshot 3.14, you'll see that the main effect of relatedness (related vs unrelated) was significant at the $p < .001$ level. The laterality and anterior-posterior main effects were also significant, and these factors both interacted significantly with relatedness. That is, the difference between related and unrelated words was largest at the sites where the voltage was largest. This pattern of interactions is exactly what would be expected given the multiplicative relationship between the magnitude of an internal ERP generator and the observed scalp distribution (see Chapter 10 in Luck, 2014).

You've now completed a fairly sophisticated analysis of the N400 experiment. Congratulations! That was a lot of steps, and it took us two chapters to get to this point.

However, I should note that I don't generally recommend the analysis approach used in this exercise, which involves scoring a component from multiple sites and including electrode site factors in the statistical analysis. The reasoning is described in the text box below. Sometimes it is justifiable, such as when your scientific hypothesis leads to a prediction of different effects over the left and right hemispheres. But unless you have a really important reason to compare the effects across electrode sites, it's usually better to limit your analysis to a single site or create a waveform that averages across multiple sites (see Zhang & Kappenman, 2024 for evidence that averaging across a cluster of electrodes is usually the best option). We'll explore the latter option in the next exercise.

Minimizing the Number of Factors in an Analysis

The problem with including one or more electrode site factors is that it leads to a large number of statistical tests, increasing the likelihood that you'll get one or more significant effects that are a result of random noise in the data. Look at the table of statistics at the top of Screenshot 3.9—how many p values do you see? Seven!

> Ordinarily, you would expect a 5% probability that an effect will be significant ($p < .05$) when the null hypothesis is true. However, if the null hypothesis were true for all seven of these tests, the chance that one or more would be significant ($p < .05$) would be greater than 30%!
>
> As we increase the number of factors in an ANOVA, the number of main effects and interactions skyrockets, and the odds that one or more will be significant by chance become extremely high (Cramer et al., 2015; Frane, 2021). For example, in a 5-way ANOVA, you are more likely than not to obtain a significant-but-bogus-effect even if all effects are in reality null. As a result, it is difficult to trust the results of such analyses. My general advice is therefore to minimize the number of factors (see Luck & Gaspelin, 2017 for a detailed discussion).

This page titled 3.9: Exercise- A More Complex Analysis is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 3.9: Exercise- A More Complex Analysis by Steven J Luck is licensed CC BY 4.0.

3.10: Exercise- ERP Channel Operations

In the next two exercises, we'll take a look at two kinds of processing operations that are commonly applied to ERPs in which either the bins or the channels are mathematically recombined. For example, we could average together two bins or two channels. These kinds of operations are carried out with **Bin Operations** and **Channel Operations**, respectively.

Let's start by applying channel operations to the 10 ERPsets we've been analyzing in this chapter. In the previous exercise, I noted that it is often better to average across channels rather than to include channels as a factor in the statistical analysis. In the present exercise, we'll average across the three midline channels where the N400 is typically largest (Cz, CPz, and Pz).

Before we look at how the averaging process is implemented in ERPLAB, let's think for a minute about how you would compute the average of these three channels by hand. The average of three values is just the sum of those three values divided by three. So, to compute the average, you would use this equation:

$$\frac{Cz + CPz + Pz}{3} \tag{3.10.1}$$

This is how you perform channel operations in ERPLAB. That is, you specify an equation that describes exactly what you want to compute, and ERPLAB computes it for you. The only difference is that you need to use the channel numbers rather than the channel names.

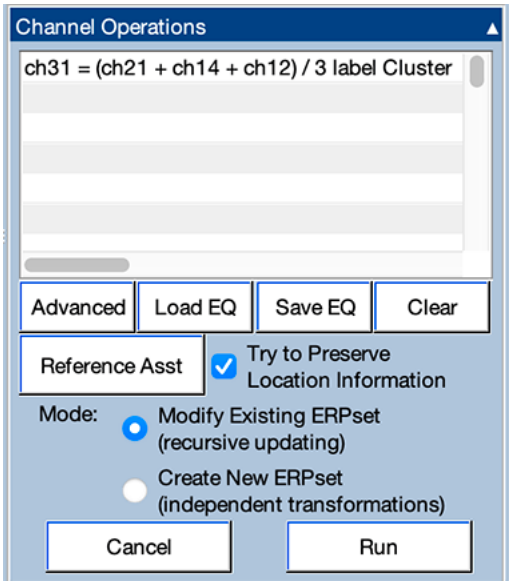
Let's give it a try. If necessary, load the 10 ERPsets into ERPLAB and make sure that all 10 are selected in the ERPsets panel. Now open the **Channel Operations** panel.

As shown in Screenshot 3.15, the top section of the panel is a text window that allows you to write one or more equations. We currently have 30 channels (which you can see in the **Bin & Channel Selection** panel), so the new channel we will create will be channel 31. We'll call this new channel **Cluster**. To make this happen, we will use the equation:

ch31 = (ch21 + ch14 + ch12) / 3 label Cluster

It's just like the previous equation, except that we use channel numbers instead of channel names, and we add **label Cluster** to the end of the equation to indicate the name of the new channel. Type this equation into the top line of the text box in the panel (or copy and paste it). Make sure that all the other parts of the panel match Screenshot 3.15 and then click **RUN**.

Screenshot 3.15

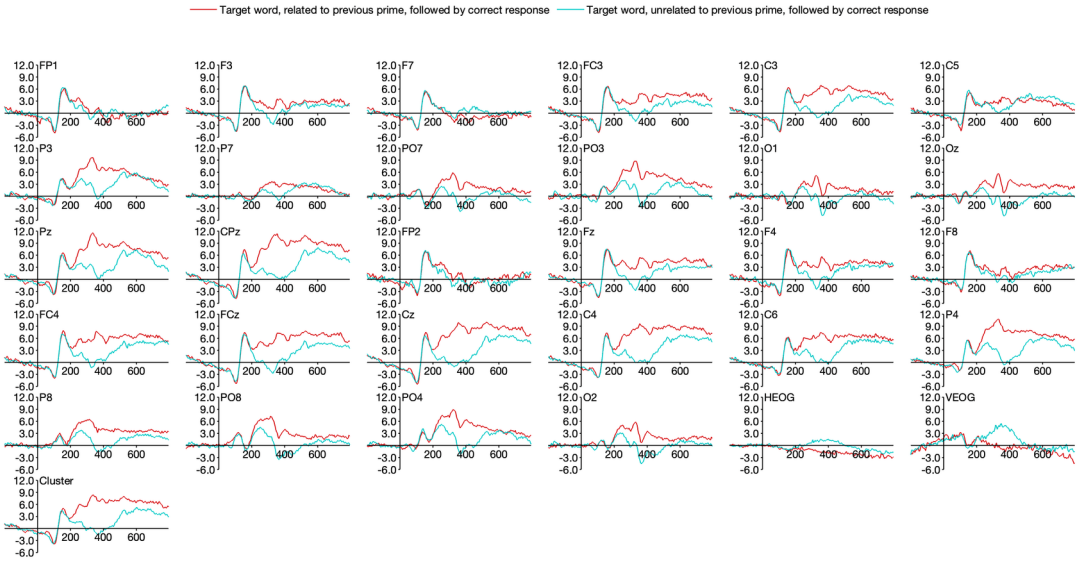


Because we're modifying a set of existing ERPsets rather than creating new ERPsets, you won't see a window for saving the ERPsets. However, if you look at the **Bin & Channel Selection** panel, you can see that we now have a Channel 31 named **Cluster**.

You won't see this new channel in the plotting region. This is because we've defined a custom channel layout (GridLocations_Custom.tsv), and it doesn't include Channel 31. To see the new channel, go to **Plot Settings** and change the Grid Layout to **Default** (and click **Apply**). If you still don't see the new channel, make sure that all channels are selected in the **Bin & Channel Selection** panel.

Now let's make a grand average of the updated ERPsets by going to the **Average Across ERPsets (Grand Average)** panel and clicking **Run**. You should now see something like the plot shown in Screenshot 3.16 (assuming that you still have Bins 3 and 4 selected in the **Bin & Channel Selection** panel).

Screenshot 3.16



Note that the **EEG** tab also contains a **Channel Operations** panel, which operates on EEGsets (continuous or epoched) rather than on averaged ERP data. Ordinarily, it is best to create cluster channels in an EEGset prior to averaging, so that the data quality metrics are computed during the averaging process.

This page titled 3.10: Exercise- ERP Channel Operations is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 3.10: Exercise- ERP Channel Operations by Steven J Luck is licensed CC BY 4.0.

3.11: Exercise- ERP Bin Operations

Now that we've seen how to combine channels with **Channel Operations**, let's see how to combine bins using **Bin Operations**. One of the most common reasons to combine bins is to make a *difference wave*, in which we subtract one bin from another. In an N400 experiment, for example, we can isolate the brain activity that differs between related and unrelated target words by constructing an unrelated-minus-related difference wave.

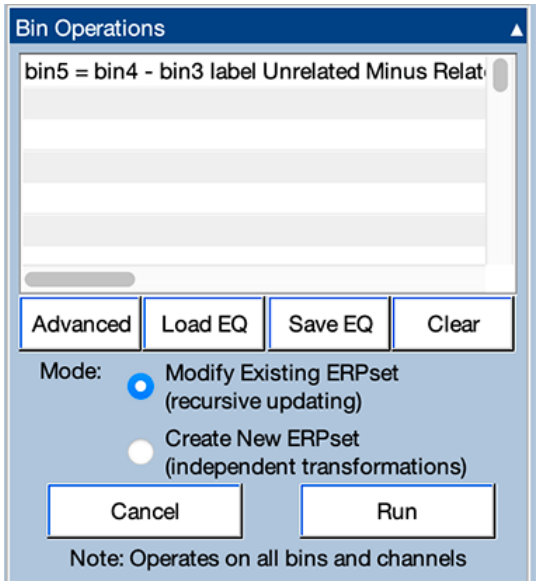
Our ERPsets currently have 4 bins, with Bin 3 being related targets and Bin 4 being unrelated targets. To create an unrelated-minus-related difference wave as Bin 5, we use the following equation:

bin5 = bin4 - bin3 label Unrelated Minus Related Target

To get started, clear the grand average ERPset from the previous exercise (using the **Clear** button in the **ERPsets** panel) and select the 10 single-participant ERPsets.

Now open the **Bin Operations** panel. Just as with Channel Operations, you will see a set of lines for writing equations at the top of the panel. Type (or copy and paste) the equation into the top line of the panel and make sure that everything else is set up as shown in Screenshot 3.17. Then click **Run** to perform the operation on the selected ERPsets.

Screenshot 3.17

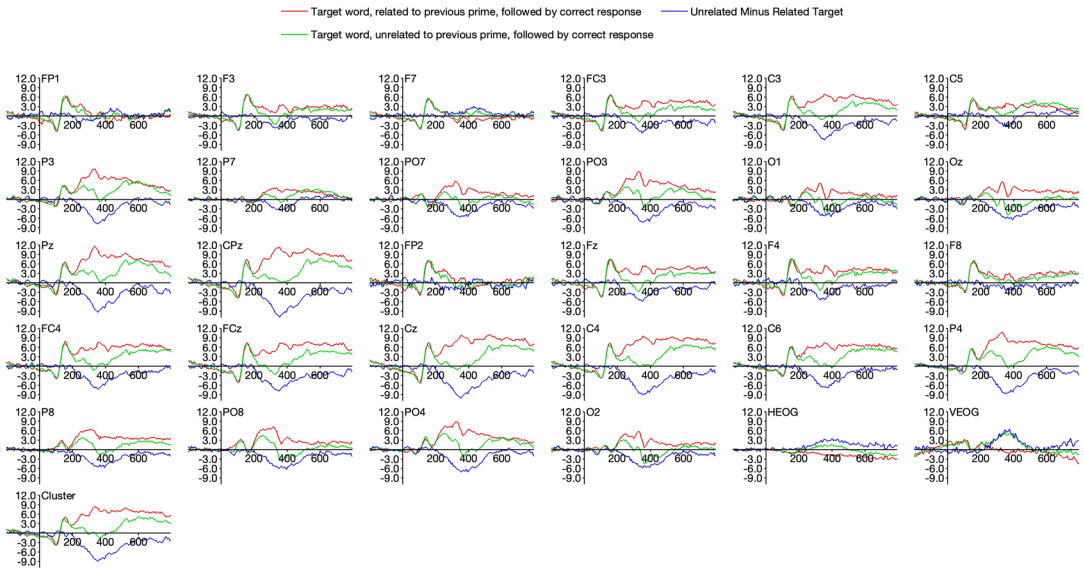


If you look in the **Bin & Channel Selection** panel, you'll see that you now have a Bin 5. If you select Bins 3, 4, and 5 (and click **Apply**), you'll be able to see how the difference wave (Bin 5) is related to the two "parent" waves (Bins 3 and 4).

Now let's make a grand average of the updated ERPsets by going to the **Average Across ERPsets (Grand Average)** panel and clicking **Run**. You should now see something like the plot shown in Screenshot 3.18.

You can see that the voltage in Bin 5 at a given time point is the difference between the voltages for Bins 3 and 4 at that time point. Note that the difference wave subtracts away any brain activity that is identical for the related and unrelated targets, such as the negativity at ~100 ms and the positivity at ~180 ms. All that remains is the brain activity that differentiates between the related and unrelated targets. This is an excellent way of isolating the brain activity of interest from the ERP waveforms. If you read my general book on ERPs (Luck, 2014), you'll see that I'm a big fan of using difference waves to isolate specific brain responses.

Screenshot 3.18



Lost Information as a Result of Bin and Channel Operations

The original EEG data files that we used when we began processing this experiment contained information about the 3-dimensional locations of the individual electrode sites. When you use **Channel Operations** to create a new channel, ERPLAB has no way of knowing the 3-dimensional location that should be used for this channel. And if you modify a channel, ERPLAB isn't smart enough to know whether the original channel location is still valid. To avoid making assumptions that might turn out to be incorrect, ERPLAB discards the channel information for any new or changed channels when you perform **Channel Operations**. (If you check the **Try to preserve location information** box in the GUI, ERPLAB will make a guess about the locations for changed locations, which usually works pretty well.)

The data quality information (including the SEM at each time point) is also lost when you perform Channel Operations. ERPLAB is not smart enough to know how the data quality should be updated for new or modified channels. Similarly, ERPLAB isn't smart enough to estimate the data quality when you perform **Bin Operations**. You can get the data quality information by applying Channel Operations on the EEG, prior to averaging. However, you can't perform Bin Operations on single-trial EEG data, so this doesn't help you get data quality information for difference waves. Instead, you need to use a method called *bootstrapping* to estimate the data quality for a difference wave (see Luck et al., 2021).

The bottom line is that if information about channel locations or data quality is missing, it is likely a result of Bin Operations or Channel Operations.

This page titled 3.11: Exercise- ERP Bin Operations is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 3.11: Exercise- ERP Bin Operations by Steven J Luck is licensed CC BY 4.0.

3.12: Review of Processing Steps

To review, here are the steps we carried out in this chapter:

- Loaded the single-participant ERPsets for each of our 10 participants
  - I provided a script that makes this faster
- Examined the number of accepted and rejected trials, ERP waveforms, and data quality measures for each participant to check for problems
- Made a grand average from our 10 participants
- Applied a low-pass filter to the ERPsets to attenuate the high-frequency noise
  - We did this first for the grand average, and then for the single-participant waveforms
- Obtained N400 amplitude "scores" (mean voltage from 300-500 ms) for the related and unrelated targets in each participant's averaged ERP waveforms
  - We used the Preview window to verify that the scoring process was working correctly
- Performed a simple statistical analysis with the data from a single channel and a more complex analysis with the data from multiple channels
  - The N400 amplitude scores were exported into a text file and then imported into a statistical package
  - ERPLAB does not perform statistical analyses—we did not want to "reinvent the wheel," and it is difficult to anticipate every possible statistical analysis someone would want to perform
  - We also compared the table of means from the statistical analyses with our grand average ERP waveforms to make sure that the analysis was performed correctly
- Used Channel Operations to create a new "cluster" channel that was an average of three of the original channels
- Used Bin Operations to create a difference wave by subtracting one bin from another

This page titled 3.12: Review of Processing Steps is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 3.12: Review of Processing Steps by Steven J Luck is licensed CC BY 4.0.

3.13: Matlab Scripts For This Chapter

I've provided two scripts in the data folder for this chapter. One is called **load_erpsets.m**, and it loads the 10 ERPsets into eStudio. The second is called **preprocess_and_average.m**, and it does the preprocessing and averaging for all 10 participants. The third is called **postprocessing.m**, and it carries out all the remaining steps in this chapter. These scripts are reasonably simple, and I've put lots of explanatory comments in them, so they're a good way to start learning scripting. Give them a try!

Here's an important thing you need to know about running scripts: When you run scripts that alter the set of EEGsets or ERPsets loaded in memory, you won't see this reflected in the **EEGsets** and **ERPsets** panels in **eStudio**. This is because eStudio doesn't know that the **EEG**, **ALLEEG**, **ERP**, and **ALLERP** variables have been updated. However, if you click the **Refresh** button in the **EEGsets** or **ERPsets** panel, that panel will be updated to reflect the current contents of **ALLEEG** or **ALLERP**.

Let's test this out. Go to the **ERP** panel in **eStudio**, and clear any ERPsets currently loaded in the **ERPsets** panel. Now run the **load_erpsets.m** script (by loading the script and clicking the green **Run** arrow in the main Matlab GUI. That script will load the 10 ERPsets that are stored in the exercises folder for this chapter. You can tell that these 10 ERPsets are loaded by typing **ALLERP** in the Command window. But they don't show up in the **ERPsets** panel (because **eStudio** doesn't know that **ALLERP** has been updated). Click the **Refresh** button in the **ERPsets** panel, and you'll now see the ERPsets in this panel.

This page titled 3.13: Matlab Scripts For This Chapter is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 3.13: Matlab Scripts For This Chapter by Steven J Luck is licensed CC BY 4.0.

3.14: Key Takeaways and References

Key Takeaways

- There are many steps between putting electrodes on the participant's head prior to the recording session and seeing the *p* value from the statistical analysis after the whole study is complete. As a result, there are many opportunities for some kind of problem or error that can lead you to draw the wrong conclusion from the study. As a result, it's crucially important that you constantly check for problems and errors.
- Some problems can be detected during preprocessing and averaging. For example, the response button may have been broken, leading to no event codes for responses. Or the participant's blinks may have been smaller than typical, causing many blinks to escape the artifact detection procedure. You can often catch these errors by checking the number of rejected and accepted trials and by examining the continuous and epoched EEG data.
- The process of scoring the amplitude or latency of an ERP component can also go awry for some participants, so you should use the Preview window in the Measurement Tool to examine the scores alongside the waveforms for each participant.
- Once you've entered the scores into a statistical analysis, you should compare the descriptive statistics from your statistical package with the grand average waveforms to make sure that the analysis was performed correctly. It's really easy to accidentally put the scores in the wrong order.

References

Cousineau, D. (2005). Confidence intervals in within-subjects designs: A simpler solution to Loftus and Masson's method. *Tutorials in Quantitative Methods for Psychology*, *1*, 42–45.

Cramer, A. O. J., van Ravenzwaaij, D., Matzke, D., Steingroever, H., Wetzels, R., Grasman, R. P. P. P., Waldorp, L. J., & Wagenmakers, E.-J. (2015). Hidden multiplicity in exploratory multiway ANOVA: Prevalence and remedies. *Psychonomic Bulletin & Review*, *23*, 640–647.

Farrens, J. L., Simmons, A. M., Luck, S. J., & Kappenman, E. S. (2019). Electroencephalogram (EEG) Recording Protocol for Cognitive and Affective Human Neuroscience Research. *Protocol Exchange*. https://doi.org/10.21203/rs.2.18328/v3

Frane, A. V. (2021). Experiment-Wise Type I Error Control: A Focus on 2 × 2 Designs. *Advances in Methods and Practices in Psychological Science*, *4*(1), 2515245920985137. https://doi.org/10.1177/2515245920985137

Luck, S. J. (2014). *An Introduction to the Event-Related Potential Technique, Second Edition*. MIT Press.

Luck, S. J., & Gaspelin, N. (2017). How to get statistically significant effects in any ERP experiment (and why you shouldn't). *Psychophysiology*, *54*, 146–157. https://doi.org/10.1111/psyp.12639

Luck, S. J., Stewart, A. X., Simmons, A. M., & Rhemtulla, M. (2021). Standardized measurement error: A universal metric of data quality for averaged event-related potentials. *Psychophysiology*, *58*, e13793. https://doi.org/10.1111/psyp.13793

Morey, R. D. (2008). Confidence intervals from normalized data: A correction to Cousineau (2005). *Tutorials in Quantitative Methods for Psychology*, *4*, 61–64.

Zhang, W., & Kappenman, E. S. (2024). Maximizing signal-to-noise ratio and statistical power in ERP measurement: Single sites versus multi-site average clusters. *Psychophysiology*, *61*, e14440. https://doi.org/10.1111/psyp.14440

This page titled 3.14: Key Takeaways and References is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 3.14: Key Takeaways and References by Steven J Luck is licensed CC BY 4.0.

4: FILTERING THE EEG AND ERPS

Learning Objectives

- Part 1 focuses on the basics of filtering in the frequency domain. You will learn to:
  - Compare the frequency content of an ERP waveform with the frequency response function of a filter to predict how well the filter will attenuate the noise in the data
  - Identify filter distortions
  - Choose filter settings that will work reasonably well for most experiments
- Part 2 focuses on how filtering works in the time domain. You will learn to:
  - Determine the *impulse response function* of a filter and conceptualize filtering as a process that replaces each point in the unfiltered waveform with a scaled-and-shifted copy of this function
  - Think of an ERP waveform as a series of *impulses*, one at each time point
  - Predict how a filter will distort an ERP waveform on the basis of the filter's impulse response function
- Part 3 focuses on how to choose the optimal filter settings for a given type of study. You will learn to:
  - Conceptualize what it means for a filter to be optimal
  - Create artificial waveforms that resemble an experimental effect
  - Quantify the waveform distortion that occurs when a set of filters is applied to the artificial waveform
  - Quantify the improvement in data quality produced by each of a set of filters
  - Choose a filter that best balances data quality with waveform distortion

You must use filters in ERP experiments. At a minimum, your amplifier's hardware includes an antialiasing filter that must be used prior to digitizing the EEG. In almost all ERP experiments, additional filtering is important for reducing sources of noise that would otherwise create large measurement error and reduce your statistical power. However, when filters are misused, they can dramatically distort your data, leading to incorrect conclusions. As a result, it's vitally important that you understand how filters work and the conditions under which they can significantly distort your ERP waveforms.

Consider, for example, the waveforms shown below in Figure 4.1. This is a simulation of an experiment in which there are two conditions, A and B. If you were to describe these results in a journal article, you might say that both the N1 and P2 components are larger in Condition B than in Condition A. Or you might be more careful and say that there is greater extended negativity in Condition B than in Condition A up until the peak of the N1 wave, along with a greater positivity in the P2 latency range for condition B than for Condition A. But both of these descriptions contain a clear falsehood that is a result of filtering.

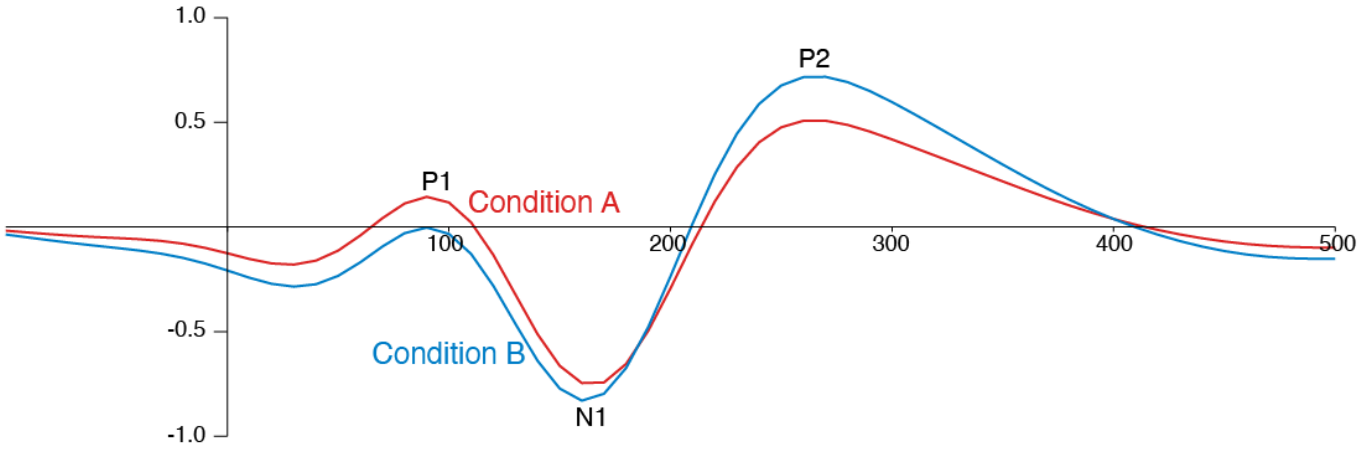


Figure 4.1. Simulation of an experiment with two conditions, A and B.

These waveforms were filtered using a combination of a 1 Hz high-pass filter and a 10 Hz low-pass filter (with a roll-off of 48 dB/octave). These would be somewhat unusual filter settings in some areas of research, but they would be fairly typical in others. And these filter settings created a completely artifactual effect in the early part of the waveform in Figure 4.1. I know with 100% certainty that the early effect is artifactual, because I simulated these waveforms so I know what the true effect is. These

waveforms consist of a P1, an N1, and a P2, and the amplitude of the P2 is 50% larger in Condition B than in Condition A. The true effect does not contain a more negative voltage in Condition B than in Condition A at any time point. The more negative voltage prior to the N1 peak for Condition B in the filtered waveform is entirely an artifact of filtering.

Can you imagine publishing a paper with these waveforms, concluding that there is a greater negativity for Condition B than for Condition A in the first part of the waveform, and later seeing a paper from another group of researchers demonstrating that your conclusion was based on a filtering artifact? That would be incredibly embarrassing. This sort of thing has actually happened. I don't want it to happen to you, so this chapter takes a deep dive into how filters work in the time domain. You may already know how they work in the frequency domain, but ERPs are a time-domain phenomenon, and it is essential for ERP researchers to understand how filters work in the time domain.

This chapter focused on giving you an intuitive, non-mathematical understanding of how filters work. If you want a more mathematical approach, see the free online Chapter 12 from An Introduction to the Event-Related Potential Technique.

Should You Read This Entire Chapter?

This chapter is divided into three parts. The first part (sections 4.1-4.5) provides some basic information about filters, including some practical recommendations for filter settings that work reasonably well for many kinds of ERP research. The second part (sections 4.6-4.10) is more theoretical and is designed to provide you with an intuitive understanding of how filters work in the time domain. The third part (sections 4.11-4.16) is more practical and shows you how you can determine the optimal filter settings for your research with the help of some of the tools included in ERPLAB.

If you are new to ERPs, I would recommend reading only the first part now. Once you have more experience, you can come back and read the second and third parts.

If you already have substantial ERP experience, I recommend that you read at least the first two parts. If you do research that yields relatively clean data (i.e., with cooperative participants, lots of trials, a gel-based EEG recording system, and a low-noise recording environment), you can skip the third part. But if you must deal with very noisy data, the third part will be important for you.

4.1: Data for this Chapter

4.2: Classes of Filters

4.3: Exercise- Assessing the Frequency Content of the Noise

4.4: Exercise- Filtering the 60 Hz Noise

4.5: Practical Advice About Filtering

4.6: Exercise- Filtering as a Linear Operation

4.7: Exercise- The Impulse Response Function

4.8: Exercise- Filtering a Series of Impulses (and a Sneaky Introduction to Convolution)

4.9: Predicting Waveform Distortion from the Impulse Response Function

4.10: Exercise- High-Pass Filtering

4.11: Background- How Should We Define the Optimal Filter?

4.12: Background- Quantifying Signal, Noise, and Waveform Distortion

4.13: Exercise- Creating artificial waveforms

4.14: Exercise- Quantifying Waveform Distortion in the Artificial Waveforms

4.15: Exercise- Quantifying Data Quality

4.16: Selecting the Optimal Filter

4.17: Matlab Scripts for this Chapter

4.18: Key Takeaways and References

This page titled 4: Filtering the EEG and ERPs is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 4: Filtering the EEG and ERPs by Steven J Luck is licensed CC BY 4.0.

4.1: Data for this Chapter

The data we'll be using for the exercises in this chapter can be found in the Chapter_4 folder in the master folder: https://doi.org/10.18115/D5G045.

This page titled 4.1: Data for this Chapter is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 4.1: Data for this Chapter by Steven J Luck is licensed CC BY 4.0.

4.2: Classes of Filters

There are four main classes of filters used in EEG/ERP research. They're typically named in terms of the frequencies that they pass, not the frequencies that are filtered out (much as an air filter passes air and filters out dust).

- A *low-pass* filter passes low frequencies and filters out high frequencies (e.g., muscle activity).
- A *high-pass* filter passes high frequencies and filters out low frequencies (e.g., gradual drifts resulting from skin potentials).
- A *bandpass* filter passes an intermediate band of frequencies and filters out the lower and higher frequencies. A bandpass filter is the same as filtering twice, once with a low-pass filter and once with a high-pass filter.
- A *notch* filter passes all frequencies except for a narrow band (e.g., 60 Hz). A notch filter may be used during an EEG recording when AC electrical devices produce so much contamination of the EEG that it's hard to see the signal.

I don't ordinarily recommend applying notch filters (unless they are necessary during the recording process); it's usually better to use a low-pass filter that attenuates all the high frequencies. However, if you don't want to use a low-pass filter with a cutoff of 20 or 30 Hz (e.g., because you are interested in relatively high-frequency activity), Section 4.5 describes an approach that is better than using a notch filter.

This page titled 4.2: Classes of Filters is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

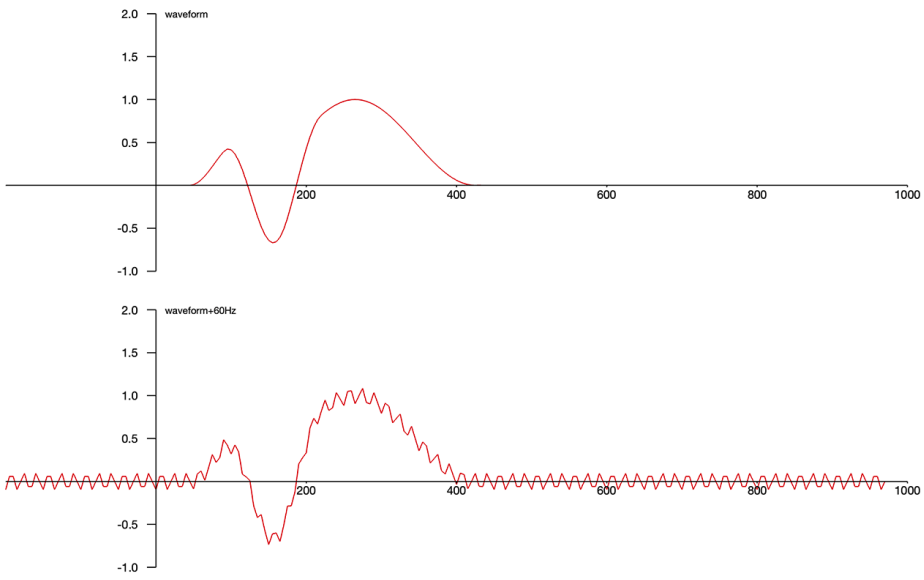
- 4.2: Classes of Filters by Steven J Luck is licensed CC BY 4.0.

4.3: Exercise- Assessing the Frequency Content of the Noise

We're going to start by thinking about filtering as a frequency-domain operation, in which we suppress some frequencies and pass others. If you don't already know the basics of Fourier analysis and how filtering works in the frequency domain, I recommend that you stop and learn those basics before going through the rest of this chapter. A quick way to learn those basics is to watch the videos for Part 1 and Part 2 from Chapter 6 of my free online Introduction to ERPs course. Each of the videos is only about 5 minutes long, so they'll get you up to speed quickly.

To start fresh, click the **Reset** button under the lower right corner of the plot region and then click **Reset ERPLAB Studio** in the window that appears. Set **Chapter_4** to be Matlab's current folder. Go to the **ERP** tab in eStudio and use the **Load** button in the **ERPsets** panel to load the ERPset file named **waveforms.erp**. It should look something like Screenshot 4.1.

Screenshot 4.1



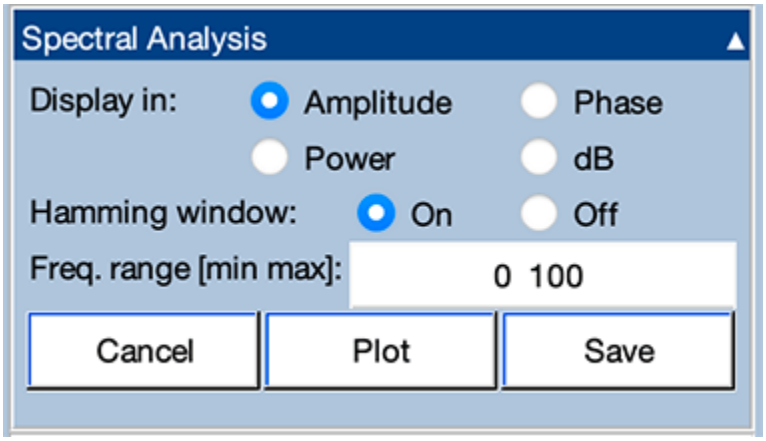
The first channel shows an artificial waveform that I created in Excel. The second channel is the sum of the first channel and a 60 Hz sine wave (like the *line noise* that is often picked up from electrical devices in the recording environment).

Line Noise

AC electrical lines run at 60 Hz in North America and some other parts of the world. Europe, China, India, and some other regions use 50 Hz. We often call this the *line frequency* to be agnostic about whether it is 50 or 60 Hz. The noise produced by this signal is called *line noise*.

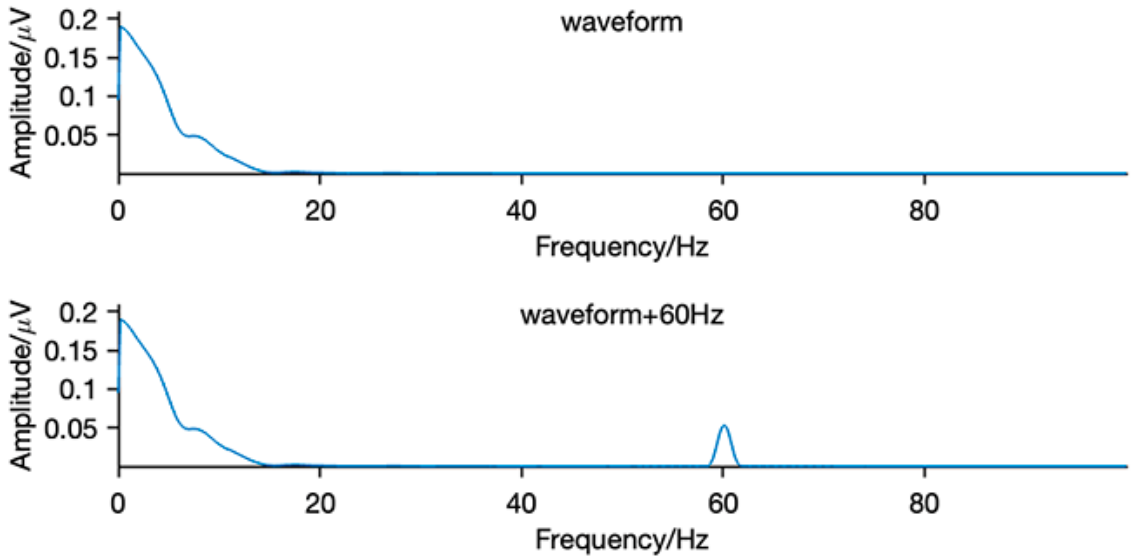
Before we filter the data, let's perform a Fourier transform on these waveforms so that we can see their frequency content. Go to the **Spectral Analysis** panel, make sure it's set up as shown in Screenshot 4.2, and then click the **Plot** button.

Screenshot 4.2



You should see something like the spectral plot in Screenshot 4.3. The X axis is the frequency, and the Y axis is the amplitude at this frequency. This plot tells us that we could reconstruct the original time-domain waveform by summing together a set of sinusoids with the set of amplitudes shown at each frequency in the plot. We'd also need to know the phase at each frequency to reconstruct the original waveform, but phase information isn't usually shown with ERP data. You can get a quick introduction to the Fourier transform in Chapter 6 of my online Introduction to ERPs course (or just watch this YouTube video). You can find a more detailed treatment in Chapters 7 and 12 of Luck (2014).

Screenshot 4.3



As you can see from the amplitude spectrum, the original ERP waveform mostly consists of relatively low frequencies. This is fairly typical of the waveforms you would see in most perceptual, cognitive, and affective experiments. In low-level sensory experiments, you might see more high-frequency activity.

The amplitude spectrum for waveform+60Hz is just like the amplitude spectrum for the original waveform, except that there is also activity around 60 Hz. This is because I created that ERP waveform by summing together the original waveform and a 60 Hz waveform. Note that the peak amplitude around 60 Hz is approximately 0.05 μV. That will become relevant in the next exercise, where we will apply a filter to reduce the amount of 60 Hz noise.

When you're first starting out in ERP research, you should plot Fourier transforms like these prior to filtering so that you have a good idea of the frequency content of the noise in your data. This can help you figure out where the noise is coming from (because different sources of noise have different frequency content). By knowing where the noise is coming from, you may be able to eliminate it in future recordings. It's better to reduce the noise before it contaminates your data rather than relying on filters and other signal processing techniques. For reasons described in Luck (2014), I call this *Hansen's Axiom*: "There is no

substitute for clean data." As we will see later in this chapter, filters reduce the temporal precision of your data. And isn't temporal precision one of the most important features of the ERP technique?

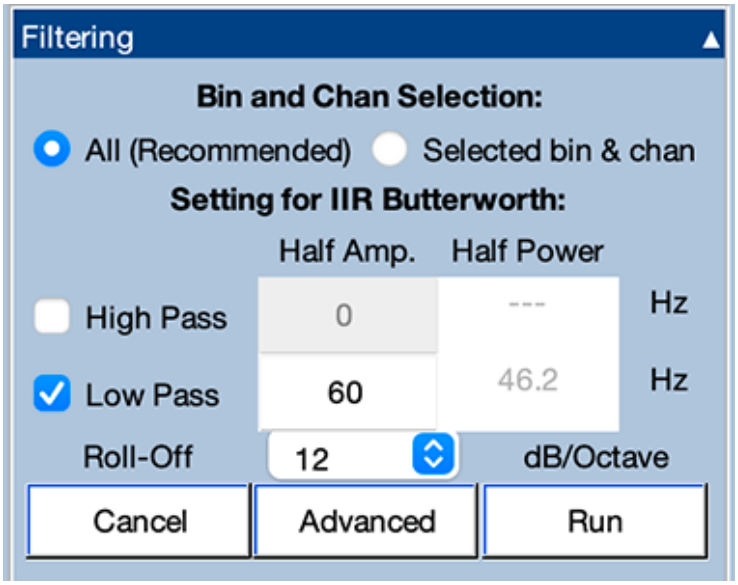
This page titled 4.3: Exercise- Assessing the Frequency Content of the Noise is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 4.3: Exercise- Assessing the Frequency Content of the Noise by Steven J Luck is licensed CC BY 4.0.

4.4: Exercise- Filtering the 60 Hz Noise

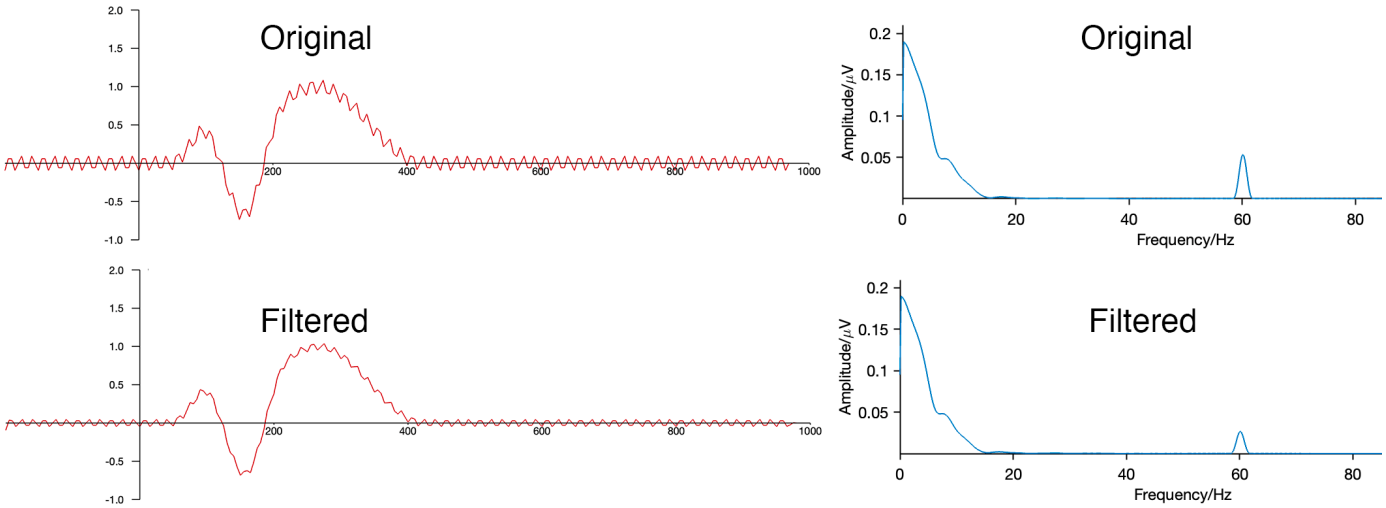
Now let's try to reduce the 60 Hz noise from the waveform from the previous exercise. Make sure that **waveforms** is still the selected ERPset, and then go to the **Filtering** panel, set it up as shown in Screenshot 4.4 (with 60 as the **Low Pass** cutoff), and click **Run**.

Screenshot 4.4



Name the new ERPset **waveforms_filt60** (to indicate that you filtered it at 60 Hz). If you look at the new version of waveform+60 in the plotting region, you'll see that it still has quite a bit of 60 Hz noise in it—maybe about half as much as before (see Screenshot 4.5). And if you go to the **Spectral Analysis** panel and click **Plot**, you'll see that the peak at 60 Hz is approximately 0.025 µV (which is half of the peak voltage of 0.05 µV that you saw before; see Screenshot 4.5).

Screenshot 4.5



You might have expected that entering a cutoff of 60 Hz would have led all of the 60 Hz activity to be filtered out. However, the **60** that you entered is actually the *half-amplitude* cutoff frequency, which means that the amplitude at 60 Hz will be cut in half. To understand this, we need to take a closer look at the filter. To do this, go to the ERPsets panel and select the original **waveforms** ERPset, and then go back to the **Filtering** panel and click the **Advanced** button. A new window will appear that will look like Screenshot 4.6. You'll probably need to set the **X limits** right below the waveform to **0 100**.

Screenshot 4.6



The function shown in this window is called the *frequency response function* of the filter. It tells you the *gain* that will be applied to each frequency. The gain is just a multiplication factor. A gain of 1 means that the amplitude of that frequency will be multiplied by 1 (i.e., unchanged). A gain of 0.75 means that the amplitude of that frequency will be multiplied by 0.75, which also means that it will be reduced by 25%. The half-amplitude cutoff of 60 Hz that we specified means that the gain of the filter is 0.5 at 60 Hz; this is the point at which the gain is reduced by half (which is why it's called the *half-amplitude* frequency). That's why the filter reduced the amplitude of the 60 Hz noise by only 50% (see Screenshot 4.5).

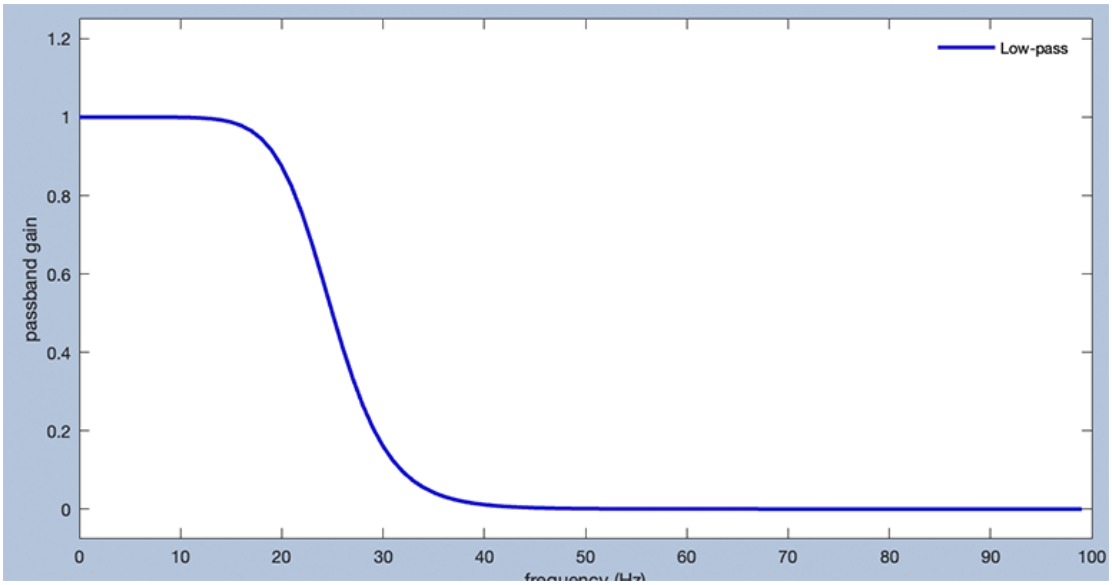
The shape of the filtering function is determined by two factors: the cutoff frequency and the *roll-off*. The roll-off specifies how rapidly the function drops, and it's specified in decibels (dB, a logarithmic amplitude scale) per octave (a logarithmic frequency scale). This is essentially the slope of the frequency response function at its steepest point.

The filter we've specified has a fairly gentle roll-off of 12 dB/octave. As a result, even though it's nominally a 60 Hz filter, the gain is still well above zero at 80 Hz, and there is noticeable attenuation as low as 25 Hz.

Stay in the **Advanced** window and change the low-pass cutoff to 25 Hz. You'll see that the frequency response function changes, and the half-amplitude point is now at 25 Hz instead of 60 Hz. But the gain of this filter is still well above zero at 60 Hz, which might seem to indicate that it is not ideal for getting rid of 60 Hz noise. In just a bit, you will see that this filter is actually reasonably good for minimizing 60 Hz noise.

Now try changing the roll-off to 48 dB per octave. As shown in Screenshot 4.7, the resulting frequency response function looks a lot better. The gain is now near 1 for everything below about 15 Hz, and it's near zero for everything above about 40 Hz. However, as we'll see later in this chapter, this sharper frequency response function means that the filter produces more distortion in the time domain. It looks great, but looks can be deceiving.

Screenshot 4.7



Change the roll-off back to 12 dB/octave, click **Run** to apply the filter, and then name the new ERPset **waveforms_filt25**. You can see that, even though the waveform still wiggles a little bit at 60 Hz, the oscillation is mostly gone. And in a real ERP waveform, any remaining 60 Hz would be much smaller than the noise level of the data and have little or no impact. So, this filter works fine as long as the amplitude of the 60 Hz noise isn't too great.

Now let's try a more severe filter. Make the original unfiltered ERPset active (by selecting it in the **ERPsets** menu) and filter it just as before, except set the low-pass cutoff to **10** Hz with a roll-off of **12** dB/octave. Click **RUN** and then name the new ERPset **waveforms_filt10**. This filter more fully eliminates the 60 Hz noise, but as we will see in a moment, it reduces the temporal resolution of the data. You can also see a weird little dip in the waveform at the very beginning of the baseline period. This is called an *edge artifact;* we will talk about these artifacts in a later section of this chapter.

To better see the distortion produced by the 10 Hz filter, let's overlay the filtered and unfiltered data. Unfortunately, you can't directly overlay waveforms from different ERPsets (unless you launch the **Advanced Waveform Viewer**). However, there's a little trick for getting around this limitation: We can append multiple ERPsets together into a single ERPset, which creates different bins for the data from each original ERPset. We can then overlay those bins.

Let's append the ERPsets from the original data, the data filtered at 25 Hz, and the data filtered at 10 Hz. Select those three ERPsets in the **ERPsets** panel (without selecting the ERPset that was filtered at 60 Hz). Then go to the **Append ERPsets** panel, set it up as shown in Screenshot 4.8, click **Run**, and save the resulting ERPset with its default name of **append**.

Screenshot 4.8

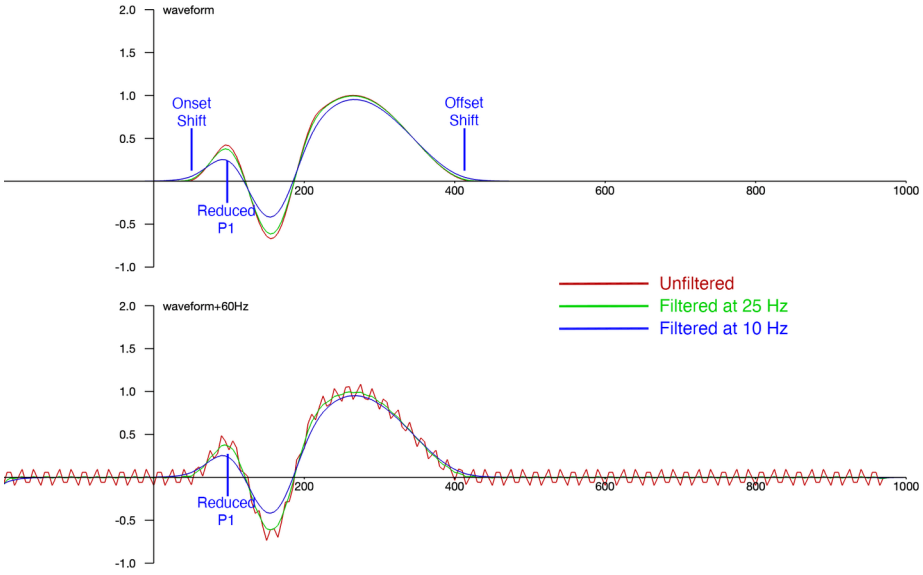


If you look at the Bin & Channel Selection panel, you'll see that this ERPset has 3 bins: Bin 1 is the original unfiltered data; Bin 2 is the data after filtering at 25 Hz; and Bin 3 is the data after filtering at 10 Hz. These three bins are overlaid in the plotting region, as shown in Screenshot 4.9.

The red line in the bottom set of waveforms shows the original waveform with the 60 Hz noise added. The green line shows the effect of filtering at 25 Hz: The green line falls nearly perfectly between the upward and downward deflections of the red line.

That's just what we want: The same basic waveform, but without the little "uppies and downies" from the noise. The blue waveform shows the effect of filtering at 10 Hz. You can see that the first positive peak (analogous to a P1) and the first negative peak (analogous to an N1) are reduced in amplitude by this filter.

Screenshot 4.9



It's easier to see the distortion produced by a filter by filtering noise-free waveforms. That's what is shown at the top of the screenshot. The 25-Hz filtered waveform lies almost perfectly over the unfiltered waveform, producing only a tiny amount of distortion. However, the 10-Hz filtered waveform has significantly reduced P1 and N1 amplitudes. Moreover, the onset time of the P1 at the beginning of the waveform is earlier in the 10-Hz filtered data than in the original data, and the offset time of the third positive peak (analogous to a P2 wave) is delayed in the 10-Hz filtered data.

This example shows how a little bit of filtering can help reduce noise (e.g., the 25-Hz filter), but too much filtering can distort the data (e.g., the 10-Hz filter). The next section will provide some simple advice about how to select filters that will do a reasonably job of reducing noise without producing significant waveform distortion. This will be followed by several sections that explain how filters work in the time domain so that you can understand when and how they will distort your waveforms. The last several sections will show you how to choose filters for your own data so that you reduce the noise but do not produce too much waveform distortion.

This page titled 4.4: Exercise- Filtering the 60 Hz Noise is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 4.4: Exercise- Filtering the Artificial Waveforms by Steven J Luck is licensed CC BY 4.0.

4.5: Practical Advice About Filtering

The sections following this one will take a deep dive into how filters work in the time domain, why they produce waveform distortion, and how you can choose the optimal filter settings for your own data so that you reduce as much noise as possible without producing too much waveform distortion. But many readers won't want to take this deep dive now, so this section will provide several pieces of important practical advice that all readers will want to know.

When to Filter

Let's start with the question of when to filter. With typical parameters, high-pass filters can create "edge artifacts" at the beginning and end of the waveform. As a result, it is usually a bad idea to apply high-pass filters to epoched EEG or averaged ERPs. Instead, high-pass filters should be applied to the continuous EEG so that these artifacts only impact the very beginning and very end of the recording. In addition, I recommend starting the recording 10-20 seconds before the task starts and ending it 10-20 seconds after the task ends. That way, the artifacts will be in a part of the recording that doesn't contribute to your final analyses. To further minimize edge artifacts, you should use the option labeled "Remove mean value (DC offset) before filtering" when you are filtering continuous EEG data (but not when you are filtering epoched EEG or averaged ERPs). This option is active by default, but you could disable it by clicking the **Advanced** button in the **Filtering** panel. Don't disable it unless you really know what you're doing.

With typical parameters, you can apply low-pass filters to the continuous EEG, the epoched EEG, or the averaged ERPs. For most researchers, it will be simplest just to apply both the low- and high-pass filters to the continuous EEG. In my lab, we often apply only a high-pass filter to the continuous EEG and save our low-pass filtering for the averaged ERPs, but that's mainly for philosophical reasons rather than practical reasons (see Luck, 2014).

If you are going to apply both a low-pass filter and a high-pass filter to the continuous EEG, you can apply these filters separately in two steps, or you can use a single band-pass filter that attenuates both the low and high frequencies in a single step. The result will be the same either way.

Dealing with 50- or 60-Hz Line Noise

If you have 50- or 60-Hz line noise in your data, a low-pass filter with a cutoff around 20 Hz can usually reduce it to insignificant levels. However, if this noise is huge, or if you don't want to use a low-pass filter to remove it, I recommend avoiding notch filters (which attenuate one narrow band of frequencies and pass everything else). Instead, you can use EEGLAB's cleanline plugin (see Bigdely-Shamlo et al., 2015 for important details). If cleanline doesn't work well for you, you can try the newer Zapline Plus method (de Cheveigné, 2020; Klug & Kloosterman, 2022). However, Zapline has not been as extensively tested (as of the time of this writing).

Recommended Filter Settings

The most difficult part about filtering is deciding on the specific filter to use (i.e., the cutoff frequency and the roll-off slope). In general, I recommend a roll-off of 12 dB/octave unless you really know what you're doing. Choosing the cutoff frequency is much more complicated, because the optimal cutoff frequency depends on the nature of your ERPs, the amount and nature of the noise in the EEG, and whether you will be scoring amplitudes or latencies. And if you choose the wrong cutoff frequency, you might leave too much noise in your data, making it impossible to get statistically significant effects. Or you might choose a filter that dramatically distorts your waveforms, leading you to draw incorrect conclusions (as in the example given in the introduction to this chapter).

My lab has developed a procedure for determining the optimal filter settings for a given situation that balances the need to reduce noise and the need to avoid serious waveform distortion (Zhang et al., 2024a). The next part of this chapter is devoted to giving you the background needed to understand this procedure, and the last part shows you how to implement the procedure in ERPLAB. However, you may not want to go through all of that, at least not right now. So, I will provide some simple and approximate advice here that you can use without reading the rest of the chapter.

If you are doing research on typical cognitive or affective ERP components in a cooperative participant population, and you can get a decent number of trials per participant, you will probably be fine using a high-pass cutoff at 0.1 Hz and a low-pass cutoff at 20 Hz. This should reduce the noise reasonably well without producing significant waveform distortion. If your data has a lot more low-frequency noise (e.g., because you are studying infants or young children, or you are recording from people while they are moving around in the world), you should raise the high-pass cutoff to at least 0.2 Hz and possibly as high as 0.5 Hz (but at 0.5 Hz you risk some significant waveform distortion).

I also have more specific recommendations that may be relevant for you. These recommendations come from a study (Zhang et al., 2024b) in which we applied our procedure for determining optimal filter settings to the 7 common ERP components from the ERP CORE (N170, MMN, P3b, N400, N2pc, LRP, ERN). The optimal filter settings are shown below in Table 4.1.

If you are using these components (or similar components) in your research, and you are getting high-quality EEG data from cooperative participants, you should be able to use the filter settings shown in the table. For example, if you are looking at the P600 in a language study, our P3b or N400 settings will probably work well for you. However, the settings shown in Table 4.1 were derived from relatively clean EEG/ERP data that were obtained from highly motivated neurotypical college student participants using a high-quality EEG system with a gel-based conductive medium and an electrically shielded chamber. If you have noisier data, then our filter settings might not be appropriate for you. However, we also simulated noisier data by adding white noise to the EEG, and Table 4.1 indicates changes you should make if you have somewhat noisier data. But we did not simulate data with other kinds of noise or greater amounts of noise, so the settings shown in the table will not be appropriate for all research.

If you are recording from a much noisier population (e.g., infants) or using a lower-quality EEG system (e.g., a saline-based or dry electrode system), these settings may not be optimal for you. They may also be inappropriate if you are examining a component that is very different from the 7 components in the ERP CORE (e.g., short-duration components like the auditory brainstem responses or long-duration components such as the CDA). In these cases, I strongly recommend that you read the following sections and apply our procedure to your own data to determine the optimal filters.

Table 4.1. Recommended half-amplitude filter cutoffs (with a roll-off of 12 dB/octave) for each combination of scoring method and ERP component in the ERP CORE (adapted from Zhang et al., 2024b)

	Mean amplitude	Mean amplitude	Peak amplitude	Peak amplitude	Peak latency	Peak latency	50% area latency	50% area latency
	High-pass	Low-pass	High-pass	Low-pass	High-pass	Low-pass	High-pass	Low-pass
N170	0.9	≥30 or none	0.9	≥30 or none (or 30*)	≤0.9	10-20	10-20	10-20
MMN	0.5	≥20 or none	0.5	≥20 or none (or 30*)	≤0.5	10 (or 5†)	10 (or 5†)	≥20 or none
N2pc	0.5	≥20 or none	0.5	20-40 (or 20*)	0.01 or none	10	10	5 (or 10‡)

	Mean amplitude	Mean amplitude	Peak amplitude	Peak amplitude	Peak latency	Peak latency	50% area latency	50% area latency
P3	0.2	≥10 or none	0.2	≥10 (or 10*)	0.2	10 (or 5*)	10 (or 5*)	10 (or 5*)
N400	0.2	≥10 or none	0.2	≥10 (or 10*)	0.2	10 (or 5*)	10 (or 5*)	10 (or 5*)
LRP	0.3	≥30 or none	0.3	30 (or 20*)	0.3	5 (or 10‡)	5 (or 10‡)	5 (or 10‡)
ERN	0.4	≥20 or none	0.4	≥20 or none	0.4	10	10	10

*For data with moderate amounts of high-frequency noise

†For data with moderate amounts of broadband noise (if distortion of onset and offset times can be tolerated)

‡To avoid distortion of onset and offset times

Filters for Scoring Onset Latency

Table 4.1 shows the optimal filters for scoring mean amplitude, peak amplitude, peak latency, and 50% area latency. If you want to score the onset latency of an ERP component, you may want a lower half-amplitude cutoff. My lab often applies a 10 Hz low-pass filter (12 dB/octave) to the averaged ERPs prior to scoring onset latency. We find that onset latency measures are highly sensitive to noise, and we get much better statistical power by filtering at 10 Hz.

This might seem problematic given that a 10 Hz cutoff produces a substantial distortion of the onset latency. However, when we quantify the onset latency of an ERP component, we don't measure the time that the waveform first deviates from zero. Instead, we measure the time at which the voltage reaches 50% of the peak voltage (see Chapter 9 in Luck, 2014, for a justification of this approach). This 50% point is nearly identical for filtered and unfiltered waveforms in most cases. Also, we're usually comparing the ERPs from two conditions that have both been filtered at 10 Hz, so the effect of the filtering should be equivalent for both conditions. However, we sometimes want to examine the absolute latency at which a difference wave first deviates from zero. In these cases, we do minimal filtering or we quantify the amount of latency shift produced by the filter (see, e.g., Bae & Luck, 2018).

This page titled 4.5: Practical Advice About Filtering is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck.

4.6: Exercise- Filtering as a Linear Operation

We're now beginning the second part of the chapter, where you will do exercises to help you understand how filtering works in the time domain and why filters can distort ERP waveforms. An important thing to keep in mind is that filtering in the time domain is mathematically equivalent to filtering in the frequency domain. Whether your software actually filters in the time domain or in the frequency domain is an issue of convenience, because one approach might be simpler to implement or run faster. But you will get the same waveform distortion no matter whether the filtering is implemented in the time domain or the frequency domain. However, it's a lot easier to understand *why* filters distort ERP waveforms when we think about them in the time domain rather than the frequency domain.

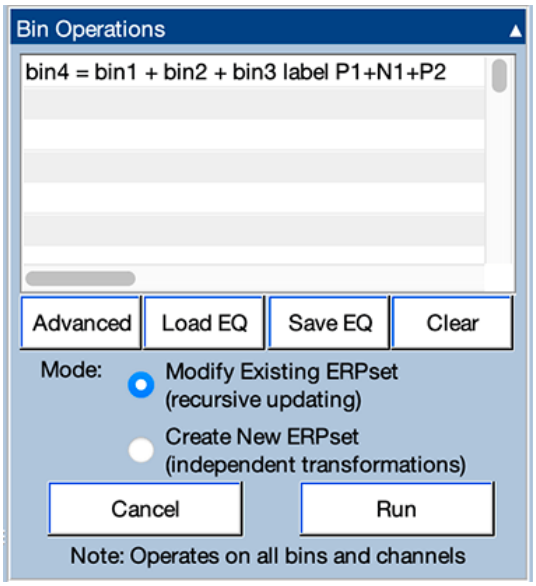
The first step to understanding filtering in the time domain is understanding a key property of most filters, namely *linearity*. That sounds complicated, but this exercise will show you that it's actually very simple.

To get started, go to the **ERPsets** panel and load the ERPset named **P1+N1+P2**. You'll see that it contains 3 bins, one with a simulated P1 wave, one with a simulated N1 wave, and one with a simulated P2 wave. I'll explain how you can create simulated components like these later in the chapter.

An observed ERP waveform is simply the sum of the underlying components (see Chapter 2 in Luck, 2014, or this video from the Introduction to ERPs online course). We are therefore going to create an ERP waveform by summing together these three components.

To do this, go to the **Bin Operations** panel and type this equation into the first line of the text box: **bin4 = bin1 + bin2 + bin3 label P1+N1+P2** (see Screenshot 4.14). This equation creates a new bin 4 by summing together bins 1, 2, and 3. Make sure that the **Mode** is set to **Modify Existing ERPset** (as in the screenshot). Click **Run**, and now you should be able to see this summed waveform in the plotting region.

Screenshot 4.14

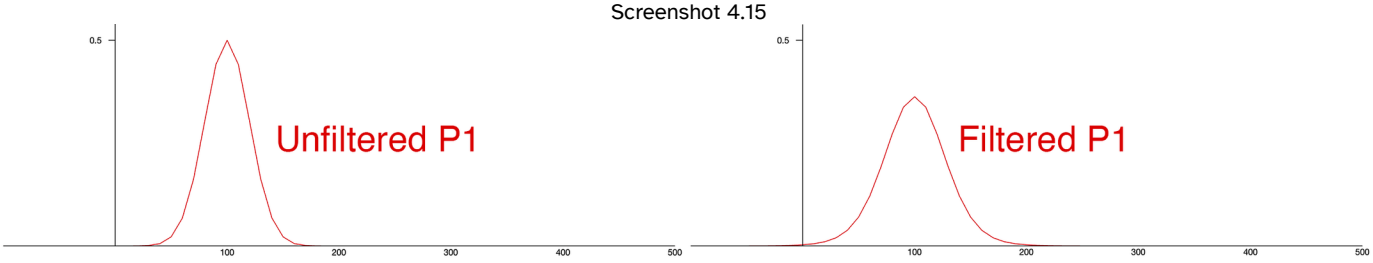


Now we're going to apply the 10 Hz filter we used in the previous exercise to all 4 bins. This will allow you to see how the filter distorts each component as well as the whole waveform. Go to the **Filtering** panel, indicate that you want a half-amplitude cutoff of **10** Hz and a slope of **12** dB/octave, make sure that the **Bin and Chan Selection** buttons at the top are set to **All**, click **Run**, and name the resulting ERPset **P1+N1+P2_filt10**.

Let's take a close look at the effect of filtering on the P1 wave. Select only Bin **1** in the **Bin & Channel Selection** panel, and then go back and forth between **P1+N1+P2** and **P1+N1+P2_filt10** in the **ERPsets** panel. You should now be able to compare the filtered and unfiltered P1 waves, which should look like Screenshot 4.15. Just as you saw in the previous exercise, this filter "smears out" the P1, reducing its amplitude and shifting the onset and offset times. You can see the same thing for the N1 and P2

waves by comparing Bin 2 or Bin 3 in **P1+N1+P2** and **P1+N1+P2_filt10**. And you can see the effect of filtering on the whole ERP waveform by comparing Bin 4 in the two ERPsets.

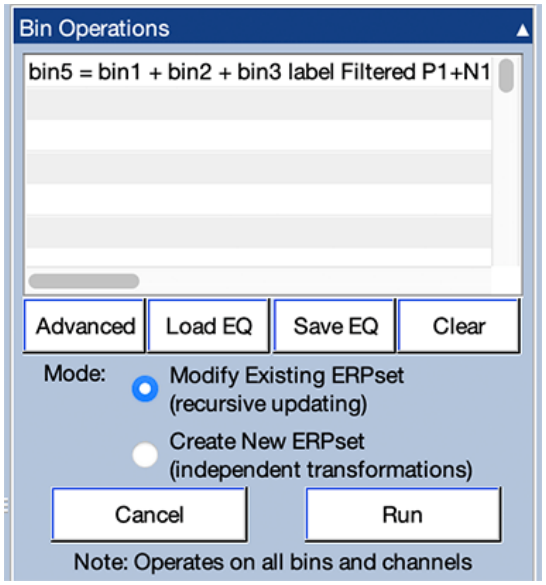
Screenshot 4.15



Now I want to demonstrate a very cool and extremely cool property of filtering. Recall that, prior to filtering, you created the overall ERP waveform (Bin 4) by summing together the P1, N1, and P2 components. Could we instead construct the overall filtered ERP waveform by summing together the filtered versions of the P1, N1, and P2 components? In other words, is the filtered Bin 4 (i.e., the filtered overall waveform) equal to the sum of the filtered Bins 1–3 (the filtered P1, N1, and P2 components)? Take a guess, and then we can see the answer.

To see the answer, let's sum together Bins 1-3 in the filtered ERPset and see if the result is the same as the overall filtered waveform in Bin 4. To do this, we're going to make a new Bin 5 that sums together the filtered components in Bins 1, 2, and 3. Make sure that **P1+N1+P2_filt10** is the selected ERPset, then go to the **ERP Bin Operations** panel and replace the equation in the text box with this: **bin5 = bin1 + bin2 + bin3 label Filtered P1+N1+P2**. (see Screenshot 4.16). Now click **Run** to create Bin 5.

Screenshot 4.16



Whereas Bin 4 contains the result of filtering the entire waveform, the new Bin 5 contains the sum of the filtered P1, N1, and P2 components. If you select just Bins 4 and 5 in the **Bin & Channel Selection** panel, you will see that the waveforms in these two bins lie right on top of each other. You can't even see that there are two waveforms, but if you go back and forth between Bins 4 and 5 to see them one at a time, you can verify that they are exactly the same.

This demonstrates an important principle of most filters: Filtering is a linear operation. Linear operations can be combined in any order and will yield the same result. In other words, you will get the same result whether you (a) sum the components together (which is also a linear operation) and then filter this overall ERP waveform, or (b) filter the individual components and then sum them together.

New ERP researchers (and even experienced ERP researchers) often ask me "What is the right order for the different EEG and ERP processing steps?" You now know a big part of the answer: The order doesn't matter for linear processing steps such as filtering, because any ordering of linear steps yields the same end result. For a more complete discussion of this general issue,

see the Appendix in Luck (2014). Appendix 3 of the present book provides an example pipeline that implements an ideal order of processing steps for many studies.

This page titled 4.6: Exercise- Filtering as a Linear Operation is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck.

- 4.4: Exercise- Filtering the Artificial Waveforms by Steven J Luck is licensed CC BY 4.0.

4.7: Exercise- The Impulse Response Function

In the previous exercise, you saw that an ERP waveform consists of the sum of several underlying components, and you saw that filtering the underlying components and then summing them together is equivalent to filtering the whole waveform. In this exercise and the next, we will take this one step further, breaking the waveform into a set of *impulses*, one at each time point. We will see that filtering these impulses and then summing them together is equivalent to just filtering the whole waveform.

These exercises might seem a little abstract, but they will give you a deep, intuitive understanding of how filters actually work and why they can distort the ERP waveform. Once you've developed this intuitive understanding, we will move on to the very practical issue of how to select the filters for your data that will minimize noise in your data without producing significant waveform distortion.

There are many types of filters, and this chapter focuses on a common class called *finite impulse response* filters (even though this ends up being a slight oversimplification of the filters that are actually used in ERPLAB). To understand these filters, you need to think of an ERP waveform as a set of impulses, one at each time point in the waveform. This is illustrated in Figure 4.2. Panel A of this figure shows the same overall waveform that you saw in the previous exercise, which was created by summing together P1, N1, and P2 components.

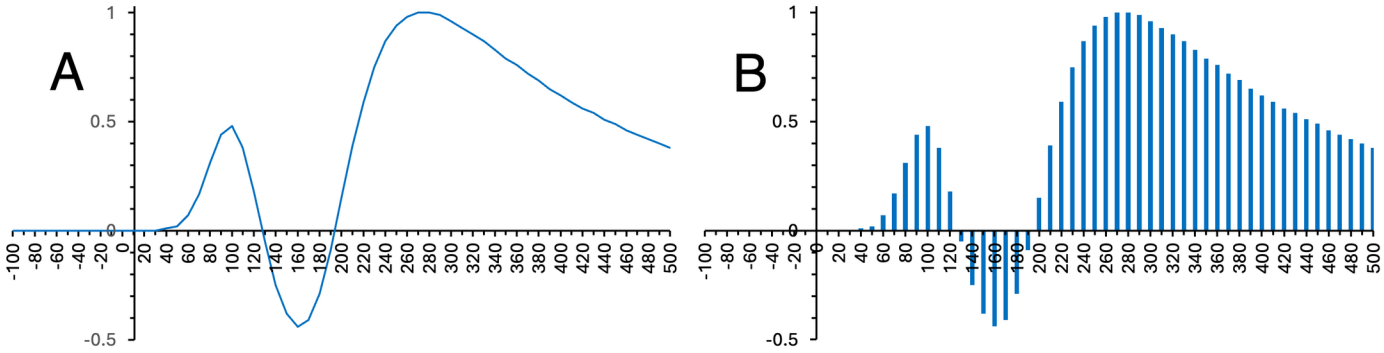


Figure 4.2. (A) Artificial ERP waveform drawn in the usual way. This waveform was created by summing together artificial P1, N1, and P2 components. (B) Same waveform as in (A), but drawn as an impulse at each time point.

These waveforms are drawn in the usual way, with lines connecting the voltages at the individual time points as if we have a continuous function. However, we actually have a set of discrete values. In this example, we have one voltage value every 10 ms (a sampling rate of 100 Hz). Panel B shows an equally valid way of plotting the same waveform, with a bar at each time point.

To understand how filters work, it is helpful to think of the bars in Panel B as being sequences of *impulses*. An impulse is a waveform that is zero everywhere except for a nonzero value at a single time point. Figure 4.3 shows an example of an impulse of amplitude 1 μV at time zero. The bars in Figure 4.2 can be considered a sequence of the kind of impulse shown in Figure 4.3, with one impulse at each time point and an impulse amplitude that matches the voltage at that time point.

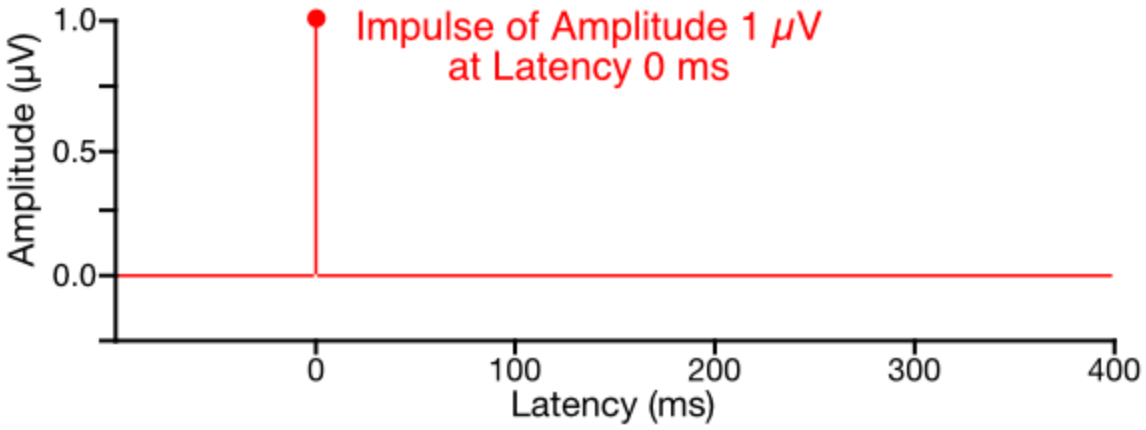
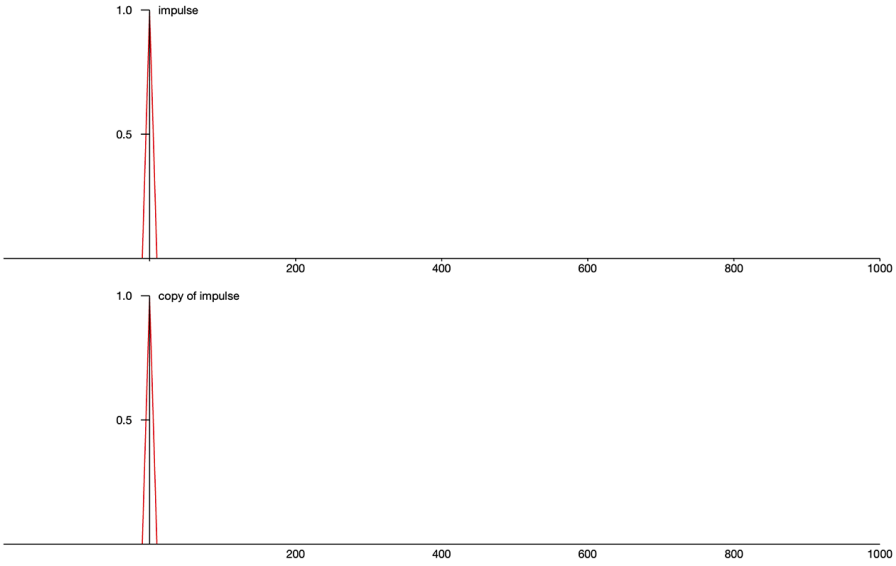


Figure 4.3. Example of an impulse at time zero with an amplitude of 1 μV. The voltage is 0 μV at all other time points.

Why do we want to conceive of an ERP waveform as a sequence of impulses? The reason is that we can predict the effects of a filter on the whole waveform by looking at the effects of that filter on the individual impulse shown in Figure 4.3. In this exercise, we will therefore take a close look at the effects of filtering the impulse shown in Figure 4.3, which is stored in an ERPset file named **impulse.erp**.

To see this impulse, reset everything with the **Reset** button, go to the **ERP** tab, and load **impulse.erp**. It should look like the waveform shown in Screenshot 4.10. This ERPset has two channels, with the same impulse in both channels (for reasons that will be explained soon).

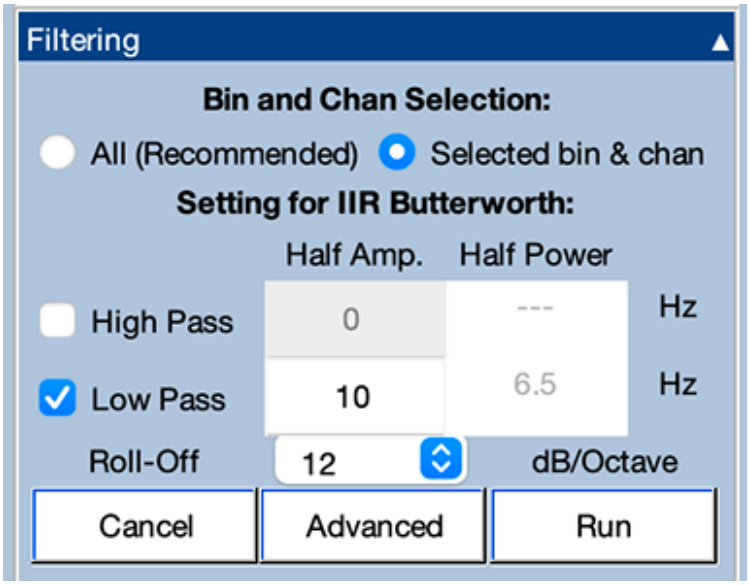
Screenshot 4.10



You can see the impulse (a voltage of 1 μV) at time zero. It looks like a narrow triangle rather than a pure impulse because the waveform is sampled at 100 Hz (one sample every 10 ms), so there is a line going from 0 μV at -10 ms to 1 μV at 0 ms and back down to 0 μV at 10 ms.

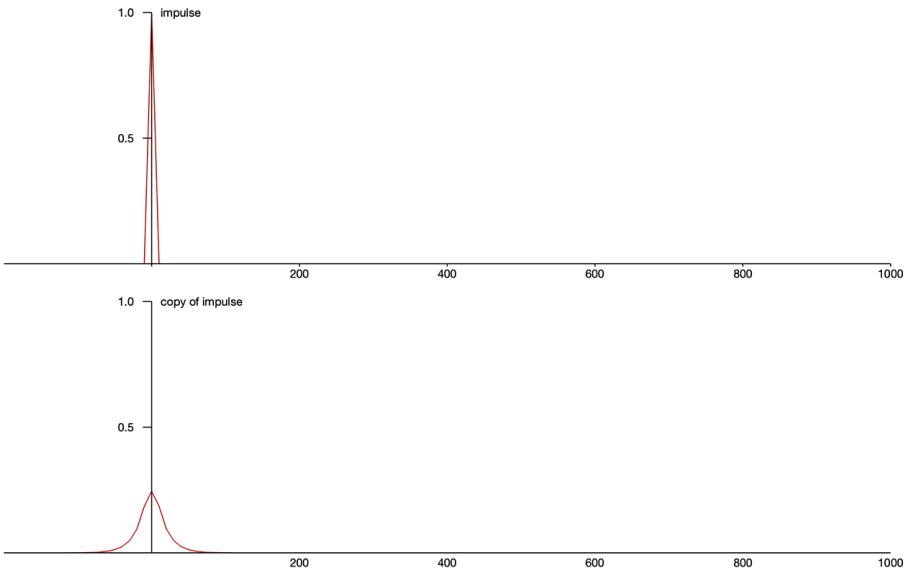
Now we're going to look at the effects of applying a filter to the impulse. We'll apply it just to the impulse in Channel 2 so that we can see both the unfiltered waveform (in Channel 1) and the filtered waveform (in Channel 2). First, go to the **Bin & Channel Section** panel, select just Channel **2**, and click **Apply**. Now go to the **Filtering** panel and set it up as shown in Screenshot 4.11, which tells it to filter just the selected bin(s) and channel(s) and specifies a half-amplitude cutoff at 10 Hz and a slope of 12 dB/octave.

Screenshot 4.11



Click **Run** and name the ERPset **impulse0_filt10**. Now go to the Bin & Channel Section panel, select **All** channels, and click **Apply**. You should now see the unfiltered impulse at the top of the plot area and the filtered impulse at the bottom, like Screenshot 4.12.

Screenshot 4.12



Remember when you applied this same 10 Hz filter to the simulated ERP waveform in the previous exercise. And remember how it reduced the amplitude of the P1 wave and shifted the onset and offset times of the waveform? You can see those same effects in the filtered impulse: Its amplitude is greatly reduced, with a peak of approximately 0.24 µV, and its onset and offset times have been shifted. Informally speaking, the filter "smeared out" the impulse.

You can also think of this example as showing that filters reduce the temporal resolution of the ERP waveform. Instead of a nice precise impulse at time zero, the voltage in the filtered waveform is spread out over a period of approximately ±50 ms.

The key to understanding the kinds of filters used by ERPLAB is to understand something called the *impulse response function*, which is simply the waveform that is produced when the filter is applied to an impulse of amplitude 1 at time zero. You created the impulse response function corresponding to our 10 Hz filter when you applied this filter to the impulse in the **impulse0** ERPset. That is, the bottom part of Screenshot 4.12 shows the impulse response function of the 10 Hz filter.

Here's why impulse response functions are so important: Filtering an ERP waveform is equivalent to *convolving* the ERP waveform with the impulse response function. The mathematical process of *convolution* will be explained in the next couple sections.

You don't actually need to filter an impulse to see the impulse response function in ERPLAB. To see the impulse response function for the 10 Hz filter, go to the **Filtering** panel, click the **Advanced** button, and change the **Display** button from **Filter frequency response** to **Filter impulse response**. As you can see from Screenshot 4.13, the impulse response function is now plotted. Only the right half of the function is shown, but the left half is just the mirror image. Like the impulse response function that you created by filtering the impulse (Screenshot 4.12), the peak amplitude of the impulse response function in the Advanced window is approximately 0.24 µV (Screenshot 4.13). You can also see that the amplitude nears zero starting around 70 ms.

Screenshot 4.13



In the next section, we'll take a closer look at the idea that an ERP waveform is really just a sequence of impulses, and that understanding the effects of a filter on a single impulse can tell you how the filter will impact the whole waveform.

A Slight Oversimplification

In this chapter, I discuss how finite impulse response (FIR) filters work, because they are quite easy to understand. However, ERPLAB implements filtering using a specific type of infinite impulse response (IIR) filter called a Butterworth filter. As long as you use a shallow roll-off (e.g., 12 dB/octave), ERPLAB's filters provide a reasonable approximation of a FIR filter. So, everything I say in this chapter is approximately correct for ERPLAB's filters as long as you use a shallow roll-off.

The key difference between FIR and IIR filters is that the output of an IIR filter feeds back into the filter's input. This means that the filter is nonlinear, with a response that could theoretically extend infinitely in time. The main advantage is that IIR filters require fewer coefficients than FIR filters, making them run faster and potentially reducing edge effects (which will be described later).

This page titled 4.7: Exercise- The Impulse Response Function is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 4.5: Exercise- The Impulse Response Function by Steven J Luck is licensed CC BY 4.0.
- 4.7: Background- Filtering with a Running Average by Steven J Luck is licensed CC BY 4.0.

4.8: Exercise- Filtering a Series of Impulses (and a Sneaky Introduction to Convolution)

In the preceding exercise, you learned about the impulse response function of a filter. In the present exercise, we're going to see how a filter's impulse response function can be used to understand the effects of the filter on an entire ERP waveform. This exercise is a little long, but it's worth it because by the end you should truly understand how filters work!

To keep things simple, we're going to use a simplified waveform that contains only a single ERP component (the P1 wave that was combined with the N1 and P2 waves to create the simulated ERP waveform in the previous two exercises). Figure 4.4 shows the P1 waveform, with Panel A showing the usual format and Panel B showing the waveform as a series of impulses.

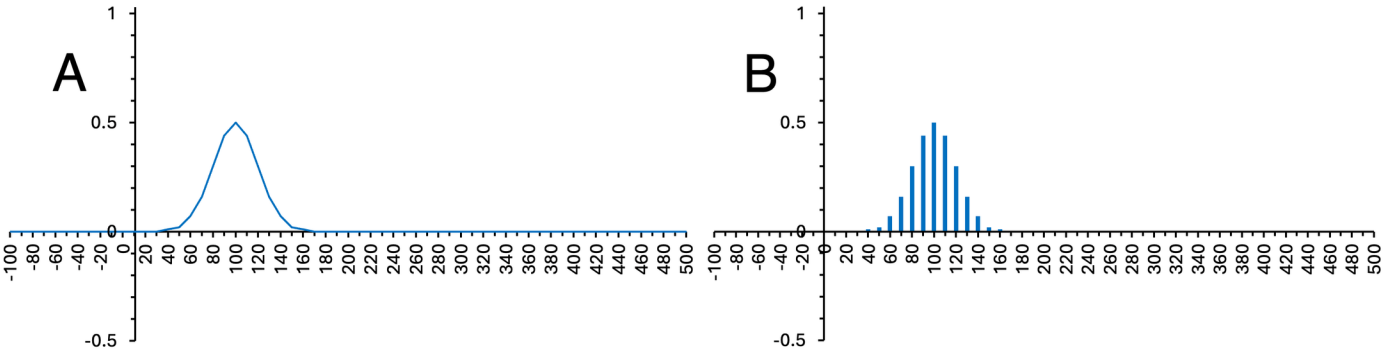
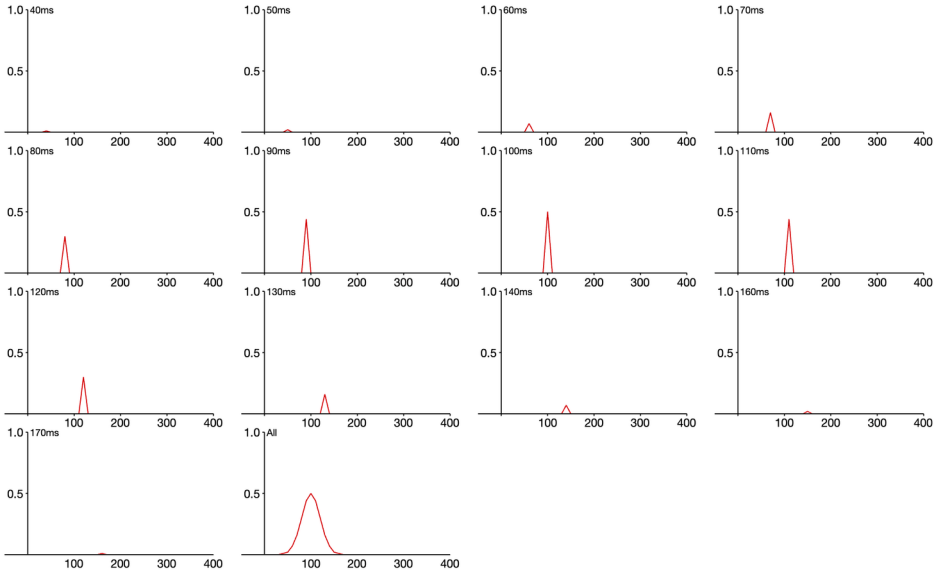


Figure 4.4. Two ways of drawing the P1 wave, which actually consists of one voltage value every 10 ms (because the sampling rate in this example is 100 Hz). (A) Conventional approach, in which the points are connected with lines to make it look like a continuous waveform. (B) Alternative approach to plotting, in which the voltage at each time point is represented with an impulse so that we can more easily conceptualize the role of the impulse response function in filtering.

The waveform in Figure 4.4 is stored in an ERPset named **P1_Separated**, which you should load using the **ERPsets** panel. As shown in Screenshot 4.17, Channel 14 of this ERPset contains the P1 wave. Each of the other channels contains the impulse for one of the time points in the P1 wave. For example, Channel 1 is labeled **40 ms** and contains a tiny impulse corresponding to the amplitude of the P1 wave at 40 ms. Channel 7 is labeled **100 ms** and contains a 0.5 µV impulse corresponding to the 0.5 µV peak amplitude of the P1 wave at 100 ms. As mentioned in the previous exercise, the impulses look like triangles because there is a voltage value every 10 ms. For the impulse at 100 ms, for example, a line connects a voltage of zero at 90 ms with a voltage of 0.5 at 100 ms and a voltage of zero at 110 ms.

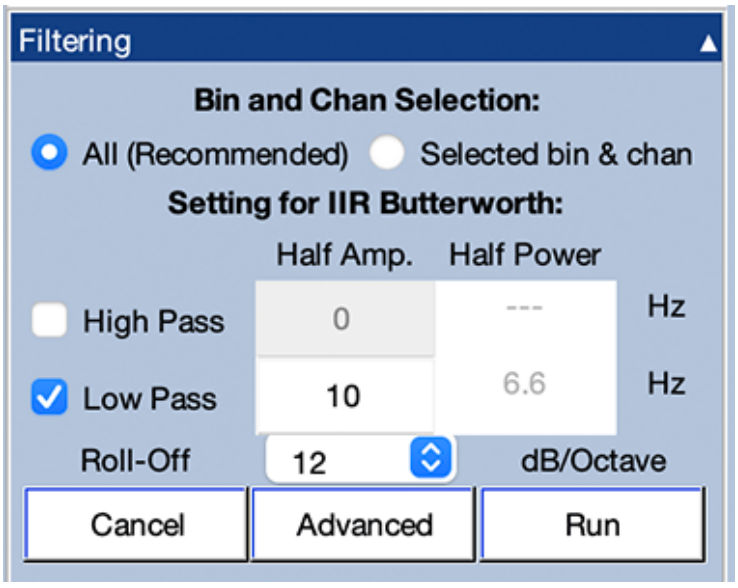
Screenshot 4.17



Our next step will be to apply the 10 Hz filter to this ERPset. The filter will be applied to each of the individual impulses in Channels 1–13 as well as the P1 wave in Channel 14. You will then see that the sum of the filtered impulses is equal to the filtered P1 wave (because the filter is linear).

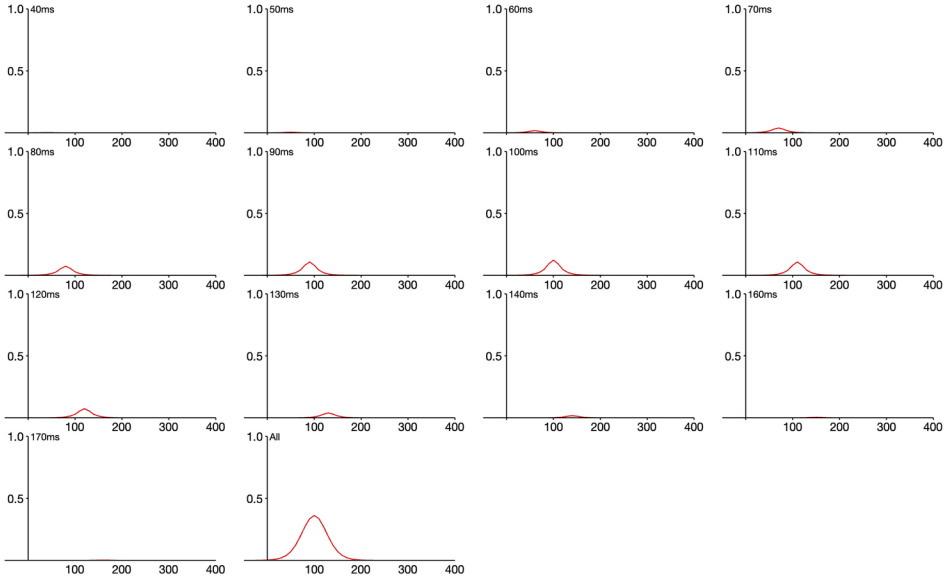
Let's do it! Go to the **Filtering** panel and set it to filter with a half-amplitude cutoff of **10** Hz and a roll-off of **12** dB/octave (as in Screenshot 4.18). Make sure that **Bin and Chan Selection** is set to **All** so that you filter all the channels. Click **Run**, and name the resulting ERPset **P1_Separated_filt10**.

Screenshot 4.18



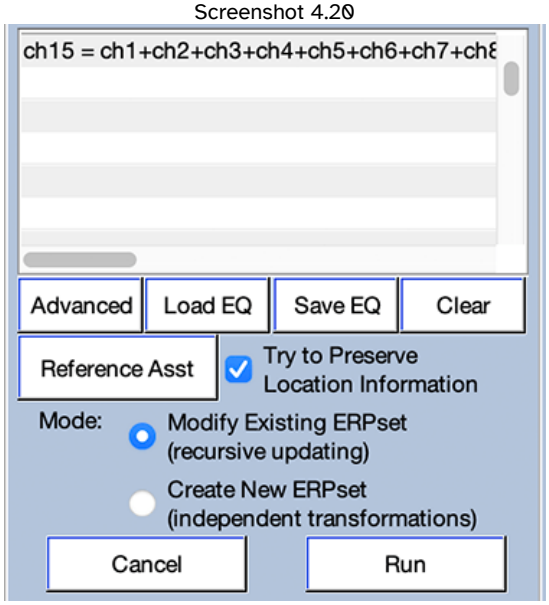
You should see something like Screenshot 4.19. Notice that each of the impulses in Channels 1–13 has been "smeared out" by the filter and looks like the filtered impulse response function that we saw in the preceding exercise. You can also see the filtered P1 wave in Channel 14.

Screenshot 4.19



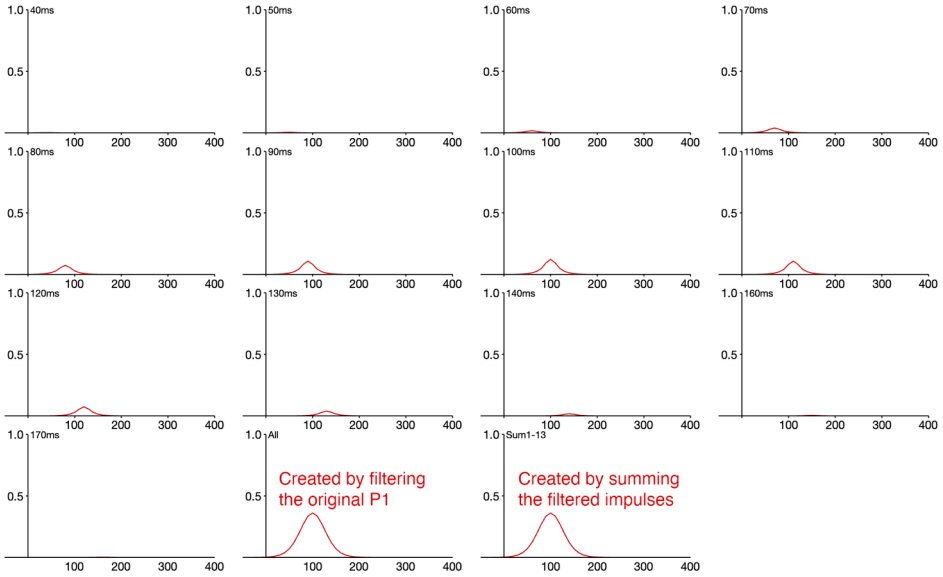
Now we're going to create a new Channel 15 that is the sum of the filtered impulses from Channels 1–13. To do this, go to the **Channel Operations** panel and type the following into the top row of the text box: **ch15 = ch1+ch2+ch3+ch4+ch5+ch6+ch7+ch8+ch9+ch10+ch11+ch12+ch13 label Sum1-13**. Once the panel is set up as shown in Screenshot 4.20, click **Run**.

Screenshot 4.20



As shown in Screenshot 4.21, you should now see that the new Channel 15, which was created by summing the filtered impulses in Channels 1-13, is exactly the same as Channel 14, which was created by filtering the P1 waveform as a whole.

Screenshot 4.21



This is analogous to what you saw in a previous exercise, where you filtered the individual P1, N1, and P2 waves and then summed them together, which created the same result as filtering the whole ERP waveform. Here, you filtered the impulses for each time point and then summed them together, which created the same result as filtering the whole P1 waveform.

Figure 4.5 puts all of this together. Panel E is a reminder of the impulse response function for the 10 Hz filter we have been using (i.e., the output of the filter when the input is a 1 μV impulse at time zero). Note that the peak amplitude is 0.14 μV.

Panel A shows that we can represent an ERP waveform as a set of impulses. [In this example, the sampling rate was 100 Hz, so there was one point every 10 ms. A sampling rate of 250 Hz (one point every 4 ms) would be more typical, but it's easier to see the impulse responses at the slower rate.]

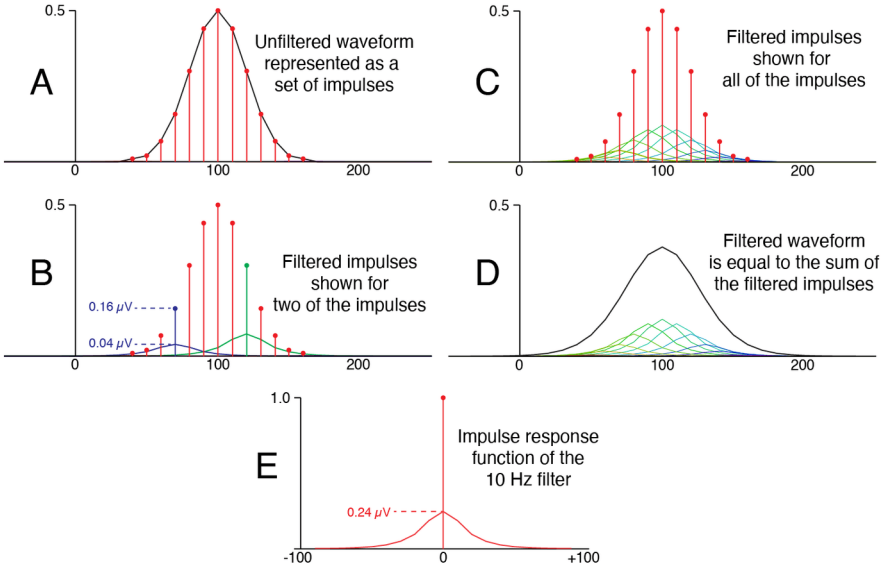


Figure 4.5. Overview of filtering in the time domain. (A) Unfiltered P1 waveform and corresponding impulses representing the voltage at each time point. (B) Filtered impulses for two of the impulses (indicated with blue and green impulses and waveforms). (C) All of the original and filtered impulses. (D) The filtered impulses along with the sum of the filtered impulses, which is equal to the filtered version of the P1 waveform. (E) Impulse response function of the 10 Hz filter used in this exercise and the preceding exercise.

Panel B shows the impulses, along with the filtered versions of two of the impulses (one shown in blue and the other shown in green). The filtered version of an impulse is essentially a copy of the filter's impulse response function that has been shifted over to the time of the impulse and scaled by the height of the impulse. For example, the blue impulse in panel B leads to a copy of

the impulse response function that is centered at the time of that impulse (70 ms) and is scaled by the height of the impulse (0.16 μV). The impulse response function was 0.24 μV at its peak, so the filtered version of the blue impulse has a peak height of 0.16 x 0.24 = 0.04 μV.

Panel C shows the filtered versions of all of the impulses. Each filtered version of an impulse is a copy of the impulse response function from Panel E that has been shifted so that it's centered at the time of the impulse and scaled by the amplitude of the impulse.

Panel D shows that the filtered version of the P1 waveform can be created by summing together all of the filtered responses to the impulses (which are scaled-and-shifted copies of the impulse response function in Panel E). In other words, to filter a waveform, you replace each point in the waveform with a scaled-and-shifted copy of the impulse response function and then sum them together.

Congratulations—you now know how filtering works in the time domain! And I hope you can see how the shape of the filtered waveform is influenced by the shape of the impulse response function. For example, if we had a narrower impulse response function, the filtered waveform would be less "smeared out" than the one shown in Figure 4.5D.

I also played a little trick on you in this exercise: I taught you the mathematical concept of convolution without you realizing it. Convolution involves the same scaling-and-shifting process that is involved in filtering. When we convolve two functions, A and B, we replace each point in A with a scaled-and-shifted copy of B and then sum them together. That's exactly what you just did when you filtered the P1 wave using the impulse response function of the 10 Hz filter: You filtered function A (the P1 waveform) by replacing each point in it with a scaled-and-shifted copy of function B (the impulse response function). In other words, filtering is achieved in the time domain by convolving the ERP waveform with the filter's impulse response function. [See the box below for one additional detail about convolution.]

An additional detail about convolution

I left out one little detail about convolution in the discussion above. When we convolve functions A and B, we first mirror-reverse function B before we do the scaling-and-shifting process. However, if B is mirror-symmetrical, we don't need to worry about this detail because the original and mirror-reversed versions of the function are identical. The software-based filters that we use for ERPs almost always have mirror-symmetrical impulse response functions, so the mirror-reversal step is unnecessary. This detail is only important for *causal* filters, which are used very rarely (see Chapter 8 in Luck, 2014, for more information).

This page titled 4.8: Exercise- Filtering a Series of Impulses (and a Sneaky Introduction to Convolution) is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 4.6: Exercise- Applying the Impulse Response Function to a Series of Impulses by Steven J Luck is licensed CC BY 4.0.

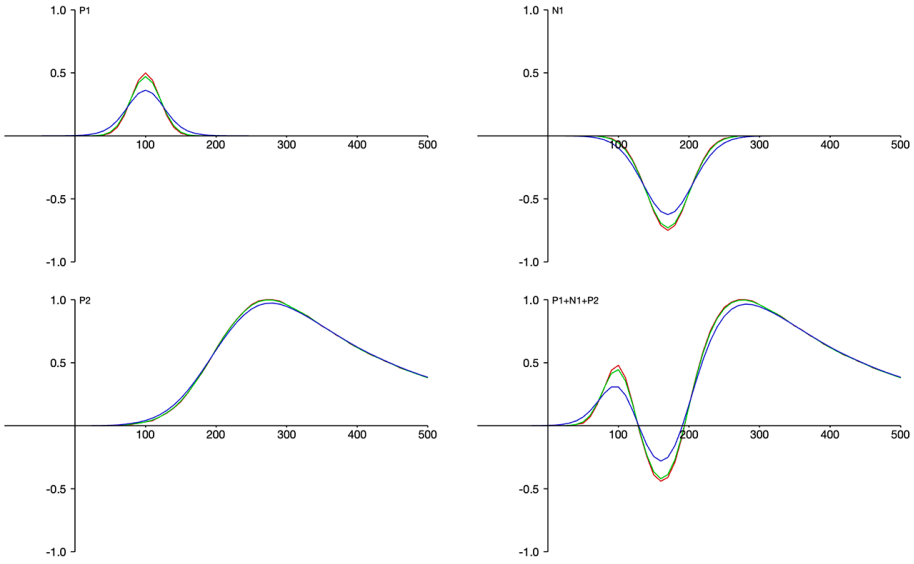
4.9: Predicting Waveform Distortion from the Impulse Response Function

In Exercise 4.4, you saw that a 10 Hz filter “smeared out” an ERP waveform, producing reduced P1 and N1 amplitudes and spreading out the onset and offset of the waveform. However, a 25 Hz filter produced hardly any distortion at all. The present exercise will show you how this pattern of distortion can be predicted from the impulse response functions of the 10 Hz and 25 Hz filters.

Before we see how this works, click the **Reset** button under the lower right corner of the plot region and then click **Reset ERPLAB Studio** in the window that appears.

Now go to the **ERPsets** panel in the **ERP** tab and load **P1+N1+P2+All_Filtered.erp**. This ERPset has the individual P1, N1, and P2 components from the previous exercises, along with the entire ERP waveform (created by summing together the three components). Each of these waveforms is in a separate channel. Bin 1 has the unfiltered data; Bin 2 has the data filtered at 25 Hz; Bin 3 has the data filtered at 10 Hz. It should look like Screenshot 4.22. You can see that the 10 Hz filter reduced the P1 and N1 amplitudes and shifted the onset and offset times of the individual components and the entire waveform.

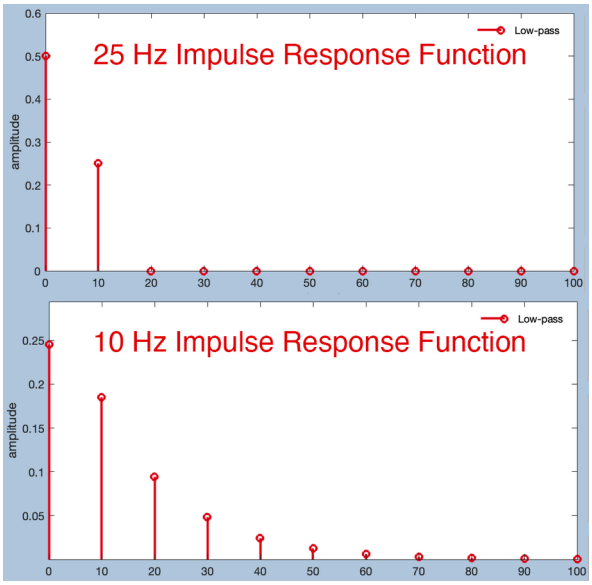
Screenshot 4.22



Now let’s see how this is related to the impulse response functions of the 10 Hz and 25 Hz filters. Go to the **Filtering** panel and click the **Advanced** button. Within the **Advanced** window, set the Low-Pass filter to **25** Hz. To see the impulse response function, select **Filter impulse responses** in the **Display** section of the **Advanced** window, and make sure the **X limits** are set to **0 100**. As shown in Screenshot 4.23 (top), this impulse response function has a peak value of about 0.5 at time zero and then falls rapidly to zero by 20 ms.

Now change the half-amplitude cutoff to **10** Hz. The impulse response function should now look like the one in Screenshot 2.23 (bottom). It has a peak value of about 0.24 at time zero and then falls gradually to zero by about 70 ms.

Screenshot 23



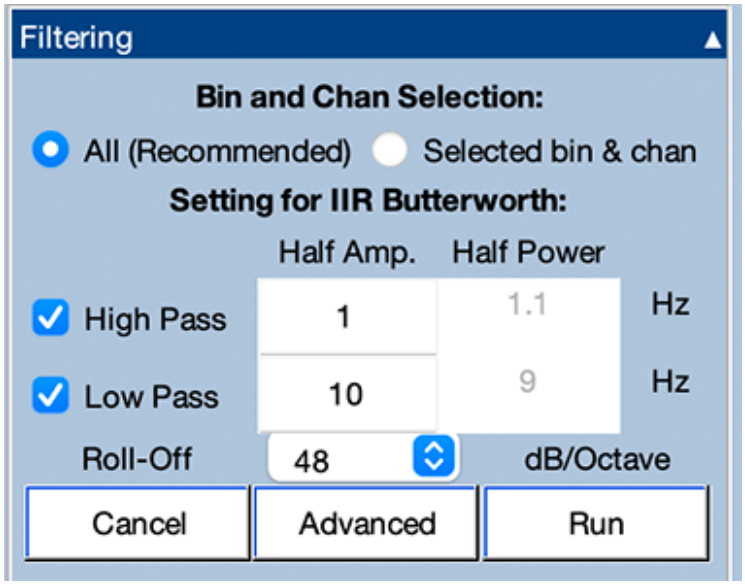
The fact that the 10 Hz impulse response function has a peak amplitude that is much lower than that of the 25 Hz impulse response function explains why the 10 Hz filter reduces the amplitudes of the P1 and N1 components much more than the 25 Hz filter (Screenshot 4.22 above). Note that the amplitude of the P2 wave is only slightly reduced by the 10 Hz filter; this is a result of an interaction between the shape of the impulse response function and the shape of the waveform that is too complicated to explain here (see the free online Chapter 12 from An Introduction to the Event-Related Potential Technique for a more complete description of filtering with an impulse response function).

The greater temporal spreading produced by the 10 Hz filter than by the 25 Hz filter is explained by the fact that the 10 Hz impulse response function is much wider than that of the 25 Hz impulse response function. Remember that, when we filter the data, each point in the ERP waveform is replaced by a scaled-and-shifted copy of the impulse response function. Replacing an early point in the ERP waveform with a broad impulse response function will cause the filtered waveform to start earlier than the unfiltered waveform. Similarly, replacing a late point in the ERP waveform with a broad impulse response function will cause the filtered waveform to end later than the unfiltered waveform.

When you look at the filtered and unfiltered versions of the overall ERP waveform (the lower right panel of Screenshot 4.23), you can see the temporal spreading at the beginning and end of the overall waveform. But you should keep in mind that each of the underlying components is also being smeared out. You can't really see this in the overall waveform, because you can't tell when a given underlying component begins and ends. But, because this is simulated data and you have the waveforms for the underlying components, you can see that their onset and offset times are also being distorted. This ability to see the smearing of the individual components is one of many advantages of looking at the effects of a filter on simulated data.

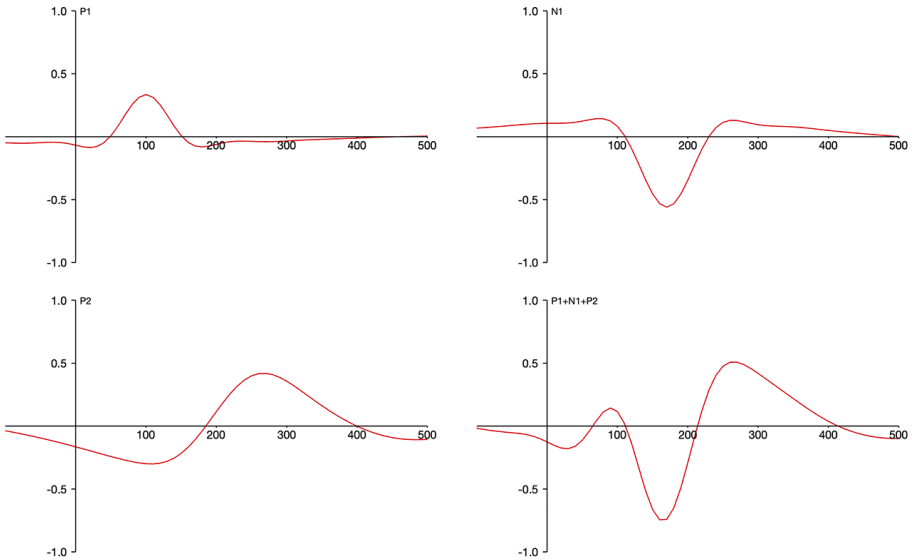
Now we're going to look at the effects of an even more extreme filter. First, load the ERPset file named **P1+N1+P2+All.erp** into the **ERPsets** panel. This ERPset contains the unfiltered data. Now go to the **Filtering** panel and select a high-pass cutoff of **1** Hz, a low-pass cutoff of **10** Hz, and a roll-off of **48** dB/octave (as shown in Screenshot 4.24).

Screenshot 4.24



Click **Run** and name the ERPset **P1+N1+P2+All_filt1-10_48dB**. The result should look like Screenshot 4.25.

Screenshot 4.25

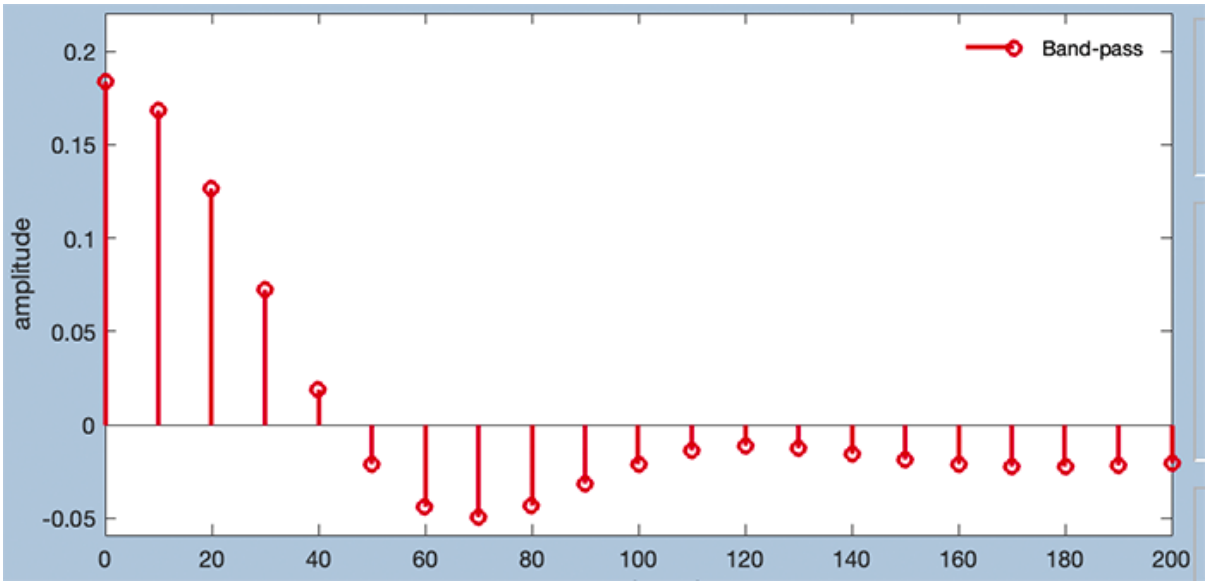


You can see a ton of distortion in the individual components and in the overall waveform. Each of these waveforms now spreads broadly in both directions, starting in the prestimulus baseline period. However, the most concerning distortion in the overall waveform is that the filter has induced an artifactual negative peak at approximately 30 ms.

By applying a filter like this to real data, you might think you've discovered a new ERP component when in fact you've just created an artifactual peak. Imagine how embarrassing it would be to publish a paper claiming that you've discovered a new component and then having other researchers demonstrate that you fell prey to a filtering artifact. This sort of thing has actually happened, and I don't want it to happen to you. This is why we have spent so much time going through the details of how filters work in the time domain. If you just look at the frequency response function of a filter, you would not predict this pattern of distortion. But it's easily predictable if you look at the impulse response function. And it's even more obvious if you try filtering some simulated ERP components and ERP waveforms.

Let's take a look at the impulse response function of this filter. With the same filter settings you used to create this ERPset, click the **Advanced** button in the **Filtering** panel, change the **Display** to **Filter impulse response**, and set the **X limits** to **0 200**. You should see something like Screenshot 4.26.

Screenshot 4.26



This view isn't always ideal, so I applied the filter to an actual impulse and created the impulse response function shown in Figure 4.6. Now you can see both sides of the function, and you can see the timing plotted the same way that eStudio plots ERP waveforms.

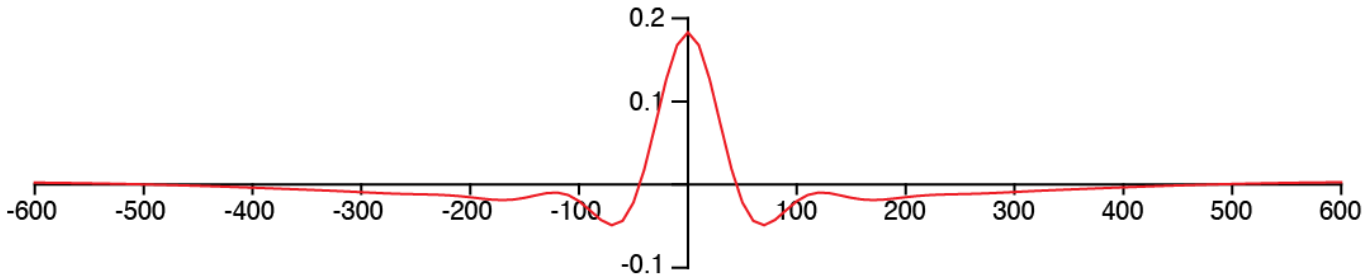


Figure 4.6. Result of filtering an impulse of amplitude 1 at time zero with a bandpass of 1-10 Hz and a roll-off of 48 dB/octave.

The first thing to notice is that this function extends for hundreds of milliseconds both before and after time zero. Because filtering replaces each point in the ERP waveform with a scaled-and-shifted copy of the impulse response function, you can immediately see that this filter is going to spread the onset and offset of the waveform by a large amount.

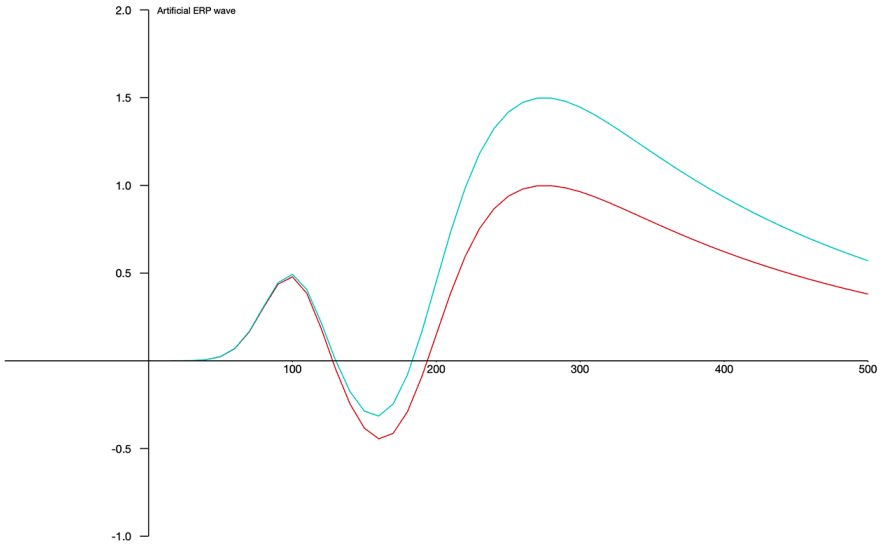
The second thing to notice is that this function contains both a positive peak around time zero and then negative values at longer latencies. Let's think about the effects of these negative values for the time point at 250 ms in the P2 component. There is a large positive voltage at this time point in the unfiltered waveform. Filtering the waveform replaces this voltage with a scaled-and-shifted copy of the impulse response function. The negative values from 50-400 ms on each side of the impulse response function will then become negative values up to 400 ms before and after 250 ms in the filtered values. If you look at the filtered P2 wave, you will see that it contains large and long-lasting negative values before and after the time period of the original unfiltered P2 wave. It doesn't look exactly like the impulse response function, of course, because the scaling-and-shifting process is repeated for every time point, and then the results are summed together.

Now let's take a look at a simulation showing how these artifacts can lead to completely incorrect conclusions when you are comparing two different experimental conditions or two different groups of subjects. I mentioned this simulation in the opening of the chapter (to get you motivated to go through all the exercises you've now completed). Now we're going to take a closer look.

Start by loading the ERPset named **Simulated P2 Effect.erp**. This ERPset is a simulation of two experimental conditions, A and B. As in previous simulations, the Condition A waveform is the sum of a P1 wave, an N1 wave, and a P2 wave. Condition B is the same, except that the amplitude of the P2 wave was increased by 50% (see Screenshot 4.27). As you can see from the waveforms, this effect begins at approximately 100 ms, which is because the P2 wave begins at that time. Condition B remains more positive than Condition A from 100 ms until the end of the waveforms. Note that we could also think of this as a simulation of a control

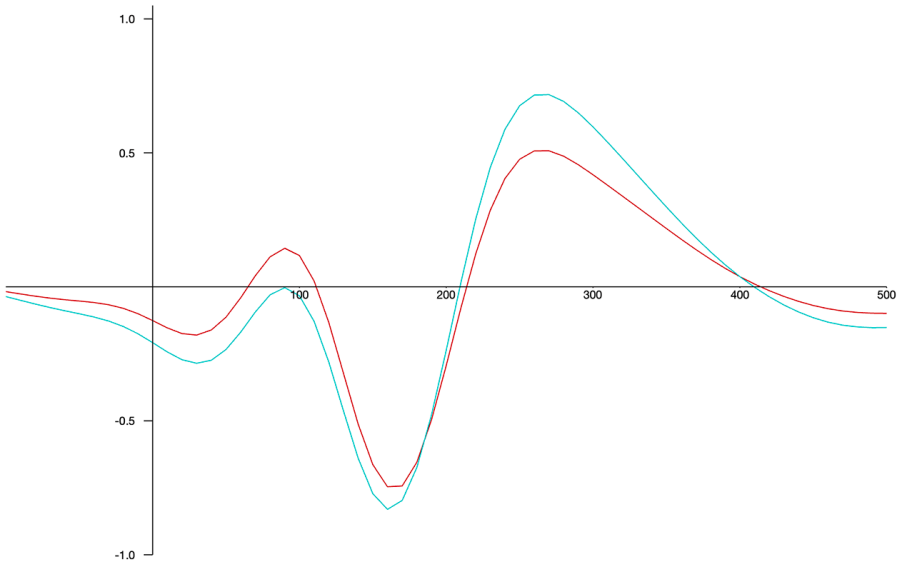
group and a patient group rather than two experimental conditions.

Screenshot 4.27



Now go to the Filtering panel and filter this ERPset with the parameters from the last several screenshots (high-pass cutoff of 1 Hz, low-pass cutoff of 10 Hz, roll-off of 48 dB/octave). You can name the resulting ERPset **Simulated P2 Effect_filt1-10_48dB**. The result should look like Screenshot 4.28.

Screenshot 4.28



This is the set of waveforms that you saw in the opening of the book. Here's what you read in that section: *If you were to describe these results in a journal article, you might say that both the N1 and P2 components are larger in Condition B than in Condition A. Or you might be more careful and say that there is greater extended negativity in Condition B than in Condition A up until the peak of the N1 wave, along with a greater positivity in the P2 latency range for condition B than for Condition A. But both of these descriptions contain a clear falsehood that is a result of filtering.*

Now we can explain why filtering produced this bogus increase in negative voltage in the early part of the Condition B waveform. Condition B had a larger P2 than Condition A. We saw earlier that the broad set of negative values in the filter's impulse response causes the large positivity of the P2 to be preceded and followed by artifactual negative voltages (see the lower left panel of Screenshot 4.25). Because the P2 was bigger in Condition B than in Condition A, this negative artifact was also larger in condition B than in Condition A. And this is what is primarily responsible for the bogus difference between conditions in the early part of

the filtered waveforms.

I hope you now feel like you have a general understanding of how filtering works in the time domain, why it's important to look at a filter's impulse response function, and why it's valuable to look at the effects of a filter on simulated waveforms. The next sections of the book will show you how to take what you've learned and use this knowledge to select filter settings that will be effective at reducing noise but will avoid producing the kinds of waveform distortions that you've seen in this exercise. But before we get to that, we need to take a look at the distortions produced by high-pass filters. That will be the focus of the next exercise.

But first I want to remind you of something I said earlier, namely that filtering in the time domain is mathematically equivalent to filtering in the frequency domain. You will get the same waveform distortion no matter whether the filtering is implemented in the time domain or the frequency domain. However, it's a lot easier to understand why filters distort ERP waveforms when we think about them in the time domain rather than the frequency domain.

This page titled 4.9: Predicting Waveform Distortion from the Impulse Response Function is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck.

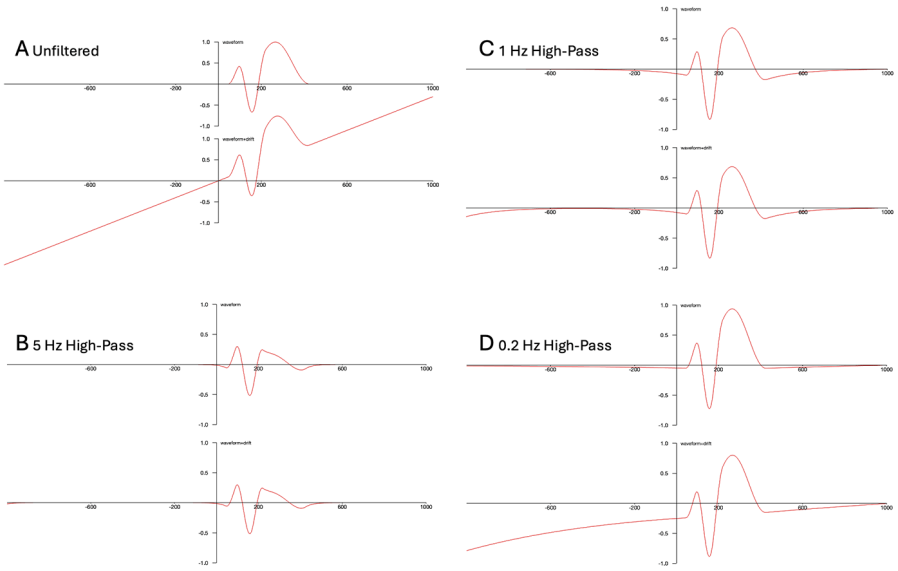
4.10: Exercise- High-Pass Filtering

Up to this point, we've focused mainly on using low-pass filters to filter averaged ERPs. The principles are exactly the same for filtering epoched or continuous EEG. The principles are also the same for high-pass filters, except that the impulse response functions are quite different.

With typical parameters (e.g., a 0.1 Hz half-amplitude cutoff frequency), high-pass filtering requires a very long impulse response function, which can lead to "edge artifacts" in epoched EEG and ERP data (see Chapter 7 in Luck, 2014). Consequently, I recommend applying high-pass filters only to continuous EEG in almost all cases. However, it's a little easier to create simulated waveforms and visualize them with ERPs than with continuous EEG, so we'll apply high-pass filters to simulated ERP data in this exercise. As a compromise, I created simulated waveforms with a longer-than-usual epoch (from -1000 to +1000 ms). But with real data, you'd almost always want to apply high-pass filters to the continuous EEG data rather than to averaged ERPs, even if you have a long epoch length.

To get started, reset everything with the **Reset** button, go to the ERPsets panel in the ERP tab, and load the ERPset file named **waveform+drift.erp**. You should see waveforms like those shown in Screenshot 4.29A. Channel 1 is the same artificial waveform we used in Exercises 4.3 and 4.4, but with a longer epoch. Channel 2 is the same waveform, but with a linear drift superimposed on it (which is the kind of noise typically produced by artifactual potentials arising from the skin and picked up by the EEG electrodes). Channel 3 (not shown in the screenshot) is an impulse that we can use to visualize the impulse response functions of the filters we'll be using.

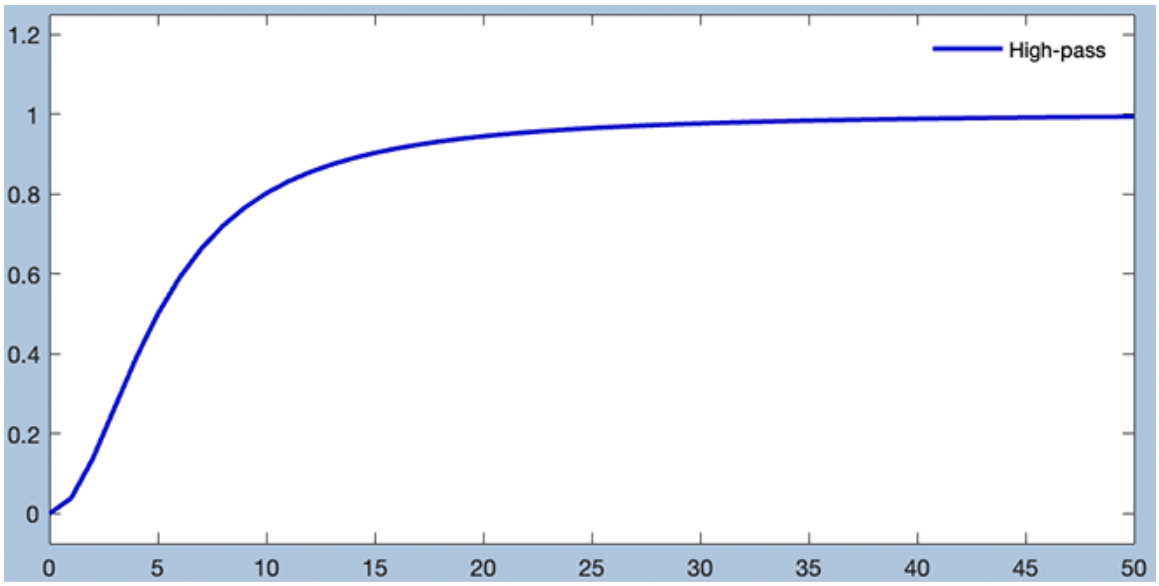
Screenshot 4.29



We're going to start by using a high-pass filter cutoff of 5 Hz, which means that we're filtering out frequencies below 5 Hz. I wouldn't ordinarily recommend using this cutoff with real data (except for a few special cases, such as research on the auditory brainstem response). However, this will make it easier to see what the impulse response function looks like and how it impacts the filtered waveform.

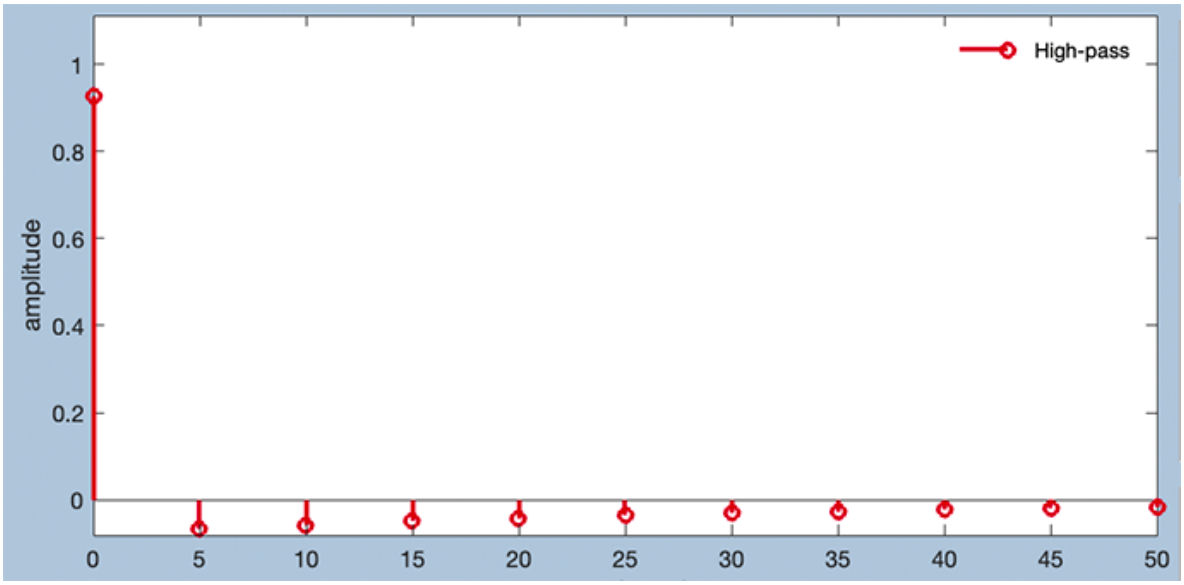
Go to the **Filtering** panel, turn off the low-pass filter, turn on the high-pass filter with a cutoff of **5** Hz, and set the roll-off to **12** dB/octave. Now click the **Advanced** button to see the frequency response function for this filter. As shown in Screenshot 4.30, the gain is near zero for the lowest frequencies, reaches 0.5 at 5 Hz (because that's the half-amplitude cutoff frequency), and then starts nearing the asymptote of 1.0 by approximately 10 Hz.

Screenshot 4.30



Now look at the impulse response function by changing the **Display** setting from **Filter frequency response** to **Filter impulse response**. It should look like Screenshot 4.31. It's very different from the impulse response function of a low-pass filter. Low-pass and high-pass filters have opposite effects (removing high versus low frequencies), so they have largely opposite impulse response functions. Whereas the low-pass impulse response functions we've looked at had positive values near time zero, this high-pass impulse response function is negative near time zero (but is near 1.0 right at time zero). The reasons for this are discussed in Chapters 7 and 12 of Luck (2014). Here, you'll just have to take my word for it.

Screenshot 4.31



Click **Run** to apply the filter. You should see something like Screenshot 4.29B. First look at the impulse (Channel 3), which now shows the impulse response function of the filter. You can see the negative values surrounding time zero, but they're pretty small. This is because the impulse response function for a high-pass filter must sum to zero. If the impulse response extends for a long time period, the individual values must be very small.

Now look at the waveform+drift channel (Channel 2). The good news is that the filter has virtually eliminated the drift. The bad news is that the filter has produced artifactual negative peaks at the beginning and end of the waveform. You can also see these artifactual peaks in the channel without the drift (Channel 1). The filter has also reduced the amplitude and changed the shape of the ERP waveform. That's partly because the negative voltages produced by the negative values of the impulse response function partly cancel out the positive voltages of the P2 wave. These distortions are why I would never recommend using a filter like this

with real data (except for the auditory brainstem responses, which are largely confined to higher frequencies).

If you think about the impulse response function, you can understand why the filter produces artificial negative peaks at the beginning and end of the waveform. The unfiltered waveform starts and ends with positive values. When we replace these values with the impulse response function, the negative values to the left and right of the current point in the impulse response function produce negative values before and after the positive peaks. Note that if the waveform began and ended with negative peaks, the artifactual peaks would be positive (because a negative voltage from the unfiltered waveform multiplied by a negative value in the impulse response function creates a positive value).

Next we're going to try a filter that's not quite as extreme, but still has a higher cutoff frequency than I'd ordinarily recommend, namely 1 Hz. Make the original waveform+drift ERPset active again in the ERPsets panel, change the cutoff from **5** Hz to **1** Hz in the **Filtering** panel, and run the filter.

The resulting ERPset should look like Screenshot 4.29C. For a clearly view, you can go to the ERPsets panel and cycle back and forth among the unfiltered ERPset, the 5 Hz filtered ERPset, and the 1 Hz filtered ERPset. The 1 Hz filter still largely eliminates the drift in Channel 2, but without reducing the amplitude of the ERP waveform as much as the 5 Hz filter, so that's an improvement. However, the artifactual negative peaks at the beginning and end of the waveform are still present. That's why I wouldn't ordinarily recommend a 1 Hz cutoff.

Now let's try a high-pass filter cutoff that we have found to be ideal for studies of the P3b and N400 components, namely 0.2 Hz (see Zhang et al., 2024b). Make the original **waveform+drift** ERPset active again in the ERPsets panel, change the cutoff from **1** Hz to **0.2** Hz in the Filtering panel, and run the filter. As shown in Screenshot 4.29D, the filter has not completely eliminated the drift in Channel 2. However, it has produced no noticeable distortion of the ERP waveform. That is, the filter hasn't attenuated the waveform, and it hasn't produced any artifactual peaks. In other words, it didn't filter out the low-frequency noise as well as the other filters, but it produced much less waveform distortion. This is an inevitable trade-off with high-pass filters: You will either filter out all the noise but have significant waveform distortion or fail to filter out all the noise and not produce waveform distortion.

Most low-frequency drift happens on the single trials as a result of skin potentials, with an upward-going shift on some trials and a downward-going shift on other trials. The residual drift that you can see in Channel 2 of the data filtered at 0.2 Hz won't typically be present in averaged ERPs. In fact, if you have a reasonable number of trials, the averaged ERP should look like what you saw in Channel 1 (which is almost identical to the unfiltered data).

However, the fact that some drift remains on the single trials does mean that we will have more trial-to-trial variation in the amplitudes of our ERP components, which decreases our ability to precisely estimate a participant's true amplitudes. So, there is a downside to a filter that doesn't completely eliminate the low-frequency drifts on the single trials. As we will see in the following exercises, your goal in selecting filter settings should be to get the best possible signal quality without producing significant waveform distortion. You will need to balance these two factors, because the settings that produce the very best signal quality will often produce too much distortion. The following exercises will give you a principled method for balancing these two factors.

Evolving recommendations

In the first edition of An Introduction to the Event-Related Potential Technique (Luck, 2005), I recommended using 0.01 Hz as the half-amplitude cutoff. Over the following years, however, my collaborators and I systematically compared a variety of different cutoffs, and we typically found that 0.1 Hz produced the best noise reduction without any substantial distortion of the waveforms (Kappenman & Luck, 2010; Tanner et al., 2015). More recently, Guanghui Zhang, David Garrett, and I developed a more formal approach to determining the optimal settings, and we found that 0.2 Hz is best for the P3b and N400 components, with even higher cutoffs for some other components (Zhang et al., 2024b). This is an example of how scientific methods evolve over time. We started with recommendations that were based on a combination of lab lore and informal empirical testing. We then moved to more rigorous empirical testing, and we later moved to even more rigorous testing combined with a more detailed analysis of what is needed for different types of studies.

This page titled 4.10: Exercise- High-Pass Filtering is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck.

4.11: Background- How Should We Define the Optimal Filter?

Before I explain our procedure for determining optimal filter settings, we need to discuss the question of what it means for a filter to be optimal for ERP research.

First, a filter needs to be able to reduce noise, as illustrated in Figure 4.7. Panel A shows a 30-second period of EEG from Subject 6 in the ERP Core N400 experiment without any filtering. You can see that it contains both high-frequency noise (very rapid up-and-down shifts that make the waveforms look "fuzzy") and low-frequency noise (gradual drifts over time, such as the pink waveform from the P03 electrode site that falls completely out of view just before 24 seconds). Panel B shows the same data after application of a low-pass filter with a half-amplitude cutoff at 25 Hz and a 12 dB/octave roll-off. This filter reduces the high-frequency "fuzz". Panel B shows what happens when we add a high-pass filter with a half-amplitude cutoff at 0.2 Hz and a 12 dB/octave roll-off. This filter eliminates the gradual drifts over time, keeping the waveforms largely parallel to the horizontal axis.

The point of Figure 4.7 is that part of the definition of an optimal filter is that it should do a good job of reducing the noise in the data.

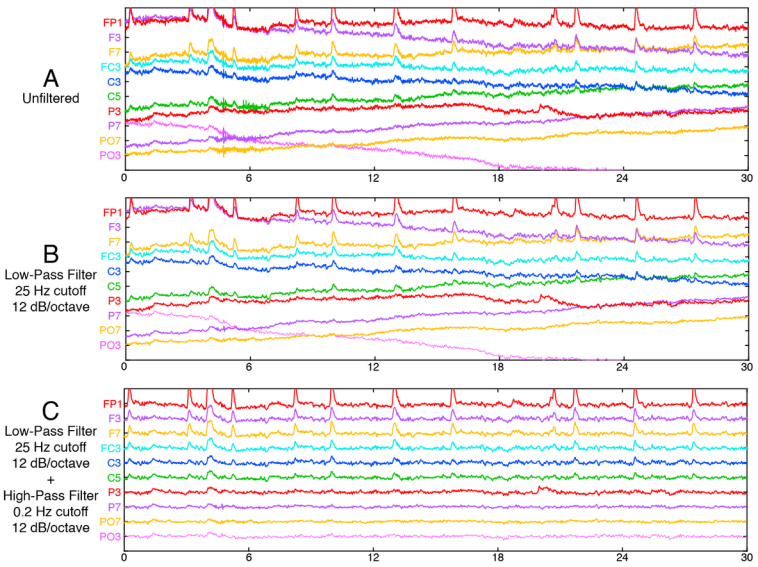


Figure 4.7. Application of low-pass and high-pass filters to a 30-second period of EEG. (A) Unfiltered data. (B) Result of applying a low-pass filter with a half-amplitude cutoff of 25 Hz and a roll-off of 12 dB/octave. (C) Result of applying both a low-pass filter with a half-amplitude cutoff of 25 Hz and a high-pass filter with a half-amplitude cutoff of 0.2 Hz (roll-off of 12 dB/octave for both).

The logic behind using these filters is that the EEG recordings contain a lot of noise in really low frequencies and in really high frequencies, whereas the ERP signals of interest are mainly in intermediate frequencies. Consequently, if we filter out the really low and really high frequencies, we will be getting rid of a lot of noise without having much impact on the signals of interest. However, if we don't set our cutoff frequencies appropriately, some of the signal may be lost along with the noise.

This "loss of signal" problem is illustrated in Figure 4.8, which shows what happens when we use a low-pass cutoff that is too low and a high-pass cutoff that is too high. Panel A illustrates this by showing the impact of a 5 Hz low-pass filter on a simulated N170 component. The N170 is very narrow and sharp, meaning that it has a lot of energy above 5 Hz. This energy is eliminated by the filter, greatly reducing the amplitude of the N170. Panel B shows a simulated P3b component; it has much less high-frequency energy, so its amplitude isn't reduced very much by the 5 Hz low-pass filter. This brings up an important point: Whether or not a filter reduces the size of the signal depends on the frequency content of the signal, which will be different for different ERP components.

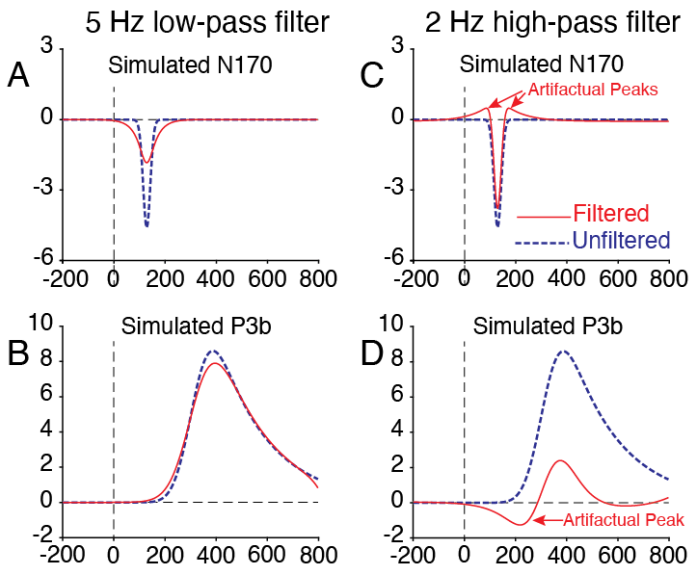


Figure 4.8. Examples of how filtering can decrease the amplitude of the ERP components in a way that depends on the nature of the filter and the nature of the ERP component. A 5 Hz low-pass filter dramatically reduces the amplitude of a simulated N170 (A) but not a simulated P3b (B). By contrast, a 2 Hz high-pass filter has relatively little impact on the amplitude of the N170 but dramatically decreases the amplitude of the P3b. The 2 Hz high-pass filter also produces artifactual peaks in both the N170 and P3b waveforms.

Panels C and D show what happens when we instead use a high-pass filter. This filter has only a minimal impact on the N170, but it dramatically reduces the amplitude of the P3b. This is because the N170 has very little energy below 2 Hz, whereas the P3b has a lot. Once again, the impact of a filter depends on the waveshape (and therefore the frequency content) of the ERP component. Low-pass filters tend to reduce the amplitude of sharp, narrow components (which are typically the initial sensory responses), whereas high-pass filters tend to reduce the amplitude of slow, broad components (which are typically the later cognitive responses).

When we combine what we've learned from Figures 4.7 and 4.8, we see that we want a filter that will decrease the noise as much as possible without decreasing the signal by very much. We can make this more formal by considering the *signal-to-noise ratio*, which is simply the size of the signal divided by the size of the noise. A filter is typically useful only if it decreases the noise more than it decreases the signal, resulting in an increased signal-to-noise ratio. And the best filter is the one that gives us the best signal-to-noise ratio, subject to two caveats.

The first caveat is that filters may also distort the waveform of the ERP. As you saw in the preceding exercises and can see in Figure 4.8A, low-pass filters tend to cause the onset of a waveform to be shifted earlier and the offset to be shifted later. And as you have also learned, high-pass filters tend to produce artifactual peaks in the waveform such as those in Figures 4.9C and 4.9D. Thus, you will want to choose a filter that gives you the best signal-to-noise ratio without producing unacceptable levels of waveform distortion.

The second caveat is that we don't care much about whether the filter reduces the amplitude of an ERP component when our goal is to assess the latency rather than the amplitude of the component. Of course, the latency scores might be affected if the amplitude gets so small that it's hard to determine the latency of the component. But outside of that situation, we don't really care about the amplitude reduction shown in Figure 4.8 when we're scoring latencies instead of amplitudes.

We're now ready for our formal definition of what constitutes the optimal filter for a given ERP study: **the optimal filter is the one that maximizes the data quality without producing unacceptable levels of waveform distortion**. We define data quality as the signal-to-noise ratio when we're scoring amplitudes, and we define data quality simply in terms of the noise when we're scoring latencies.

This page titled 4.11: Background- How Should We Define the Optimal Filter? is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck.

- 4.6: Exercise- Applying the Impulse Response Function to a Series of Impulses by Steven J Luck is licensed CC BY 4.0.

4.12: Background- Quantifying Signal, Noise, and Waveform Distortion

In the preceding section, we defined the optimal filter for a given ERP study as the filter that maximizes the data quality without producing unacceptable levels of waveform distortion. We define data quality as the signal-to-noise ratio when we're scoring amplitudes, and we define data quality simply in terms of the noise when we're scoring latencies.

To make this definition useful in practice, we also need to define what we mean by signal, what we mean by noise, and what we mean by unacceptable levels of waveform distortion. That's what we will do in this section.

Defining and Quantifying the Signal

We define the signal in terms of the amplitude or latency score you are trying to obtain in a given study. In the context of Figure 4.8 from the previous section, this might be the mean amplitude from 300-500 ms for the P3b component (Figure 4.8BD), or it might be the latency of the most negative voltage between 150 and 200 ms for the N170 component (Figure 4.8AC). You might be surprised that we are defining the signal in terms of relatively abstract scores like these rather than something like the amplitude of the raw EEG. However, these scores are what you will be putting into your statistical analyses and using to test your scientific hypotheses, so these are the signals you care about. And, as illustrated in Figure 4.8, a given filter can have very different effects on different ERP components, so we want to define the signal in a way that reflects the component we will be scoring.

Figure 4.9 illustrates another important reason to use the score as the signal. The top panel in this figure shows the N400 difference wave (unrelated minus related) from the ERP Core N400 experiment that was the focus of Chapters 2 and 3. The bottom panel shows the same waveform with high-frequency noise added (the kind of noise you would get from muscle activity).

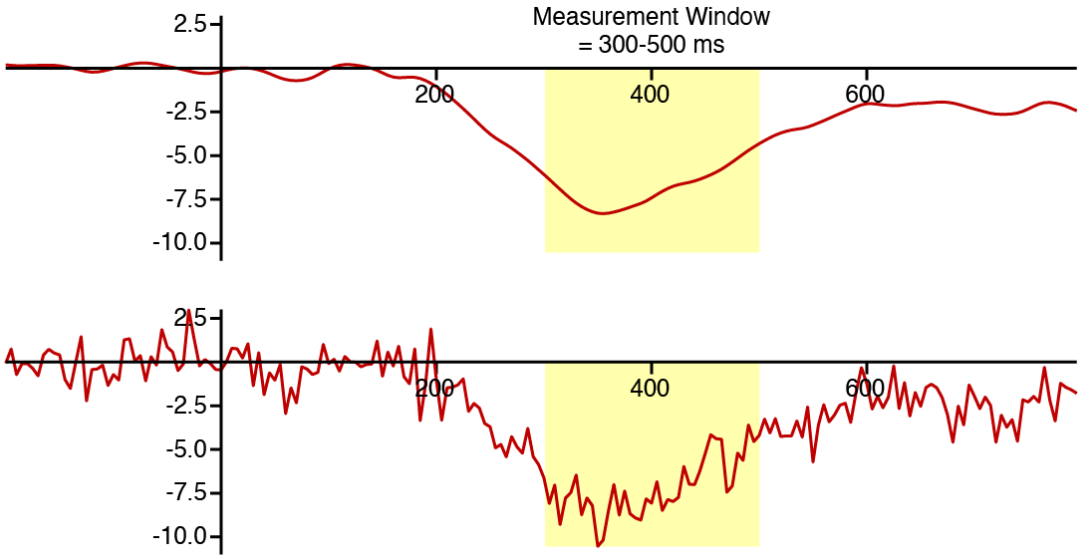


Figure 4.9. N400 difference waveform (unrelated minus related) and the same waveform with high-frequency noise added.

Imagine that you wanted to quantify N400 amplitude as the most negative voltage between 300 and 500 ms (the voltage of the N400 peak). The high-frequency noise could have a big impact on the score you obtain. Now imagine that you decided to quantify N400 amplitude as the mean voltage between 300 and 500 ms. With this scoring method, the rapid upward and downward voltage deflections ("uppies" and the "downies") in the high-frequency noise will largely cancel out, so the high-frequency noise will have only a minimal impact on the N400 score. Consequently, applying a low-pass filter to the data prior to scoring will improve the N400 peak amplitude score but will have little impact on the N400 mean amplitude score. This shows yet again that the optimal filter settings will depend on the specific score we are obtaining in a given experiment.

Defining and Quantifying the Noise

Now that we've defined the signal, we're ready to define the noise. You might be expecting the noise to be defined in terms of the raw EEG. However, you just saw in Figure 4.9 that the effect of a given type of noise depends on how you are scoring the ERP waveform. To take this into account, we're going to define noise in terms of any factors that cause our scores to be imprecise.

Precision is classically defined in terms of whether you get the same value every time you try to measure something. For ERPs, a score is precise to the extent that we would get the same score every time we repeated the experiment in a given participant (assuming no learning or fatigue). If we repeat the N400 experiment 100 times in a given participant, measure the N400 amplitude for each experiment, and find that our N400 amplitude score is between 7.1 and 7.3 μV for all 100 repetitions, that would be a very precise score!

This is what the standardized measurement error (SME) is designed to quantify. You may recall that the SME was introduced in Chapter 2.10 (see Luck et al., 2022 for a detailed description). Whenever you compute averaged ERPs in ERPLAB, you will get the SME for a default set of time windows (e.g., 300-400 ms). This SME value reflects the imprecision of the mean voltage score from a given time window. You can also specify a custom time window that matches the measurement window you're planning to use in a given experiment (e.g., 300-500 ms for the N400 experiment in Figure 4.9).

When you are scoring the amplitude of an ERP component as the mean voltage in a specific time window, it's trivial for ERPLAB to compute the SME using an analytic approach. We call this the analytic SME or aSME. It's also possible to compute the SME for other scores, such as peak amplitude or peak latency. However, this requires a much more complicated approach called bootstrapping, which leads to the bootstrapped SME or bSME.

No matter whether you are using aSME or bSME, the SME value is a great way of quantifying the noise in your data in a way that reflects the extent to which the noise impacts the specific amplitude or latency scores in your study. For this reason, we use the SME as our metric of noise when defining the optimal filter.

Bottom line: The signal is defined as the score for a given study, the noise is defined as the SME of that score, and the signal-to-noise ratio is the score divided by the SME of that score.

Defining and Quantifying Waveform Distortion

Now let's consider what we mean by "unacceptable levels of waveform distortion" and how to quantify it. I have to admit that this part is more subjective, because what is acceptable in one context might be unacceptable in another. At a minimum, the threshold for "unacceptable" is likely to differ across research areas and participant populations. But we did our best at coming up with something that could be easily quantified and is likely to work well in many situations.

Filters produce two main types of waveform distortions. First, low-pass filters may cause the filtered waveforms to begin earlier and end later than unfiltered waveforms (see Screenshot 4.9 in section 4.4 or Screenshot 4.22 in section 4.9). In most areas of research, this is not a major problem, because researchers are typically interested in the difference in timing across groups or conditions, not the absolute onset or offset time of an effect. For example, if a filter shifts the onset time of the ERPs 20 milliseconds earlier in both Group A and Group B, this won't typically change the difference in latency between Group A and Group B. Occasionally, it is useful to examine the absolute latency at which a difference wave first deviates from zero. In these cases, you can simply quantify the amount of latency shift produced by the filter and take that into account (see, e.g., Bae & Luck, 2018). But in most cases, the change in ERP onset and offset times will not lead to incorrect conclusions.

The main filter artifacts that are likely to lead to incorrect conclusions are the artifactual peaks that can be produced by high-pass filters. You saw an example in the opening of this chapter, and you saw additional examples in section 4.10. Figure 4.10 below provides another example, showing the effects of applying a high-pass filter with different cutoffs to a simulated N400 effect. With a cutoff of 0.1 Hz, the filtered and unfiltered waveforms are nearly identical. If we increase the cutoff to 0.5 Hz, we can see a distinct positive peak preceding the negative N400 wave in the filtered waveform. This artifactual positive peak is even bigger with a 1 Hz cutoff, and with a 2 Hz cutoff the artifactual positive peak is almost as large as the negative peak of the actual N400. If you used a filter of 0.5 Hz or higher in a real N400 experiment, you might think the earlier positivity is a real component, leading to completely incorrect conclusions. I think almost everyone would agree that a distortion that leads to an incorrect conclusion is an "unacceptable level of waveform distortion."

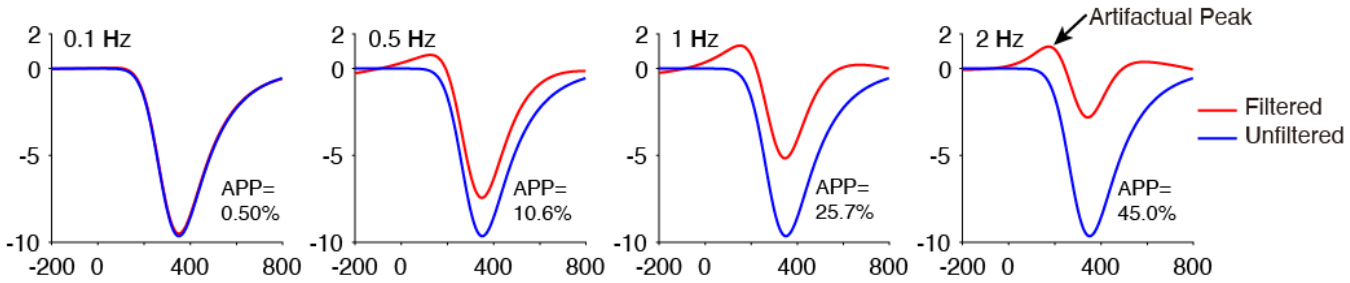


Figure 4.10. Examples of how a simulated N400 waveform is distorted by high-pass filters with different cutoffs. The amount of waveform distortion is quantified as the *artifactual peak percentage*, which is the amplitude of the artifactual peak expressed as a percentage of the amplitude of the true peak (with both measured from the filtered waveform).

In our procedure for determining optimal filter settings, we came up with a simple way of quantifying this kind of artifactual peak. We simply measure the size of the artifactual peak and express it as a percentage of the size of the real peak. We call this the *artifactual peak percentage* or *APP*. The APP is shown for each of the filter settings in Figure 4.10. It is only 0.5% for the 0.1 Hz filter, but it jumps to 10.6% for the 0.5 Hz filter.

For reasons that are described in Zhang et al. (2024a), we obtain the APP from simulated ERP waveforms like those in Figure 4.10. In the next exercise, you will see how you can create these simulated waveforms in ERPLAB and have them approximate the true waveforms in your data. After filtering a simulated waveform, you can use the **Measurement Tool** panel to score the peak amplitude of the true peak and the artifactual peak in the filtered waveform. You'll then take the absolute value of these two peak amplitudes (to make sure we end up with a positive APP value) and then compute:

APP = 100 x absolute value of the artifactual peak amplitude ÷ absolute value of the true peak amplitude

Once you have quantified the waveform distortion as the APP, you will need to decide whether the APP is acceptable or unacceptable. This is where things get subjective and context-dependent. How much distortion is too much?

For the kind of research my lab typically conducts, where we obtain very clean data from highly cooperative participants, we decided that an APP threshold of 5% would be reasonable. That is, we avoid any filters with an APP of >5%. This would be like a 0.5 μV artifactual negative peak preceding a 10 μV P3b component or a 0.1 μV artifactual positivity preceding a 2 μV N2pc. These artifactual peaks are small enough that they would just look like noise and would not be statistically significant in the kinds of experiments we run. By contrast, if we used a filter that produced an APP of 10%, this would be like a 1 μV negativity preceding a 10 μV P3b component or a 0.2 μV artifactual positivity preceding a 2 μV N2pc. These effects might be statistically significant and therefore lead to incorrect conclusions. For these reasons, a 5% APP threshold seems reasonable in my lab's research (both our basic science research using college student participants and our translational research on schizophrenia using a broad range of patients and control subjects).

However, a 10% threshold might make more sense in other areas of research. For example, studies of infants and young children often involve noisier data and fewer trials than the research in my lab, and a 1 μV artifactual negativity would be unlikely to reach statistical significance in a P3b study in infants. However, such studies might really benefit from using a 0.5 Hz cutoff instead of a 0.1 Hz cutoff, because the higher cutoff will reduce more of the low-frequency noise.

I can't tell you what APP threshold you should use in your research. The higher the threshold, the more aggressive you can be with your filters, leading to less noise and more statistical power. But higher thresholds might lead to incorrect conclusions if the artifactual peaks are large enough that they might often be statistically significant. I'm hopeful that the different subareas of ERP research will, over time, reach a consensus about the most appropriate APP thresholds in those subareas.

Putting It All Together to Find the Optimal Filter

Now that you've learned how we quantify data quality and waveform distortion, it's time to put everything together and describe the process of finding an optimal filter. The process is described in more detail by Zhang et al. (2024a), and you can find details of how we applied it to the 7 components of the ERP CORE in Zhang et al. (2024b). In addition, you can find example Matlab scripts at https://osf.io/98kqp/.

Here are the basic steps of the process:

1. Find a dataset that is typical for your research in terms of the ERP component(s) you will be examining, the participant population, and the type of EEG recording system. If you don't already have an appropriate dataset, you can probably find one online. There is a list of online EEG/ERP datasets on github, and many recent papers provide access to their data. Note that it is not usually appropriate to apply our procedure for determining optimal filter settings to a dataset and then use those filter settings to analyze the same dataset. That would be "double dipping".
2. Create one or more artificial waveforms that simulate what the effect of interest would look like in the absence of noise.
3. Apply many different filters to these waveforms and measure the artificial peak percentage (APP) to see which filters produce acceptable versus unacceptable levels of waveform distortion.
4. Apply many different filters to the real data and quantify the size of the signal and the magnitude of the noise after the application of each filter.
5. Select the filter settings that produce the best signal-to-noise ratio (or lowest noise level for latency scores) without producing unacceptable levels of waveform distortion.

Step 1 is up to you, but the following exercises will show you how to implement Steps 2-5 using ERPLAB Studio.

This page titled 4.12: Background- Quantifying Signal, Noise, and Waveform Distortion is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck.

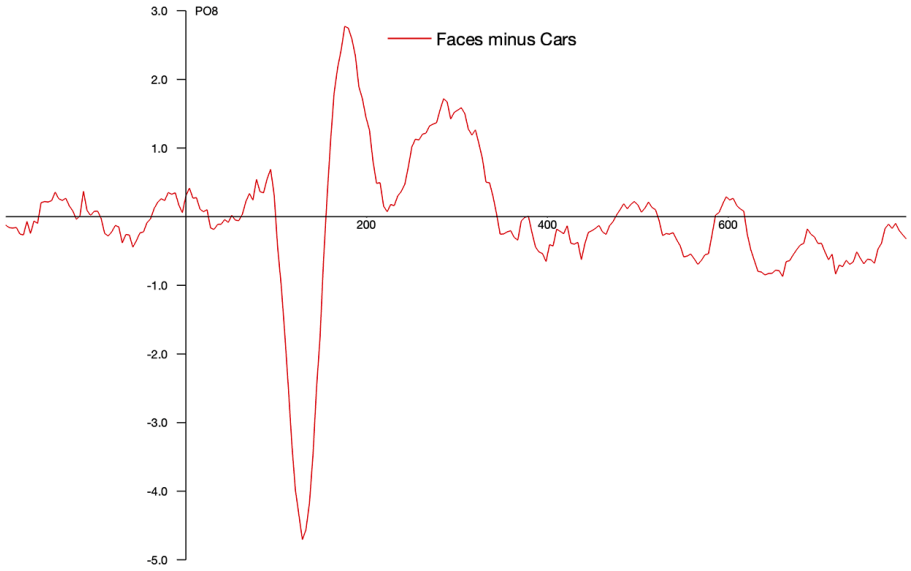
4.13: Exercise- Creating artificial waveforms

This exercise will focus on the second step of the process that was outlined at the end of the preceding section, namely creating one or more artificial waveforms that will be used to assess waveform distortion.

ERPLAB Studio contains a tool for creating these waveforms that you can access via the **Create Artificial ERP Waveform** panel in the **ERP** tab (or via scripts). This tool allows you to create an artificial waveform with a small number of parameters and overlay this with a real waveform that provides an approximation of the true effect. Let's take a look at how this works, first for the N170 component and then for the N400 component. Then we'll consider the question of how you can make an educated guess about what the true effect will look like for less well-studied components.

Start by clicking the **Reset** button under the lower right corner of the plot region and then clicking **Reset ERPLAB Studio** in the window that appears. Then load the ERPset named **N170.erp** using the **ERPsets** panel and select only the **PO8** electrode site (channel **26**) from the **Bin & Channel Selection** panel. The result should look like Screenshot 4.32.

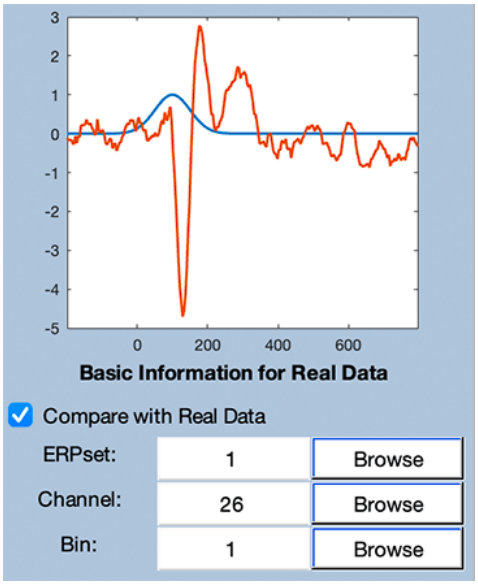
Screenshot 4.32



You are now looking at the grand average N170 component from the ERP CORE (Kappenman et al., 2021). In this experiment, participants saw faces, cars, scrambled faces, and scrambled cars, and on each trial they pressed a button to indicate whether or not the stimulus was scrambled (irrespective of whether it was a face or a car). In the ERPset you just loaded, the N170 component was isolated by constructing a faces-minus-cars difference wave. This is a common approach, which subtracts away any brain activity that is equivalent for faces and cars, leaving the differential activity. This differential activity is mainly the N170 component (which is primarily the additional activity associated with face processing), although there are some smaller positive components after the N170.

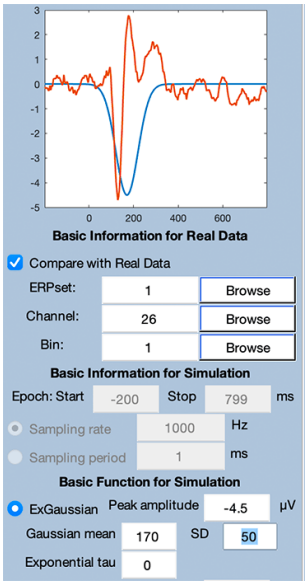
Now we're going to create an artificial waveform that closely resembles the N170 component (without the subsequent positivities). Go to the **Create Artificial ERP Waveforms** panel, and check the box labeled **Compare with Real Data** near the top of the panel. We have only one ERPset, so you can leave the ERPset box set to **1**. We want to select the **PO8** channel, so click the **Browse** button next to the **Channel** box, click on the **P08** channel (which is channel **26**), and click OK. You should now see **26** in the **Channel** box. There is only one bin, so you can leave the **Bin** box at **1**. You should see something like Screenshot 4.33.

Screenshot 4.33



The real N170 grand average should be shown in red, and the current artificial ERP waveform should be shown in blue. It's obviously not a good match, because the artificial waveform is a positivity peaking around 100 ms. Let's change the artificial waveform so that it's a negativity that more closely matches the N170, with an amplitude of approximately -4.5 μV and a latency of approximately 170 ms. Within the panel, go to the section labeled **Basic Function for Simulation** and change the **Peak amplitude** to **-4.5** and the **Gaussian mean** to **170**, with the **SD** set to **50**. This should create an artificial waveform with an amplitude of -4.5 μV and a latency of 170 ms. As shown in Screenshot 4.34, the match is better, although the artificial waveform is too wide and it peaks too late.

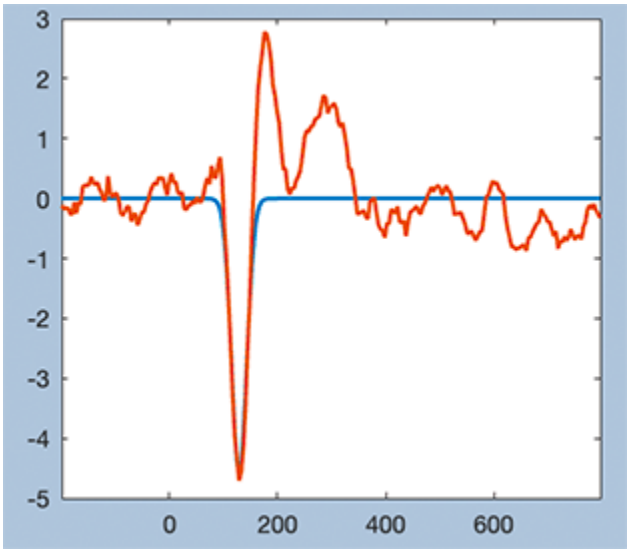
Screenshot 4.34



As you can probably tell, this tool is using a Gaussian (normal distribution) function to simulate the waveform. Try playing around with the latency (the mean of the Gaussian), the width (the standard deviation or SD of the Gaussian) and the peak amplitude (the height of the Gaussian) to get it to better match the observed N170.

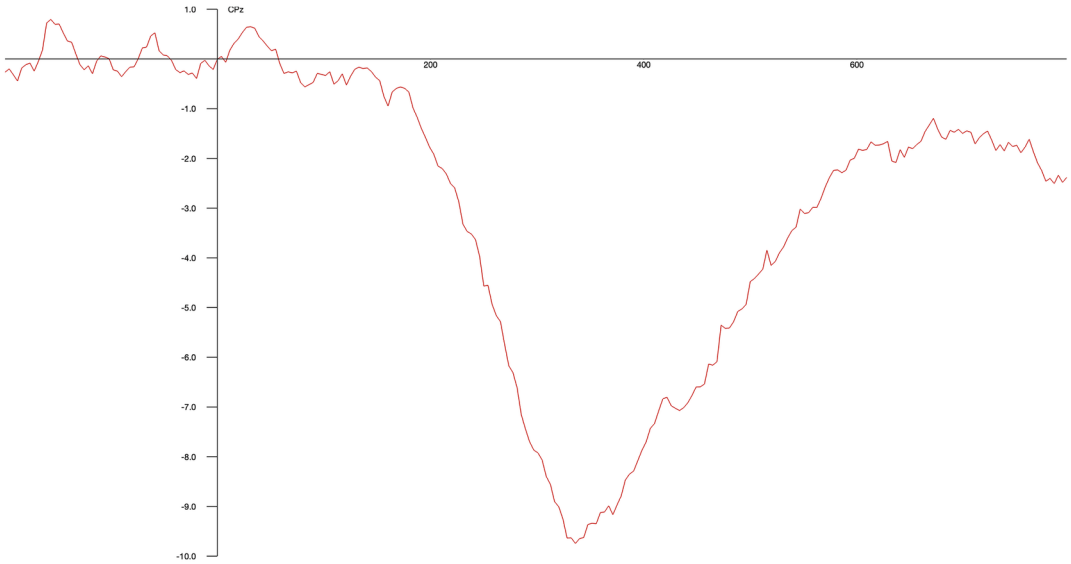
In our journal article, we found the best fit with a Peak amplitude of -4.6, a Gaussian mean of 129 ms, and an SD of 14 (see Screenshot 4.35). It's not perfect, and it doesn't attempt to account for the positive waves that follow the N170, but it's a pretty good match to the N170. Once you are satisfied with the match between your simulated N170 and the grand average, click the **Create ERPset** button at the bottom of the panel, and name the new ERPset **Simulated N170**. You should also save the ERPset to the folder for this exercise, because you will need it later.

Screenshot 4.35



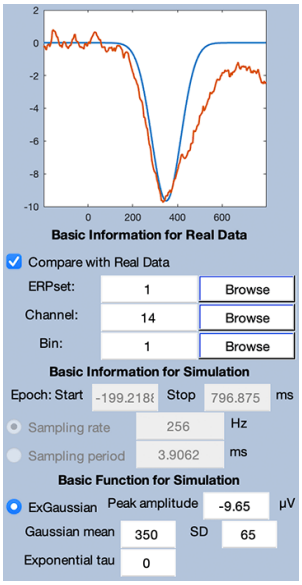
Now we're going to create a simulated N400 to match the grand average from the ERP CORE N400 experiment. Start by clearing the loaded ERPsets from the **ERPsets** panel and then using this panel to load the file named **N400.erp**. This ERPset has one bin corresponding to the N400 difference wave (unrelated targets minus related targets). Select the **CPz** channel (Channel 14) from the **Bin & Channel Selection** panel. You should see something like Screenshot 4.36, with a large N400 peaking at approximately 350 ms.

Screenshot 4.36



Now go to the **Create Artificial ERP Waveforms** panel, make sure the box labeled **Compare with Real Data** is checked, and set the **ERPset** box to **1**. We want to select the **CPz** channel (because this is where the N400 is largest), so click the **Browse** button next to the **Channel** box, click on the **CPz** channel, and click **OK**. You should now see **14** in the Channel box. There is only one bin, so you can leave the **Bin** box at **1**. Now go to the section of the panel labeled **Basic Function for Simulation** and change the **Peak amplitude** to **-9.65**, the **Gaussian mean** to **350**, and the SD to **65**. You should see something like Screenshot 4.37.

Screenshot 4.37

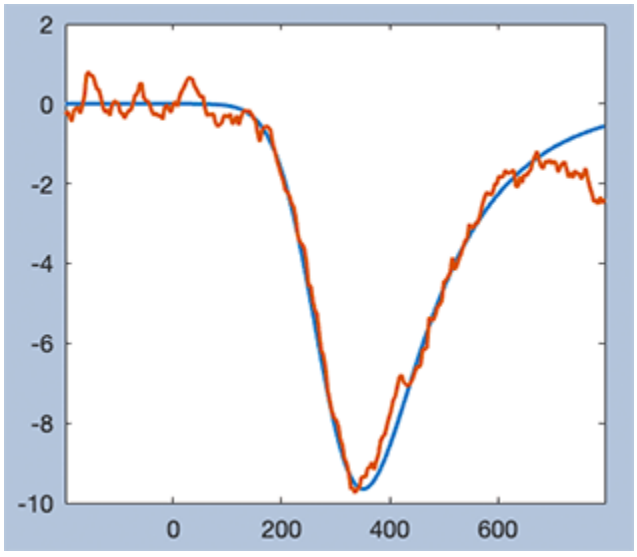


Although the artificial waveform is a pretty good match for the part of the waveform prior to the peak, it's a poor match for the part of the waveform after the peak. This is because the artificial waveform is a mirror-symmetrical Gaussian function, whereas the N400 difference wave has a rightward skew. This kind of skew is found in many higher-level ERP components (for an explanation, see Chapter 9 in Luck, 2014).

To better fit skewed waveforms, ERPLAB uses an *ExGaussian* function to create artificial ERP waveforms. This function is a convolution of an exponential function (which can have different amounts of skew) and the symmetrical Gaussian function. The skewness of the exponential function is determined by the *tau* (time constant) parameter. Up until this point, we've used a value of zero for tau, resulting in a pure Gaussian function. But now we need to use a nonzero tau value to simulate the skewed N400 difference wave.

To do this, go to the section of the panel labeled **Basic Function for Simulation** and enter a value of **1400** in the **Exponential tau** box. This gives us approximately the right amount of skewness, but now the latency of the artificial waveform has shifted rightward. To account for this, change the **Gaussian mean** to **280** ms. You should now see a pretty good match, as shown in Screenshot 4.38. To create an ERPset with this simulated N400, click the **Create ERPset** button at the bottom of the panel, name the new ERPset **Simulated N400**, and save it to disk.

Screenshot 4.38



You've now seen the process of creating artificial waveforms that simulate two different ERP components. In the next exercise,

you'll apply several different filters to these waveforms and quantify the amount of waveform distortion. But I suspect that you have some questions about the process of creating the artificial waveforms, which I will try to answer here.

First, you may be wondering what to do if your simulated waveform doesn't perfectly match the real data. For example, our simulated N170 waveform doesn't include the positivities that follow the N170 in the grand average, and our simulated N400 waveform doesn't match the very end of the grand average waveform.

Our goal here is not perfection. We are just trying to figure out a reasonable set of filtering settings that will do a good job of reducing noise without producing too much waveform distortion. The criterion for "too much waveform distortion" is somewhat arbitrary, so perfection is not even well defined. You should also keep in mind that slightly different filter settings will produce only slightly different results, so you don't need to get the settings exactly right. Your goal is to determine the filter settings that are approximately optimal. So don't worry if the simulated effect doesn't perfectly match the grand average waveform.

A second issue arises when the grand average waveform is quite noisy, so you don't know what the true effect actually looks like. This wasn't a problem for the ERP CORE experiments, because we had plenty of trials and plenty of participants, and we had hundreds of prior studies to compare our waveforms with. But it's often a problem if you're embarking on a new program of research, and you have only one or two somewhat noisy previous studies to rely on. If you're in this situation, I'd recommend creating multiple simulated waveforms, each reflecting a different plausible estimate of the true effect. You can then choose filter settings that won't produce unacceptable levels of distortion with any of them.

A third question is what to do when the grand average waveform you are using to estimate the real effect contains multiple components that are each reasonably large. For example, in many oddball paradigms, the P3b is isolated with a rare-minus-frequent difference wave, but there may also be an N2 preceding the P3b in this difference wave. Or you may be in a situation where you can't even make a difference wave. In these situations, I recommend making multiple artificial waveforms, one for each component. You can then sum these together to make sure that the sum does a good job of matching the combined components in the grand average waveform. Then, you can examine the waveform distortions produced for each component individually and for their sum. This is what we did in Section 4.8 for the combination of the P1, N1, and P2 components. The box below provides concrete information about how to combine the components using eStudio.

Combining simulated ERPsets

Once you have created the individual simulated components in separate ERPsets using the **Create Artificial ERP Waveforms** panel, you can combine them together into a single ERPset using the **Append ERPsets** panel. To do this, first select the ERPsets for the simulated components in the **ERPsets** panel. Then go to the **Append ERPsets** panel and tell it to append the selected ERPsets. This will create a new ERPset in which each simulated component is in a separate bin. You can then use the **Bin Operations** panel and write an equation that creates a new bin by summing together the bins for all of the simulated components. For example, when I did this with the simulated P1, N1, and P2 components, the individual components were in bins 1-3, and I used this equation in the **Bin Operations** panel to combine them together: **b4 = b1 + b2 + b3 Label P1+N1+P2**

This page titled 4.13: Exercise- Creating artificial waveforms is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck.

4.14: Exercise- Quantifying Waveform Distortion in the Artificial Waveforms

This exercise will focus on the third step of the process that was outlined at the end of Section 4.12, namely quantifying the waveform distortion that occurs when we apply a set of filters to the simulated N170 and N400 waveforms you created in the preceding exercise.

The left portion of Figure 4.11 illustrates this part of the overall process. Once we have the simulated ERP waveform, we loop through a set of candidate filters. For each filter, we apply the filter to the simulated waveform and compute the artifactual peak percentage (APP) to quantify the waveform distortion for that filter. The right portion of Figure 4.11 illustrates the resulting APP values across all the candidate filters. We compare the APP values with our threshold, and the ones that are below the threshold are considered to produce "acceptable" levels of waveform distortion. These are the ones that we will further examine to quantify how they impact the data quality (signal and noise). The ones above the threshold are ruled out. In this example, we will use an APP threshold of 5% for the reasons described in Chapter 4.12.

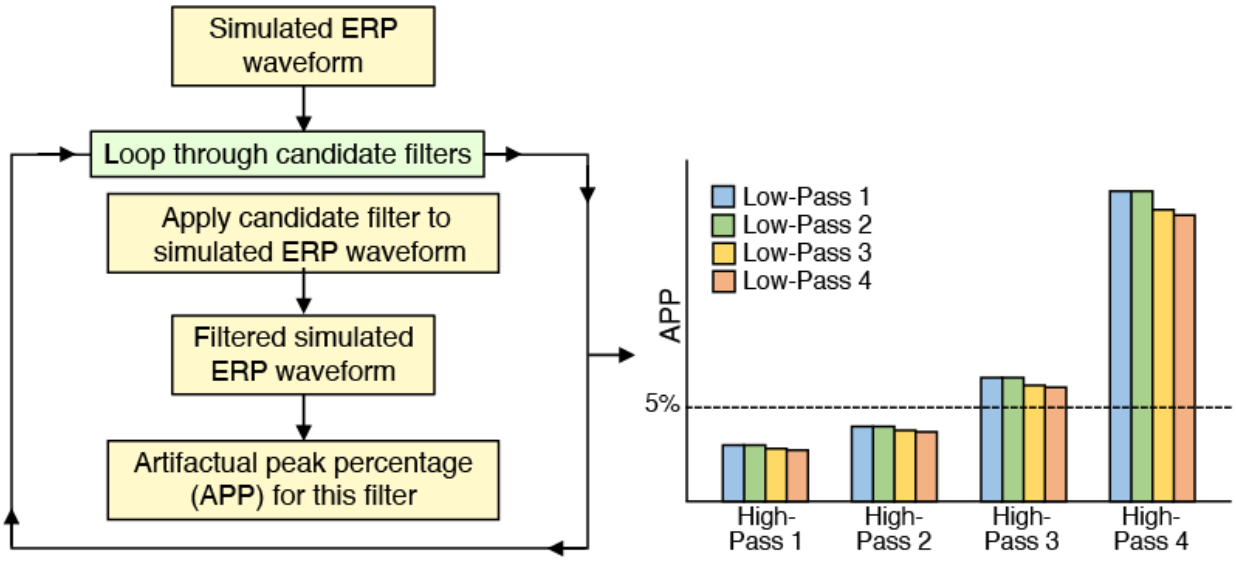


Figure 4.11. Process of assessing waveform distortion (artifactual peak percentage, APP) for a set of candidate filters.

How do you know what set of filters to try? For most cognitive and affective research, I recommend starting with all combinations of the low-pass and high-pass cutoffs shown in Table 4.2. I recommend using a roll-off of 12 dB/octave; we examined steeper roll-offs, and we never found them to be useful for basic filtering (although that might not be true with all types of studies, so you could explore different roll-offs as well as different combinations of low-pass and high-pass cutoffs).

Table 4.2. Initial set of filter cutoffs to assess for most cognitive and affective studies. All 5x5 combinations of these cutoffs should be assessed.

Table 4.2. Initial set of filter cutoffs to assess for most cognitive and affective studies. All 5x5 combinations of these cutoffs should be assessed.

High-pass cutoff	Low-pass cutoff
0.1 Hz	40 Hz

High-pass cutoff	Low-pass cutoff
0.5 Hz	30 Hz
1 Hz	20 Hz
2 Hz	10 Hz

If you are looking at short-latency (<100 ms) sensory components, you'll probably want to shift both the low-pass and high-pass cutoffs to higher values. If you are looking at very slow cognitive and affective components (lasting 500 ms or longer), you'll probably want to shift the high-pass cutoffs lower but keep the low-pass cutoffs the same.

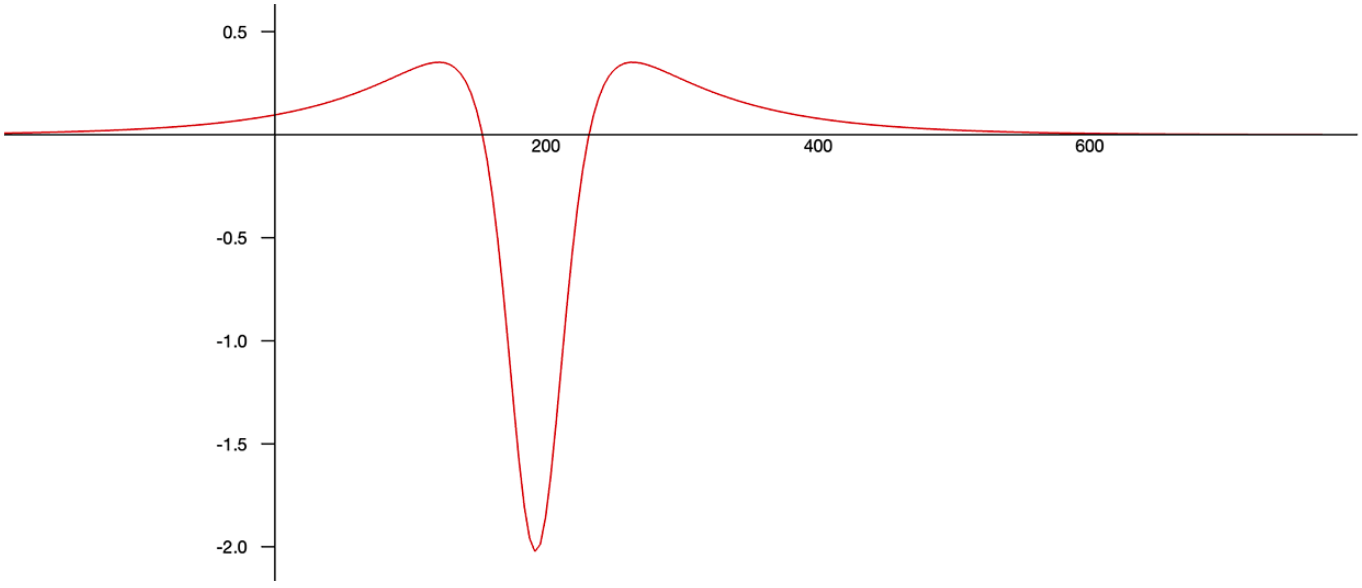
Once you try these, it should be clear whether you've bracketed the range of good candidates (i.e., with some that have too much waveform distortion, and others that don't improve the signal quality enough). And once you have the right range, you can look at a finer set of gradations within that range (as will be discussed below).

In the present exercise, we will see how ERPLAB Studio can be used to compute the APP values for this set of candidate filters. Start by clicking the **Reset** button under the lower right corner of the plot region and then clicking **Reset ERPLAB Studio** in the window that appears. Then load the ERPset named **Simulated N170.erp** that you created in the preceding exercise (a copy is also present in the **Extra Files** folder within the folder for the Chapter 14 exercises).

Let's see what happens with the most extreme filter in our set of candidate filters, with a high-pass cutoff at 2 Hz and a low-pass cutoff at 10 Hz. This filter passes the narrowest range of frequencies, so it's likely to produce the greatest waveform distortion.

Go to the **Filtering** panel and select a high-pass cutoff at **2** Hz and a low-pass cutoff at **10** Hz, with a roll-off of **12** dB/octave. Then click **Run** and name the resulting ERPset **Simulated N170_filt2-10**. It should look like Screenshot 4.39, with artifactual positive peaks surrounding the negative N170 peak.

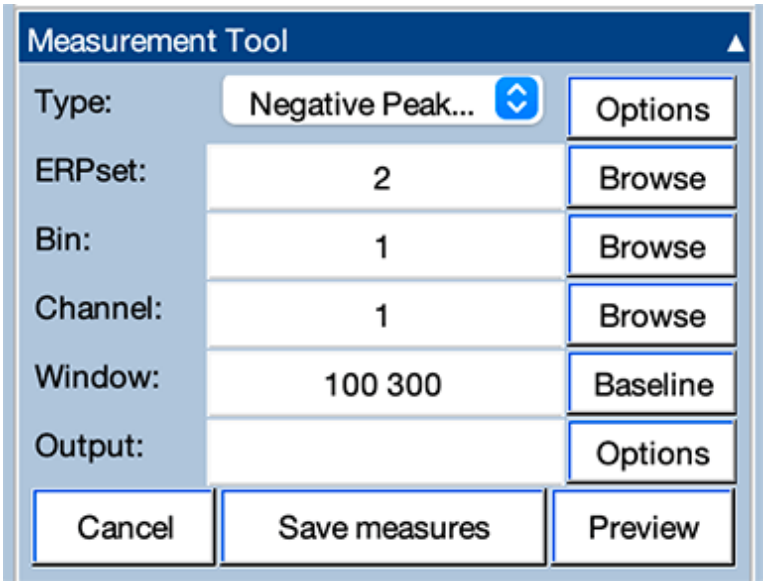
Screenshot 39



The next step is to score the amplitudes of the artifactual peaks along with the true N170 peak. Go to the **Measurement Tool** panel and set it up to score the peak amplitude of the N170, as shown in Screenshot 4.40. The measurement **Type** should be **Negative Peak Amplitude**. The filtered waveform should be ERPset #2, so you can put a **2** in the **ERPset** field. The ERPset

has only 1 bin and 1 channel, so you should put a **1** into the **Bin** and **Channel** boxes. A reasonable window for the N170 peak is 100-300 ms, so put **100 300** into the **Window** box. You can leave the **Output** box empty, because we're only getting one value and we can do that easily using the Measurement Tool's preview function.

Screenshot 4.40



To get the N170 peak amplitude score, click the **Preview** button. You can see that the peak amplitude is -2.059 µV. Click **Close** to close the preview window and go back to the **Measurement Too**l panel.

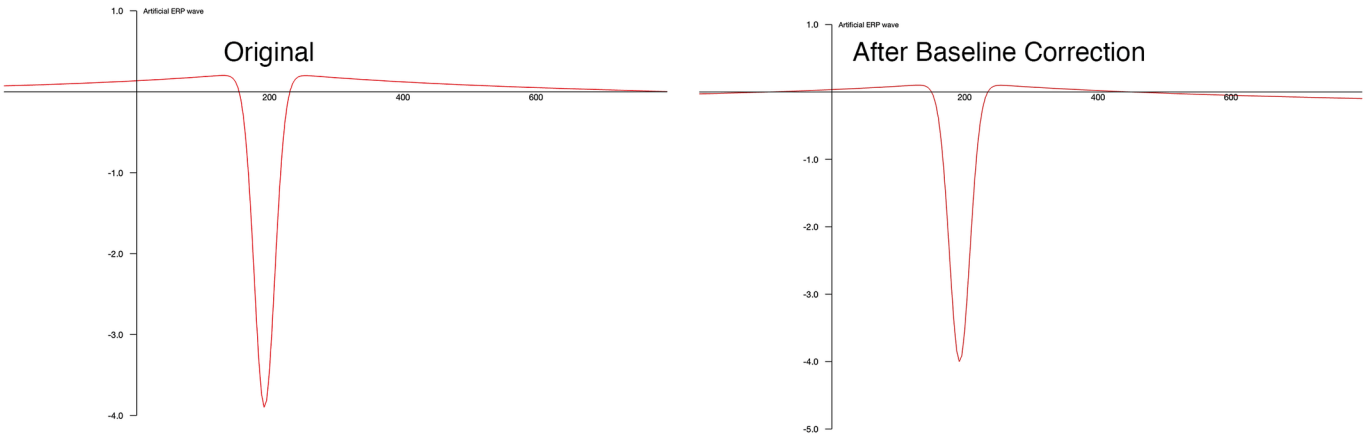
Now score the amplitude of the first artifactual positive peak by changing the **Type** to **Positive Peak Amplitude** and changing the **Window** to **100 200**. If you click the **Preview** button, you can see that the artifactual positive peak has an amplitude of +0.316 µV. Click **Close** to close this window. The first and second positive peaks are nearly identical, so we only need to score the first one. If you have multiple artifactual peaks, but they have different amplitudes, use the one with the greatest amplitude.

The artifactual peak percentage is simply 100 times the absolute value of the artifactual positive peak's amplitude divided by the absolute value of the N170's peak amplitude: APP = 100 * 0.316 / 2.059 = 15.3%. This is clearly above our threshold of 5%, so this filter produces too much waveform distortion to be considered further.

Now repeat this process with cutoffs of 0.5 Hz and 30 Hz. That is, filter the original **Simulated N170** ERPset with these cutoffs and then measure the amplitude of the true N170 peak amplitude and the artifactual positive peak preceding the N170. You should find values of -4.002 µV and +0.100 µV, respectively, which gives us APP = 100 * 0.003 / 4.324 = 2.5%. These filter settings produced only a little bit of waveform distortion, and the APP was under the 5% threshold.

The amplitude of the artifactual positive peak, +0.100 µV, may not seem to match what you saw when you first looked at the waveform (Screenshot 4.41, left). In that waveform, the peak amplitude appears to be about 20% of the way between the baseline and the 1.0 µV marker, which should be an amplitude of about 0.2 µV. This discrepancy is a result of ERPLAB's policies about baseline correction. Filtering the N170 wave created an overall positive offset during the prestimulus period, which you can see when you look at the main eStudio plotting window (and Screenshot 4.41, left). However, the default behavior for the **Measurement Tool** is to re-baseline the data using the average voltage in the prestimulus period prior to measuring anything. If you look at the waveform in Preview, you will see that it has been re-baselined. This is usually the desired behavior, but it can be overridden by clicking the **Baseline** button next to the measurement window in the **Measurement Tool** panel.

Screenshot 4.41



If you want to manually re-baseline the data prior to running the Measurement Tool, you can use the **Baseline Correction & Linear Detrend** panel to perform a baseline correction on the ERPset (see the result of this in Screenshot 4.41, right).

The results so far indicate that a bandpass of 0.5 – 30 Hz is acceptable for the N170 component, whereas a bandpass of 2 – 10 Hz leads to an unacceptable artifactual peak percentage. It's important to realize, however, that the amount of waveform distortion depends on the shape of the component being filtered, not just the filter settings. To see this for yourself, repeat these steps for these two filters with the **Simulated N400** ERPset (which you can find in the **Extra Files** folder). I recommend using a measurement window of **100 300** for the artifactual positive peak and **200 500** for the true N400 peak.

You should find an artifactual positive percentage of 9.76% (100 x 0.737 ÷ 7.555) for the 0.5 – 30 Hz bandpass and 44.2% (100 x 1.173 ÷ 2.653) for the 2 – 10 Hz bandpass. Both of these are well above the 5% threshold, so neither would be an acceptable filter for the N400. This shows why it is important to select filter cutoffs that are appropriate for the specific ERP effects in a given study.

It would take quite a bit of time and effort to repeat this process for all 25 of the cutoff combinations listed in Table 4.2. It could be worthwhile, especially if you're going to run many similar studies and can use the same filter settings for all of them. But the easiest solution in most cases is to write a script that will automatically compute the APP values for every candidate filter. I've created a script named **Script1_APP.m** for this purpose. You can find it in the folder for this chapter. It creates the simulated N170 and N400 waveforms, filters them, uses the Measurement Tool to get the peak amplitudes of the true and artifactual peaks, and computes the APP. Here's what the output looks like:

bandpass 0.10 to 10.00 Hz: N170 APP = 0.77%, N400 APP = 0.04%
bandpass 0.10 to 20.00 Hz: N170 APP = 0.07%, N400 APP = 0.18%
bandpass 0.10 to 30.00 Hz: N170 APP = 0.08%, N400 APP = 0.21%
bandpass 0.10 to 40.00 Hz: N170 APP = 0.08%, N400 APP = 0.22%
bandpass 0.50 to 10.00 Hz: N170 APP = 2.28%, N400 APP = 9.49%
bandpass 0.50 to 20.00 Hz: N170 APP = 2.56%, N400 APP = 9.72%
bandpass 0.50 to 30.00 Hz: N170 APP = 2.51%, N400 APP = 9.76%
bandpass 0.50 to 40.00 Hz: N170 APP = 2.47%, N400 APP = 9.77%
bandpass 1.00 to 10.00 Hz: N170 APP = 7.24%, N400 APP = 24.84%
bandpass 1.00 to 20.00 Hz: N170 APP = 7.14%, N400 APP = 25.28%
bandpass 1.00 to 30.00 Hz: N170 APP = 6.96%, N400 APP = 25.35%
bandpass 1.00 to 40.00 Hz: N170 APP = 6.83%, N400 APP = 25.37%
bandpass 2.00 to 10.00 Hz: N170 APP = 15.36%, N400 APP = 44.23%
bandpass 2.00 to 20.00 Hz: N170 APP = 15.12%, N400 APP = 44.76%
bandpass 2.00 to 30.00 Hz: N170 APP = 14.79%, N400 APP = 44.85%
bandpass 2.00 to 40.00 Hz: N170 APP = 14.53%, N400 APP = 44.87%

This script would be fairly easy to modify for your own research. You would just update the parameters that control the size, latency, and shape of the components, the filter cutoffs to test, and the time windows used to measure the true and artifactual components.

As you can see from the script's output, most of these filters yield unacceptably high levels of waveform distortion (APP values of >5%). Anything with a high-pass cutoff of >0.5 Hz is unacceptable for the N170, and anything with a high-pass cutoff of > 0.1 Hz is unacceptable for N400. A reasonable next step would be to look at high-pass cutoffs between 0.1 and 1 Hz with a finer granularity (e.g., every 0.1 Hz) to get a more precise idea of the point at which the waveform distortion becomes unacceptable. However, we won't do that here.

The final step, which will be demonstrated in the next section, is to determine how well the various acceptable filters minimize noise. The optimal filter will then be the one that minimizes noise the best while still having an APP of <5%.

This page titled 4.14: Exercise- Quantifying Waveform Distortion in the Artificial Waveforms is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck.

- 4.6: Exercise- Applying the Impulse Response Function to a Series of Impulses by Steven J Luck is licensed CC BY 4.0.

4.15: Exercise- Quantifying Data Quality

As I noted earlier, the optimal filter is defined as the one that maximizes data quality without producing unacceptable levels of waveform distortion. We saw how to quantify waveform distortion in the previous exercise, so now we'll look at how to quantify data quality. For amplitude measures, we need to take into account the fact that filters typically reduce the amplitude of the component we're trying to measure (see Section 4.12). Consequently, data quality for amplitude measures is defined as the signal-to-noise ratio (SNR), where the signal is quantified as the amplitude score for the component we are measuring, and the noise is defined as the standardized measurement error (SME) for that score. Filters don't suppress latencies the way they suppress amplitudes, so for latency scores we simply look at the SME for the score.

Whereas we could use simulated data for quantifying waveform distortion, we need to use real data to get the scores and SME values that are used to compute the SNR. This is because the nature of the signal and the nature of the noise impact the usefulness of a filter, so we need to have real data to know what scores and SME values to use.

In this section, we'll use the ERP CORE N400 experiment as our example. To keep things simple, we're going to use the same subset of 10 participants that we used in Chapter 3.

As you saw in Chapter 2, each trial contains a prime stimulus followed by a semantically related or unrelated target stimulus. There are four bins in each ERPset: 1) Prime followed by related target; 2) Prime followed by unrelated target; 3) Target preceded by related prime; 4) Target preceded by unrelated prime. The score of interest in this study was the mean amplitude between 300 and 500 ms in the CPz channel, measured from a difference wave created by subtracting Bin 4 minus Bin 3 (unrelated minus related targets). We will measure this score from the grand average unrelated-minus-related difference wave as our estimate of signal. The noise will be the standardized measurement error (SME) for this score (combined across participants in a special way that I will describe below).

Overview of Computing the SNR

Figure 4.12 provides an overview of the whole process. For a given candidate filter, you loop through all 10 participants to get an ERPset and an SME value from each participant for the filtered data. You then make a grand average of the single-subject ERPsets, make the unrelated-minus-related difference wave from the grand average, and compute the mean amplitude from 300-500 ms from this difference wave. That gives you the score.

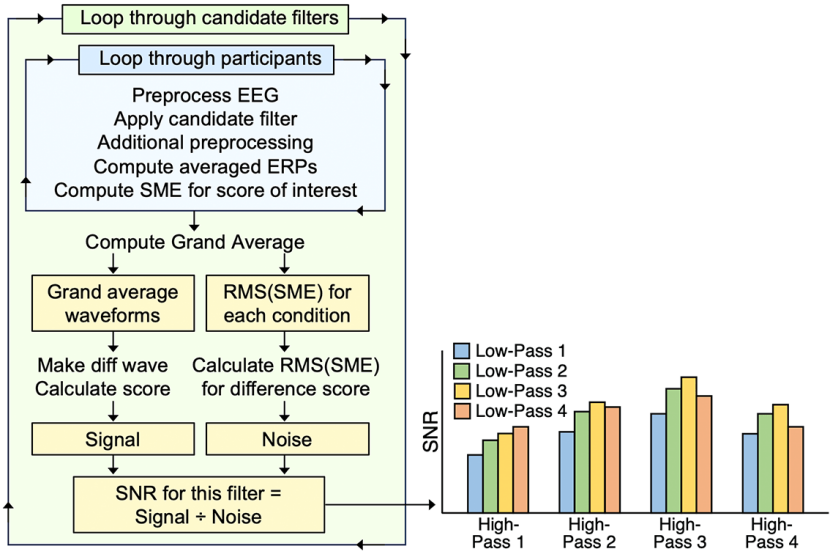


Figure 4.12. Flow chart describing the process of quantifying signal, noise, and signal-to-noise ratio (SNR) for a set of candidate filters.

For each subject, we also want to get the SME value corresponding to the mean amplitude from the difference wave. When we make the averaged ERPs for a given subject, we can tell the averaging routine that we want it to compute the SME for the mean

amplitude from 300-500 ms for each of the 4 bins. When we make the grand average across subjects, the routine that makes the grand average waveforms also combines the SME values across the subjects. Instead of taking the mean of the SME values, however, it combines them using the root mean square (RMS), which is more appropriate mathematically (see Luck et al., 2022, for the reasoning).

The resulting RMS(SME) values quantify the data quality for the N400 time period in each of the four bins. However, we need the RMS(SME) for the N400 amplitude score as measured from the unrelated-minus-related difference wave. Fortunately, there is a little trick we can use to compute this from the RMS(SME) values for the N400 amplitude scores measured separately from the original related and unrelated waveforms (explained below). This gives us our estimate of the noise. We then compute the signal-to-noise ratio by taking the N400 amplitude score from the grand average difference wave (the signal) and dividing it by the RMS(SME) of the unrelated-minus-related N400 difference score (the noise).

This process is repeated for each candidate filter, giving us an SNR value for each filter (as shown in the bar graph at the right of Figure 4.12). We can then combine this SNR information with the information about waveform distortion from the preceding exercise and choose the filter that gives us the best SNR without exceeding our threshold for waveform distortion.

That's pretty complicated, so let's see how it works with actual data. We'll focus on a single filter (0.1–10 Hz) for now, but I've created a script you can use to repeat the process for all the candidate filters.

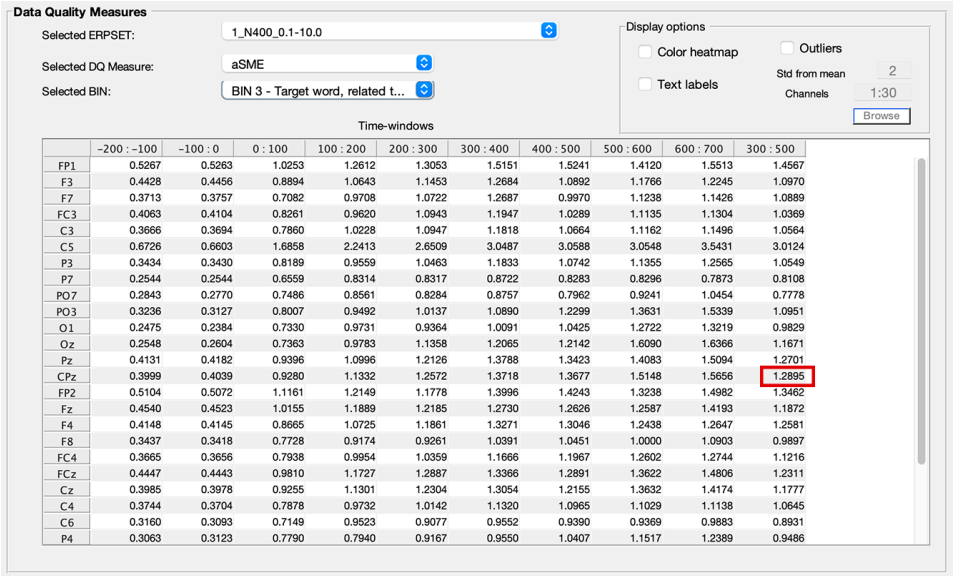
First Steps

To get started, reset everything with the **Reset** button, then go to the **ERP** tab and click **Load ERP** from the **ERPsets** panel. Navigate to the **N400_Data** folder inside the exercises folder for this chapter and load the 10 ERPsets that are in that folder. You can load all 10 in one operation by shift-clicking or command-clicking to select multiple files.

I've already done most of the steps in the blue box in Figure 4.12: I preprocessed the EEG up to the point of assigning events to bins, applied the 0.1–10 Hz filter, did some additional preprocessing (i.e., artifact detection), and computed the averaged ERP waveforms for the 4 bins. When I ran the averaging routine, I had it compute the SME for the mean voltage between 300 and 500 ms (along with the default time ranges). You can see how this works in the script named **Script0_Make_ERPs.m**.

To see the SME values, select the ERPset for subject 1 (**1_N400_0.1-10.0**) in the **ERPsets** panel. Note that I included the filter settings (0.1 to 10 Hz) in the ERPset name. Now go to the **View Data Quality Metrics** panel and click **Show in a table**. You will see the SME values for several different time windows at each channel. At the top, you will see that Bin 1 is selected. Change this to Bin 3 (target preceded by related prime). You should see something like Screenshot 4.42.

Screenshot 4.42



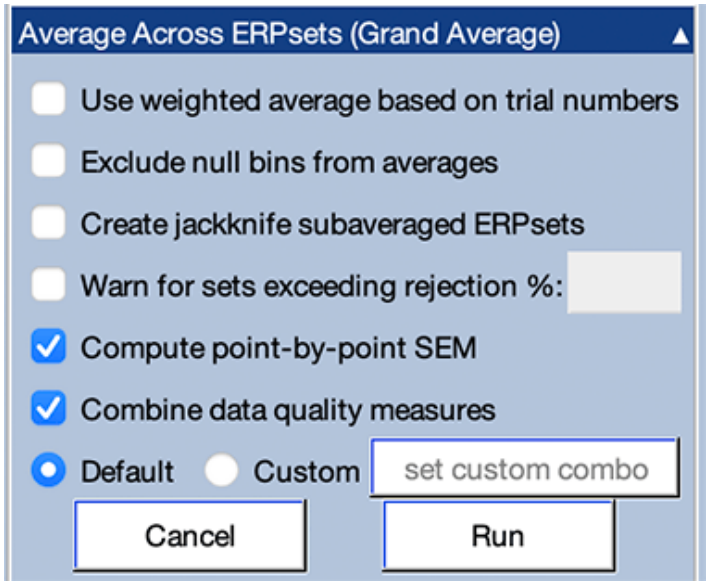
	-200 : -100	-100 : 0	0 : 100	100 : 200	200 : 300	300 : 400	400 : 500	500 : 600	600 : 700	300 : 500
FP1	0.5267	0.5263	1.0253	1.2612	1.3053	1.5151	1.5241	1.4120	1.5513	1.4567
F3	0.4428	0.4456	0.8894	1.0643	1.1453	1.2684	1.0892	1.1766	1.2245	1.0970
F7	0.3713	0.3757	0.7082	0.9708	1.0722	1.2687	0.9970	1.1238	1.1426	1.0889
FC3	0.4063	0.4104	0.8261	0.9620	1.0943	1.1947	1.0289	1.1135	1.1304	1.0369
C3	0.3666	0.3694	0.7860	1.0228	1.0947	1.1818	1.0664	1.1162	1.1496	1.0564
C5	0.6726	0.6603	1.6858	2.2413	2.6509	3.0487	3.0588	3.0548	3.5431	3.0124
P3	0.3434	0.3430	0.8189	0.9559	1.0463	1.1833	1.0742	1.1355	1.2565	1.0549
P7	0.2544	0.2544	0.6559	0.8314	0.8317	0.8722	0.8283	0.8296	0.7873	0.8108
PO7	0.2843	0.2770	0.7486	0.8561	0.8284	0.8757	0.7962	0.9241	1.0454	0.7778
PO3	0.3236	0.3127	0.8007	0.9492	1.0137	1.0890	1.2299	1.3631	1.5339	1.0951
O1	0.2475	0.2384	0.7330	0.9731	0.9364	1.0091	1.0425	1.2722	1.3219	0.9829
Oz	0.2548	0.2604	0.7363	0.9783	1.1358	1.2065	1.2142	1.6090	1.6366	1.1671
Pz	0.4131	0.4182	0.9396	1.0996	1.2126	1.3788	1.3423	1.4083	1.5094	1.2701
CPz	0.3999	0.4039	0.9280	1.1332	1.2572	1.3718	1.3677	1.5148	1.5656	1.2895
FP2	0.5104	0.5072	1.1161	1.2149	1.1778	1.3996	1.4243	1.3238	1.4982	1.3462
Fz	0.4540	0.4523	1.0155	1.1889	1.2185	1.2730	1.2626	1.2587	1.4193	1.1872
F4	0.4148	0.4145	0.8665	1.0725	1.1861	1.3271	1.3046	1.2438	1.2647	1.2581
F8	0.3437	0.3418	0.7728	0.9174	0.9261	1.0391	1.0451	1.0000	1.0903	0.9897
FC4	0.3665	0.3656	0.7938	0.9954	1.0359	1.1666	1.1967	1.2602	1.2744	1.1216
FCz	0.4447	0.4443	0.9810	1.1727	1.2887	1.3366	1.2891	1.3622	1.4806	1.2311
Cz	0.3985	0.3978	0.9255	1.1301	1.2304	1.3054	1.2155	1.3632	1.4174	1.1777
C4	0.3744	0.3704	0.7878	0.9732	1.0142	1.1320	1.0965	1.1029	1.1138	1.0645
C6	0.3160	0.3093	0.7149	0.9523	0.9077	0.9552	0.9390	0.9369	0.9883	0.8931
P4	0.3063	0.3123	0.7790	0.7940	0.9167	0.9550	1.0407	1.1517	1.2389	0.9486

Because we are scoring the N400 as the mean voltage between 300 and 500 ms at the CPz electrode site, the only relevant value here is for 300-500 ms (the rightmost column) at CPz (highlighted with a red box in the screenshot). This SME value is

1.2895. If you switch to Bin 4, you'll see that the corresponding SME value is 1.0920. You can now close the Data Quality window.

Now let's make a grand average of the 10 ERPsets. Select all 10 ERPsets in the **ERPsets** panel, and then go to the **Average Across ERPsets (Grand Average)** panel. Set it up as shown in Screenshot 4.43. Note that we're telling it to combine the data quality measures across the 10 ERPsets using the defaults (which will cause it to combine the SME values using the RMS operation). Click **Run** and name the ERPset **grand_0.1-10.0**.

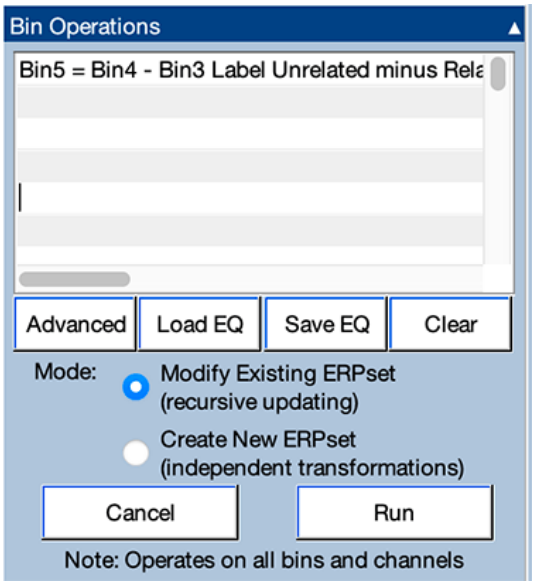
Screenshot 4.43



Computing the Signal

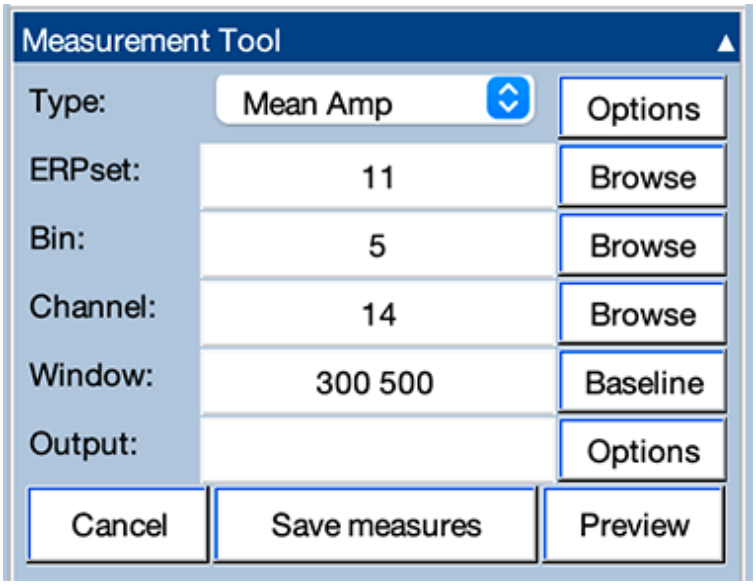
The next step will be to compute the signal, which is the N400 amplitude score from the unrelated-minus-related difference wave for the target. To create the difference wave, go to the **Bin Operations** panel and type the following into the top line of the text box: **Bin5 = Bin4 - Bin3 Label Unrelated minus Related Targets**. That equation will create a new Bin 5 with the difference wave. Make sure that the panel is set up as shown in Screenshot 4.44 and that the grand average is selected in the **ERPsets** panel, and then click **Run**. The new bin will be inserted into the grand average ERPset. You should be able to see it in the plotted waveforms.

Screenshot 4.44



The next step is to get the mean amplitude at CPz between 300 and 500 ms from this difference wave. You'll do this by setting up the **Measurement Tool** as shown in Screenshot 4.45. Click **Preview** to see the mean amplitude score, which is -7.846 μV. This is the signal portion of our signal-to-noise ratio.

Screenshot 4.45



Computing the Noise and the SNR

Now we need to get the noise value, which is the RMS(SME) for the mean amplitude score we just computed. Make sure that the grand average is still selected in the **ERPsets** panel, then go to the **View Data Quality Metrics** panel and click **Show in a table**. If you go to the **Selected BIN** menu near the top of the window, you will see that Bins 1-4 are available but Bin 5 (the difference wave) is not listed. This is because we used Bin Operations to compute the difference wave from Bins 3 and 4, and Bin Operations doesn't know how the SME for this difference wave is related to the SME values from Bins 3 and 4. Bin Operations can recombine bins in a virtually infinite number of ways, and there is no way of knowing what the appropriate SME value would be for most of those recombinations. Consequently, it doesn't try to provide SME values for any new bins that it creates.

Fortunately for us, there is a simple way to compute the SME for a simple difference wave by using the SME values of the two parent waves (see Zhang et al., 2024b). The formula for the SME for a difference between conditions A and B is simply: $SME_{A-B} = sqrt(SME_A^2 + SME_B^2)$. So, we can compute the SME for the unrelated-minus-related difference wave from the SME values from the SME values from Bins 3 and 4. In the data quality window, select Bin 3 (related targets) and find the SME value for the 300-500 ms time window at the CPz electrode site. You should see a value of 1.5191. Now select Bin 4 (unrelated targets) and find the corresponding SME value, which should be 1.3216. The SME for the unrelated-minus-related difference wave for this time window and electrode site is sqrt(1.5191 + 1.32162) = 1.685. This is the noise value for computing the SNR.

We now have the signal (-7.486) and the noise (2.01), and we can just divide the absolute value of the signal by the noise to get the SNR: |-7.486| ÷ 1.685 = 4.44.

A Little Warning

The trick we just used to compute the SME for a difference wave is valid only when the two bins are statistically independent of each other for each individual participant. This will generally be true if different trials are used for the two bins (like the related and unrelated target trials in the present example). It will definitely not be true if you are making a difference wave from different channels of the same bin (as in a contralateral-minus-ipsilateral difference wave for N2pc, LRP, or CDA). In such cases, you can use bootstrapping to estimate the SME (see Luck et al., 2022).

Repeating the Process for Each Candidate Filter

We need to repeat this process multiple times to get the SNR for each of our candidate filters. This would take quite a bit of time to do using the eStudio GUI. For each subject, you would need to go back to the continuous EEG, apply one of the candidate filters, run artifact detection, and then make an average. You would then go through all of the steps you just completed in this exercise.

It's much more realistic to do this using a script. In the folder for this chapter, you will see a script for this named Script2_Signal_and_Noise.m. Go ahead and run the script by loading it into the Matlab editor and then clicking the big green triangle labeled Run. The script will take a couple minutes to run, and you can monitor its progress in the Matlab Command window. You will then see the following table of values in the Command window:

Bandpass=0.1-10.0: Signal=-7.85, Noise=2.01, SNR=3.90
Bandpass=0.1-20.0: Signal=-7.93, Noise=2.02, SNR=3.92
Bandpass=0.1-30.0: Signal=-7.94, Noise=2.03, SNR=3.91
Bandpass=0.1-40.0: Signal=-7.96, Noise=2.03, SNR=3.91
Bandpass=0.5-10.0: Signal=-5.61, Noise=1.40, SNR=3.99
Bandpass=0.5-20.0: Signal=-5.69, Noise=1.43, SNR=3.99
Bandpass=0.5-30.0: Signal=-5.71, Noise=1.43, SNR=3.98
Bandpass=0.5-40.0: Signal=-5.72, Noise=1.44, SNR=3.98
Bandpass=1.0-10.0: Signal=-3.69, Noise=0.92, SNR=3.99
Bandpass=1.0-20.0: Signal=-3.77, Noise=0.96, SNR=3.92
Bandpass=1.0-30.0: Signal=-3.76, Noise=0.97, SNR=3.88
Bandpass=1.0-40.0: Signal=-3.79, Noise=0.97, SNR=3.90
Bandpass=2.0-10.0: Signal=-1.78, Noise=0.46, SNR=3.85
Bandpass=2.0-20.0: Signal=-1.80, Noise=0.50, SNR=3.59
Bandpass=2.0-30.0: Signal=-1.80, Noise=0.51, SNR=3.50
Bandpass=2.0-40.0: Signal=-1.84, Noise=0.52, SNR=3.54

The next section will demonstrate the process of using this information to select the optimal filter.

This page titled 4.15: Exercise- Quantifying Data Quality is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck.

4.16: Selecting the Optimal Filter

At the end of the previous section, a script generated this table of signal, noise, and SNR values for applying several different filter combinations to the N400 experiment.

Bandpass=0.1-10.0: Signal=-7.85, Noise=2.01, SNR=3.90
Bandpass=0.1-20.0: Signal=-7.93, Noise=2.02, SNR=3.92
Bandpass=0.1-30.0: Signal=-7.94, Noise=2.03, SNR=3.91
Bandpass=0.1-40.0: Signal=-7.96, Noise=2.03, SNR=3.91
Bandpass=0.5-10.0: Signal=-5.61, Noise=1.40, SNR=3.99
Bandpass=0.5-20.0: Signal=-5.69, Noise=1.43, SNR=3.99
Bandpass=0.5-30.0: Signal=-5.71, Noise=1.43, SNR=3.98
Bandpass=0.5-40.0: Signal=-5.72, Noise=1.44, SNR=3.98
Bandpass=1.0-10.0: Signal=-3.69, Noise=0.92, SNR=3.99
Bandpass=1.0-20.0: Signal=-3.77, Noise=0.96, SNR=3.92
Bandpass=1.0-30.0: Signal=-3.76, Noise=0.97, SNR=3.88
Bandpass=1.0-40.0: Signal=-3.79, Noise=0.97, SNR=3.90
Bandpass=2.0-10.0: Signal=-1.78, Noise=0.46, SNR=3.85
Bandpass=2.0-20.0: Signal=-1.80, Noise=0.50, SNR=3.59
Bandpass=2.0-30.0: Signal=-1.80, Noise=0.51, SNR=3.50
Bandpass=2.0-40.0: Signal=-1.84, Noise=0.52, SNR=3.54

That's a lot of numbers, so I converted the values into a bar graph (Figure 4.13).

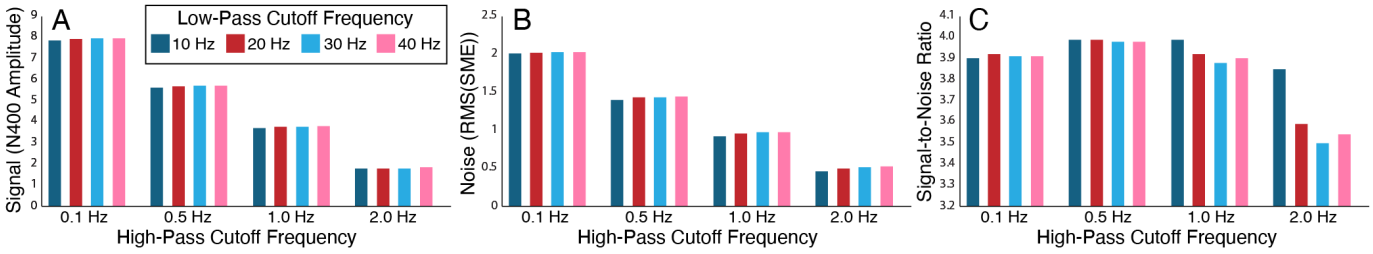


Figure 4.13. Signal (A), noise (B), and signal-to-noise ratio (SNR, C) for several combinations of low-pass and high-pass cutoffs applied to N400 mean amplitude.

As you can see, the cutoff frequency of the low-pass filter (ranging from 10-40 Hz) didn't have much impact on either the signal (panel A) or the noise (panel B). That's because we measured the mean amplitude over a 200-ms time window. The uppies and downies of high-frequency noise largely cancel out over this time window. Low-pass filtering has a much bigger impact on scores like peak amplitude and peak latency.

However, the high-pass cutoff had a large effect: As the high-pass cutoff increased, both the signal and the noise decreased. The signal decreased because the N400 contains a lot of low-frequency information, and filtering out more of this information (by increasing the high-pass cutoff) reduced the size of the N400. The noise decreased because there was a lot of low-frequency noise in the data, and this noise decreased more and more as the high-pass cutoff was increased.

The reduction in signal and the reduction in noise were approximately equal in most cases, so the SNR (panel C) did not change much as the high-pass cutoff increased. The exception was the 2 Hz cutoff, which reduced the signal more than it reduced the noise, causing the SNR to decrease. The best overall SNR came from a high-pass cutoff of 0.5 Hz (and the different low-pass cutoffs produced nearly identical SNR values with this high-pass cutoff).

From these results, you might think that a 0.5 high-pass cutoff would be ideal for the N400. However, we also need to take into account the waveform distortion produced by the filters. As we saw in Section 4.14, a high-pass cutoff of 0.5 Hz or higher led to unacceptably high levels of waveform distortion for the N400 (APP > 5%).

Consequently, we can really only consider the filters with a high-pass cutoff of 0.1 Hz. Fortunately, the SNR for a 0.1 Hz high-pass

filter was only slightly lower than the SNR for a 0.5 Hz high-pass filter. Thus, the 0.1 Hz cutoff gives us an SNR that is very high without producing any meaningful waveform distortion. It doesn't really matter what low-pass cutoff we use (as long as we are scoring mean amplitude).

In our paper on the optimal filter settings for the ERP CORE paradigms (Zhang et al., 2024b), we did a more fine-grained exploration of the different possible cutoff values, and we found that a high-pass cutoff of 0.2 Hz was optimal for the N400. It produced a slightly better SNR than the 0.1 Hz cutoff while still producing minimal waveform distortion. This 0.2 Hz cutoff was ideal whether we were scoring mean amplitude, peak amplitude, peak latency, or 50% area latency.

For mean amplitude, we found that the 0.2 Hz high-pass cutoff could be combined with any low-pass cutoff of 10 Hz or higher. For peak amplitude, we found that a low-pass cutoff at 10 Hz was helpful when scoring peak amplitude (especially with noisy data), and a low-pass cutoff at 5-10 Hz was helpful when scoring peak latency or 50% area latency.

You have now seen all the steps needed to determine the optimal filter settings for a given dataset. This means that you can now determine the optimal filters for your own data. The process takes a long time if you do it using the eStudio GUI, but it's pretty quick if you do it with a script. If you know a little about Matlab scripting, it should be straightforward for you to modify the scripts I have provided so they can be used with your own data.

Just imagine how impressed reviewers of your papers will be when they read your Methods section and find that you have rigorously determined the optimal filter settings for your data!

This page titled 4.16: Selecting the Optimal Filter is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck.

4.17: Matlab Scripts for this Chapter

I have provided three scripts for this chapter:

- **Script1_APP.m** demonstrates how to quantify the waveform distortion produced by filters (the artifactual peak percentage, APP) for Section 4.14
- **Script2_Signal_and_Noise.m** demonstrates how to quantify the effects of filtering on the signal, the noise, and the signal-to-noise ratio (SNR) for Section 4.15
- **Script0_Make_ERPs.m** was used to make the ERPs that are used for the GUI-based demonstration in Section 4.15

This page titled 4.17: Matlab Scripts for this Chapter is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 4.13: Matlab Script for this Chapter by Steven J Luck is licensed CC BY 4.0.

4.18: Key Takeaways and References

Key Takeaways

- You can filter EEG and ERPs in the frequency domain using the Fourier transform, or you can filter in the time domain using an impulse response function. These two approaches are mathematically identical and produce exactly the same result. However, we mainly care about the time domain in ERP research, so it's helpful to think about filtering as a time-domain operation.
- The impulse response function of a filter is just the output of the filter when the input is an impulse of amplitude 1 at time zero.
- You can think of an ERP waveform as a sequence of impulses, one at each time point. The output of a filter for a given input waveform can be computed by replacing each impulse in the input waveform with a copy of the impulse response function that has been scaled by the amplitude of the impulse and then summing them together.
- The amount of temporal "smearing" produced by a filter is easily understood by the width of the impulse response function. Heavy filtering can introduce artifactual peaks in your waveform, especially with high-pass filters or steep roll-offs, potentially causing you to draw completely bogus conclusions.
- The optimal filter is defined as the one that maximizes data quality without producing unacceptable levels of waveform distortion. There is no optimal filter for all studies: the optimal filter settings will depend on the shape of your experimental effect, the nature of the noise in your data, and how you are scoring the amplitude or latency of the ERP component.
- A paper by Zhang et al. (2024b) provides optimal filter values for the 7 ERP components in the ERP CORE. If you are conducting similar research, you can probably adapt those filter settings. However, if your research is quite different, you may need to determine the optimal filter settings for your data.
- Determining the optimal filter settings for your data takes some time and effort, but I have provided scripts that you can modify to make the process quicker.

References

Bae, G. Y., & Luck, S. J. (2018). Dissociable decoding of working memory and spatial attention from EEG oscillations and sustained potentials. *The Journal of Neuroscience*, *38*, 409–422. https://doi.org/10.1523/JNEUROSCI.2860-17.2017

Bigdely-Shamlo, N., Mullen, T., Kothe, C., Su, K.-M., & Robbins, K. A. (2015). The PREP pipeline: Standardized preprocessing for large-scale EEG analysis. *Frontiers in Neuroinformatics*, *9*. https://doi.org/10.3389/fninf.2015.00016

de Cheveigné, A. (2020). ZapLine: A simple and effective method to remove power line artifacts. *NeuroImage, 207,* 116356. https://doi.org/10.1016/j.neuroimage.2019.116356

Klug, M., & Kloosterman, N. A. (2022). Zapline-plus: A Zapline extension for automatic and adaptive removal of frequency-specific noise artifacts in M/EEG. *Human Brain Mapping, 43(9)*, 2743–2758. https://doi.org/10.1002/hbm.25832

Kappenman, E. S., & Luck, S. J. (2010). The effects of electrode impedance on data quality and statistical significance in ERP recordings. *Psychophysiology*, *47*, 888–904. https://doi.org/10.1111/j.1469-8986.2010.01009.x

Luck, S. J. (2005). *An Introduction to the Event-Related Potential Technique*. MIT Press.

Luck, S. J. (2014). *An Introduction to the Event-Related Potential Technique, Second Edition*. MIT Press.

Luck, S. J., Stewart, A. X., Simmons, A. M., & Rhemtulla, M. (2021). Standardized measurement error: A universal metric of data quality for averaged event-related potentials. Psychophysiology, 58, e13793. https://doi.org/10.1111/psyp.13793

Mitra, P. P., & Pesaran, B. (1999). Analysis of Dynamic Brain Imaging Data. *Biophysical Journal*, *76*(2), 691–708. https://doi.org/10.1016/S0006-3495(99)77236-X

Tanner, D., Morgan-Short, K., & Luck, S. J. (2015). How inappropriate high-pass filters can produce artifactual effects and incorrect conclusions in ERP studies of language and cognition. *Psychophysiology*, *52*, 997–1009. https://doi.org/10.1111/psyp.12437

Zhang, G., Garrett, D. R., & Luck, S. J. (2024a). Optimal filters for ERP research I: A general approach for selecting filter settings. Psychophysiology, 61, e14531. https://doi.org/10.1111/psyp.14531

Zhang, G., Garrett, D. R., & Luck, S. J. (2024b). Optimal filters for ERP research II: Recommended settings for seven common ERP components. Psychophysiology, 61, e14530. https://doi.org/10.1111/psyp.14530

This page titled 4.18: Key Takeaways and References is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 4.14: Key Takeaways and References by Steven J Luck is licensed CC BY 4.0.

5: REFERENCING AND OTHER CHANNEL OPERATIONS

Learning Objectives

In this chapter, you will learn to:

- Avoid the incorrect assumption that an ERP waveform primarily reflects signals from the active electrode site and instead recognize that the waveform equally reflects signals from the active and reference sites
- Compute the absolute voltage at the scalp electrodes given the internal generator voltage and the propagation weights between the generator and the scalp sites
- Compute the voltage between each electrode site and the ground electrode given the absolute voltages at each site
- Compute the referenced voltage at each site given the voltage between each site and the ground electrode
- Re-reference the voltage to another site
- Use simple algebra to determine the correct equations for re-referencing your data
- Use two alternatives to referencing your data, namely current density and global field power
- Implement several different reference schemes and alternatives to referencing in ERPLAB Studio with the Channel Operations panel
- Implement common referencing scenarios using a script

This chapter dives deeper into Channel Operations, especially with respect to how the reference electrode works and how you can re-reference your data. Once you know what you're doing, re-referencing the data will take you just a few seconds, and many researchers don't give it much thought. However, **the reference has an enormous impact on your ERP waveforms**, so you really need to understand what you're doing when you reference or re-reference your data.

The exercises in this chapter are designed to give you greater insight into what the reference site does, why we need to reference the data, and how re-referencing the data can clarify or obfuscate the results depending on what new reference you choose. We'll also discuss the mechanics of re-referencing in eStudio so that you can have confidence that you're doing it correctly.

We'll start with simulated data so that you can see how the original and referenced data on the scalp are related to the underlying neural generator. The simulations use Excel rather than eStudio, which makes it easier to see exactly what's going on. You can use Google Sheets instead of Excel, if you prefer. I'm assuming that you know the basics of spreadsheets, including how an equation in one cell can be used to compute a value on the basis of the values stored in other cells.

5.1: Data for This Chapter

5.2: Background- Understanding Active, Reference, and Ground Electrodes

5.3: Exercise- Working with the Artificial Data

5.4: Exercise- Average Mastoids as the Reference

5.5: Exercise- Re-Referencing the N400 ERP CORE Data

5.6: Exercise- The Average Reference

5.7: What is the Best Reference Site?

5.8: Exercise- Current Density

5.9: Exercise- Global Field Power

5.10: Exercise- Referencing the EEG Data from the ERP CORE N400 Experiment

5.11: Exercise- Other Common Re-Referencing Scenarios

5.12: Matlab Script For This Chapter

5.13: Key Takeaways and References

This page titled 5: Referencing and Other Channel Operations is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 5: Referencing and Other Channel Operations by Steven J Luck is licensed CC BY 4.0.

5.1: Data for This Chapter

The data we'll be using for the exercises in this chapter can be found in the Chapter_5 folder in the master folder: https://doi.org/10.18115/D5G045.

This page titled 5.1: Data for This Chapter is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 5.1: Data for This Chapter by Steven J Luck is licensed CC BY 4.0.

5.2: Background- Understanding Active, Reference, and Ground Electrodes

In this section, we'll review how active, reference, and ground electrodes work in EEG recordings. You can find a quick overview in this video and this video from the online Introduction to ERPs course and a detailed discussion in Chapter 5 of Luck (2014). Here, I'll explain the concepts using the artificial ERP waveforms shown in Figure 5.1. It helps to use artificial data in this context because we know what the true signals are. Later in the chapter, you'll apply what you've learned to real data.

In the artificial example shown in Figure 5.1A, the generator dipole is represented by the arrow, with the positive side pointing toward the Pz electrode. The broken line represents the transition between the positive and negative sides of the dipole, and the voltage is zero along this line. Unfortunately, we don't know the location of this zero line when we're looking at real data, and we don't have a single zero line when more than one dipole is active (which is almost always the case).

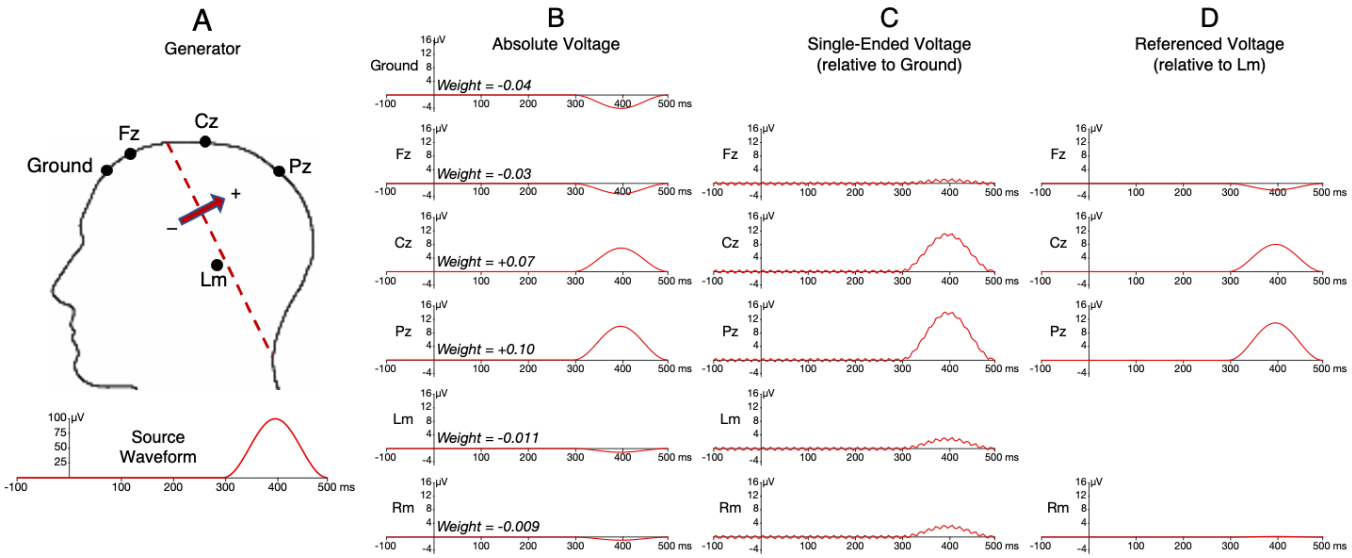


Figure 5.1. Example of active, ground, and reference electrodes. (A) Generation of the ERP. The arrow represents the generator dipole, and the source waveform shows the change in voltage over time at this dipole. The broken line represents the band of zero voltage at the transition between the positive and negative sides of the dipole. (B) Absolute voltage at each electrode site (which is known in this simulation but cannot be directly recorded). The absolute voltage at a given site is the source waveform multiplied by the weight for that site. (C) Single-ended voltage at each electrode site (i.e., the potential between a given site and the ground electrode). In most systems, this signal is present only inside the amplifier and is not available in the system's output. Note that a small amount of 60 Hz noise from the amplifier's ground circuit contaminates these signals. (D) Voltage at each site referenced to the left mastoid (Lm) electrode. This signal is obtained by subtracting (either in hardware or in software) the single-ended Lm signal from the signal at each of the other electrodes.

Voltage is the potential (pressure) for charges to move from one place to another place, so there is no such thing as the voltage at a single electrode site. However, it is convenient to use the term *absolute voltage* to refer to the potential between one electrode and the average of the entire surface of the head. We use the average of the surface of the head in our definition of absolute voltage because the average voltage across the entire surface of the head is assumed to be zero. This assumption is true only for perfectly spherical heads (Yao, 2017), but it is a reasonable approximation for our present purposes.

Figure 5.1B shows the absolute voltage that we would expect at each of our recording electrodes. The absolute voltage at a given electrode site is equal to the source waveform multiplied by a weighting factor that represents the degree to which voltage is conducted from the specific generator to a given electrode site. For example, we're assuming that 10% of the voltage from the generator dipole is conducted to the Pz electrode site, so the weight for that site is 0.10. The source waveform has a peak amplitude of 100 μV, so the absolute voltage waveform at Pz peaks at 10 μV. The weights are negative for the electrodes on the negative side of the dipole, so the waveforms are negative-going at those sites. (The weights shown in Figure 5.1A are not the true weights, but are just examples that produce nice round numbers.) If this set of concepts about ERP generation is unfamiliar to you, you can learn more by reading Chapter 2 of Luck (2014) or by watching this video and this video from my online Introduction to ERPs course.

There is no way to measure the absolute voltage at a given electrode site. The absolute voltage is just a convenient hypothetical entity for explaining how reference electrodes work. The EEG amplifier would actually measure the voltage between each electrode site and the ground electrode. The voltage between two electrodes is simply the difference between the absolute voltages at those two sites. For example, the absolute voltage at Pz peaks at 10 μV and the absolute voltage at the ground electrode peaks at -4 μV, so the voltage between Pz and ground is 14 μV (10 minus -4). We call the voltage between a given site and the ground electrode, which is what an EEG amplifier actually measures, the *single-ended signal* (shown in Figure 5.1C).

EEG amplifiers contain noise in the *ground circuit* (which is the part of the amplifier that the ground electrode is connected to). Because all electrodes are initially measured with respect to the ground electrode, the noise in the ground circuit is present with approximately equal amplitude in the single-ended signals from all of the electrode sites. In Figure 5.1C, I added some 60 Hz noise to every signal to simulate this noise. However, this noise is often much *much* **much** larger, obscuring the actual EEG signals.

EEG recording systems therefore contain *differential amplifiers*, which use a trick to subtract away the noise from the ground circuit. The trick is to use another electrode as the *reference electrode*. The single-ended signal at the reference electrode is also recorded relative to the ground electrode, so it also contains the noise from the ground circuit. Consequently, if we subtract the reference electrode signal from the signals at our other electrodes (our *active electrodes*), the noise is approximately the same in the active and reference electrodes, so the noise is subtracted away. This is shown in Figure 5.1D, in which the single-ended signal from the left mastoid (Lm) electrode is subtracted from the signal at each of the other electrodes to create a *referenced* or *differential* signal. You can see that the referenced waveforms no longer have the 60 Hz noise that is visible in the single-ended signals. If you didn't follow this brief overview of referencing, you can watch this video and this video from the online Introduction to ERPs course or read the more detailed description in Chapter 5 of Luck (2014).

In most EEG systems, the referencing subtraction is performed in the amplifier's hardware, so you have no way of accessing the single-ended signals. You'll only ever see the referenced signals. There are, however, some exceptions. The BioSemi ActiveTwo system (which we used for the ERP CORE experiments) does not subtract the reference in hardware and instead outputs the single-ended signals. The researcher then subtracts the reference from the single-ended signals in software, after the recording session is over. During the recording, this system will show the referenced signals on the screen (to minimize noise from the ground circuit), but only the single-ended signals are saved to the file. This confuses many researchers, who do not realize that the saved data has not been referenced. If you use BioSemi, don't forget to subtract the reference! The Brain Products ActiCHamp system also obtains the single-ended signals, but the data collection software performs the referencing subtraction before the data are saved to a file. This is less confusing.

In the example shown in Figure 5.1, the Lm electrode is near the zero line for the generator dipole. As a result, the referenced voltages at each site are close in amplitude to the absolute voltages. However, that will not typically be the case, so you shouldn't assume that the referenced voltages are a good approximation of the absolute voltages. Instead, you should always think of the voltage at a given electrode site as being the difference between the signal at the so-called active electrode and the signal at the so-called reference electrode. I use the phrase "so-called" here because we are simply making a difference between two sites, and both contribute equally to the referenced voltage. If there is a large deflection in the absolute voltage at the reference electrode, then an inverted version of this deflection will be present at every so-called active site. (It's inverted because we subtract the reference.) So, when you see a waveform labeled "Cz", you are not looking at the voltage at the Cz electrode site. You are looking at the potential between the Cz site and the reference site, which is equivalent to the absolute voltage at Cz minus the absolute voltage at the reference site. The so-called active and so-called reference sites are equal contributors to this voltage. In fact, in some areas of research, Cz is used as the reference (which is equivalent to inverting the waveform). It's therefore more accurate to say that a waveform is from the "Cz channel" rather than from the "Cz electrode".

If there is one thing I hope you learn from this chapter, it's that **you need to think of a given ERP waveform as equally reflecting signals from the so-called active and so-called reference sites, not as being primarily from the active site**. The reference site you choose for your analyses can have a huge impact on how the waveforms look and which channels show the experimental effects. Unfortunately, there is no perfect reference site. In most cases, I recommend simply using whatever is common in your subfield. That way, your data can be compared with the data from other studies. If you use a different reference site, your data may end up looking quite odd, and you may think that you've discovered new effects.

This page titled 5.2: Background- Understanding Active, Reference, and Ground Electrodes is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 5.2: Background- Understanding Active, Reference, and Ground Electrodes by Steven J Luck is licensed CC BY 4.0.

5.3: Exercise- Working with the Artificial Data

In the Chapter_5 data folder, you will find an Excel spreadsheet named **simulated_data.xlsx**, which contains everything I used to create the simulated data in Figure 5.1. Open the file in Excel (or import it into Google Sheets). You will see that each line is a time point. The first column shows the latency of the time point (in ms), and the second column is the source waveform. There are also columns for the absolute voltage at each electrode site, with the weighting factor for a given site in the second row.

If you look at the equations in the cells that compute the absolute voltage values, you'll see that the absolute voltage at a given electrode site was computed by multiplying the weighting factor for that site by the source waveform. You'll also see that the single-ended voltage was created by subtracting the absolute voltage at the ground electrode from the absolute voltage at each other site, and then adding the 60 Hz noise. And you'll see that the referenced voltage was computed by subtracting the single-ended signal at the Lm electrode from the single-ended signal at a given active site. At the right side of the spreadsheet, you'll see plots of the absolute voltages, the single-ended voltages, and the referenced voltages. Take a careful look at the equations in the spreadsheet and make sure that you understand how the absolute voltage is related to the source waveform and the weights, how the single-ended voltage is related to the absolute voltage, and how the referenced voltage is related to the single-ended voltage.

The spreadsheet also contains sheets with copies of the data formatted for exporting as a text file (using the **Tab-delimited Text** format), which can easily be imported into eStudio (by clicking the **Import ERP** in the **ERPsets** panel and selecting the **Universal Text** format). This allows you to create simulated data and see how the various ERPLAB processes work.

In the spreadsheet, change the weight above the **Absolute_Ground** label from **-.04** to **-.10**. You'll see that this greatly increases the magnitude of the absolute voltage in the ground channel. And because the ground is subtracted from the other signals to create the single-ended voltage, this also changes all the single-ended voltage waveforms. But did it change the referenced waveforms?

No, it did not! The nature of the referencing procedure means that any signals or noise at the ground electrode are subtracted away from the referenced voltages. This means that you can place the ground electrode anywhere on the head, and the location does not matter. You might then wonder why we use a ground electrode at all. The answer is simple: The amplifier will freak out if there isn't a ground electrode. You need to have a ground, and it needs to be appropriately attached to the head (or anywhere on the body). But the location does not matter.

Now try changing the weight for the Lm channel from **-.011** to **-.20**. This simulates changing the location of the reference electrode, moving it farther away from the zero line. What did this do to the referenced waveforms? Not only are some of them now larger, but the polarity of the Fz waveform has also changed from negative to positive. This demonstrates how the signal at the reference electrode can have a large impact on the referenced waveforms from the so-called active electrodes.

Play around with the spreadsheet some more. For example, try changing the weights for the other electrodes or the magnitude of the 60 Hz noise.

This page titled 5.3: Exercise- Working with the Artificial Data is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 5.3: Exercise- Working with the Artificial Data by Steven J Luck is licensed CC BY 4.0.

5.4: Exercise- Average Mastoids as the Reference

For historical reasons, many ERP studies use the mastoid process (the thick bone behind the ear) as the reference electrode. This is why the data shown in Figure 5.1D use the left mastoid as the reference. However, it seems a little odd to use a reference that is lateralized to one side of the head. Might this lead to some kind of hemispheric bias in the data? In reality, this isn't usually a significant problem. However, just to be safe, many researchers use the average of the left and right mastoids (Lm and Rm) as the reference. In this exercise, we're going to look at two ways of transforming the data to reference the simulation data to the average of Lm and Rm. This exercise exemplifies an important principle, namely that data recorded using one electrode site as the reference can easily be re-referenced offline, in software, to one of the other electrode sites (or some combination of electrode sites).

Open up the spreadsheet from the previous exercise and make sure everything is back to the way it was originally (or download the spreadsheet again). Make 3 new columns just to the right of the columns for the referenced data, and label them **Fz-Avg**, **Cz-Avg**, and **Pz-Avg**. If you look at how the referencing is done for the Fz channel at the first time point (cell O4 of the spreadsheet), you'll see that the referenced value is the single-ended value for Fz (cell **J4**) at this time point minus the single-ended value for Lm (cell **M4**, but with a "$" symbol before the letter so that it remains column **M** even if we paste it somewhere else). This is the subtraction that I used to reference the data in Figure 5.1.

We want to change this so that we subtract the average of Lm and Rm from Fz. That average is simply **(Lm+Rm)/2**, which would be **($M4+$N4)/2** given that Lm is in column **M** and Rm is in column **N** of our spreadsheet. So, to create a value for Fz using the average of Lm and Rm as the reference, we need the equation for the first time point to be **=J4-($M4+$N4)/2**. Go ahead and put this equation into cell **S4**, which should be in the new column that you labeled **Fz-Avg**. If you then copy and paste this equation into cells **T4** and **U4**, the **J4** should update to **K4** and **L4**, respectively. If you then copy and paste these three cells to all the remaining time points, the row numbers should update, and you'll have the appropriate values for all the cells.

Now compare the new values you created to the original referenced values (with Lm as the reference). They should be pretty similar. This is because the single-ended signal at Rm is nearly identical to the single-ended signal at Lm, so the average of Lm and Rm is nearly identical to the Lm signal. In turn, this is because the weights for Lm and Rm are pretty similar. However, there may be situations in which the Lm and Rm signals are not so similar (e.g., if the generator dipole happened to be near Lm without a generator near Rm). This is why it's a good idea to use the average of Lm and Rm rather than just using one side.

In most systems, you don't have access to the single-ended signals, so you wouldn't be able to use this approach for referencing the data to the average of the two mastoids. For example, the original data might all be referenced to Lm. However, if you have a recording of Rm (also referenced to Lm), there is a trick you can use to re-reference the data to the average of Lm and Rm. The trick was described by Paul Nunez in his classic book on the biophysics of EEG (Nunez, 1981), and the algebra is spelled out in Chapter 5 of Luck (2014). Specifically, if you subtract 50% of the Rm signal from each channel that was already referenced to Lm, this is equivalent to taking the single-ended data from the active electrode and subtracting the average of the single-ended Lm and Rm signals. For example, to re-reference Fz to the average of Lm and Rm, you would take the already-referenced Fz channel (which is really Fz – Lm) and subtract 0.5 of the Rm channel (which is really Rm – Lm).

Create three new columns labeled **Fz-Avg2**, **Cz-Avg2**, and **Pz-Avg2**. Put equations into these columns that use this different approach to re-referencing. That is, take the values that were already referenced (Columns O-R) and subtract 0.5 times the Rm channel from the Fz, Cz, and Pz channels. Once you've created these three new channels, compare them with the previous set you created. You should see that these two ways of referencing to the average of Lm create exactly identical results. However, the first method can't be used in most systems because it requires access to the single-ended signals, so you may need to use the second method. When you're working with real data instead of simulated data, you'll do the referencing in eStudio using the **Channel Operations** panel rather than in a spreadsheet. However, Channel Operations uses equations that are much like the spreadsheet equations. I hope that the experience you've now gotten with the spreadsheet will give you a better understanding of how the equations work in Channel Operations.

This page titled 5.4: Exercise- Average Mastoids as the Reference is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 5.4: Exercise- Average Mastoids as the Reference by Steven J Luck is licensed CC BY 4.0.

5.5: Exercise- Re-Referencing the N400 ERP CORE Data

Now let's see how re-referencing is accomplished in ERPLAB with the data from the N400 ERP CORE experiment. To keep things simple for this exercise, we'll just work with the grand average ERPs.

Start by clicking the **Reset** button under the lower right corner of the plot region and then clicking **Reset ERPLAB Studio** in the window that appears. Go to the **ERP** tab, and use the **Current Folder** button in the **ERPsets** panel to set the current folder to be the **Chapter_5** folder. Click the **Load ERP** button and load the file named **Grand_N400_diff.erp**. Then select Bins 3, 4, and 5 in the **Bin & Channel Selection** panel. To get a better channel layout, go to the **Plot Settings** panel, click **Custom** for the **Grid Layout**, click the **Import** button, and select the file named **GridLocations_Custom.tsv**. Click **Apply**, and then you should see the electrodes laid out in their approximate positions on the scalp.

The waveforms should look familiar from the previous chapters. As before, Bins 3 and 4 are the related and unrelated target words, respectively. Bin 5 is the unrelated-minus-related difference wave. You can see a beautiful N400, especially in the difference wave and especially in the CPz channel.

But remember, the waveform in a given channel is determined just as much by the reference electrode as by the active electrode, even though the active electrode is typically used to name the channel. Unless we know what was used as the reference, we don't really know what we're looking at in this plot. When I'm reading a journal article, I go to the Methods section and find out what was used as the reference before I look at any of the waveforms. If you look at the Methods section for the paper on the ERP CORE (Kappenman et al., 2021), you'll see that we used the average of the P9 and P10 electrode sites as the reference for the N400 experiment (and most of the other experiments). P9 and P10 are quite close to the left and right mastoids, so the waveforms look almost identical to what we would have gotten using the average of Lm and Rm as the reference. However, it's easier to get a good electrical connection with P9 and P10, so we now use these electrodes as our standard reference sites.

Once you understand that referencing is just a matter of subtraction, you can use some very simple algebra to figure out how to re-reference data that have already been referenced. To demonstrate, we're going to re-reference the N400 data to the Cz electrode site. Almost all N400 studies use the mastoids (or something nearby) as the reference, so Cz would be an unusual choice for an N400 experiment. However, Cz is used as the default reference in some EEG recording systems (e.g., the EGI system), so it's easy to imagine that someone would look at N400 data with a Cz reference (especially someone who didn't understand the importance of the reference location). As you'll see, the data look quite different with a Cz reference.

Let's start with the algebra. I'm not much of a math person, so I promise it will be simple. For the CPz channel, the waveforms that you just looked at—with the average of P9 and P10 as the reference—can be expressed conceptually as:

$$CPz_{Referenced} = CPz_{Absolute} - [(P9_{Absolute} + P10_{Absolute})/2]$$

In other words, the referenced voltage at CPz is just the absolute voltage at CPz minus the average of the absolute voltages from P9 and P10. That's not how the referenced CPz channel was actually created, but it's conceptually equivalent. Here's the corresponding expression for the Cz channel:

$$Cz_{Referenced} = Cz_{Absolute} - [(P9_{Absolute} + P10_{Absolute})/2]$$

Our goal is to re-reference the CPz channel so that Cz is now the reference. In other words, we want to create a new channel defined as:

$$CPz_{Referenced} = CPz_{Absolute} - Cz_{Absolute}$$

It turns out that we can get this by just taking the data that have been referenced to the average of P9 and P10 and subtracting Cz from each channel. For CPz, this gives us:

$$CPz_{Referenced} - Cz_{Referenced}$$

$$= (CPz_{Absolute} - [(P9_{Absolute} + P10_{Absolute})/2]) - (Cz_{Absolute} - [(P9_{Absolute} + P10_{Absolute})/2])$$

$$= CPz_{Absolute} - Cz_{Absolute} - [(P9_{Absolute} + P10_{Absolute})/2] - [-(P9_{Absolute} + P10_{Absolute})/2]$$

= $CPz_{Absolute}$ - $Cz_{Absolute}$

The original reference is in both signals, so it drops out when we do the subtraction. Pretty cool, eh? I think so, but maybe that's why I've devoted much of my career to ERP research.

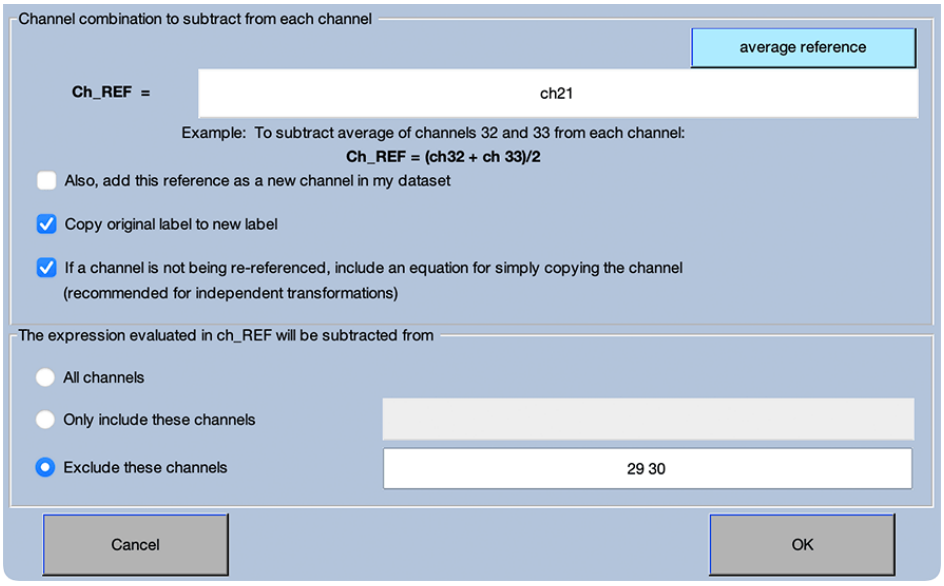
Okay, let's try it. With **Grand_N400_diff.erp** selected, go to the **Channel Operations panel** in the ERP tab (not the one in the EEG tab). Then change the **Mode** from **Modify existing ERPset** to **Create new ERPset**. We used the **Modify existing ERPset** mode in Chapters 3 and 4 when we added a new channel that was the average of several existing channels. But now we don't want to add new channels to our existing set of channels—we want to create a brand-new ERPset in which all of the channels have Cz as the reference. Otherwise, we would have a huge number of channels in our ERPset.

When we create a new ERPset, we use **nch** (short for "new channel") to indicate the new channels that we're creating and **ch** to indicate the original channels. For our first channel, FP1, we would subtract the original Cz channel (**ch21**) from the original FP1 channel (**ch1**) to create the new channel (**nch1**). The formula we would enter into the Channel Operations GUI would therefore be: **nch1 = ch1 - ch21 Label FP1**.

We would then repeat this for channels 2-28. Channels 29 and 30 are the horizontal and vertical electrooculogram (EOG) channels, which use a different reference, so we would just create new channels that are exact copies of the original channels for these sites.

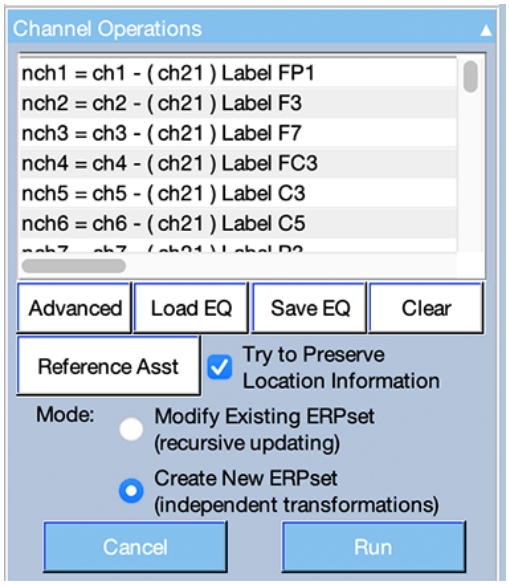
To avoid typing all these equations, click the **Reference Asst** button. You'll see a new window that looks like Screenshot 5.1. If you type **ch21** in the text box labeled **Ch_REF** near the top of the window, the Reference assistant will create appropriate equations using **ch21** as the reference for every channel. Make sure that the checkboxes are set as shown in the screenshot. Also, select **Exclude these channels** and enter **29 30** into the text box, which will cause it not to create re-referencing equations for the horizontal and vertical EOG channels. Then click **OK**.

Screenshot 5.1



Now the **Channel Operations** panel should be filled with equations, as shown in Screenshot 5.2. The equation for each channel simply creates a new channel that is the original channel minus channel 21 (Cz). However, if you scroll down to channels 29 and 30, you'll see that the equations for these channels just copy the original horizontal and vertical EOG channels without changing the reference.

Screenshot 5.2



You can now click **Run** to execute this set of equations. Because you've selected the **Create new ERPset** option, you'll see the usual window for saving a new ERPset. It will suggest a name composed of the original ERPset name with **_chop** (for "channel operations") appended to the end. However, let's use **_CzRef** instead of **_chop** so that the name is more informative. You should save it as a file if you're not going to do the next step right away.

Channel Operations and the ERPLAB Design Philosophy

Creating a separate equation for each channel might seem overly complicated. After all, we're applying the same operation to almost every channel, so there's considerable redundancy in the list of equations. Other EEG/ERP analysis systems have much more concise ways of specifying how to re-reference the data. However, by specifying a literal equation for each channel, you know exactly what the operation is doing to your data. In other systems, it's not obvious exactly what the software is doing when you re-reference the data. In fact, when I've used other systems, I've resorted to passing artificial data through the re-referencing procedure so that I could figure out exactly what it was doing. In ERPLAB, the use of explicit equations for each channel leaves no uncertainty about how the re-referencing works. This reflects one of our core design philosophies when we created ERPLAB: No magic! We want researchers to know exactly what our software is doing to their precious data.

As an example of this philosophy, check out our documentation page on Timing Details. When you say you want to measure the mean amplitude between 300 and 500 ms, what happens if your sampling rate is 256 Hz and you don't have time points at exactly 300 and 500 ms? We describe the exact algorithm that we use to round up or down.

The equation approach we use in Channel Operations has another benefit: It's incredibly flexible. You can perform all kinds of interesting transformations of the data, going way beyond re-referencing. For example, you can take the absolute value of a channel to *rectify* it (which is useful if you have a channel that contains EMG data). You can create a new channel with the global field power (Skrandies, 1989), as will be described in a later section. You can compute the difference between two channels. And, as you saw in Chapter 3, you can create a new "cluster" channel that is the average of a subset of your channels.

Now take a look at the data from Bins 3–5. You will see that the Cz channel is now flat, because a channel minus itself is zero. You'll also see that most of the channels now have a more positive voltage for the unrelated targets than for the related targets, and the unrelated-minus-related difference is now positive instead of negative. In other words, with Cz as the reference, we have a P400 instead of an N400! Moreover, rather than being largest in the CPz channel, the unrelated-minus-related difference is now largest in the F7 channel. This illustrates the very important point that you cannot interpret the data from a given channel unless you know the reference site as well as the active site.

Next, let's re-reference the data using Oz as the reference. That is, select **Grand_N400_diff** again, click the **Clear** button in the **Channel Operations** panel to clear out the old equations, and then follow the same steps you used to re-reference the data to

Cz but with **ch12** instead of **ch21** as the reference site. Name the new ERPset **Grand_N400_diff_OzRef**. When you look at the difference wave, you'll see a small N400 in the CPz channel and an opposite-polarity "P400" in the F7 channel. So, you can see that the choice of the reference determines whether a given ERP component is positive, negative, or positive at some sites and negative at others. The reference electrode also impacts which channel has the biggest effects. It really matters!

This page titled 5.5: Exercise- Re-Referencing the N400 ERP CORE Data is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 5.5: Exercise- Re-Referencing the N400 ERP CORE Data by Steven J Luck is licensed CC BY 4.0.

5.6: Exercise- The Average Reference

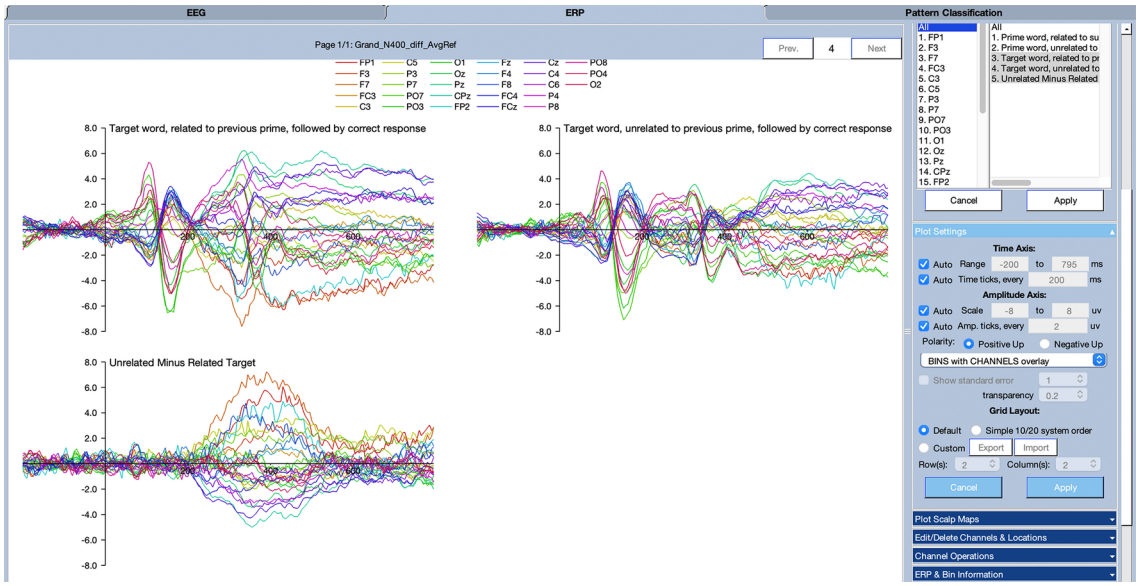
Earlier in this chapter, we defined the *absolute voltage* as the potential between a given electrode and the average of the entire surface of the head. This is a strictly theoretical concept, because we can't record from the entire surface of the head. How would you get an electrode on the bottom side of someone's skull?

However, many researchers try to approximate the absolute voltage by using the average across all the electrode sites in their recording as the reference. Although this has been shown to be a good approximation for one particular generator site when an extremely broad array of electrodes is used (Dien, 1998), it often provides a very poor approximation of the absolute voltage (see Chapter 7 in Luck, 2014). Nonetheless, the *average reference* can still be useful under some conditions. For example, it is commonly used for studies of face-elicited N170 activity (Rossion & Jacques, 2012), and we used it for the N170 paradigm in the ERP CORE (Kappenman et al., 2021).

In this exercise, we'll re-reference the data in **Grand_N400_diff.erp** to the average of all the EEG electrodes. To accomplish this, load **Grand_N400_diff.erp** into eStudio if it's not already loaded, and make sure that the original version of this ERPset is selected in the **ERPsets** panel (rather than the ERPsets that you created with Cz or Oz as the reference). Go to the **Channel Operations** panel and click the **Clear** button to clear out any equations that remain in the text box from the last time you used this routine, and make sure that the **Mode** is set to **Create new ERPset**.

Now click the **Reference Asst** button. In the text box labeled **Ch_REF** near the top of the window, type **avgchan(1:28)**. This indicates that you want to use the average of channels 1–28 as the reference. We're excluding channels 29 and 30, which are the EOG channels. As before, select **Exclude these channels** with **29 30** in the text box. Then click **OK**. You'll see that the list of equations in the **Channel Operations** panel subtracts **avgchan(1:28)** from each individual channel, except for the EOG channels. If you go through the algebra of this subtraction, you'll see that this will create EEG channels that are equal to the active electrode minus the average of all the electrodes. Now click **Run** and name the new ERPset **Grand_N400_diff_AvgRef**.

If you look at the waveforms for Bins 3-5, you'll see that the difference between related and unrelated targets with the average reference is positive at some sites and negative at others. This is an inevitable consequence of using the average of all sites as the reference. That is, at every moment in time, the average-referenced voltages across the different electrode sites must sum to zero, so the voltage will be negative at some electrode sites and positive at others. This is easier to see with a *butterfly plot*, in which we overlay the waveforms for all the channels. To make a butterfly plot, go to the **Plot Settings** panel and change the pop-up menu in the middle from **CHANNELS with BINS overlay** to **BINS with CHANNELS overlay** (see Screenshot 5.3). Now, each individual panel of the plot shows one bin, with all the channels overlaid. The result often looks like a butterfly (although the example shown in Screenshot 5.3 does not).



Screenshot 5.3

In the butterfly plot, you can see that the voltage at any given time point is positive at some electrode sites and negative at others. And if you summed together all the channels for a given bin, you would get a value of zero at every time point. This is true both for *parent waveforms* (e.g., the waveforms for the related and unrelated target words) and for difference waveforms. If you select **Grand_N400_diff** in the **ERPsets** panel to see the butterfly plot with the original reference (the average of P9 and P10), you'll see that it is no longer the case that the channels all sum to zero at each time point. For example, almost all of the channels are negative during the N400 period in the difference wave. It's only when you use the average of all sites as the reference that you necessarily have equal amounts of positive and negative at a given time point.

So, when you use the average reference, don't think you've discovered something interesting when you find that your experimental effect is positive in some channels and negative in others. It's just a necessary consequence of the algebra. Some components and experimental effects will exhibit such polarity inversions with other references, but it is inevitable with the average reference.

Heads are not actually spheres

Using the average across sites as the reference in order to approximate the absolute voltage assumes that the surface of the head sums to zero, but this is only true for spheres. I have yet to meet someone with a spherical head. And no neck. Fortunately, there is a way to estimate the true zero, called the *Reference Electrode Standardization Technique* (REST), and there is an EEGLAB plugin that implements it (Dong et al., 2017). To use it, you'll need to launch the EEGLAB GUI instead of the eStudio GUI. But after you've applied this reference and saved the resulting EEGset, you can open the EEGset in eStudio for any further processing.

This page titled 5.6: Exercise- The Average Reference is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 5.6: Exercise- The Average Reference by Steven J Luck is licensed CC BY 4.0.

5.7: What is the Best Reference Site?

Figure 5.2 shows the data for a few key channels with the four references we've used so far (the average of P9 and P10, Cz, Oz, and the average of all the EEG sites). As you can see, the choice of reference electrode has a massive effect on the waveforms. But which of these is the correct way to reference the data? That is, which reference site will give us the true waveforms?

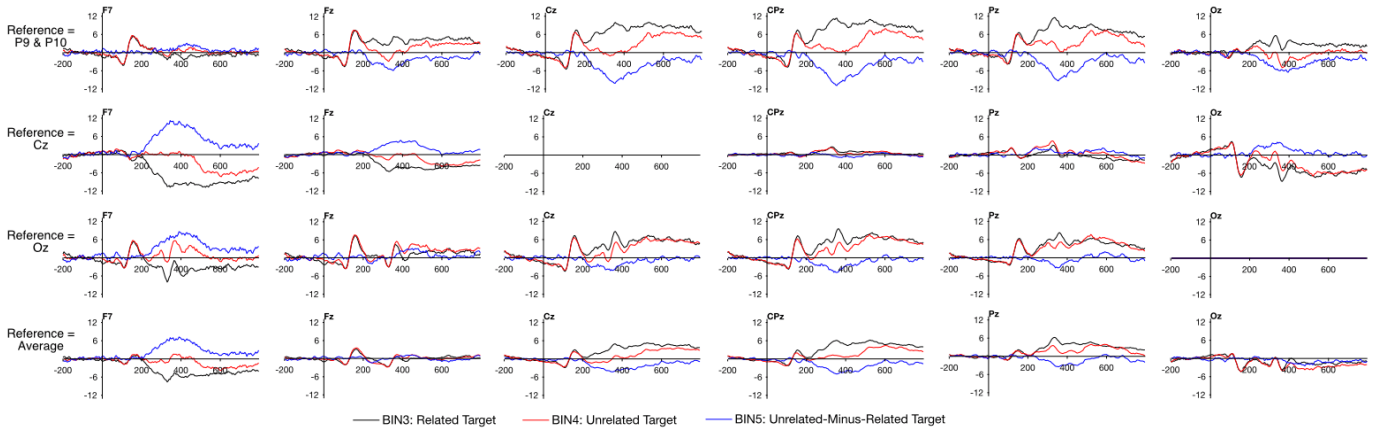


Figure 5.2. Grand average ERP waveforms for selected channels from the N400 ERP CORE experiment with four different references.

If you think about it, these are not meaningful questions. Voltage is the potential for electrical current to flow between two places. It makes no sense to talk about the voltage at a single electrode site. Even if we could measure (and not merely estimate) the absolute voltage, it would have no special truth status. For example, the cortical generator site of a given ERP component could be quite far away from the electrode with the largest voltage.

Even if there is no "correct" way to reference the data, is there a "best" way? I discuss this issue in detail in Chapter 5 of Luck (2014). My bottom line is that the best approach is usually to use whatever is most common in a particular area of research. If you don't use the same reference as most other studies, then people can't compare your data with the data from other similar studies. And you might think you've discovered a new effect. For example, most language ERP studies use the average of the mastoids as the reference, and if you use the average reference, you might think you've discovered a new "P400" component at the F7 electrode site.

If you have a good reason to use an atypical reference in a given study, you should also show what the data look like with the typical reference (e.g., by providing the waveforms with the typical reference in online supplementary materials). That way, no one will be confused about the relationship between your study and other studies.

My final piece of practical advice about the reference electrode—especially when you're first starting out in ERP research or looking at a new component—is to look at your data with multiple different references (as in Figure 5.2). That way, you won't be lulled into thinking that the waveform in a given channel primarily reflects brain activity at the so-called active electrode for that channel. That is, you'll see that the reference has a big effect on your data, and you'll realize that no matter what reference you use, you're looking at the potential between two sites.

I've made this recommendation to many people, but I don't know how many of them have followed my advice. However, I do know that the people who have followed it and then discussed their experiences with me said that it really helped them understand their data better. So give it a try!

This page titled 5.7: What is the Best Reference Site? is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 5.7: What is the Best Reference Site? by Steven J Luck is licensed CC BY 4.0.

5.8: Exercise- Current Density

The need to use a reference electrode to measure voltage can sometimes make it difficult to answer the scientific question of interest. Fortunately, there are two common ways of transforming the data into a reference-free signal, namely converting the voltage to *current density* or computing the *global field power*. I've used both transformations, and they can be quite useful.

Let's start with current density (also called *current source density*). Unlike voltage, which always involves two places, current is the flow of charge elements at a single point. Consequently, there is no reference electrode for measures of current. Unfortunately, we can't directly measure the current flowing out of the scalp at a given electrode site. But fortunately, we can *estimate* the current flow from the pattern of voltage across a set of electrodes. To estimate the current flow perpendicular to the scalp (the *current density* or *current source density*) at a given time point, we apply the *Laplacian* transform to the distribution of voltage across the scalp at that time point. The details are described in Chapter 7 of Luck (2014). Here, we'll see how it's actually done using ERPLAB.

Load **Grand_N400_diff.erp** into eStudio if it's not already loaded, make sure that it's selected in the **ERPsets** panel, and select Bin 5 (unrelated minus related target) in the **Bin & Channel Selection** panel. Also, go to the **Plot Settings** panel and change the pop-up menu in the middle back to **CHANNELS with BINS overlay**.

The Laplacian transform requires that the 3-dimensional locations of the electrodes are specified in the ERPset. They should already be present in **Grand_N400_diff**. If you needed to add them, you would go to the **Edit/Delete Channels & Locations** panel and click on the **Add or edit chan locations** button.

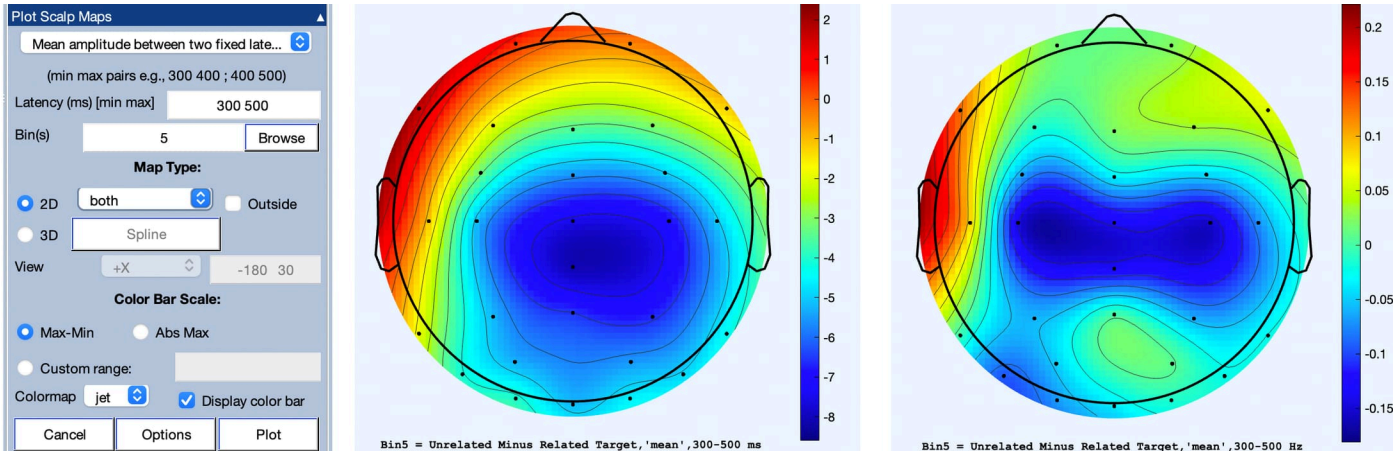
The Laplacian transform also requires that all channels have the same reference, so we'll need to eliminate the bipolar VEOG and HEOG channels from our data (channels 29 and 30). To do this, go to the **Edit/Delete Channels & Locations** panel, type **29 30** in the **Chan:** text box, and click **Delete chan**. Those channels will no longer be plotted in the eStudio GUI, and they are now gone in the **Bin & Channel Selection** panel.

Now we're ready to convert from voltage to current density using the **Convert Voltage to CSD** panel. You'll see some parameters that control the conversion, but you can just leave them at their default values. Click **Run** to do the conversion and name the resulting ERPset **Grand_N400_diff_CSD**.

If you look at the CPz channel for Bin 5, you'll see an N400 (a negativity peaking around 400 ms). However, if you look at the surrounding sites, you'll see that the N400 current density has a much more focused scalp distribution than the N400 voltage that you plotted prior to performing the Laplacian transformation. For example, the N400 is quite large at Pz in the voltage waveforms but near zero in the current density waveforms. This is typical: The Laplacian transformation creates a narrower scalp distribution. This is sometimes very useful, because it allows us to separate components that have overlapping voltage distributions. Once we convert voltage to current density, the components may be at distinct sites, allowing us to measure them separately.

We can see this better by plotting scalp maps. Let's start with the voltage. Select the original **Grand_N400_diff** ERPset (with the EOG channels removed) in the **ERPsets** panel, and go to the **Plot Scalp Maps** panel. Set the plotting parameters as shown in Screenshot 5.4. We're going to plot the difference wave (Bin 5), using the mean voltage from 300 to 500 ms. Make sure that the **Map Type** is set to **both** (so that it shows both a color within each region and a contour line between regions). When everything is set, click **Plot**, and you should see a scalp map like the one in the middle of Screenshot 5.4. Note that the negative voltage is centered at the CPz electrode site and broadly distributed, with a slight bias toward the right hemisphere (which is typical for the N400).

Now select the ERPset with the current density and repeat the procedure for plotting the scalp map. The result should look like the map at the right of Screenshot 5.4. Note that the negativity is now much sharper, and you can actually see separate foci over the left and right hemispheres. The most important thing, however, is that we are now looking at (an estimate of) the current flowing out of the scalp at each location, not a potential between each location and the reference site. The location of the original reference site no longer matters.



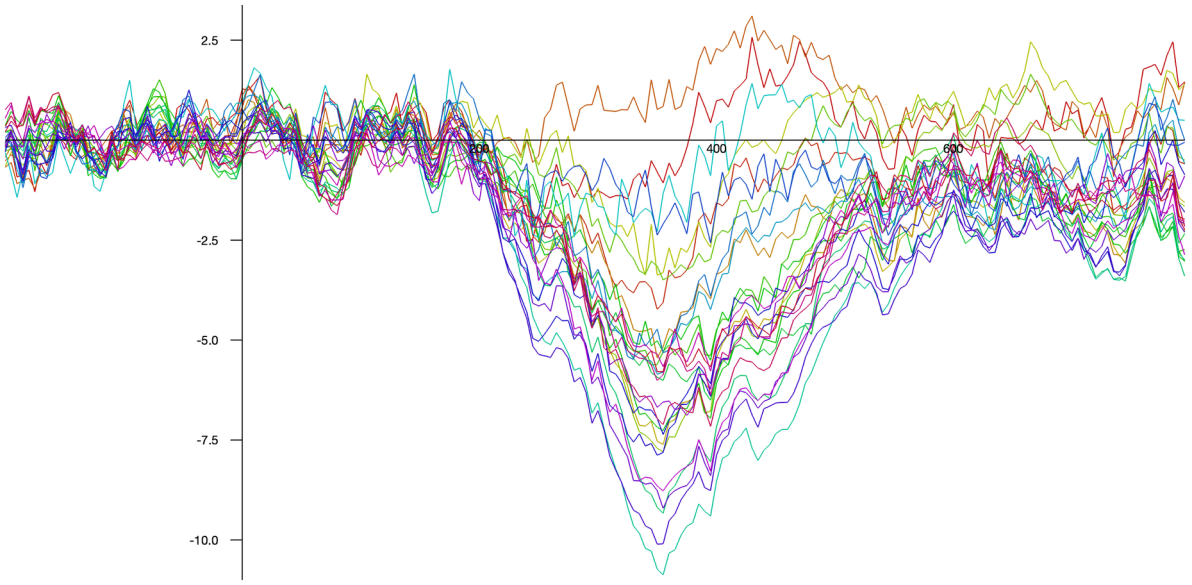
Screenshot 5.4

This page titled 5.8: Exercise- Current Density is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 5.8: Exercise- Current Density by Steven J Luck is licensed CC BY 4.0.

5.9: Exercise- Global Field Power

Another useful reference-free transformation is *mean global field power (MGFP)*. This transformation is easiest to explain visually. To begin, reset everything with the **Reset** button, then load **Grand_N400_diff.erp** in the **ERPsets** panel. View the difference wave as a butterfly plot by selecting Bin 5 and all the EEG channels (channels 1–28) in **Bin & Channel Selection** and then selecting **BINS with CHANNELS overlay** in **Plot Settings**. Once you click **Apply**, it should look like Screenshot 5.5. Notice that all the waveforms are relatively close together until the start of the N400 at ~200 ms, and they come together again at the end of the N400 at ~600 ms.



Screenshot 5.5

In general, when the amplitude of a given ERP component increases at one site, there will also be an increase in the spread of the voltages across all the electrode sites. Mean global field power quantifies the amount of spread as the standard deviation (SD) across electrode sites at each point in time. However, it is only valid when all the electrode sites share the same reference, which is why Screenshot 5.5 shows only the EEG channels (the EOG channels were referenced differently).

To compute the mean global field power for the EEG electrodes, go to the **Channel Operations** panel and make sure that the **Mode** is set to **Modify existing ERPset**. We're going to create a new channel (channel 31) with the mean global field power for the EEG channels (channels 1–28). To do this, put the following equation in the **Channel Operations** text box: **ch31 = mgfperp(1:28) label MGFP**.

Now click **Run**. If you go to the Bin & Channel Selection panel, you should now see that there is a channel 31 labeled MGFP. Select this channel along with channels 1-28 and click **Apply**. The MGFP channel will be visible as a positive value at all time points (because the standard deviation of a set of values is always positive). You should be able to see that the height of the MGFP channel tracks the spread across the EEG channels.

Mean global field power has several useful characteristics. First, the reference electrode has no impact on MGFP (because it contributes equally to each channel and is effectively a constant). Second, it aggregates all the scalp signals into a single channel, which can avoid the need to select which channels to include in an analysis (Hamburger & Van der Burgt, 1991). Third, it is typically cleaner than the individual-channel waveforms (because noise is typically minimized by transformations that combine the data from multiple sites). However, you should keep in mind that the standard deviation across channels will increase as the noise level increases, so special methods are necessary to compare mean global field power amplitudes across conditions that differ in the number of trials or any other factor that might impact the noise level (Files et al., 2016).

In my lab, we like to use mean global field power to ask when information about a stimulus has reached the cortex. That is, we look at the first point at which this signal starts increasing above the baseline level (see, e.g., Kiat et al., 2021). We then use this

information to determine the start of the measurement window for our primary analyses. By using mean global field power for this purpose, we get a very clean signal, and we don't have to choose a specific electrode site.

This page titled 5.9: Exercise- Global Field Power is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck.

- 5.9: Exercise- Global Field Power is licensed CC BY 4.0.

5.10: Exercise- Referencing the EEG Data from the ERP CORE N400 Experiment

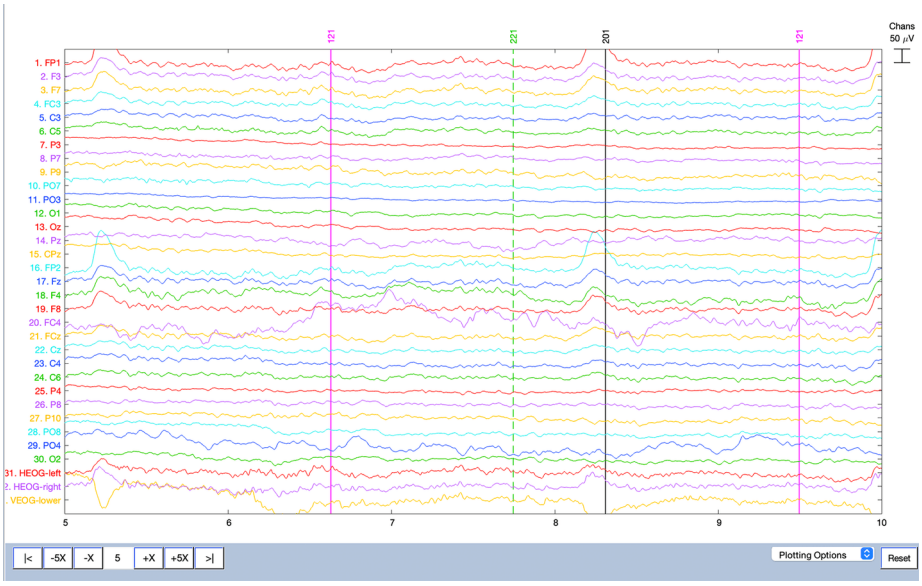
For the sake of simplicity, the preceding exercises in this chapter were performed on grand average ERP waveforms. This is fine for visualization, but you will ordinarily need to reference or re-reference the single-subject EEG data prior to artifact rejection or correction. In this exercise, we'll see how to reference and re-reference the continuous EEG using the data from the ERP CORE N400 experiment.

Keep in mind that the ERP CORE data were recorded with a BioSemi ActiveTwo EEG recording system, which saves the single-ended data rather than saving the referenced data (see Section 5.2). In this exercise, we will reference the data to the average of P9 and P10. Because these data were recorded single-ended and have never been referenced, we will be "referencing" the EEGset rather than "re-referencing" it.

Referencing All Channels to the Average of P9 and P10

To begin, go to the **EEGsets** panel in the **EEG** tab, click **Load EEG**, and choose the EEGset named **6_N400_unreferenced.set**. This is the N400 EEGset for the participant we looked at in Chapter 2. In the exercises for that chapter, the EEGset had already been referenced, but the version for this exercise has not been referenced. However, I did do some preprocessing on this EEGset (e.g., filtering from 0.1–30 Hz) to save you some steps.

Once you've loaded the EEGset, click the **+X** button at the lower left of the eStudio window to scroll to the 5-second point. You should see something like Screenshot 5.6.

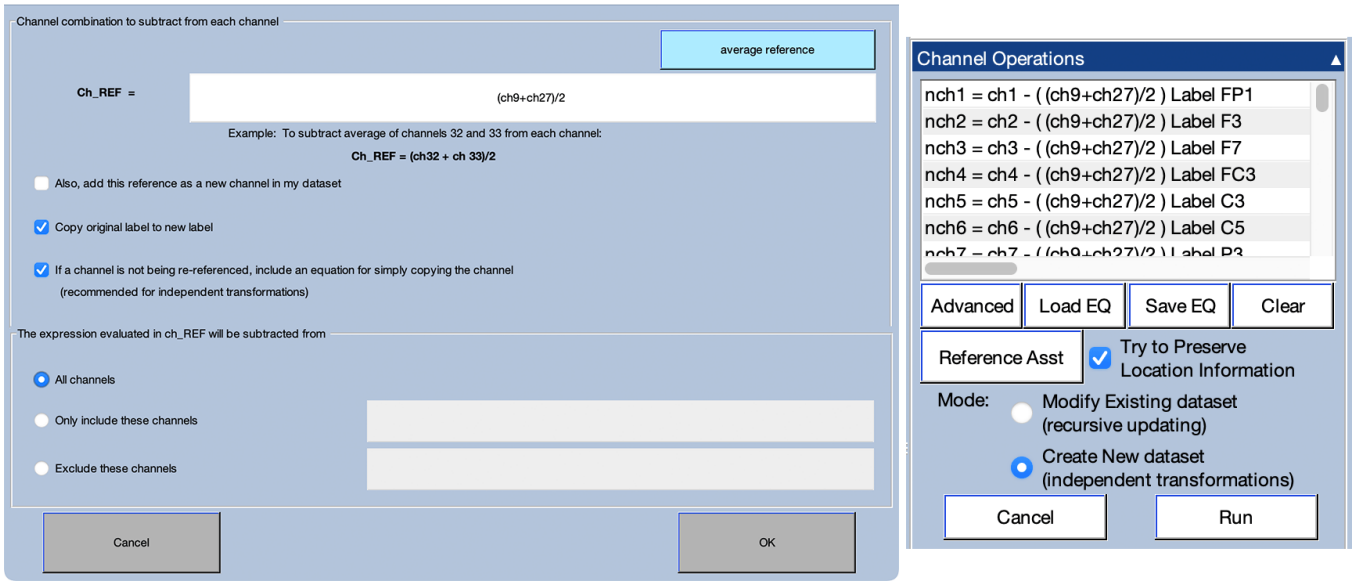


Screenshot 5.6

To reference this EEGset to the average of P9 and P10, start by going to the **Channel Operations** panel in the **EEG** tab and setting the Mode to **Create new dataset**. Note that there are **Channel Operations** panels in both the **EEG** tab and the **ERP** tab; the only major difference is whether they operate on EEGsets or ERPsets. For this exercise, you should use the one in the **EEG** tab.

Click the **Reference Asst** button and set it up as shown on the left side of Screenshot 5.7, with **(ch9+ch27)/2** in the **Ch_REF** text box. P9 is in Channel 9, and P10 is in Channel 27, so this simple equation gives us the average of P9 and P10. Make sure that the check boxes are set as shown in Screenshot 5.7 and that the radio button for **All channels** is selected. Then click **OK** to create the equations. In the **Channel Operations** panel, you should now see that each channel being created will be computed as the

original channel minus the average of P9 and P10 (see the right side of Screenshot 5.7).



Screenshot 5.7

Click **Run**, and name the new dataset **6_N400_ref** to indicate that it has now been referenced.

You can compare the unreferenced data to the referenced data by going back and forth between **6_N400_unreferenced** and **6_N400_ref** in the **EEGsets** panel. They should look only slightly different. This is because my lab puts a huge amount of effort into having clean EEG recordings, including recording within a shielded chamber and using very high-quality EEG recording systems, such as the Biosemi ActiveTwo or the BrainProducts actiCHamp. In many labs, the referenced data will look much cleaner than the unreferenced data.

Creating Bipolar EOG Channels

The next step is to create special *bipolar* versions of the vertical EOG and horizontal EOG (VEOG and HEOG) channels, which are used to detect blinks and eye movements, respectively. If you aren't already familiar with the idea of using bipolar VEOG and HEOG channels, I recommend watching this video from the online Introduction to ERPs course. I will provide just a brief explanation here.

To understand why bipolar channels are useful for isolating blinks, take a look at the channel labeled VEOG-lower at the bottom of the plot. This electrode was located just below the right eye, and the negative-going deflection that you can see in this channel shortly after the 5-second mark is an eyeblink. If you look at FP2 (channel 16) in the same time period, you'll see a positive-going deflection. This is a key attribute of eyeblinks: They are opposite in polarity under the eyes (where VEOG-lower is located) and over the eyes (where FP2 is located).

However, most brain activity is quite similar at VEOG-lower and FP2 (because they are quite close together). We can therefore isolate the blink-related EOG activity and largely eliminate the brain activity by subtracting the FP2 signal from the VEOG-lower signal. In addition, because the blink activity is positive at FP2 and negative at VEOG-lower, this subtraction also makes the blinks larger (because a negative minus a positive is an even larger negative). When we try to reject trials with blinks, this subtraction makes our job much easier, because it makes the blinks bigger and makes non-blink EEG activity smaller. These are the reasons why we want to create a *bipolar VEOG* channel in which we subtract FP2 from VEOG-lower. This is the same as saying that we will re-reference the VEOG-lower channel to the Fp2 channel.

All voltages are actually "bipolar" in the sense of having two poles (active and ground or active and reference). However, the term *bipolar* is used in EEG recordings when a channel uses a special reference that is different from the other channels. We will leave all our EEG channels referenced to the average of P9 and P10, and only the VEOG-bipolar channel will use FP2 as the reference.

We also usually create a bipolar HEOG signal to isolate horizontal eye movements. During a typical EEG recording, we place

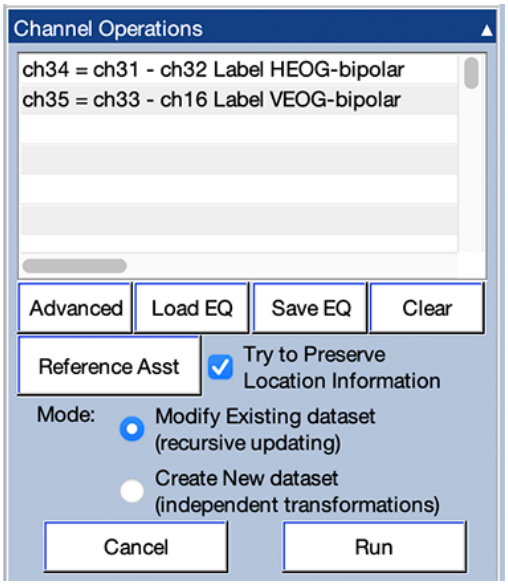
the HEOG-L electrode next to the left eye and the HEOG-R electrode next to the right eye. When the eyes move leftward, this produces a negative voltage at HEOG-left and a positive voltage at HEOG-right. This reverses for rightward eye movements. By creating a bipolar HEOG channel (HEOG-right minus HEOG-left), we can effectively double the size of the eye movement voltage. In addition, brain activity is usually quite similar at these two sites, so this subtraction also eliminates most of the brain activity. Thus, the bipolar HEOG signal is very useful when we try to reject trials with horizontal eye movements.

We're going to keep the original VEOG-lower and HEOG channels, referenced to the average of P9 and P10, for reasons that will become clear in the chapter on artifact correction. We will create additional VEOG-bipolar and HEOG-bipolar channels. To do this, first make sure that **6_N400_ref** is selected in the **EEGsets** panel. Then go to the **Channel Operations** panel (in the EEG tab) and click the **Clear** button to remove the equations you created previously. Then change the **Mode** to Modify Existing dataset (because we're going to add two new channels to **6_N400_ref**). Now type these two equations into the **Channel Operations** text box:

ch34 = ch31 - ch32 Label HEOG-bipolar

ch35 = ch33 - ch16 Label VEOG-bipolar

The list of equations should look like that shown in Screenshot 5.8. Click **Run** to update **6_N400_ref** with the new channels. You'll need to refer to this dataset in the next exercise, so save it as a file by clicking the **Save EEG** button in the **EEGsets** panel. This will save it in a file named **6_N400_ref.set** in the current folder.



Screenshot 5.8

If you look at the waveforms in the plotting region, the most obvious change is that the bipolar EOG channels are now present. You can see that blinks are larger in the VEOG-bipolar channel than in the VEOG-lower or FP2 channels (because a negative minus a positive is a bigger negative). This will make it easier to reject trials with eyeblinks later. It will also make it easier for us to determine which ICA component reflects eyeblinks if we do artifact correction.

The N400 task used stimuli presented in the center of the monitor, so there aren't any obvious horizontal eye movements. We'll see what those look like in the chapter on artifact rejection.

This page titled 5.10: Exercise- Referencing the EEG Data from the ERP CORE N400 Experiment is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 5.10: Exercise- Referencing the EEG Data from the ERP CORE N400 Experiment by Steven J Luck is licensed CC BY 4.0.

5.11: Exercise- Other Common Re-Referencing Scenarios

The previous exercise began with the single-ended data produced by the BioSemi system we used for the ERP CORE experiments, so we *referenced* the data rather than *re-referencing* it. Most EEG systems output differential (referenced) data rather than single-ended data, but it is often necessary to re-reference the data from these systems offline. In this exercise, we'll see how to implement two common re-referencing scenarios you might encounter with differential data. In the first scenario, the data were recorded with a left mastoid reference, but you want to use the average of the left and right mastoids as the reference. In the second, the data were recorded with Cz as the reference (which is the default in the EGI system), and you again want to use the average of the mastoids as the reference. Another common scenario would be to take referenced data and re-reference to the average of all sites, but we already saw how to do that in an earlier exercise. (That exercise used averaged ERPs rather than EEG, but re-referencing works the same with ERP data and EEG data.)

I've created two versions of the EEG data from the preceding N400 example for these two scenarios, and you can find them in the Chapter_5 folder. The file named **6_N400_LmRef.set** has EEG and EOG data referenced to the left mastoid (Lm), including a right mastoid (Rm) channel that has EEG data from Rm referenced to Lm. The file named **6_N400_CzRef.set** has EEG and EOG data referenced to Cz (including Lm and Rm channels that were referenced to Cz). Go ahead and load these two files into eStudio using the **EEGsets** panel.

Let's start by re-referencing the data in **6_N400_LmRef.set** to the average of the left and right mastoids. As we saw in the exercises using the spreadsheet, we can do this by simply subtracting 50% of the voltage between Rm and Lm from the voltage in each channel. Select **6_N400_LmRef** in the **EEGsets** panel and then go to the **Channel Operations** panel (in the **EEG** tab). Click **Clear** to remove any existing equations and set the **Mode** to **Create new dataset**. You can then use the **Reference Asst** to create the appropriate equation for each channel. But this time, I'm not going to tell you how to do it; you should figure it out for yourself. You can look at the list of **Existing Channels** in the Channel Operations GUI to figure out the channel number for the Rm signal. (If you get stuck, the equations are in a file named **Re-Reference_Lm.txt**. But you'll learn a lot more if you figure it out for yourself.) Once you have the equations, click Run in Channel Operations and name the new EEGset **6_N400_LmRef_LmRm.** The waveforms should look only slightly different than the waveforms before re-referencing.

If you did it the same way I did, you'll still have a channel labeled **Rm**, but now it's the voltage between Rm and the average of Lm and Rm. It's not a very useful channel, but it doesn't hurt to keep it. Alternatively, you could keep the original signal, with Rm referenced to Lm. Or you could just eliminate that channel. Because you specify the equations used by **Channel Operations**, you have complete control!

Now select **6_N400_CzRef** in the **EEGsets** panel so that we can re-reference it to the average of the left and right mastoids. This is pretty simple: We just need to subtract the average of the two mastoids from each channel. Again, I'm not going to tell you how to do it. Don't forget to look at the **Channel and IC Selection** panel to figure out the channel numbers for the Lm and Rm signals. (If you get stuck, the equations are in a file named **Re-Reference_Cz.txt**.) You can name the resulting EEGset **6_N400_CzRef_LmRm**. The average of Lm and Rm is pretty different from Cz, so this change in reference creates a clearly visible difference in the waveforms.

Where did these datasets come from?

I don't actually have files for the N400 experiment that were recorded using Lm or Cz as the reference, so I created approximations by applying Channel Operations to the single-ended data. I simply relabeled the P9 and P10 channels as **Lm** and **Rm**. That is, we're just going to pretend that P9 was actually Lm and P10 was actually Rm. In **6_N400_LmRef.set**, we'll pretend that the data were referenced to Lm (even though I actually referenced the data to P9). This file contains a channel labeled **Rm**, which we're pretending is the voltage between Rm and Lm (but is actually the voltage between P10 and P9). In **6_N400_CzRef.set**, I simply referenced the data to Cz (including channels labeled **Lm** and **Rm** that actually have the data from P9 and P10, referenced to Cz).

This page titled 5.11: Exercise- Other Common Re-Referencing Scenarios is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 5.11: Exercise- Other Common Re-Referencing Scenarios by Steven J Luck is licensed CC BY 4.0.

5.12: Matlab Script For This Chapter

I've provided a script called **referencing_examples.m** in the Chapter_5 folder. This script implements the EEG referencing/re-referencing procedures from the last few exercises. It shows how you can store the equations in text files, which is quite convenient.

This page titled 5.12: Matlab Script For This Chapter is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 5.12: Matlab Script For This Chapter by Steven J Luck is licensed CC BY 4.0.

5.13: Key Takeaways and References

Key Takeaways

- Voltage is the potential for charges to flow from one location (the so-called active site) to another location (the so-called reference site). There is no such thing as the voltage at a single electrode site.
- When you're looking at data from a channel that is labeled with the so-called active electrode site for that channel (e.g., Pz), the waveforms are impacted equally by activity at the so-called active and so-called reference sites. If you don't know what the reference was, you really don't know what you're looking at. You should always find out what reference was used before you spend time looking at ERP waveforms.
- There is no "correct" reference location. No matter what you use, the data are impacted equally by the so-called reference and so-called active electrodes.
- The "best" reference is usually whatever is common in your subarea (because using the same reference facilitates comparisons across studies).
- To avoid falling into the trap of thinking that the waveform from a given channel is primarily a result of the so-called active electrode for that channel, it can be helpful to look at your data with multiple different reference sites. For this purpose, it is usually sufficient to re-reference the grand average ERPs.
- It is simple to re-reference data offline. In many cases, you can make one of your channels (or the average of a set of channels) the reference by simply subtracting it from the other channels (assuming that they all started with the same reference).
- You can avoid the reference issue altogether by converting your voltage waveforms into current density or global field power.

References

Dien, J. (1998). Issues in the application of the average reference: Review, critiques, and recommendations. *Behavior Research Methods, Instruments & Computers*, *30*, 34–43.

Dong, L., Li, F., Liu, Q., Wen, X., Lai, Y., Xu, P., & Yao, D. (2017). MATLAB Toolboxes for Reference Electrode Standardization Technique (REST) of Scalp EEG. *Frontiers in Neuroscience*, *11*. https://doi.org/10.3389/fnins.2017.00601

Files, B. T., Lawhern, V. J., Ries, A. J., & Marathe, A. R. (2016). A Permutation Test for Unbalanced Paired Comparisons of Global Field Power. *Brain Topography*, *29*, 345–357. https://doi.org/10.1007/s10548-016-0477-3

Hamburger, H. L., & Van der Burgt, M. A. G. (1991). Global Field Power measurement versus classical method in the determination of the latency of evoked potential components. *Brain Topography*, *3*(3), 391–396. https://doi.org/10.1007/BF01129642

Kappenman, E. S., Farrens, J. L., Zhang, W., Stewart, A. X., & Luck, S. J. (2021). ERP CORE: An Open Resource for Human Event-Related Potential Research. *NeuroImage*, *225*, 117465. https://doi.org/10.1016/j.neuroimage.2020.117465

Kiat, J. E., Hayes, T. R., Henderson, J. M., & Luck, S. J. (2022). Rapid extraction of the spatial distribution of physical saliency and semantic informativeness from natural scenes in the human brain. Journal of Neuroscience, 42, 97–108. https://doi.org/10.1523/JNEUROSCI.0602-21.2021

Luck, S. J. (2014). *An Introduction to the Event-Related Potential Technique, Second Edition*. MIT Press.

Nunez, P. L. (1981). *Electric Fields of the Brain*. Oxford University Press.

Rossion, B., & Jacques, C. (2012). The N170: Understanding the time course of face perception in the human brain. In S. J. Luck & E. S. Kappenman (Eds.), *The Oxford Handbook of Event-Related Potential Components* (pp. 115–141). Oxford University Press.

Skrandies, W. (1989). Data reduction of multichannel fields: Global field power and Principal Component Analysis. *Brain Topography*, *2*(1), 73–80. https://doi.org/10.1007/BF01128845

Yao, D. (2017). Is the Surface Potential Integral of a Dipole in a Volume Conductor Always Zero? A Cloud Over the Average Reference of EEG and ERP. *Brain Topography*, *30*(2), 161–171. https://doi.org/10.1007/s10548-016-0543-x

This page titled 5.13: Key Takeaways and References is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 5.13: Key Takeaways and References by Steven J Luck is licensed CC BY 4.0.

6: ASSIGNING EVENTS TO BINS, AVERAGING, BASELINE CORRECTION, AND ASSESSING DATA QUALITY

Learning Objectives

In this chapter, you will learn to:

- Design experiments that avoid subtle sensory confounds by following the Hillyard Principle
- Create bin descriptor files for assigning events to bins
- Predict how the signal-to-noise ratio will change as you change the number of trials being averaged together
- Assess how the number of trials and the baseline period impact the data quality as quantified with the standardized measurement error (SME)
- Create response-locked as well as stimulus-locked averages
- Compare the ERPs for correct trials and error trials
- Conduct sequential analyses
- Evaluate the impact of overlapping activity from the previous trial
- Implement a sequential analysis using a script

This chapter takes a close look at how events are assigned to bins. This is not something that gets discussed a lot in the ERP literature, even in methodology papers, but it's absolutely fundamental. After all, "event" is the first word in "event-related potential." Also, ERPLAB's tool for assigning events to bins (BINLISTER) is fairly powerful but also not very user-friendly, so we'll want to make sure you have a good grasp on how it works.

ERPLAB's online documentation contains a detailed manual page on BINLISTER, and you should read it if you want to learn all the details of how this important routine works. I won't repeat all those details here. Given that we already have a good manual page on BINLISTER, I even wondered if it was worthwhile writing a chapter on bin assignment for this book. But I decided that the best way to learn about something like this is to actually use it. And once I started creating the exercises, I realized that they brought up some concepts about ERP data analysis that are important even if you end up using a different software package for analyzing your data.

This chapter will focus on the visual oddball P3b experiment from the ERP CORE, including an analysis of sequential effects and an analysis of the error-related negativity produced on error trials. The oddball paradigm has probably been used more than any other ERP paradigm over the years, so it's good to have a thorough understanding of it even if you don't plan to use it in your own research. The particular version of this paradigm that we implemented for the ERP CORE contains some subtleties that are useful for learning about the design of ERP experiments. And the oddball paradigm is particularly well suited for exploring some of the issues that come up in assigning events to bins.

We'll start by taking a close look at the details of the experimental paradigm. Then we'll perform several different analyses of one participant's data so you can see some of the different ways that events can be assigned to bins and the issues that arise in this fundamental step. Along the way, we'll see some important issues that arise in averaging, including overlap from previous trials. We'll also take a closer look at the baseline correction procedure, which seems simple but can end up creating problems in some situations. We'll also look carefully at how data quality varies according to the number of trials being averaged together and according to the specific time period being used for baseline correction.

6.1: Data for This Chapter

6.2: Design of the ERP CORE Visual Oddball P3b Experiment

6.3: The Event Code Scheme

6.4: Overview of Bin Descriptor Files

6.5: Exercise - A Basic Assignment of Events to Bins

6.6: Exercise - Looking at the Averaged ERPs

6.7: Exercise - The Signal-to-Noise Ratio
6.8: Exercise - Response-Locked Averaging
6.9: Exercise - Comparing Correct and Error Trials to See the Error-Related Negativity
6.10: Exercise - Sequential Analysis of the P3b
6.11: Exercise - Combining Bins
6.12: Exercise - Overlap
6.13: Matlab Script For This Chapter
6.14: Key Takeaways and References

This page titled 6: Assigning Events to Bins, Averaging, Baseline Correction, and Assessing Data Quality is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 6: Assigning Events to Bins, Averaging, Baseline Correction, and Assessing Data Quality by Steven J Luck is licensed CC BY 4.0.

6.1: Data for This Chapter

The data we'll be using for the exercises in this chapter can be found in the Chapter_6 folder in the master folder: https://doi.org/10.18115/D5G045.

We'll focus on the data from one participant (Subject 12), but the data from all participants who were included in the final analysis are also provided in a folder named **Data**. To keep things simple, some preprocessing steps have already been applied to the data provided here (as will be described later).

Also, whereas the data in the previous chapters were downsampled from the original sampling rate of 1024 Hz to 200 Hz, the data for this chapter were downsampled to 256 Hz, as in the published data. This creates some slight complications, but those complications will be informative.

This page titled 6.1: Data for This Chapter is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 6.1: Data for This Chapter by Steven J Luck is licensed CC BY 4.0.

6.2: Design of the ERP CORE Visual Oddball P3b Experiment

In a typical oddball experiment, there are two categories of stimuli, one of which is rare (called the *oddballs* or *targets*) and one of which is frequent (called the *standards* or *nontargets*). For example, you might present a sequence of Xs and Os in the middle of a video monitor, with 10% Xs and 90% Os. The Rare/Oddball category will elicit a much larger P3b component than the Frequent/Standard category, but only if participants are actively discriminating between the two categories. For example, if participants are asked only to judge the colors of the Xs and Os, you won't see a larger P3 to the Rare Xs than to the Frequent Os. However, if participants are instructed to press one button for Xs and another button for Os, this requires making an active categorization of each stimulus as an X or an O, and you'll see a much larger P3b for the Xs than for the Os. However, it's not the motor response per se that leads to the P3b. The Rare Xs will also produce a larger P3b than the Frequent Os if the task is to press a button for the Frequent Os and make no response for the Rare Xs.

Researchers have known since the 1960s that the amplitude of the P3b is inversely related to the probability of the eliciting stimulus. That is, the P3b will be larger if the oddballs are 20% probable than if they are 30% probable, and the P3b will be even larger if the oddballs are 10% probable (Duncan-Johnson & Donchin, 1977). A fundamentally important but not widely appreciated fact is that it is not the probability of the physical stimulus that determines P3b amplitude, but instead the probability of what I call the *task-defined category*. For example, my lab once ran an oddball experiment in which 15% of the stimuli were the letter E, and the other 85% were randomly selected from all the non-E letters (Vogel et al., 1998). The task was to press one button for the letter E and another button for non-E letters. The letter E was more common than any other individual letter, but the task required participants to categorize each stimulus as E or non-E, and the E category was less frequent than the non-E category. As a result, the E stimuli elicited a much larger P3b than the non-E stimuli. You can learn more about the P3b component in Chapter 3 of Luck (2014) or in John Polich's chapter on the P3 family of components in the Oxford Handbook of ERP Components (Polich, 2012).

Many oddball experiments contain an obvious confound: If 10% of the stimuli are Xs and 90% are Os, then the Xs and Os differ both in the shape of the letter and the probability of occurrence. The difference in shape probably doesn't have much impact on the P3b component, but confounds like this are easy to avoid, so I'm always surprised that so many experiments have this confound. An easy way to solve this is to counterbalance the probabilities: Use 10% Xs and 90% Os for half the trial blocks and 90% Xs and 10% Os for the other half. This makes it possible to compare the ERP elicited by an X when it is Rare to the ERP elicited by an X when it is Frequent. And we can do the same for the Os. In other words, we can hold the stimuli constant and vary only the probability. To make things simpler, we can average the X-Rare and O-Rare ERPs together and compare the result to the average of the X-Frequent and O-Frequent ERPs. Many experiments use this approach (but not as many as I would like!).

However, as illustrated in Figure 6.1, a subtle *adaptation* confound still remains when this counterbalancing is used. Imagine that X is Rare and O is Frequent in the first trial block, and participants press one button for the Xs and another for the Os. Neurons in visual cortex that are sensitive to the O shape will tend to become adapted (i.e., less responsive) as a result of the frequent occurrence of the O stimuli, but neurons that are sensitive to the X shape will not become adapted given the infrequent occurrence of this shape. As a result, the sensory response will be smaller for the O stimuli than for the X stimuli. In the second trial block, O is Rare and X is Frequent. Now the X-sensitive neurons become more adapted than the O-sensitive neurons, and the sensory response will be smaller for the Xs than for the Os. So, even though we have counterbalanced which stimulus is Rare and which is Frequent, we still get a larger sensory response for the Rare category than for the Frequent category.

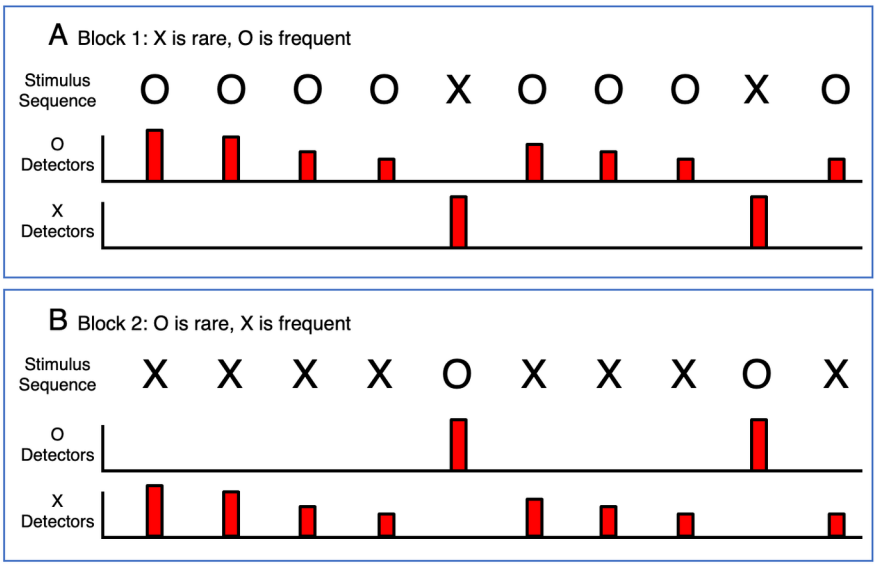


Figure 6.1. Example of the sensory adaptation confound that is present in nearly all oddball experiments. The height of each bar represents the magnitude of the sensory response to a given stimulus.

This exemplifies what I call the *Hillyard Principle* of experimental design: Keep the stimuli constant and vary only the psychological conditions. To follow this principle, you must keep the entire sequence constant across conditions, which we are not doing in the counterbalanced design shown in Figure 6.1. That is, we are using different sequences of stimuli to create our Rare and Frequent categories, and any differences in the ERPs for these categories could be a result of physical differences in the stimulus sequences rather than the psychological categories of Rare and Frequent.

The Hillyard Principle

The Hillyard Principle is named after my PhD advisor, Steve Hillyard. When I was a grad student, we were constantly reminded to keep the stimuli constant and vary the psychological conditions (usually by varying the instructions). Steve was a master of experimental design, and he had a huge impact on the field by developing extremely rigorous designs (and instilling this ethos into his many graduate students and postdocs).

The sensory ERP components are very sensitive to small differences in stimuli, and the Hillyard Principle is especially important when you see differences between conditions at relatively short latencies (e.g., <200 ms after stimulus onset). If an experiment does not follow the Hillyard Principle, it's usually impossible to interpret any early effects (unless, of course, the goal of the experiment was to examine the effects of stimulus manipulations on sensory activity). However, it's a good idea to follow the Hillyard Principle even when you're looking at later effects, because it's difficult to be 100% certain that a late effect isn't a consequence of an early sensory confound.

Some experimental questions are difficult to answer while following the Hillyard principle. For example, imagine that you wanted to compare the ERPs elicited by nouns and verbs (presented as auditory speech signals). You don't have any control over what nouns and verbs sound like, and it would be difficult to create instructions that make a participant treat the word "chair" as a noun in some trial blocks and a verb in other blocks. But if you were really motivated, you could actually achieve this kind of control by using two groups of participants who spoke different languages. For example, if you compared monolingual English speakers and monolingual Mandarin speakers, you could ask whether nouns and verbs that are known by an individual produce a difference that is not present for nouns and verbs that are unknown by that individual.

In the ERP CORE, we designed the P3b experiment to follow the Hillyard principle. As illustrated in Figure 6.2, the stimuli were the letters A, B, C, D, and E, presented in random order in the center of the video display. Each of these letters was 20% probable. Each participant received 5 trial blocks, each containing 40 trials, and a different letter was designated the target in each block. For example, a participant might be told that D was the target in the first block, A was the target in the second block, and so on. The participants were instructed to press one button for the target letter and another button for any of the four nontarget letters. For example, when D was designated the target, they would press one button for D and a different button for A, B, C, or E. As a result, D was in the Rare category when it was designated the target and was in the Frequent category when one of the other four

letters was designated the target. Thus, we followed the Hillyard Principle: we kept the sequence of stimuli constant and varied only the task instruction.

What it Means to Keep the Sequence Constant

If we had used the same exact sequence of letters in each trial block, it is possible that participants would have learned the sequence. We therefore created a new randomized sequence for each block, but these sequences were created using the same rules and differed only randomly. That rules out any **systematic** differences in the physical stimuli between trial blocks.

The bottom portion of Figure 6.2 shows the grand average ERP waveforms for the Rare and Frequent stimulus categories (from the Pz channel, with the average of P9 and P10 as the reference). The Rare waveform contains equal numbers of trials for which A, B, C, D, and E were designated the target. The Frequent waveform also contains equal numbers of trials for which A, B, C, D, and E were designated the target. Thus, the larger P3b observed for the Rare category than for the Frequent category must reflect the rareness of the task-defined category, not the rareness of the physical stimuli. It took a lot of thought and effort to design the experiment this way, but I really enjoy the process of designing experiments, especially when I need to come up with some kind of creative "trick" to rule out potential confounds.

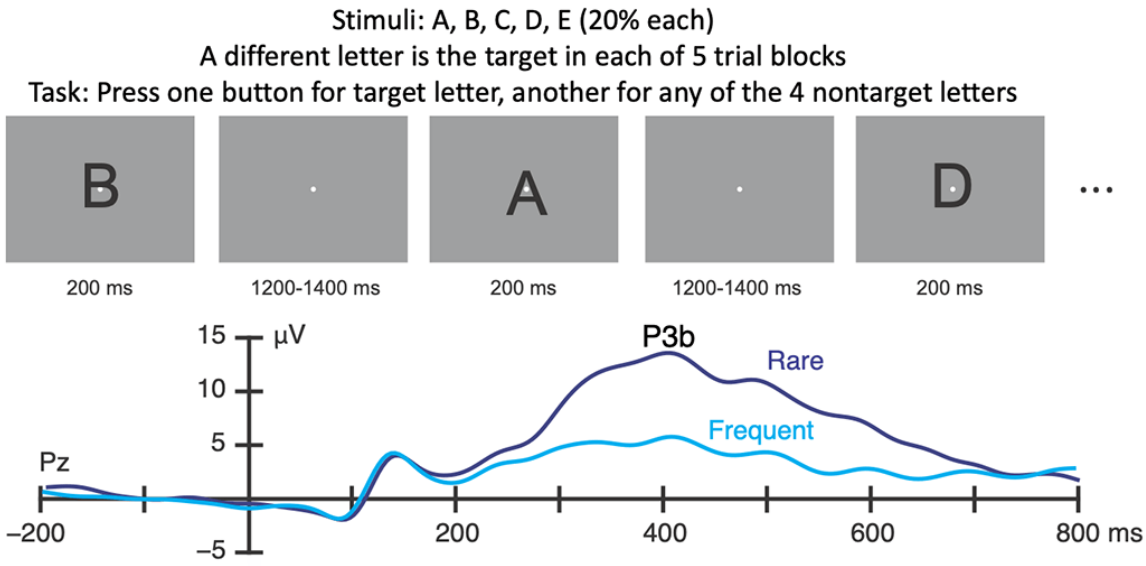


Figure 6.2. Experimental paradigm and grand average ERP waveforms from the ERP CORE visual oddball P3b experiment.

This page titled 6.2: Design of the ERP CORE Visual Oddball P3b Experiment is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 6.2: Design of the ERP CORE Visual Oddball P3b Experiment by Steven J Luck is licensed CC BY 4.0.

6.3: The Event Code Scheme

In the ERP CORE P3b experiment, there were many different combinations of stimuli (5 different letters) and task instructions (which of the 5 letters was the target). In complicated experiments like this, some care is needed to come up with a scheme for representing each possible stimulus/task combination as a separate event code using integers between 1 and 255. Table 6.1 shows our scheme. We used the tens place in the event code to indicate which letter was the target and the ones place to indicate what the current letter was. For example, event code 23 indicates that B was the target letter in the current block (because 23 has a 2 in the tens place) and that the letter C was presented on the current trial (because 23 has a 3 in the ones place). In this scheme, the Rare category is coded as event codes 11, 22, 33, 44, and 55, and the Frequent category is coded as the other combinations. We also used 201 and 202 as the event codes that were generated when the participant pressed a response button, with 201 for a correct response and 202 for an incorrect response.

Table 6.1. Event code scheme for the ERP CORE visual oddball P3b experiment.

Event Code	Target Letter for Current Block	Tens place	Current Stimulus	Ones place
11	A	1	A	1
21	B	2	A	1
31	C	3	A	1
41	D	4	A	1
51	E	5	A	1
12	A	1	B	2
22	B	2	B	2
32	C	3	B	2
42	D	4	B	2
52	E	5	B	2
13	A	1	C	3

Event Code	Target Letter for Current Block	Tens place	Current Stimulus	Ones place
23	B	2	C	3
33	C	3	C	3
43	D	4	C	3
53	E	5	C	3
14	A	1	D	4
24	B	2	D	4
34	C	3	D	4
44	D	4	D	4
54	E	5	D	4
15	A	1	E	5
25	B	2	E	5
35	C	3	E	5
45	D	4	E	5
55	E	5	E	5

This page titled 6.3: The Event Code Scheme is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 6.3: The Event Code Scheme by Steven J Luck is licensed CC BY 4.0.

6.4: Overview of Bin Descriptor Files

Now let's see how event codes are assigned to bins with BINLISTER, creating one bin for the Rare category and another for the Frequent category (collapsed across which specific letter was presented). We'll exclude trials in which the buttonpress response was incorrect or was unreasonably fast or slow.

As you'll recall from Chapter 2, a *bin descriptor file* is used to tell BINLISTER how event codes should be assigned to bins. In the Chapter_6 folder, you'll find a bin descriptor file named **BDF_P3.txt**, which we'll use for this exercise. Make sure that the Chapter_6 folder is the current folder in Matlab, and double-click the **BDF_P3.txt** file from the Current Folder pane in Matlab to open it. Here's what you should see:

bin 1
Rare, Correct
.{11;22;33;44;55}{t<200-1000>201}

bin 2
Frequent, Correct
.{12;13;14;15;21;23;24;25;31;32;34;35;41;42;43;45;51;52;53;54}{t<200-1000>201}

Each bin is described by a set of three lines. The first is the bin number (which must be in consecutive order, beginning with 1). The second line is the label for the bin (which can be anything you like). The third line is the actual *bin descriptor*, which tells BINLISTER what event code(s) belong in a bin. A really simple bin descriptor would be **.{1}**, which would mean that any occurrence of event code 1 would be included in the bin. A slightly more complicated bin descriptor would be **.{1;21}**, which would mean that any occurrence of event code 1 or event code 21 would be included in the bin. Note that the list of event codes is placed inside a set of curly brackets ("{}") to create an *event list*.

You can specify that a sequence of events is needed by having more than one event list in the event descriptor. For example, the bin descriptor **.{1;21}{2;22}** would indicate that an event code of 1 or 21 must be followed by an event code of 2 or 22. The fact that the period is immediately to the left of {1;21} means that the 1 or the 21 will be the time-locking event for averaging. The event descriptor **{1;21}.{2;22}** would indicate that 2 or 22 is the time-locking event and must be preceded by 1 or 21.

In the example shown above, event codes 11, 22, 33, 44, and 55 will serve as the time-locking event for Bin 1. Given how the event codes were defined in Table 6.1 in the previous section, this means that Bin 1 includes event codes corresponding to an A stimulus when A was the target, B when B was the target, C when C was the target, D when D was the target, or E when E was the target. We could have instead created a separate bin for each of these five target letters and then combined the five bins after averaging using **ERP Bin Operations**. However, it was simpler to combine them at the BINLISTER stage. The event list for Bin 2 contains all the event codes for the nontarget letters. You can verify this by comparing the event descriptors with the list of event codes in Table 6.1.

In the bin descriptor for Bin 1, we require that the event code for the target (11, 22, 33, 44, or 55) is followed by the event code for a correct response (201). The response event is listed as **t<200-1000>201**, which means that the event code 201 must occur 200 to 1000 ms after the time-locking event. This ensures that the response time (RT) wasn't so fast that it was a fast guess (an RT of <200 ms) or so slow that it reflects poor attention (an RT of >1000 ms). We call this a *time-conditioned event list*.

Note that if we didn't use a time-conditioned event list and instead used **.{11;22;33;44;55}{201}** as the event descriptor, the response event code (201) would need to directly follow the stimulus event code, with no other event codes between. However, by using a time-conditioned event list to specify that the 201 must be between 200 and 1000 ms after the stimulus event code, other event codes may occur between the stimulus and the response.

Also, if a time-conditioned event list appears prior to the time-locking event, time flows backward from the time-locking event. For example, if you specify **{t<200-800>15}.{100}**, BINLISTER will search for an event code of 100 preceded by an event code of 15 that occurred 200-800 ms prior to the 100. Additional details can be found in the BINLISTER documentation.

Should you exclude trials with incorrect behavioral responses?

By specifying that the stimulus event code needed to be followed by the event code for a correct response, our bins excluded trials with an incorrect behavioral response. Some studies exclude error trials in this manner, but others do not. Here's the general principle I use to decide whether trials with an incorrect response should be excluded: If the errors in a given task are likely the result of lapses of attention, then exclude the error trials; if the errors in a given task mainly occur because the task is very difficult, then don't exclude the error trials.

For example, errors in most oddball paradigms are typically a result of lapses of attention, so we exclude the error trials. Indeed, the ERPs are quite different on error and correct trials in the oddball paradigm (Falkenstein et al., 1990). By contrast, in tasks using the change detection paradigm to study visual working memory, most errors occur because of limits in storage capacity (Luck & Vogel, 2013). As a result, brain activity is similar on correct and error trials in these tasks (Luria et al., 2016), so we don't exclude the errors (which would reduce the number of trials per bin quite a lot, decreasing the signal-to-noise ratio).

This page titled 6.4: Overview of Bin Descriptor Files is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 6.4: Overview of Bin Descriptor Files by Steven J Luck is licensed CC BY 4.0.

6.5: Exercise - A Basic Assignment of Events to Bins

Enough theory—let's make some bins! To make sure there aren't any unwanted settings left over from the previous exercise, start by clicking the **Reset** button under the lower right corner of the plot region and then clicking **Reset ERPLAB Studio** in the window that appears. Make sure that **Chapter_6** is Matlab's current folder. Using the **EEGsets** panel, load the EEGset named **12_P3_corrected.set**. Scroll through the EEG and familiarize yourself with it. For example, take a look at the sequence of event codes and match them with the codes in Table 6.1. When I load an EEGset or ERPset, I always look at the waveforms before I do anything else. Many times, this visual inspection has made me realize that I had the wrong data or that there was something about the data that was incompatible with what I was planning to do next.

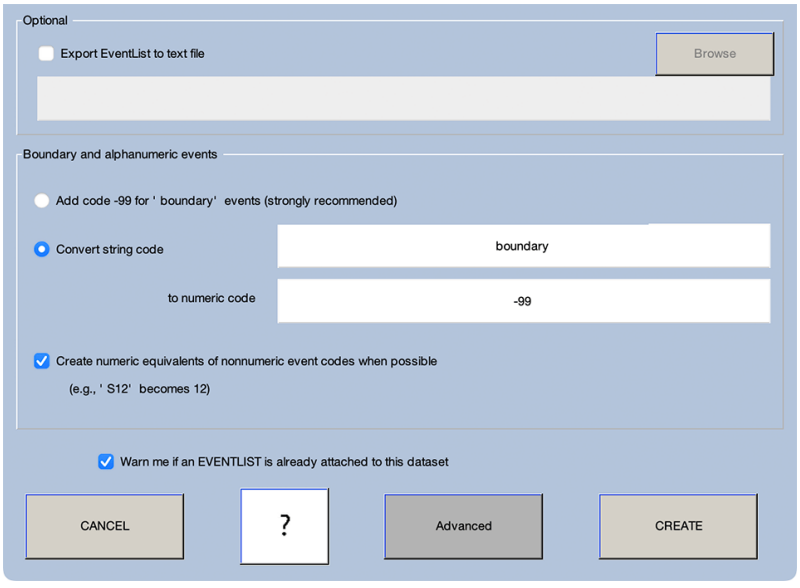
Do you see any eyeblinks in the VEOG-lower or FP1/FP2 channels? Did you see much low-frequency drift in the EEG? The answer is "no" for both questions. This gives you some clues about the preprocessing steps that have already been applied to this dataset. What operations do you think have already been applied?

To keep the exercises in this chapter simple, I have already applied an *artifact correction* procedure to the EEGsets. Instead of excluding epochs that contain blinks and eye movements, the voltages for the blinks and eye movements have been estimated and subtracted from the waveforms. That way, we won't need to throw out any trials in the exercises in this chapter. This is why the filename for the dataset has **_corrected** in it. A high-pass filter (half-amplitude cutoff at 0.1 Hz) has also been applied to remove slow drifts.

Creating an EventList

Before we can assign events to bins, we need to create a data structure called an EventList and attach it to the EEG structure. EEGLAB stores event codes in a somewhat complicated fashion, and the EventList makes the event codes much easier to see and manipulate.

To create the EventList, go to the **EventList** panel and click the **Create** button. You should see a window like the one shown in Screenshot 6.1. There are several options in the window, and a whole lot more become available if you click the **Advanced** button, but most users can just use the default options. If you want to know more about the EventList structure, you can read this page in the ERPLAB Studio Manual, and if you want to know more about the available options for creating an EventList, you can read this page.

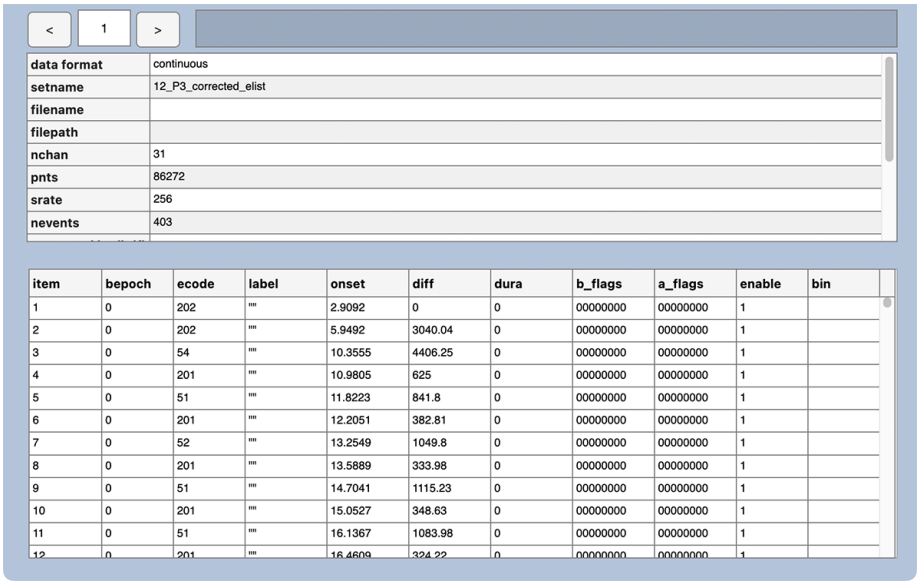


Screenshot 6.1

Go ahead and click the **Create** button inside the Create EventList window. This will create the EventList from the event code

information inside the current EEGset and attach it to the EEGset, creating a new EEGset. You can use the automatically suggested name for this new EEGset, **12_P3_corrected_elist**, and you should save it to disk by checking the **Save EEGsets to disk** and **Use EEG name as filename** options (because you will need this file for a later exercise). Note that the EventList panel has a little table showing the number of occurrences of each event code. That can be useful for verifying that your stimulus presentation program is working the way you expect.

Now click the **View** button in the **EventList** panel to see the EventList you just created. You should see a new window like the one shown in Screenshot 6.2. The top portion of the window provides "header" information, such as the name of the EEGset (**setname**), the number of channels (**nchan**), and the sampling rate (**srate**). The bottom portion shows the list of event codes. The key columns are **item** (the item number for each event), **ecode** (the event code for that event), **onset** (the time of the event code relative to the onset of the recording, in seconds), **diff** (the amount of time between the current event code and the previous event code, in milliseconds), and **bin** (the bins that this event code has been assigned to). You can see that the first two items are event code 202 (incorrect response); these were automatically generated by the stimulus presentation script and can be ignored. Item 3 is the event code for the first stimulus in the trial block. It has an event code 54, which is the letter D when E was the target according to Table 6.1 in Section 6.3. Item 4 has event code 201, which is the event code for a correct response. The **diff** column tells us that the amount of time between this response and the preceding stimulus was 625 ms.



< 1 >

data format	continuous
setname	12_P3_corrected_elist
filename	
filepath	
nchan	31
pnts	86272
srate	256
nevents	403

item	bepoch	ecode	label	onset	diff	dura	b_flags	a_flags	enable	bin
1	0	202	""	2.9092	0	0	00000000	00000000	1	
2	0	202	""	5.9492	3040.04	0	00000000	00000000	1	
3	0	54	""	10.3555	4406.25	0	00000000	00000000	1	
4	0	201	""	10.9805	625	0	00000000	00000000	1	
5	0	51	""	11.8223	841.8	0	00000000	00000000	1	
6	0	201	""	12.2051	382.81	0	00000000	00000000	1	
7	0	52	""	13.2549	1049.8	0	00000000	00000000	1	
8	0	201	""	13.5889	333.98	0	00000000	00000000	1	
9	0	51	""	14.7041	1115.23	0	00000000	00000000	1	
10	0	201	""	15.0527	348.63	0	00000000	00000000	1	
11	0	51	""	16.1367	1083.98	0	00000000	00000000	1	
12	0	201	""	16.4609	324.22	0	00000000	00000000	1	

Screenshot 6.2

If you scroll through the list of events, you'll see that item 45 was event code 55 (the letter E when E was the target), and it was followed by event code 202 (an incorrect response). Once we assign event codes to bins, item 45 should not be included in the bin for targets, because we are including only stimuli that are followed by a correct response. If you scroll further, you'll see that item 121 was event code 23 (the letter B when C was the target), and it was followed by a correct response (event code 201). However, the **diff** column for the response indicates that it occurred 173.83 ms after the stimulus, which is too fast for a controlled behavior. Consequently, item 121 will be excluded from the bin for the nontargets.

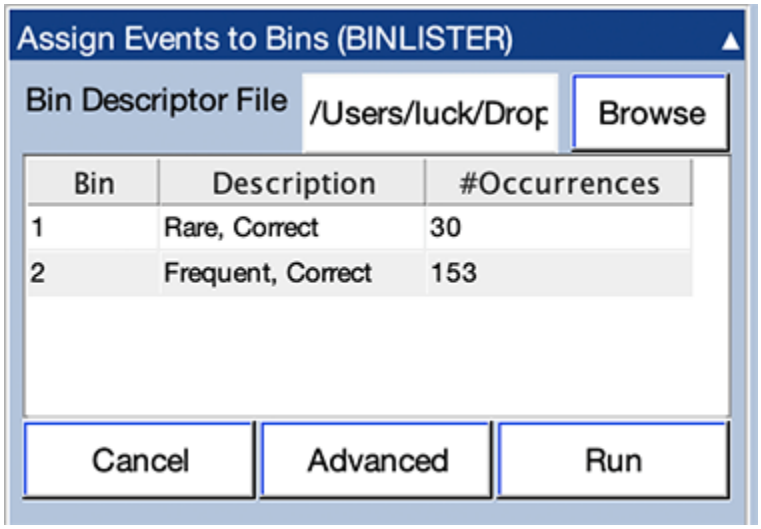
You may now close the window showing the EventList.

Assigning Events to Bins

Now let's assign the events to bins. Go to the **Assign Events to Bins (BINLISTER)** panel and use the **Browse** button to select the bin descriptor file (**BDF_P3.txt**). There are a bunch of options that you could access by clicking the **Advanced** button, but you won't ordinarily need any of them. If you're interested, you can read about them in the manual page for this panel. Just click **Run** to perform the BINLISTER process, and give the new EEGset the suggested name (**12_P3_corrected_elist_bins**).

You can now see the number of trials that were assigned to each bin in the panel (see Screenshot 6.3). There were 30 events that matched the bin descriptor for Bin 1, and 153 that matched the bin descriptor for Bin 2. How many should we have had? In the task description provided at the beginning of the chapter, you learned that there were 5 equiprobable letters, 5 blocks of trials

(one with each letter as the target), and 40 trials per block. This gives us 200 total trials. Given that 1 of the 5 letters was the target in each block, the target probability was 20%. This means that we should have had 40 targets and 160 nontargets over the course of a session. Why, then, do we only have 30 instances of Bin 1 and 153 instances of Bin 2?



Screenshot 6.3

The answer is that our bin descriptors for Bins 1 and 2 specified that a given stimulus event code must be followed by a correct response within 200-1000 ms, and the response was incorrect, too fast, or too slow on some of the trials. These trials were not assigned to any bin. (Later, we will do an exercise where we have separate bins for correct and incorrect trials.)

To see that everything worked correctly, we need to look at the EventList, because the BINLISTER process simply updates the EventList to add bin numbers to any events that matched a bin description. Go to the **EventList** panel and click the **View** button to pop up a window showing the updated EventList. Note that the **bin** column, which was empty before, now has a 1 or a 2 for many of the events. If you look more closely, you'll see that most of the stimulus events have a 1 or a 2 in the **bin** column, but none of the response events have anything in the **bin** column. This is because we are using stimuli rather than responses as the time-locking events in this analysis. Later, you'll see an example of how to make response-locked averages.

Most of the stimulus events are assigned to Bin 2 (the Frequent nontarget category), because 80% of the stimuli were in this category. But if you scroll down, you'll see that item 25 was a target (event code 55, which was the letter E when E was the target) and is assigned to Bin 1. Items 43 and 45 were also targets, but they were not assigned to any bin. This is because each of them was followed by an event code 202 (indicating an incorrect response). If you scroll down to item 121, you will see that it was a nontarget stimulus, and it was followed by a correct response, but the response was faster than 200 ms (i.e., the **diff** column for the response event had a value of 173.53). Thus, event 121 was not assigned to a bin.

After you're finished looking at the EventList, you can close this window.

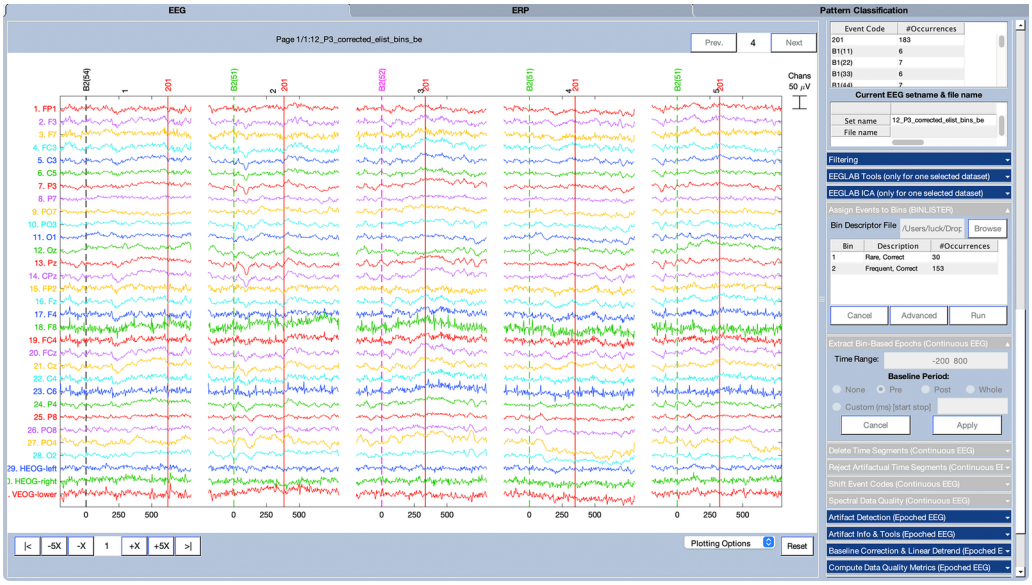
This page titled 6.5: Exercise - A Basic Assignment of Events to Bins is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 6.5: Exercise - A Basic Assignment of Events to Bins by Steven J Luck is licensed CC BY 4.0.

6.6: Exercise - Looking at the Averaged ERPs

Now let's make averaged ERPs for the bins we created. We still have continuous EEG data, so we first need to extract fixed-length epochs from the data, with the time-locking event code as time zero. Make sure that the final EEGset from the previous exercise is selected in the **EEGsets** panel (**12_P3_corrected_elist_bins**), and then go to the **Extract bin-based epochs** panel. Use the default epoch of **-200 800** and select **Pre** as the baseline period (which will use the prestimulus interval). Click **Apply** and keep the suggested name of **12_P3_corrected_elist_bins_be** for the new dataset.

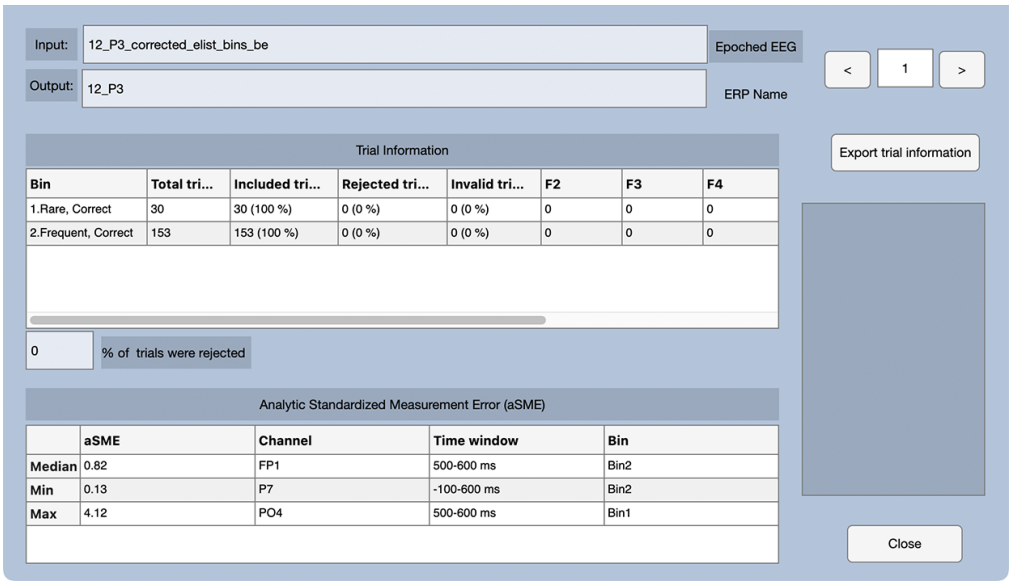
When you do this, two very noticeable things will happen. First, different panels will now be grayed out, because some panels operate only on epoched data and other panels operate only on continuous data. Second, the plotting region of the eStudio GUI will show 5 distinct data epochs instead of a single continuous set of waveforms (see Screenshot 6.4). Take a look at the epoched EEG data to make sure everything looks okay.



Screenshot 6.4

Now let's make averaged ERPs from these EEG epochs using the **Compute averaged ERPs** panel. You should be able to use the default settings and just click **RUN**. You'll then see the usual panel for naming and saving the resulting ERPset. Name it **12_P3**, and click the **Save ERPsets to disk** and **Use ERP name as filename** boxes so that the file will be named **12_P3.erp**.

Two things will then happen. First, the main eStudio GUI will switch from the **EEG** tab to the **ERP** tab, with **12_P3** in the ERPsets panel. Second, a window will appear summarizing the results of the averaging process, as shown in Screenshot 6.5. This window allows you to verify that the expected number of trials was averaged in each bin. It also provides a summary of the aSME metric of data quality. We'll say more about that later in this chapter.

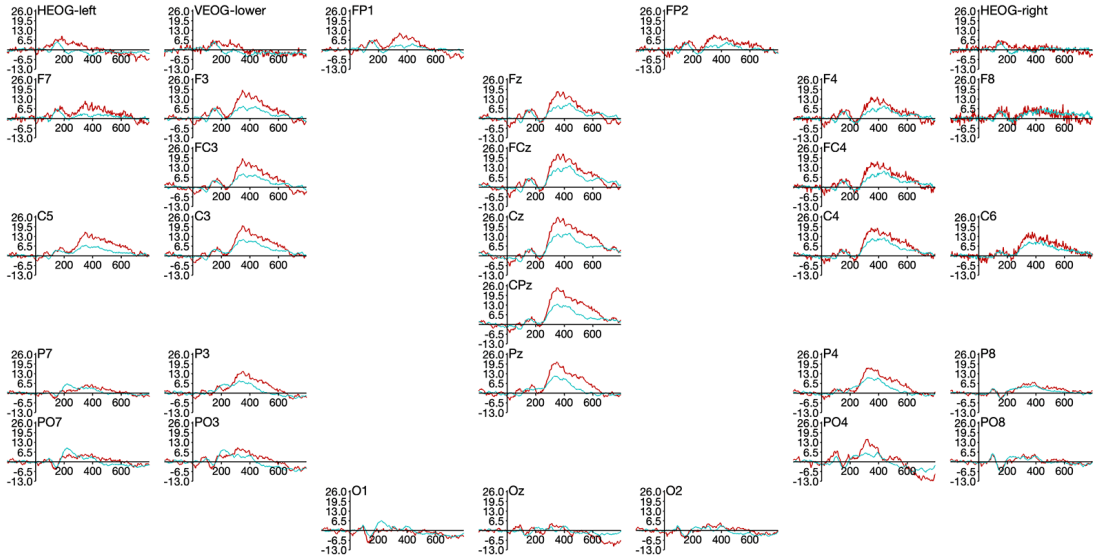


Screenshot 6.5

You can close that window and take a look at the averaged ERPs in the main eStudio GUI. To get a layout that better matches the scalp locations of the electrodes, go to the **Plot Settings** panel, change the **Grid Layout** to **Custom**, click the **Import** button, select the file named **GridLocations_CustomP3b.tsv**, and then click the **Apply** button. I've created the grid locations file for you, but you can read how to do it yourself in the ERPLAB Studio documentation.

You should now see something like Screenshot 6.6. It's a beautiful effect, with a larger P3b for the rare stimuli than for the frequent stimuli, with a maximum difference between conditions at the midline central and parietal sites.

Screenshot 6.6



But note that I intentionally chose a participant with beautiful prototypical effects for the exercises in this chapter. There are tremendous individual differences in ERP waveforms (largely due to nonfunctional differences in biophysical factors like skull thickness), and the ERPs for most of the participants in this study don't look this much like the grand average. Averaging (whether across trials or across participants) is often necessary, but the result is something of a fiction. As I like to say, "averaging hides a multitude of sins."

Although published papers often focus on just a few electrode sites, you should always look at the whole set of channels to verify that the scalp distribution is sensible. With a reference at or near the mastoids, the P3b effect (defined as the difference between Rare and Frequent) should be biggest along the midline near Cz and Pz. This participant's P3b is a little more frontal than is typical for a visual paradigm, and I would ordinarily expect a smaller effect at Fz and a larger effect at Oz. But given the large range of individual differences in ERPs, this participant's data look pretty normal.

You can also see quite a bit of high-frequency noise in the frontal and central channels over the right hemisphere (especially F8). The noise in F8 and surrounding channels is probably EMG from the temporalis muscle (the muscle near the temples that is used to contract the jaws). If you scroll through the original EEG data, you can also see the high-frequency noise in the single trials. My guess is that this participant was clenching their jaw just a little bit, mainly on the right side.

This page titled 6.6: Exercise - Looking at the Averaged ERPs is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 6.6: Exercise - Looking at the Averaged ERPs by Steven J Luck is licensed CC BY 4.0.

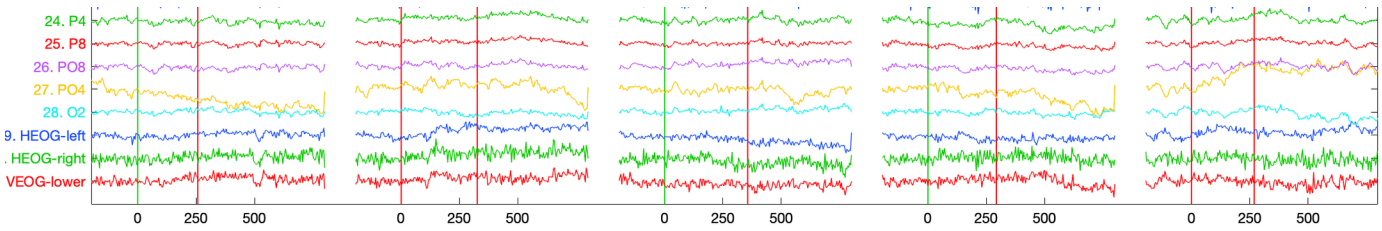
6.7: Exercise - The Signal-to-Noise Ratio

In the previous exercise, we saw that there was a lot of high-frequency noise in the right-hemisphere electrodes, especially F8. In the present exercise, we're going to quantify the impact of this high-frequency noise on the data quality and also see how the data quality depends on the number of trials being averaged together. We'll use the **12_P3** ERPset you created in the previous exercise. If you don't still have it loaded in the **ERPsets** panel, go ahead and load it now from the Chapter_6 folder.

Take a look at the analytic standardized measurement error (aSME) values for this ERPset by going to the **View Data Quality Metrics** panel and clicking **Show in a table**. There's a ton of information in the table, and I find that it helps to select the **Color heatmap** option. Interestingly, the **aSME** values from the F8 electrode are not particularly high. The worst values are at PO4.

This disconnect between the **aSME** values and the high-frequency noise you can see in the waveforms occurs because these **aSME** values tell you about the precision of the measurements of the mean voltage over 100-ms periods. As we discussed in Chapter 4, high-frequency noise has very little impact on the mean voltage over a 100-ms period, because the upward and downward noise deflections (the uppies and downies) cancel out. Low-frequency noise, by contrast, has a big impact on the mean voltage over a given time period. You can't easily see low-frequency noise in an averaged ERP, so let's go back to the **EEG** tab and take a look at the epoched EEG file from the previous exercise (**12_P3_corrected_elist_bins_be**; there's a copy in the Extra Files folder if you don't still have it loaded).

The PO4 channel should be a yellow waveform near the bottom of the display (see Screenshot 6.7). If you scroll through the epochs, you'll see that this channel often drifts upward or downward across an epoch, much more than the other channels. Screenshot 6.7 shows an example of epochs 105-109. This drift in the signal is why PO4 has the worst (largest) aSME value. If we were to quantify P3b amplitude as the mean voltage between 300 and 500 ms (as recommended by Kappenman et al., 2021), the random upward and downward low-frequency drifts in PO4 would mean that our P3b amplitude score could be quite far from the participant's true score for that channel (i.e., the score we would obtain with an infinite number of trials). This is what aSME represents: the extent to which the measured value is likely to differ randomly from the true value. By contrast, the high-frequency noise in F8 wouldn't have much impact on our ability to get the true value.



Screenshot 6.7

If we instead quantified P3b amplitude as the *peak* voltage between 300 and 500 ms, the high-frequency noise would be a bigger problem. Computing the standardized measurement error for peak amplitude is more complicated, so we're not going to look at it now, but if we did, I'm sure that the high-frequency noise in F8 would produce a large SME value. The take-home message is that the effect of noise on your ability to precisely measure the amplitude or latency of a given ERP component depends on both the nature of the noise (e.g., high-frequency versus low-frequency) and the nature of the method used to quantify the amplitude or latency (e.g., mean amplitude versus peak amplitude).

Now let's look at how the data quality differs between the Rare and Frequent averages. A standard idea in the ERP literature is that the signal-to-noise ratio (SNR) of an averaged ERP increases according to the square root of the number of trials (all else being equal). I have to admit that I didn't understand exactly what was meant by "signal" and "noise" in the signal-to-noise ratio until a few years ago, when we started developing the SME metric of data quality. The "signal" part of the signal-to-noise ratio is the true value of an amplitude score (i.e., the amplitude we would obtain with an infinite number of trials).

The noise is quantified as the *standard error* of the amplitude score. Let's unpack that. Imagine that our score is the mean voltage between 300 and 400 ms in the averaged ERP waveform. This is equivalent to taking the mean voltage between 300 and 400 ms on each individual trial and averaging them together. The standard error of this score can be estimated using the standard analytic formula for the standard error of the mean: SD ÷ sqrt(N). That is, we compute the mean voltage between 300 and 400 ms on each trial, take the standard deviation (SD) of these voltages, and divide by the square root of the number of trials.

Because the denominator is sqrt(N), this standard error gets smaller if we increase the number of trials according to sqrt(N). So, the denominator of the signal-to-noise ratio decreases according to the square root of the number of trials, causing the overall SNR to increase according to the square root of the number of trials.

In our P3b oddball experiment, 20% of the trials were oddballs, so there were 4 times as many Frequent trials as Rare trials. Consequently, sqrt(N) was twice as great for the Frequent condition as for the Rare condition (because sqrt(4) = 2). And this implies that the standard error should be half as large for the Frequent condition as for the Rare condition. The SME value is a generalized version of the standard error; it gives you the standard error for any amplitude or latency measure that is obtained from an averaged ERP waveform (see the box below for more details). When we are scoring the amplitude as the mean voltage during a specific time period, we can use the simple formula for the standard error [SD ÷ sqrt(N)] to estimate the SME, so we call this the *analytic standardized measurement error* or *aSME*.

Take a look at the aSME values for the Rare and Frequent conditions. You should see that the values are approximately half as large for the Frequent condition (Bin 2) as for the Rare condition (Bin 1) for any given combination of time period and electrode site. That is, the noise (quantified as the SME) is about half as big in the condition with four times as many trials. This is exactly what we would expect from the idea that the signal-to-noise ratio varies according to the square root of the number of trials.

The aSME is just an estimate, so we wouldn't expect it to be perfectly predicted by the number of trials in a finite data set. To get a more robust estimate, I made an average across all the time ranges and channels, and I found a mean of 0.617 for the Frequent condition and 1.372 for the Rare condition. This isn't quite a 1:2 ratio. But there's a good explanation for this: although there were 4 times as many Frequent trials as Rare trials in the experiment, we ended up excluding more Rare trials from the averages because of incorrect responses. Take a look at the actual number of trials in Bins 1 and 2—does the difference between the average aSME values for the Rare and Frequent trials make sense given the actual sqrt(N) for the Rare and Frequent trials?

Some details about the SME

With ERPLAB's default settings, the aSME values indicate the standard error of the amplitude scores that you'd get by quantifying the amplitude as the mean value in a set of consecutive 100-ms time periods. You can easily change the parameters (in the averaging step) to select other time intervals. Ordinarily, we would measure the P3b as the mean amplitude between 300 and 500 ms. If you're interested, you can have ERPLAB estimate the SME values for this time range by re-averaging the data and selecting **custom parameters** in the **Data Quality Quantification** section of the averaging routine.

ERPLAB's default SME values are estimated using the *analytic* formula for the standard error of the mean [SD ÷ sqrt(N)], so we call these *analytic SME* (aSME) values. This formula can't be used to estimate the SME for other amplitude or latency scores (e.g., peak amplitude or peak latency), and something called *bootstrapping* is used instead. This is more complicated and currently requires scripting. This is described in our original paper on the SME (Luck et al., 2021), and we have provided example scripts for computing bootstrapped SME values (https://doi.org/10.18115/D58G91).

This page titled 6.7: Exercise - The Signal-to-Noise Ratio is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 6.7: Exercise - The Signal-to-Noise Ratio by Steven J Luck is licensed CC BY 4.0.
- 6.6: Exercise - Looking at the Averaged ERPs by Steven J Luck is licensed CC BY 4.0.

6.8: Exercise - Response-Locked Averaging

In this exercise, we're going to create response-locked averages rather than stimulus-locked averages. That is, when we create our EEG epochs, the event code for the response will be time zero.

An important feature of BINLISTER is that it does not make a distinction between stimulus events and response events. An event code is an event code, no matter whether it represents a stimulus, a response, an eye movement, a heartbeat, or a sudden change in the FTSE stock index. Any event code can be the time-locking event (time zero). And a bin can be defined by any sequence of event codes.

Open the bin descriptor file named **BDF_P3_Response_Locked.txt** in the Matlab text editor (by double-clicking it in the Current Folder pane). Here's what you should see:

bin 1
Rare, Correct
{t<200-1000>11;22;33;44;55}.{201}

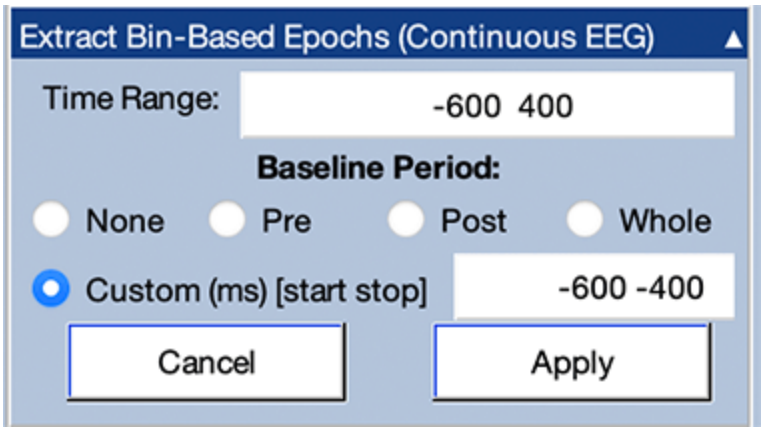
bin 2
Frequent, Correct
{t<200-1000>12;13;14;15;21;23;24;25;31;32;34;35;41;42;43;45;51;52;53;54}.{201}

It's a lot like the original bin descriptor file (**BDF_P3.txt**). The most important change is that the period symbol in each bin descriptor is to the left of the response event list rather than being to the left of the stimulus event list. The period goes to the left of the time-locking event, so this is how we indicate that the response event code (201) will be the time-locking event. Also, the **t<200-1000>** part now appears as part of the stimulus event list. The timing is always given relative to the time-locking event, which is now the response, so we need to specify that the stimulus must be 200-1000 ms prior to the response rather than the response being 200-1000 ms after the stimulus.

Start by clicking the **Reset** button under the lower right corner of the plot region and then clicking **Reset ERPLAB Studio** in the window that appears. Then load **12_P3_corrected_elist.set** (which you created in an earlier exercise) using the **EEGsets** panel. Then go to the **Assign Events to Bins (BINLISTER)**, load **BDF_P3_Response_Locked.txt** as the bin descriptor file, and click **Run**. You can name the new EEGset **12_P3_corrected_elist_resplock**.

Now go to the **EventList** panel and click the **View** button to look at the updated EventList. If you look at the **bin** column on the far right, you'll now see bin numbers on the lines for the response events (event code 201), with a 1 if the response is preceded by a target event code (11, 22, 33, 44, or 55) and a 2 if the response is preceded by a nontarget event code. This shows us that the responses will be used as time zero when we epoch and then average the data.

Close the EventList window and then epoch the data using the **Extract Bin-Based Epochs** panel with the settings shown in Screenshot 6.8. Rather than using the default epoch of -200 to 800 ms, we're now using an epoch of -600 to 400 ms. This is because much of the activity occurs before rather than after time zero in a response-locked average. Also, rather than using **Pre** as the baseline correction interval (which would use the period from -600 to 0 ms), we're using a **Custom** baseline period of **-600 -400**. This period should be prior to stimulus onset on most trials because most of the RTs are <400 ms. (See this page in the ERPLAB Studio Tutorial to read about a way to "trick" ERPLAB into using the actual prestimulus interval as the baseline in a response-locked average.) Click **Apply** and name the resulting EEGset **12_P3_corrected_elist_resplock_be**.

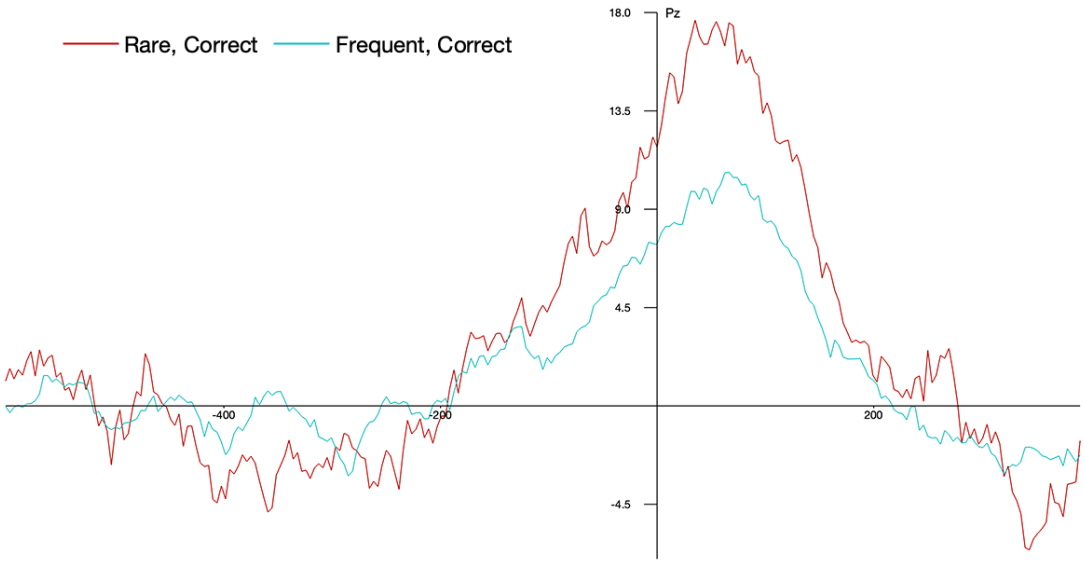


Screenshot 6.8

If you look at the EEG waveforms for the first epoch, you'll see an event code labeled **B2(201)** at time zero. The **B2** means that this epoch has been assigned to Bin 2, and the **201** indicates that the event code was 201. You'll also see that the epoch begins 600 ms before this event and ends 400 ms after the event. You can't see the stimulus event code prior to this response because it was more than 600 ms prior to the response and therefore falls outside the epoch. (You can verify this by looking at the **diff** column in the EventList.) However, you can see the stimulus event code prior to the response in the second epoch. The stimulus event code isn't at time zero, and it doesn't begin with **B2** because it wasn't the time-locking event. Scroll through several pages of epochs to make sure you understand what's in this file.

Now average the data using the **Compute Averaged ERPs** panel, just as you did for the stimulus-locked data, but using **12_P3_Response_Locked** when you are prompted for the name of the ERPset. You'll see a new window showing the number of trials per bin and a summary of the aSME values. You can close that window and look at the ERP waveforms in the **ERP** tab.

After you've looked at all the ERP waveforms, select **Pz** in the **Bin & Channel Selection** panel to get a close-up view of this channel (see Screenshot 6.9). You can see that the brain activity now begins approximately 200 ms before the time-locking event (the response), because the stimulus is now before time zero. In addition, the P3 peaks shortly after time zero.



Screenshot 6.9

The P3b component is usually tightly time-locked to the participant's decision about whether the current stimulus belongs to the Rare category or the Frequent category, so P3b latency usually (but not always) varies from trial to trial according to the response time. As a result, it can be informative to look at the P3b waveform in both stimulus- and response-locked averages. For an example from the basic science literature, see Luck & Hillyard (1990) or the condensed description of that study in Luck (2014;

especially Figure 8.8 and the surrounding text). For an example with a clinical population, see Luck et al. (2009).

This page titled 6.8: Exercise - Response-Locked Averaging is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 6.8: Exercise - Response-Locked Averaging by Steven J Luck is licensed CC BY 4.0.

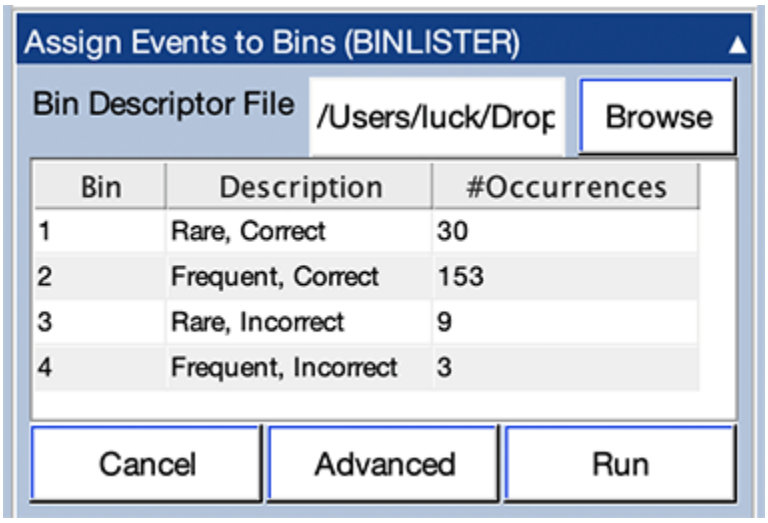
6.9: Exercise - Comparing Correct and Error Trials to See the Error-Related Negativity

In this exercise, we're going to create bins for the error trials as well as for the correct trials. This will allow us to look at the *error-related negativity* (ERN, also called *Ne*), a negative-going ERP response over frontocentral electrode sites that is produced when the participant makes an obvious error (see the excellent review by Gehring et al., 2012). Errors are pretty common on the Rare trials in the oddball paradigm because the response for the Frequent category becomes highly primed. When you're doing the task, you find yourself pressing the Frequent button even though you realize (a moment too late) that you should be pressing the Rare button.

Although the first published report of the ERN used an oddball task (Falkenstein et al., 1990), this task is less than ideal for looking at errors because error trials are a small subset of an already-rare stimulus category, leading to very few error trials. In the ERP CORE, we instead used a *flankers* task to look at the ERN, and we didn't even analyze the error trials in the oddball task. But because you've done all the other exercises in this chapter using the data from the ERP CORE oddball task, we're going to analyze the error trials in this task and look for the ERN. Do you think we'll see an ERN? I wasn't sure we'd see it until I analyzed the error trials for the first time!

My first step was to create a new bin descriptor file that includes bins for the error trials. The file is named **BDF_P3_Accuracy.txt**—go ahead and open it in the Matlab text editor (by double-clicking it in the Current Folder pane). You'll see that I just added Bins 3 and 4 for the Rare and Frequent stimuli followed by an error response (event code 202) in addition to the original Bins 1 and 2 that required that the stimulus was followed by a correct response (event code 201).

To get started, select **12_P3_corrected_elist** in the EEGsets panel. Note that this may require you to first select the **Continuous EEG** radio button. (If you don't still have this EEGset loaded, you can load it from the file in the Chapter_6 folder.) Then run BINLISTER, using **BDF_P3_ Accuracy.txt** as the bin descriptor file and naming the new EEGset **12_P3_corrected_elist_accuracy**. The BINLISTER panel will now indicate that, as before, there were 30 Rare trials followed by a correct response and 153 Frequent trials followed by a correct response in Bins 1 and 2 (see Screenshot 6.10). However, there were only 9 Rare trials followed by an incorrect response and a measly 3 Frequent trials followed by an incorrect response. And this is actually more error trials than was typical (probably because this participant's response times were faster than those of most participants in this experiment). In the ERP CORE flankers experiment, which was designed to look at the ERN, we controlled the number of errors by telling participants to speed up if they made errors on fewer than 10% of trials and telling them to slow down if they made errors on more than 20% of trials. But the oddball experiment was not designed to look at the ERN, so we didn't try to control the number of errors. Some participants made a lot, and some made hardly any.



Screenshot 6.10

As usual, the next step is to epoch the data with the **Extract Bin-Based Epochs** panel. The panel may still have the settings

from the response-locked averaging we did in the previous exercise, but now we're going to make stimulus-locked averages, so make sure that the epoch time range is set to **-200 800** and the baseline correction period is set to **Pre**. Click **Apply** and use the suggested EEGset name (**12_P3_corrected_elist_accuracy_be**). Now average the data using the **Compute Averaged ERPs** panel. You should name the resulting ERPset **12_P3_Accuracy** and save it to disk in a file named **12_P3_Accuracy.erp**. You'll need it for a later exercise.

You will see the summary window showing the number of trials per bin and a summary of data quality, including the best, worst, and median aSME values. You should always look at these values to make sure there isn't a problem with the data quality. You'll see that the maximum value was 10.07 for the bin with Frequent stimuli followed by an incorrect response, which had only 3 trials. Once you've looked at these values for a large number of participants, you'll get a sense of what range of values is normal. For almost any kind of ERP research, an aSME of 10.07 would be concerning and indicate that you should take a closer look at what is going on. You might find that one of the channels is very noisy, or you might find (as in this case) that one of the bins has very few trials.

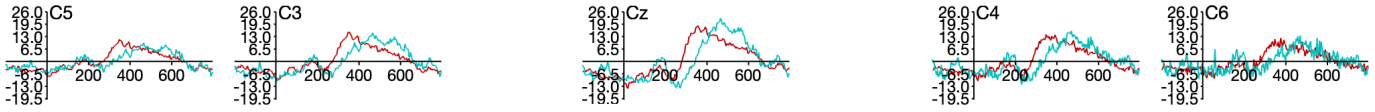
Close the averaging summary window and take a look at all the aSME values by clicking the **Show in a table** button in the **View Data Quality Metrics** panel. For the correct trials (Bins 1 and 2), the values look pretty reasonable (<2 in most cases). The SME quantifies the standard error of measurement, and a standard error of between 1 and 3 μV is reasonably small relative to the >15 μV amplitude of the P3b. Now look at the values for Rare stimuli followed by an incorrect response (Bin 3). Most of the values are still less than 3 μV, which seems reasonable given the small number of trials. If you look at the values for Bin 4, however, you'll see a lot of values that are quite a bit larger. This makes sense given that we had only 3 trials in this bin.

How many trials do you need? Some researchers have tried to provide a simple answer to this question, but there is no simple answer because it depends on both the number of participants and the magnitude of the effect being studied (Baker et al., 2020; Boudewyn et al., 2018).

Now plot the ERPs for the correct and incorrect Rare trials by specifying Bins 1 and 3 and all channels in the **Bin & Channel Selection** panel. To get a layout that better matches the scalp locations of the electrodes, go to the **Plot Settings** panel, change the **Grid Layout** to **Custom**, click the **Import** button, select the file named **GridLocations_CustomP3b.tsv**, and then click the **Apply** button.

Now we can finally answer the question of whether errors produce an ERN in this experiment (at least for this one participant). In most channels, you can see that the voltage is more negative from ~200-400 ms on the error trials than on the correct trials. The effect is largest at the midline sites, such as Cz in Screenshot 6.11. This is the ERN! Replication is a cornerstone of science, and it sure is nice to see that this effect can be replicated.

You can see that voltage is more positive for error trials than for correct trials from ~400-600 ms. This effect has also been seen in many prior studies, and it's called the *error positivity* or *Pe*.



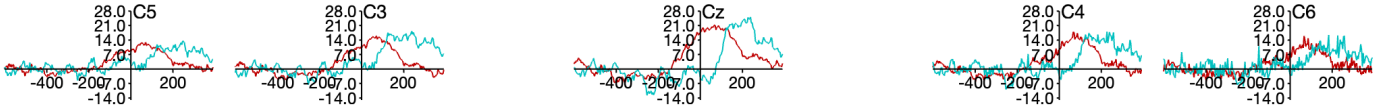
Screenshot 6.11

You should also try plotting the correct and incorrect Frequent waveforms (Bins 2 and 4). However, it's hard to see much because there were only 3 incorrect Frequent trials, so the waveforms are extremely noisy.

The ERN usually occurs right around the time of the response. When RTs vary greatly from trial to trial, the ERN in the stimulus-locked averages occurs at different times on different trials, and this makes the ERN look "smeared out" in stimulus-locked averages. The waveforms also look as if the P3b is just shifted to the right on the error trials. In fact, when Bill Gehring first saw the ERN, he was looking at stimulus-locked averages, and he wasn't sure it was a real component. However, when he made response-locked averages, the ERN was a big, beautiful deflection at time zero. You can read the story of how he discovered the ERN in Chapter 3 of Luck (2014).

The participant we've been looking at in this chapter didn't have a lot of RT variability, so the ERN is easily visible in the stimulus-locked averages. But let's make response-locked averages to see if this makes the ERN even clearer. To do this, follow the instructions in the previous exercise for making a response-locked average (especially using an epoch of -600 to 400 ms), but use **BDF_P3_Accuracy_Response_Locked.txt** as the bin descriptor file.

As shown in Screenshot 6.12, the response-locked ERN for Bins 1 and 3 appears as a sharper negative deflection, peaking shortly after time zero (near the time of the button press).



Screenshot 6.12

This page titled 6.9: Exercise - Comparing Correct and Error Trials to See the Error-Related Negativity is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 6.9: Exercise - Comparing Correct and Error Trials by Steven J Luck is licensed CC BY 4.0.

6.10: Exercise - Sequential Analysis of the P3b

A classic finding in the ERP literature is that the P3b component elicited by an oddball is smaller if the previous trial was also an oddball than if the previous trial was a standard (Squires et al., 1976). In this exercise, we will perform a sequential analysis to see this effect using the ERP CORE P3b paradigm. This will bring up some important general issues about ERPs as well as showing you more about the process of assigning events to bins.

I've provided a bin descriptor file for this analysis named **BDF_P3_Sequential.txt**—go ahead and open it in the Matlab text editor (by double-clicking it in the Current Folder pane). You'll see that we have 4 bins: Bin 1, Rare preceded by Rare; Bin 2, Rare preceded by Frequent; Bin 3, Frequent preceded by Rare; Bin 4, Frequent preceded by Frequent. The bin descriptors were modified to require that the time-locking stimulus was preceded by either the Rare or Frequent stimulus and a correct response. For example, Bin 1 is defined as shown in Figure 6.3.

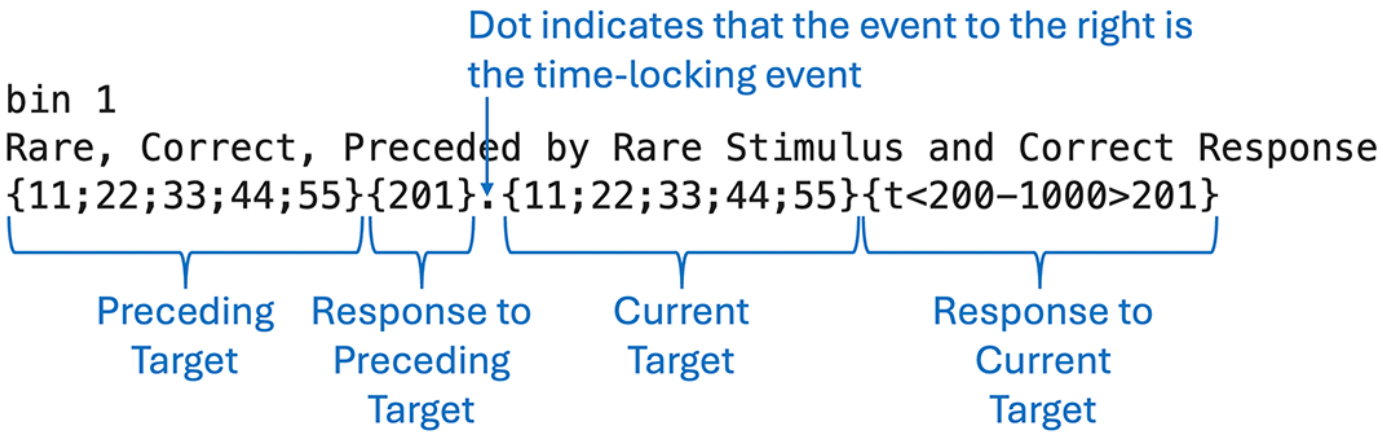
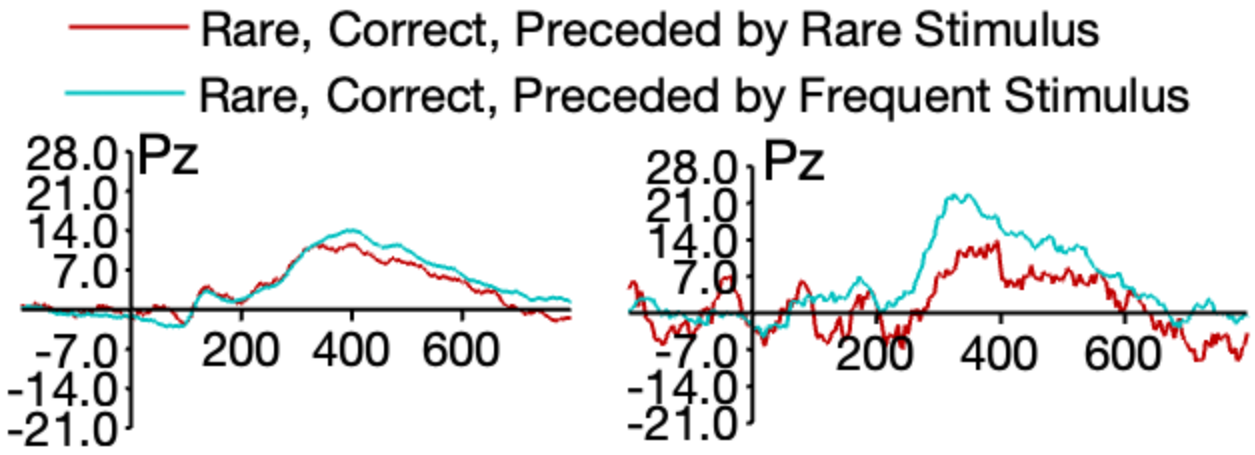


Figure 6.3. Bin descriptor for Bin 1 of the sequential analysis.

Let's make some ERPs with these bins. Go to the **EEG** tab and select **12_P3_corrected_elist** in the **EEGsets** panel (or load it from the file in the Chapter_6 folder.) Then run BINLISTER, using **BDF_P3_ Sequential.txt** as the bin descriptor file and naming the new EEGset **12_P3_corrected_elist_bins_seq**. We'll need the resulting dataset for a later exercise, so save it as a file named **12_P3_corrected_elist_bins_seq.set**.

Take a look at the number of trials per bin in the BINLISTER panel. There were only 6 Rare trials that were preceded by Rare trials (with correct responses on both trials). That's not a lot! This was a pretty short experiment (about 10 minutes), and we would ordinarily use a longer session with a lot more trials to do a sequential analysis. In fact, when I was developing this exercise, the first subject I tried didn't have a clear sequential effect; there was a hint of an effect, but the ERPs were so noisy that it wasn't very clear. I then wrote a script to do the analysis for all the participants (which is provided in the Chapter_6 folder). Fortunately, when I looked at the grand average, I saw the nice effect shown on the left of Screenshot 6.13, in which the P3b for the Rare stimulus was slightly larger when the preceding trial was the Frequent stimulus than when it was the Rare stimulus. I then noticed that Subject 12 exhibited this effect very strongly, so I used Subject 12 for all the exercises in this chapter.



Screenshot 6.13

The next step is to epoch the data with the **Extract Bin-Based Epochs** panel. Make sure that the epoch time range is set to **-200 800** and the baseline correction period is set to **Pre**. Click **Apply** and use the suggested name of **12_P3_corrected_elist_bins_seq_be**. Now average the data using the **Compute Averaged ERPs** panel, naming the resulting ERPset **12_P3_Sequential** and saving it as a file named **12_P3_Sequential.erp**. Now select the bins for the Rare stimuli (Bins 1 and 2) in the **Bin & Channel Selection** panel (in the ERP tab). If you look at the Pz channel in the plotting region, you should see something like the waveforms shown on the right of Screenshot 6.13. Once again, we've successfully replicated a finding from prior research!

However, we need to worry about the data quality given the small number of trials. Click the **Show in a table button** in the **View Data Quality Metrics** panel and take a look at the aSME values for Bins 1 and 2. You'll see that most of the values are worse (higher) for Bin 1 than for Bin 2, which is not surprising given that we had 6 trials in Bin 1 and 22 trials in Bin 2. But if the effect is large enough and we have enough participants, we may still obtain a statistically significant difference.

Sequence or Time?

Although many P3b effects have traditionally been interpreted in terms of the sequence of stimuli and sequential probability, many of these effects appear to be primarily a result of the amount of time between stimuli of the same category (Polich, 2012). That is, there was more time on average between Rare stimuli than between Frequent stimuli, and this may explain the larger P3b for the Rare stimulus category. Similarly, the amount of time between a Rare stimulus and the most recent Rare stimulus was longer when the Rare stimulus was preceded by a Frequent stimulus than when the Rare stimulus was preceded by a Rare stimulus, and this may explain the larger P3b for Rare preceded by Frequent than for Rare preceded by Rare.

This page titled 6.10: Exercise - Sequential Analysis of the P3b is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 6.10: Exercise - Sequential Analysis of the P3b by Steven J Luck is licensed CC BY 4.0.
- 6.2: Design of the ERP CORE Visual Oddball P3b Experiment by Steven J Luck is licensed CC BY 4.0.

6.11: Exercise - Combining Bins

In the previous exercise, we took the Rare and Frequent epochs and subdivided them according to whether the previous trial was Rare or Frequent, yielding four bins instead of two. We should be able to reconstruct our original two bins (Rare preceded by anything and Frequent preceded by anything) from the four new bins. However, there is a little trick that you need to know about.

Let's start by thinking about how to reconstruct the original Rare bin, in which the preceding trial could be either Rare or Frequent. We could simply take Bin 1 (Rare preceded by Rare) and average it with Bin 2 (Rare preceded by Frequent). In the **Bin Operations** panel, we could implement this with the following equation:

bin5 = (bin1 + bin2) / 2 Label Rare preceded by Rare or Frequent

However, this would not give us the same waveforms that we obtained in our original analysis that disregarded whether the previous trial was Rare or Frequent. Can you figure out why?

The reason is that this equation gives Bins 1 and 2 equal weight in the new bin, but there were 6 trials with Bin 1 and 22 trials with Bin 2 (which you can see in the **BINLISTER** panel or in the **ERP & Bin Information** panel). The equation above assumes that there were equal numbers of trials in each bin. To give each trial equal weight (which is what happened when the identity of the previous trial was disregarded by BINLISTER), we would need the following equation in **ERP Bin Operations**:

bin5 = (6*bin1 + 22*bin2) / 28 Label Rare preceded by Rare or Frequent

However, the exact equation would depend on how many trials were in Bins 1 and 2, which might vary across participants. To make this easier, **ERP Bin Operations** contains a function called *wavgbin* for performing *weighted averaging*, in which each bin is automatically weighted by the number of trials.

Let's see how these different methods of averaging work. Make sure that the ERPset from the previous exercise (**12_P3_Sequential**) is selected in the ERPsets panel. Then go to the **Bin Operations** panel, clear out any previous equations with the **Clear** button, and enter the following three equations:

bin5 = (bin1 + bin2) / 2 Label Rare preceded by Rare or Frequent, equal weight

bin6 = (6*bin1 + 22*bin2) / 28 Label Rare preceded by Rare or Frequent, manually trial-weighted

bin7 = wavgbin(bin1, bin2) Label Rare preceded by Rare or Frequent, automatically trial-weighted

Make sure that the **Mode** is set to **Modify existing ERPset**, because we're going to add these three bins to the four existing bins. Then click **Run** to update the ERPset with the added bins.

Now select the Pz electrode and Bins 5, 6, and 7 in the **Bin & Channel Selection** panel. You can't actually see the waveforms for Bin 6 because it's identical to Bin 7, and the Bin 7 waveform exactly covers up the Bin 6 waveform. This shows that the **wavgbin** function is working properly. However, the P3b is slightly larger for these bins than for Bin 5, which equally weights the two bins. Why is that? You should be able to figure it out given what you know about the amplitude of the P3b in Bins 1 and 2 and the nature of the equations used for Bins 5, 6, and 7.

An obvious question is whether you should use unweighted averaging (as in Bin 5) or weighted averaging (as in Bins 6 and 7) when combining bins together. There isn't a single answer to this question. If you're trying to create something equivalent to what you would have gotten if you hadn't subdivided the trials to begin with, then you'll want to use weighted averaging. But beyond that, the answer will depend on the logic of your experimental design and your scientific questions.

This page titled 6.11: Exercise - Combining Bins is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 6.11: Exercise - Combining Bins by Steven J Luck is licensed CC BY 4.0.

6.12: Exercise - Overlap

A concern often arises in a sequential analysis like the one we performed in this chapter, namely that the results are distorted by overlapping activity from the previous trial. A simple example of overlap is shown in Figure 6.4, which simulates an experiment in which a stimulus is presented every 700 ms. The falling edge of the P3b from one trial is present during the prestimulus baseline period of the next trial, which impacts the baseline correction procedure. That is, baseline correction involves taking the mean voltage during the baseline period and subtracting it from each point in the waveform. The positive voltage from the overlap in the baseline causes us to overestimate the true baseline voltage, and the entire waveform is shifted downward as a result. You might think that we should just skip the baseline correction step, but it's essential in 99.9% of experiments because there are huge sources of low-frequency noise that would contaminate the waveform if we did not subtract the baseline.

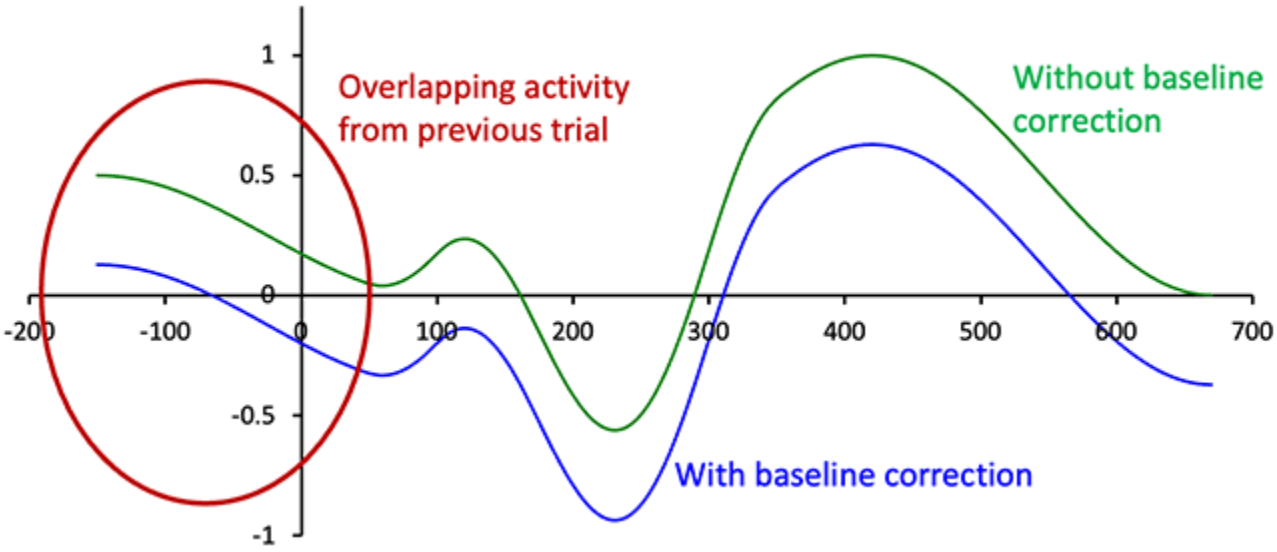


Figure 6.4. Example of the impact of overlap on an ERP waveform before and after baseline correction. In this example, we are assuming that a stimulus is presented every 700 ms, so the falling edge of the P3b from one trial is present during the prestimulus period of the next trial. When this positive voltage is subtracted out by the baseline correction process, it shifts the poststimulus waveform downward.

Some amount of overlap is present in most experiments, but it's not usually a problem unless it differs across conditions. For example, if the P3b on the previous trial was larger in Condition A than in Condition B, this would produce a larger positive voltage in the baseline of Condition A. The baseline correction procedure would then push the rest of the waveform farther downward in Condition A than in Condition B. As a result, the differences in overlap between Conditions A and B during the prestimulus period end up creating a difference between the waveforms for these conditions in the poststimulus period. This is a fundamentally important issue, and I recommend reading Marty Woldorff's foundational paper on overlapping ERP activity (Woldorff, 1993). In fact, it's on my list of papers that every new ERP researcher should read.

I hope you can now see why I'm a little worried about comparing the Rare-preceded-by-Rare waveform with the Rare-preceded-by-Frequent waveform. The overlapping voltage in the baseline might be larger when the previous trial was Rare (yielding a large P3b) than when it was Frequent. The baseline correction procedure would then push the waveform farther downward on Rare-preceded-by-Rare trials than on Rare-preceded-by-Frequent trials, artificially creating the appearance of a smaller (less positive) P3b in the Rare-preceded-by-Rare waveform than in the Rare-preceded-by-Frequent waveform.

However, if the time between trials is long enough, the P3b from the previous trial will have faded prior to the baseline period of the current trial. In the ERP CORE visual oddball P3b experiment, the time from one stimulus onset to the next (the *stimulus onset asynchrony* or *SOA*) was 1400-1600 ms. The 200-ms prestimulus interval that we've used for the current trial therefore began at least 1200 ms after the onset of the stimulus from the previous trial. That seems like it ought to be enough time for the P3b from the previous trial to end, but it's difficult to be certain without additional analyses.

In this exercise, we're going to repeat the sequential analysis from the previous exercise, but taking a closer look at the overlapping activity from the previous trial. In particular, we're going to dramatically increase the length of the prestimulus portion

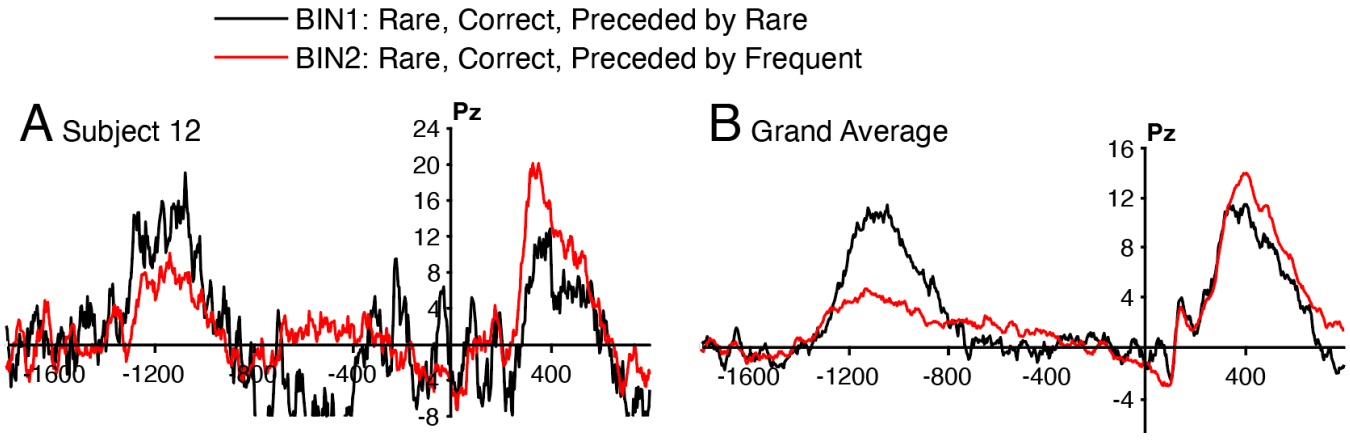
of the epoch so that we can see the previous-trial ERP as well as the current-trial ERP. And we're going to see what happens when we use different parts of the prestimulus period for baseline correction.

We're simply going to change the epoch length, so we can start with the EEGset you created using BINLISTER in the previous exercise, named **12_P3_corrected_elist_bins_seq**. Go to the **EEG** tab and select this EEGset in the **EEGsets** panel (which may require loading it from the file you saved in the previous exercise). Now epoch the data using the **Extract bin-based epochs** panel using an epoch time range of **-1800 800** and a baseline correction period of **-1800 -1600**. The maximum SOA in this experiment was 1600 ms, so the period from -1800 to -1600 ms is always prior to the previous trial. By using this as our baseline, we can be certain that the baseline correction won't be influenced by whether the previous trial was the Rare or Frequent category.

Now average the data, naming the ERPset **12_Sequential_LongBaseline** and saving it as a file named **12_Sequential_LongBaseline.erp**. Select Bins 1 and 2 in the **Bin & Channel Selection** panel.

Screenshot 6.14A shows what it should look like in the Pz channel. You can still see that the current-trial P3b is larger for Rare-preceded-by-Frequent than for Rare-preceded-by-Rare. That suggests that the sequential effect that we saw before was not a result of the combination of overlap and baseline correction. However, the data are pretty noisy, so I ran this analysis on all the participants. Screenshot 6.14B confirms that the sequential effect in the grand average remains when the data are baselined relative to the prestimulus period from the previous trial.

The grand average waveforms also make it clear that differential overlap is a real concern. That is, the persisting voltages from the previous trial were somewhat different according to whether that trial was Rare or Frequent, potentially contaminating the baseline period from the current trial. However, the overlapping activity seems to go in the opposite direction of the sequential effect, being more negative (near time zero) when the previous trial was Frequent than when it was Rare, whereas the current-trial P3b was more positive.



Screenshot 6.14

Although the grand averages looked fine when we used the prestimulus period from the prior trial as the baseline interval, this approach can decrease the precision and reliability of the ERP waveforms. As illustrated in Figure 6.5, this decline in precision occurs because the single-trial EEG voltage tends to drift gradually over time in a random direction, and the trial-to-trial variability increases as you get farther away from the baseline period. This increased variability increases the error in measuring the "true" amplitude, increasing variability across participants and thereby decreasing statistical power. Indeed, the waveforms in some of the channels look pretty crazy with this distant baseline.

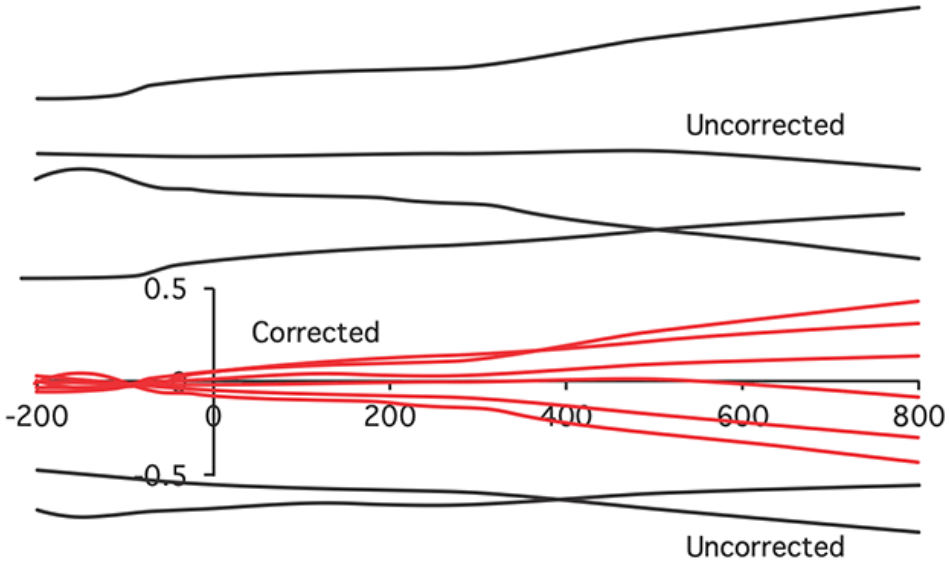


Figure 6.5. Illustration of the combined effects of low-frequency drift and baseline correction. The EEG is superimposed on low-frequency signals (mainly arising from the skin) that drift slowly over time in a random direction. When the data are baseline-corrected, this forces the voltages to be similar during the baseline period, but the voltage tends to drift farther away from the baseline value, causing a gradual increase in trial-to-trial variability over time.

You can use the Standardized Measurement Error to quantify this increase in measurement error. Click **Show in a table** in the **View Data Quality Metrics** panel and look at the aSME values. We'll want to compare these values to the corresponding values from the original sequential analysis in which the baseline correction period was -200 to 0 ms. To do this, leave the data quality window for the current ERPset open, select the version of this ERPset with the original baseline (**12_P3_Sequential**) in the **ERPsets** panel, and click **Show in a table** in the **View Data Quality Metrics** panel to see the aSME values for that ERPset.

Take a close look at the aSME values for the Pz channel in Bin 1 for the two ERPsets. You should see that they are substantially worse (larger) when the -1800 to -1600 period was used as the baseline. That's exactly what we'd expect from Figure 6.5. So, although using a distant time period for baseline correction can sometimes be valuable for assessing overlap, it comes at the cost of increased measurement error. As a result, I find that analyses like these are mainly useful descriptively and not for statistical analysis. In other words, it's usually enough to look at the grand averages and see that the effects are largely the same with the two different baseline periods without performing a statistical analysis on the data with the distant baseline period.

This page titled 6.12: Exercise - Overlap is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 6.12: Exercise - Overlap by Steven J Luck is licensed CC BY 4.0.

6.13: Matlab Script For This Chapter

I've provided a script called **sequential_analysis.m** in the Chapter_6 folder that implements the sequential analysis.

This page titled 6.13: Matlab Script For This Chapter is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 6.13: Matlab Script For This Chapter by Steven J Luck is licensed CC BY 4.0.

6.14: Key Takeaways and References

Key Takeaways

- To ensure that ERP effects reflect psychological processes instead of low-level physical stimulus differences, it is important to follow the Hillyard Principle: Keep the stimuli constant and vary only the psychological conditions.
- The impact of noise on data quality depends on both the nature of the noise (e.g., low- versus high-frequency) and the method of quantifying an ERP component (e.g., mean versus peak amplitude).
- The standardized measurement error (SME) quantifies the standard error of an ERP amplitude or latency measurement. In simple cases, the standard error decreases as the square root of the number of trials increases. As a result, the signal-to-noise ratio increases as the square root of the number of trials increases (all else being equal).
- Data quality typically declines when the activity being measured is distant in time from the baseline correction period.
- It can be useful to look at the same data using both stimulus-locked and response-locked averages.
- When averaging bins together, you need to think carefully about whether to weight the two bins equally or to weight by the number of trials per bin.
- Overlapping activity from previous trials can be a significant confound, especially when combined with baseline correction.

References

Baker, D. H., Vilidaite, G., Lygo, F. A., Smith, A. K., Flack, T. R., Gouws, A. D., & Andrews, T. J. (2020). Power contours: Optimising sample size and precision in experimental psychology and human neuroscience. *Psychological Methods*. http://dx.doi.org/10.1037/met0000337

Boudewyn, M. A., Luck, S. J., Farrens, J. L., & Kappenman, E. S. (2018). How Many Trials Does It Take to Get a Significant ERP Effect? It Depends. *Psychophysiology*, *55*, e13049. https://doi.org/10.1111/psyp.13049

Duncan-Johnson, C. C., & Donchin, E. (1977). On quantifying surprise: The variation of event-related potentials with subjective probability. *Psychophysiology*, *14*, 456–467.

Falkenstein, M., Hohnsbein, J., Joormann, J., & Blanke, L. (1990). Effects of errors in choice reaction tasks on the ERP under focused and divided attention. In C. H. M. Brunia, A. W. K. Gaillard, & A. Kok (Eds.), *Psychophysiological Brain Research* (pp. 192–195). Elsevier.

Gehring, W. J., Liu, Y., Orr, J. M., & Carp, J. (2012). The error-related negativity (ERN/Ne). In S. J. Luck & E. S. Kappenman (Eds.), *The Oxford Handbook of Event-Related Potential Components* (pp. 231–292). Oxford University Press.

Kappenman, E. S., Farrens, J. L., Zhang, W., Stewart, A. X., & Luck, S. J. (2021). ERP CORE: An Open Resource for Human Event-Related Potential Research. *NeuroImage*, *225*, 117465. https://doi.org/10.1016/j.neuroimage.2020.117465

Luck, S. J. (2014). *An Introduction to the Event-Related Potential Technique, Second Edition*. MIT Press.

Luck, S. J., & Hillyard, S. A. (1990). Electrophysiological evidence for parallel and serial processing during visual search. *Perception & Psychophysics*, *48*, 603–617.

Luck, S. J., Kappenman, E. S., Fuller, R. L., Robinson, B., Summerfelt, A., & Gold, J. M. (2009). Impaired response selection in schizophrenia: Evidence from the P3 wave and the lateralized readiness potential. *Psychophysiology*, *46*, 776–786.

Luck, S. J., Stewart, A. X., Simmons, A. M., & Rhemtulla, M. (2021). Standardized Measurement Error: A Universal Measure of Data Quality for Averaged Event-Related Potentials. *Psychophysiology*. https://doi.org/10.1111/psyp.13793

Luck, S. J., & Vogel, E. K. (2013). Visual Working Memory Capacity: From Psychophysics and Neurobiology to Individual Differences. *Trends in Cognitive Sciences*, *17*, 391–400.

Luria, R., Balaban, H., Awh, E., & Vogel, E. K. (2016). The contralateral delay activity as a neural measure of visual working memory. *Neuroscience & Biobehavioral Reviews*, *62*, 100–108. https://doi.org/10.1016/j.neubiorev.2016.01.003

Polich, J. (2012). Neuropsychology of P300. In S. J. Luck & E. S. Kappenman (Eds.), *The Oxford Handbook of Event-Related Potential Components* (pp. 159–188). Oxford University Press.

Squires, K., Wickens, C., Squires, N. K., & E. Donchin. (1976). The effect of stimulus sequence on the waveform of the cortical event-related potential. *Science*, *193*, 1142–1146. https://doi.org/10.1126/science.959831

Vogel, E. K., Luck, S. J., & Shapiro, K. L. (1998). Electrophysiological evidence for a postperceptual locus of suppression during the attentional blink. *Journal of Experimental Psychology: Human Perception and Performance*, *24*, 1656–1674.

Woldorff, M. (1993). Distortion of ERP averages due to overlap from temporally adjacent ERPs: Analysis and correction. *Psychophysiology*, *30*, 98–119.

This page titled 6.14: Key Takeaways and References is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 6.14: Key Takeaways and References by Steven J Luck is licensed CC BY 4.0.

7: INSPECTING THE EEG AND INTERPOLATING BAD CHANNELS

Learning Objectives

In this chapter, you will learn to:

- Inspect the raw data to determine what problems exist that might require intervention
- Determine whether an EEG channel is “bad” and replace the bad channel with interpolated values
- Identify “by eye” the most common types of artifacts, including blinks, saccadic eye movements, and muscle noise

I often say that it’s a miracle that we can record meaningful brain activity from electrodes on the skin, with a thick skull between our electrodes and the brain. Although the EEG is miraculous, it's also an imperfect measure of brain activity. There are many types of artifacts that can contaminate the data, and some of the electrodes may have poor electrical connections to the scalp, creating *bad channels*.

Several preprocessing steps are necessary prior to averaging so that artifacts and bad channels don’t lead you to draw incorrect conclusions from the ERPs. In particular, we can replace the data from bad channels by interpolation from the good channels, and we can apply artifact rejection and correction to minimize the impact of artifacts.

Prior to performing these steps, it is **extremely important** that you first perform a visual inspection of a given participant’s EEG so that you understand what kinds of problems are present in that dataset. You also need to spend a lot of time looking at raw EEG data when you are learning to perform preprocessing so that you learn how to identify the most common types of problems.

In this chapter, you will see how to visually inspect EEG data and how to identify bad channels and common artifacts. You’ll also learn how to replace bad channels with interpolated data. The subsequent chapters will cover how to reject and correct artifacts. We’ll look at data from one participant in the mismatch negativity (MMN) experiment from the ERP CORE (Kappenman et al., 2021).

Before we get started, it's important to think about what our goal is for these steps. I find that most people haven’t thought clearly about what they are trying to achieve in their preprocessing pipeline, and this leads to pipelines that are less than ideal.

So, what is the goal of EEG preprocessing? It’s simple: **our fundamental goal is to accurately answer the scientific question that the experiment was designed to address**. If we include bad channels or large artifacts, this will likely move us away from this goal. However, if we throw out a lot of trials or participants, this will also tend to move us away from our goal by reducing our statistical power. Most preprocessing steps—such as interpolating bad channels or rejecting trials with artifacts—have both costs and benefits, and we need to ask whether the benefits of a given processing step outweigh the costs in terms of answering the scientific question that the experiment was designed to address. We will discuss how to assess these costs and benefits as we go through this chapter and the next two chapters.

7.1: Data for This Chapter

7.2: Design of the Mismatch Negativity (MMN) Experiment

7.3: Video Demonstration- Performing an Initial Inspection of a Participant’s EEG

7.4: Background- Interpolating Bad Channels

7.5: Exercise - Interpolating Bad Channels

7.6: Matlab Script For This Chapter

7.7: Key Takeaways and References

This page titled 7: Inspecting the EEG and Interpolating Bad Channels is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 7: Inspecting the EEG and Interpolating Bad Channels by Steven J Luck is licensed CC BY 4.0.

7.1: Data for This Chapter

The data we'll be using for the exercises in this chapter can be found in the Chapter_7 folder in the master folder: https://doi.org/10.18115/D5G045.

This page titled 7.1: Data for This Chapter is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 7.1: Data for This Chapter by Steven J Luck is licensed CC BY 4.0.

7.2: Design of the Mismatch Negativity (MMN) Experiment

The exercises in this chapter and the chapters on artifact rejection and correction will use data from the ERP CORE MMN experiment, and this section will provide a brief overview of the experimental design and main results.

As illustrated in Figure 7.1.A, the experiment involved a sequence of frequently occurring *standard* tones (1000 Hz, 80 dB, p = .8) and rare *deviant* tones that were slightly softer (1000 Hz, 70 dB, p = .2). Each tone was presented for 100 ms, and successive tones were separated by a silent interstimulus interval of 450-550 ms. The MMN does not require active attention, so the tones were task-irrelevant. Instead, the participants were instructed to simply watch a small silent movie of a sand drawing in the middle of the computer monitor (Figure 7.1.B). This kept the participants awake and alert, minimizing alpha-band EEG oscillations. The movie was quite small to avoid large eye movements. Additional methodological details can be found in Kappenman et al. (2021).

The grand average ERPs for the standards and the deviants are shown in Figure 7.1.C. The waveforms are shown for the FCz channel, where the MMN is typically largest. The MMN can be seen as a more negative voltage for the deviants than for the standards from approximately 125–225 ms. The MMN is typically isolated from the other overlapping ERP components by means of a deviant-minus-standard difference wave, which is shown in Figure 7.1.D.

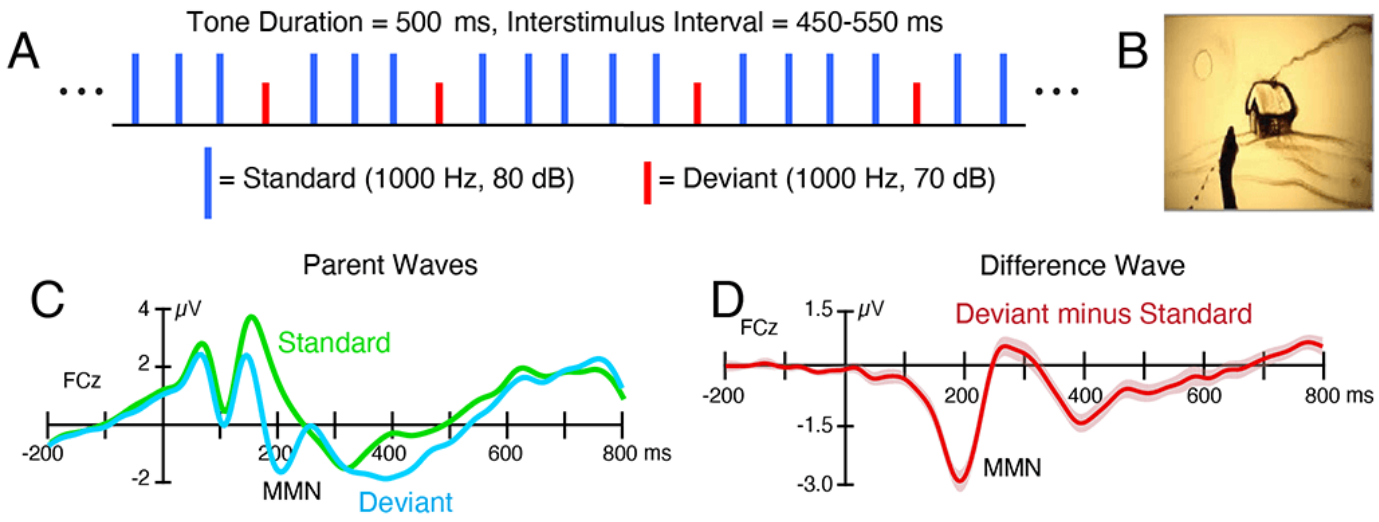


Figure 7.1. Experimental paradigm and results from the ERP CORE Mismatch Negativity experiment. (A) Example sequence of standard tones (p=.8) and deviant tones (p=.2). The tones were task-irrelevant. (B) One frame of the silent movie that participants watched while the tones were playing. (C) Grand average ERP waveforms elicited by the standard and deviant tones, averaged over all 40 participants. (D) Grand average difference wave, which was created by subtracting the averaged ERP waveform for the deviant trials from the averaged ERP waveform for the standard trials.

This page titled 7.2: Design of the Mismatch Negativity (MMN) Experiment is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 7.2: Design of the Mismatch Negativity (MMN) Experiment by Steven J Luck is licensed CC BY 4.0.

7.3: Video Demonstration- Performing an Initial Inspection of a Participant's EEG

As I mentioned at the beginning of the chapter, it's important to scan through each participant's EEG to see what kinds of artifacts are in the data. The best way to learn to do this is to go through the data from several participants with an expert who can show you what to look for. I've tried to simulate that experience by creating a video where I go through the data from Subject #1 in the ERP CORE mismatch negativity experiment.

Every participant is different, but Subject #1 is a good example because the recording contains many of the problems you'll encounter. In other words, Subject #1 is worse than usual for a basic science experiment in my lab, which is a good thing for seeing examples of problems in EEG data.

When I first started going through Subject #1's data, the EEG seemed super clean (and therefore not a good example). But as you'll see in the video, things go downhill pretty rapidly after the first couple minutes of the recording.

You should follow along on your own computer while you watch the video. To do this, reset everything with the **Reset** button, set **Chapter_7** to be Matlab's current folder, and the load the EEGset named **1_MMN_preprocessed.set**.

Go ahead and watch the video now. You can access it at this YouTube link. It's about 10 minutes long, and you'll definitely need to watch it before you read the rest of this chapter and the chapters on artifact rejection and correction.

This page titled 7.3: Video Demonstration- Performing an Initial Inspection of a Participant's EEG is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 7.3: Video Demonstration- Performing an Initial Inspection of a Participant's EEG by Steven J Luck is licensed CC BY 4.0.

7.4: Background- Interpolating Bad Channels

In the video demonstration, the C5 channel seemed quite problematic, and I mentioned that we might want to interpolate that channel. Also, the F8 channel had a lot of high-frequency noise toward the end of the session, so we should consider interpolating that channel as well. In this exercise, we'll discuss how to decide whether a channel should be interpolated and then look at the actual interpolation process.

Interpolation is performed separately at each time point, using the voltage values from the other channels at that time point. There are several algorithms that can be used to estimate a reasonable value for one channel on the basis of the other channels, and they all work reasonably well. I like to use the *spherical* algorithm that we "borrowed" from EEGLAB. It treats the electrodes as if they're located on a spherical head, and it fits a polynomial function to the distribution of voltages, leaving out the to-be-interpolated channel. Then, the value of the polynomial function at the to-be-interpolated location is used as the estimated voltage at that location. This process is repeated independently at each time sample. You get a pretty reasonable waveform at the interpolated location, but keep in mind that it's just an imperfect estimate of the true waveform.

The decision about whether to interpolate a given channel ultimately comes down to our fundamental goal I mentioned at the beginning of the chapter, namely accurately answering the scientific question that the experiment was designed to address. I'll provide some general guidance, but ultimately you need to think about whether interpolation gets you closer to the truth or farther from the truth. Imagine that you've submitted a manuscript to a journal, and one of the reviewers visits your lab to see how you actually processed your data. Would you be happy to explain to this reviewer how you decided whether to interpolate? Or would you feel a little embarrassed?

Let's start with an extreme case. Imagine that an electrode was broken and the signal from that electrode was pure noise. And imagine that the broken electrode wasn't being used in any of the key analyses. For example, we measured MMN amplitude at FCz for the main analyses in the ERP CORE paper (Kappenman et al., 2021), so the C5 and F8 electrodes didn't play a major role in our analyses (although they did make a minor contribution when we plotted scalp maps). One way to deal with the broken electrode would be to completely discard this participant's data. This would reduce the sample size in our analyses, which tends to decrease our ability to draw accurate conclusions about the population, so that's not a great option. The other main option would be to interpolate the data from the broken electrode. Given that the broken electrode doesn't contribute to the main analyses, being able to include the participant by interpolating and thereby increasing your sample size seems like it serves the truth much more than excluding the participant from all analyses.

Now let's consider an extreme case in the opposite direction. Imagine that you're analyzing data collected in another lab with really poor recording methods (or a really challenging participant), and the data from FCz and the 8 closest electrodes look terrible. Not only is the key channel for the analysis bad, but so are the surrounding channels, which will make it difficult to interpolate accurately. In this case, including the participant in the final analyses seems like it will not add real information and will take us further from the truth rather than closer.

Several years ago, I posted a message on Twitter/X asking how researchers decide whether any channels should be interpolated. Some people indicated that they do it informally by visual inspection. Bad channels are relatively rare in my lab, so we also use visual inspection. Other researchers indicated that they used an automated method, such as the Clean Rawdata plugin in EEGLAB or the PREP pipeline (Bigdely-Shamlo et al., 2015). Many researchers who said that they use an automated method indicated that the algorithm fails often enough that they visually confirm the results. I've never tried the automated methods, but the algorithms seem reasonable (especially the PREP pipeline), and I would expect them to work well when verified by visual inspection. However, they mainly rely on statistical criteria, such as how well a given channel correlates with other channels (with a low correlation suggesting a problem given that true brain signals spread broadly across the scalp and therefore produce high correlations between nearby electrode sites).

This will certainly work for detecting things like loose electrodes. However, if an electrode is properly connected but the signal contains biological noise (e.g., muscle activity), the question is whether the noise decreases your ability to precisely quantify the amplitude or latency value that will be the dependent variable in your statistical analyses (because this is how we reach our fundamental goal of accurately answering the scientific question that the experiment is designed to address). The Standardized Measurement Error (SME) metric of data quality is ideally suited for this purpose because it quantifies the extent to which the noise in the data distorts the specific amplitude or latency value you will be obtaining from the averaged ERPs. With this in mind, let's consider the actual cases of the C5 and F8 electrodes from Subject #1 in the ERP CORE MMN experiment.

This page titled 7.4: Background- Interpolating Bad Channels is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 7.5: Background- Interpolating Bad Channels by Steven J Luck is licensed CC BY 4.0.

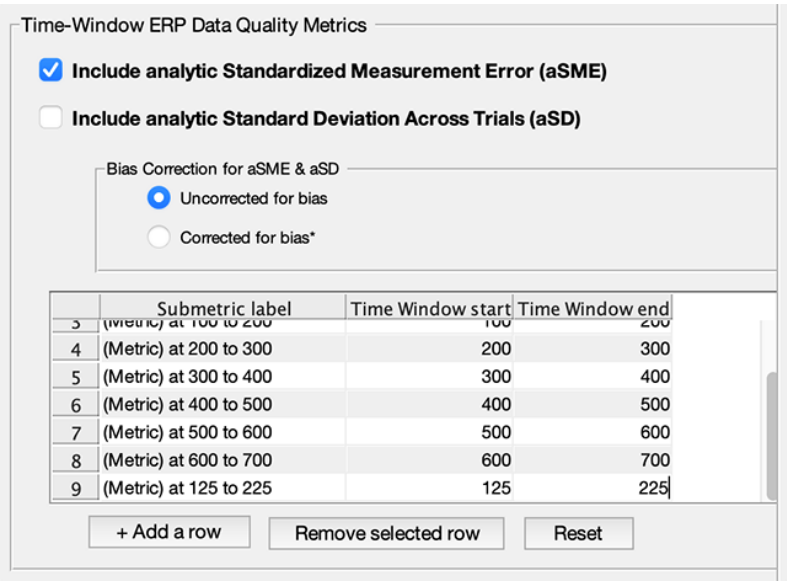
7.5: Exercise - Interpolating Bad Channels

If you don't already have the **1_MMN_preprocessed.set** EEGset loaded, you should load it now. This dataset has already been preprocessed, including referencing to the average of P9 and P10 and high-pass filtering with a half-amplitude cutoff at 0.1 Hz. An EventList has been added, and BINLISTER has been run to assign events to bins. We have only two bins: Deviant Preceded by Standard (Bin 1, 200 epochs)) and Standard Preceded by Standard (Bin 2, 585 epochs).

You've already looked carefully at the EEG while going through the video, and now we need to make a decision about whether C5 and/or F8 should be interpolated. To help make this decision, let's calculate the SME. First, we need to epoch the continuous EEG, so go to the **Extract bin-based epochs** panel and epoch the data with the default parameters (epoch of -200 to 800 ms with **Pre** as the baseline).

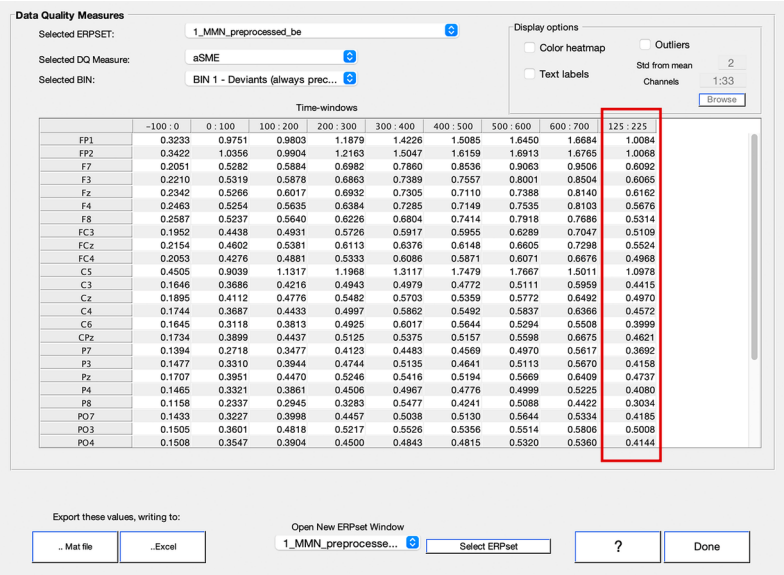
Now we're ready to compute the SME. The SME is automatically computed when we create averaged ERPs, but we don't need to look at averages right now, so you should instead get the SME values using the **Compute Data Quality Metrics** panel. This computes the SME in the same way that **Compute Averaged ERPs** does, but without creating an ERPset.

In this experiment, MMN amplitude is scored as the mean voltage from 125 to 225 ms, so we want to get the SME values for this time range. To do this, select **Custom parameters** and click the **Options** button in the **Compute Data Quality Metrics** panel. A new window will appear, as shown in Screenshot 7.1. Click **+Add a row** to add a new time window. You'll need to scroll down to see the new row. Change the submetric label to **(Metric) at 125 to 225** and change the time window start and end fields to **125** and **225** (see the screenshot). Then click **Save** to get back to the panel and click **RUN**.



Screenshot 7.1

A table with the data quality values will then appear (see Screenshot 7.2). If you look at the aSME values for Bin 1, the rightmost column will show the values for the 125-225 ms time range. You'll see that the aSME values for Fp1, Fp2, and VEOG-bipolar are all substantially higher than the values for the other channels. That's because of blinking, so you can ignore those channels. (You could perform artifact correction prior to examining the aSME to avoid this issue, especially if you were concerned that one of these channels should be interpolated.) Of the remaining channels, C5 really stands out, with a value that is much higher than the other channels.



Screenshot 7.2

The data quality table GUI includes two tools that can help you find problematic values. First, the **Color heatmap** option colors each cell according to the value relative to the other cells. Try that, and you'll see that the C5 channel pops out. The EOG, Fp1, and Fp2 sites also pop out, but that's because we haven't gotten rid of the blinks.

The second way of identifying problematic channels is to click the **Outliers** button. This will cause the table to highlight any cells that are more than N standard deviations (SDs) away from the mean aSME value for that time period. The default setting of N is 2, which means that it will highlight cases that are more than 2 SDs above the mean. But when computing the SD, it would make sense to leave out the EOG, Fp1, and Fp2 channels, because they are a poor comparison for the other channels as a result of the blinking. The **Outliers** feature therefore allows you to specify a subset of the channels for computing the SD. Enter **3:28** in this text box. You'll now see that the C5 channel is highlighted in most or all time periods. When combined with the fact that C5 was clearly problematic in our visual inspection of the data, these aSME results provide good reason to interpolate this channel.

Oz will also be highlighted for the 600-700 ms time window. However, the value at Oz for the 125-225 window is just fine, and that's the window that matters for the MMN. Consequently, we don't need to interpolate Oz.

What about F8, which also looked pretty bad in our visual inspection? The aSME for F8 is well within the range of the other frontal electrode sites. This indicates that the high-frequency noise in F8 isn't really impacting our ability to quantify the amplitude of the MMN using the mean voltage between 125 and 225 ms. That noise could be problematic for other measures (e.g., peak amplitude), but it's not problematic for the planned analyses of the present study (especially given that the main analysis will be limited to FCz). So, there would be little value in interpolating this channel given the way the data will be analyzed.

In the analyses provided in the ERP CORE paper (Kappenman et al., 2021), F8 was interpolated because we did not yet have the SME metric and it "looked" noisy. Now I would make a different decision. I guess you can teach an old dog new tricks!

Now that we've decided to interpolate C5 but not F8, it's time to implement the interpolation. Close the **Data Quality** viewer window, and select the original continuous dataset (**1_MMN_preprocessed**) in the **EEGsets** panel. Then go to the **Interpolate Channels** panel and select **Create new dataset**. Put **11** in the **Interpolated chan** box (because C5 is Channel 11). We don't want our bipolar EOG electrodes to be used for the interpolation, because they don't have the same reference as C5 and would mess up the interpolation, so check the **Ignored chans** checkbox and put **32 33** in the corresponding text box. Select **Spherical** as the **Interpolation method** and click the **Run** button. Name the resulting dataset **1_MMN_preprocessed_interp** (and you may want to save it to your disk because we'll use it in the following exercises).

Now scroll through the EEGset. The C5 channel should now look beautiful, even later in the session when the original C5 channel looked terrible (e.g., between 125 and 130 seconds). Success! However, keep in mind that the C5 channel now contains *estimated* voltages, not *measured* voltages. But that's good enough for our present purposes, especially given that the main analyses will be performed on a different channel.

One last note about interpolation: If you are using the average across sites as the reference, the data from any bad channels will contaminate all the channels. You might therefore want to use a single electrode (or a pair of electrodes such as P9 and P10) as the reference prior to interpolation. You can then re-reference to the average of all sites after interpolation. See Appendix 3 for a detailed discussion of when to implement interpolation. A more complex but more robust approach is implemented by the PREP pipeline (Bigdely-Shamlo et al., 2015).

This page titled 7.5: Exercise - Interpolating Bad Channels is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck.

- 7.6: Exercise - Interpolating Bad Channels by Steven J Luck is licensed CC BY 4.0.
- 8.9: Exercise- Better Blink Detection by Steven J Luck is licensed CC BY 4.0.

7.6: Matlab Script For This Chapter

There is no script for this chapter. The script for the next chapter shows how to implement interpolation along with artifact rejection.

This page titled 7.6: Matlab Script For This Chapter is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 7.7: Matlab Script For This Chapter by Steven J Luck is licensed CC BY 4.0.

7.7: Key Takeaways and References

Key Takeaways

- The overarching goal for your EEG preprocessing is to maximize the likelihood that you will obtain an accurate answer to the scientific question your study is designed to answer. You can ignore any of my specific suggestions for your preprocessing pipeline if you have a better way of reaching that goal.
- You can more easily obtain an accurate answer to your scientific question if you look carefully at each participant's data prior to doing the preprocessing. By examining the data, you'll be able to adjust the preprocessing to reflect the unique problems of each individual participant's data.
- The standardized measurement error (SME) provides a useful metric for knowing whether a "bad channel" is really problematic with respect to the analyses you will be performing with your data.
- Interpolation is a low-risk procedure when the channel being interpolated will not be used for your main analyses. But if the channel will be used in your main analyses, you need to think carefully about whether to interpolate the channel or exclude the participant from analysis.

References

Bigdely-Shamlo, N., Mullen, T., Kothe, C., Su, K.-M., & Robbins, K. A. (2015). The PREP pipeline: Standardized preprocessing for large-scale EEG analysis. *Frontiers in Neuroinformatics*, *9*. https://doi.org/10.3389/fninf.2015.00016

Kappenman, E. S., Farrens, J. L., Zhang, W., Stewart, A. X., & Luck, S. J. (2021). ERP CORE: An open resource for human event-related potential research. *NeuroImage*, *225*, 117465. https://doi.org/10.1016/j.neuroimage.2020.117465

This page titled 7.7: Key Takeaways and References is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 7.8: Key Takeaways and References by Steven J Luck is licensed CC BY 4.0.

8: ARTIFACT DETECTION AND REJECTION

Learning Objectives

In this chapter, you will learn to:

- Conceptualize artifact rejection in terms of the overarching goal of accurately answering the scientific question that your experiment was designed to address
- Implement algorithms that work particularly well for detecting blinks, saccadic eye movements, and a broad class of artifacts termed *commonly rejected artifactual potentials (C.R.A.P.)*
- Select optimal parameters for the artifact detection parameters
- Evaluate the effectiveness of your artifact rejection procedures on the averaged ERP waveforms, including both data quality and confounds
- Implement a two-stage artifact rejection procedure for ensuring that experiments with lateralized stimuli or lateralized responses are not contaminated by small but consistent eye movements

EEG recordings are often filled with large artifacts. In most areas of research, blinks are the most problematic. They're large (often 200 μV), occur frequently (on over 50% of trials in many participants), and may differ systematically across groups or conditions, creating a significant confound if they aren't properly addressed. In research with infants, small children, or people who are required to move around during the task, movement-related artifacts are also a major issue. In my own area of research, eye movements toward lateralized targets are the most significant artifact.

However, in most of the ERP papers I read that use artifact rejection, it doesn't seem that much thought went into the strategy for dealing with artifacts. These papers typically use a very primitive algorithm for detecting trials with artifacts, and they use the same rejection threshold for all participants (even though artifacts differ quite a bit across individuals). This chapter is designed to help you conceptualize and implement artifact rejection in a more sophisticated manner, allowing you to minimize artifact-related confounds and maximize your data quality, thereby achieving the ultimate goal of providing an accurate answer to your scientific question.

Most ERP researchers use *artifact correction* instead of *artifact rejection* to deal with blinks. In artifact correction, a method such as independent component analysis is used to estimate and correct for the artifactual activity so that trials containing these artifacts do not need to be rejected. This means that more trials are available for averaging, which typically leads to cleaner averaged ERPs and greater statistical power.

However, most sophisticated ERP labs use a combination of artifact correction and artifact rejection. Artifact correction is mainly used to correct for eye blinks and eye movements and tends not to work well for some of the other common kinds of artifacts, such as movement artifacts. Rejection is therefore used for those artifacts. My lab has actually demonstrated that the combined use of artifact correction and artifact rejection is valuable across a broad range of ERP paradigms, both because it avoids confounds and because it improves statistical power (Zhang et al., 2024c).

In addition, ICA-based artifact correction requires reasonably long recording sessions, especially if you have a lot of channels. If you have short recordings and a lot of channels, you may need to reject trials with blinks rather than correcting them.

The bottom line is that you need to have a good understanding of artifact rejection even if you plan to use artifact correction. That is the goal of this chapter. The next chapter will discuss artifact correction and how to combine correction and rejection.

8.1: Data for This Chapter

8.2: Overview

8.3: Background- Why Do We Reject Artifacts?

8.4: Background- The General Approach

8.5: Exercise- Simple Blink Detection

8.6: Exercise- Adjusting the Threshold
8.7: Exercise- Manually Marking and Unmarking Artifacts
8.8: An Iterative Approach to Setting Parameters
8.9: Exercise- Effects of Artifact Rejection on Data Quality
8.10: Exercise- Better Blink Detection
8.11: Exercise- Commonly Recorded Artifactual Potentials (C.R.A.P.)
8.12: Exercise- Detecting Eye Movements
8.13: Using Artifact Detection to Avoid Changes to Visual Inputs
8.14: The ERP CORE N2pc Experiment
8.15: Exercise- Visualizing the Eye Movements
8.16: Exercise- Using the Averaged HEOG to Visualize Consistent Eye Movements
8.17: Exercise- A Two-Stage Strategy for Eliminating Small But Consistent Eye Movements
8.18: Matlab Script For This Chapter
8.19: Key Takeaways and References

This page titled 8: Artifact Detection and Rejection is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 8: Artifact Detection and Rejection by Steven J Luck is licensed CC BY 4.0.

8.1: Data for This Chapter

The data we'll be using for the exercises in this chapter can be found in the Chapter_8 folder in the master folder: https://doi.org/10.18115/D5G045.

This page titled 8.1: Data for This Chapter is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 8.1: Data for This Chapter by Steven J Luck is licensed CC BY 4.0.

8.2: Overview

In both ERPLAB Classic and ERPLAB Studio, artifact rejection involves two steps. In the first step, we flag epochs that contain artifacts. Then, during the averaging process, we simply don't include the flagged epochs. When Javier Lopez-Calderon and I designed the artifact rejection process in ERPLAB, Javier suggested that we should refer to the process of flagging epochs containing artifacts as *artifact detection*, because those epochs aren't actually deleted from the EEG dataset. And we should refer to the exclusion of flagged artifacts during averaging as *artifact rejection*. I thought this was a great idea, so that's the terminology used by ERPLAB.

EEGLAB and ERPLAB also contain a separate set of routines that actually *delete* problematic segments of data from the continuous EEG. These routines are primarily used as a preprocessing step in the artifact correction process, as will be described in the next chapter. In the present chapter, the term *artifact rejection* will be used only in the context of epoched EEG data.

If you don't already know how the major types of artifacts are generated and what they look like in EEG recordings, I recommend that you watch this video and this video from the online Introduction to ERPs course. For an even deeper dive, I recommend Chapter 6 in Luck (2014). The goal of the present chapter is to make the theoretical background from these sources more concrete and demonstrate the practical issues that arise in real data.

We'll focus on data from a few example participants whom I selected not because they had "good" data but because they were quite challenging. The data will come from two of the ERP CORE experiments (Kappenman et al., 2021), one looking at the mismatch negativity (MMN) and one looking at the N2pc component. The MMN paradigm was described in the previous chapter, and the N2pc paradigm will be described later in the present chapter. We'll mainly consider blinks and eye movements, because they're the most common large artifacts, but the exercises will also teach you general principles that you can use for other kinds of artifacts and other types of experiments.

Artifact correction has many advantages over artifact rejection, and it will be covered in the next chapter. In almost all cases, however, I recommend combining rejection and correction. Also, the problems created by artifacts are the same whether you're using rejection or correction, and those problems will be described in the present chapter.

Organization of the Chapter

Artifact detection is conceptually simple, but it requires a lot of decisions, and you need to know how to make the best decisions to achieve the best possible data. As a result, this chapter is pretty long. Here's the overall structure:

- The first part of the chapter describes three main problems that are typically addressed by artifact rejection and provides an overview of the detection+rejection process.
- The second part of the chapter takes you through a series of exercises in which you'll see how to detect and reject blinks, eye movements, and other miscellaneous artifacts in the context of the MMN experiment.
- The last part of the chapter takes you through exercises that teach you how to detect and reject small but consistent eye movements, which are especially problematic in experiments with lateralized target stimuli (especially N2pc and CDA experiments) or lateralized responses (mainly LRP experiments). If you don't conduct experiments of this sort, you can skip this part of the chapter.

The exercises focus on data from only two participants in each experiment. I recommend that you also look at the data from additional participants and repeat the artifact detection procedures with those participants. Practice makes perfect! You can find the data from additional participants in the MMN_Data and N2pc_Data folders inside the Chapter_8 folder.

Keep in mind that we did nothing to try to minimize blinking when designing and running these experiments because we knew we would mainly use artifact correction to deal with blinks. As a result, many participants blinked on a large proportion of trials. We would have needed to exclude many of these participants if we had rejected rather than corrected blinks.

This page titled 8.2: Overview is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 8.2: Overview by Steven J Luck is licensed CC BY 4.0.

8.3: Background- Why Do We Reject Artifacts?

Let's start by asking why we should exclude epochs containing artifacts. You might think this is a dumb question. Obviously, we don't want any artifacts in the epochs that we will be using to make our averaged ERPs! However, every single time point in every scalp EEG recording in human history contains artifactual activity. That is, the scalp EEG signal is always a mixture of brain activity, non-neural biological signals (e.g., skin potentials, EMG), and non-biological signals (e.g., line noise from nearby electrical devices). If we rejected every epoch containing an artifact, we wouldn't have any data left.

We therefore reject epochs that contain *problematic* artifacts, defined as artifacts that interfere with the fundamental goal described in the previous chapter: **accurately answering the scientific question that the experiment was designed to address**. There are three common ways in which artifacts can be problematic from this perspective:

1. **Reduced Statistical Power**. Artifacts add noise to the data, reducing the signal-to-noise ratio (SNR) of our averaged ERPs. This makes our amplitude and latency measurements less precise, which in turn decreases our statistical power. However, when we reject epochs containing artifacts, we have fewer epochs in our averages, and that also makes the averages noisier and decreases our power. As a result, we need to balance the need to eliminate epochs with large artifacts with the need to include as many epochs as possible. This is true even if artifact correction is used to deal with blinks and eye movements, because rejection will be needed for other artifacts (see Zhang et al., 2024c). Interestingly, the reduced number of trials resulting from artifact correction often outweighs the benefits of eliminating noisy data when multivariate decoding methods are applied to ERPs (Zhang & Luck, in press), so I don't recommend using artifact rejection in most decoding studies.
2. **Systematic Confounds**. Artifacts can produce systematic confounds in our studies. For example, participants in the ERP CORE N400 experiment blinked more for semantically related targets than for semantically unrelated targets, artificially exaggerating the size of the relatedness effect and distorting its scalp distribution (Zhang & Luck, in press). In addition, blinks were slightly but significantly more common for cars than for faces in the ERP CORE N400 experiment.
3. **Sensory Input Problems**. In visual experiments, EOG artifacts can indicate a problem with the sensory input. For example, if a participant blinks near the time of a stimulus and the eyes are closed when the stimulus appears, this means that the stimulus wasn't actually seen by the participant. Similarly, a deflection in the horizontal EOG can mean that the eyes weren't pointed at the center of the display. The first exercises in this chapter will use data from an auditory experiment so that we won't need to deal with this issue initially. However, we'll switch to a visual experiment in the last part of the chapter to examine how ocular artifacts might alter the sensory input.

Artifact correction can be much better than artifact rejection for addressing the problem of reduced statistical power, because we get to keep all of our epochs. Correction can also help with systematic confounds, but only to the extent that the correction fully removes the artifacts and doesn't produce any new artifacts. For example, if correction reduces the blinks by 99%, the remaining blink activity would still be 1-2 μV in the frontal channels (because uncorrected blinks are typically 100-200 μV in these channels). Indeed, we found that ICA did not fully eliminate the difference in blink-related activity between related and unrelated targets at frontal electrode sites in the ERP CORE N400 experiment (Zhang & Luck, in press). Fortunately, the amount of residual artifact was small enough to be negligible at the CPz electrode site where the N400 was measured. Artifact correction doesn't help at all with sensory input problems. For example, if participants are looking leftward in one condition and rightward in another condition, correcting for the EOG voltage produced by the eye movements doesn't eliminate the confound of a different sensory input in the two conditions.

For these reasons, I recommend combining artifact correction and artifact rejection for most experiments. You can use correction to minimize the noise produced by blinks (and certain other artifacts, as discussed in Chapter 6 in Luck, 2014). And then you can use rejection to eliminate epochs with blinks or eye movements near the time of the stimulus (for visual experiments) and to eliminate epochs that contain large artifacts that are not easily corrected (e.g., occasional EMG bursts).

When we're using rejection to deal with reduced statistical power, we would ideally have an algorithm that determines which epochs should be removed to best balance the benefits of eliminating noisy epochs with the cost of reducing the number of epochs that are included in our averages. There are several methods that take this approach (e.g., Jas et al., 2017; Nolan et al., 2010; Talsma, 2008). However, they try to optimize the signal-to-noise ratio in a generic sense, which may not actually maximize statistical power for the specific amplitude or latency measurement that you will be using to answer your scientific question.

The standardized measurement error (SME) was specifically designed to quantify the data quality for your specific amplitude or latency measure and is directly related to statistical power (Luck et al., 2021). The SME can therefore be used to determine the artifact detection parameters that will lead to the best power. At this moment, ERPLAB doesn't include an automated approach

for determining which trials should be rejected to minimize the SME, but you can manually check the SME to compare different artifact detection parameters. We'll use this approach in several of the exercises later in this chapter (see Zhang & Luck, in press for a more detailed and rigorous approach).

Keep in mind, however, that low noise isn't the only consideration. For example, imagine that a participant blinked on every trial. This would be very consistent, which would lead to a low SME (because the SME reflects the amount of trial-to-trial variability in the data). However, the resulting averaged waveforms would mainly contain blink activity instead of ERPs, which could lead to completely incorrect conclusions. So, you need to consider potential confounds as well as the data quality when selecting artifact detection parameters.

You should also keep in mind that the SME quantifies the data quality of the averaged ERPs (which is, of course, influenced by the noise in the EEG). As a result, the SME depends on the number of trials being averaged. That's a good thing, because the number of trials can have a big impact on your statistical power (Baker et al., 2020).

This page titled 8.3: Background- Why Do We Reject Artifacts? is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 8.3: Background- Why Do We Reject Artifacts? by Steven J Luck is licensed CC BY 4.0.

8.4: Background- The General Approach

The general artifact detection/rejection procedure is pretty straightforward. For each participant, you apply an artifact detection algorithm to the epoched EEG data. That algorithm determines which epochs contain artifacts, and those epochs are "marked" or "flagged" in the EventList. When you compute averaged ERPs, the flagged epochs are simply excluded from the averages.

There are two main classes of approaches to determining which epochs should be flagged. The approach I prefer involves knowing as much as possible about the nature of the artifacts (e.g., the typical waveshape and scalp distribution of a blink) and designing algorithms that are tailored to those artifacts. The other approach involves asking which epochs are extreme or unusual in a statistical sense. I don't like this statistical approach as much because it's not clear that "weird" epochs are necessarily problematic. How many movies have you seen about high school students in which the "popular" students rejected kids who seemed "weird" but were actually quite delightful? I just don't like the idea of rejecting trials that seem "weird" but might actually be delightful.

I suspect that the two approaches actually end up mostly flagging the same epochs for rejection, so it may not matter which approach you use in the end. And the most important factor when deciding what approach to take is to have a clear understanding of the ultimate goal of artifact rejection. As described in the previous chapter, **the ultimate goal is to accurately answer the scientific question that the experiment was designed to address**. So, go ahead and use statistical approaches to flagging epochs for rejection if that does a better job of leading you to this goal. More broadly, you should feel free to ignore any of my specific pieces of advice if you have a better way of accurately answering your scientific questions.

As described in detail in Chapter 6 of Luck (2014), I recommend setting the artifact detection parameters individually for each participant. In the present chapter, I will show you how to select appropriate parameters manually. There are also completely automated approaches to selecting the parameters (e.g., Jas et al., 2017; Nolan et al., 2010). I haven't used those approaches myself, but they seem fairly reasonable. However, many people who use these approaches on a regular basis recommend verifying that the parameters are working well and not just accepting them blindly. So, these approaches end up not being fully automatic. An ERP Boot Camp participant, Charisse Pickron, suggested another excellent use for the automated algorithms: When you're first learning to set artifact detection parameters, you can check your parameters against the automated parameters so that you have more confidence in the parameters that you've set.

Some participants have so many artifacts that an insufficient number of trials remains to create clean averaged ERP waveforms. The standard procedure is to exclude those participants from the final analyses. However, you must have an objective, a priori criterion for exclusion. Otherwise, you will likely bias your results (as explained in the text box below). In my lab's basic science research, we always exclude participants if more than 25% of trials are rejected because of artifacts (aggregated across conditions). In our research on schizophrenia, where the data are noisier and the participants are much more difficult and expensive to recruit, we exclude participants if more than 50% of trials are rejected. We apply these criteria rigidly in every study, without fail. A different criterion might make sense in your research. Just make sure that the criterion is objective and determined before you see the data.

Although this chapter focuses on detecting and rejecting artifacts, I would like to encourage you to start thinking about artifacts before you record the EEG. This advice follows from something I call *Hansen's Axiom*: "There is no substitute for clean data" (see Luck, 2014). It's much better to minimize artifacts during the recording instead of trying to reject or correct them afterward. Strategies for minimizing artifacts are described in Chapter 6 of Luck (2014).

Excluding Participants is Dangerous!

Imagine that you run an experiment, and your key statistical analysis yields a *p* value of .06 (the most hated number in science!). You had spent two years running the study, and the effect is going in the predicted direction, but you know you can't publish it if the effect isn't statistically significant. Given the millions of steps involved in an ERP experiment, you go back through your data and analyses to make sure there wasn't an error. And imagine you find that 80% of the trials were rejected for one of the participants, leading to incredibly noisy data. You would (very reasonably) conclude that this participant should not have been included in the final analyses. So, you repeat the analyses without this participant, and now the *p* value is .03. Hallelujah! You can now publish this important study.

Now imagine that the *p* value was originally .03, so you have no reason to go back through all the data. And imagine that your final sample included a participant with 80% of trials rejected and very noisy data. And further imagine that excluding this participant would lead to a *p* value of .06. But because the effect was significant in the initial analysis, you had no reason to go back through the data, so you wouldn't notice that this participant should have been excluded. And even if you did, would you really have the intestinal fortitude to exclude the participant, even though this means that your *p* value is now .06?

This example shows why you need an a priori criterion for excluding participants. If you decide whom to exclude after you've seen the results of the experiment, you're more likely to notice and exclude participants when it makes your *p* value better (because it was >.05 before you excluded participants) than when it makes your *p* value worse (because you don't notice participants who should be excluded when $p < .05$). As a result, this creates a bias to find $p < .05$ even when there is no true effect. So, you should develop an a priori criterion for excluding participants before you see the results and always follow that criterion.

And you should make it clear that you used an a priori criterion when you write up your results for publication. In my lab, for example, we always include a statement such as "We always excluded participants for whom more than 25% of trials were rejected; three participants were excluded for this reason in the present study." We even include a statement like this when none of the participants exceed this criterion, writing "... no participants were excluded for this reason in the present study." That way, readers can see that we really do have an a priori rule that we always follow.

This page titled 8.4: Background- The General Approach is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 8.4: Background- The General Approach by Steven J Luck is licensed CC BY 4.0.

8.5: Exercise- Simple Blink Detection

It's now time to see how artifact detection works in actual data. We'll start by detecting blinks. Although I generally recommend using ICA-based artifact correction rather than artifact detection+rejection to deal with blinks, it's useful to learn the general principles of artifact detection+rejection with blinks because they are big and easy to detect.

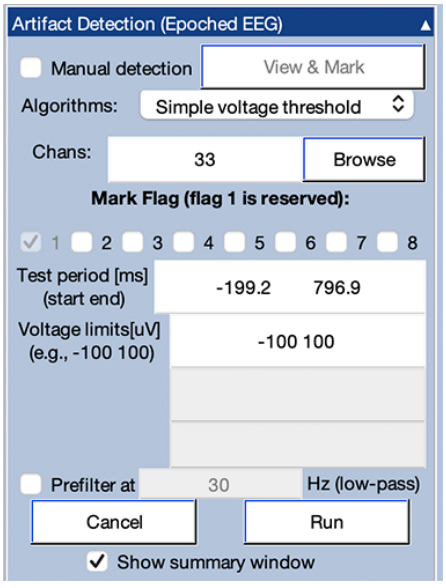
We'll begin with the most common artifact detection algorithm, which simply asks whether the voltage falls outside a specified range at any point in the epoch for a given channel. This is a pretty primitive algorithm, and the following exercise will show you a much better approach.

Start by resetting everything with the **Reset** button. Then use the **EEGsets** panel to load the data from Subject #1 in the MMN experiment, which is in the file named **1_MMN_preprocessed_interp**. This is the same dataset that we examined in the previous chapter, after the C5 channel was interpolated and BINLISTER was run to assign events to bins. Artifact detection operates on epoched data, so use the **Extract Bin-Based Epochs** to extract epochs using an epoch of -200 to 800 ms and **Pre** as the baseline. Use the recommended name for the resulting EEGset (**1_MMN_preprocessed_interp_be**).

Now go to the **Artifact Detection** panel and set it up as shown in Screenshot 8.1. Start by setting the **Algorithm** to **Simple voltage threshold**. This simple algorithm flags an epoch if the voltage exceeds a threshold value at any time within a specified time window in any of the selected channels. We're going to look for blinks in this first exercise, and they will be largest in the VEOG-bipolar channel, so there's no point in looking in any other channel. Click the Browse button next to the Chans text box and select the VEOG-bipolar channel (channel 33). This will cause **33** to appear in the **Chans** box. Make sure that the **Voltage limits** box is set to **-100 100**, which indicates that an epoch should be flagged for rejection if the voltage is more negative than -100 µV or more positive than +100 µV at any time in this channel.

You might wonder why the default **Test period** is set to **-199.2 796.9** rather than **-200 800** (which is what you specified for the epoch). The answer is that the sampling rate for this EEGset is 256 Hz. As a result, we have a sample every 3.90625 ms, and we don't have samples at exactly -200 and +800 ms. In the previous chapters, I downsampled to 200 Hz instead, yielding a sample every 5 ms. But I thought it was time for you to see what happens when the sampling period isn't a nice round number.

Note that the checkbox for **1** is checked in the Mark Flag section. There are 8 different artifact detection flags, and flag 1 must be set for any artifact. I'll discuss how to use the other flags later. Click **Run** to apply the algorithm to the selected dataset, and name the resulting EEGset **1_MMN_preprocessed_interp_be_ar100**.

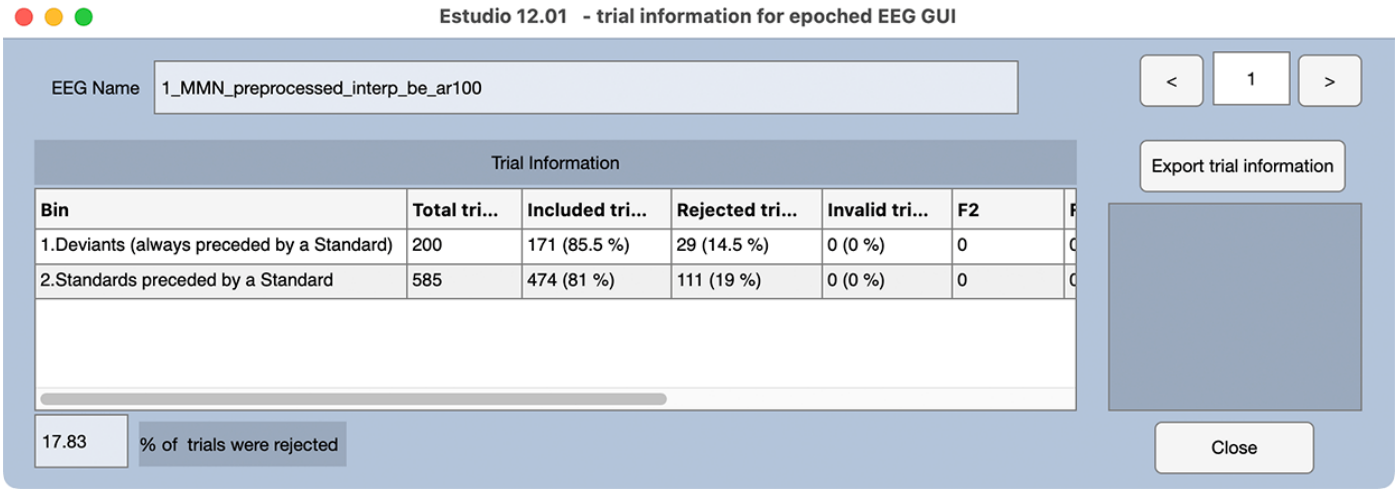


Screenshot 8.1

Once the algorithm has finished, you will see the window shown in Screenshot 8.2, which summarizes the artifact detection

results. You can see the number and percentage of epochs in which artifacts were detected in each bin and the total across bins. In most cases, I mainly worry about the total (because the percentage is much more meaningful when based on a large number of trials). You can see that 17.83% of epochs were rejected. That's reasonable. As noted above, my lab always throws out any participants for whom more than 25% of epochs were rejected, so this participant would be retained.

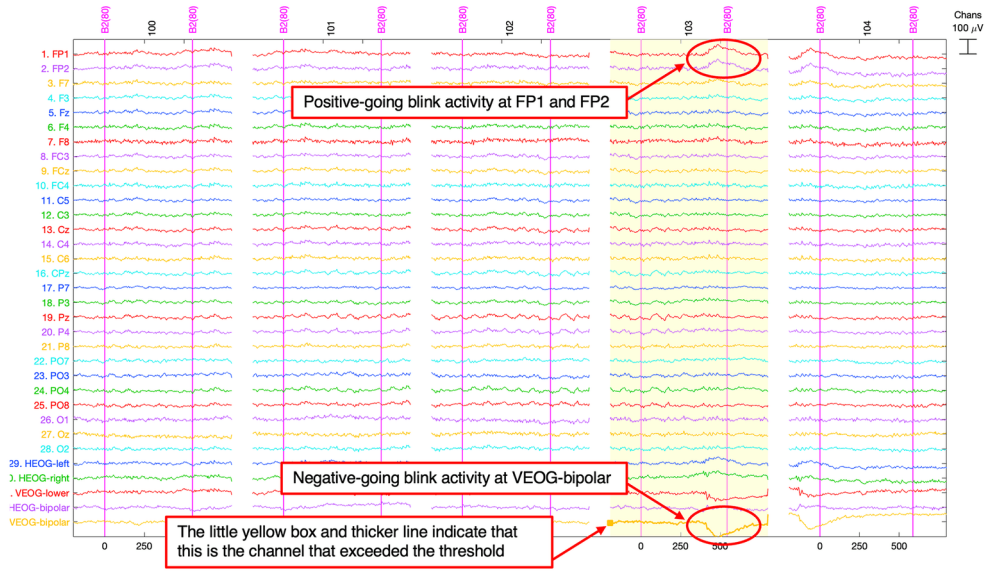
You can now close this window. If you want to see this information again later, you can go to the **Artifact Info & Tools** panel and click the **Artifact Summary** button.



Screenshot 8.2

Now let's scroll through the EEGset and see how well the algorithm performed at flagging epochs with blinks and not flagging epochs without blinks. I recommend setting the **Vertical scale** to 100 in **Plot Settings**.

At first you won't see any blinks, because this participant had beautiful EEG in the beginning of the session. When you get to epochs 100-105, you will see that epoch 103 has a yellow background (see Screenshot 8.2). This means that the epoch has been marked for rejection. The waveform for the offending channel is drawn with a thicker line width and has a little box before the first time point of the epoch. You can see a classic blink shape and scalp distribution in this epoch. Success!



Screenshot 8.3

But this is immediately followed by a failure in Epoch 104. Because we have a stimulus every 500 ms, but each epoch lasts for 1000 ms, the initial part of one epoch is the same as the latter part of the previous epoch. So, the blink that peaked just before

500 ms in Epoch 103 appears just before time zero in Epoch 104. Because the blink is during the baseline in Epoch 104, the baseline correction procedure reduced the maximum voltage during the epoch, and the blink did not exceed the threshold for detection.

Scroll one more page (to epochs 105-109). You'll notice a large muscle burst in Epochs 107 and 108 that isn't flagged. But that's OK—we're looking for blinks right now, and we'll test for other artifacts later. If you keep scrolling, you'll also see a blink that appears in the poststimulus period of Epoch 121 and in the prestimulus period of Epoch 122. This blink was larger than the one in Epochs 103 and 104, and the blink was successfully detected in both Epochs 121 and 122. Keep scrolling and looking for successes and failures of the detection algorithm. The blink appearing in Epochs 162 and 163 was also successfully detected. However, the blink that appears in Epochs 169 and 170 was missed in Epoch 170.

If you keep scrolling, you'll also see that high-frequency noise (almost certainly EMG) caused Epoch 463 to be flagged. You can tell that there was no blink in this epoch because there was no positive-going voltage in Fp1 and Fp2, just a small negative-going voltage in VEOG-lower combined with some high-frequency noise. There is no reason to reject this epoch: This high-frequency noise burst was narrowly localized to the EOG channels and is unlikely to impact our scalp EEG recordings. Epochs 525 and 526 are also unnecessarily flagged for rejection. In these epochs, a combination of a slow, non-blink-like voltage deflection and high-frequency noise in Fp2 (but without an opposite-polarity deflection in VEOG-lower) produced a large enough voltage in VEOG-bipolar for the voltage to exceed our ±100 μV threshold.

You should now save the EEGset with the artifacts flagged by clicking the **Save EEG** button in the **EEGsets** panel. It will automatically give the file the same name as the EEGset but with a .set extension (**1_MMN_preprocessed_interp_be_ar100.set**). We'll need it for a later exercise.

This page titled 8.5: Exercise- Simple Blink Detection is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 8.5: Exercise- Simple Blink Detection by Steven J Luck is licensed CC BY 4.0.

8.6: Exercise- Adjusting the Threshold

In this exercise, we'll see how adjusting the threshold changes which epochs are flagged for rejection. Let's start by seeing if we can detect some of the blinks that were missed with our ±100 μV threshold. Select **1_MMN_preprocessed_interp_be** in the **EEGsets** panel, and then change the **Voltage limits** to **-50 50** in the **Artifact Detection** panel to indicate that an epoch should be flagged for rejection if the voltage is more negative than -50 μV or more positive than +50 μV at any time in the VEOG-bipolar channel. Click **Run** to run the artifact detection routine and save the resulting EEGset with the name **1_MMN_preprocessed_interp_be_ar50**.

You should now see a window summarizing the number of rejected trials. Whereas 17.83% of epochs were flagged when our threshold was ±100 μV, now 42.93% have been flagged. If we were to use this ±50 μV threshold, this participant would need to be excluded from the final analyses (because my lab excludes participants if more than 25% of trials were rejected). Obviously, you don't want to exclude participants if you don't have to, so let's see if we really want to use this threshold.

If you scroll through the epochs, you'll see that the blinks in Epochs 104 and 170, which were missed with the ±100 threshold, have been detected with this threshold. That's the good news. But if you keep scrolling, you'll see the bad news: Many epochs without a clear blink are now flagged for rejection (e.g., Epochs 408, 424, 432, and 435-437). In general, decreasing the threshold for rejection increases our *hit rate* (the proportion of blinks that were detected) but also increases our *false alarm rate* (the proportion of non-blink epochs that are flagged for rejection). [Remember: You can go directly to a particular epoch by typing the number in the text box under the lower left corner of the plotting area and then hitting return/enter.]

Now let's try increasing our threshold to avoid flagging epochs 463, 525, and 526, which were unnecessarily flagged for rejection with our original threshold of ±100 μV. Select **1_MMN_preprocessed_interp_be** in the **EEGsets** panel, and run the artifact detection routine using voltage limits of **-150 150**. You should save the resulting EEGset with the name **1_MMN_preprocessed_interp_be_ar150** (we will need it in a later exercise).

The percentage of flagged trials has now dropped to 11.34%. That's good insofar as increasing the number of accepted trials will increase our signal-to-noise ratio (all else being equal). But it might be bad if a lot of blinks have escaped detection.

If you scroll through the data, you'll see that Epochs 463, 525, and 526 are no longer flagged for rejection, which is good. However, several clear blinks have been missed (e.g., Epochs 103, 191, 201). In general, increasing the threshold for rejection decreases the hit rate but also decreases the false alarm rate.

The take-home message of this exercise is that adjusting the threshold impacts both the hit rate and the false alarm rate, making one better and the other worse. You'll need to choose a threshold that balances the hit rate and false alarm rate in a way that best helps you achieve the fundamental goal, which is to accurately answer the scientific question that the experiment is designed to address. Is that goal best met by ensuring that all epochs with blinks are rejected, even if this means rejecting some perfectly fine epochs? Or is the goal best met by optimizing the number of included epochs, even if a few blinks escape rejection?

The answer will depend on the nature of your scientific question, the details of your experimental design, and the nature of the artifacts in your data. In particular, if blinks differ systematically across bins (especially in the time range of the ERP components of interest), then you will usually need to make sure that the vast majority are rejected to avoid confounds. And if you have a reasonably large number of trials, throwing out a few trials without blinks won't really change your signal-to-noise ratio very much (see the text box below). So, in most cases, I recommend erring on the side of throwing out too many trials rather than allowing some large artifacts to remain in the data. We will look at this issue more formally in a later section, using the standardized measurement error (SME) to quantify the data quality resulting from the different thresholds you've tried.

Also, as you'll see in a later exercise, you can both increase your hit rate and decrease your false alarm rate by choosing a better algorithm for determining which epochs contain artifacts. The simple voltage threshold we've used in this example is a very primitive way of detecting blinks, and I'm always amazed that many software packages don't provide better algorithms.

Don’t Stress About Rejecting a Few Extra Trials

It’s easy to get stressed out about excluding a bunch of trials because of artifacts. Is excluding 20% or 50% of trials going to cause a 20% or 50% reduction in your data quality? It turns out that excluding trials has a smaller impact on data quality than you might expect.

This is because the signal-to-noise ratio (SNR) increases as a function of the square root of the number of trials. This square root rule is really annoying when you’re designing your experiment, because doubling the number of trials only increases your SNR by 41% (because sqrt(2) = 1.41). But the same rule means that you don’t lose very much SNR when you have to exclude some trials.

As an example, imagine that your single-trial SNR is 1:2 or 0.5 (i.e., your signal is half as big as your noise in the raw EEG epochs). If you average together 100 trials, the resulting SNR is 0.5 x sqrt(100) = 5. Now imagine that you have to exclude 20 trials because of artifacts. Now your SNR is 0.5 x sqrt(80) = 4.47. That is, you’ve decreased the number of trials by 20%, but your SNR has dropped by only about 10%.

Now imagine that there are actual artifacts on 20 of the 100 trials, but your rejection threshold is set a little low, and you end up rejecting 5 extra trials in addition to the 20 trials with true artifacts. As a result, you have an average of only 75 trials instead of 80 trials. We already saw that the SNR with 80 trials would be 4.47. With 75 trials, it would be 0.5 x sqrt(75) = 4.33. That's a very small reduction. So, don't worry if you are rejecting a few trials that don't have artifacts.

Now let's think about what would happen if you set your rejection threshold a little too high, and now you are rejecting only 15 trials. Of the 85 trials in your average, 5 of them would have artifacts. And if those artifacts are blinks, they can be huge (200 µV at the frontal pole). Those trials that should have been rejected can really add a lot of noise to your data. So, you should worry more about not rejecting enough trials than rejecting too many.

This page titled 8.6: Exercise- Adjusting the Threshold is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 8.6: Exercise- Adjusting the Threshold by Steven J Luck is licensed CC BY 4.0.

8.7: Exercise- Manually Marking and Unmarking Artifacts

In many ERP studies of infants, the researchers are lucky if they get 50 trials from a participant. And they may get as few as 10 per condition. Every trial counts, so they want to be extra careful to ensure that the correct trials are marked for rejection. And because there are so few trials, it doesn't take long to do the artifact detection manually (by looking at each epoch and deciding whether it should be marked for rejection). In this exercise, you'll see how to perform this manual artifact detection in eStudio.

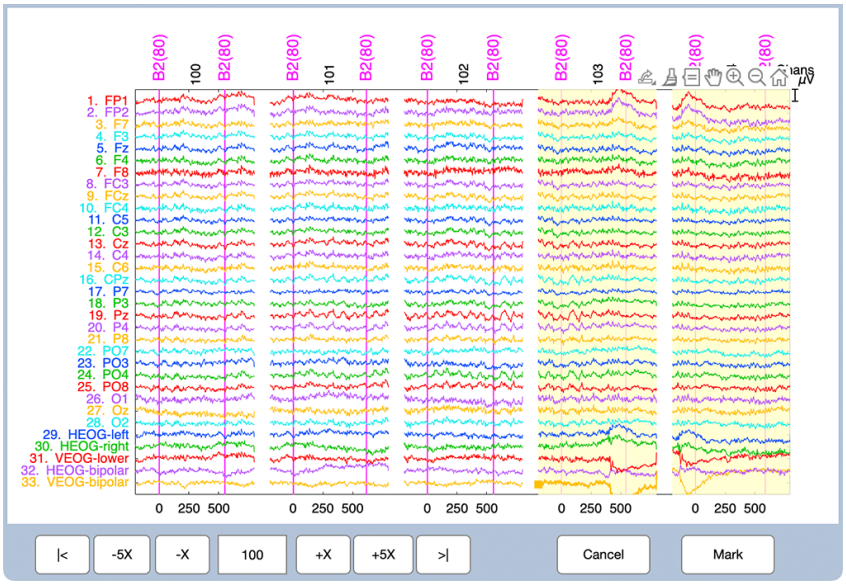
Another use case for manual rejection was brought to my attention by Rick Addante, when he was a graduate student here at UC Davis. He conducted an ERP study of people with dense amnesia resulting from brain damage (Adante et al., 2012). These individuals are very difficult to recruit, and his study ended up having only three amnesia patients and six carefully matched control participants. Although he had hundreds of trials for each participant, he needed to be super careful about artifact rejection because he had so few patients. And because he had a total of only nine participants, he could perform the artifact detection manually in a reasonable amount of time.

Let's see how it works. If you still have **1_MMN_preprocessed_interp_be_ar100** loaded in the **EEGsets** panel, select it now. If you don't have it loaded, load it now (if you didn't save it earlier, you can find a copy in the Extra Files folder). This file was created when you performed artifact detection on the VEOG-bipolar channel with a threshold of ±100 µV. Scroll to epochs 100-104 (or type **100** <return> in the text box in the middle of the scrolling controls). You should see that epoch 103 is marked for rejection, with a little square at the beginning of the VEOG-bipolar waveform indicating that this is the channel in which the artifact was detected.

Now go to the **Artifact Detection** panel and check the checkbox at the top labeled **Manual detection,** and then click the **View & Mark** button. If you wait patiently for a few seconds, a new window will pop up that shows the EEG epochs. You will use this window for toggling the artifact detection flags. That is, if you see an unflagged epoch that should be rejected, you can mark it for rejection by clicking on it. And if you see a flagged epoch that should not be rejected, you can click on it to unmark it.

Note that the main eStudio window is "frozen" while this manual detection window is open, so you can't access any of the panels in the main window until you close the manual detection window.

Scroll to epochs 100-104 in the manual window by typing **100** <return> in the text box in the middle of the scrolling controls for this window. You should again see that epoch 103 is marked for rejection. Notice that epoch 104 also has a clear blink, but it was just a little too small to be detected with a threshold of ±100. You can simply click anywhere in that epoch to mark it for rejection. You can tell that it's marked for rejection because its background will turn yellow, as shown in Screenshot 8.4.



Screenshot 8.4

Now scroll to epochs 105-109 by clicking the **+X** button in the scrolling controls. You'll see that epochs 107 and 108 have large EMG bursts. I don't ordinarily reject trials like those, because the noise can be handled by a 20 Hz low-pass filter. However, for the sake of this exercise, go ahead and click on both of those epochs to mark them for rejection. They should now have a yellow background.

You can also click on a marked epoch to unmark it. To see this, scroll ahead to epochs 120-24. You should see that epochs 121 and 122 are marked for rejection. Click on those two epochs to unmark them. You can tell that they've been unmarked because they should no longer have a yellow background.

To save the changed marks in a new EEGset, click on the **Mark** button in the lower right corner of the window. This will bring up the standard window for naming the new EEGset; you can use the suggested name of **1_MMN_preprocessed_interp_be_ar100_manmark**.

In the main eStudio window, you should see that epoch 104 now has a yellow background, indicating that it has been marked for rejection. The waveforms for every channel in this epoch have the little square at the beginning because eStudio did not know which particular channel you were trying to mark for rejection.

In the main eStudio window, type **105** <return> to see epochs 105-109. You should see that epochs 107 and 108 (with the large EMG) have now been marked for rejection. And if you go to epochs 120-124, you should see that epochs 121 and 122 are no longer marked for rejection. [Keep in mind that we unmarked these epochs solely as a demonstration, not because you would ordinarily want to keep these two epochs.]

This page titled 8.7: Exercise- Manually Marking and Unmarking Artifacts is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck.

8.8: An Iterative Approach to Setting Parameters

Many researchers use a preset artifact detection threshold for all participants, but this is not usually optimal. A threshold that works well for one participant might fail to flag all the blinks for another participant, and it might lead many non-blink epochs to be flagged in yet another participant. I therefore recommend the interactive and iterative approach shown below in Figure 8.1. It involves starting with an initial set of best-guess parameters, then seeing whether these parameters adequately flag the epochs that should be rejected, and then adjusting the parameters as necessary until you are satisfied. As will be described in the next section, it also involves using the standardized measurement error (SME) to help you determine which parameters lead to the best balance of eliminating noisy trials while still having enough trials to obtain a good averaged ERP waveform. We will see how to check the SME in the next section.

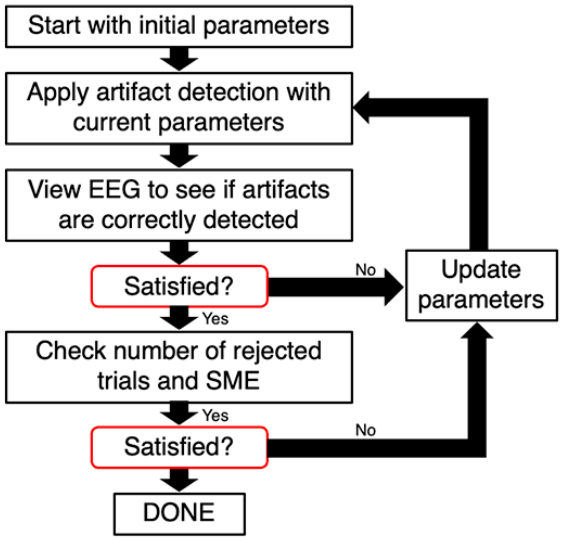


Figure 8.1. Iterative procedure for setting artifact detection parameters. If no satisfactory parameters can be found without exceeding the maximal allowable percentage of trials with artifacts, then the participant must be excluded from the final analyses.

The threshold is not the only parameter you might adjust. For example, if you are using artifact correction to remove the artifactual voltages produced by blinks but you still want to reject blinks that occur at the time of a stimulus (which make it so that the participant can't see the stimulus), you can adjust the time window used to look for blinks. And as we will see in later sections, other artifact detection algorithms have additional parameters that control their operation. But the threshold is usually the main one you will adjust individually for each participant.

Make sure you keep a record of the parameters you choose for each participant. I recommend using a spreadsheet for this. The example script at the end of the chapter shows you how a script can read the artifact detection parameters from a spreadsheet and then perform the artifact detection with these parameters. That will keep you from having a meltdown when Reviewer 2 requests an analysis that requires performing artifact detection again. In fact, it's a good idea to set the parameters manually and then immediately reprocess the data using a script with those parameters. Processing the data manually is important so that you see the data at each step of the process, which can alert you to problems in the data for a given participant that require special processing. However, it's easy to make a mistake when you're processing data by pointing and clicking in a GUI, and scripts are valuable because they avoid these point-and-click errors. You can get the best of both worlds if you start by manually processing the data in the eStudio GUI, manually determining the rejection parameters, saving those parameters in a spreadsheet, and then reanalyzing the data using a script that reads these parameters.I find this iterative approach is reasonably quick and effective for most kinds of experiments (5-10 minutes for most participants once you've become well practiced). However, other approaches may be better in certain cases.

For example, if you are doing a huge analysis with hundreds of datasets (e.g., datasets you obtained from online repositories), you might want to use one of the algorithms that automatically set the parameters for each participant (e.g., Jas et al., 2017; Nolan et al., 2010; Talsma, 2008). However, you should still do a quick visual verification for each participant to make sure that the parameters are reasonable.

Another case where a different approach is warranted arises when you have a small number of trials per participant (as in many infant studies) or a small number of highly valuable participants (as in some studies of lesion patients). In these cases, it is worth taking a manual approach to artifact detection because every epoch is important, and there aren't a lot of epochs to check. This was demonstrated in the preceding section.

Will this lead to bias?

You might be concerned that manually setting the artifact detection parameters might somehow bias the data. This can indeed be a legitimate concern. For example, if you set a more lenient threshold for participants in Group A than for participants in Group B, then more blinks may escape detection in Group A, and the voltages from these extra blinks in group A may create an artifactual difference in the ERPs between the two groups. There is a simple solution to this problem: The person who is setting the artifact detection parameters should be blind to group membership. This is what my lab does in our research comparing people with schizophrenia to matched control participants.

This potential for artifactual differences is mainly a problem in studies comparing different groups of participants (or different ages, etc.). In studies that solely examine within-participant effects in a single group of participants, it is unlikely that the results could be biased by the manual setting of artifact detection parameters. This is because the same parameters are used for all conditions. That is, if the rejection criterion is too lenient, this will be true for all the conditions of the experiment. I suppose that a truly devious researcher could play around with the threshold and try to find one that maximizes the difference between conditions in a given participant. But this would have to be a conscious decision to fudge the results. If someone wants to fudge the results, this would be a very difficult and time-consuming way to do it. So I don't worry that the manual setting of artifact detection parameters is going to bias the results in studies of within-participant effects.

This page titled 8.8: An Iterative Approach to Setting Parameters is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 8.7: An Iterative Approach to Setting Parameters by Steven J Luck is licensed CC BY 4.0.

8.9: Exercise- Effects of Artifact Rejection on Data Quality

Rejecting trials with large artifacts can improve your data quality, because these artifacts cause large trial-to-trial variation in ERP amplitudes or latencies. However, if we are too aggressive in rejecting trials (i.e., our threshold for detection is very low), we might end up excluding so many trials that we end up making our data quality worse rather than better. Thus, there is a tradeoff between getting rid of noise (which is good) and reducing the number of trials (which is bad).

To determine whether the benefit of less noise outweighs the cost of fewer trials, the procedure for setting parameters that I described in the preceding section includes checking the standardized measurement error (SME) as part of the process. The SME is influenced by both the amount of noise in the included trials and the total number of trials included. In fact, the SME balances these factors in a way that exactly matches how they are balanced when you perform your statistical analyses. All else being equal, a lower SME means you have greater statistical power (a greater chance of finding that $p < .05$ for a real effect).

In the present section, we will see how this works with the MMN data you've been processing. To begin, reset everything with the **Reset** button, and then load the following EEGsets that you created earlier in the chapter:

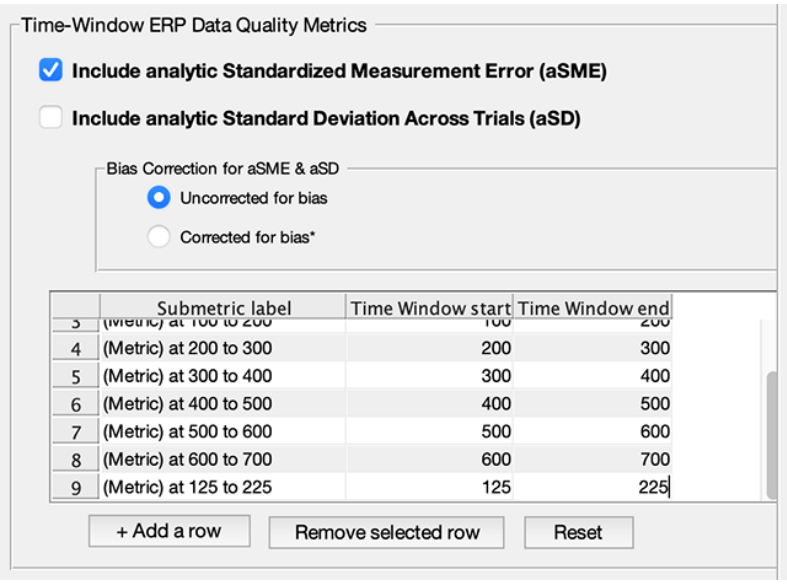
- 1_MMN_preprocessed_interp_be
- 1_MMN_preprocessed_interp_be_ar50
- 1_MMN_preprocessed_interp_be_ar100
- 1_MMN_preprocessed_interp_be_ar150

If you didn't save these files when you created them, you can find copies in the Extra Files folder inside the Chapter_8 folder.

These four EEGsets are equivalent except for the threshold that was used for artifact detection. We're going to use the **Compute Data Quality Metrics** panel to compute the aSME for each of those EEGsets. This panel computes the same aSME values that you would get by using the **Compute Averaged ERPs** panel, but without creating the ERPsets. When you're choosing a threshold for detection, you shouldn't look at the averaged ERP waveforms, because that might bias you to use a threshold that maximizes the effect of interest in the ERPs. By using the **Compute Data Quality Metrics** panel, you can select a threshold on the basis of data quality without seeing the ERP waveforms, which will avoid any unintentional biasing of your results.

In the ERP CORE, the MMN is measured as the mean voltage between 125 and 225 ms. We are going to tell the software to compute the aSME for this time range.

To get started, go to the **Compute Data Quality Metrics** panel, select the radio button for **Custom** parameters, and click the **Options** button. A new window will appear that will allow you to select a custom time period for computing the aSME (see Screenshot 8.5). Click **+Add a row** to add a new time window. You may need to scroll down to see the new row. Change the submetric label to **(Metric) at 125 to 225** and change the **Time Window start** and **Time Window end** fields to **125** and **225**. Then click **Save** to get back to the panel.



Screenshot 8.5

Before you compute the data quality metrics, go to the **EEGsets** panel and select all four of the EEGsets. This will allow you to compute the data quality metrics for all four in one step. Now click **Run** in the **Compute Data Quality Metrics** panel to compute the aSME values.

The table for viewing the aSME values will then appear, showing the aSME values from Bin 1 for one of the four EEGsets. Select **1_MMN_preprocessed_interp_be** in the **Selected EEGset** menu near the top so that the aSME values are shown for this EEGset. No epochs were marked for rejection in this EEGset, so the aSME values show you how much noise you will have if you don't reject any blinks. Take a look at the aSME values for the 125-225 ms time window along the right side of the table. Let's focus on the relatively frontal electrode sites, where the blinks would be expected to be largest and therefore create the most trial-to-trial variation and hence the most noise.

You should see values that look like those in the **No Rejection** column in Figure 8.2 below. You can already see that the aSME values are worse (higher) for the FP1 and FP2 electrodes, which are closest to the eyes and therefore have the largest EOG voltages when the eyes blink.

	No Rejection	Threshold = 150	Threshold = 100	Threshold = 50
	125 : 225	125 : 225	125 : 225	125 : 225
FP1	1.0084	0.7487	0.7005	0.7946
FP2	1.0068	0.7695	0.7214	0.7881
F7	0.6092	0.5486	0.4849	0.5405
F3	0.6065	0.5403	0.5455	0.6823
Fz	0.6162	0.5769	0.5901	0.7126
F4	0.5676	0.5114	0.5116	0.5935
F8	0.5314	0.4940	0.5114	0.5796
FC3	0.5109	0.4867	0.4950	0.6109
FCz	0.5524	0.5349	0.5436	0.6497
FC4	0.4968	0.4798	0.4821	0.5294

Figure 8.2. aSME values for the 125-225 ms time window for different blink detection thresholds.

Now select **1_MMN_preprocessed_interp_be_ar150** in the **Selected EEGset** menu to see the aSME values when epochs exceeding a threshold of 150 were marked for rejection. You can see that the aSME values for this EEGset (labeled **Threshold = 150** in Figure 8.2) are quite a bit lower than the aSME values obtained without any rejection of blinks. This reduction in aSME is larger for electrodes closer to the eyes (e.g., FP1 and FP2) than for electrodes farther from the eyes (e.g., FC3 and FC4).

That makes sense, because rejecting trials with blinks should have the biggest impact at sites where the blinks are largest. At sites where the blinks are smaller, excluding epochs with blinks does eliminate trials with some noise, which is good, but it also reduces the number of trials per average, which is bad. These two factors almost perfectly balance each other at FC3 and FC4, leading to only a small reduction in aSME when epochs with large blinks are rejected. By contrast, the reduction in noise greatly outweighs the reduction in trials at FP1 and FP2, leading to a substantial reduction in aSME. Note that the aSME value takes into account both the amount of trial-to-trial variation (noise) and the number of trials, quantifying the overall noise level when both factors are considered.

Now select **1_MMN_preprocessed_interp_be_ar100** in the **Selected EEGset** menu to see the aSME values when epochs exceeding a threshold of 100 were marked for rejection (labeled **Threshold = 100** in Figure 8.2). With this threshold, you did a better job of marking all the epochs containing blinks, and the aSME values for this threshold are therefore lower than the aSME values for the threshold of 150, at least at the sites closest to the eyes.

Now select **1_MMN_preprocessed_interp_be_ar50** in the **Selected EEGset** menu to see the aSME values when epochs exceeding a threshold of 50 were marked for rejection (labeled **Threshold = 50** in Figure 8.2). With this threshold, every epoch with a blink was marked for rejection, but a lot of epochs that were a little noisy but did not contain a blink were also marked for rejection. You can see that the aSME values have now gone up compared to the threshold of 100. This means that the cost of excluding so many trials from the averaged ERPs is outweighing the benefit of having less noise on the trials that are included.

This demonstrates how you can check the aSME values for each of several thresholds to see which threshold leads to the cleanest data (which is a part of the procedure shown in the flowchart in Section 8.8). This takes some time, but you will get pretty fast at doing it once you have more experience.

If you use ICA-based artifact correction to deal with blinks, you won't need to worry about the rejection threshold for blinks. However, you will likely reject trials with other kinds of noise, as shown later in this chapter, and checking the aSME values will help you select a threshold that leads to the best possible data quality and the greatest chance of obtaining the "magic" $p < .05$ if there is a real effect in your data.

This page titled 8.9: Exercise- Effects of Artifact Rejection on Data Quality is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck.

8.10: Exercise- Better Blink Detection

In a previous exercise, we saw that changing the threshold for rejection affects both the hit rate (the proportion of artifacts detected) and the false alarm rate (the proportion of non-artifact trials that are flagged for rejection). If you change the threshold to make one better, this inevitably makes the other one worse. However, there is something you can change that can improve both the hit rate and the false alarm rate. Specifically, you can use an artifact detection algorithm that is better designed to isolate blinks from other kinds of voltage deflections.

The simple voltage threshold algorithm that we have used so far in this chapter is an overly simplistic way of detecting blinks. It treats any large voltage as a blink, not taking into account the shape of the blink waveforms. Depending on the threshold you choose, it will typically either flag some trials that don't contain blinks or miss some of the true blinks. We can improve blink detection by taking into account the fact that blinks are relatively short-term changes in voltage that typically last ~200 ms. In this exercise, we'll look at two artifact detection algorithms that take this into account and work much better for blink detection.

The first is called the *moving window peak-to-peak* algorithm, and it's illustrated in Figure 8.3. With default parameters and an epoch of -200 to 800 ms, this algorithm will start by finding the difference in amplitude between the most positive and most negative points (the *peak-to-peak* voltage) between -200 and 0 ms (a 200-ms window). Then, the window will move 100 ms to the right, and the algorithm will find the peak-to-peak voltage between -100 and 100 ms. The window will keep moving by 100 ms, finding the peak-to-peak amplitude from 0 to 200 ms, 100 to 300 ms, and so on until 600-800 ms. This yields 9 different peak-to-peak amplitudes. The largest of these 9 peak-to-peak amplitudes is then compared to the rejection threshold to determine if this epoch should be marked for rejection.

Figure 8.3A illustrates the application of this algorithm to a trial with a blink. You get a large peak-to-peak amplitude during the period of the blink because of the relatively sudden change in voltage. However, the algorithm isn't "fooled" by the slow drift shown in Figure 8.3.B.

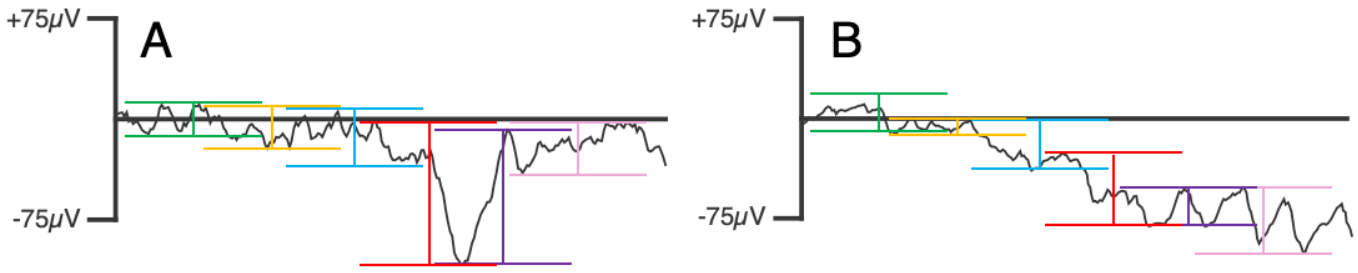
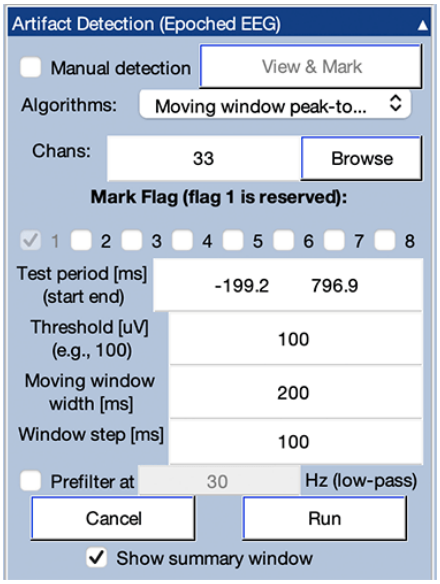


Figure 8.3. Moving window peak-to-peak algorithm. The peak-to-peak amplitude is determined in each window, and then the maximum of these values for a given epoch is compared with the rejection threshold.

Let's try it. Select **1_MMN_preprocessed_interp_be** in the **EEGsets** panel and then go to the **Artifact Detection** panel and select **Moving window peak-to-peak** in the **Algorithms** menu**.** Set it up as shown in Screenshot 8.6, with a window width of 200 ms and a window step of 100 ms (so that we get a 200-ms window every 100 ms). Set the threshold to **100** and the channel to **33** (VEOG-bipolar). Click **Run** to run the algorithm and accept the recommended name for the new ERPset (**1_MMN_preprocessed_interp_be_ar**).



Screenshot 8.6

A summary window will appear indicating that a total of 28.15% of trials have been flagged for rejection across the two bins. That's a lot more than we had with the absolute voltage threshold; we'll discuss the reasons for that in a moment.

If you scroll through the data starting at epoch 100, you'll see that every clear blink has now been flagged (including Epochs 104 and 170, which were missed by the absolute threshold algorithm). However, some trials with muscle noise have now been flagged for rejection (e.g., Epochs 107 and 108). Here's why: Imagine that the muscle noise causes the voltage to vary from -55 to +55 µV between 200 and 400 ms. This doesn't exceed the absolute threshold of ±100 µV, but it creates a peak-to-peak amplitude of 110 µV, exceeding our threshold for the peak-to-peak amplitude. One way to solve this would be to increase the threshold to something like 120 µV. However, this would cause us to start missing real blinks.

Another approach would be to apply a low-pass filter prior to artifact detection. Let's give that a try. Select **1_MMN_preprocessed_interp_be** again in the **EEGsets** panel and then go back to the **Artifact Detection** panel and but check the box labeled **Prefilter at** and enter a cutoff of **30** in the corresponding text box. This option creates a hidden copy of the dataset, applies a low-pass filter to it with the cutoff you specified (30 Hz in this case), and applies the artifact detection algorithm to this hidden copy. The artifact detection flags are then copied to the original dataset. This will suppress most of the EEG activity, allowing you to avoid flagging epochs with large EMG, but you end up with your original unfiltered data.

Click **Run** and name the resulting EEGset **1_MMN_preprocessed_interp_be_ar2**. You'll see that only 12.99% of trials are flagged for rejection (compared to 28.15% without prefiltering). If you scroll through the data starting at epoch 100, you'll see that all the clear blinks are flagged, but the trials with EMG noise are not (e.g., Epochs 107 and 108). If you check the data quality for this EEGset using the **Compute Data Quality Metrics** panel, you'll see that it was better for this EEGset than for the best of the EEGsets for which we used the absolute voltage threshold (**1_MMN_preprocessed_interp_be_ar150**).

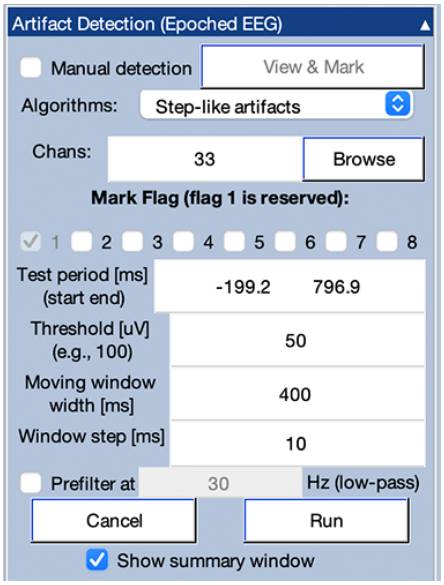
So, the moving window peak-to-peak algorithm is definitely superior to the absolute voltage threshold algorithm. It doesn't make a huge difference with this participant, but it makes a big difference for some participants and some experimental paradigms. However, in many cases, you'll want to use the low-pass prefilter option.

Now let's look at another algorithm that works even better but doesn't require any low-pass filtering. I call this algorithm the *step function*, because I developed it to detect the step-shaped voltage deflections produced by saccadic eye movements in N2pc paradigms. I eventually discovered that it also works great for detecting blinks.

The step function also involves a moving window, with somewhere between 200 and 400 ms as a reasonable default value for most studies. Within a 400-ms window, this algorithm calculates the difference between the mean voltage in the first half of the window and the mean voltage in the second half of the window. It then finds the largest of these differences for all the windows in a given epoch, and it compares the absolute value of this difference to the rejection threshold. You can use a lower detection threshold with the step function than with the other algorithms, which helps avoid false alarms. Also, you will get the largest voltage from a blink if the center of the window is just slightly before the start of the blink, and a smaller step size (e.g., 10 ms)

tends to be better.

Let's try it. Select **1_MMN_preprocessed_interp_be** in the EEGset panel and then go to the **Artifact Detection** panel and select **Step-like artifacts** in the **Algorithms** menu. Set the parameters as shown in Screenshot 8.7. Specifically, set the window width to **400** and the window step to **10** (so that we get a 400-ms window every 10 ms). Set the threshold to **50** and the channel to **33**. Turn off the prefiltering checkbox. Click **Run** and name the resulting EEGset **1_MMN_preprocessed_interp_be_step50**. Click **Save** in the **EEGsets** panel to save this EEGset to disk; you'll need it for the next exercise.



Screenshot 8.7

You should first note that 13.38% of trials have been flagged for rejection. This is a little more than what we obtained when we used the moving window peak-to-peak algorithm with the prefiltering option. But note that no filtering is required with the step function: when the step function algorithm averages across each half of the 200-ms window, high-frequency activity is virtually eliminated. And if you scroll through the data, you'll see that the step function algorithm has flagged a couple of epochs that were previously missed (e.g., Epochs 266 and 331). Moreover, if you check the data quality for this EEGset using the **Compute Data Quality Metrics** panel, you'll see that it was even better than what we obtained with the moving window peak-to-peak algorithm.

In my experience, the step function works slightly better than the moving average peak-to-peak algorithm (especially when there is a lot of EMG noise) and significantly better than absolute voltage thresholds. It's what I recommend for detecting blinks in most cases.

This page titled 8.10: Exercise- Better Blink Detection is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 8.9: Exercise- Better Blink Detection by Steven J Luck is licensed CC BY 4.0.

8.11: Exercise- Commonly Recorded Artifactual Potentials (C.R.A.P.)

Now that you understand the basics of artifact detection and rejection in the context of blinks, we'll shift to a class of miscellaneous artifacts that I like to call Commonly Recorded Artifactual Potentials (C.R.A.P.). Whereas virtually all participants in all experiments exhibit blinks, which have a predictable waveshape and scalp distribution, C.R.A.P. artifacts tend to be idiosyncratic in their waveshape and scalp distribution. This category includes EMG bursts, movement artifacts, skin potentials, sudden voltage changes of unknown origin, and other kinds of "crap" in the data.

You can really improve your data quality by setting the right parameters to detect and reject C.R.A.P. artifacts. Even if you use the same parameters for each participant, we've shown that rejecting C.R.A.P. improves data quality and increases statistical power (Zhang et al., 2024c). You can get even better data quality by adjusting the parameters individually for each participant, as will be shown in the exercises in this section. It takes some time, but it's time well spent. If you think about how many months (or years) you spend collecting data for a study, a little extra time setting artifact detection parameters is very worthwhile because it may be the difference between a *p* value of .06 (boo!) and a *p* value of .02 (yay!).

General approach to C.R.A.P. detection in the ERP CORE experiments

In the ERP CORE experiments, our approach to C.R.A.P. detection was to apply both a simple voltage threshold and the moving window peak-to-peak algorithm to all EEG channels. The simple voltage threshold is particularly good for detecting large drifts in voltage, and a threshold of ±200 μV works well for most participants. The moving window peak-to-peak algorithm is good for detecting muscle bursts and idiosyncratic artifacts that involve sudden voltage changes. The threshold for this routine needs to be adjusted for each participant to ensure that large artifacts are detected without throwing out too many epochs. In most participants, we used a threshold between 100 and 150 μV.

Subject 1 from the ERP CORE MMN experiment does not have very interesting C.R.A.P. artifacts, so we will start by looking at the data from Subject 6. In addition, it's a lot easier to demonstrate these artifacts in the absence of blinks, so we will work with a copy of Subject 6's data in which ICA-based artifact correction has been applied to estimate and remove the blink-related voltages (using procedures that will be described in the next chapter).

Start by resetting everything with the **Reset** button, and then load the EEGset named **6_MMN_blinks-corrected_be** using the **EEGset** panel. As you will see, this EEGset has already been epoched (as indicated by the **_be** suffix in the EEGset name). You will also see that there is a VEOG-bipolar channel in which the blinks have not been corrected. This channel is useful for telling us when blinks occurred. For example, you can see from this channel that there was a large blink in epoch 3 (which is easier to see if you select the **Stack** option in the **Plot Settings** panel). You can also see that there is no blink-related activity in the FP1 and FP2 channels in this epoch, indicating that the artifact correction worked reasonably well.

In this example, we will assume that the MMN will be measured as the mean amplitude from 125-225 ms in a cluster of electrode sites. Zhang and Kappenman (2024) carefully examined the best channels for scoring the amplitudes for each of the ERP CORE paradigms. They found that, in most cases, averaging across a cluster of electrodes around the site with the maximum effect was the best option. For the MMN, the maximum effect was at FCz, and they recommended scoring from the average of Fz, FCz, Cz, FC3, and FC4. You can average across sites after making the averaged ERPs (or even after scoring the values from each individual site). However, we will want to compute the aSME metric of data quality to see if artifact rejection is improving the data quality, and this requires creating the cluster (i.e., averaging across sites) in the EEG, prior to averaging. If you look at the current EEGset, you'll see that it includes a channel named **Cluster** (channel 17), which I created by averaging together the Fz, FCz, Cz, FC3, and FC4 channels. We will try to maximize the data quality for this cluster channel.

Our approach will be to test all EEG channels for C.R.A.P. and then exclude the entire epoch (all channels) if C.R.A.P. is found in any channel. You might think that it would be better to check for C.R.A.P. only in the channel(s) being used for measurement. That way, you won't reject trials with C.R.A.P. in other channels that won't impact data quality in the measurement channel(s). I had the same intuition. But when we tried this, we found that it did not lead to better data quality than excluding trials that had C.R.A.P. in any channel (Zhang et al., 2024c). The reason for this is that C.R.A.P. is usually present across a wide range of electrodes, but it might not exceed the threshold for rejection in all sites. As a result, if you check for C.R.A.P. only in the measurement electrode(s), you will sometimes fail to mark epochs for rejection when C.R.A.P. is present. However, there may be some cases where this is a

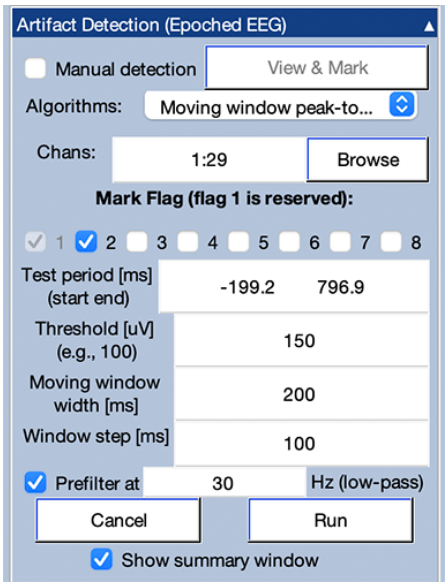
reasonable approach.

If an artifact is occurring at only a single electrode site, and it is present throughout much of the recording session, then you should interpolate that site. If you have an artifact at a single site that occurs only infrequently, you can apply artifact detection to that site but then interpolate that site for those epochs rather than rejecting those epochs (click here for more information).

Finding rapid voltage changes with the moving window peak-to-peak algorithm

Let's first try to detect C.R.A.P. in the loaded EEGset using the moving window peak-to-peak algorithm and a threshold of 150 µV. Go to the **Artifact detection** panel and set it up as shown in Screenshot 8.8. Note that we are going to look for C.R.A.P. only in the EEG channels (**1:29**), not in the EOG channels. We're also going to prefilter at 30 Hz so that high-frequency noise doesn't cause epochs to be flagged (because this noise has minimal impact on mean amplitude scores). Also note that the box for Flag **2** should be checked. This will cause every epoch that is flagged for rejection to have Flag 2 set. When we later detect C.R.A.P. with the absolute voltage threshold algorithm, we will use Flag **3**, and the different flags will allow us to determine whether a given epoch was flagged because of its peak-to-peak amplitude or its absolute voltage.

Screenshot 8.8



Click the **Run** button and name the resulting EEGset **6_MMN_blinks-corrected_be_mw150** (where mw150 indicates that you used the moving window peak-to-peak algorithm with a threshold of 150). You'll then see a summary table showing that only 17 epochs were marked for rejection (4 Deviants and 13 Standards). Is it even worthwhile to reject only 2% of epochs?

To answer that question, you should first scroll through the new EEGset and see what the EEG looks like in the marked epochs versus the unmarked epochs. For the first 35 epochs, the EEG looks very clean and no epochs are marked (i.e., no epochs have a yellow background), although there are some blinks and eye movements visible in the EOG channels. Epoch 37 seems to have some large voltage swings in several of the EEG channels, but these artifacts were not detected by the algorithm. However, the **Cluster** channel did not show a large voltage swing, so it's fine that this epoch was not marked for rejection.

If you keep scrolling, you'll eventually see that epochs 82-85 have some very large artifacts (> 100 µV) in the EEG channels, and they are marked for rejection. The **Cluster** channel didn't exceed the threshold for rejection, but this channel was quite noisy in those epochs, so it's good that those epochs were marked for rejection. This is an example of why it can be good to reject an epoch even if the artifact is flagged in a channel not being used for scoring the component of interest.

Skip ahead to Epoch 120. You'll see that it's marked for rejection, and the **Cluster** channel was one of the channels that exceeded the threshold for rejection. Epoch 121 has a very large swing in voltage, but it did not exceed the threshold. This is

because the moving window peak-to-peak algorithm is looking for relatively sudden changes in voltage, not gradual drifts like this. In a moment, we will use the simple voltage threshold algorithm to detect drifts like this.

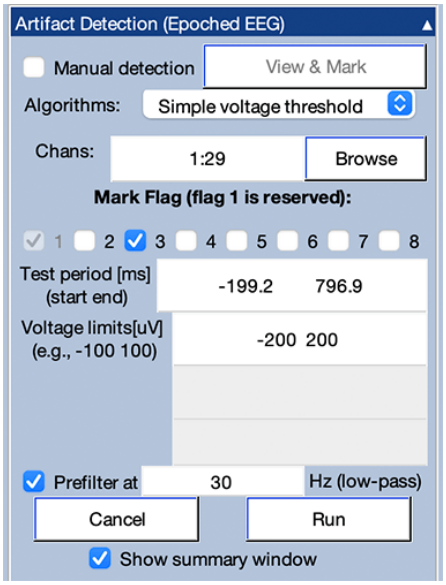
Once you feel that you've viewed enough epochs to see what kinds of voltage patterns exceeded the threshold, you should quantify the data quality. Let's first do this for the original EEGset, prior to artifact detection, so select **6_MMN_blinks-corrected_be** in the **EEGsets** panel. Now go to the **Compute Data Quality Metrics** panel. It should still have **Custom parameters** selected from the last time you used it (to include the MMN measurement window of 125-225 ms). If not, take a look at Screenshot 8.5 in Section 8.9 to see how to set this up. Then click **Run** to see the data quality table.

Now repeat the process of computing the data quality with **6_MMN_blinks-corrected_be_mw150** selected in the **EEGsets** panel. If you look in the **125:225** column of the two tables you just created, you'll see that the aSME values in many channels are quite a bit lower after artifact rejection compared to before artifact rejection. In the **Cluster** channel, for example, the aSME went from 0.8167 before rejection to 0.6030 after rejection (in Bin 1). Even though we've rejected only about 2% of the epochs, we've reduced the aSME by over 25%. A few epochs with large C.R.A.P. can really impact your data quality!

Finding slow voltage drifts with the simple voltage threshold algorithm

Now let's see if we can improve the data quality further by rejecting epochs that had large voltage swings but were not marked for rejection by the moving window peak-to-peak algorithm. Make sure that **6_MMN_blinks-corrected_be_mw150** is still selected, because we want to keep those marks and add to them. Now go to the Artifact Detection panel and set it up to use the **Simple voltage threshold** with a threshold of **-200 200** (as shown in Screenshot 8.9). Note that we're now going to set Flag **3** instead of Flag **2**.

Screenshot 8.9



Click the **Run** button and name the resulting EEGset **6_MMN_blinks-corrected_be_mw150_svt200** (where svt200 indicates that you used a simple voltage threshold of 200). The summary table will show the number of epochs flagged for each bin by the previous algorithm in the **F2** column (4 Deviants and 13 standards) and the number flagged by the simple voltage threshold in the **F3** column (5 Deviants and 15 standards). However, many of the same epochs were flagged by both algorithms, so the total percentage rejected has gone up only slightly from 2.17% before to 2.55% now. That won't change the end result very much, so let's try a lower threshold that will flag more trials.

Go back to the Artifact Detection panel, change the **Voltage limits** to **-100 100**, click **Run**, and name the resulting EEGset **6_MMN_blinks-corrected_be_mw150_svt100**. You'll see that the number of epochs flagged with **F3** has gone up quite a bit, and 4.2% of epochs have now been marked for rejection.

If you scroll through the epochs, you'll see that Epochs 37, 86, and 121 are now marked for rejection. These epochs have large

but gradual drifts that the moving window peak-to-peak algorithm missed but were detected by the simple voltage threshold algorithm once we made the threshold low enough. However, if you generate the data quality table for this EEGset, you'll see that the aSME values haven't changed much.

You could try lowering the threshold to **-80 80**, **-60 60**, or **-50 50**. I tried all of these thresholds, and I found that the percentage of epochs marked for rejection increased as the threshold decreased (not surprisingly). However, the aSME value for the Cluster channel remained about the same. This is because the fact that the remaining epochs were cleaner (which makes the aSME better) was balanced by the fact that there were fewer trials left for averaging (which makes the aSME worse). I also tried a threshold of **-40 40**, but this led to the rejection of over 30% of trials (which would cause the subject to be excluded given my lab's a priori exclusion criterion of 25%) and made the aSME worse.

As I was developing this exercise, I wanted to show you that the simple voltage threshold algorithm was useful, and I was initially annoyed that it didn't improve the data quality over and above the improvement produced by the moving window peak-to-peak algorithm. I even tried finding another subject for whom it was helpful, but I couldn't find one. Then I realized that this is exactly what I should have expected. The slow drifts that the absolute voltage threshold picks up aren't a big problem when you are measuring the voltage at relatively short latencies (such as the 125-255 ms time window used for the MMN). At these short latencies, the baseline correction procedure takes care of much of the drift; a slow drift doesn't have time to change the voltage much at these short latencies. We could probably just use the moving window peak-to-peak algorithm for detecting artifacts in this paradigm. However, it doesn't hurt to also reject trials with slow drifts using the simple voltage threshold algorithm as long as we don't use an overly strict threshold. And there are many paradigms where it will help.

Example of slow drift from the P3b paradigm

To see how the simple voltage threshold algorithm can be helpful when you are looking at later time windows, we're going to look at an example from the P3b paradigm (which was described in Section 6.2 of Chapter 6). Start by clearing out everything from the **EEGsets** panel by clicking **Clear All**, and then load **30_P3_blinks-corrected_be**. I chose this EEGset because it contains tons of epochs with slow drifts. As in the previous examples, ICA-based artifact correction was used to minimize blink activity, but the EEGset also contains uncorrected VEOG-bipolar and HEOG-bipolar channels so you can see when the subject blinked or made an eye movement. I also added a channel labeled **Cluster**, which is the average of the Cz, CPz, Pz, P3, and P4 electrode sites (as recommended for scoring the P3b by Zhang and Kappenman, 2024).

Start by running the moving window peak-to-peak algorithm with a threshold of 150 (exactly as shown above in Screenshot 8.8). You can name the resulting EEGset **30_P3_blinks-corrected_be_mw150**. The summary window will show that 22 of the epochs were marked for rejection, which is 11.89% of the relatively small number of trials in this paradigm. If you scroll through the data, you'll see that many epochs with slow drifts were not marked (e.g., Epochs 57 and 63, which have fairly large drifts in the **Cluster** channel). The marked epochs mostly have relatively sudden voltage changes (e.g., Epochs 81 and 88).

Now let's see whether rejecting the marked epochs improves the data quality. Go to the **Compute Data Quality Metrics** panel. It should still have **Custom parameters** selected from the last time you used it, but with the custom time period for the MMN. Change that custom time period to 300-600 ms (and change the name accordingly). You can consult Screenshot 8.5 in Section 8.9 if you need a reminder of how to do this.

Once you've updated the custom time period, let's see what the data quality would be like without any rejection. Previously, you accomplished this by going to the **EEGsets** panel and selecting the original EEGset. But there is a quicker way: In the **Compute Data Quality Metrics** panel, select the radio button labeled **Include all epochs (ignore artifact detection)**. Then click **Run** to see the data quality table for this dataset without excluding the marked epochs. Now select the radio button labeled **Exclude epochs marked during artifact detection**, and then click **Run** again to see the data quality table for this dataset after excluding the marked epochs. By consulting the two data quality tables you just created, you can see that the aSME during the **300:600** time window was improved by excluding the marked epochs in most channels, including the **Cluster** channel (for both Bin 1 and Bin 2).

Now try adding the simple voltage threshold algorithm with a threshold of 200 (exactly as shown in Screenshot 8.9), naming the resulting EEGset **30_P3_blinks-corrected_be_mw150_svt200**. You should see that 22.16% of trials are now marked (which is getting close to my lab's limit of 25%, above which we would need to exclude the entire subject from further analysis). If you look through the data, you'll see that epochs with large drifts are now marked (e.g., Epochs 57 and 63, which were not marked before). If you look at the data quality metrics, you'll see that the aSME for the **Cluster** site in the **300:600** time window is about the same as it was with just the moving window peak-to-peak algorithm in Bin 1, but the aSME is now much improved for this case in Bin 2.

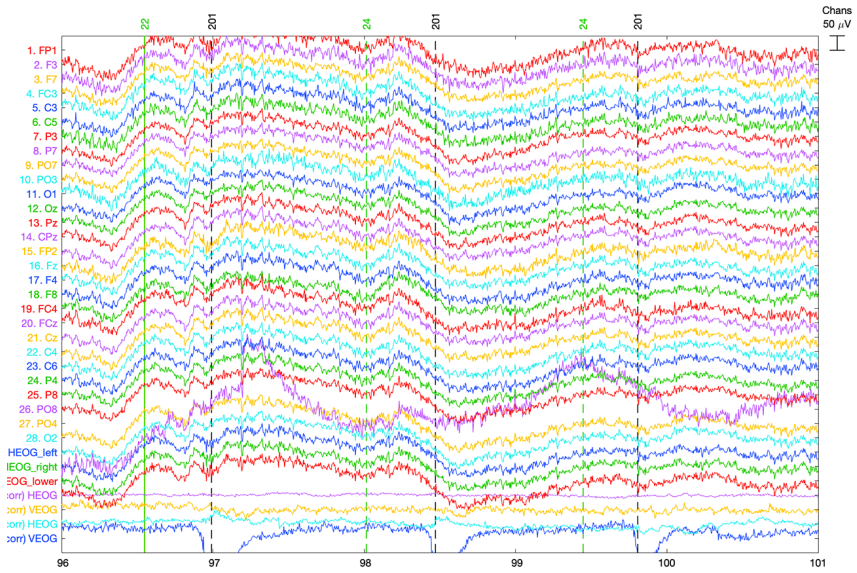
Just like any other metric, the aSME can be better estimated for a bin with more trials, so you should pay more attention to aSME values from bins with more trials (such as Bin 2 in the present paradigm). Or, if you want to get really serious about quantifying data quality, you can estimate the aSME for the difference between two bins. In the P3b paradigm, for example, the ultimate score is the mean voltage from 300-600 ms in a rare-minus-frequent difference wave. As discussed in Section 4.15 of the chapter on filtering, you can obtain the aSME corresponding to a difference wave between conditions A and B using this formula: $\mathbf{SME_{A-B} = sqrt(SME_A^2 + SME_B^2)}$. The overall aSME was sqrt(4.8086^2 + 5.3852^2) = 7.2196 without any rejection, sqrt(3.5405^2 + 3.2736^2) = 4.82199 after the moving window peak-to-peak rejection, and sqrt(3.6970^2 + 1.9024^2) = 3.9459 after both the moving window peak-to-peak and simple voltage threshold rejection. Thus, adding the simple voltage threshold improved the aSME value corresponding to the rare-minus-frequent difference wave.

A very challenging example from the P3b paradigm

Up to this point, I've selected example EEGsets for which it's fairly easy to mark the artifacts and end up with better data quality. Now I want to show you a very difficult case so you can experience the failure you will occasionally encounter when you perform artifact detection on your own data. (For fans of Star Trek, this is like the Kobayashi Maru).

I recommend resetting everything with the **Reset** button before going further. Then load **9_P3_blinks-corrected_continuous** using the **EEGsets** panel. I wanted you to start with the continuous EEG so that you better understand why the epoched data looks so bad and why artifact detection will be so difficult. If you scroll through the data, you'll see that it looks like a roller coaster ride, going up and down and up and down. This participant also blinked near the time of the buttonpress response (event code 201) on almost every trial. An example is shown in Screenshot 8.10. I have no idea what is causing these roller-coaster-like voltage deflections. They're too fast to be skin potentials, but they are too slow to be the kinds of movement artifacts we typically see in my lab (which are usually sudden movements). This is the sort of EEGset that led me to develop the acronym C.R.A.P. for miscellaneous artifacts.

Screenshot 8.10



Once you've looked through the continuous EEG, load the epoched EEG file, **9_P3_blinks-corrected_be**. If you scroll through the data, you'll see these same upward and downward deflections, which often go up and then down (or down and then up) within a single epoch. These deflections are often over 50 μV, which is very large relative to the ERP waveforms we obtained in this experiment. Moreover, these large deflections are going to produce tremendous trial-to-trial variation in the mean voltage between 300 and 600 ms, making it very difficult to obtain a precise estimate of this subject's P3b amplitude. In fact, this subject's aSME without any rejection is about twice as high as the aSME from the previous example without any rejection and about four times as high as the average ASME across all participants after rejection.

Rather than taking you step by step through the artifact detection process for this subject, I'm going to describe the various

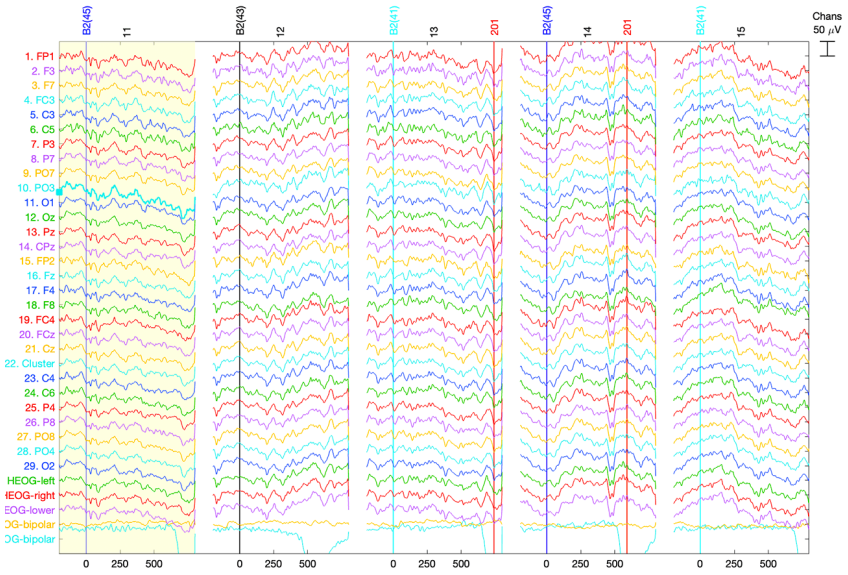
artifact detection parameters I tried. I've provided the EEGsets for each of these parameters. However, if you're up for the challenge, you can try to come up with your own artifact detection parameters before reading about how I came up with my parameters.

I started by applying the moving window peak-to-peak algorithm with a threshold of 150 (see the file named **9_P3_blinks-corrected_be_mw150**). Not a single epoch was flagged by this (which you can see for yourself by loading the file, going to the **Artifact Info & Tools** panel, and clicking **Artifact Summary**). That makes sense, because there weren't any large, sudden voltage changes in the EEG channels of this EEGset.

Next I applied the simple voltage threshold algorithm with a threshold of 200 (see **9_P3_blinks-corrected_be_mw150_svt200**). Only two epochs were flagged. That makes sense, because there weren't many large drifts that went consistently upward or consistently downward within a single epoch. Excluding the two flagged epochs had no real impact on the aSME.

My usual approach to C.R.A.P. detection is to start with these two steps and then make adjustments to the thresholds if necessary. This was a case where adjustments were definitely necessary. I started by reducing the simple voltage threshold to 100 (**9_P3_blinks-corrected_be_mw150_svt100**). That flagged 27% of trials, which is above my lab's maximum of 25%, which would have led this subject to be excluded. Also, it only reduced the aSME by a modest amount. And, as you can see in Screenshot 8.11, it mainly flagged epochs that had a consistent upward or downward drift (like Epoch 11), and it failed to flag epochs that had large up-and-down shifts within the epoch (like Epoch 14).

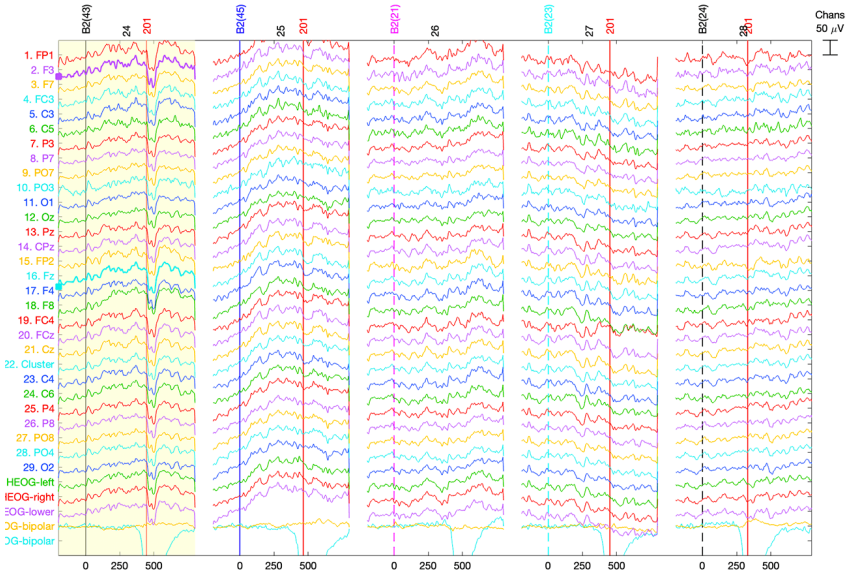
Screenshot 8.11



I next tried raising the threshold to 120 to avoid going over my lab's limit of 25% rejection (**9_P3_blinks-corrected_be_mw150_svt120**). That reduced the number of flagged epochs to 15%, but now it more frequently failed to flag high-noise trials, and the aSME was hardly any better than what I observed without any rejection.

Next, I went back to the simple voltage threshold of 200 and added the moving window peak-to-peak algorithm with a lower threshold of 100 (**9_P3_blinks-corrected_be_mw150_svt200_mw100**). This marked 11.5% of the epochs. It successfully flagged epochs with sudden changes in voltage (e.g., Epoch 24 in Screenshot 8.12), but it failed to mark epochs that had large but more slowly changing voltage fluctuations (e.g., Epoch 25 in Screenshot 8.12). Also, the aSME was now even worse than it had been without any rejection.

Screenshot 8.12



I tried decreasing the moving window threshold even further, down to 80, but this put me over 25% of epochs flagged, and the aSME was still bad (**9_P3_blinks-corrected_be_mw150_svt200_mw80**). The problem was that the algorithm wasn't flagging epochs like Epoch 25 in Screenshot 8.12. The voltage in those epochs changed too slowly to produce a large voltage deflection within the 200 ms sliding window, but because the artifacts went both upward and downward within a single epoch, the simple voltage threshold algorithm also failed to flag them.

Then I realized that I might be able to detect these artifacts if I increased the width of the sliding window. I tried a few different window widths, but what ended up working the best was a window width of 995 ms (which is the entire width of the epoch). When I tried a 995 ms moving window width with a threshold of 100, it did a great job of flagging the epochs that had large voltage deflections (**9_P3_blinks-corrected_be_mw150_svt200_mw100-995**). And it brought the aSME down quite a bit, from 8.5124 without rejection to 5.7999 with rejection (for the rare trials in Bin 1). But that's still not great; the average aSME for rare trials in the P3b paradigm was less than 2.0 (Zhang & Luck, 2023). Also, the P3b on rare trials in this paradigm had an amplitude of about 15 μV, and a standard error of about 5.7999 relative to an amplitude of 15 μV is pretty poor. Moreover, nearly 50% of trials were marked for rejection, greatly exceeding my lab's rule for excluding subjects.

The fact that the aSME was improved by over 30% by rejecting trials in this manner, even though the number of trials was reduced by half, makes it clear that the rejected trials really were adding a lot of noise. In other words, the peak-to-peak amplitude over a 995 ms window with a threshold of 100 was flagging bad trials and wasn't flagging good trials. But even the unflagged trials contained a lot of noise, so the aSME still remained very high.

I tried several other artifact detection parameters, but everything I tried led to a poor aSME or over 25% of trials rejected (or both). I had to admit defeat. There was no way that this subject could be included in the final analyses of the ERP CORE P3b experiment. In fact, this subject was excluded in the published paper because the person who set the artifact detection paradigm, Jaclyn Farrens, was also unable to find parameters that worked for this subject's data. Sometimes, you just have to throw out some of the participants in a study because of poor data.

I'd like to end this set of exercises by noting that there is nothing magic about excluding subjects for whom more than 25% of trials were rejected. That seems to work well in my lab's basic science studies using college students as the participants. But in my research on schizophrenia, where the participants are much more expensive to recruit and test, and the data are noisier, we use a threshold of 50% instead of 25%. You will need to choose a reasonable threshold for your own research, balancing the benefits of throwing out participants with C.R.A.P.P.Y. data with the costs of needing to recruit and test extra participants. Just make sure that you set this criterion a priori so that you don't bias your results (as discussed in Section 8.4).

This page titled 8.11: Exercise- Commonly Recorded Artifactual Potentials (C.R.A.P.) is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 8.12: Exercise- Commonly Recorded Artifactual Potentials (C.R.A.P.) by Steven J Luck is licensed CC BY 4.0.
- 8.5: Exercise- Simple Blink Detection by Steven J Luck is licensed CC BY 4.0.

8.12: Exercise- Detecting Eye Movements

In this exercise, we'll look at another common artifact, saccadic eye movements. Depending on the nature of the experiment, saccadic eye movements can be an enormous problem or largely irrelevant. How can you tell? The answer, as always, is to think about the three problems that artifact rejection is designed to solve (reduced power, confounds, and changes in the sensory input). Because the MMN experiment uses auditory stimuli, changes in eye position won't directly impact the sensory input. And because participants were watching a silent movie at fixation during the experiment, there is no reason to suspect that the eyes will produce a confound by moving in different directions for deviant versus standard trials (although this is something that we should verify rather than assume). So, the main question is whether the eye movements add significant noise and thereby decrease our statistical power. We can look at the SME values to determine whether rejecting trials with eye movements helps us (because it reduces a source of noise) or hurts us (because it reduces the number of trials).

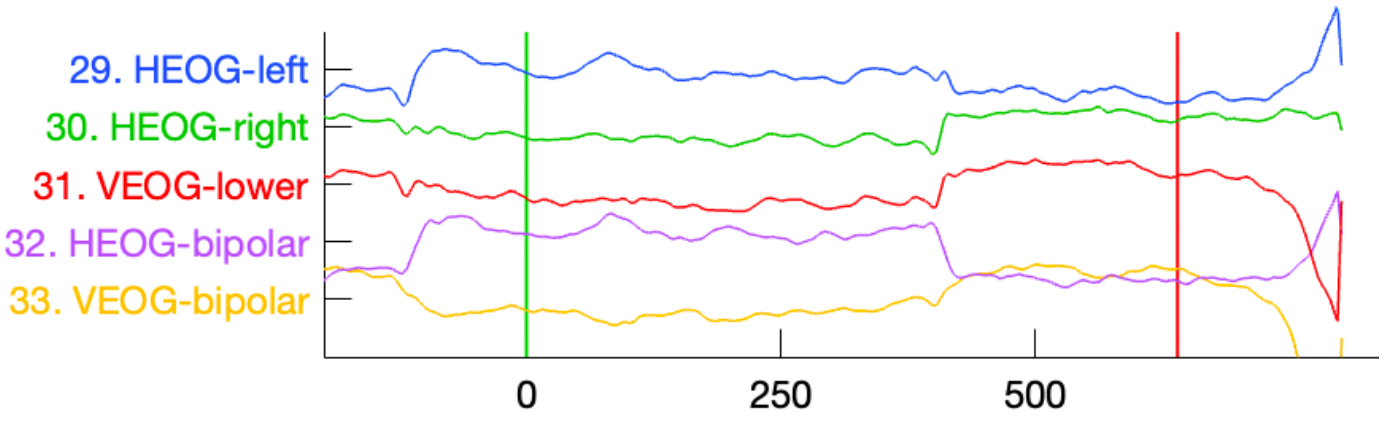
Subject #1 in the MMN experiment made no obvious eye movements while doing the task, so we're going to look at Subject #10 for this exercise. At this point, I'd recommend resetting everything with the **Reset** button. You can then load the dataset named **10_MMN_preprocessed.set**. Take a quick look to familiarize yourself with this participant's EEG. Ordinarily, you'd look carefully and think about whether to interpolate any channels, but here we'll just focus on the eye movements.

To make the eye movements easier to visualize, we're going to start by applying a low-pass filter to minimize high-frequency noise. Go to the **Filtering** panel and apply a low-pass filter with a 30 Hz half-amplitude cutoff and a slope of 12 dB/octave. Save the resulting dataset as **10_MMN_preprocessed_filt**. Then go to the **Extract Bin-Based Epochs** panel and extract epochs using a time range of **-200 800** ms and **Pre** as the baseline. Name the resulting EEGset **10_MMN_preprocessed_filt_be**. Now we're ready to look at the eye movements.

Why We're Filtering Now

We're going to use the step function to detect eye movements in this exercise. In a previous section, I said that filtering out high-frequency noise isn't necessary for the step function. So, why am I asking you to filter the data here? The answer is simple: At this point, I want you to learn what eye movements look like, and the filtering will make that easier by smoothing out the "fuzz" in the waveforms. The filtering will have virtually no impact on the operation of the step function algorithm.

Unless the participant is tracking moving objects or is walking around, most of the eye movements you'll see will be *saccades* (sudden shifts in gaze position). You can see this by scrolling to Epoch 7. Screenshot 8.13 shows a blowup of what the EOG channels look like in this epoch.



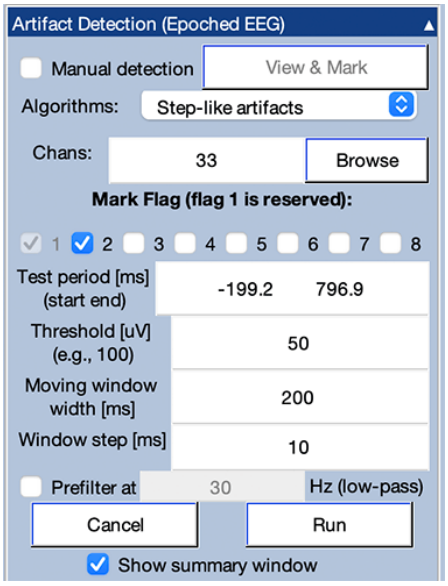
Screenshot 8.13

You can see a classic saccadic eye movement pattern at approximately 400 ms. The voltage is fairly flat for a few hundred milliseconds, and then there is a sudden shift in the EOG channels, followed by a relatively flat signal at a different voltage level until a blink occurs near the end of the epoch. This voltage pattern occurs for saccades because gaze is fixed at one location for

a period of time, then moves rapidly to a new location, and then remains fixed at that new location for a period of time.

At ~400 ms, the voltage goes more positive at HEOG-right and more negative at HEOG-left, indicating a rightward eye movement. But the voltage simultaneously shifts in the positive direction at VEOG-lower, meaning that the eye movement is actually angled downward as well.

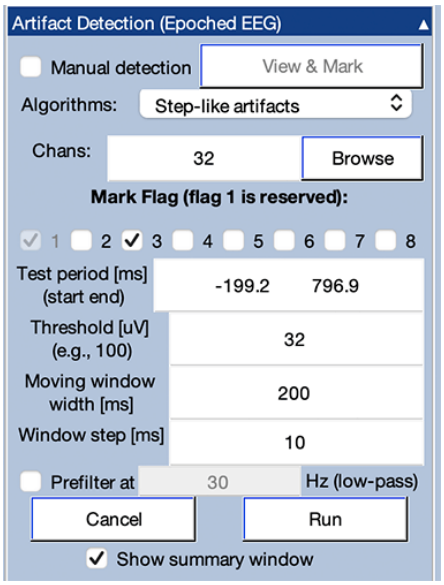
This participant has a fairly large number of blinks, and we should deal with those before we assess the effects of the eye movements. To flag the blinks, set up the **Artifact Detection** panel to detect step-like artifacts in the VEOG-bipolar channel (channel 33), setting the parameters as shown in Screenshot 8.14. Make sure to set Flag 2. Click **Run** and name the resulting EEGset **10_MMN_preprocessed_filt_be_noblinks**. If you scroll through the data, you'll see that the algorithm did a good job of flagging epochs with clear blinks. Over 30% of epochs were marked for rejection, which exceeds my lab's usual standard for rejecting subjects, but we will ignore that in this exercise. (In the real analyses, blinks were eliminated by means of artifact correction, so it was not necessary to exclude this participant.



Screenshot 8.14

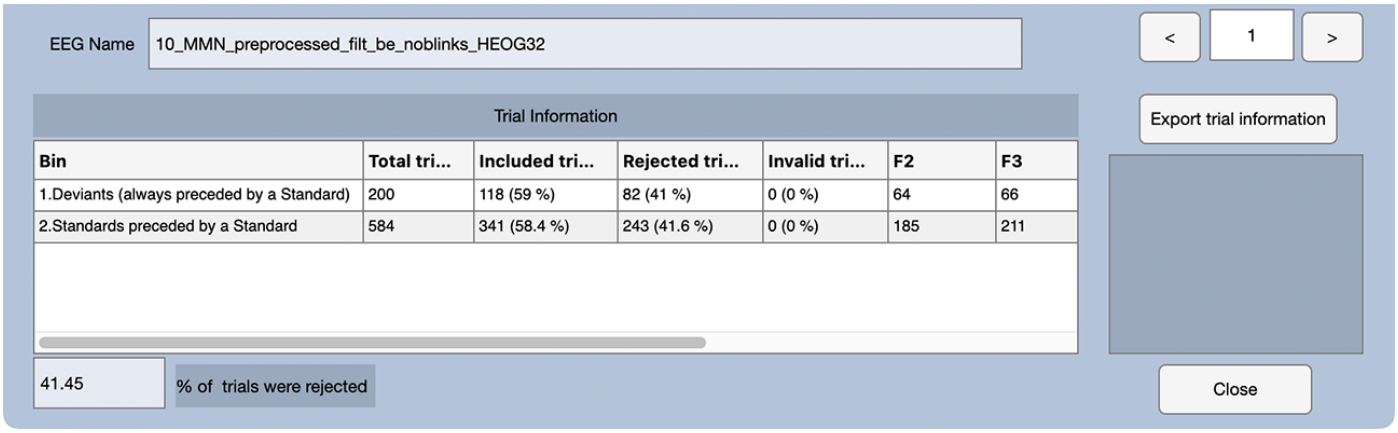
Now we're going to flag trials with horizontal eye movements. In some tasks, the stimuli are presented to the left or right of fixation, so most of the eye movements are horizontal. In the MMN task, participants watched a silent movie, so the eye movements might be in any direction. Consequently, we would ordinarily want to detect both horizontal and vertical eye movements in this task. However, it will be easier to understand what's going on in this exercise if we focus on horizontal eye movements.

To flag the horizontal eye movements, start with the EEGset you just created (**10_MMN_preprocessed_filt_be_noblinks**) and set up the **Artifact detection** panel to look for step-like artifacts in the HEOG-bipolar channel (32) with a threshold of 32 µV, as shown in Screenshot 8.15. I like to use multiples of 16 µV for horizontal eye movements because, for the average participant, each degree of eye rotation increases the HEOG voltage by 16 µV (Lins et al., 1993). A threshold of 32 should therefore detect eye movements of approximately 2° or larger. Make sure to set the checkbox for Flag **3,** click **Run**, and name the resulting EEGset **10_MMN_preprocessed_filt_be_noblinks_HEOG32**.



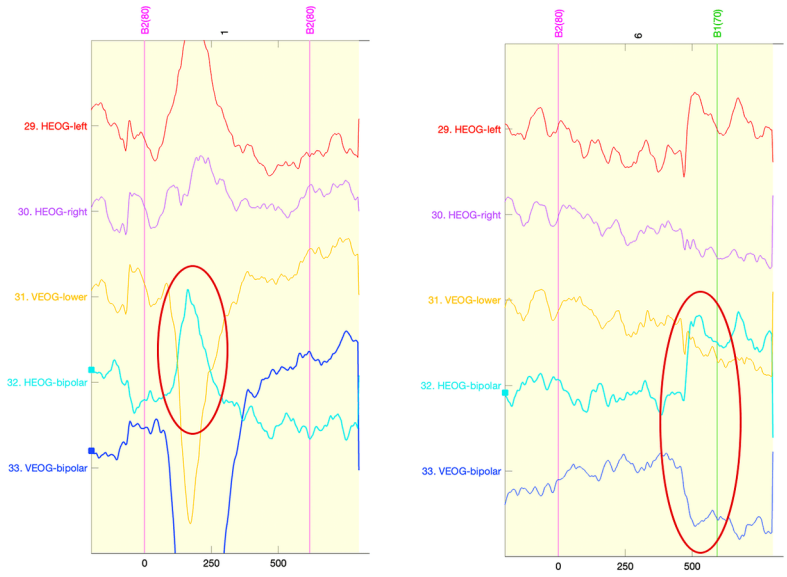
Screenshot 8.15

You should now see a summary window like the one in Screenshot 8.16. When we previously detected trials with blinks, 31.8% of epochs were flagged for rejection (collapsed across bins). That has now increased to 41.45%. The column labeled **F2** shows the number of epochs with Flag 2 set (blinks), and the column labeled **F3** shows the number of epochs with Flag 3 set (horizontal eye movements). In Bin 1, for example, 82 epochs were flagged for rejection overall, including 64 epochs with Flag 2 and 66 epochs with Flag 3. 82 total epochs were marked for rejection, but 64 + 66 is not equal to 82; this is because many epochs were flagged for both blinks and eye movements. There were 18 epochs flagged for eye movements that weren't flagged for blinks (because 82 total rejected epochs – 64 epochs flagged for blinks = 18 epochs flagged only for eye movements).



Screenshot 8.16

Now take a look at the EEG and EOG data. To focus on blinks and eye movements, I recommend selecting only channels 29-33 in the **Channel and IC Selection** panel (and then clicking **Apply**). You'll see that both the VEOG-bipolar and HEOG-bipolar channels were flagged in the very first epoch (i.e., the waveforms are drawn in thicker lines and there is a little square at the beginning of the waveform). The HEOG-bipolar waveform in Epoch 1 is not an eye movement—it doesn't show the sudden step from one voltage level to another voltage level that is produced by a saccadic eye movement. Instead, the blink that you can see in the VOEG-bipolar channel has "leaked" into the HEOG-bipolar channel (see the region marked with the red oval on the left side of Screenshot 8.17). In an ideal world, a blink would produce equal activity on each side of the two eyes, and the HEOG-left minus HEOG-right subtraction would therefore eliminate the blink in the HEOG-bipolar channel. However, if one HEOG electrode was placed a little higher or lower than the other, the blink activity won't be identical at the left and right sites, and the subtraction won't completely eliminate the blink activity. This is one reason why a large number of epochs were flagged for both blinks and eye movements—many of the artifacts detected in the HEOG-bipolar channel were blinks rather than saccades.



Screenshot 8.17

In Epoch 6, you'll see a clear eye movement in the HEOG-bipolar and VEOG-bipolar channels (the sudden step-like voltage change circled on the right side of Screenshot 8.17), which was correctly flagged. The fact that the step-like voltage change occurred simultaneously in both HEOG-bipolar and VEOG-bipolar indicates that the eye movement was probably diagonal rather than purely horizontal. Only the HEOG-bipolar channel was flagged, however, because we applied a threshold of 32 µV to the HEOG-bipolar channel, whereas we used a threshold of 50 when trying to detect blinks in the VEOG-bipolar channel.

If you scroll through the whole dataset, you'll see that many eye movements were successfully flagged, but others were missed (e.g., Epochs 3, 67, 190, 389). Let's reduce the threshold and see if we can detect those artifacts. First, save the current EEGset by clicking **Save EEG** in the **EEGsets** panel. You'll need it for the next exercise.

Now repeat the artifact detection process, keeping **10_MMN_preprocessed_filt_be_noblinks_HEOG32** as the active dataset so that we can add to the previous detections. Reduce the threshold to **16** in the **Artifact Detection** panel and select the **4** flag instead of the **3** flag. Click **Run** and name the new EEGset **10_MMN_preprocessed_filt_be_noblinks_HEOG16.**

Bin	Total tri...	Included tri...	Rejected tri...	Invalid tri...	F2	F3	F4
1.Deviants (always preceded by a Standard)	200	71 (35.5 %)	129 (64.5 %)	0 (0 %)	64	66	128
2.Standards preceded by a Standard	584	183 (31.3 %)	401 (68.7 %)	0 (0 %)	185	211	394

67.6 % of trials were rejected

Screenshot 8.18

A window will appear with an artifact summary like the one in Screenshot 8.18. You can see that we've now rejected a lot more trials than before (67.6% of the total; remember, we're not going to worry about the 25% threshold for rejecting subjects in this exercise). The **F3** column shows how many trials were flagged with the previous threshold of 32 µV, and the **F4** column shows how many trials were flagged with the new threshold of 16 µV. If you scroll through the data, you'll see that almost all trials with a clear horizontal eye movement are now flagged for rejection. Click **Save EEG** in the **EEGsets** panel to save this EEGset for a later exercise.

We've now done a good job of detecting the eye movements, but do we really want to throw away so many trials? To answer that question, you need to see how the aSME changed as a result of the various steps of artifact rejection. I've done this for you. The table below shows the aSME values for Bin 1 in the 125-225 ms time window in the fronto-central cluster of electrode sites (Fz, FCz, Cz, FC3, FC4).

aSME values for Bin 1 in the 125-225 ms time window in the fronto-central cluster of electrode sites (Fz, FCz, Cz, FC3, FC4)

No Rejection	Blink Rejection	Blink Rejection + HEOG32	Blink Rejection + HEOG16
0.9717	0.8043	0.7987	0.9369

As you can see from the table, rejecting blinks improved the data quality relative to not rejecting anything. Adding the saccadic eye movement rejection with a threshold of 32 µV yielded almost exactly the same data quality as rejecting only blinks. Lowering the eye movement threshold to 16 µV caused the data quality to get worse instead of better (because the number of trials remaining for averaging was so low).

Why didn't rejecting horizontal eye movements significantly improve the data quality? The reason is that horizontal eye movements produce opposite-polarity voltages over the left and right sides of the head, which cancel out in the bilaterally symmetric cluster of electrode sites we're using to score the MMN amplitude. We probably also rejected some vertical eye movements, which can cause substantial noise in the frontocentral cluster. With the 32 µV threshold, the benefits of reducing this noise were approximately canceled by the cost of reducing the number of trials, so the data quality was not changed much. With the 16 µV threshold, the cost of reducing the number of trials outweighed the reduction in noise, so the data quality became worse rather than better.

In many cases, you can use ICA to correct for eye movements as well as blinks, avoiding the need to reject any trials. If that's not feasible, then you'll need to look at the aSME to see whether the benefits of rejecting trials with eye movements outweigh the costs.

It's also important to consider whether eye movements create a confound by changing the visual input entering the eyes. That's not an issue for the auditory stimuli used in the MMN paradigm, but it can be an important issue in visual tasks. This issue will be considered in the next section.

This page titled 8.12: Exercise- Detecting Eye Movements is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 8.10: Exercise- Detecting Eye Movements by Steven J Luck is licensed CC BY 4.0.

8.13: Using Artifact Detection to Avoid Changes to Visual Inputs

Up to this point, we've focused on the first two of the three problems associated with artifacts, namely noise and systematic differences in voltage across conditions. The remainder of the chapter will focus on the third problem, namely that blinks and eye movements can change the sensory input when visual stimuli are used.

For most researchers, this is not a big issue. Once you've dealt with the noise produced by blinks and eye movements (either by rejection or correction), problematic changes in sensory input are relatively rare. If you're using artifact correction for blinks, you should still reject any trials with a blink that occurs at the time of the stimulus, because these trials are obviously not valid (see the chapter on artifact correction for details). If the stimuli are presented in the middle of the display, participants won't make a lot of eye movements, and if they do, they probably won't differ systematically across conditions. This is what we saw in the MMN experiment. Some participants made eye movements as they watched the silent movie in the middle of the display, but these eye movements didn't vary across conditions, and they couldn't impact the sensory processing of the main stimuli (the auditory tones). They were a potential source of noise, but not a confound.

However, eye movements can be a significant systematic confound in some types of studies, mainly those using peripheral visual stimuli. For example, consider the spatial cuing paradigm shown in Figure 8.4, in which an arrow is used to indicate the likely location of a subsequent target. Many studies have used this paradigm to determine whether covert shifts of visual attention to the cued location cause the P1 and N1 waves to be larger when the stimulus is presented at the cued location compared to the uncued location (e.g., Eimer, 1994; Luck et al., 1994; Mangun & Hillyard, 1991). However, participants are likely to shift their gaze toward the cued location in these studies. If that happens, the target will appear in the center of gaze when it is presented at the cued location, whereas it will appear in the periphery when it is presented at an uncued location. We know that foveal stimuli produce larger sensory responses than peripheral stimuli, so this difference in the retinal location of the stimuli is a major confound that must be avoided in these studies. To avoid this confound, we can reject trials with eye movements. However, this is more difficult than it seems, even if you're using a high-quality eye tracker. The next exercise will explain how to do this effectively.

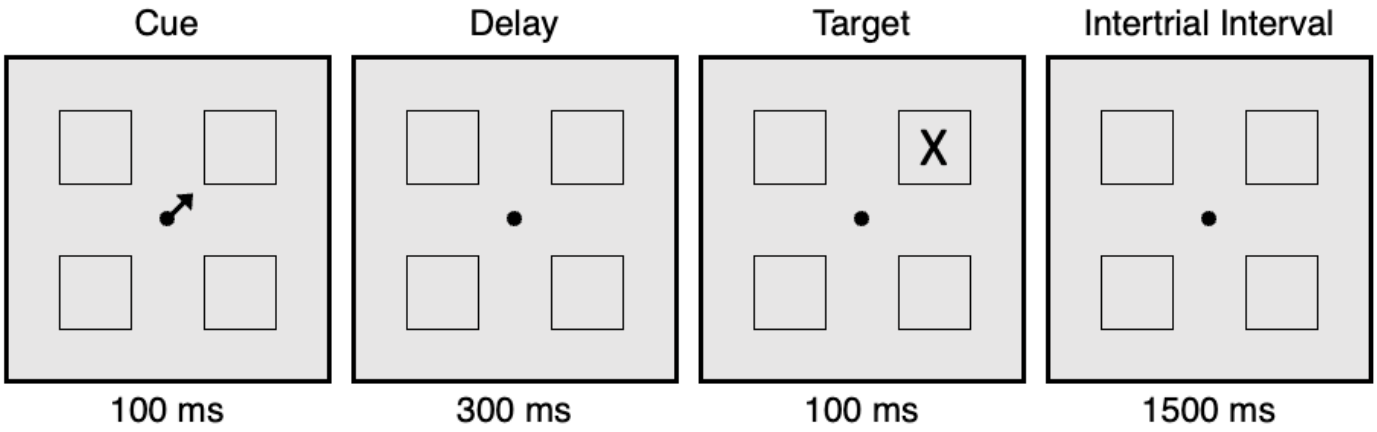


Figure 8.4. Prototypical spatial cuing paradigm. The cue indicates the likely location of the target. After a short delay, the target appears at the cued location (80% of trials) or one of the uncued locations (20% of trials). Participants press one of two buttons, as quickly as possible, to indicate whether the target is an X or an O. Participants are also instructed to maintain fixation on the central point and focus their "covert" attention onto the cued location. The goal is to determine whether sensory processing is enhanced at the cued location relative to the uncued location.

Small eye movements are also a problem in studies that look at lateralized visual ERP components, such as the N2pc component and contralateral delay activity (CDA). Both of these components are negative voltages contralateral to the location of a to-be-perceived or to-be-remembered object or set of objects. There are two specific problems that arise in these experiments. First, if the eyes move to the relevant location, then this location is now foveal, and that may impact the lateralization that would otherwise be observed. This problem is especially acute if the stimulus is presented for more than 200 ms, which is the approximate amount of time required to make a controlled eye movement in these paradigms. With long stimulus durations, you may have a brief period of time in which the relevant stimuli are lateralized (prior to the eye movement) followed by a longer period in which they are foveal (after the eye movement).

The second problem is not due to the change in the sensory input per se but is instead a confound resulting from changes in the EOG voltage. If participants tend to look leftward when the relevant stimuli are on the left side and rightward when the relevant stimuli are on the right side, then the EOG will be negative on the right side of the head when the relevant stimuli are on the left side and negative on the left side of the head when the relevant stimuli are on the right side. In other words, the EOG will appear as a negative voltage over the hemisphere contralateral to the relevant information, just like the N2pc and CDA. Moreover, the EOG is so large that even a small eye movement in the direction of the relevant information can produce a contralateral negativity that is as large or larger than the N2pc and CDA. The next exercise describes how to address both of these problems.

In theory, eye movements can also be a confound in studies of lateralized motor responses, such as the lateralized readiness potential (LRP; a negative voltage over the hemisphere contralateral to the response hand). This is because participants may make a small, unconscious eye movement toward the hand that responds. Such eye movements would produce a negative voltage over the right hemisphere for a left-hand response and a negative voltage over the left hemisphere for a right-hand response. That's the same pattern as the LRP. The strategy described in the following sections for eliminating these small eye movements for the N2pc and CDA can also be used for the LRP.

If you don't use lateralized visual stimuli or look at the LRP, then you can probably skip the rest of the chapter. However, you might want to read it and do the exercises anyway, because they provide good examples of the general principles of artifact rejection.

This page titled 8.13: Using Artifact Detection to Avoid Changes to Visual Inputs is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 8.13: Using Artifact Detection to Avoid Changes to Visual Inputs by Steven J Luck is licensed CC BY 4.0.

8.14: The ERP CORE N2pc Experiment

The following exercise will focus on data from the ERP CORE N2pc experiment. This section will describe the experimental design and event code scheme (see Kappenman et al., 2021 for a detailed description). Example stimuli are shown in Figure 8.5.A. N2pc experiments are typically designed to study how people focus their covert attention onto peripheral stimuli without looking at the stimuli, so participants in this experiment were instructed to maintain gaze on the central fixation point at all times.

The experiment was divided into 8 blocks of 40 trials, and participants were instructed to attend to pink in half the blocks and blue in the other half. The task was to find the square of the attended color (using peripheral vision) and press one of two buttons to indicate whether this square had a gap on the top or on the bottom. Each stimulus array had 1 pink square and 11 black squares on one side, along with 1 blue square and 11 black squares on the other side. Except for these constraints, the stimulus locations varied randomly from trial to trial. Most importantly, pink and blue were always on opposite sides, but we randomized which side contained pink and which side contained blue on each trial. Consequently, participants could not know where to shift attention until a given display appeared.

The N2pc component reflects the focusing of attention onto a visual object and is largest when this object is surrounded by nearby distractors (see review by Luck, 2012). It's a negative voltage in the N2 latency range (typically from 200-300 ms) at posterior electrode sites, and its distinguishing characteristic is that the negative voltage is larger over the hemisphere contralateral to the attended object. For example, the voltage for left-hemisphere electrodes will typically be more negative when the target is in the right visual field than when it is in the left visual field, whereas the voltage for right-hemisphere electrodes will typically be more negative when the target is in the left visual field than when it is in the right visual field.

To make the N2pc easier to visualize, we typically collapse the data into a contralateral waveform (left hemisphere when the target is on the right averaged with right hemisphere when the target is on the left) and an ipsilateral waveform (left hemisphere when the target is on the left averaged with right hemisphere when the target is on the right). These collapsed waveforms are shown in Figure 8.5.B. You can see that the voltage is more negative (less positive) from ~200-300 ms in the contralateral waveform than in the ipsilateral waveform. This voltage is summed with the other ERP components that are active at the same time, which are mainly positive. Thus, the overall voltage is typically positive in both the contralateral and ipsilateral waveforms, but more negative for contralateral than for ipsilateral. To isolate the N2pc from these other components, we make a contralateral-minus-ipsilateral difference wave, as shown in Figure 8.5.C. This difference wave subtracts away all the nonlateralized components, making it easier to see and quantify the time course of the N2pc component.

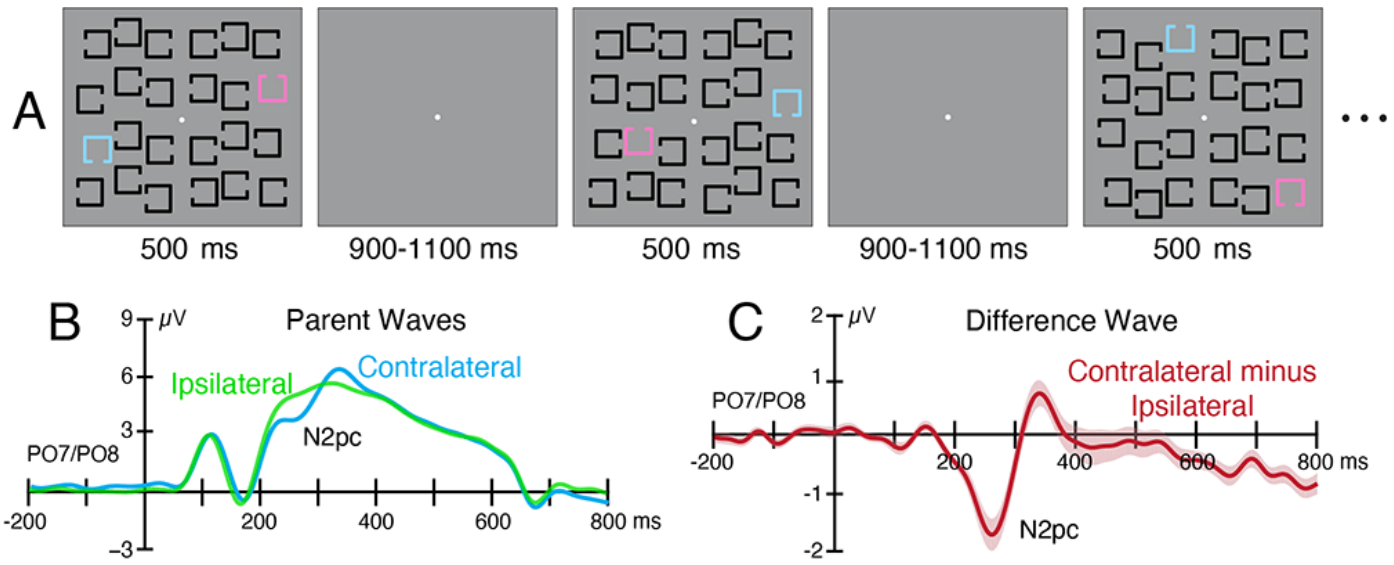


Figure 8.5. ERP CORE N2pc task (A), grand average ERP waveforms (B), and grand average contralateral-minus-ipsilateral difference wave.

The event code scheme is summarized in Table 8.3. You can see that the event code for each stimulus indicates which color was relevant for the current trial block (in the 100s place), which side contained the target square on the current trial (10s place), and whether the gap on the target square was on the top or bottom of the square (1s place). There were also event codes for correct and incorrect responses.

Table 8.3. Event codes for the ERP CORE N2pc experiment.

	Relevant Color for Current Block	Target Side	Target Gap Location	E
Stimuli	Blue	Left	Top	11
	Blue	Left	Bottom	11
	Blue	Right	Top	12
	Blue	Right	Bottom	12
	Pink	Left	Top	21
	Pink	Left	Bottom	21
	Pink	Right	Top	22
	Pink	Right	Bottom	22
	Accuracy			**E**
Responses	correct			2(
	incorrect			2(

The bin descriptor file is located in the Chapter_8 folder (named **BDF_N2pc.txt**), and the bin descriptors are shown below. You can see that there are separate bins for left and right targets, irrespective of which color was relevant in the current block and whether the target gap was on the top or the bottom. However, we require a correct response between 200 and 1000 ms after the stimuli.

bin 1
Left Target
.{111;112;211;212}{t<200-1000>201}

bin 2
Right Target
.{121;122;221;222}{t<200-1000>201}

This page titled 8.14: The ERP CORE N2pc Experiment is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 8.14: The ERP CORE N2pc Experiment by Steven J Luck is licensed CC BY 4.0.

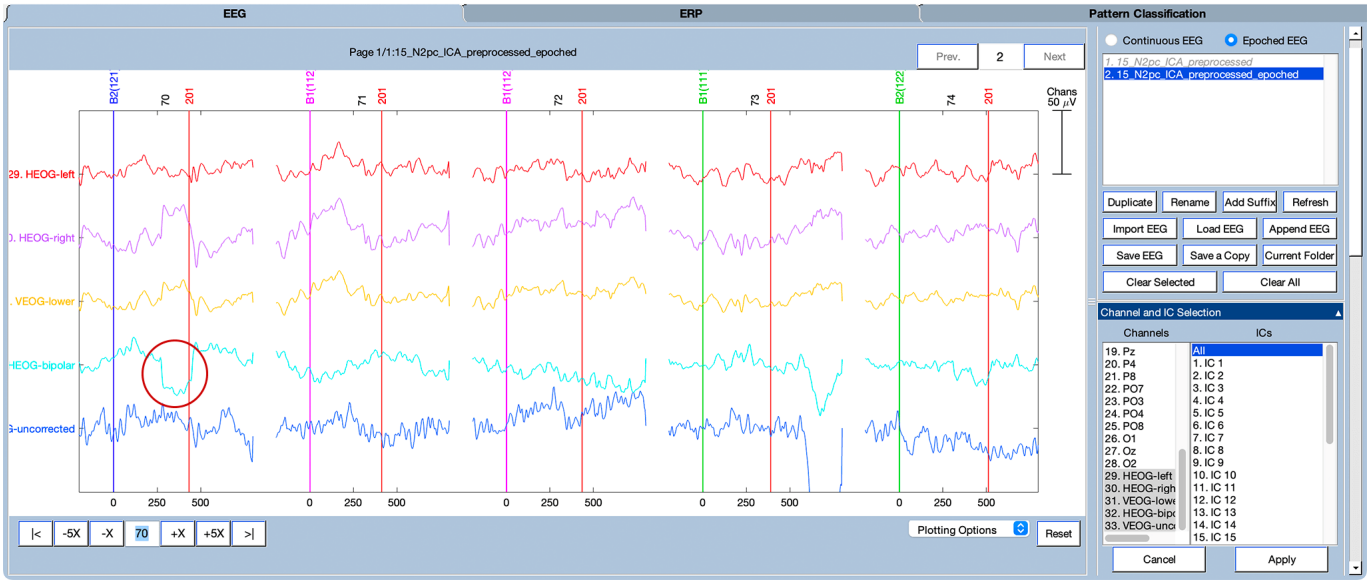
8.15: Exercise- Visualizing the Eye Movements

Now that you know about the ERP CORE N2pc paradigm, let's take a look at the eye movements. As you can imagine, it's difficult to maintain gaze on the central fixation point and not look toward the target. But as described above, it's important to make sure that the N2pc results aren't impacted by eye movements toward the target, which will change the location of the target relative to the center of gaze and also create a negative voltage over the contralateral hemisphere.

In this exercise, we'll focus on the data from Subject 15. If you look in the Chapter_8 folder, you'll see two datasets from this participant, one containing the continuous data (**15_N2pc_ICA_preprocessed.set**) and one containing the epoched data (**15_N2pc_ICA_preprocessed_epoched.set**). To make this exercise simpler, ICA-based artifact correction was already applied to these datasets to eliminate blinks. However, the datasets include a VEOG-uncorrected channel that contains the original uncorrected VEOG-bipolar signal. We like to keep this uncorrected channel so that we can reject any trials on which the participant blinked during the time period of the stimulus and therefore could not see the stimulus. This particular participant never blinked during that time period, however, so we don't need to worry about rejecting those epochs in this exercise. I also applied several other preprocessing operations, including filtering (bandpass 0.1–30 Hz, 12 dB/octave). The EEG channels and the HEOG-left and HEOG-right channels have all been referenced to the average of P9 and P10, and there is also an HEOG-bipolar channel (HEOG-left minus HEOG-right).

Before we get started on the exercise, reset everything with the **Reset** button, load the continuous dataset (**15_N2pc_ICA_preprocessed.set**) using the **EEGsets** panel, and scroll through the data. You should always scroll through the continuous data as the first step so that you know what's in the file (as we did in the video demonstration in the previous chapter). Subject 15 has some weird stuff in the F7 channel near the end of the session, but we won't worry about that for this exercise.

Now load the epoched dataset (**15_N2pc_ICA_preprocessed_epoched.set**), select only the EOG channels (29-33) in the Channel and IC Selection panel, and scroll through the data. This participant did a good job of following the fixation instructions initially, but you'll start to see a fair number of eye movements beginning at Epoch 70 (see Screenshot 8.19, with the eye movement circled in Epoch 70).



Screenshot 8.19

If you look at the event codes and the polarity of the HEOG deflections, you'll see that most of the clear eye movements are toward the side containing the target. Remember that the HEOG-bipolar channel was calculated as HEOG-left minus HEOG-right, and the dipole is positive at the front of the eyes, so a leftward eye movement produces a positive deflection and a rightward eye movement produces a negative deflection. For example, Epoch 70 has a negative deflection, indicating a rightward eye movement. You can see that the time-locking event code for this epoch is labeled "B2(121)", indicating that the event code was 121 (attend-blue, target on the right, gap on the top) and was assigned to Bin 2 (right-side targets). This is important because it

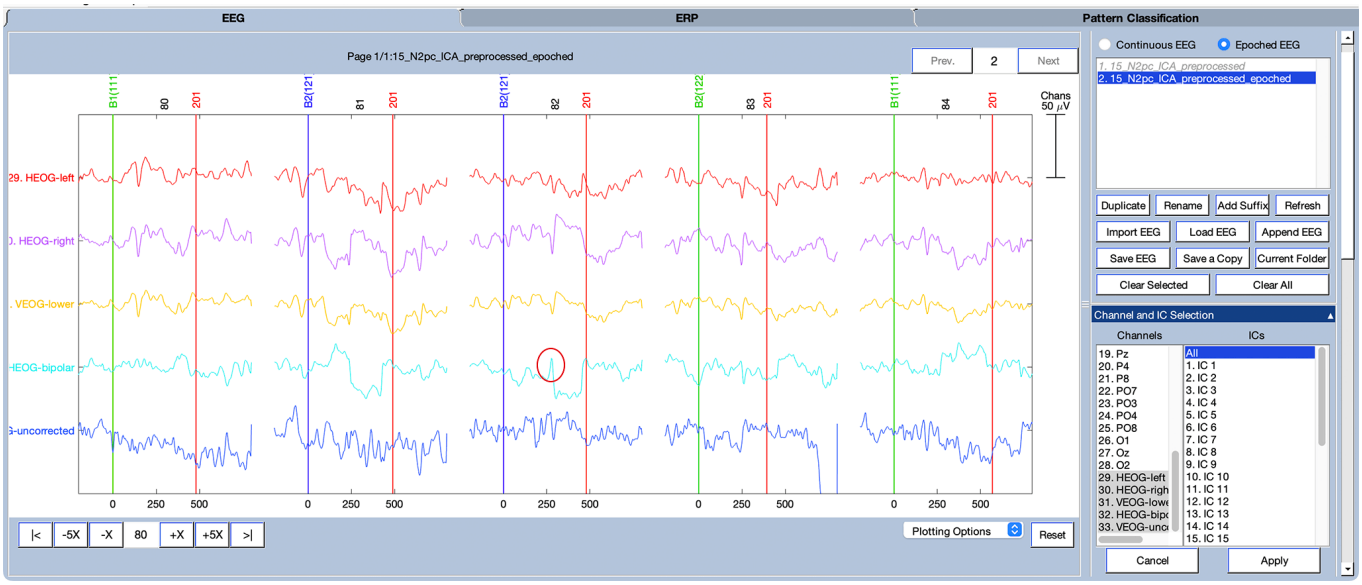
indicates that we had a rightward eye movement on a trial with a right-side target.

When participants try to maintain central fixation but occasionally fail, eye movements will typically consist of a rapid shift away from the fixation point, a period of stability at the new location for a few hundred milliseconds, and then a rapid shift back to the fixation point. This leads to a “boxcar” shape in the HEOG signal (a flat signal at one voltage level corresponding to the location of the fixation point, a sudden change to a different voltage level for 100–500 milliseconds, and then a return back to the original voltage level corresponding to the fixation point). You can see this pattern in Epoch 70. The voltage level changes suddenly at approximately 285 ms after the time-locking event and then returns to the original level approximately 180 ms later.

If you look at epoch 73, you’ll see a voltage deviation in the HEOG-bipolar channel, but it isn’t a saccadic eye movement. It doesn’t have the classic boxcar shape. Instead, it has the typical shape of a blink and matches the time course of the much larger deflection in the VEOG-bipolar channel. This indicates that the deflection in HEOG-bipolar is some blink voltage that has leaked through to the HEOG electrodes.

There’s a small leftward (positive) eye movement at approximately 570 ms after the time-locking event in Epoch 78, which had a left-side target. Epoch 81, with a right-side target, has a somewhat complicated pattern that looks like a brief leftward movement followed by a clearer rightward movement. Epoch 82, also with a right-side target, has a clear rightward eye movement.

There’s also a brief spike in voltage at the onset of the eye movement in Epoch 82 (circled in Screenshot 8.20). That’s probably the *spike potential*, which is caused by the muscle activity that produces the eye movement. Ordinarily, the muscle contraction that produces an eye movement occurs briefly at the start of the eye movement (to overcome the inertia in eye position), but then the muscle activity becomes too small to see as the eyes maintain their new location (which requires very little muscle activity). The present data have been low-pass filtered, so the spike potential isn’t very clear. Without the filtering, the spike potential is quite large in some participants. It’s fairly localized to frontal electrode sites, and it’s easy to filter out, so I don’t usually worry about it as an artifact. However, if you perform time-frequency analyses, it’s easy to mistake this EMG burst for gamma-band EEG activity (Yuval-Greenberg et al., 2008), so be cautious when someone says they’re seeing gamma-band EEG oscillations at frontal electrode sites.



Screenshot 8.20

If you keep scrolling through the data, you’ll see quite a few eye movements (mainly in the direction of the target, and mainly starting around 200 ms after stimulus onset). We clearly have a lot of work to do to make sure that these eye movements don’t confound our N2pc data!

This page titled 8.15: Exercise- Visualizing the Eye Movements is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 8.15: Exercise- Visualizing the Eye Movements by Steven J Luck is licensed CC BY 4.0.
- 8.5: Exercise- Simple Blink Detection by Steven J Luck is licensed CC BY 4.0.

8.16: Exercise- Using the Averaged HEOG to Visualize Consistent Eye Movements

In an N2pc (or CDA or LRP) experiment, we're mainly concerned with systematic eye movements in the direction of the target, not random eye movements. A good way to assess these systematic eye movements is to make one averaged HEOG waveform for all the left-target trials and another averaged waveform for all the right-target trials. Any differences between these two waveforms in the HEOG-bipolar channel are likely to reflect systematic eye movements toward the target. That's what we'll look at in this exercise, before we work on rejecting trials with artifacts.

To begin the exercise, make sure that **15_N2pc_ICA_preprocessed** is still selected in the **EEGsets** panel. Go to the **Compute Averaged ERPs** panel, click **Run** (with the default settings), and name the resulting ERPset **15_N2pc_no_rejection**. You will see a summary window showing you the number of trials that were included in each bin. Close this window, go to the **Bin & Channel Selection** panel (in the ERP tab), select only Channel 32 (HEOG-bipolar), and click **Apply**. You should see something like Screenshot 8.21A.



Screenshot 8.21

Starting just after 200 ms, you can see a positive voltage deviation (leftward eye movement) for the left-target trials and a negative voltage deviation (rightward eye movement) for the right-target trials. (There are also small differences between left- and right-target trials at earlier latencies, but they must be random noise because it takes at least 150 ms for the visual system to find a color-defined target in a visual search array and execute a saccade to it.)

The negative voltage in the right-target average is larger than the positive voltage for the left-target average. This is a fairly common pattern, and it may be related to the widely observed right hemifield bias for spatial attention. The main issue, however, is the overall difference in HEOG activity between left-target trials and right-target trials. To visualize this, we can make a left-target-minus-right-target difference wave. Go to the **Bin Operations** panel, and enter this equation into the top row of the text box: **bin3 = bin1 - bin2 label Left Target Minus Right Target**. Make sure that the **Mode** is set to **Modify Existing ERPset**, and then click **Run**.

If you select only Bin 3 in the **Bin & Channel Selection** panel, you'll see that the difference between the left- and right-target averages is approximately 8 µV between approximately 250 and 450 ms (see Screenshot 8.21B). However, keep in mind that this reflects a mixture of some trials with a large eye movement and some trials without.

Is this a large enough difference to be problematic? To answer this question, we need to think about why horizontal eye movements are problematic for interpreting an N2pc experiment (or a CDA experiment, LRP experiment, etc.). First, we need to

ask whether our N2pc effect might actually consist of HEOG voltage that has propagated from the eyes to the posterior electrodes where the N2pc is ordinarily observed. Fortunately, a great paper by Lins et al. (1993) provides propagation factors for blinks, vertical eye movements, and horizontal eye movements. They don't provide values for the PO7 and PO8 electrodes, where the N2pc was measured in the original ERP CORE paper, but they do provide values for the O1 and O2 electrodes and the P5 and P6 electrodes, which are on either side of the PO7 and PO8 electrodes. Lins et al. reported a propagation of 1% from HEOG-bipolar to O1 and O2 and a propagation of 3% from HEOG-bipolar to P5 and P6. If we split the difference and assume a ~2% propagation to PO7 and PO8, the 8 μV voltage deflection we observed at the HEOG channel in this participant would be expected to lead to a voltage of 0.16 μV at P5 and P6. That's quite a bit smaller than the N2pc, which is typically 1-2 μV, but still as big as some between-condition differences in N2pc amplitude. And in many participants, the HEOG-bipolar voltage would be quite a bit larger than 8 μV.

We also need to consider whether the eye movements caused the target to shift toward the center of the retina, reducing the lateralization that is necessary for seeing an N2pc (because it is defined by the difference between contralateral and ipsilateral channels). Again, the paper by Lins et al. (1993) provides useful information. Specifically, they found that the HEOG-bipolar signal increases by 16 μV for every degree of lateral eye rotation in the average participant (which my lab has confirmed several times). Because the difference between left-target and right-target trials in the average HEOG-bipolar signal was approximately 8 μV, we can conclude that there was a 0.5° difference in eye rotation between left-target and right-target trials, on average. However, this is just an average. It's quite plausible that the target was fully foveated by ~250 ms on a substantial proportion of trials, with little or no eye movement on other trials. On the subset of trials with large eye movements, we can't be certain that a target on the left side of the video display was actually in the left visual field and that a target on the right side of the video display was actually in the right visual field.

If we want to be careful, we need to both decrease the artifactual EOG voltage produced by the eye movements and eliminate trials with large deviations in eye position. Artifact correction can be used to reduce the artifactual EOG voltage (as described in the next chapter), but we need to reject trials with large eye movements to deal with large deviations in eye position.

This page titled 8.16: Exercise- Using the Averaged HEOG to Visualize Consistent Eye Movements is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 8.16: Exercise- Using the Averaged HEOG to Visualize Consistent Eye Movements by Steven J Luck is licensed CC BY 4.0.

8.17: Exercise- A Two-Stage Strategy for Eliminating Small But Consistent Eye Movements

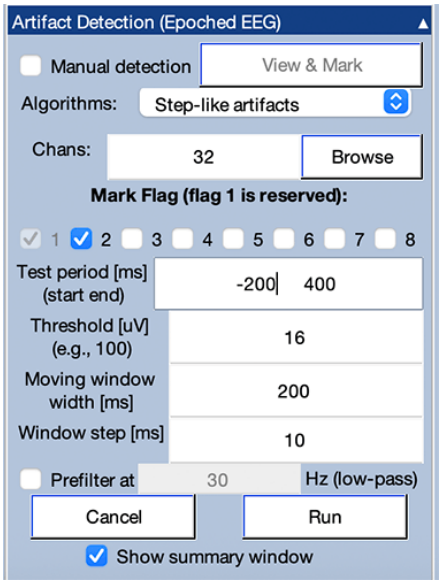
If eye movements were large on some trials and completely absent on others, rejection of trials with eye movements would be easy. However, people may make small but consistent eye movements toward the target. Small eye movements produce small HEOG deflections, and in practice it is difficult to detect eye movements smaller than approximately 1° (16 μV) on single trials in most participants. In this exercise, we'll see how to use a *two-stage procedure* (originally described by Woodman & Luck, 2003) to ensure that the data are not contaminated by small but consistent eye movements toward the target. In Stage 1, we throw out single trials with eye movements of greater than ~1°. In Stage 2, we look at the averaged HEOG-bipolar waveforms for left- and right-target trials to assess the effects of any small eye movements that remain after Stage 1.

Let's start with Stage 1. If we're not thoughtful about the artifact rejection parameters used at this stage, we'll end up rejecting so many trials that we'll need to exclude the participant. One way to minimize the number of rejected trials is to look for eye movements only until the end of the N2pc measurement window. The N2pc measurement window in the ERP CORE N2pc experiment was 200-275 ms, but to keep things simple we'll assume a measurement window of 200-300 ms here. If the eyes move toward the target after 300 ms, this won't impact our N2pc amplitude measurements, so we won't reject trials with those late eye movements. This will give us more trials in our averages.

To get started on Stage 1, make sure that **15_N2pc_ICA_preprocessed_epoched** is selected in the **EEGsets** panel. Go to the **Artifact detection** panel, and set the parameters as shown in Screenshot 8.22. We're specifying a moving window of **200** ms, and the step function gives us the absolute value of the difference in mean voltage between the first and second halves of this window (the first 100 ms and the last 100 ms). We're specifying a Window Step of **10**, which means that we're shifting this window in 10-ms increments. We're specifying a test period of **-200 400** rather than the entire epoch so that we don't have to throw out trials with eye movements after the N2pc measurement window (200–300 ms).

Timing Details

The last moving window being tested will be from 200–400 ms, so an eye movement that starts at 300 ms will be detected. If we ended the test period at 300 ms, the last moving window would be from 100–300 ms, which corresponds to the difference in mean voltage between the 100–200 and 200–300 ms periods. An eye movement that began at, for example, 280 ms would only influence the voltage during the very last part of this period and would probably be missed. But this eye movement would be caught by a moving window from 180–380 ms. In general, if you are using a 200 ms moving window (which works well for eye movements), the window should end 100 ms after the time period of interest.



Screenshot 8.22

We've specified a threshold of **16** µV, which means that we will detect eye movements that are ~1° or larger. I like this threshold because 1° is a nice round number, and this threshold works reasonably well with most adult participants (a higher threshold is needed for participants with noisy HEOG-bipolar signals to avoid rejecting too many trials that don't actually have eye movements; selecting the option to prefilter the data at 20 or 30 Hz can help with such participants).

Go ahead and click **Run** and name the resulting EEGset **15_N2pc_ICA_preprocessed_epoched_step16**. As usual, the first thing to do is look at the window showing the percentage of rejected trials. A total of 12% were rejected, which is very reasonable in terms of not reducing the signal-to-noise ratio very much. Now you should scroll through the data. As before, I recommend selecting only the EOG channels for display. You'll see that the first clear eye movement, in Epoch 70, has been flagged for rejection. The blink that leaked through to the HEOG-bipolar channel in Epoch 73 was outside our window of -200 to 400 ms and was therefore not flagged, which is good. The eye movement in Epoch 78 was also outside our window and was not flagged. That's also good, because the eye movement is too late to impact our N2pc measurement, so we want to keep this trial.

If you keep scrolling, you'll see that the eye movements in Epochs 81, 86, and 88 were flagged. There's a small leftward (positive) eye movement in Epoch 84 that wasn't flagged, but it was after our rejection window, so that's good. If you go through the whole session, you'll see that the step function did an excellent job of flagging clear eye movements that occurred during or prior to the 200–300 ms time period that we plan to use to measure N2pc amplitude.

Now let's see what rejecting these epochs will do to the data quality. Go to the **Compute Data Quality Metrics** panel, select the radio button for **Default parameters** (because the 200-300 ms window typically used for the N2pc is one of the default windows), select the radio button for **Include All Epochs,** and click **Run** to see the table of data quality values without any artifact rejection. Run it again after selecting the radio button for **Exclude epochs marked during artifact detection** so you can see the data quality after rejection of the eye movements.

If you look at the aSME values for the PO7 channel in Bin 1 for the 200-300 ms time period, you'll see that the aSME increased only slightly from the original data (0.4504) to the data excluding the marked epochs (0.4693). So, we've eliminated eye movements that exceeded ~1° of eye rotation without much decline in data quality. That's good!

Now we need to implement Stage 2 of our two-stage process. Stage 2 is designed to deal with the fact that we probably failed to reject a substantial number of smaller eye movements (<1°) that were directed toward the target side. These small eye movements would create a negative voltage over the contralateral hemisphere that might impact our N2pc measurements. They might also change the lateralization of the target for trials on which the target was very close to the fixation point. To assess the possibility of small but consistent eye movements remaining in the data, we need to look at the averaged HEOG-bipolar waveforms for the left-target and right-target trials.

To do this, run the averaging routine, making sure that it's set to exclude epochs marked for rejection. Then make a left-target-minus-right-target difference wave using **ERP Bin Operations** just as you did in the previous exercise. Now select Channel **32**

(HEOG-bipolar) and Bin **3** (difference wave) in the **Bin & Channel Selection** panel. There is still an HEOG deflection, but it's now approximately 3 μV whereas it was approximately 8 μV before we rejected trials with eye movements (Screenshot 8.21A in the preceding section).

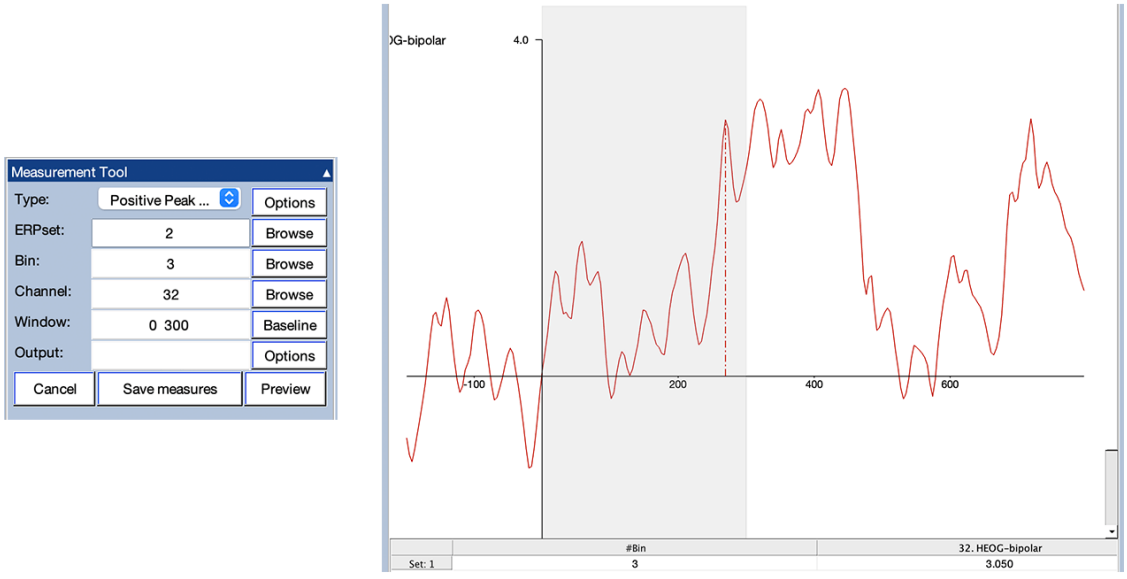
But there is still some consistent eye movement activity in the direction of the target during this time window. We could try to eliminate this residual eye movement activity by decreasing our rejection threshold. However, in my experience, you can never completely remove this activity in most participants without rejecting a huge proportion of trials. For example, I tried reducing the threshold to 8 μV with the present participant, but the residual HEOG signal was quite large even though 55.4% of trials were rejected. Not surprisingly, this also increased the aSME value quite a bit. So, unless you are using a high-resolution eye tracker to reject trials with eye movements, you'll always have some residual HEOG activity after artifact rejection in most participants in experiments with lateralized targets.

The question then becomes, how much residual HEOG activity can we tolerate? If we think of this question in terms of the goals described at the beginning of the chapter, we can break it into two sub-questions: 1) Is enough of the residual HEOG activity being propagated to the N2pc measurement electrodes to create a significant confound? 2) Is the amount of eye rotation implied by the residual HEOG activity large enough to create a significant change in the sensory input?

For my lab's basic science experiments, we can afford to be extremely conservative in our answers to these questions. Our threshold for "good enough" in these experiments is a difference between left-target and right-target trials of <3.2 μV during the N2pc measurement window. In terms of eye rotation, this is an average difference in eye rotation of <0.2° between the left-target and right-target trials (which I like to think of as approximately a difference of ±0.1°). That's a pretty tiny deviation (although we need to keep in mind that this is an average, and the deviation on single trials might be up to 1° with our 16 μV threshold). So, this seems "good enough" in terms of the change to the sensory input.

We ordinarily measure the N2pc at all of the parietal and occipital electrode sites, and the propagation factor is 3% or less from the HEOG-bipolar sites to each of these sites (according to Lins et al., 1993). Thus, a voltage difference of <3.2 μV at HEOG-bipolar corresponds to a voltage difference of <0.1 μV at the sites where we are measuring the N2pc component. That seems "good enough" in terms of any confounding voltage in our N2pc measurements.

Have we succeeded in meeting this 3.2 μV criterion in the present participant? It's difficult to be sure in the current plot, because the peak activity in the difference wave is hovering right around 3 μV. To determine exactly how large the residual HEOG voltage is during the 200-300 ms period, go to the Measurement Tool panel and set it up as shown on the left side of Screenshot 8.23. This will measure the largest positive peak between stimulus onset and the end of the N2pc measurement window at 300 ms. You may need to change the ERPset number if the ERPset you created after artifact rejection isn't ERPset 2 in the **ERPsets** panel.



Screenshot 8.23

Now click the **Preview** button. The preview window will appear and will contain something like the image shown on the right side of Screenshot 8.23. The gray area shows the measurement window. Under the waveform, you can see that the peak voltage in

the measurement window is 3.050 μV, which is less than our threshold of 3.2 μV (barely!). In other words, the small amount of residual eye movement activity for this participant meets our criterion for "good enough."

[Note that the main eStudio window is frozen until you close the preview window, so you should close it now.]

What should you do if the residual eye movement activity for a given participant is >3.2 μV? The first step is to try changing the rejection parameters. Most obviously, you can try reducing the rejection threshold. Sometimes changing the rejection time window can also help. Your goal is to see if you can reduce the residual HEOG activity to <3.2 μV between stimulus onset and the end of the N2pc measurement window (or the measurement window for whatever component you're studying) without rejecting too many trials.

As mentioned before, my lab automatically excludes participants in our basic science experiments if more than 25% of trials are rejected (which includes trials rejected for other reasons, such as blinks). If we can't get the residual HEOG under 3.2 μV without rejecting more than 25% of trials, we exclude the participant from the final analyses. We've used this approach in every N2pc experiment we've conducted for about 30 years, and it has worked very well. We end up excluding approximately 20% of participants, which is tolerable. In our schizophrenia studies, we find that both the patient and control groups make more eye movements than the college-age participants in our basic science studies, so we double our thresholds. That is, we require that the residual HEOG activity is <6.4 μV, and we exclude participants for whom more than 50% of trials were rejected.

What thresholds should you use in your own analyses? I can't answer that question, because it depends on the nature of your research. As always, your choice should be made on the basis of the fundamental goal of increasing your ability to accurately answer the scientific question that your studies are designed to address. And you might use aSME values rather than the percentage of trials rejected to decide whether too many trials have been rejected for a given participant. Whatever criteria you choose, however, it is extremely important that those criteria are set prior to analyzing the data.

I hope that these N2pc exercises have provided you with a clear procedure for minimizing eye movement artifacts in experiments with lateralized targets or responses. But even more, I hope these exercises serve as a good example of how to conceptualize the goals of artifact rejection and how to achieve those goals.

This page titled 8.17: Exercise- A Two-Stage Strategy for Eliminating Small But Consistent Eye Movements is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 8.17: Exercise- A Two-Stage Strategy for Eliminating Small But Consistent Eye Movements by Steven J Luck is licensed CC BY 4.0.

8.18: Matlab Script For This Chapter

I've created a script named **MMN_artifact_rejection_example.m** that shows how to implement the interpolation and artifact detection processes described in this chapter. You can find it in the Chapter_8 folder. It runs on the data from Subjects 1-10 in the ERP CORE MMN experiment. The datasets for these participants are in a subfolder named **MMN_Data**.

The script demonstrates how you can put subject-specific information in an Excel spreadsheet, such as which channels to interpolate and what artifact detection parameters to use, and then have the script read this information and use it to control the interpolation and artifact detection processes. This is a super useful trick!

I didn't spend much time customizing the parameters. You can probably do a better job given what you've learned in this chapter.

Most of these participants have a lot of blinks and would need to be excluded from the final analyses because they exceed our criterion of 25% rejected trials. As I noted before, this is because we planned to use artifact correction rather than rejection for blinks, and we did nothing to minimize blinking. I should also note that Subject 7 has a ton of low-frequency drift (probably coming from the reference electrodes, because it's present in all the EEG channels) and was excluded from the final analyses in the ERP CORE paper.

This page titled 8.18: Matlab Script For This Chapter is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 8.18: Matlab Script For This Chapter by Steven J Luck is licensed CC BY 4.0.

8.19: Key Takeaways and References

Key Takeaways

- The overarching goal in designing an artifact rejection strategy is to maximize the likelihood that you will obtain an accurate answer to the scientific question your study is designed to answer. You can ignore any of my specific suggestions for implementing artifact rejection if you have a better way of reaching that goal. The common ways in which artifacts interfere with that goal and need to be corrected or rejected are:
  - They are a large source of noise and therefore reduce your statistical power.
  - They differ systematically across groups or conditions, creating a confound.
  - They indicate a problem with the sensory input (e.g., closed eyes during the presentation of a visual stimulus).
- In most cases, you will reject trials with artifacts to address one or more of these issues.
- Decreasing the threshold for rejection typically reduces the confounding effects of artifacts and the problems with the sensory input, and it may also reduce the noise caused by the artifacts. However, when the threshold gets too low, the number of trials remaining in the averaged ERP waveforms gets small enough that the data quality suffers. You can use the aSME values to help find the optimal threshold for rejection.
- You will typically want to implement several different artifact detection procedures for each participant so that you can intelligently detect the different types of artifacts. This is often achieved with one procedure for detecting blinks, another for detecting eye movements, and one or two more for detecting C.R.A.P.
- Dealing with small but consistent eye movements is tricky, because small eye rotations are difficult to detect but can be a significant confound in experiments with lateralized stimuli or lateralized responses. The two-stage procedure described near the end of this chapter deals with this by using the greater precision of averaged HEOG waveforms to determine whether the small eye movements that escaped rejection were large enough to have a substantial impact.

References

Addante, R. J., Ranganath, C., Olichney, J., & Yonelinas, A. P. (2012). Neurophysiological evidence for a recollection impairment in amnesia patients that leaves familiarity intact. Neuropsychologia, 50(13), 3004–3014. https://doi.org/10.1016/j.neuropsychologia.2012.07.038

Baker, D. H., Vilidaite, G., Lygo, F. A., Smith, A. K., Flack, T. R., Gouws, A. D., & Andrews, T. J. (2020). Power contours: Optimising sample size and precision in experimental psychology and human neuroscience. *Psychological Methods*. http://dx.doi.org/10.1037/met0000337

Eimer, M. (1994). “Sensory gating” as a mechanism for visuospatial orienting: Electrophysiological evidence from trial-by-trial cuing experiments. *Perception & Psychophysics*, *55*, 667–675. https://doi.org/10.3758/BF03211681

Jas, M., Engemann, D. A., Bekhti, Y., Raimondo, F., & Gramfort, A. (2017). Autoreject: Automated artifact rejection for MEG and EEG data. *NeuroImage*, *159*, 417–429. https://doi.org/10.1016/j.neuroimage.2017.06.030

Kappenman, E. S., Farrens, J. L., Zhang, W., Stewart, A. X., & Luck, S. J. (2021). ERP CORE: An open resource for human event-related potential research. *NeuroImage*, *225*, 117465. https://doi.org/10.1016/j.neuroimage.2020.117465

Lins, O. G., Picton, T. W., Berg, P., & Scherg, M. (1993). Ocular artifacts in EEG and event-related potentials I: Scalp topography. *Brain Topography*, *6*, 51–63.

Luck, S. J. (2012). Electrophysiological correlates of the focusing of attention within complex visual scenes: N2pc and related ERP components. In S. J. Luck & E. S. Kappenman (Eds.), *The Oxford Handbook of ERP Components* (pp. 329–360). Oxford University Press.

Luck, S. J. (2014). *An Introduction to the Event-Related Potential Technique, Second Edition*. MIT Press.

Luck, S. J., Hillyard, S. A., Mouloua, M., Woldorff, M. G., Clark, V. P., & Hawkins, H. L. (1994). Effects of spatial cuing on luminance detectability: Psychophysical and electrophysiological evidence for early selection. *Journal of Experimental Psychology: Human Perception and Performance*, *20*, 887–904.

Luck, S. J., Stewart, A. X., Simmons, A. M., & Rhemtulla, M. (2021). Standardized measurement error: A universal metric of data quality for averaged event-related potentials. *Psychophysiology*, *58*, e13793. https://doi.org/10.1111/psyp.13793

Mangun, G. R., & Hillyard, S. A. (1991). Modulations of sensory-evoked brain potentials indicate changes in perceptual processing during visual-spatial priming. *Journal of Experimental Psychology: Human Perception and Performance*, *17*, 1057–1074.

Nolan, H., Whelan, R., & Reilly, R. B. (2010). FASTER: Fully Automated Statistical Thresholding for EEG artifact Rejection. *Journal of Neuroscience Methods*, *192*(1), 152–162. https://doi.org/10.1016/j.jneumeth.2010.07.015

Talsma, D. (2008). Auto-adaptive averaging: Detecting artifacts in event-related potential data using a fully automated procedure. *Psychophysiology*, *45*(2), 216–228. https://doi.org/10.1111/j.1469-8986.2007.00612.x

Woodman, G. F., & Luck, S. J. (2003). Serial deployment of attention during visual search. *Journal of Experimental Psychology: Human Perception and Performance*, *29*, 121–138.

Yuval-Greenberg, S., Tomer, O., Keren, A. S., Nelken, I., & Deouell, L. Y. (2008). Transient induced gamma-band response in EEG as a manifestation of miniature saccades. *Neuron*, *58*(3), 429–441. https://doi.org/10.1016/j.neuron.2008.03.027

Zhang, G., Garrett, D. R., Simmons, A. M., Kiat, J. E., & Luck, S. J. (2024c). Evaluating the effectiveness of artifact correction and rejection in event-related potential research. *Psychophysiology, 61*, e14511. https://doi.org/10.1111/psyp.14511

Zhang, G., & Luck, S. J. (2023). Variations in ERP data quality across paradigms, participants, and scoring procedures. *Psychophysiology*, *60*, e14264. https://doi.org/10.1111/psyp.14264

Zhang, W., & Kappenman, E. S. (2024). Maximizing signal-to-noise ratio and statistical power in ERP measurement: Single sites versus multi-site average clusters. Psychophysiology, 61, e14440. https://doi.org/10.1111/psyp.14440

This page titled 8.19: Key Takeaways and References is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 8.19: Key Takeaways and References by Steven J Luck is licensed CC BY 4.0.

9: ARTIFACT CORRECTION WITH INDEPENDENT COMPONENT ANALYSIS

Learning Objectives

In this chapter, you will learn to:

- Conceptualize artifact correction in terms of the overarching goal of your research and the specific problems that artifacts pose when you try to reach that goal
- Decompose the EEG into a set of independent components (ICs), identify ICs that represent artifacts, and reconstruct the EEG without the artifactual ICs
- Apply special preprocessing steps to obtain optimal ICs
- Intelligently choose which ICs should be removed from your data
- Combine artifact correction with artifact rejection to maximize data quality, eliminate confounds, and ensure that the sensory input is properly controlled
- Evaluate the effectiveness of the combined artifact correction+rejection procedure

This chapter explains how to use independent component analysis (ICA) to correct certain kinds of artifacts (especially blinks and eye movements). ICA-based artifact correction is a real godsend for experiments in which artifact rejection would throw out too many trials. And it can improve the data quality for almost any experiment by allowing you to include most or all of the trials in your averaged ERPs.

However, ICA-based artifact correction massively changes your data. Every single data point is impacted. And if done improperly, ICA can make your data worse and lead to incorrect conclusions. One analogy I like is using a *backburn* to deal with a wildfire (i.e., starting a controlled fire to eliminate the fuel for the wildfire). If you're not careful with a backburn, it can get out of control and burn what you were trying to save. Similarly, if you're not careful with ICA, it can get out of control and make your data worse rather than better. You really need to know what you're doing with ICA to get the best results and avoid getting burned.

Before I wrote this chapter, I did a lot of reading to make sure I was up to date and that the strategies described in this chapter would reflect the current state of the art. I also spent a lot of time applying ICA to the ERP CORE data and carefully assessing the results. The Makeig group at UCSD are the world's experts at ICA-based artifact correction, so most of the recommendations in this chapter are based on their recommendations. EEGLAB's ICA documentation is an excellent resource, especially the videos created by Arnaud Delorme. The page of informal advice from Makoto Miyakoshi (called Makoto's Preprocessing Pipeline) is also extremely useful. I recommend that you read these sources after reading the present chapter. I also recommend reading the general overview of artifact correction near the end of Chapter 6 of Luck (2014), along with the online supplement to that chapter.

As you read this chapter, keep in mind that the ultimate goal of artifact correction is the same as the ultimate goal of artifact detection, which is to accurately answer the scientific question that the experiment was designed to address. In addition, you should keep in mind the three main problems that we need to address in artifact rejection and correction: reduced statistical power as a result of increased noise, systematic confounds, and sensory input problems. Artifact correction will never be perfect, but it can still help you address these problems (especially combined with intelligent rejection of problematic artifacts that cannot be corrected with ICA). In the last section of this chapter, I describe an approach developed by my lab to evaluate whether the combination of artifact correction and rejection is actually effective (Zhang et al., 2024).

9.1: Data for this Chapter

9.2: Exercise- A First Pass at ICA-Based Blink Correction

9.3: Background- A Quick Conceptual Overview of ICA

9.4: Exercise- Making ICA Work Better

9.5: Exercise- Transferring the Weights, Identifying the ICs, Correcting the EEG, and Interpolating

9.6: Exercise- Deleting C.R.A.P. Prior to ICA

9.7: General Recommendations for ICA-Based Artifact Correction
9.8: Background- Evaluating the Effectiveness of Artifact Correction+Rejection
9.9: Exercise- Determining Whether Blinking is a Confound
9.10: Exercise- Was Blink Correction Effective?
9.11: Exercise- Does Artifact Correction+Rejection Improve Data Quality?
9.12: Matlab Scripts For This Chapter
9.13: Key Takeaways and References

This page titled 9: Artifact Correction with Independent Component Analysis is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 9: Artifact Correction with Independent Component Analysis by Steven J Luck is licensed CC BY 4.0.

9.1: Data for this Chapter

The data we'll be using for the exercises in this chapter can be found in the Chapter_9 folder in the master folder: https://doi.org/10.18115/D5G045.

This page titled 9.1: Data for this Chapter is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 9.1: Data for this Chapter by Steven J Luck is licensed CC BY 4.0.

9.2: Exercise- A First Pass at ICA-Based Blink Correction

I find that it's easiest to understand ICA by starting with an example and then explaining how it works after you've seen it in action. We're going to start with the data from Subject 10 from the ERP CORE MMN experiment, which we looked at in the previous chapter in the section on eye movement detection.

First, reset everything with the **Reset** button so that any previous settings don't change how things look in this exercise. Then set **Chapter_9** to be Matlab's current folder using the **Current Folder** button in the **EEGsets** panel, and load the EEGset named **10_MMN_preprocessed.set**. *Make sure you're getting the data from the Chapter_9 folder rather than the Chapter_8 folder, because the data are preprocessed a little differently but the filenames are the same.*

The dataset you loaded has been high-pass filtered at 0.1 Hz and referenced to the average of P9 and P10 (except for the VEOG-bipolar and HEOG-bipolar channels). Scroll through the EEG to remind yourself what it looks like. You'll see lots of blinks and eye movements and a fair amount of EMG noise.

When we looked at this participant's data in the previous chapter, the EEG had been low-pass filtered. Without the low-pass filtering, you can now more easily see the *spike potential* at the beginning of each eye movement, which is a result of the muscle activity that sets the eyes in motion. This is illustrated in Figure 9.1, which shows the first 5 seconds of data from Subject 10. You can see a brief spike in several channels at the moment when the HEOG voltage starts to change from one level to another (which indicates a change in gaze location from one location to another). You may want to adjust the vertical scale and which channels are displayed so that you can see these voltage changes more clearly.

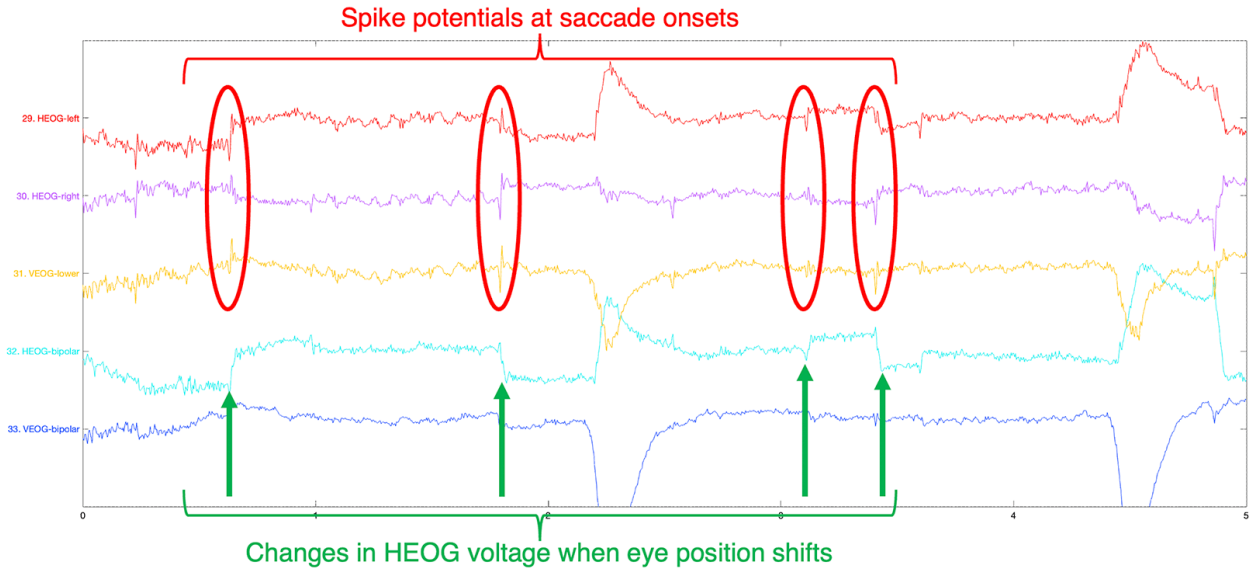
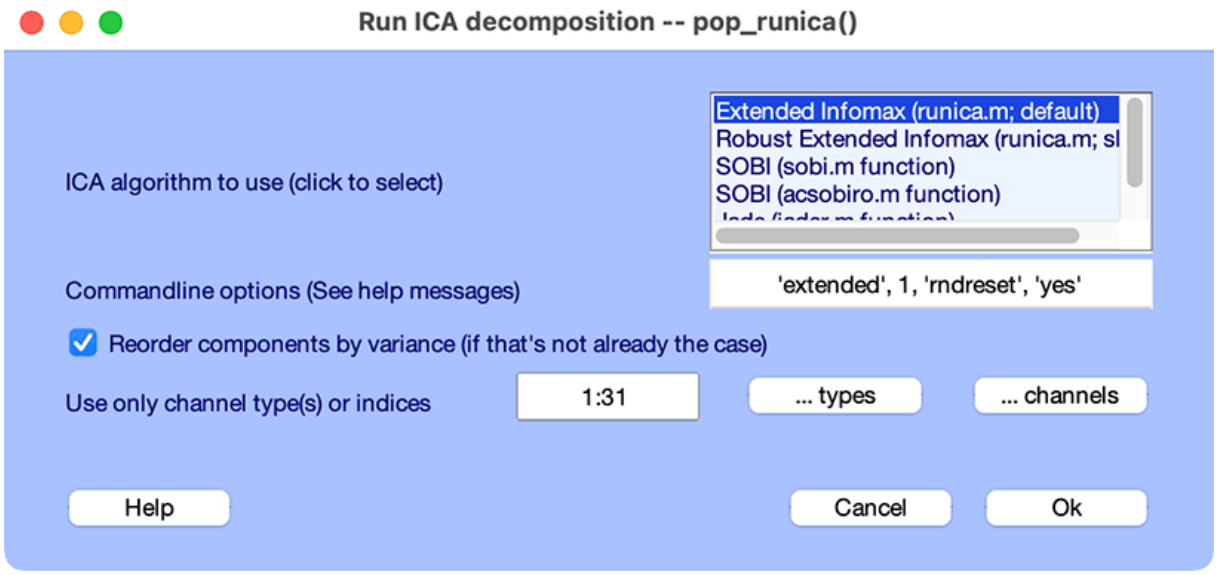


Figure 9.1 Examples of spike potentials and HEOG voltage changes associated with saccadic eye movements.

Now let's start the artifact correction process. Go to the **EEGLAB ICA** panel, click the **Decompose data** button, and set the parameters as shown in Screenshot 9.1. Everything is probably already set correctly by default, except for the channels, which you should set to **1:31**. With this channel list, we're including all the channels that have the same reference, and we're excluding VEOG-bipolar and HEOG-bipolar. ICA assumes that the channels are linearly independent of each other, and the VEOG-bipolar and HEOG-bipolar were created by recombining other channels, so they would violate this assumption. However, they are useful channels to have later so that we can tell when blinks and eye movements occurred even after we've corrected the EEG data.



Screenshot 9.1

Click **OK** to start the ICA decomposition process, which figures out the independent components that best represent the current EEGset. You'll see some text appear in the Matlab command window, followed by a long series of lines that look like this:

step 1 - lrate 0.001000, wchange 16.88431280, angledelta 0.0 deg

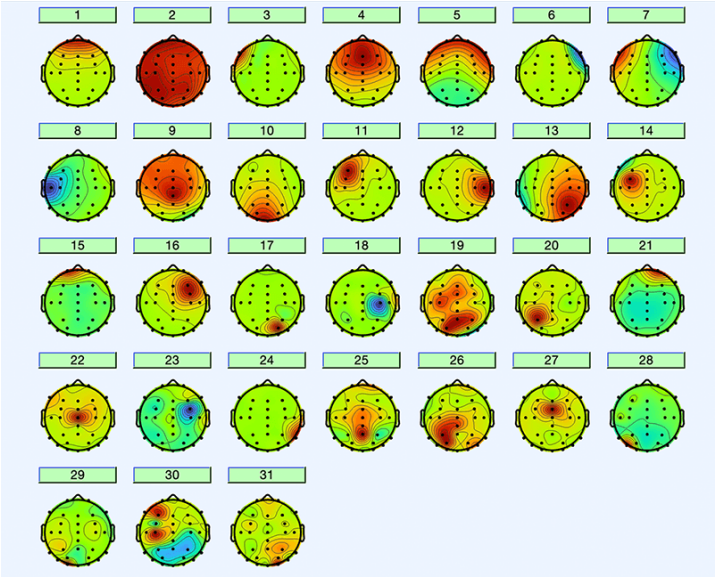
step 2 - lrate 0.001000, wchange 0.90426626, angledelta 0.0 deg

ICA works by training a machine learning algorithm (much like a neural network), and it can take the algorithm a very long time to converge on a solution. The amount of time depends on the number of channels, the noise level of the data, and the speed of your computer. It took over 5 minutes to reach a solution for this dataset on an old laptop, but only about 30 seconds on my current laptop (which has an Apple M4 Pro processor). If you don't want to wait for it to finish, you can kill the process by clicking the **Interrupt** button that should be visible while ICA is running. If that doesn't work, you can usually kill a Matlab process by typing **Ctrl-C**.

If you waited for the decomposition process to finish, you can name the resulting EEGset **10_MMN_preprocessed_ica**. If you didn't wait for the process to finish, you can just load the file **10_MMN_preprocessed_ica.set**, which has the ICA weights that I got when I ran the decomposition.

ICA decomposes the EEG data into a set of *independent components* or *ICs*. These are statistically defined components, not ERP components, and they may not be biologically meaningful. I will refer to them as *ICs* rather than as *components* to maintain this important distinction.

Each IC is a *spatial filter*. When we apply an IC to the distribution of voltages over the scalp at a given moment in time, the output is the magnitude of that IC at that moment in time. Conveniently, ICA creates scalp maps of the ICs showing how each channel is weighted by the IC. Take a look at the scalp maps now by clicking **Inspect/label ICs** in the **EEGLAB ICA** panel. Tell the tiny window that appears that you want to plot components **1:31**. You should then see a new window that looks like Screenshot 9.2. You'll also see a window that says **While you are using the Inspect/Label ICs tool, the main ERPLAB Studio window will be frozen. Click "Okay" when you have closed the Inspect/Label ICs tool.**



Screenshot 9.2

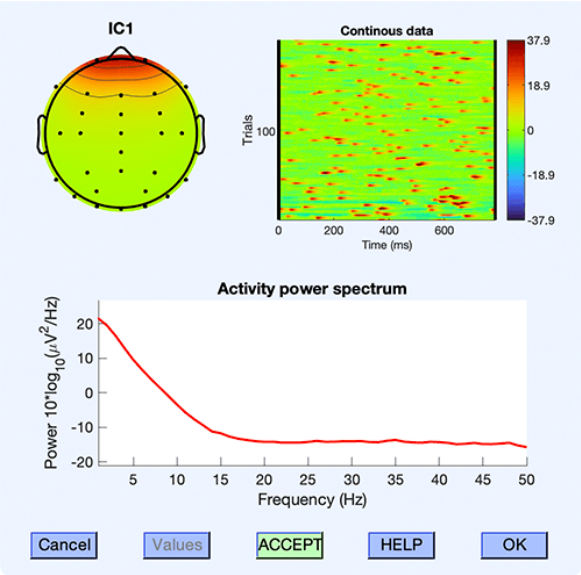
If you loaded the EEGset that already had the weights in it, the scalp maps you see should look exactly like those in the screenshot. However, if you ran the component decomposition process yourself, the maps might look a little different. This is because the ICA machine learning algorithm starts from a set of random weights and therefore doesn't yield exactly the same results every time. If things are working well, then the results should be similar if you repeat the decomposition process multiple times. If you get very different results, then something is wrong (usually either very noisy data, not enough data, or a linear dependency among your channels).

There are some things that may differ across repetitions that don't indicate a problem. First, the ordering of the ICs might change a little. For example, the scalp maps for ICs 13 and 14 might be swapped in your data. Second, there might be small changes in the weights for a given scalp map. Third, the polarity of one or more maps might be inverted (which we will discuss in more detail later). These are normal and no cause for concern. However, if the maps you obtained are radically different from those in Screenshot 9.2, then something is wrong. Perhaps you forgot to specify channels **1:31** in the window of parameters that appeared when you clicked **Decompose data**.

There are 31 ICs because we gave the algorithm data from 31 channels. Ordinarily, the number of ICs is equal to the number of channels. This is necessary to make the math work. The fact that the number of ICs equals the number of channels reflects an important way in which ICA is an imperfect method for identifying the true components underlying your data: Putting more or fewer electrodes on the scalp doesn't change the number of sources of brain activity, and yet it changes the number of ICs. We will discuss this more later.

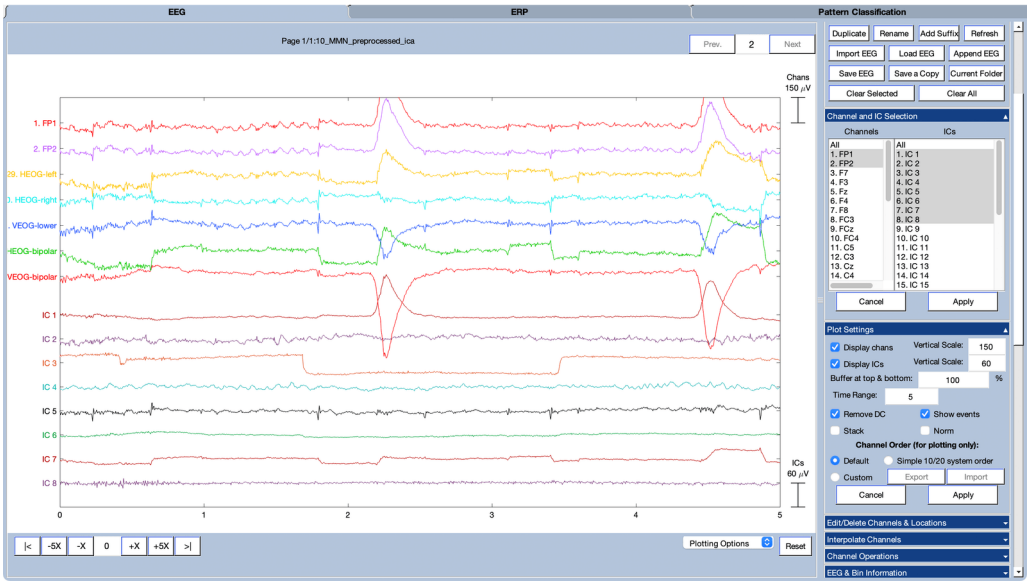
After the decomposition algorithm finishes, it reorders the components in terms of the amount of variance in the data they account for, with IC 1 accounting for the greatest variance. If the participant blinked frequently, the IC corresponding to the blinks is usually in the top five ICs (and is often IC 1). In our current example, IC 1 has the kind of frontally focused scalp distribution that you'd expect for a blink. But is this really a blink IC, or is it some other kind of artifact, or maybe even frontally distributed brain activity?

There are a few things we can check to determine whether IC 1 reflects blinks. If you click the number above IC 1, a new window will pop up showing you more details about this IC. In the upper right of the window, you'll see a heat map showing how the magnitude of this IC varies over time. The X axis is time within a "trial" ("trials" are arbitrary time periods when continuous data are used). The Y axis is time over the course of the recording. It's labeled **Trials** but actually corresponds to time in continuous data. You can see many small red blobs. Each of those is a brief time period in which the magnitude of IC 1 was large. This is exactly what we'd expect for blinks: many brief blips of voltage. The window also shows the frequency spectrum. The spectrum for IC 1 shows a gradual decline as the frequency increases, leveling out around 20 Hz, which is also characteristic of blinks (but is also seen for many types of non-blink activity). Close this window, then close the window with the IC scalp maps, and click **Okay** in the window that says **While you are using the Inspect/Label ICs tool...** to unfreeze the main eStudio window.



Screenshot 9.3

The next thing you can do (which I find the most informative) is to directly compare the time course of the ICs with the time course of the EEG and EOG signals. Start by going to the **Plot Settings** panel, checking the box for **Display ICs**, setting the **Vertical Scale** to **150** for the channels and **60** for the ICs, and clicking Apply. Then go to the **Channel and IC Selection** panel, select FP1, FP2, and all the EOG channels from the **Channels** column, select ICs 1-8 from the **ICs** column, and click **Apply**. The result should look like Screenshot 9.4. Now you can look at how the activations for ICs 1-8 vary over time and compare them to the key EOG and EEG signals at the corresponding time points. Try scrolling through the first 60 seconds of data to see whether IC 1 matches the time course of the blinks in the VEOG and frontal EOG channels.

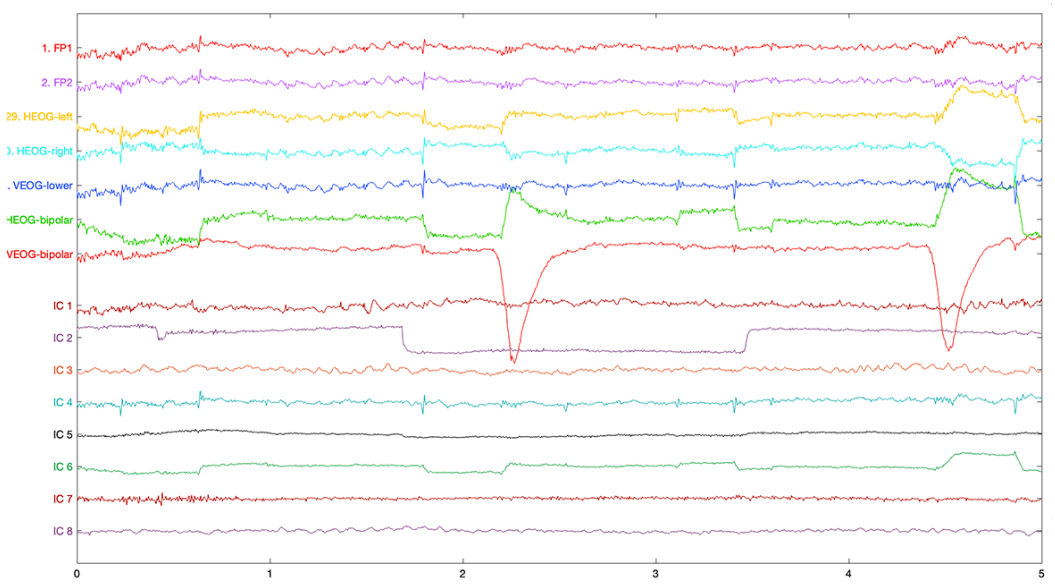


Screenshot 9.4

The first thing to notice is that each blink that you can see in the EOG and EEG channels is accompanied by a deflection with the same shape in IC 1 (e.g., at 2.2 and 4.4 seconds in Screenshot 9.4). You should also notice that there are occasional step-like deflections in both VEOG-bipolar and IC 1 (e.g., between 16 and 17 seconds). These are vertical eye movements. Often, but not always, blinks and vertical eye movements will be captured by the same IC. Given the scalp distribution of IC 1 and its close correspondence in time with the blinks and eye movements in the VEOG-bipolar channel, we can be quite sure that this IC reflects vertical EOG activity.

Now let's remove IC 1 from the EEG/EOG data to eliminate the blinks and vertical eye movements. Ordinarily, we would also remove other artifact-related ICs at this point (e.g., horizontal eye movements), but we'll just focus on blinks and vertical eye movements for now. To do this, go to the **EEGLAB ICA** panel and click the **Remove ICs** button. A small window will appear; put **1** in the text box labeled **List of component(s) to remove from data** so that we remove activity corresponding to IC 1, and click **OK**. Name the resulting EEGset **10_MMN_preprocessed_ica_pruned** and save it to disk with this name.

As shown in Screenshot 9.5, the blinks should now be eliminated from all channels except for the VEOG-bipolar channel (which we excluded from ICA when we specified channels 1:31 in the ICA decomposition step). Scroll through the first 60 seconds of data. You can still see ordinary EEG deflection and horizontal eye movements, but the blinks are largely gone. It's like a miracle!



Screenshot 9.5

You can also see that the vertical eye movements are largely gone as well. For example, if you go back to the previous EEGset, you'll see a sudden step in voltage at 5.7 seconds, but this step is gone after correction.

This really is a minor miracle. We've nearly eliminated the contribution of blinks and vertical eye movements to the EEG at every single time point. Now we don't need to reject trials with blinks, which should increase the data quality substantially.

This page titled 9.2: Exercise- A First Pass at ICA-Based Blink Correction is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 9.2: Exercise- A First Pass at ICA-Based Blink Correction by Steven J Luck is licensed CC BY 4.0.

9.3: Background- A Quick Conceptual Overview of ICA

Chapter 6 in the 3rd edition of An Introduction to the Event-Related Potential Technique provides a conceptual overview of how ICA works in general and how it is applied to artifact correction. Here I'll provide a quick summary. Several different algorithms are available for performing the ICA decomposition. For the vast majority of cases in which ICA is used for artifact correction, there aren't big differences among the algorithms, so here I'll focus on EEGLAB's default ICA algorithm (*Infomax*, implemented with the **runica** routine).

The first thing you should know is that ICA is purely a statistical technique, and it was not developed for neural data per se. It knows nothing about brains or electricity. It doesn't know that the data are coming from electrodes or where the electrodes are located. Most ICA algorithms don't even know or care about the order of time points. They just see each time point as a set of N abstract variables, one for each of the N channels. Infomax uses a machine learning algorithm (much like a neural network) that learns a set of N ICs that are maximally independent when applied to the data.

By *maximally independent*, I mean that the activation level of one IC provides no information about the activation levels of the other ICs at that time point. For example, if the blink-related voltage level at each time point does not predict other sources of activity at the same time point, blinks will likely be extracted as a separate IC. However, it's not a problem if blink activity at one time point predicts activity from other sources at earlier or later time points.

ICA learns an *unmixing matrix*, which converts the EEG data at a given time point to the activation level of each IC. The inverse of the unmixing matrix is the *mixing matrix*, which is just the scalp distribution of each IC. You can also think of the scalp distribution of an IC as a set of weights. The voltage produced by an IC in a given channel at a given time point is the activation level of the IC at that time point multiplied by the weight for that channel. Some randomness is applied to the learning algorithm, so you won't end up with exactly the same set of ICs if you repeat the decomposition multiple times.

An important practical consideration is that the machine learning routine needs a lot of data to adequately learn the ICs. The EEGLAB team has provided an informal rule for this, which is that the number of time points in the dataset must be at least 20 x (# channels)^2. It's probably the number of minutes of data that matters rather than the number of time points, but the key thing to note is that the number of channels is squared. This means that doubling the number of channels requires four times as much data. For example, you would need four times as many minutes of data for a 64-channel recording as for a 32-channel recording (and sixteen times as much data for a 128-channel recording as for a 32-channel recording).

ICA is somewhat like principal component analysis (PCA). However, whereas PCA tries to lump as much variance as possible into the smallest number of components, ICA tries to make the components maximally independent. ICA is also like PCA insofar as it just takes the dataset and represents it along a different set of axes. You can go from the original data to the ICA decomposition with the unmixing matrix, and then you can apply the mixing matrix to the ICA decomposition and perfectly recover the original data.

This decomposition-and-recovery sequence is how ICA corrects for artifacts. After running the ICA decomposition to get the ICs, you simply set one or more of the ICs to have an activation of zero at each time point and then use the mixing matrix to recover the original data (but without the artifactual ICs). This means that ICA influences your data at every single time point. When you remove a blink IC, ICA doesn't just find time periods with blinks and correct the data during those time periods. It reconstructs your EEG data at every time point, but with the artifactual ICs set to zero. There will be some nonzero activity in the blink IC at each time point, so zeroing this IC at each time point means that the data will be changed at least slightly at every time point. This is actually good, because there may be quite a lot of EOG activity between blinks as a result of small changes in eye rotation or eyelid position, and ICA will remove this non-neural activity when you remove the IC corresponding to blinks.

ICA makes several important assumptions (see Luck, 2014), but two are particularly important to know about. The first is that the scalp distribution of a given source of activity must remain constant over the entire session. For example, we can assume that the locations of the eyes relative to the electrode sites will not change over the course of a session (unless there is some kind of catastrophe), so blinks and eye movements meet this criterion. Similarly, the location of the heart relative to the electrodes doesn't change over time, so the EKG artifact also meets this criterion. However, the scalp distribution produced by skin potentials will depend on which sweat pores are activated, which may change over time, so skin potentials do not meet this assumption. By the way, this assumption means that you must perform ICA separately for each participant (because the scalp distributions will differ at least slightly across participants).

There is dispute in the literature about whether ICA works well with EMG. The argument against using ICA with EMG is that different muscle fibers may contract at different time points, changing the scalp distribution. The argument for using ICA is that the scalp distribution does not actually change very much over time. To be on the safe side, my lab doesn't use ICA for EMG. We minimize EMG by having participants relax during the EEG recording, and we can filter out the remaining EMG so that it has minimal impact on our results. However, if you cannot avoid having a lot of EMG in your data, and you can't filter it out without creating other problems (e.g., because you're looking at high-frequency ERP activity), you can read the literature and decide for yourself whether the benefits of using ICA for EMG outweigh the costs.

A second key assumption of ICA is that the number of true sources of activity is equal to the number of channels. This is related to the fact that the number of ICs must be equal to the number of channels in order for the math to work.

Exceptions Make the Rule

There are occasional exceptions to the rule that the number of ICs is equal to the number of channels, particularly when you are using the average of all sites as the reference. See Makoto's Preprocessing Pipeline or EEGLAB's ICA documentation for details.

As I mentioned earlier, the fact that the number of ICs must equal the number of channels means that ICA is an imperfect method. You don't change the number of sources of activity when you add or subtract electrodes! Also, there will always be more sources of activity in the EEG signal than there are channels (because each synapse in the brain is a potential source of activity). As a result, ICA will lump multiple true components into the same IC. In addition, a single true source may also be split among multiple ICs. So, you will definitely have lumping of true components, and you will likely have some splitting as well.

Given the failure of EEG data to meet this second assumption, you may wonder whether it is valid to use ICA for artifact correction. As famously noted by the statistician George Box, all statistical models are wrong, and the question is not whether they are correct but whether they are useful (Box, 1976). In practice, ICA is useful for correcting some kinds of artifacts despite the invalid assumptions. The saving grace of ICA is that the lumping and splitting problems are minimal for components that account for a lot of variance (e.g., components that are both large and frequently occurring). Most participants blink a lot, and blinks are very large, so ICA typically works very well for blinks. Depending on the experiment and the participant, eye movements can be large or small, and they can be frequent or rare. In my experience, ICA works only modestly well for eye movements, and it can't correct for the change in sensory input produced by the change in gaze position, so we only use ICA to correct for eye movements when necessary. However, I recently came across a nice paper by Dimigen (2020) showing that ICA can work quite well for large and frequent eye movements when the right preprocessing steps are applied prior to the ICA decomposition (as I'll discuss in more detail later). Drisdelle et al. (2017) also provide evidence that ICA can work well for eye movements in certain types of paradigms.

ICA can be applied either to continuous or epoched EEG. When my lab first started using ICA many years ago, I emailed Scott Makeig and Arnaud Delorme to get their advice, and they recommended applying it to the continuous EEG. They still give this advice today in the EEGLAB documentation. You can apply ICA to epoched data if necessary, but the epochs must be at least 3 seconds long (e.g., -1000 to +2000 ms). Adjacent epochs cannot contain the same data points, so this means that you must have relatively long trials for this approach to work. If you get your pipeline set up properly (see Chapter 11 and Appendix 3), there isn't any reason why you'd need to apply ICA to epoched data, so my view is that the safest thing to do is to apply it to the continuous data. As described in the text box below, there may also be a practical advantage.

A Practical Advantage

Over the years, we've found a significant practical advantage to doing ICA at the earliest possible stage of EEG preprocessing (which means applying it to continuous EEG, because epoching is a relatively late stage). Specifically, ICA is a time-consuming process that you don't want to repeat if you can possibly avoid it. If you need to change some of your processing steps after you've already analyzed your data once, putting ICA at the earliest possible stage minimizes the likelihood that this change will require repeating the ICA.

The ICA decomposition process typically takes somewhere between 2 minutes and 2 hours depending on the nature of your data and your computer. If you need to process data from 30 participants, this is now between 60 minutes and 60 hours. That can be done overnight while you're asleep, but another 2-20 minutes of human effort are required for each participant to make sure that the decomposition has worked properly and to determine which ICs should be removed. That's 60 to 600 minutes of your precious time.

> What's the likelihood that you will need to re-process your data? In my experience, the likelihood is close to 100%! Reviewers always seem to want some change (or some secondary analysis). And when you're new to ERP analysis, you're likely to do something that is less than optimal and will require a re-analysis. But if you've done the artifact correction at the earliest possible point in your processing pipeline, chances are good that you won't need to repeat this time-consuming part of your pipeline.

A key step in ICA-based artifact correction is to determine which ICs correspond to artifacts and should be removed. There are automated algorithms for this, but I recommend doing it manually for the vast majority of studies. As you will see, you need to carefully determine whether a given IC should be removed, which requires taking into account the three underlying goals of artifact rejection and correction, and this often goes beyond what an algorithm can do.

ICA-based artifact correction massively changes your data, and we know we are violating at least one of its assumptions, so I recommend being conservative in using it. We almost always use it for blinks, and we sometimes use it for eye movements, but we don't ordinarily use it for other kinds of artifacts. If we frequently encountered large EKG artifacts, we'd probably use ICA for those as well. Some labs use ICA for anything that looks "weird", but I personally don't like that approach. There are other ways of dealing with these other types of artifacts, and I just don't trust an algorithm to solve every problem in my data.

Finally, don't forget Hansen's Axiom: There's no substitute for good data. Do everything you can to minimize artifacts during the recording, and then you won't end up getting an ulcer from worrying about how to deal with a ton of artifacts during the analysis.

This page titled 9.3: Background- A Quick Conceptual Overview of ICA is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 9.4: Background- A Quick Conceptual Overview of ICA by Steven J Luck is licensed CC BY 4.0.

9.4: Exercise- Making ICA Work Better

Our first attempt at ICA-based blink correction was quite simplistic, and as a result the decomposition was not very good (as evidenced by the irregularly shaped scalp maps for some of the ICs in Screenshot 9.2). The main reason for this is that our dataset (like most EEG datasets) contains some really large activity that is difficult for ICA to handle because it is either infrequent or lacks a consistent scalp distribution (e.g., skin potentials, movement artifacts, and other miscellaneous C.R.A.P.). In this exercise, we'll look at an approach that will minimize the effects of those large noise sources and also make the decomposition run faster. It's a win-win!

There are several steps for making the ICA decomposition work better. They rely on a little trick, which is to create a new dataset in which you will remove signals that will be problematic for ICA, do the decomposition on this new dataset, and then transfer the ICA weights back to the original dataset. This approach was controversial when I wrote the second edition of *An Introduction to the Event-Related Potential Technique*, so you won't see it discussed in Chapter 6 of Luck (2014). However, most experts have converged on the idea that this approach is both theoretically justified and practically useful. I've now used it many times myself, and I have seen only benefits and no costs. That's pretty unusual, because I tend to be pretty skeptical about methods like ICA. In fact, some of my former grad students have a hard time believing that I now recommend using it.

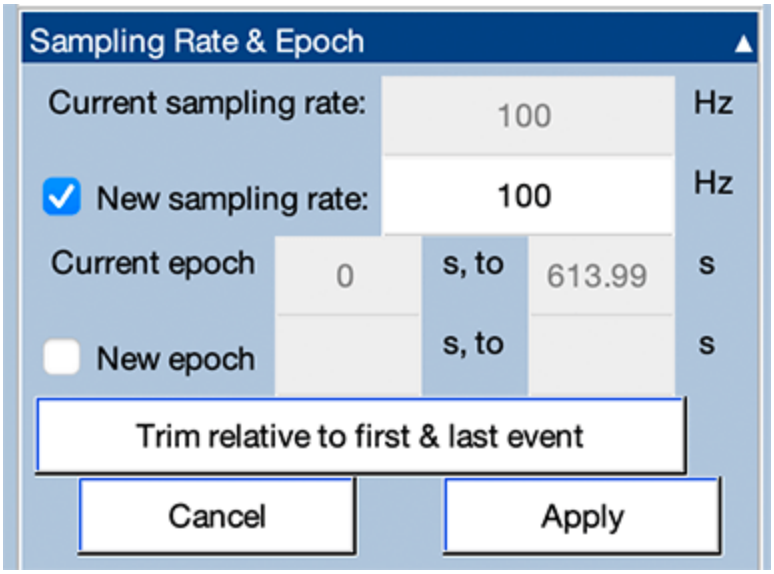
The first thing we're going to do is heavily filter the data, using filter settings that I would never ever EVER ordinarily use for ERP experiments. However, the filtered data will be used only to perform the ICA decomposition, and then we'll transfer those ICA weights to the original minimally-filtered data. Several studies have shown that this heavy filtering leads to more accurate artifact correction (Dimigen, 2020; Klug & Gramann, 2021; Winkler et al., 2015).

Why Heavy Filtering is OK for the ICA Decomposition

Filters change the time course of an underlying IC, but not the scalp distribution. Heavy filtering is therefore fine for the data used for the ICA decomposition, because we only care about scalp distributions at this stage, not timing. Once we have the IC weights, which are entirely spatial, we'll transfer these weights back to the original data without the temporal distortion caused by the filtering.

To get started, reset everything with the **Reset** button, and then load the original EEGset for Subject 10 (**10_MMN_preprocessed**). Go to the **Filtering** panel and specify a high-pass cutoff of 1 Hz and a low-pass cutoff of 30 Hz, with a slope of 48 dB/octave. The option for removing the DC offset should be set by default, and you should leave it on. Run the routine and name the resulting EEGset **10_MMN_preprocessed_filt**. We've now eliminated most of the skin potentials and EMG, both of which may violate the assumptions of ICA and interfere with the decomposition.

Our next step is to downsample the data from 256 Hz to 100 Hz. This just makes the ICA decomposition run faster (as does the reduction of noise produced by the filtering). To do this, go to the **Sampling Rate & Epoch** panel and set it up as shown in Screenshot 9.6, entering **100** as the new sampling rate. Click **Apply** and then name the resulting EEGset **10_MMN_preprocessed_filt_100Hz**. Note that this step doesn't change the end result, but it does make things go faster.

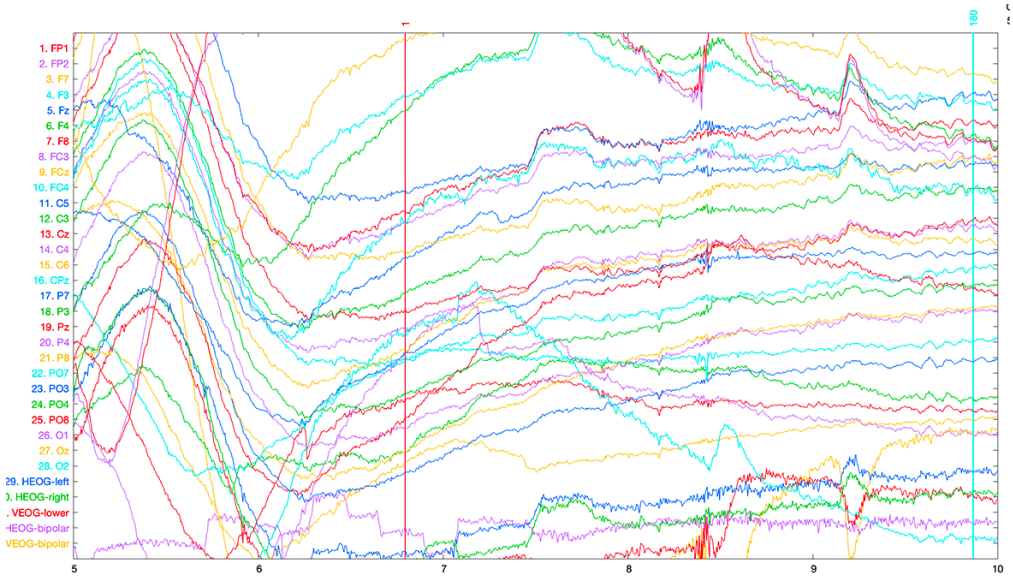


Screenshot 9.6

The next thing we're going to do to improve ICA is remove the segments of data during breaks, when the participants may be moving, scratching their heads, chewing, etc. The voltages produced by these actions are typically large but don't have a consistent scalp distribution. EEGLAB allows you to manually select and delete these time periods from a continuous dataset. ERPLAB adds an automated procedure for this. It just finds periods of time without any event codes and deletes them. Let's give it a try.

First, let's see what the break periods look like in the original data. Select **10_MMN_preprocessed** in the **EEGsets** panel and scroll through several 5-second time periods. You can see that this EEGset begins with a long break period, because the stimuli don't begin until just after 15 seconds. (There is an event code 1 just after 22 seconds, but that's not the beginning of the stimuli.) You can see tons of blinks and eye movements prior to the onset of the stimuli. You can also see a large EMG burst between 23 and 24 seconds. Once the stimuli start, the EEG and EOG signals start looking a lot cleaner.

Subject 10 actually has pretty clean EEG during the break periods. If you load **6_MMN_preprocessed**, you'll see some crazier noise prior to the onset of the stimuli (see Screenshot 9.7). This is the kind of thing that really interferes with the ICA decomposition process.

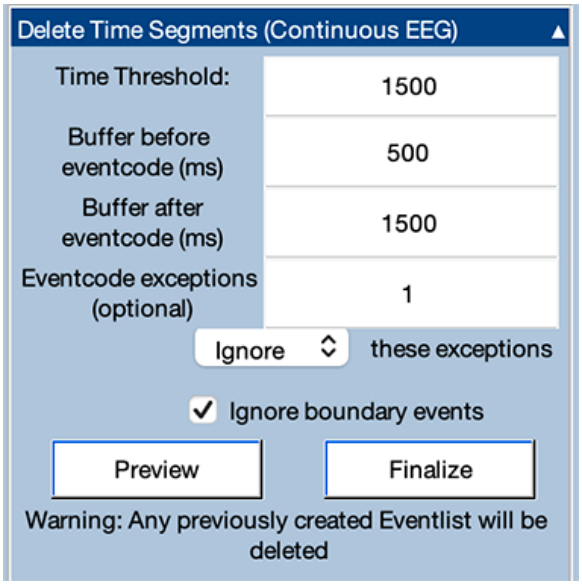


Screenshot 9.7

Now select **10_MMN_preprocessed_filt_100Hz** in the **EEGsets** panel, go to the **Delete Time Segments** panel, and set it up as shown in Screenshot 9.8.

This includes specifying **1500** as the **Time Threshold**. This tells the routine to delete segments in which there are no event codes for at least 1500 ms. The event codes in the MMN experiment occur every ~500 ms, so periods of 1500 ms without an event code must be breaks. Of course, you'd need to use a longer value for experiments with a slower rate of stimuli. Also specify **500** as the amount of time prior to the first event code in a block of trials. This will make sure that we have at least 500 ms for the prestimulus baseline period for the first event in a trial block. Similarly, specify **1500** ms as the time period after the last event code in a block so that we have a good period of data following this event code.

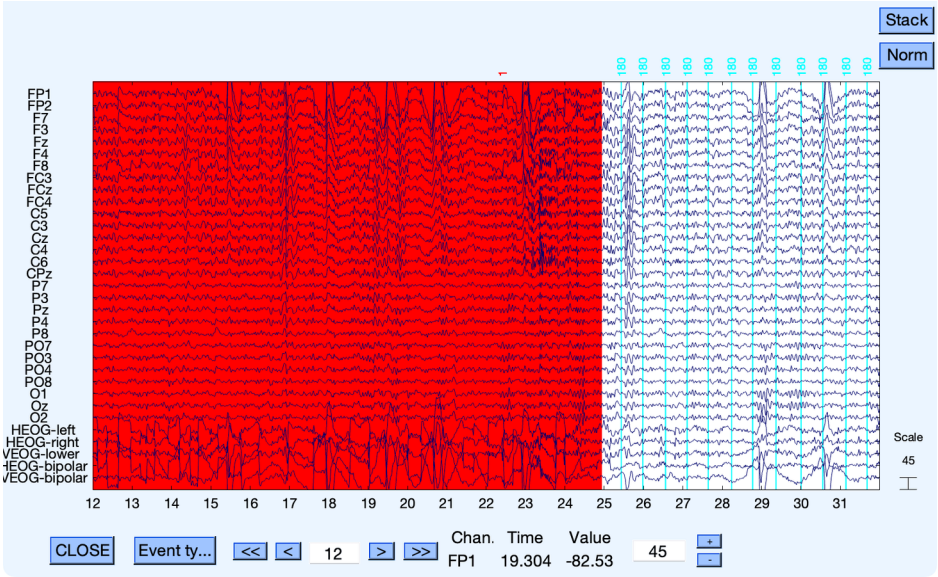
Enter **1** in the field for **Eventcode exceptions**. This tells the routine to ignore this event code. Our stimulus presentation program sent some event codes with a value of 1 during the breaks, and we want those to be ignored when looking for break periods. Similarly, check the box for ignoring boundary events. Boundary events occur whenever there is a temporal discontinuity in the data. Most commonly, this happens when you have a separate data file for each block and then concatenate them together into a single file. When you do this, a boundary event is inserted at the time point of the transition between one block and the next. Boundary events often occur during breaks, so we want them to be ignored when we're looking for long periods between event codes.



Screenshot 9.8

Once you have everything set, click the **Preview** button. This will bring up a new window showing the EEG, with sections that will be deleted marked with a red background. As shown in Screenshot 9.9, the first part of the recording is marked in red until just before 25 seconds. It shifts to white 500 ms before the first event code 180 (ignoring the event code 1). That's because you indicated that there should be a 500 ms buffer before the first event code.

If you type 600 into the text box at the bottom of the window to skip ahead to 600 seconds, you'll see that the break period at the very end of the recording is marked in red (starting just after 610 seconds). The red region starts 1.5 seconds after the last event code. That's because you indicated that there should be a 1500 ms buffer after the last event code. We have only one block in the MMN experiment, so nothing else is marked in red.

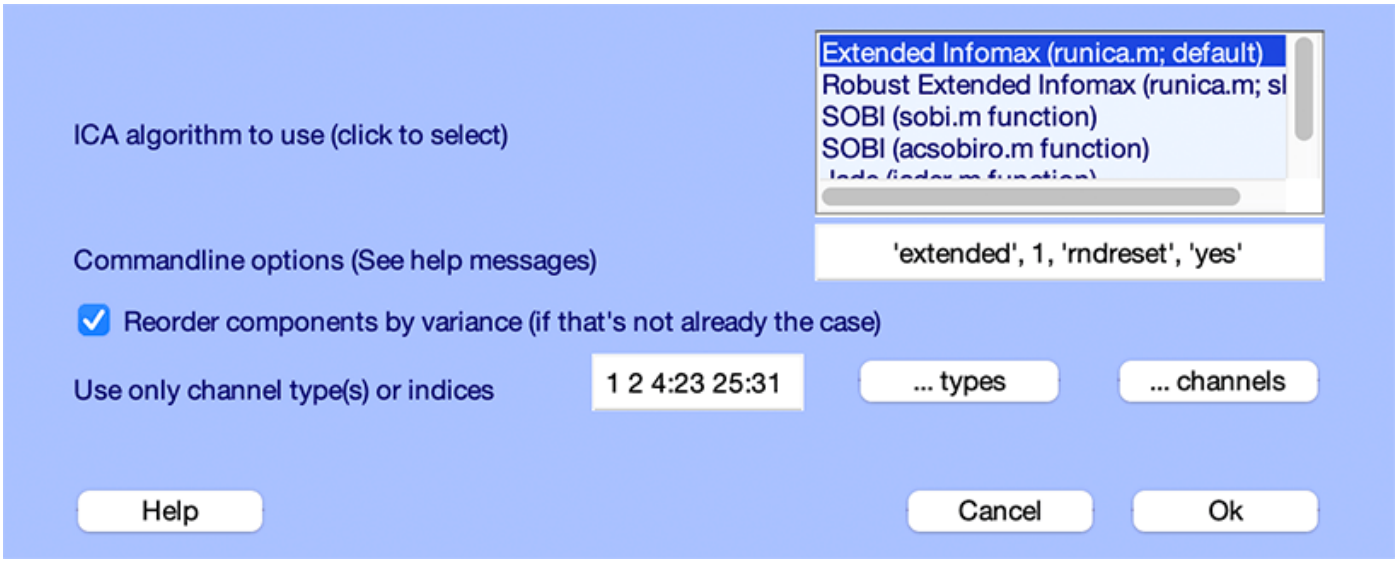


Screenshot 9.9

Now close the window showing the marked portions of the EEG, click the Finalize button in the **Delete Time Segments** panel, and name the new EEGset **10_MMN_preprocessed_filt_100Hz_del**.

Our final pre-ICA step is to deal with bad channels. F7 and PO4 both show some crazy behavior in this EEGset. If we leave them in the data, we'll end up with an IC that just represents F7 and another that just represents PO4. In fact, you can see this in our original ICA decomposition in Screenshot 9.2, where IC 3 just represents the F7 channel and IC 17 just represents the PO4 channel. We're going to interpolate these channels eventually, so why include them in the ICA decomposition? They're just sucking up ICs that would be better spent on valid data. We could interpolate these channels prior to the ICA decomposition, but that would create a linear dependency among the channels that might mess up the decomposition. Instead, we'll just exclude them from the ICA decomposition process, and then we can interpolate them after artifact correction is complete.

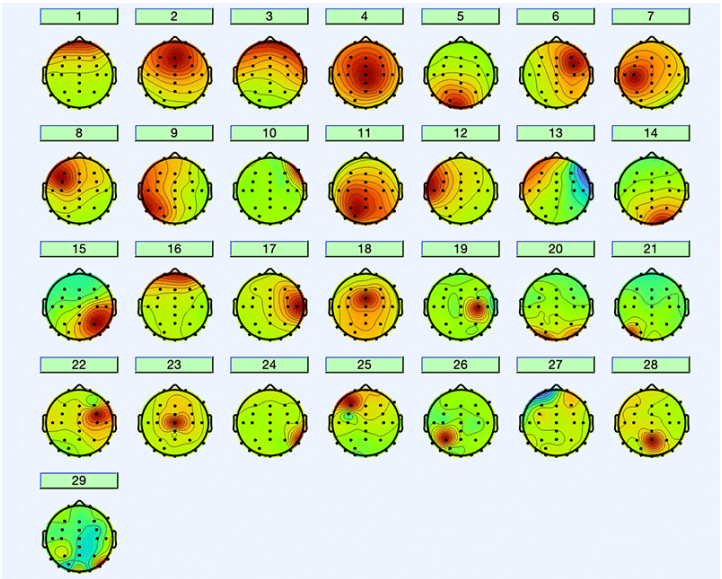
It's now time to apply the ICA decomposition to this EEGset by going to the **EEGLAB ICA** panel and clicking the **Decompose data** button. We're going to exclude Channels 3 and 24 (the channels for F7 and PO4, respectively), and we're also going to exclude Channels 32 and 33 (the bipolar EOG channels). To do this, type **1 2 4:23 25:31** into the text box labeled **Use only channel type(s) or indices**. Make sure everything else is set up as shown in Screenshot 9.10, and click **Ok** to start the decomposition process. As before, it might take several minutes. When it is done, name the resulting EEGset **10_MMN_preprocessed_filt_100Hz_del_ICAweights**. If you don't want to wait for it to finish, you can just load the EEGset with that name from the Chapter_9 folder, which I created after running the decomposition.



Screenshot 9.10

Once you have this EEGset, you can view the IC scalp maps by clicking **Inspect/label ICs** in the **EEGLAB ICA** panel. Tell the tiny window that appears that you want to plot components **1:29** (there are only 29 because we left out the two EEG channels that will need to be interpolated and the two bipolar EOG channels). You should then see a new window that looks like Screenshot 9.11. You'll also see a window that says **While you are using the Inspect/Label ICs tool, the main ERPLAB Studio window will be frozen. Click "Okay" when you have closed the Inspect/Label ICs tool.**

Your scalp maps may differ slightly from those shown in Screenshot 9.11, especially the ordering and polarity of the ICs. These maps are much nicer than our original maps (Screenshot 9.2). Almost all have either a single focus with a largely monotonic decline (e.g., IC 1 and IC 2) or a bipolar configuration (e.g., IC 13 and IC 25). IC 20 has two foci that are approximately mirror-symmetric across the left and right hemispheres, which is also a perfectly normal pattern (and arises when the two hemispheres operate synchronously). The only irregular-looking map is for IC 29. Because the maps are ordered according to the amount of variance they explain, you shouldn't worry if the last couple of maps aren't perfect.



Screenshot 9.11

When you're done looking at the IC maps, close the window with the maps and click **Okay** in the window that says **While you are using the Inspect/Label ICs tool...** to unfreeze the main eStudio window.

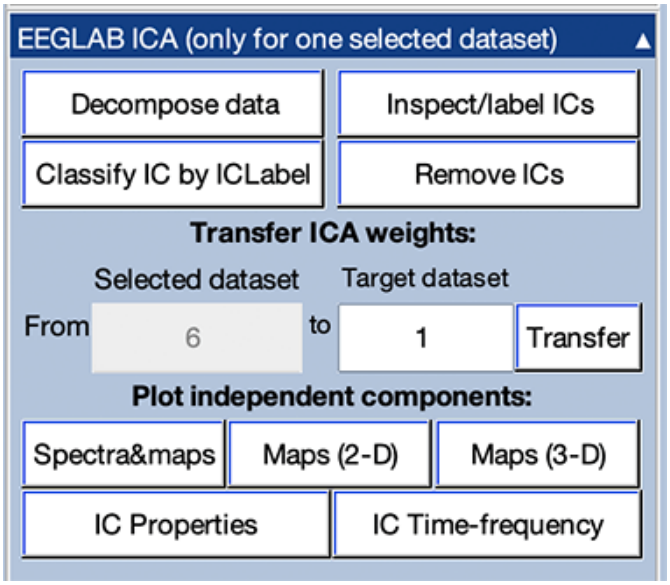
This page titled 9.4: Exercise- Making ICA Work Better is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 9.5: Exercise- Making ICA Work Better by Steven J Luck is licensed CC BY 4.0.
- 9.2: Exercise- A First Pass at ICA-Based Blink Correction by Steven J Luck is licensed CC BY 4.0.

9.5: Exercise- Transferring the Weights, Identifying the ICs, Correcting the EEG, and Interpolating

Our goal in heavily filtering the data was to improve the ICA decomposition, but as was described in the chapter on filtering, this kind of filtering is usually a bad idea for ERP analyses. So, we're going to take the weights from our optimized ICA decomposition and transfer them to the original dataset, which has not been distorted by heavy filtering.

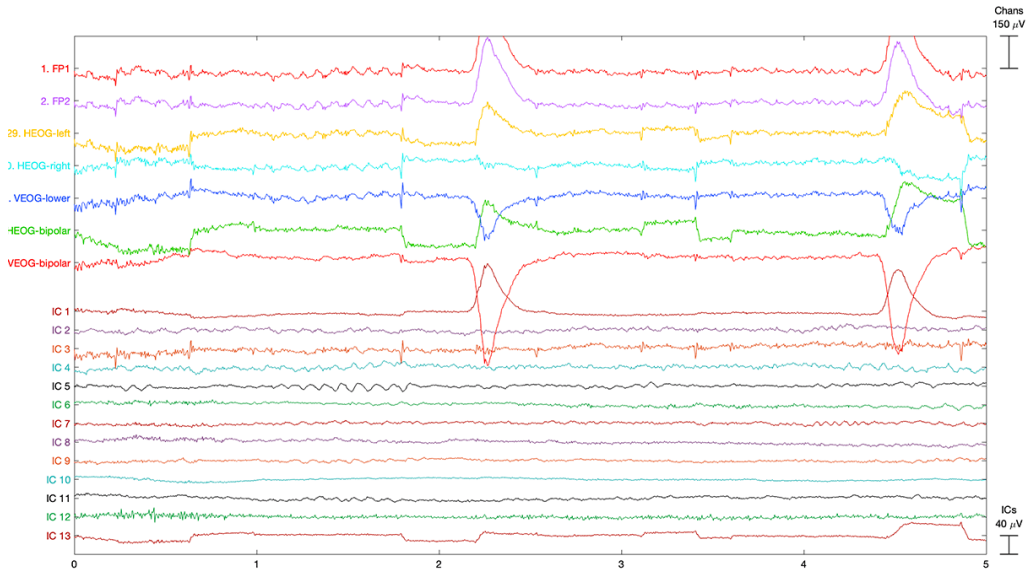
To do this, first look at the **EEGsets** panel and note the EEGset number for the original EEGset (**10_MMN_preprocessed**) and the EEGset containing the ICA weights (**10_MMN_preprocessed_filt_100Hz_del_ICAweights**). The former is probably #1, and the latter is probably #5 or #6, but they might differ from this if you've done some other processing. Make sure that **10_MMN_preprocessed_filt_100Hz_del_ICAweights** is the one EEGset selected in the EEGsets panel. Now go to the **EEGLAB ICA** panel and look at the section labeled **Transfer ICA weights**. The number corresponding to the EEGset with the ICA weights should be in the box labeled **Selected dataset** (EEGset 6 in Screenshot 9.12). Enter the number corresponding to the original EEGset in the box labeled **Target dataset** (EEGset 1 in Screenshot 9.12). Click Transfer to transfer the weights from **10_MMN_preprocessed_filt_100Hz_del_ICAweights** to **10_MMN_preprocessed**. Name the resulting EEGset **10_MMN_preprocessed_transferredICAweights** and save it to disk.



Screenshot 9.12

The original EEGset with the transferred weights should now be active in the **EEGsets** panel, and you can see that the IC weights were successfully transferred because there are now 29 ICs listed in the **ICs** column in the **Channel and IC Selection** panel. These ICs have the same weight maps that you saw in the previous exercise (Screenshot 9.11).

Now let's compare the time course of the ICs with the time course of the EEG and EOG data. To do this, go to the **Plot Settings** panel, check the box for **Display ICs**, set the **Vertical Scale** to **150** for the channels and **40** for the ICs, and click **Apply**. Then go to the **Channel and IC Selection** panel, select FP1, FP2, and all the EOG channels from the **Channels** column, select ICs 1-13 from the **ICs** column, and click **Apply**. The result should look like Screenshot 9.13.



Screenshot 9.13

You can see that the time course of IC 1 closely matches the time course of the blinks and vertical eye movements in VEOG-bipolar. To the naked eye, there isn't any obvious difference between IC 1 from this decomposition and IC 1 from our original decomposition (Screenshots 9.2-9.4). However, given that the decomposition as a whole improved, IC 1 from our new decomposition is less likely to be contaminated by activity from other sources (less lumping).

Now let's see if we can figure out what's going on with the other ICs and see if any of them should be removed along with IC 1. Let's start by looking for an IC related to horizontal eye movements. IC 13 seems like an excellent candidate: it matches the time course of the HEOG-bipolar channel quite well, with sudden changes in voltage at approximately 0.75 seconds, 1.8 seconds, 2.3 seconds, etc. If you scroll to the next time period (5-10 seconds), you'll see that IC 13 continues to match the jumps in voltage in the HEOG-bipolar channel. Note that we're looking at the break period before the stimuli have started. We deleted the break periods in the dataset used for the ICA decomposition, but we've now transferred the weights to the original dataset, which still has data during the break periods. Those weights work just fine for the time points during the break periods, even though they weren't used for the ICA decomposition process.

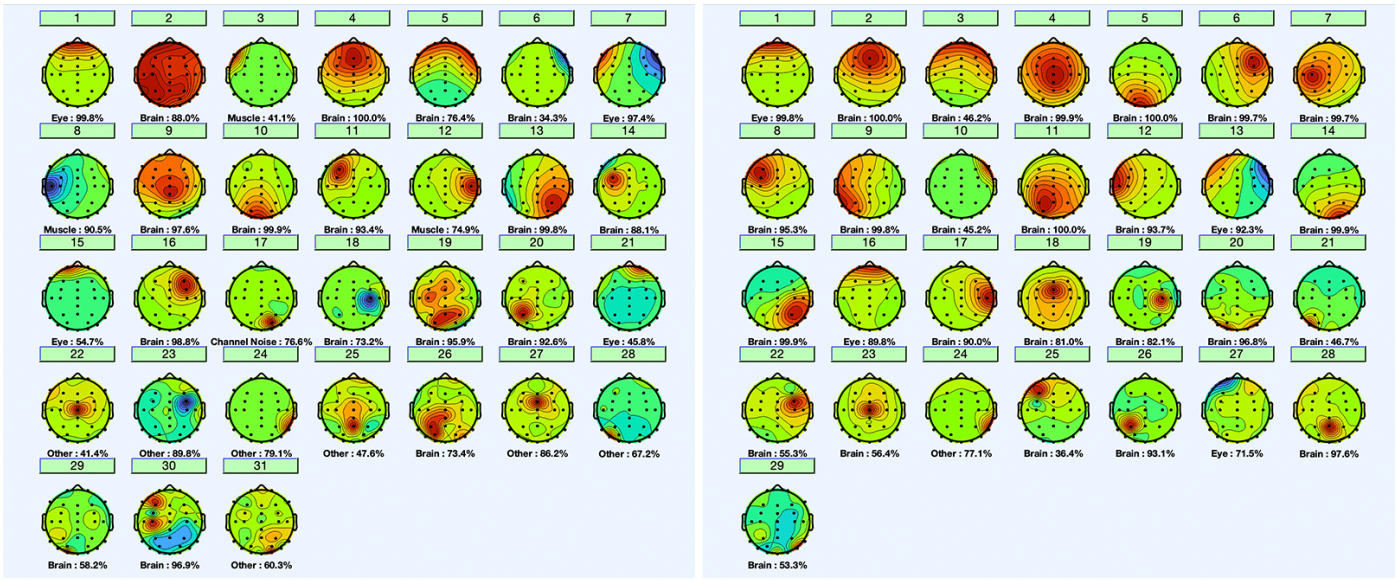
If you look at the scalp map for IC 13 (Screenshot 9.6), you'll see that it has the distribution you'd expect for horizontal eye movements (i.e., opposite-polarity foci just to the sides of the two eyes). This is definitely a horizontal eye movement IC, and we should consider removing it from the EEG.

Although none of the other ICs show the same pattern as the HEOG-bipolar channel, IC 3 has a sharp voltage spike at the onset time of each shift in eye position. This is the spike potential shown in Figure 9.1, which is the muscle contraction that causes the eyes to move. Given that the spike potential and the change in HEOG voltage are closely linked, you might wonder why they end up in different ICs. The answer is that ICA tries to find ICs that are maximally independent at a given time point. The HEOG signal is sustained, and the spike potential occurs only at the very beginning of the eye movement, so you often have a large HEOG voltage with no spike potential. In addition, the spike potential slightly precedes the HEOG change, so you can have a large spike potential without a simultaneous HEOG voltage. Consequently, they end up as different ICs. (The magnitude of the spike potential in the EEG/EOG recordings varies considerably across participants, so you don't always see a separate IC for it.)

EEGLAB has a tool called ICLabel that is designed to automatically classify ICs on the basis of their scalp maps, their time courses, and their frequency content (Pion-Tonachini et al., 2019). Let's give it a try, starting with the initial ICA decomposition that we performed at the beginning of the chapter. Select **10_MMN_preprocessed_ICA** from the **EEGsets** panel (you may need to load it from the Chapter_9 folder if you don't still have it loaded). Then go to the **EEGLAB ICA** panel and click **Classify IC by ICLabel**. You should see a new window of IC scalp maps that looks like the left side of Screenshot 9.14.

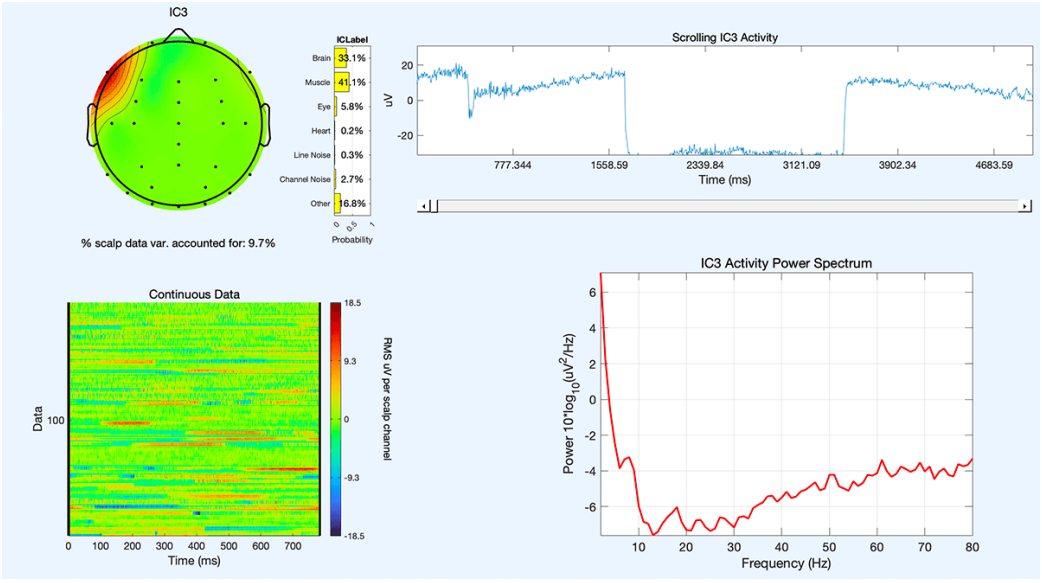
Under each scalp map, you will see the algorithm's guess about the nature of that IC, along with a percentage indicating the confidence of that guess. For example, IC 1 is labeled **Eye** with a confidence of **99.8%**. That seems right given the frontal scalp map and the fact that the time course of this IC closely matched the time course of the blinks (Screenshot 9.4 in Section 9.2). IC 2 is classified as **Brain** with a confidence of **88.0%**. The scalp map doesn't look like a perfect dipole, so it makes sense that the

confidence is not close to 100%.



Screenshot 9.14

IC 3 is classified as **Muscle**, but with a confidence of only **41.1%**. If you click on the box with the **3** in it, you'll see a new window like the one shown in Screenshot 9.15. Muscle activity is often relatively focused on the scalp, which fits the scalp map in this window. In the part of the window labeled **Scrolling IC3 Activity**, you can see the time course of this IC. It shows a pattern that looks a little bit like a saccade, but if you scroll along in that part of the window, you won't see a lot of these boxcar-shaped patterns. The time course also shows quite a bit of high-frequency noise, which you can also see in the power spectrum. That's consistent with muscle activity. But the overall pattern is pretty ambiguous. This just isn't a very "good" IC in terms of matching the kinds of patterns we should see in EEG recordings. And yet this IC is #3 in terms of variance accounted for. This is the kind of thing that you don't want to see if the ICA decomposition is working well. If ICA is working well, then this automatic classification algorithm should be able to label most of the ICs with high confidence (especially for the top half of the ICs). But with this decomposition, we see that IC 5 has a confidence of 76.4%, IC 6 has a confidence of 34.3%, and IC 15 has a confidence of 54.7%.



Screenshot 9.15

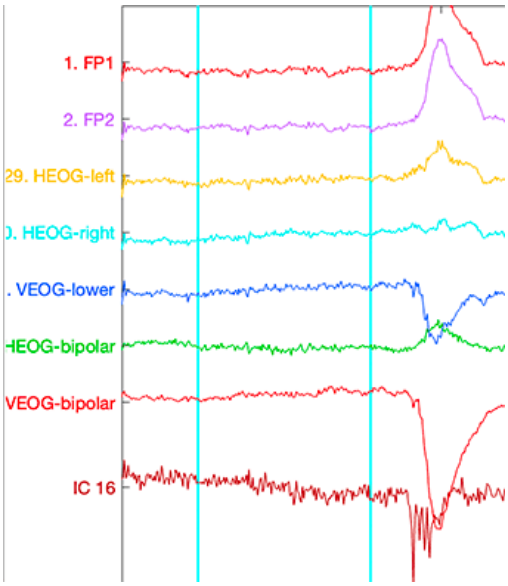
Let's see if things look any better with our second attempt at ICA decomposition, where we eliminated the kinds of C.R.A.P. that

interfere with the decomposition. Close the window showing IC 3, select **10_MMN_preprocessed_transferredICAweights** in the EEGsets panel, and click **Classify IC by ICLabel** in the **EEGLAB ICA** panel. You should see a new window of IC scalp maps that looks like the right side of Screenshot 9.14. Your maps might be a little different, and some polarities might be reversed, but they should otherwise look quite a bit like those in the screenshot (especially the top half of the ICs).

The first thing to notice is that the ICLabel algorithm was able to label all but 2 of the top 15 ICs with a confidence of >90%. IC 1 is still labeled as **Eye** with a confidence of **99.8%**, and IC 2 has a much more brain-like scalp map and is classified as **Brain** with a confidence of **100.0%** instead of a confidence of **88.0%**. IC 3 is still not labeled with good confidence, but ICs 4-9 are labeled as **Brain** with a confidence of >95%. This is definitely an improvement over the original decomposition.

IC 13 is classified as **Eye** with a confidence of **92.3%**. It has the classic horizontal eye movement scalp map, with opposite-polarity foci just lateral to the left and right eyes. And we already saw that the time course of this IC matched the time course of the eye movements in the HEOG-bipolar channel (Screenshot 9.13). We will definitely want to remove this IC.

IC 16 has a scalp distribution that is much like IC 1, and it is classified as **Eye** with a confidence of **89.8%**. Blinks often appear in two or even three ICs, so you should take a close look at any IC that has a blink-like scalp map. If you scroll through the component activations and the EEG/EOG data in the main eStudio window, you'll see that IC 16 exhibits a high-frequency burst at the beginning of each blink, often superimposed on a negative dip. Screenshot 9.16 shows this for a blink that occurred at approximately 28.8 seconds. The burst could be the EMG activity from the muscles that produce the blink. However, the slower negative dip doesn't seem like EMG, so IC 16 may be mixing EMG and EOG signals (and maybe some EEG as well). Little EMG bursts like those in IC 16 don't really have much impact when we score the MMN as the mean voltage from 125-225 ms, so there doesn't seem to be a good reason to remove this IC. Also, because it's the 16th IC, it's not accounting for very much variance, which provides further reason not to remove it given that it is not clearly isolating a single problematic ocular artifact.



Screenshot 9.16

Blink-related EOG activity is primarily a result of the eyelids sliding across the eyeballs, but there may also be a slight vertical rotation of the eyeballs that produces a voltage. These two effects have similar but slightly different scalp distributions, so they may appear as separate ICs (especially when you have 60 or more channels and therefore 60 or more ICs). In theory, the two eyelids could start and stop moving at slightly different times, which might also produce separate ICs. We don't see any clear evidence of this in the present EEGset.

Now let's take a look at the other ICs. IC 5 has a relatively focused scalp distribution over the occipital pole. If you click on the **5** above the scalp map for this IC, the window that appears will show you that the frequency content has a clear peak at 10 Hz. You can also see bursts of 10 Hz activity if you scroll through the time course in the part of the window labeled **Scrolling IC5 Activity**. All of these properties are consistent with the classic alpha-band oscillation first described by Hans Berger in his original EEG study (Berger, 1929). This is an example where ICA is isolating a biologically meaningful component of the EEG. Unfortunately, that does not happen all that often in typical experiments, and there is nothing about the math behind ICA that guarantees that ICs will be true components.

If you close the window for IC 5 and open the corresponding window for IC 2, you'll see that it has a clear peak at approximately 7 Hz. Given this power spectrum and the fact that the scalp map peaks over midline frontal cortex, this is probably the commonly observed *midfrontal theta* oscillation. You can see bursts of oscillatory activity in this IC at a variety of time points (e.g., 127-128 s, 219 s, 297-302 s, 401-403 s), and you can see corresponding bursts of oscillations in the far frontal channels. However, the oscillation bursts sometimes occur simultaneously in other ICs (e.g., at 127 s), and IC 2 also shows some transient (non-oscillating) activity, so it's not clear that this IC does a very good job of isolating the midfrontal theta signal.

Check out the rest of the ICs and see if you can figure out what they represent. You'll see that some contain alpha-band activity, like IC 5, but with somewhat different scalp topographies. This isn't surprising given that alpha can be observed over different cortical areas at different times. Many ICs have no obvious interpretation.

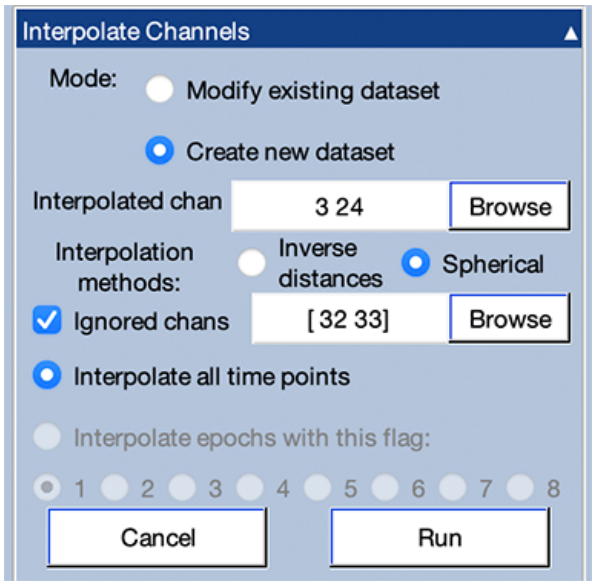
Here's an important question: Should you rely on ICLabel to classify ICs and determine which ones come from the eyes and should therefore be removed in the artifact correction process? The experts I know like to use ICLabel to provide an initial classification, but then they visually confirm the classifications before using them to determine which ICs to remove during correction. When my lab has used ICLabel, we have ended up doing the same thing. I would say that ICLabel gets it right 80-90% of the time when everything is going well. That's pretty good, but not as good as combining ICLabel with your own judgment.

Perhaps the best thing about ICLabel is that it can help you learn what different kinds of ICs should look like, so that you can develop the expertise needed to combine your judgments with the ICLabel classifications.

Now let's correct for the two clear ocular artifact ICs, 1 and 13. Make sure that **10_MMN_preprocessed_transferredICAweights** is selected in the **EEGsets** panel, and then click **Remove ICs** in the **EEGLAB ICA** panel. In the small window that appears, enter **1 13** in the text box labeled **List of component(s) to remove from data**, click the **Ok** button, and name the new EEGset **10_MMN_preprocessed_transferredICAweights_pruned**.

If you look at the resulting EEG, you'll see that the blinks and horizontal eye movements have been removed from all channels except the VEOG-bipolar and HEOG-bipolar channels. In the frontal EEG channels, you'll still see the little spike potentials at the beginning of each horizontal eye movement, and brief EMG bursts at the beginning of each blink, but these are not a problem when we are scoring the MMN as the mean voltage from 125-225 ms.

Now that we've corrected the EEG, we can interpolate the two EEG channels that we left out of the ICA decomposition process in the previous exercise, namely F7 and PO4. To do this, go to the **Interpolate Channels** panel and set it up as shown in Screenshot 9.17. You can either put the interpolated data in the existing EEGset or create a new one. I like to create a new one in case I ever want to go back to the un-interpolated data, so make sure **Create new dataset** is selected. We're interpolating channels 3 and 24 (F7 and PO4). We're going to ignore the bipolar EOG channels because they do not use the same reference as the other channels. Note that it's perfectly fine to use the original EOG channels that have the same reference as the EEG channels, which have perfectly valid signals, so we're not excluding those channels. Click **Run** and name the new EEGset **10_MMN_preprocessed_transferredICAweights_pruned_interp**.



Screenshot 9.17

You are now done with all the steps of artifact correction and interpolation for this participant, and you have beautiful data for the rest of the processing steps. It probably took you quite a while to go through all these steps, but once you've done it a few times, it will be pretty quick (except, perhaps, for the ICA decomposition step). And it's time well spent!

This page titled 9.5: Exercise- Transferring the Weights, Identifying the ICs, Correcting the EEG, and Interpolating is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 9.6: Exercise- Transferring the Weights and Assessing the ICs by Steven J Luck is licensed CC BY 4.0.

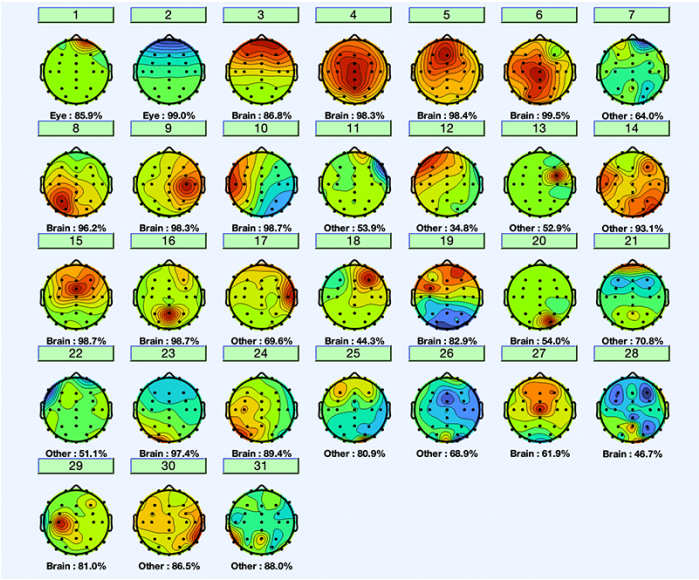
9.6: Exercise- Deleting C.R.A.P. Prior to ICA

In this exercise, we're going to look at another type of large, idiosyncratic voltage deflection that should be eliminated prior to the ICA decomposition, namely huge C.R.A.P. In particular, we want to get rid of voltage deflections that are sufficiently large to impact the quality of the ICA decomposition but are sufficiently infrequent or irregular in their scalp distribution that they can't be well captured by a single IC.

The participant we've been working with so far, Subject 10, doesn't really have any huge C.R.A.P., so we're going to look at Subject 6 for this exercise. Reset everything with the **Reset** button, and then load the **6_MMN_preprocessed_filt_100Hz_del** EEGset. This EEGset has already been preprocessed to improve the ICA decomposition, including filtering from 1-30 Hz, resampling at 100 Hz, and deletion of the break periods.

As usual, you should start by scrolling through the data to see what kinds of artifacts are present. In addition to the usual blinks and eye movements, you'll see some odd-looking voltage deflections at several time points, including 70, 72, 96, 254, 403, and 413 seconds. These deflections are somewhat like blinks, but much larger and with a different and variable scalp distribution. For example, the deflection at ~70 seconds is present in Fp2 but not Fp1, and the deflection at ~72 seconds is huge in VEOG-lower and smaller at Fp1 and Fp2, with no polarity inversion. I have no idea what caused these deflections, but they're large, rare, and have an inconsistent scalp distribution, so they're likely to mess up the ICA decomposition.

To see this, let's take a look at the ICA decomposition, which I did just as in the previous exercise (filtering from 1-30 Hz to minimize skin potentials and other slow drifts). Load the dataset I created with the decomposition (**6_MMN_preprocessed_filt_100Hz_del_ICAweights**) and then look at the ICs by clicking the **Classify IC by ICLabel** button in the **EEGLAB ICA** panel (and accepting the suggested parameters in the two small windows that appear). The IC maps are shown in Screenshot 9.18. (You can do the decomposition yourself if you'd like, but your maps might look a little different.)



Screenshot 9.18

The first thing you should notice is that many of the maps are irregular, even in the top half of the ICs (e.g., ICs 6, 7, and 14). That's a strong hint that the decomposition did not work well. The next thing you should note is that IC 1 (which must account for a lot of variance, because it's the first IC) is classified as **Eye** but only with a confidence of **85.9%**. Its scalp map is somewhat like that of a typical blink, but it's larger over the right eye than over the left. Click on the **1** above the map for IC 1 to see its properties. The time course shows that it is strongly active during a few brief time periods and not at other times. Again, that's unusual.

Now close the window for IC 1 and click on the **2** above the map for IC 2 to open a window showing its properties. This IC is classified as **Eye** with a confidence of **99.0%**. It has the kind of scalp map we'd expect for blinks, and the time course contains the kind of distributed bursts that we'd expect for blinks. Note that the weights in the IC 2 scalp map are negative, whereas we

saw positive weights at the frontal sites for blink-related ICs in our previous exercises. This is because the polarity of the IC maps is arbitrary, and a given IC may have either positive weights or negative weights. Just for fun, I repeated the decomposition a second time, and I found that IC 2 had positive weights in this repetition of the decomposition. The polarity of the weights in the maps is completely arbitrary, and the polarity of the activation values will also be reversed to come up with the right voltage polarity when we reconstruct the data. So, reversals of map polarity are not a problem.

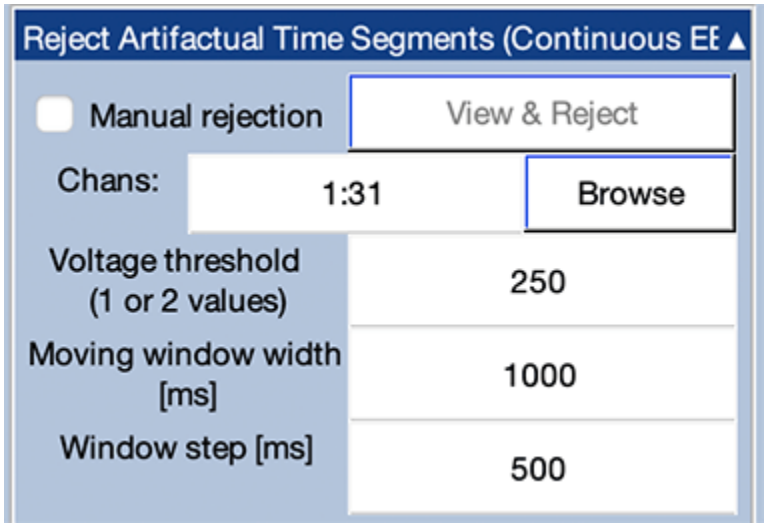
To see this, select EEG channels 1, 2, and 29-33 and ICs 1-12 in the **Channel and IC Selection** panel, and turn on the plotting of ICs in the **Plot Settings** panel. If you scroll through the EEG voltages and IC activations, you'll see that IC 2 has negative deflections at the times of blinks, whereas we saw positive deflections for the blink IC in our previous examples (e.g., Screenshot 9.4). If we multiply the negative activation values by the negative weights for the frontal sites, we'll get a positive voltage at the frontal sites. So, it doesn't matter if we have positive weights along with positive activations or negative weights along with negative activations. In either case, blinks will be reconstructed as a positive voltage at the frontal sites (but a negative voltage at VEOG-lower). So, IC 2 seems like it's doing a decent job of capturing blinks.

If you skip ahead to 70-75 seconds, you'll see two of the crazy artifacts that you saw earlier, one at about 70.5 seconds and the other at about 72 seconds. IC 2 exhibits a deflection for each of the artifacts. IC 1 also exhibits a large deflection for the first one but not for the second one. IC 3 exhibits a large deflection for the second one but not for the first one. So, you can see that IC 1 and IC 3 are very much influenced by these strange, idiosyncratic artifacts. That's a waste of ICs, and those artifacts violate ICA's assumption that all of the sources in the data have consistent scalp distributions over time. This can mess up everything.

Another thing you should notice in the IC maps is that there isn't an IC with the scalp topography we'd expect for horizontal eye movements. There were many clear horizontal eye movements in the HEOG-bipolar channel when we scrolled through the data, but we haven't captured these eye movements as a distinct IC in this decomposition. That also indicates that it didn't work well. (Some participants don't have many horizontal eye movements, and the lack of an IC for horizontal eye movements would not be a problem for those participants.)

Finally, you can see that the ICLabel algorithm had low confidence for many of the ICs. That can also indicate a poor decomposition.

Now let's try to improve the decomposition by deleting the time periods with the huge C.R.A.P. artifacts. Make **6_MMN_preprocessed_filt_100Hz_del** the active dataset and go to the **Reject Artifactual Time Segments** panel. This routine is like the moving window peak-to-peak amplitude routine for artifact detection that we used in the previous chapter, but with two differences. First, it operates on continuous EEG rather than epoched EEG. Second, it deletes sections of the data with artifacts rather than simply marking them.

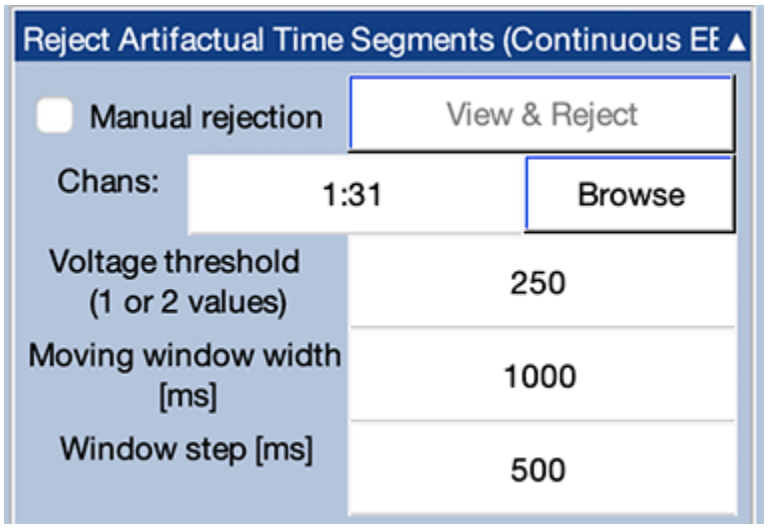


Screenshot 9.19

Set up the top half of the panel as shown in Screenshot 9.19. We're using a threshold of **250**; we don't want to reject blinks or other ordinary artifacts, so we need a higher threshold for this routine than we would use for normal artifact detection. 250 is a good starting threshold, but you may need to adjust it for some participants, especially those with extra-large blinks. Also, if you apply this routine to data that haven't been heavily filtered, you'll often need a higher threshold to avoid deleting a large portion of

the data. We're using a moving window width of **1000** and a step size of **500;** these parameters will cause the algorithm to look for 1000-ms time periods in which the peak-to-peak voltage exceeds the threshold, shifting the window in 500-ms increments. We don't want to flag trials with ordinary blinks or eye movements, so we'll exclude the EOG channels and tell it to use channels **1:31**. (You might also want to exclude Fp1 and Fp2 if you have such large blinks in these channels that blinks end up exceeding the threshold for rejection.) Leave all the check boxes unchecked, and click the **Preview** button. I also recommend checking the box labeled **Join artifactual segments separated by less than _____ ms** (and putting 500 in the text box). This will avoid having brief periods of unrejected data between two nearby rejected periods.

In the window that appears, scroll through the data using the **>>** button until you are able to see the crazy artifact at 70.5, as shown in Screenshot 9.20. You can see that the time period around this artifact is marked in gray. This indicates that this section will be deleted. The section just after 72 seconds also looks a little weird, but only in the EOG channels, so it's good that this section isn't marked for deletion. If you keep scrolling, you will see that the time periods around the crazy artifacts at 96, 254, 403, and 413 seconds have also been marked for deletion.



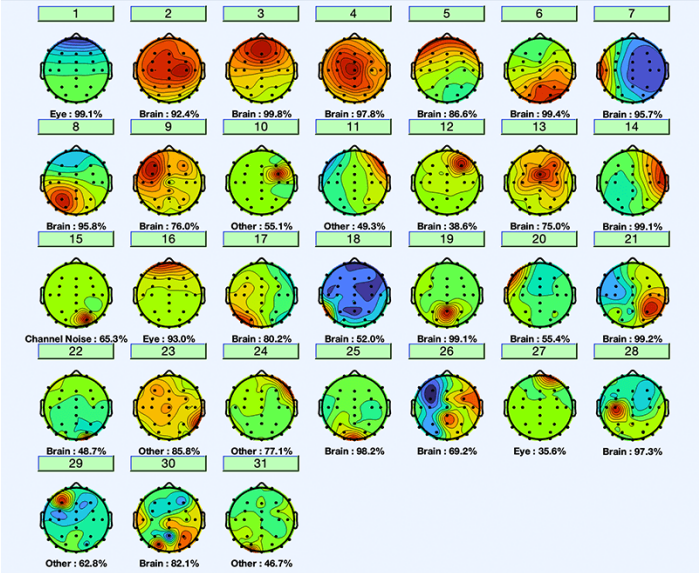
Screenshot 9.20

To actually delete the marked segments, close the preview window, go back to the **Reject Artifactual Time Segments** panel, click the **Finalize** button, and name the resulting EEGset **6_MMN_preprocessed_filt_100Hz_del_car** (where **_car** stands for "continuous artifact rejection"). If you look at the 5-second period starting at 68 seconds in the main eStudio window, you'll see that the artifactual activity at ~70 seconds is gone and has been replaced with a **boundary** event, which has been inserted to mark the discontinuity in the data that was produced by deleting the segment from ~70-73 seconds.

When the EEG is recorded from highly cooperative participants using a high-quality EEG system (as was true in the ERP CORE experiments), many or most participants won't actually have any C.R.A.P. that exceeds the threshold for rejection in the continuous EEG data (especially when the data have been filtered from 1-30 Hz). But it's still worth running this routine, because you may have a few participants who will benefit. And if you're recording from less cooperative participants (e.g., kids) or using a noisier EEG system (e.g., dry electrodes), this step may help a lot.

Note that you can also manually reject sections with huge C.R.A.P. using the **Manual rejection** button in the **Reject Artifactual Time Segments** panel. It's usually faster and more easily scriptable to do the rejection automatically, but sometimes it's useful to do it manually instead (or in addition). Just remember that your goal is to delete segments that will cause problems for ICA, not to delete segments with ordinary artifacts. That is, you want to delete segments with voltage deflections that are large, strange, and infrequent, especially if they have an inconsistent scalp distribution.

Now let's see how deleting the huge C.R.A.P. has impacted the ICA decomposition. You can see the resulting decomposition by loading the EEGset named **6_MMN_preprocessed_filt_100Hz_del_car_ICAweights** and then clicking the **Classify IC by ICLabel** button in the **EEGLAB ICA** panel (and accepting the suggested parameters in the two small windows that appear). The IC maps are shown in Screenshot 9.21.



Screenshot 9.21

The first thing to notice is that the scalp maps are more regular than those we obtained without deleting the huge C.R.A.P. (see Screenshot 9.18). There are still some irregular maps, but mainly in the bottom half, which don't account for much variance (e.g., ICs 23 and 26). The second thing to notice is that IC 1 is now classified as **Eye** with **99.1%** confidence and has the usual scalp map for a blink component. The third thing to notice is that we now have an IC with the usual scalp map for horizontal eye movements (IC 11). ICLabel's confidence is only 49.3%, but if you look at the time courses of the ICs and the EOG channels, you'll see that IC 13 matches the eye movements in the HEOG-bipolar channel quite well. So, by deleting segments with huge C.R.A.P., we've obtained a much better ICA decomposition, even though we only deleted ~10 seconds' worth of data.

Note that IC 11 is labeled **Other**, and it's narrowly focused over the FC4 electrode. I took a close look at this electrode in the original EEGset, and I saw a lot of odd behavior. I would consider interpolating this electrode (and therefore excluding it from the ICA decomposition).

The next steps are to transfer the ICA weights to the original EEGset (which you can find in a file named **6_MMN_preprocessed**) and then remove ICs 1 and 11. You can do that or load **6_MMN_preprocessed_ICAweights_pruned**.

Now scroll through the resulting dataset. You'll see that this EEGset still has data during the break periods at the beginning and end. If you scroll to 78 seconds, you'll see that the huge C.R.A.P. artifacts have not been deleted from this EEGset, but they are now at about 80 and 81.5 seconds. We will take care of these artifacts by using the artifact detection methods described in the preceding chapter to mark them (after epoching), and we'll exclude the marked epochs during the averaging process.

You should also notice that the artifact correction process has done a pretty good job of eliminating the voltages produced by blinks and eye movements. It hasn't taken care of all the other artifacts in this very noisy EEGset. But ICA is not designed to correct all artifacts, only those like blinks and eye movements that are large, frequent, and have consistent scalp distributions over time.

This page titled 9.6: Exercise- Deleting C.R.A.P. Prior to ICA is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 9.8: Exercise- Deleting C.R.A.P. Prior to ICA by Steven J Luck is licensed CC BY 4.0.

9.7: General Recommendations for ICA-Based Artifact Correction

In this section, I summarize my recommendations for ICA-based artifact correction. These recommendations reflect my assessment of the current state of the art. It is difficult to determine the optimal approach to ICA analytically or with simulations, so these recommendations are mainly based on a combination of informal experience (my own and others') and empirical studies (which are often unsatisfying because ground truth is not usually known). As a result, these recommendations may change as more information accumulates, and other researchers may reasonably disagree with some of them.

Also, these recommendations are based largely on studies of compliant adult participants, and some changes may be necessary for other populations. So, use them as a starting point for designing your own artifact correction approach, not as a recipe to be followed blindly. And make sure that your approach is designed to achieve the fundamental goal of obtaining an accurate answer to the scientific question your study is designed to answer (which typically involves reducing noise and avoiding confounds from artifactual voltages and from changes in the sensory input).

If you are (or become) an ICA expert, there are more sophisticated approaches that you may want to take that might improve the correction a bit or make it work well under unusual situations. My recommendations are for the average researcher who wants to use an approach that almost always works very well.

Now that I'm done with the caveats, here are my recommendations:

- If you will be correcting for an artifact that is relatively small or relatively rare, you may want to design your experiment so that you have some special trial blocks in which that artifact is larger or more frequent. For example, imagine that you want to correct for eye movements in a task where participants are instructed not to move their eyes, but they occasionally make unintentional eye movements. There may be enough eye movements during the task to create confounds, but there may not be enough to produce an IC that represents the eye movements accurately. To solve this, you can run a couple blocks in which the participants are asked to make eye movements to an object that appears in various locations. You will then include those blocks along with your main task when you perform the ICA decomposition. This will produce a much more accurate decomposition for the eye movement IC.
- As described in Chapter 7, the first step in dealing with artifacts via rejection or correction is to scroll through the entire dataset to make sure you know what kinds of artifacts and other problems are present. Determine which channels should be interpolated, but don't perform the interpolation yet.
- The correction should ordinarily be carried out on continuous data. If, for some reason, you need to perform correction on epoched data:
  - The epochs must be at least 3 seconds long (e.g., -1000 to +2000 ms), but the epochs may not overlap with each other. This is usually possible only with paradigms that involve very long intertrial intervals.
  - No baseline correction should be applied (Groppe et al., 2009).
  - The epoching should be performed after the high-pass filtering step described below.
- The data should be referenced, and all of the channels included in the ICA decomposition should share the same reference. Any reasonable reference is fine, but extra steps are required if you use the average of all sites as the reference (see Makoto's Preprocessing Pipeline or EEGLAB's ICA documentation).
  - The simplest thing is to use a single electrode as the reference prior to the ICA decomposition and correction steps. You can then reference to whatever you like after the correction has been applied.
- Create additional bipolar channels for blinks and eye movements prior to correction, but leave them out of the ICA decomposition. These will allow you to see when blinks and eye movements happened in the corrected data, which can be valuable.
- If you have large amounts of line noise, use the *cleanline* plugin to eliminate it prior to the ICA decomposition (see Bigdely-Shamlo et al., 2015 for important details about implementing this tool). This is better than applying a low-pass filter (but a low-pass filter is sufficient for modest amounts of line noise).
- Once you have applied the above steps, make a separate copy of the dataset for use in the ICA decomposition. Apply the following steps to that copy, but not to the original data.
  - High-pass filter at 1 Hz (I recommend 48 dB/octave). If you will not be analyzing high-frequency activity (e.g., gamma-band oscillations), you should also apply a low-pass filter at 30 Hz (48 dB/octave).
  - Downsample the data to 100 Hz. This isn't necessary, and it doesn't change the results, but it makes everything run faster.

    - Delete time periods during breaks.
    - Delete segments of huge C.R.A.P.
    - Run the ICA decomposition with the **runica** algorithm, using the **'extended', 1** option. Leave out the bipolar channels and any channels that you plan to interpolate.
    - Verify that the ICs are reasonable (e.g., not too many irregular scalp maps, especially for the top half of the ICs). If the ICs do not look good, take another look at the data and see if there are problems you missed (e.g., channels that should be interpolated, periods of large C.R.A.P. that need to be deleted more aggressively).
    - When you are first starting out (or switching to a different kind of experimental paradigm or participant population), it's a good idea to repeat the decomposition and make sure that you get similar results each time. If you don't, then the decomposition is not working well. The RELICA plugin can be used to provide a quantitative assessment of the reliability of the decomposition.
    - If you can't get a good decomposition, check to make sure that you have enough data. The informal rule is that the number of time points in the dataset must be at least 20 x (# channels)^2 (assuming that your original data were sampled at ~250 Hz). If you don't have enough data, one option is to apply PCA first to reduce the dimensionality of your data. However, this can create significant problems (Artoni et al., 2018), and you shouldn't use it unless you have more than 128 channels.
- Once you've performed the decomposition on the heavily filtered copy of the data, transfer the ICA weights to the original version of the dataset (the version right before you made the copy).
- Examine the ICs carefully to make sure everything looks OK and to identify the key ICs. This includes looking at the scalp map, the frequency spectrum, and the time course heat map for each key IC, and then scrolling through the IC activations and voltages simultaneously to see what voltage changes co-occur with the key ICs. I also recommend looking at ICLabel's classifications to help you identify ICs that correspond to ocular artifacts. But combine the information from ICLabel with your own judgment.
- Remove the ICs that correspond to clear, well-understood artifacts, that have been well isolated by ICA, and are actually problematic (e.g., reduce your data quality or differ across conditions in the averaged ERPs). In my lab, we remove only ICs related to blinks and eye movements. However, EKG or EMG artifacts can be real problems in some areas of research, in which case you may want to remove ICs corresponding to those artifacts.
- Interpolate the bad channels that you previously identified.
- After performing artifact correction, you should perform artifact detection on the epoched data. At a minimum, you want to eliminate epochs with C.R.A.P., because these are not handled well by artifact correction. In visual experiments, you should also mark and reject epochs with blinks and eye movements that might have interfered with the perception of the stimuli (e.g., between -200 and 400 ms). You can use the bipolar EOG channels for this because they were not corrected. This will be described in some of the following sections of this chapter.
- There are additional steps you can take to assess whether artifact correction is working well, as will be described later in this chapter.

I'd like to say a few words about how interpolation interacts with artifact correction. You don't want the "bad channels" to mess up the ICA decomposition, so these channels need to be excluded from the decomposition stage. You'll then perform the interpolation after the decomposition has been performed and the data have been corrected. That way, the interpolated channels will reflect the corrected data. My lab spent about 20 minutes one day talking through the different possible orders of steps for combining interpolation and correction, and this approach was clearly the best.

This page titled 9.7: General Recommendations for ICA-Based Artifact Correction is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 9.9: General Recommendations by Steven J Luck is licensed CC BY 4.0.

9.8: Background- Evaluating the Effectiveness of Artifact Correction+Rejection

Why You Should Worry About Artifact Correction and Rejection

As described in Section 9.3, ICA-based artifact correction is based on assumptions that are simply not true of real data. Moreover, correction can massively alter your data, because it changes the voltage at every time point in every channel. I like to use the transporter in Star Trek as an analogy: it converts a person into an energy pattern, sends that energy pattern to a new location, and then converts the energy pattern back into a person. ICA-based artifact correction converts an EEGset into a set of IC activation time courses, sets the activation values to zero for some of the ICs, and then reconstructs the EEGset from the remaining ICs. If you would be nervous to beam yourself to a new location using a Star Trek transporter, then you should be nervous to correct artifacts using ICA.

Artifact rejection isn't quite as scary as artifact correction, because you are simply excluding a subset of epochs from the averaging process (and hopefully the number of excluded epochs will be small). But you can think of the excluded epochs as being missing values, and there is an enormous literature on the biases that can result from missing values (see Heise et al., 2022, for a specific example of how artifact rejection can create confounds in ERP experiments with infants and young children). Moreover, reducing the number of trials available for averaging reduces the data quality of the averages (all else being equal). So you should be at least a little nervous about using artifact rejection.

The rest of this chapter will focus on showing you how to quell your nerves by assessing the effectiveness of your artifact correction and rejection procedures. Specifically, I will show you how to implement (a slightly simplified version of) an approach that my lab came up with to assess our standard pipeline for dealing with artifacts, which involves ICA-based correction of ocular artifacts followed by the detection and rejection of epochs containing C.R.A.P. (Zhang et al., 2024). We developed this approach to evaluating the effectiveness of our pipeline for two reasons. The first reason was that we were always a little nervous about using artifact correction and rejection (just as you should be). The second reason was that Arno Delorme published a paper whose title boldly proclaimed that EEG is better left alone (Delorme, 2023).

In this paper, Arno analyzed several EEG studies using a variety of different processing pipelines, including artifact correction, and he found that the only steps that really produced much improvement in statistical power were high-pass filtering and interpolation of bad channels. In fact, many common processing steps actually decreased statistical power. I really love this paper. For one thing, I'm impressed that Arno—who has spent much of his career developing and using sophisticated signal processing methods—was willing to write a paper telling people that these methods aren't actually helpful. I also liked the paper because I have become increasingly convinced that high-pass filtering is the important step for maximizing statistical power in most ERP studies.

However, Arno's paper also had some significant shortcomings. The most important was that he focused solely on statistical power and did not discuss the fact that artifacts can produce confounds as well as reducing statistical power. In addition, he used a fairly coarse approach to determining whether a given processing step increased statistical power, so he may not have had the sensitivity to detect a positive effect of some processing steps.

My lab therefore developed an approach to assessing the benefits of artifact correction and rejection that uses a better means of assessing whether a processing step leads to improved data quality (based on, of course, the standardized measurement error [SME]). In addition, we examined confounds as well as data quality.

Our Pipeline for Artifact Correction and Rejection

Before I describe our approach to evaluating a pipeline, I will spend a few paragraphs describing my lab's general pipeline for combining artifact correction and rejection. We find that this pipeline works well for both our basic science research on attention and working memory (which uses college student participants) and for our translational research on schizophrenia (which uses adults with schizophrenia and community control subjects, all aged between 25 and 55). The only difference in procedures is that we exclude participants in our basic science studies if the percentage of trials marked for rejection is >25%, whereas we use an exclusion threshold of >50% in our translational studies. I think our pipeline is pretty good, and I recommend that you adopt it if

you do similar research. If your research is very different, you can use it as a starting point but make any necessary changes.

Our pipeline starts with artifact correction, using the approach summarized in Section 9.7. We only correct for eye blinks and eye movements, and only for the electrooculographic components of these artifacts (i.e., the voltage changes that are a consequence of the electrical dipole inside each eye, not the EMG artifacts that are sometimes visible at the onset of a blink or eye movement). In our research, we have not found it useful to correct for other kinds of artifacts.

After we have corrected for EOG artifacts, we apply artifact detection. Specifically, we mark all epochs in which large C.R.A.P. was present in any EEG channel. (To sound more professional, we refer to this as marking epochs with “extreme values” in journal articles.) We then exclude the marked epochs at the time of averaging.

You might think that rejecting epochs in which C.R.A.P. is present in any channel—even if that channel is not being used to score the ERP component of interest—would decrease our data quality by causing us to reject a lot of epochs with perfectly fine data in the measurement channel(s). We had the same thought, so we formally compared rejecting epochs with C.R.A.P. in any channel against rejecting epochs only if the C.R.A.P. was in the measurement channel. We found that both approaches yielded similar levels of data quality. This may be because some bad epochs didn’t quite exceed the C.R.A.P. detection threshold in the channel of interest but did exceed the threshold in other channels. It may also be because large C.R.A.P. in any channel indicates that something problematic was going on during that epoch (e.g., the participant was distracted by a yawn). In either case, I prefer to reject epochs with C.R.A.P. in any channel. But focusing only on the measurement channel(s) is also reasonable, and it may be preferable in some types of research. However, if you are using the average of all sites as the reference, artifacts in any channel will impact your data, so you'll probably want to test for artifacts in all channels.

In studies using visual stimuli, our pipeline also includes marking all epochs in which a blink or eye movement occurred when the stimulus was visible on the display, indicating that the blink probably interfered with the perception of the stimulus. In the ERP CORE experiments, we found that very few epochs had blinks or eye movements while the stimulus was present. As a result, rejecting these epochs had very little impact on the averaged ERPs. Because rejecting these epochs doesn't hurt if they are rare, and it seems like they should be rejected if they are common, I recommend rejecting epochs with blinks or eye movements at the time of the stimulus in all studies using visual stimuli.

Now you know the essence of our pipeline. The remainder of this chapter will demonstrate our approach to determining whether this pipeline—or any pipeline for artifact correction and rejection—is actually beneficial. By learning to use this approach and then applying it to your own data, you can determine whether you are actually performing artifact correction well. You can also determine whether your C.R.A.P. thresholds are set to optimal levels for maximizing your data quality, rejecting neither too few trials nor too many. In addition, you can determine whether, for your specific research, artifact rejection works better if you look for C.R.A.P. in all EEG channels or just in the channel(s) used for measurement. More generally, you can be less nervous about your own artifact correction and rejection pipeline once you’ve established that it actually works well for your own data.

Our Approach for Evaluating the Effectiveness of Artifact Correction and Rejection

Our approach for evaluating the effectiveness of artifact correction and rejection focuses on whether blinks are a confound and whether excluding trials with large C.R.A.P. helps or hurts data quality. A very different approach is needed for evaluating whether eye movements are a confound and whether the pipeline has effectively minimized this confound, and this was described in the last several sections of Chapter 8.

Our approach consists of the following major steps:

- Assess whether blinks are actually a confound if not corrected (i.e., whether they differed between experimental conditions)
  - If so, assess whether the artifact correction process eliminated the confound.
    - If the confound was not fully eliminated, assess whether the remaining confound was large enough to have a meaningful impact on the ERP effects.
- Assess whether data quality (SME) was improved by correction of EOG artifacts and whether it was further improved by also rejecting epochs with large C.R.A.P.

This is a bit of a simplification of the full approach, which you can read about in our paper (Zhang et al., 2024). I will add some of the additional details in the exercises in the following sections of this chapter.

This page titled 9.8: Background- Evaluating the Effectiveness of Artifact Correction+Rejection is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck.

9.9: Exercise- Determining Whether Blinking is a Confound

Our first step in assessing the effectiveness of our artifact correction+rejection pipeline is to determine whether blinks are a confound, meaning that they differ across the conditions we will be comparing. If they differ across conditions, then they could create a bogus experimental effect or distort the time course or scalp distribution of a real effect. When blinks are a confound and not just a source of noise, this increases the stakes for having an effective correction+rejection pipeline.

We have mainly focused on the MMN paradigm so far in our chapters on artifact rejection and correction, and given that the stimuli in this paradigm were task-irrelevant, it is unlikely that blinks would differ between deviants and standards. Indeed, Zhang et al. (2024) found that the blinks were random with respect to the stimuli in this paradigm. Consequently, the remainder of this chapter will focus on the N400 paradigm, which has some very interesting blink patterns that will demonstrate what you should look for in your own studies.

As a reminder, each trial in this paradigm consisted of a prime word followed by a target word, and the target word was either semantically related to or unrelated to the prime word. The task was to press one of two buttons following each target to indicate whether it was related or unrelated to the preceding prime. The N400 component was much larger (more negative) for unrelated targets than for related targets. The N400 was measured as the mean amplitude from 300-500 ms in the unrelated-minus-related difference wave at the CPz electrode site. The details are presented in Sections 2.2 and 2.3 of Chapter 2.

General Approach

Our question in this exercise is whether blinks differed between related and unrelated targets. There is no obvious reason to expect a difference, but it is important to test for a difference rather than assuming the lack of a difference. One thing my lab learned when we carefully examined the artifacts in the ERP CORE paradigms (Zhang et al., 2024) is that blinks often differed between conditions in ways that we did not expect.

Note that we will focus on blinks in this exercise and largely ignore eye movements, which were rare in the N400 paradigm. Participants were motivated to keep their gaze on the central fixation point in this paradigm, and we saw zero eye movements during the task in most participants. In addition, the ICA decomposition did not yield a clear eye movement IC for most participants.

We are going to examine differences in blinking between conditions in two ways. First, we will apply artifact detection to the uncorrected VEOG-bipolar channel and count how many trials in each condition were marked for rejection. Second, we will ask whether the time course of the blinks differed between conditions by comparing the waveforms for related targets and unrelated targets in the uncorrected VEOG-bipolar channel in the averaged ERPs.

If you looked at this information in the data from a single participant, you wouldn't know whether any differences were random or systematic. To assess whether there are systematic differences in blinking between conditions, you need to look at the data from all the participants. This would take a very long time in the eStudio GUI, so I wrote some Matlab scripts that prepared the data for you.

I limited the analyses to 20 of the 40 participants to keep things simple (and, if I'm being honest, to save myself some time). I first limited the analyses to the 10 participants that were analyzed in Chapter 3. I then realized that I had originally chosen those 10 participants because they had beautiful data, so they weren't very good for demonstrating artifact correction and detection. I therefore added another 10 participants who had more typical data. It's still not a random sample of the 40 participants who were included in the original ERP CORE study (Kappenman et al., 2021), so the results you will see here will not quite match what was reported by Zhang et al. (2024). However, the general pattern is the same.

The Data

You can find the N400 data in a folder cleverly named **N400_Data** inside the Chapter_9 folder. The initial EEGsets in this folder have already been preprocessed a little, including high-pass filtering the data at 0.1 Hz, referencing the EEG and EOG channels to the average of P9 and P10, and creating bipolar VEOG and HEOG channels. Each of these EEGsets is named **X_N400_preprocessed.set**, where **X** is the subject number. The folder also contains three scripts that implemented ICA-based

artifact correction for these 20 participants following the recommendations in Section 9.7:

- **N400_correction_rejection_phase1.m** took each participant's initial EEGset **(X_N400_preprocessed**) and made a copy of this EEGset (named **X_N400_preprocessed_optimized**) that was optimized for the ICA decomposition process. This involved downsampling to 100 Hz, filtering from 1-30 Hz, deleting break periods, and deleting any segments of the continuous data that contained extra-huge C.R.A.P. that might interfere with the ICA decomposition.
- **N400_correction_rejection_phase2.m** took each participant's optimized EEGset and performed the ICA decomposition (excluding the bipolar EOG channels and any "bad" channels that would later be interpolated). This resulted in an EEGset for each participant named **X_N400_preprocessed_optimized_weights**.
- **N400_correction_rejection_phase3.m** transferred the ICA weights back to the original EEGset and then used those weights to perform the artifact correction by reconstructing the EEG from the ICs, but without the ICs corresponding to blinks and possibly eye movements (if a clear eye movement IC was present). Interpolation was then performed on the "bad" channels. The resulting EEGset was named **X_N400_preprocessed_pruned**.

These scripts made use of Excel files that I created to store information about which channels were bad, what parameters should be used to reject huge C.R.A.P. from each participant's continuous EEG data, and which ICs corresponded to blinks and eye movements and should therefore be removed. The output files of these scripts are available in the **N400_Data** folder, so you do not need to run the scripts. However, you will find them very useful as models for writing scripts to apply this approach to your own data.

Counting the Number of Blinks Per Condition

To count the number of blinks per condition for each participant, I created and ran a script named **N400_correction_rejection_phase4_count_blinks.m**.

> **Statistics Toolbox**
>
> The script **N400_correction_rejection_phase4_count_blinks.m** uses the Matlab **ttest()** function, which is part of the Statistics and Machine Learning Toolbox. You can see what toolboxes are installed by typing **ver** on the Matlab command line. If you do not have this toolbox, I recommend consulting your IT department to see if you can get it for free as a part of your license. You can also purchase it from The Mathworks.
>
> If you cannot get this toolbox, the script has information about how to comment out the lines that use the **ttest()** function.

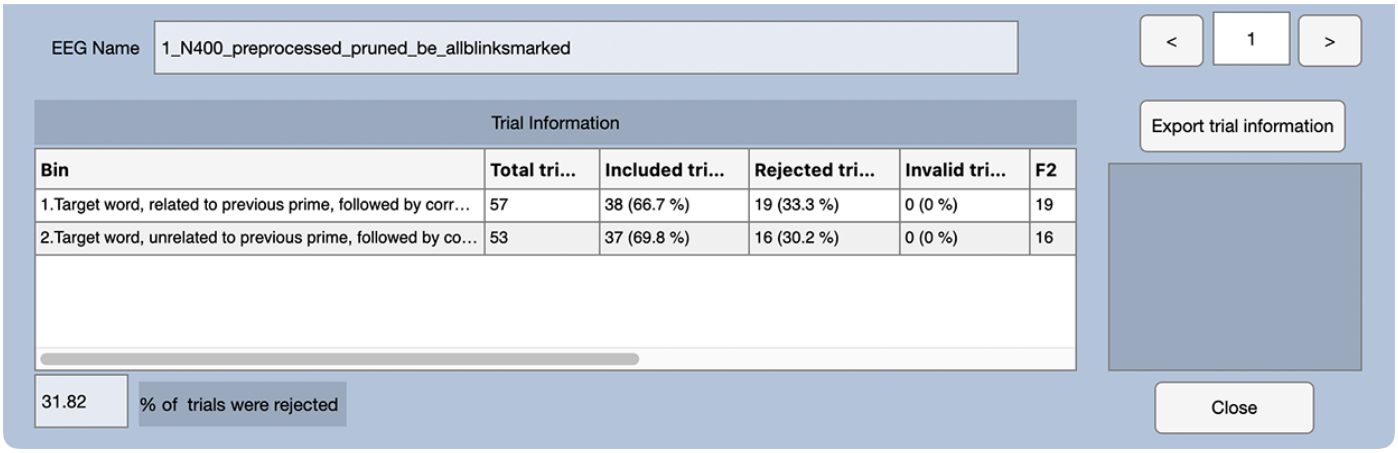
For each of our 20 participants, this script took the ICA-corrected data (**X_N400_preprocessed_pruned**), added an EventList, ran BINLISTER using a bin descriptor file named **BDF_N400_Targets.txt** to sort the events into bins (bin 1 = related targets; bin 2 = unrelated targets), and epoched the data. Then, artifact detection was applied to the uncorrected VEOG-bipolar channel to mark epochs containing blinks. I used the moving window peak-to-peak amplitude algorithm with a threshold of 100 μV, looking for blinks in every 200-ms window within the 1000-ms epoch. I checked to make sure that this threshold worked well for each of the 20 participants. The resulting EEGset was saved in a file named **X_N400_preprocessed_pruned_be_allblinksmarked.set**. Keep in mind that these epochs were marked solely to determine if the number of blinks differed between conditions; these blinks have already been corrected in the EEG channels via ICA, but we want to know if they are a confound and whether ICA eliminated this confound.

Using these EEGsets, the script created an averaged ERP for each participant. I included this step because it's really easy to get a count of the number of artifacts in each bin from an ERPset. This information could be derived from the EventList in the EEGsets, but it's a lot easier to see it in an ERPset.

To see the results of this set of steps, first reset everything with the **Reset** button. Then change the current folder to **N400_Data** using the **Current Folder** button in the **EEGsets** panel, and then load the EEGset for Subject 1 with the blinks marked (**1_N400_preprocessed_pruned_be_allblinksmarked.set**). Start scrolling through the EEG, looking for epochs that have been marked for rejection because of blinks in the VEOG-bipolar channel (which was excluded from the ICA decomposition and the subsequent artifact correction). You'll see no blinks in the first 30 epochs. Then you'll see that blinks started occurring more often. This is a common pattern; many participants initially suppress their blinking because they are focusing on the task, but then their eyes start to dry out, and they start blinking.

Because we planned to apply artifact correction to these data, the only instruction we gave participants was to avoid blinking prior to making their buttonpress response on each trial. As a result, there were lots of blinks, mainly after the response at ~400 ms. You can see that most of Subject 1's blinks were in the second half of the epoch.

Now go to the **Artifact Info & Tools** panel and click **Artifact Summary**. A window should appear with the table shown in Screenshot 9.22. You can see that 31.82% of epochs were marked as containing blinks, with 19 marked epochs for bin 1 (related targets) and 16 marked epochs for bin 2 (unrelated targets).



Screenshot 9.22

Although it's easy to view the number of marked epochs per bin in an EEGset using the eStudio GUI, this information is a little more difficult to extract using a script. The script therefore used this EEGset to create averaged ERPs, with the ERPset named **1_N400_count_blinks.erp**. Go to the **ERP** tab and load this ERPset using the **ERPsets** panel.

If you go to the **ERP & Bin Information** panel, you can again see that 19 epochs were marked for rejection in bin 1 and 16 epochs were marked in bin 2. You can also get this information by going to the Command Window and typing **ERP.ntrials**. You should see something like Screenshot 9.23. This shows the number of accepted and rejected trials for each bin. This information is directly accessible by a script.

```
>> ERP.ntrials
ans =
  struct with fields:
    accepted: [38 37]
    rejected: [19 16]
     invalid: [0 0]
     arflags: [2×8 double]
```

Screenshot 9.23

If you do a little arithmetic with these numbers, you can determine that 33.3% of trials were marked for blinks in bin 1 (related targets) and 30.2% of trials were marked in bin 2 (unrelated targets). Is that small difference just noise, or is it a consistent difference between related and unrelated targets? To answer that question, we will need to look at all of the participants.

To avoid doing a lot of arithmetic by hand, I had the script calculate these proportions for each participant, store them in an Excel spreadsheet named **Blinkrate.xslx**, and display the mean across participants for each bin along with a statistical test of the difference between the two bins. These values are shown in the Command Window, like this:

```
Blink rate for related targets: 0.439
Blink rate for unrelated targets: 0.351
t(19) = 2.93, p = 0.0086
```

You can see that, across the 20 participants included in this analysis, people blinked about 9% more following related targets than following unrelated targets, and that this was a statistically significant difference. I have no idea why people blinked more in response to the related targets, and I don't really care, because I am not a psycholinguist. However, I do care that there was a consistent difference in blinking between related and unrelated targets, because this is a confound. If we do not deal with the voltages produced by these blinks, they will create an artifactual difference between the ERPs for related and unrelated targets.

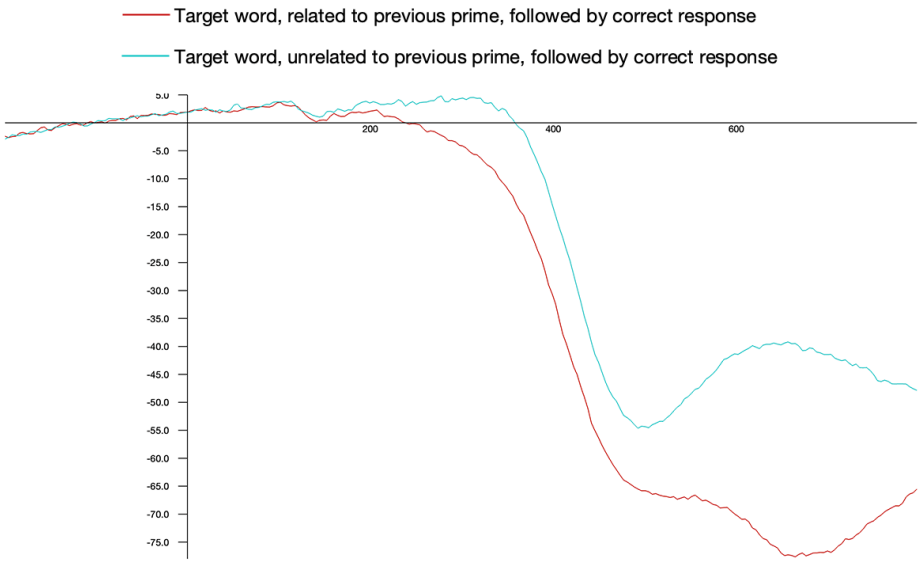
Note that you can see the spreadsheet with the single-participant values (**Blinkrate.xslx**) by going back to the Chapter_9 folder. You can run the script by going to the Chapter_9 folder and typing **N400_correction_rejection_phase4_count_blinks** in the Command Window.

Comparing the Blink Waveforms for the Two Conditions

In addition to counting the number of blinks in each condition, it's also important to examine the uncorrected VEOG-bipolar waveforms for the two conditions. Even if the number of blinks is the same across conditions, the timing might be different, and that could lead to differences in blink voltages between the two conditions at some time points. Indeed, when my lab looked at the error-related negativity data from the ERP CORE, we found that the number of blinks was nearly identical for error trials and correct trials, but the blinks were later for the error trials than for the correct trials (Zhang et al., 2024).

To avoid being fooled by idiosyncratic single-participant results, it's best to compare the blink waveforms using grand average ERP waveforms. To see this for the N400 experiment, make sure that the Chapter_9 folder is Matlab's current folder, click the **Load ERP** button in the **ERPsets** panel, and select **Grand_N400_no_correction_or_rejection.erp.** This grand average was created with a script named **N400_correction_rejection_phase4_baseline.m**, which I will describe in more detail in the next section. No artifact correction or rejection was applied in the pipeline that produced this ERPset.

Select only channel **33** (VEOG-bipolar) in the **Bin & Channel Selection** panel and click **Apply**; this will show you only the uncorrected bipolar VEOG signal. It should look something like Screenshot 9.24. You can see that the blink waveform is substantially larger (more negative) for the related targets than for the unrelated targets. The blink voltage also appears to start earlier for the related targets, and there is a substantial difference in voltage between related and unrelated targets in the N400 measurement window (300-500 ms). Uh oh! We need to make sure that our artifact correction procedure actually eliminated this confound.



Screenshot 9.24

Now go back to the **Bin & Channel Selection** panel, select **All** channels, and click **Apply**. If you look at the VEOG-lower channel

(which is the electrode under the left eye referenced to the average of P9 and P10), you will see that the voltage is more negative for unrelated targets than for related targets, just as we saw in the VEOG-bipolar channel. Blinks are negative under the eye, so this greater negative voltage for related targets is consistent with our earlier data showing that blinks were more common for related targets than for unrelated targets.

If you look at FP1 and FP2, you'll see that the voltage is more positive for related targets than for unrelated targets. Blinks are positive at FP1 and FP2, so this is also likely a result of the greater number of blinks for related targets. This difference between related and unrelated targets is smaller at the F3, Fz, and F4 electrodes, which makes sense given that the amplitude of a blink falls off as we move away from the eyes.

If you look at CPz, you'll see that the voltage is also more positive for related targets than for unrelated targets. But we could rephrase this as the voltage being more negative for unrelated targets than for related targets, which is the usual way of describing the usual N400 effect. So, is the effect at CPz the N400 effect, or is it just the difference in voltage we see as a resulting of blinking at FP1 and FP2, volume-conducted from the eyes back to CPz?

Yikes! These findings raise the possibility that the venerable N400 effect, which has been observed in hundreds of published studies, might be a blink artifact rather than reflecting differences in brain activity for related and unrelated targets. But keep in mind that we haven't done any correction or rejection for blinks, and almost all N400 studies have at least attempted to minimize blink activity. So, the N400 effect in this experiment is probably safe. But to be sure, we need to determine whether our artifact correction+rejection procedure has done a good job of reducing the blink-related activity. This shows the importance of evaluating the effectiveness of your pipeline for dealing with artifacts. If it's not effective, your effects may reflect blink artifacts instead of brain activity (or a mixture of artifacts and brain activity).

This page titled 9.9: Exercise- Determining Whether Blinking is a Confound is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck.

9.10: Exercise- Was Blink Correction Effective?

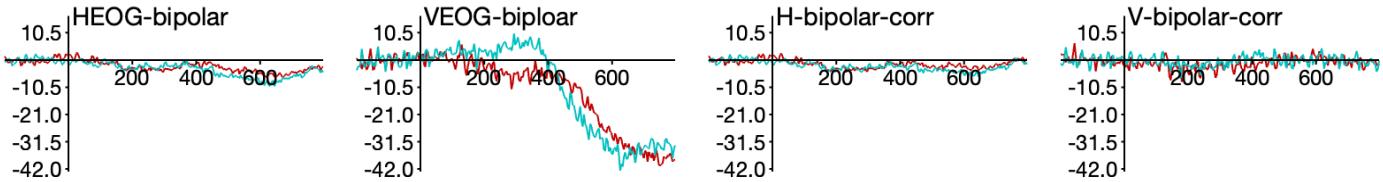
Now that we've seen that blinks are a confound in the ERP CORE N400 experiment, we'll ask whether our ICA-based blink correction procedure was effective at minimizing the voltages produced by the blinks. It will be easiest to determine this by looking at grand average data.

Two scripts were run to create grand averages that will help us determine if blink correction was successful. The first was named **N400_correction_rejection_phase4_rejection.m**. For each of our 20 participants, this script took the ICA-corrected data (**X_N400_preprocessed_pruned**), added an EventList, ran BINLISTER using a bin descriptor file named **BDF_N400_Targets.txt** to sort the events into bins (bin 1 = related targets; bin 2 = unrelated targets), and epoched the data. Then, artifact detection was applied to mark epochs with C.R.A.P. (which will be described more in the next section). An averaged ERPset was then created from this EEGset for each participant.

This script also created two new channels, named **H-bipolar-corr** and **V-bipolar-corr**. These channels allow us to better see how well the artifact correction worked by creating bipolar HEOG and VEOG signals from the corrected data. When we ran ICA, we corrected the channels we ordinarily use to detect blinks and eye movements (VEOG-lower, FP2, HEOG-left, and HEOG-right). We can therefore reconstruct bipolar channels from these corrected channels to better see whether the artifacts were fully corrected. The H-bipolar-corr channel was created by taking the corrected HEOG-left and HEOG-right channels and making a bipolar channel by computing HEOG-left minus HEOG-right. The V-bipolar-corr channel was created by taking the corrected VEOG-lower and FP2 channels and making a bipolar channel by computing VEOG-lower minus FP2. If blink correction worked well, then VEOG-lower and FP2 should have no blink activity left in them, and the difference between them should be close to zero if we assume that these two sites have equivalent brain activity. In other words, if VEOG-lower and FP2 have the same brain activity and differ only in blink activity, then the difference between VEOG-lower and FP2 should be close to zero if ICA has removed the blink activity.

Looking at the Corrected Bipolar VEOG in One Participant

To see this for Subject 1, click **Load ERP** in the **ERPsets** panel, navigate to **N400_Data > 1**, and select **1_N400_correction_plus_rejection.erp**. The uncorrected and corrected EOG channels should look like Screenshot 9.25.



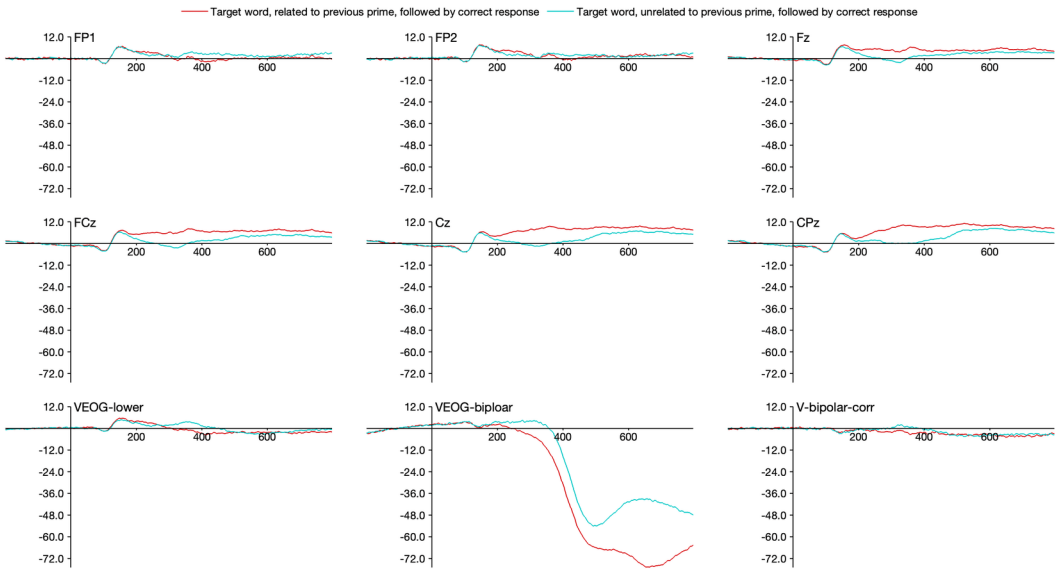
Screenshot 9.25

You can see that the corrected bipolar VEOG channel (V-bipolar-corr) is now nearly flat, indicating that the blink voltage has been largely eliminated. You can also see that the large difference between related and unrelated targets in the uncorrected bipolar VEOG channel (VEOG-bipolar) is no longer evident in the corrected bipolar VEOG channel (V-bipolar-corr). That looks like successful blink correction, at least for this one participant.

Looking at the Corrected Bipolar VEOG in the Grand Average

To more carefully evaluate the effectiveness of the blink correction, I created and ran an additional script named **N400_correction_rejection_phase5.m**. This script includes some important steps that will be described in a later section, but it did three things that were important for the present exercise. First, it made a grand average of all the single-participant ERPsets created by the previous script, which is named **Grand_N400_correction_plus_rejection.erp**. Go ahead and load this grand average using the **ERPsets** panel. Note that **N400_correction_rejection_phase5.m** uses the **ttest()** function, which requires the Statistics and Machine Learning Toolbox (see the previous section for more details). The script has instructions for commenting out the **ttest()** lines if you don't have this toolbox.

The uncorrected VEOG-bipolar signal is so large that it's hard to see the other channels. To reduce this problem, go to the **Bin & Channel Selection** panel, select channels **1, 2, 5, 9, 13, 16, 31, 33, and 35**, and click **Apply**. This will display the key electrode sites for examining blinks and the N400, as shown in Screenshot 9.26.



Screenshot 9.26

As was true for Subject 1, the corrected bipolar VEOG channel (V-bipolar-corr) is now nearly flat, indicating that the blink voltage has been greatly reduced. However, if you look closely, you can still see a difference of a couple microvolts between the related and unrelated targets from approximately 200-450 ms in this channel. That's very concerning! Note, however, that the difference between the related and unrelated targets is most pronounced from approximately 450-800 ms in the uncorrected bipolar VEOG channel (VEOG-bipolar), but there is no difference between conditions in this time range in the corrected bipolar VEOG channel. So, the correction seems to be working extremely well in the late part of the epoch, but it does not seem to be working as well during the N400 measurement window (which is when we really need it to work).

Another important thing to note is that the difference between the related and unrelated targets is much larger at the measurement site for the N400 (CPz) than in V-bipolar-corr. We would expect any residual blink activity to get smaller and smaller as we get farther away from the eyes, but the N400 effect is getting bigger. This means that at least most of the N400 effect at CPz must be brain activity rather than blink activity.

A More Detailed Analysis of the Corrected Bipolar VEOG Signal

The script also provided more detailed quantitative information about the difference between related and unrelated targets. Specifically, it created an unrelated-minus-related difference wave and measured the mean amplitude from this difference wave during the N400 measurement window (300-500 ms) at the uncorrected VEOG-bipolar channel, at the corrected V-bipolar-corr channel, and at the CPz channel. Then it did a simple statistical analysis to determine whether these amplitudes were significantly different from zero (i.e., whether these amplitudes differed significantly between the related and unrelated targets). Here's what the script printed in the Command Window:

```
Summary of mean amplitude in the N400 latency range:
Comparison of unrelated-minus-related difference in the uncorrected VEOG-bipolar channel
    mean = 12.89, t(19) = 4.07, p = 0.0007
Comparison of unrelated-minus-related difference in the corrected VEOG-bipolar channel
    mean = 1.83, t(19) = 1.46, p = 0.1602
Comparison of unrelated-minus-related difference in the corrected VEOG-bipolar channel
    mean = -7.82, t(19) = -9.74, p = 0.0000
```

Without correction, there was a 12.89 µV difference in VEOG-bipolar amplitude between related and unrelated targets in the N400 latency range, which was statistically significant. With correction, this difference was reduced to 1.83 µV and was not statistically significant. At CPz, the difference was 7.82 µV and significant.

The fact that the effect was small and not statistically significant in the corrected VEOG-bipolar channel provides some reassurance that the blink correction was reasonably effective. However, you can't "prove the null" with a p value, so these results don't provide strong evidence that the blink correction completely eliminated the confound. In fact, with all 40 participants, the difference between related and unrelated targets was significant for the corrected VEOG-bipolar channel (Zhang et al., 2024). Thus, it seems as if ICA did not completely remove the blink confound.

A Potential Alternative Explanation for the Effect in the Corrected Bipolar VEOG Signal

Given the fact that the difference between related and unrelated targets was large at CPz and became progressively smaller at electrodes near the eyes (after correction), we can conclude with confidence that the N400 effect at CPz was mainly a result of brain activity, not blink activity. However, it still seems possible that some residual blink activity was volume-conducted from the eyes to the EEG electrodes and inflated the observed difference between related and unrelated targets at the EEG sites.

However, we should also consider another possibility: Just as blink activity can propagate from the eyes to the EEG electrodes, brain activity can propagate from the EEG electrodes to the VEOG electrodes. In other words, the larger brain response for unrelated targets than for related targets that we see at CPz might be volume-conducted to the FP2 and VEOG-lower electrodes, creating the difference we have been attributing to differences in blinking.

How can we tell whether some of the effect at CPz is volume-conducted blink activity or whether the entirety of the effect at V-bipolar-corr is volume-conducted brain activity? In our journal article (Zhang et al., 2024), we developed a statistical approach to answering this question using semipartial correlations. That approach is too complicated to demonstrate here, especially because it involves statistical analyses that are not built into eStudio. However, I will give you the bottom line: The statistical analysis provided evidence that the difference between related and unrelated targets at V-bipolar-corr did not consist of volume-conducted N400 activity, whereas a portion of the N400 effect at CPz could be explained by volume-conducted blink activity.

In other words, ICA-based blink correction was not sufficient to completely eliminate blink confounds in the ERP CORE N400 paradigm. That's a sobering conclusion. I'd like to think that we did a pretty good job of implementing ICA-based blink correction, and yet a small but statistically significant blink confound remained. However, I wasn't really surprised at the failure of ICA to completely eliminate blink confounds. As I've noted multiple times in this chapter, ICA is based on assumptions that are known not to be true (most notably that the number of sources is equal to the number of electrodes). As a result, we shouldn't expect ICA to be perfect. And it is therefore important to assess whether the imperfections in ICA might lead to effects in our data that we might think reflect brain activity but instead reflect blink activity.

Fortunately, you typically need to worry about this only when the procedures described in the previous section indicate that blinks differ between conditions, as in the ERP CORE N400 paradigm.

Was the Blink Correction "Good Enough"

The French Enlightenment author and philosopher Voltaire once wrote something that is often translated (slightly incorrectly) into English as "The perfect is the enemy of the good." With that in mind, perhaps we shouldn't require that ICA-based blink correction be perfect and instead be satisfied if it is good enough. But how should we define "good enough"?

In our journal article (Zhang et al., 2024), we proposed using propagation factors to help us decide what is good enough. Propagation factors tell us how much of the voltage generated by ocular artifacts would be expected to fade by the time they reach a given EEG electrode. In one of my favorite ERP methodology papers, Lins et al. (1993) provided normative propagation factors for a variety of electrode sites. This study indicated that a given blink voltage recorded from a VEOG-bipolar configuration would be expected to produce a propagated voltage of approximately 19% at Fz, approximately 10% at Cz, and approximately 5% at Pz. The 1.83 μV effect we observed in the bipolar VEOG channel after correction would therefore be expected to produce a propagated effect of approximately 0.35 μV at Fz, 0.18 μV at Cz, and 0.09 μV at Pz. Lins did not provide a propagation factor for CPz, but it must be somewhere between the factors for Cz and Pz. Thus, the propagated blink confound at CPz should be somewhere between 0.18 μV and 0.09 μV, which would be less than 2.5% of the 7.82 μV effect observed at CPz. Thus, although ICA did not completely eliminate the confounding eyeblink activity at CPz, the residual activity was quite small. I think this is "good enough" for this study and might even be described as "excellent" given how large blinks are relative to ERPs.

When we looked at the other ERP CORE experiments, we found that ICA-based blink correction was close to perfect in some paradigms and very good but not perfect in others. That's good news. It likely means that our ICA approach will be satisfactory

in studies using similar paradigms, high-quality EEG recording systems, a cool and dry recording environment, and highly cooperative participants. If your research is dissimilar in one or more ways, or if you don't follow our ICA procedure exactly, you should not assume that our approach will yield satisfactory results in your data. Instead, you should use the procedures described here to provide rigorous evidence about the effectiveness of your artifact correction pipeline.

In ending this section, I'd like to bring up another quote about perfection, this one from the famous American football coach, Vince Lombardi. When he first became coach of the Green Bay Packers in 1959, he met with his players and explained his philosophy, which was to aim for perfection. He noted that they would never actually achieve perfection, because nothing in this world is perfect. But it is still valuable to strive for perfection. Here's the actual quote: "Gentlemen, we will chase perfection, and we will chase it relentlessly, knowing all the while we can never attain it. But along the way, we shall catch excellence." Over the next 9 years, they won 5 NFL championships, including the first two Super Bowls. I have taken Lombardi's quote and made it my own personal philosophy for conducting scientific research, always chasing perfection in an effort to catch excellence. So, I encourage you to chase perfection in implementing ICA-based artifact correction: ICA will never be perfect, but when carefully applied, it can do an excellent job of minimizing several kinds of artifacts.

This page titled 9.10: Exercise- Was Blink Correction Effective? is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck.

9.11: Exercise- Does Artifact Correction+Rejection Improve Data Quality?

In the previous exercise, we carefully evaluated our blink correction procedure and found that it was not perfect but was quite good. Our next step is to ask whether following artifact correction with C.R.A.P. rejection is also effective. More precisely, we will ask whether the benefits of having fewer trials with high noise will outweigh the costs of having fewer trials in our averaged ERPs. We will operationalize this by asking whether the standardized measurement error (SME) is improved by the rejection of trials that exceeded our threshold for C.R.A.P. The SME takes into account both the amount of noise and the number of trials in a way that maps directly onto statistical power. All else being equal, statistical power will be greater if the SME is better.

We will also ask whether our correction of blinks and eye movements improved the SME. We don't need to worry that artifact correction will reduce the number of trials, but it is still worth asking whether it reduces noise. Delorme (2023) didn't find that artifact correction improved statistical power, but the SME might provide a more sensitive measure, so the value of artifact correction for noise reduction is an open question.

The script named **N400_correction_rejection_phase4_baseline.m** created ERPsets without any artifact correction or rejection, and the SME values in these ERPsets will serve as a baseline for comparison. The script named **N400_correction_rejection_phase4_rejection.m** created ERPsets with correction but not rejection and also ERPsets with correction plus rejection, and we will ask whether the SME values in these ERPsets were better than the SME values obtained without correction or rejection.

When the ERPsets were created by these scripts, a custom SME interval of 300-500 ms was created, matching the measurement window of the N400. We will focus on the SME values from this time window at CPz, which was the N400 measurement channel.

Details of Our Artifact Rejection Procedure

Here are the details of the artifact rejection. First, we wanted to exclude trials on which participants blinked at a time that interfered with seeing the target word. For this purpose, we applied the moving window peak-to-peak algorithm during the period from -25 to +225 ms with a threshold of 100 μV. We found that these parameters effectively detected blinks that occurred while the stimulus was visible (from 0 to 200 ms). There were hardly any blinks during this time window, because the participants knew when the targets would be presented. As a result, this part of the process had virtually no impact on the averaged ERP waveforms or the SME values. But I still like doing it—it's an example of trying to chase perfection. Note that when an artifact is detected in a given epoch, there are several different flags that can be set. Flag 1 is set for any artifact, but you can set additional flags to indicate what kind of artifact was detected for a given epoch. The present script set flag F2 when a blink was detected at the time of the stimulus.

A more important part of the process was detecting and rejecting trials with extreme voltages that were unlikely to reflect brain activity (i.e., C.R.A.P.). As described in Chapter 8, we used two different algorithms to detect these extreme voltages. First, we flagged epochs with flag F3 if the voltage went beyond a range of -100 to +100 μV at any time. This was mainly used to detect relatively slow changes in voltage over the course of the epoch. Second, the moving window peak-to-peak algorithm was used to set flag F4 for epochs in which the peak-to-peak voltage exceeded 100 μV within any 200-ms window. This was mainly used to detect sudden changes in voltage, like those produced by head movements. The scripts were set up so that the artifact detection parameters are read from a spreadsheet, making it possible to specify different parameters for different participants, but the same parameters worked well for all 20 of the present participants.

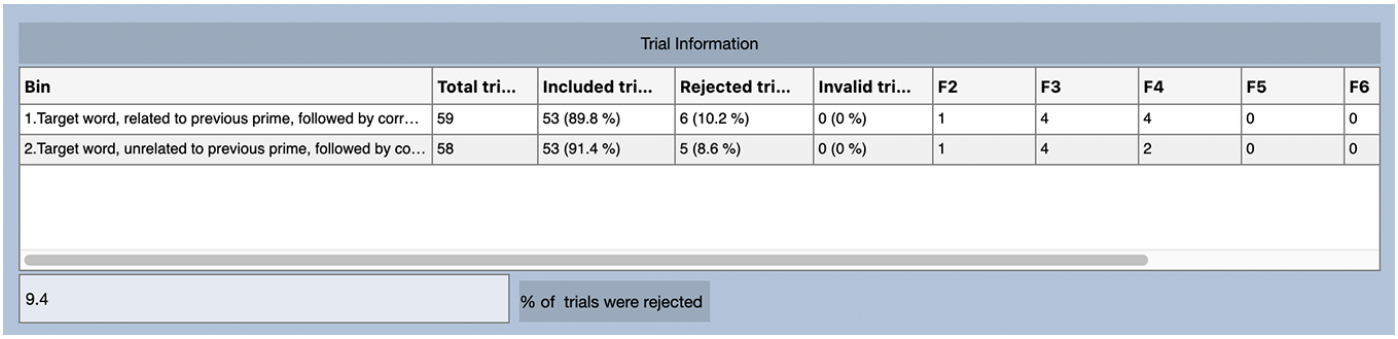
For all of the artifact detection tests used here, I enabled the option for prefiltering the data at 30 Hz to avoid getting "fooled" by high-frequency noise. We did not do that in the pipeline used in our journal article (Zhang et al., 2024), but I wish we had. It allowed me to use a lower rejection threshold that flagged more trials with real C.R.A.P. without flagging trials that merely had high-frequency noise. This kind of noise doesn't impact the mean amplitude scores used here, and it can be filtered out when other scoring methods are used, so I recommend using this prefiltering in most studies.

Assessing the Effects of Artifact Correction+Rejection on Data Quality

Let's first look at the effects of artifact correction and rejection on the data from Subject 29, who had a moderately large number of trials flagged for C.R.A.P. First reset everything with the **Reset** button. Then click Load ERP in the ERPsets panel, navigate to N400_Data > 29, and open these three ERPsets:

- **29_N400_correction_plus_rejection.erp**
- **29_N400_correction_without_rejection.erp**
- **29_N400_no_correction_or_rejection.erp**

Now let's see how many epochs were marked for rejection. Select **29_N400_correction_plus_rejection** in the **ERPsets** panel and then open the **ERP & Bin Information** panel and click the **Bin Info Table** button. This will pop up a window that includes a table with the details of the artifacts that were flagged. You will probably need to widen the window (or use the scroll bar) to see all the relevant flags. The table is shown in Screenshot 9.27. Below the table, you can see that 9.4% of epochs were flagged with artifacts. Flag F2 was set for one epoch in both bin 1 and bin 2, indicating that the participant blinked while the target word was visible for one epoch in each bin. Flag F3 was set for four epochs in each bin, indicating the number of epochs that exceeded the simple voltage threshold. Flag F4 was set for four epochs in bin 1 and two epochs in bin 2, indicating the number of epochs that exceeded the threshold for the moving window peak-to-peak amplitude algorithm. Some epochs were flagged with both F3 and F4, so the total number of flagged epochs for a given bin is less than the sum of the F2, F3, and F4 flags.



Trial Information

Bin	Total tri...	Included tri...	Rejected tri...	Invalid tri...	F2	F3	F4	F5	F6
1.Target word, related to previous prime, followed by corr...	59	53 (89.8 %)	6 (10.2 %)	0 (0 %)	1	4	4	0	0
2.Target word, unrelated to previous prime, followed by co...	58	53 (91.4 %)	5 (8.6 %)	0 (0 %)	1	4	2	0	0

9.4 % of trials were rejected

Screenshot 9.27

Now select all three ERPsets in the **ERPsets** panel, go to the **View Data Quality Metrics** panel, and click **Show in a Table**. You'll see the usual data quality table. Because you selected all three ERPsets before creating this table, the **Selected ERPset** popup menu at the top of the window will allow you to display the data quality for any of the ERPsets. Select **29_N400_no_correction_or_rejection** first. If you look at the aSME for bin 1 during the measurement window (300-500 ms) at the measurement channel (CPz), you'll see an aSME value of 2.670. This is our data quality without any correction or rejection. Now look at the value for **29_N400_correction_without_rejection**. It's very slightly better (2.5388). Now look at the value for **29_N400_correction_plus_rejection**. It's even better (2.3149). If you repeat this process for bin 2, you'll see the same pattern, with aSME values of 3.1710 for no correction or rejection, 3.1286 for correction without rejection, and 2.1534 for correction plus rejection. In other words, artifact correction improved the data quality slightly (i.e., slightly reduced the aSME), and combining correction with rejection improved the data quality somewhat more.

However, that was just for this one participant. The script named **N400_correction_rejection_phase5.m** summarizes the SME values for all 20 participants. If you run this script, you'll see that it ends by printing the following to the Command Window:

```
Mean percentage of trials marked for rejection: 3.12

Summary of aSME values:
RMS(aSME) = 2.308 without correction or rejection
RMS(aSME) = 2.255 after correction but no rejection
RMS(aSME) = 2.055 after correction and rejection
```

You can see that we're not rejecting a lot of trials on average (just 3.12%), but we're rejecting as many as 15% of trials in some participants. It turns out that a few noisy participants can really reduce your statistical power, even if the data are super clean in most participants.

The script's output shows that the SME for the study as a whole is worst when we don't perform either correction or rejection, a little better when we correct blinks (and eye movements in a few participants), and even better when we also exclude trials with large C.R.A.P. (and a few trials with eye movements at the time of the stimulus). Note that we're summarizing the SME across participants using the root mean square (RMS) of the single-participant values. The RMS is similar to the mean, but it gives greater weight to larger values; this reflects the fact that a few noisy participants have an outsized impact on your statistical power. Consequently, the RMS of the single-subject SME values is more directly related to statistical power than the mean of those values (Luck et al., 2021).

In the 20 participants included in this analysis, the combination of artifact correction and rejection reduced the SME by only 11%. That's an improvement, but it's not huge. However, those 20 participants were not a random sample, and when my lab included all 40 participants in our journal article, we found that artifact correction plus rejection cut the SME by over 50% (Zhang et al., 2024). When we looked at the other ERP CORE paradigms, we found an SME reduction of 14% for the N170 and the ERN, a reduction of 16% for the MMN, a reduction of 16% for the P3b, and a reduction of 36% for the MMN. Thus, the effects varied across paradigms. In all cases, however, the combination of artifact correction and rejection improved the data quality.

These percentage improvement values were generated by analyses of mean amplitude, and we found similar reductions in SME for peak amplitude. Interestingly, artifact correction and rejection had little or no effect on data quality for latency measures in the ERP CORE paradigms.

Conclusions About the Value of Artifact Correction and Rejection

The bottom line is that, at least for studies like the ERP CORE experiments, the combination of artifact correction and rejection described in this chapter can improve your data quality between a little and a lot, at least for amplitude measurements. Moreover, the artifact correction component of this pipeline can reduce or eliminate confounds, which is important for both amplitude and latency measurements.

In Chapter 7, I noted that the fundamental goal of EEG preprocessing is to accurately answer the scientific question that the experiment was designed to address. In Chapter 8, I further noted that artifacts interfere with answering scientific questions in 3 ways: they can reduce statistical power, they can be confounds, and they can indicate a problem with the sensory input. You've now seen rigorous, quantitative evidence that the combination of artifact correction and rejection can improve data quality (and therefore statistical power), minimize confounds (e.g., differences in blink voltages between conditions), and avoid sensory input problems (e.g., the eyes being closed when the stimulus appears). This means that the procedures described here can help you accurately answer the scientific question that your experiments are designed to address.

This page titled 9.11: Exercise- Does Artifact Correction+Rejection Improve Data Quality? is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck.

9.12: Matlab Scripts For This Chapter

I've created two sets of scripts for this chapter, one showing how to implement artifact correction in the MMN data and one showing how to evaluate the effects of artifact correction and rejection in the N400 data. You can find both sets of scripts in the Chapter_9 folder.

MMN Scripts

There are four MMN scripts. The script names begin with **MMN_artifact_correction_** and then end in **phase1**, **phase2**, **phase3,** or **phase4**. They divide the process into—you guessed it!—four phases.

- The first phase takes the original dataset and creates a copy that is optimized for the ICA decomposition by filtering from 1-30 Hz, downsampling to 100 Hz, deleting the break periods, and rejecting segments that contain huge C.R.A.P. artifacts.
- The second phase performs the ICA decomposition on this optimized dataset. I put that in a separate script because it takes quite a while to run.
- The third phase transfers the ICA weights to the original dataset, removes the artifactual ICs, and then interpolates any bad channels.
- The fourth phase finishes the EEG processing (binning, epoching, artifact detection for C.R.A.P.) and makes averaged ERPs (two different ERPsets, one in which epochs with artifacts are excluded and one in which all epochs are included)

The script processes the data from Subjects 1–10. I spent an entire afternoon going through the data carefully, deciding which channels should be interpolated, determining which ICs to remove, etc. But even after devoting that much time, I did only a superficial job. You can probably do better if you spend more time.

While I was going through the data to determine the various parameters for a given participant, I put the parameters in a set of spreadsheets. The scripts then read from these spreadsheets. For the first two phases, I didn't actually run the scripts on all 10 participants at once. As noted in the comments inside the scripts, you can modify the scripts slightly to work on one participant at a time. I went back and forth between the scripts and the GUI to set the various parameters. You won't need to do that to run the scripts, because all the parameters are in the spreadsheets (which are provided in the Chapter_9 folder). However, you'll probably want to adopt this one-participant-at-a-time approach when you're analyzing your own data.

Note that the Excel spreadsheets contain a column for comments. These comments aren't used by the scripts, but they're very useful in helping you to remember why you made various decisions. Your future self will thank you.

N400 Scripts

The initial N400 EEGsets in the **N400_Data** folder have already been preprocessed a little, including high-pass filtering the data at 0.1 Hz, referencing the EEG and EOG channels to the average of P9 and P10, and creating bipolar VEOG and HEOG channels. Each of these EEGsets is named **X_N400_preprocessed.set**, where **X** is the subject number. The first three N400 scripts implement artifact correction and interpolation of bad channels:

- **N400_correction_rejection_phase1.m** takes each participant's initial EEGset **(X_N400_preprocessed**) and makes a copy of this EEGset (named **X_N400_preprocessed_optimized**) that is optimized for the ICA decomposition process. This involves downsampling to 100 Hz, filtering from 1-30 Hz, deleting break periods, and deleting any segments of the continuous data that contain extra-huge C.R.A.P. that might interfere with the ICA decomposition.
- **N400_correction_rejection_phase2.m** takes each participant's optimized EEGset and performs the ICA decomposition (excluding the bipolar EOG channels and any "bad" channels that will later be interpolated). This results in an EEGset for each participant named **X_N400_preprocessed_optimized_weights**.
- **N400_correction_rejection_phase3.m** transfers the ICA weights back to the original EEGset and then uses those weights to perform the artifact correction by reconstructing the EEG from the ICs, but without the ICs corresponding to blinks and possibly eye movements (if a clear eye movement IC was present). Interpolation is then performed on the "bad" channels. The resulting EEGset is named **X_N400_preprocessed_pruned**.

The next script, **N400_correction_rejection_phase4_count_blinks.m**, is used to count the number of blinks in each bin.

For each of the 20 participants, this script takes the ICA-corrected data (**X_N400_preprocessed_pruned**), adds an EventList, runs BINLISTER using a bin descriptor file named **BDF_N400_Targets.txt** to sort the events into bins (bin 1 = related targets; bin 2 = unrelated targets), and epochs the data. Then, artifact detection is applied to the uncorrected VEOG-bipolar channel to mark epochs containing blinks using the moving window peak-to-peak amplitude algorithm with a threshold of 100 μV, looking for blinks in every 200-ms window within the 1000-ms epoch. The resulting EEGset is saved in a file named **X_N400_preprocessed_pruned_be_allblinksmarked.set**, and an averaged ERPset is then created from this EEGset and saved in a file named **1_N400_count_blinks.erp**. After this is completed for all 20 participants, the script calculates the proportion of trials marked for rejection in each bin for each participant and stores them in an Excel spreadsheet named **Blinkrate.xslx**. It then displays the mean across participants for each bin along with a statistical test of the difference between the two bins in the Command Window.

Two scripts are used to help provide information about the effectiveness of correction and rejection. One of these scripts does not implement artifact correction or rejection and serves as a baseline for comparison (**N400_correction_rejection_phase4_baseline.m**). For each of our 20 participants, this script took the ICA-corrected data (**X_N400_preprocessed_pruned**), added an EventList, ran BINLISTER using a bin descriptor file named **BDF_N400_Targets.txt** to sort the events into bins (bin 1 = related targets; bin 2 = unrelated targets), and epoched the data. This script also created two new channels, named **H-bipolar-corr** and **V-bipolar-corr**, that allow us to better see how well the artifact correction worked by creating bipolar HEOG and VEOG signals from the corrected data. The H-bipolar-corr channel was created by taking the corrected HEOG-left and HEOG-right channels and making a bipolar channel by computing HEOG-left minus HEOG-right. The V-bipolar-corr channel was created by taking the corrected VEOG-lower and FP2 channels and making a bipolar channel by computing VEOG-lower minus FP2. The script then creates averaged ERPs for each participant, using a custom aSME time window of 300-500 ms (the N400 measurement window).

The other script (**N400_correction_rejection_phase4_rejection.m**) is the same as **N400_correction_rejection_phase4_baseline.m** except that (a) it operates on ICA-corrected EEGsets, (b) it marks epochs with C.R.A.P. and epochs with blinks at the time of the stimulus, and (c) it creates two ERPsets for each participant, one in which all epochs were included (i.e., implementing correction but not rejection) and one in which marked epochs were excluded (i.e., implementing both correction and rejection).

The final script (**N400_correction_rejection_phase4.m**) makes grand averages from the single-participant ERPsets provided by **N400_correction_rejection_phase4_baseline.m** and **N400_correction_rejection_phase4_rejection.m**. It also provides quantitative information about the effectiveness of correction and correction+rejection. For each single-participant ERPset, it creates an unrelated-minus-related difference wave and measures the mean amplitude from 300-500 ms from CPz, from the uncorrected bipolar VEOG channel (VEOG-bipolar), and the corrected bipolar VEOG channel (V-bipolar-corr). It then calculates the mean amplitude for each of these channels and compares it against zero with a one-sample t test. This is done separately for the data without correction or rejection, for the data with correction but without rejection, and for the data with both correction and rejection.

This page titled 9.12: Matlab Scripts For This Chapter is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 9.10: Matlab Scripts For This Chapter by Steven J Luck is licensed CC BY 4.0.

9.13: Key Takeaways and References

Key Takeaways

- Just as with artifact rejection, the overarching goal in designing an artifact correction strategy is to maximize the likelihood that you will obtain an accurate answer to the scientific question your study is designed to answer. This typically involves reducing noise to maximize statistical power, avoiding artifactual voltages, and avoiding unwanted changes in the sensory input.
- Some of the assumptions of ICA are known to be incorrect, and it is therefore imperfect. In practice, ICA works best for large and frequent artifacts, such as blinks. I therefore recommend a conservative correction strategy in which you remove only the small set of ICs that correspond with well-understood artifacts and that are well isolated by the ICA decomposition.
- You can significantly improve the ICA decomposition process by performing it on a copy of the EEGset that has had addition steps applied to minimize activity that would otherwise interfere with the decomposition. This includes deleting break periods, deleting segments with extra-huge C.R.A.P., and filtering the data from 1-30 Hz. The ICA weights are then copied back to the original EEGset.
- You can determine which ICs are artifacts by looking at the scalp distribution, time course, and frequency content. You can use ICLabel to help you determine which ICs to correct, but you should combine this with manual inspection.
- In most cases, it is best to combine correction for ocular artifacts with rejection of epochs with large C.R.A.P. (along with rejection of epochs with blinks or eye movements that might interfere with the perception of the stimulus).
- Artifact correction and rejection are imperfect, and you should check their effectiveness with the following steps:
    - Assess whether blinks are actually a confound if not corrected (i.e., whether they differed between experimental conditions)
        - If so, assess whether the artifact correction process eliminated the confound.
            - If the confound was not fully eliminated, assess whether the remaining confound was large enough to have a meaningful impact on the ERP effects.
    - Assess whether data quality (SME) was improved by correction of EOG artifacts and whether it was further improved by also rejecting epochs with large C.R.A.P.

References

Artoni, F., Delorme, A., & Makeig, S. (2018). Applying dimension reduction to EEG data by Principal Component Analysis reduces the quality of its subsequent Independent Component decomposition. *NeuroImage*, *175*, 176–187. https://doi.org/10.1016/j.neuroimage.2018.03.016

Berger, H. (1929). Ueber das Elektrenkephalogramm des Menschen. *Archives Fur Psychiatrie Nervenkrankheiten*, *87*, 527–570.

Bigdely-Shamlo, N., Mullen, T., Kothe, C., Su, K.-M., & Robbins, K. A. (2015). The PREP pipeline: Standardized preprocessing for large-scale EEG analysis. *Frontiers in Neuroinformatics*, *9*. https://doi.org/10.3389/fninf.2015.00016

Box, G. E. P. (1976). Science and Statistics. *Journal of the American Statistical Association*, *71*(356), 791–799. https://doi.org/10.1080/01621459.1976.10480949

Delorme, A. (2023). EEG is better left alone. *Scientific Reports*, *13*, 2372. https://doi.org/10.1038/s41598-023-27528-0

Dimigen, O. (2020). Optimizing the ICA-based removal of ocular EEG artifacts from free viewing experiments. *NeuroImage*, *207*, 116117. https://doi.org/10.1016/j.neuroimage.2019.116117

Drisdelle, B. L., Aubin, S., & Jolicoeur, P. (2017). Dealing with ocular artifacts on lateralized ERPs in studies of visual-spatial attention and memory: ICA correction versus epoch rejection. *Psychophysiology*, *54*, 83–99. https://doi.org/10.1111/psyp.12675

Groppe, D. M., Makeig, S., & Kutas, M. (2009). Identifying reliable independent components via split-half comparisons. *Neuroimage*, *45*, 1199–1211. https://doi.org/10.1016/j.neuroimage.2008.12.038

Heise, M. J., Mon, S. K., & Bowman, L. C. (2022). Utility of linear mixed effects models for event-related potential research with infants and children. *Developmental Cognitive Neuroscience, 54*, 101070. https://doi.org/10.1016/j.dcn.2022.101070

Kappenman, E. S., Farrens, J. L., Zhang, W., Stewart, A. X., & Luck, S. J. (2021). ERP CORE: An open resource for human event-related potential research. *NeuroImage*, *225*, 117465. https://doi.org/10.1016/j.neuroimage.2020.117465

Klug, M., & Gramann, K. (2021). Identifying key factors for improving ICA-based decomposition of EEG data in mobile and stationary experiments. *European Journal of Neuroscience, 54*, 8406–8420. https://doi.org/10.1111/ejn.14992

Luck, S. J. (2014). *An Introduction to the Event-Related Potential Technique, Second Edition*. MIT Press.

Luck, S. J., Stewart, A. X., Simmons, A. M., & Rhemtulla, M. (2021). Standardized measurement error: A universal metric of data quality for averaged event-related potentials. *Psychophysiology*, *58*, e13793. https://doi.org/10.1111/psyp.13793

Näätänen, R., & Kreegipuu, K. (2012). The mismatch negativity (MMN). In S. J. Luck & E. S. Kappenman (Eds.), *The Oxford Handbook of Event-Related Potential Components* (pp. 143–157). Oxford University Press.

Pion-Tonachini, L., Kreutz-Delgado, K., & Makeig, S. (2019). ICLabel: An automated electroencephalographic independent component classifier, dataset, and website. *NeuroImage*, *198*, 181–197. https://doi.org/10.1016/j.neuroimage.2019.05.026

Winkler, I., Debener, S., Müller, K.-R., & Tangermann, M. (2015). On the influence of high-pass filtering on ICA-based artifact reduction in EEG-ERP. 2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), 4101–4105. https://doi.org/10.1109/EMBC.2015.7319296

Zhang, G., Garrett, D. R., Simmons, A. M., Kiat, J. E., & Luck, S. J. (2024). Evaluating the effectiveness of artifact correction and rejection in event-related potential research. *Psychophysiology*, *61*, e14511. https://doi.org/10.1111/psyp.14511

This page titled 9.13: Key Takeaways and References is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 9.11: Key Takeaways and References by Steven J Luck is licensed CC BY 4.0.

10: SCORING AND STATISTICAL ANALYSIS OF ERP AMPLITUDES AND LATENCIES

Learning Objectives

In this chapter, you will learn to:

- Score the magnitude of an ERP component using the peak amplitude and the mean amplitude.
- Score the timing of an ERP component using the peak latency, fractional area latency, and fractional peak latency.
- Evaluate these different scoring methods in terms of measurement error, effect sizes, and bias.
- Conduct statistical analyses that minimize the likelihood of bogus-bug-significant results.
- Correlate ERP measures with behavioral measures.

This chapter focuses on the final steps of data analysis, in which you quantify the amplitudes and/or latencies of your ERPs and conduct a statistical analysis. It doesn't cover every possible way of scoring amplitudes and latencies, and it barely scratches the surface of the statistical analysis of ERP data. However, it covers the scoring procedures that are used most often (or that should be used most often), along with some very simple statistical analyses. Additional details about scoring can be found in Chapter 9 of Luck (2014), and a much more in-depth treatment of statistical analysis can be found in Chapter 10 of that book. In particular, I encourage you to read about the *jackknife* and *mass univariate* statistical approaches.

One reason that I don't go too deeply into statistical analyses in this chapter is that ERPLAB doesn't include statistical functions, and I don't want to have to explain how to use some other statistical package. I'm assuming that you already know how to conduct basic statistical analyses (*t* tests and within-subjects ANOVAs) and have a statistical package that you can use to perform these analyses. If you don't, I recommend JASP (Love et al., 2019), which is free and easy to use. It's what I used for the analyses in this chapter. And it's really just a convenient interface into R, which provides tremendous power and flexibility (R Core Team, 2016).

The exercises in this chapter will examine the *lateralized readiness potential* (LRP), which reflects motor preparation. The data are from the ERP CORE flankers experiment. However, the lessons you will learn can be applied to almost any ERP component in almost any paradigm. And the LRP provides excellent opportunities to ask interesting questions about both amplitudes and latencies.

Quantifying amplitudes and latencies is often called the *measurement* process, and in ERPLAB it's done with the *Measurement Tool*. Recently, however, I've started using the term *scoring* instead of *measurement*. When we put electrodes on the scalp and record the EEG, that feels like we're actually measuring something (the voltages on the scalp). But applying an algorithm to an ERP waveform and hoping that it accurately captures the magnitude or timing of some underlying brain signal doesn't seem like taking a *measurement*. I now prefer the term *scoring* as a more neutral term that is used in many other research areas. For example, you might score the amplitude or latency of a given ERP component.

10.1: Data for This Chapter

10.2: Design of the Flankers Experiment

10.3: Exercise- Examining the Grand Averages

10.4: Exercise- A First Pass at Scoring and Statistical Analysis

10.5: Exercise- Simplifying the Statistical Analysis

10.6: Exercise- Peak Amplitude

10.7: Exercise- Peak Latency

10.8: Exercise- Fractional Area Latency

10.9: Exercise- Quantifying Onset Latency

10.10: Exercise- Collapsing Across Channels and Correlating Latencies with Response Times

10.11: Matlab Scripts For This Chapter
10.12: Key Takeaways and References

This page titled 10: Scoring and Statistical Analysis of ERP Amplitudes and Latencies is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 10: Scoring and Statistical Analysis of ERP Amplitudes and Latencies by Steven J Luck is licensed CC BY 4.0.

10.1: Data for This Chapter

The data we'll be using for the exercises in this chapter can be found in the Chapter_10 folder in the master folder: https://doi.org/10.18115/D5G045.

This page titled 10.1: Data for This Chapter is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 10.1: Data for This Chapter by Steven J Luck is licensed CC BY 4.0.

10.2: Design of the Flankers Experiment

This section will provide a brief overview of the experimental design and main results from the ERP CORE flankers experiment. This experiment was designed to isolate two different ERP components, the lateralized readiness potential (LRP) and the error-related negativity (ERN).

The LRP is a negative voltage over the hemisphere contralateral to a hand that is being prepared for a response. For example, if I were to tell you "raise your left pinkie," a negative voltage would be present over the motor cortex of the right hemisphere, preceding the actual pinkie response by 100-200 ms. The LRP is related to the preparation rather than the execution of the response. For example, if I were to tell you "raise your right pinkie when I say GO," the negative voltage over the right hemisphere would appear right away, even before I said "GO." Like the N2pc component, the LRP is isolated with a contralateral-minus-ipsilateral difference wave (relative to the side of the response hand). For an excellent review of the LRP, see Smulders and Miller (2012).

The ERN is a negative voltage with a maximum amplitude near FCz that occurs on error trials (usually in tasks where the error is obvious to the participant). It typically begins shortly before the actual buttonpress response and peaks 50-100 ms after the response. To get a good ERN, you need participants to make enough errors that you have a decent number of trials for averaging, but not so many errors that they're not really putting any effort into doing the task correctly. For an excellent review of the ERN, see Gehring et al. (2012).

In the ERP CORE, we used a version of the Eriksen flankers task to elicit these components (Eriksen, 1995). As illustrated in Figure 10.1.A, each stimulus array contained a central target arrow that pointed leftward or rightward with equal probability, surrounded by two flanking arrows on each side. Participants were instructed to make a rapid buttonpress response for each array, pressing with the left index finger if the central arrow pointed leftward and with the right index finger if the central arrow pointed rightward. They were instructed to ignore the flankers, which pointed in the same direction as the central target arrow on 50% of trials (called *Compatible* trials) and pointed in the opposite direction on the other 50% (called *Incompatible* trials). Each array was presented for 200 ms, and successive arrays were separated by an interstimulus interval of 1200–1400 ms. To make sure we had a good number of error trials, participants were told to speed up if they were making errors on fewer than 10% of trials and to slow down if they were making errors on more than 20% of trials.

In the ERP CORE paper (Kappenman et al., 2021), we focused our LRP and ERN analyses on ERPs that were time-locked to the response (i.e., the response rather than the stimulus was used as time zero when the data were epoched and averaged). The LRP was isolated by examining trials with left-hand versus right-hand responses. Figure 10.1.B shows the grand average ERPs, with one waveform for the ipsilateral hemisphere (C3 for left-hand response trials averaged with C4 for right-hand response trials) and another waveform for the contralateral hemisphere (C4 for left-hand response trials averaged with C3 for right-hand response trials). The LRP is a negative voltage over the contralateral hemisphere that is superimposed on other non-lateralized brain activity. To isolate the LRP from the other brain activity, we constructed a contralateral-minus-ipsilateral difference wave (Figure 10.1.C). You can see that the LRP in this difference wave starts to head in a negative direction approximately 120 ms prior to the response, and it peaks shortly before the response. (The slight positive voltage in the difference wave prior to -150 ms is likely an artifact of the 0.1 Hz high-pass filter that was applied to the continuous EEG at an early step in the analysis.)

The ERN is examined by comparing trials with correct trials and trials with incorrect trials (Figure 10.1.D). A sharp negative wave is superimposed on the positive voltage that would otherwise occur, peaking shortly after the response. This is then followed by a more positive voltage on the error trials than on the correct trials (called the error positivity or P_E). The ERN and P_E are often isolated from the other brain activity by making an error-minus-correct difference wave (Figure 10.1.E).

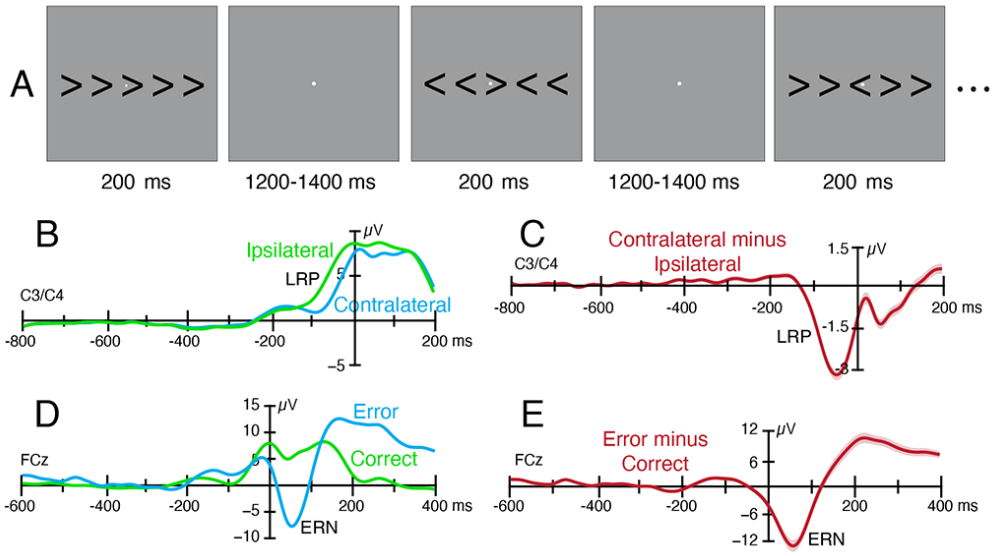


Figure 10.1. Experimental paradigm and results from the ERP CORE flankers experiment. (A) Example sequence of stimuli. Participants were instructed to press a left- or right-hand button depending on whether the central arrow pointed leftward or rightward, respectively. They were instructed to ignore the flanking arrows, which could be either compatible or incompatible with the direction of the central arrow. (B) Grand average ERP waveforms contralateral or ipsilateral to the response hand, time-locked to the response on correct trials, and averaged over compatible and incompatible trials and all 40 participants. (C) Grand average LRP difference wave, which was created by subtracting the ipsilateral ERP waveform from the contralateral ERP waveform. (D) Grand average ERP waveforms on correct trials and error trials, time-locked to the response, and averaged over compatible and incompatible trials and all 40 participants. (E) Grand average ERN difference wave, which was created by subtracting the ERP waveform for the correct trials from the ERP waveform for the error trials.

This chapter will focus on the LRP, not the ERN. In addition, we'll look at stimulus-locked LRP averages instead of response-locked averages, and we'll include only the correct trials in the averages.

In the main LRP analyses in the ERP CORE paper (Kappenman et al., 2021), we collapsed across compatible and incompatible trials for the sake of simplicity. In the present chapter, we're going to look at the LRP separately for these trial types. Many prior experiments have found that response times (RTs) are slowed on incompatible trials relative to compatible trials, and this is mainly because information about the flankers "leaks through" to response selection mechanisms. On compatible trials, this helps to activate the correct response. On incompatible trials, however, the incorrect response may be activated, which slows down the response (and often leads to errors). This can sometimes be observed in the LRP waveform, which may be contralateral to the incorrect hand briefly before becoming contralateral to the correct hand on incompatible trials (Gratton et al., 1988). In addition, the onset latency of the LRP is delayed on incompatible trials relative to compatible trials. We didn't do any of these analyses for the ERP CORE paper, but I thought they would be interesting to do in the present chapter. For one thing, they'll give us an opportunity to look at several different measures of amplitude and latency. For another, we won't know the results until we do the analyses, so there will be some drama!

This page titled 10.2: Design of the Flankers Experiment is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 10.2: Design of the Flankers Experiment by Steven J Luck is licensed CC BY 4.0.

10.3: Exercise- Examining the Grand Averages

Ordinarily, you would decide exactly how to quantify and analyze the ERP amplitudes and latencies prior to seeing the data. If you first look at the grand average ERP waveforms, the analysis parameters you select will likely be influenced by noise in the data, and you'll have a high likelihood of finding significant effects that reflect noise rather than true effects and are completely bogus. This is a super important point! I'm not going to dwell on it here, because I've written about it extensively elsewhere (see especially Luck & Gaspelin, 2017). However, keep this point in mind throughout the chapter (especially in this first exercise, where we are going to look at the data before we develop our analysis plan—exactly what you shouldn't do!).

I've already created the averaged ERP waveforms for looking at the LRP. If you go to the **Chapter_10** folder, you'll see a subfolder named **Data**, and inside that subfolder you'll see another subfolder named **ERPsets** that contains an ERPset for each of the 40 participants. To create these ERPsets, I referenced the data to the average of P9 and P10, high-pass filtered at 0.1 Hz (12 dB/octave), and then applied ICA-based artifact correction for blinks and horizontal eye movements (using the optimized approach described in the chapter on artifact correction). The next step was to add an EventList and then run BINLISTER to create 4 bins:

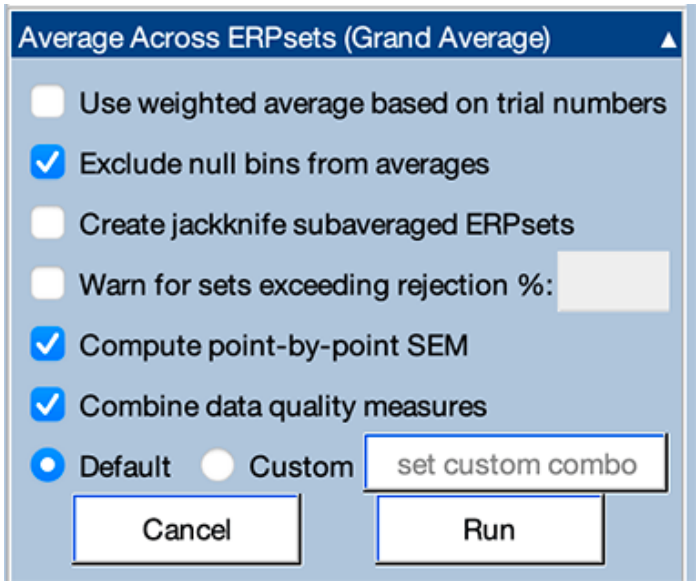
- Bin 1: Left-Pointing Target with Compatible Flankers, Followed by Left Response
- Bin 2: Right-Pointing Target with Compatible Flankers, Followed by Right Response
- Bin 3: Left-Pointing Target with Incompatible Flankers, Followed by Left Response
- Bin 4: Right-Pointing Target with Incompatible Flankers, Followed by Right Response

Note that only correct responses were included in these bins because we're going to focus on the LRP rather than the ERN.

Next, I epoched the data from -200 to 800 ms relative to stimulus onset. I then performed artifact detection to mark trials with C.R.A.P., and finally I averaged the data, excluding the marked trials.

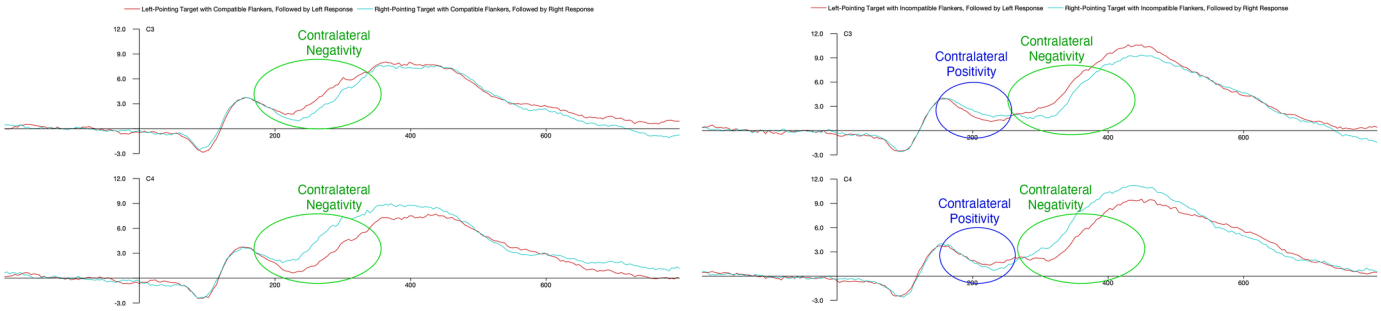
Let's load the data and make a grand average. First reset everything with the **Reset** button and set **Chapter_10** to be Matlab's current folder. Go to the **ERPsets** panel in the **ERP** tab, click **Load ERP**, and navigate to the **Chapter_10 > Data > ERPsets** folder. You can select and load all 40 ERPset files at once. The waveforms from Subject 40 will be visible in the main eStudio plotting region.

Now make a grand average by selecting all 40 ERPsets in the **ERPsets** panel, going to the **Average across ERPsets (Grand Average)** panel, and keeping all the options at their default values (as shown in Screenshot 10.1). Click **Run**, name the resulting ERPset **grand,** and save it as a file named **grand.erp**. You'll need it for a later exercise.



Screenshot 10.1

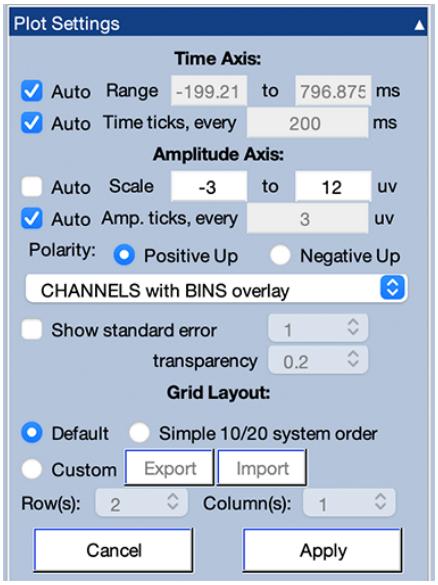
Now select the C3 and C4 channels (12 and 14) and the compatible trials (bins 1 and 2) in the **Bin & Channel Selection** panel. The result should look something like the left side of Screenshot 10.2. You can see that the voltage from ~200-400 ms is more negative for left-hand responses than for right-hand responses at C4 and more negative for right-hand responses than for left-hand responses at C3. In other words, the voltage is more negative over a given hemisphere when the response is made by the contralateral hand than when the response is made by the ipsilateral hand. This contralateral negativity is superimposed on the positive voltage of the P2 and P3 waves, so the overall voltage is positive. The LRP is simply added onto that positive voltage.



Screenshot 10.2

Now plot the incompatible trials (bins 3 and 4), which should look something like the right side of Screenshot 10.2. Again, you can see that the voltage is more negative for the contralateral hand than for the ipsilateral hand.

Is the contralaterality any different for the incompatible versus the compatible trials? It's a little difficult to tell when looking only at the compatible trials or only at the incompatible trials at a given time. So I'm going to show you a little trick for displaying both sets of waveforms at the same time. But first, let's make sure that we view both sets of waveforms with the same amplitude scale. Go to the Plot Settings panel, uncheck the **Auto** box for the **Amplitude Axis**, set the **Scale** to **-3** to **12** μV (as shown in Screenshot 10.3), and click **Apply**.



Screenshot 10.3

Now go to the **Plotting Options** popup menu in the lower right corner of the plotting region and select **Create Static/Exportable Plot**. This will pop up a new window with the current plot (which should still be bins 3 and 4). Now select bins 1 and 2 in the **Bin & Channel Selection** panel, click **Apply**, go back to the **Plotting Options** popup menu, and select **Create Static/Exportable Plot** again. This will pop up another window with the waveforms from these two bins. Now you can have both plots visible at the same time.

These windows use a standard Matlab plotting routine, which has some nice properties. One is that you can click anywhere on

a waveform, and it will show you the X and Y values (the latency and amplitude) at that point. One nice property is that you can save the plot to a file by going to the window's **File** menu and clicking **Save as...** You can save the image as a Matlab file, which allows you to reopen the plot again at a later time. You can also save it in several bitmap formats (which I generally don't like, because they look pixelated if you blow them up or print them on a printer). And you can save it in two vector graphics formats, PDF and SVG. These formats save an abstract description of the image that can be rendered beautifully at any resolution. SVG is a particularly nice format for editing in vector graphics programs such as Adobe Illustrator and Affinity Designer (which is why my lab uses fit or making pretty ERP figures).

Now that you can see both sets of waveforms at the same time, you can compare the compatible and incompatible ERPs. The main difference you can see is that the contralateral negativity starts before 200 ms for the compatible trials but doesn't start until almost 300 ms for the incompatible trials. In fact, there is a small contralateral positivity preceding the contralateral negativity for the incompatible trials.

The pattern is a little more complicated for the incompatible trials. At approximately 200 ms, you can see an opposite-polarity effect, with a more negative voltage for left-hand than for right-hand responses at C3 and a more negative voltage for right-hand than for left-hand responses at C4. This then reverses beginning at approximately 250 ms.

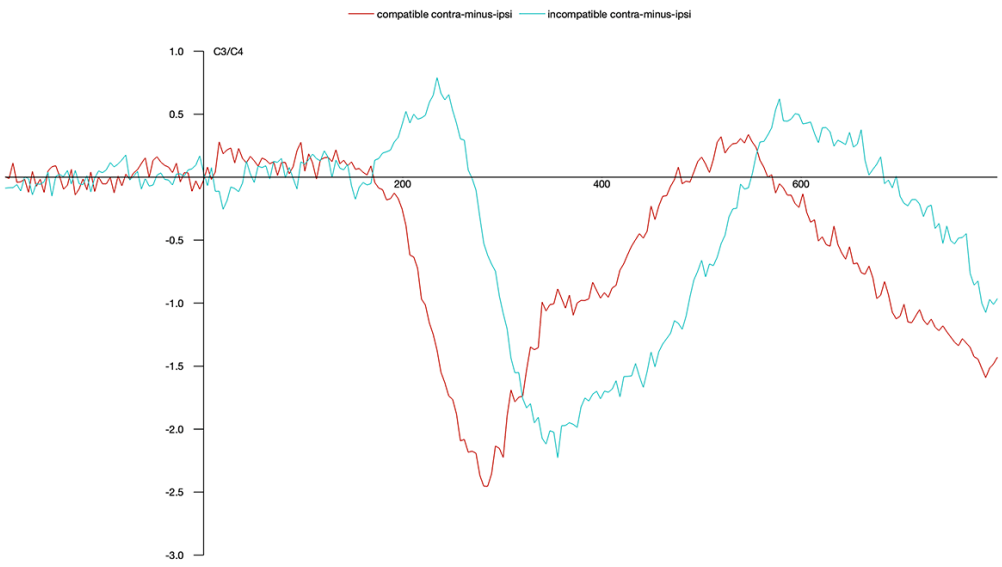
It's a little challenging to piece together everything that's happening with the waveforms shown in Screenshot 10.2. There are just a lot of waveforms to look at. We can simplify things by using ERP Bin Operations to collapse the data into contralateral waveforms (left-hemisphere waveforms on right-response trials averaged with right-hemisphere waveforms on left-response trials) and ipsilateral waveforms (left-hemisphere waveforms on left-response trials averaged with right-hemisphere waveforms on right-response trials).

Making these collapsed ERPsets is a little tricky (and is explained in detail in the ERP Bin Operations section of the ERPLAB Manual). To save some of your time, I've already made these collapsed ERPsets for each participant and made a grand average. Just load the ERPset named **grand_CI.erp** using the ERPsets panel (it's located in the main **Chapter_10** folder). You should plot the channel labeled C3/C4, and make one plot with bins 1 and 2 and another with bins 3 and 4. You can view these two plots simultaneously by using the **Create Static/Exportable Plot** trick.

This has reduced the number of waveforms, but it's still a little difficult to directly compare the effects for the compatible and incompatible trials. To provide a better comparison, and isolate the LRP from other overlapping voltages, we can use the **Bin Operations** panel to make a contralateral-minus-ipsilateral difference wave (Bin 1 minus Bin 2 for the compatible trials, and Bin 3 minus Bin 4 for the ipsilateral trials). Here are the equations:

```
bin5 = bin1 - bin2 label compatible contra-minus-ipsi
bin6 = bin3 - bin4 label incompatible contra-minus-ipsi
```

You can either run it yourself on the grand average or load **grand_CI_diff.erp** from the **Chapter_10** folder, which already has these bins. If you select the C3/C4 channel and bins 5 and 6 (and turn the **Amplitude Axis** scale back to **Auto** in the Plot Settings panel), you should see something like Screenshot 10.4.



Screenshot 10.4

Now we have only one pair of waveforms, making it much easier to compare the LRP for the compatible and incompatible trials. On the compatible trials, you can see a nice clear negativity from ~200-500 ms. On the incompatible trials, you can see an initial contralateral positivity, followed by a delayed contralateral negativity. That initial contralateral positivity for the incompatible trials occurs because people initially start preparing the incorrect response (even when they don't actually press the wrong button), and the contralateral positivity is actually a contralateral negativity relative to the hand used for the incorrect response.

This is pretty cool! Our main goal in including both compatible and incompatible trials in this experiment was to generate a sufficient number of errors for the ERN analysis (because errors are a lot more common on incompatible trials). We didn't intend to do any comparisons of compatible and incompatible trials, so this is the first time anyone has looked at these effects in the ERP CORE flankers experiment. It's gratifying to see that we found the same pattern that was observed in prior studies (e.g., Gratton et al., 1988), with a delayed LRP on incompatible trials that is preceded by an opposite-polarity deflection.

This page titled 10.3: Exercise- Examining the Grand Averages is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 10.3: Exercise- Examining the Grand Averages by Steven J Luck is licensed CC BY 4.0.
- 10.6: Exercise- Peak Amplitude by Steven J Luck is licensed CC BY 4.0.

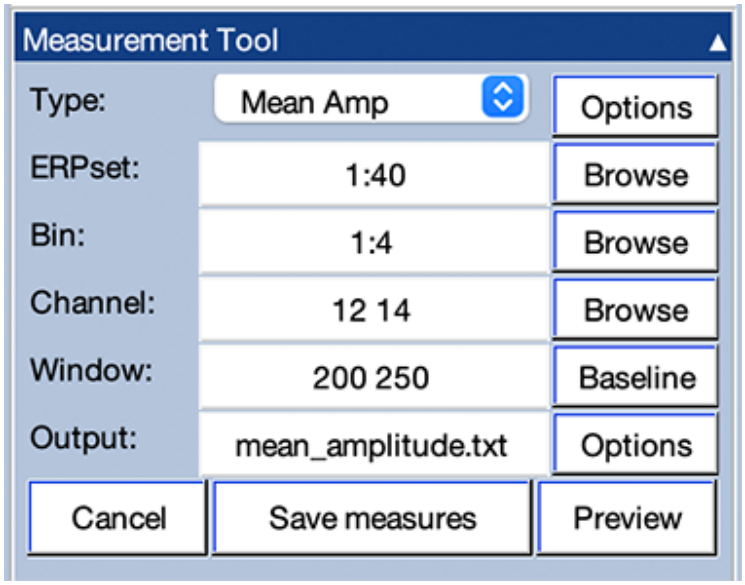
10.4: Exercise- A First Pass at Scoring and Statistical Analysis

Now let's quantify these effects and then see if they're statistically significant. In this exercise, we'll take the approach that many studies (including my own) used many years ago, but which has become less common as people figured out an approach that is both simpler and better. We'll start with the outdated approach because that will allow you to see (a) why the newer approach is better, and (b) how the newer approach tests that same scientific hypotheses as the original approach.

Let's start by scoring the LRP amplitude from each participant's ERP waveforms. There are many different ways to score the amplitude or latency of an ERP component, as described in detail in Chapter 9 of Luck (2014). In most cases, *mean amplitude* is the best way to quantify ERP amplitudes. In the present exercise, we'll measure the mean amplitude from 200-250 ms. This just means that the scoring routine will sum together the voltage values for each time point in this latency range and then divide by the number of time points. It's that simple! The simplicity of mean amplitude means that it's very easy to understand and even make mathematical proofs about, and it has some very nice properties that we'll see as we go through the next few exercises.

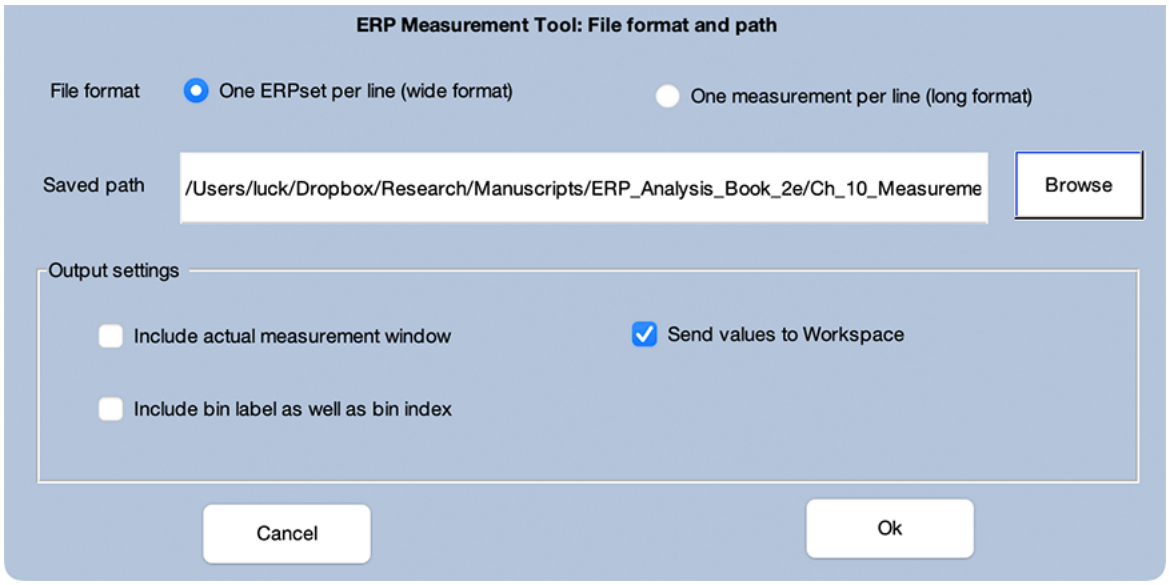
One of the most important issues involved in scoring ERPs is the choice of the time window. I chose 200-250 ms for this exercise because this is the approximate time range in which the opposite-polarity effect for incompatible trials is typically seen.

Reset everything with the **Reset** button, go to the **ERP** tab, make sure that **Chapter_10** is Matlab's current folder, and then load all 40 ERPsets from the **Chapter_10 > Data > ERPsets** folder. Then go to the **Measurement Tool** panel and enter the parameters shown in Screenshot 10.1. You'll specify that the scoring algorithm is **Mean Amp**, which we'll score from the C3 and C4 channels (**12 14**) in all four bins (**1:4**) from a time range of **200 250** (the *measurement window*). We're going to save the scores in a text file named **mean_amplitude.txt**, which will be saved in Matlab's current folder.



Screenshot 10.5

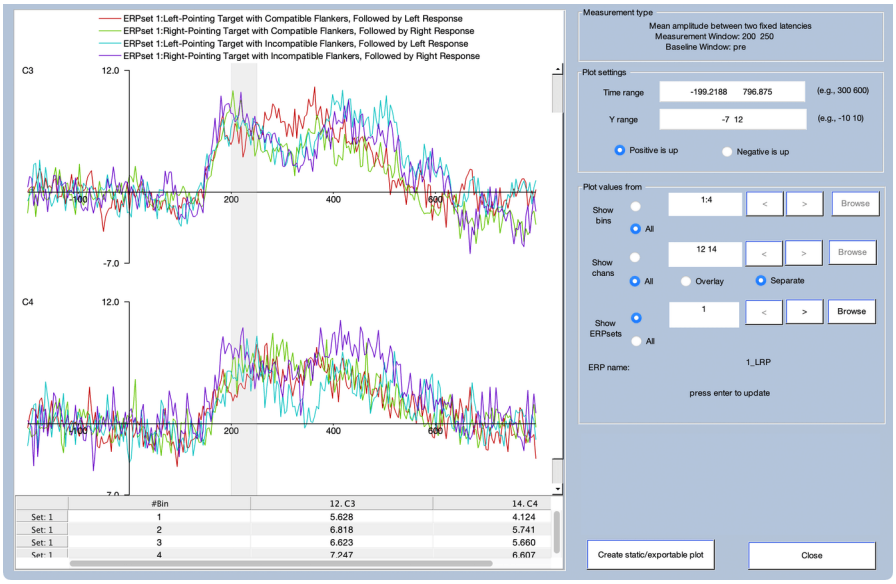
Also, click on the **Options** button next to the **Output** filename and set it up as shown in Screenshot 10.6 (although the path won't look the same). We'll discuss these options later. Click **Ok** to go back to the **Measurement Tool** panel.



Screenshot 10.6

It's really tempting to hit the **Save measures** button in the **Measurement Tool** panel and start analyzing the data, but you should always check the measurements against the ERP waveforms first. You can do this by clicking the **Preview** button. The Preview tool will open, and you'll see the ERP waveform from the first bin and first channel in the first ERPset. In the **Plot values from** portion of the window, select **All** next to **Show bins** and **All** next to **Show ERPsets**, and you should see something like Screenshot 10.7. (You may need to adjust the size of the window or the scroll bar positions to see both sets of waveforms fully.) Now you can see the data from all 4 bins at both electrode sites. The gray shaded region shows the measurement window (200-250 ms). The table at the bottom of the window shows the scores for each combination of bin and channel.

Not much can go wrong with the algorithm for measuring mean amplitude, but you may find surprising and problematic scores for some participants when you use other algorithms (e.g., peak amplitude or peak latency). Even with mean amplitude, it's humbling and informative to see how much the ERP waveforms vary across participants. For example, the waveforms you are looking at now—from Subject 1—are very noisy, don't have very distinct peaks, and don't differ much across bins. However, if you hit the **>** button on the **Show ERPsets** area of the window to look at the data from Subject 2, you'll see beautifully clean waveforms that are similar to those in the grand average, with distinct P2, N2, and P3 peaks. You'll also see that the measurement window runs right through the N2 peak in Subject 2, whereas the measurement window is at the time of a positive peak for Subject 1. Go ahead and use the **>** button to scroll through the waveforms for all 40 participants. You will see a lot of participant-to-participant variation. This is perfectly normal.



Screenshot 10.7

The variation you are seeing across participants brings up an important point about ERPs (and most other methods used in the mind and brain sciences): Averages are a convenient fiction. The ERP waveforms we get by averaging together multiple single-trial epochs may not be a good representation of the individual trials, and a grand average waveform across participants may not be a good representation of the individual participants. However, it is difficult to avoid averaging (or other methods that aggregate the data across trials or participants, such as regression). Chapters 2 and 8 in Luck (2014) discuss this issue in more detail. For now, we're just going to ignore this complication.

Once you've finished scanning through all the ERP waveforms using the Preview tool, click **Close** to go back to the **Measurement Tool** panel, and then click **Save measures** to save the scores to a file named **mean_amplitude.txt**. This file should now be present in the **Chapter_10** folder. Double-click on this file in Matlab's **Current Folder** pane to open it in the Matlab text editor. You'll see that it consists of a set of tab-separated columns. Matlab's text editor doesn't handle the tabs very well, so the column headings may not line up properly. I recommend opening it instead in a spreadsheet program like Excel. Here's what the first few lines should look like:

Example of amplitude scores

bin1_C3	bin1_C4	bin2_C3	bin2_C4	bin3_C3	bin3_C4	bin4_C3	bin4_C4	E
5.628	4.124	6.818	5.741	6.623	5.66	7.247	6.607	
6.902	7.534	4.72	8.7	3.629	6.546	6.178	5.47	
3.149	5.122	1.19	1.962	4.361	4.309	4.441	4.638	

Each row contains the data from one participant, and each column holds the score (mean amplitude value) for a given bin/channel combination for that participant. The Measurement Tool can also output the measurements in a "long" format in which each score is on a separate line (which is selected using the Options window shown in Screenshot 10.6). This long format is particularly good for using pivot tables to summarize the data in Excel, and it works well with some statistical packages. The "wide" format shown in the table above is ideal for statistical packages in which all the data for a given participant are expected to be in a single row (e.g., JASP).

Now that we have the scores, let's do a statistical analysis using a traditional ANOVA. You can use any statistical package you like. As I mentioned earlier, I recommend JASP if you don't already have a package that can do basic *t* tests and within-subjects ANOVAs. I used JASP 0.95.4 for this chapter.

The ANOVA should have three within-subjects factors, each with two levels: Electrode Hemisphere (left or right), Response Hand (left or right), and Compatibility (Compatible or Incompatible). When you load the data into your statistical software and specify the variables, it's really easy to get the columns in the wrong order. Your first step in the statistical analysis should therefore be to examine the table or plot of the descriptive statistics provided by your statistical software so that you can make sure that the data were organized correctly. Figure 10.3 shows what I obtained in JASP.

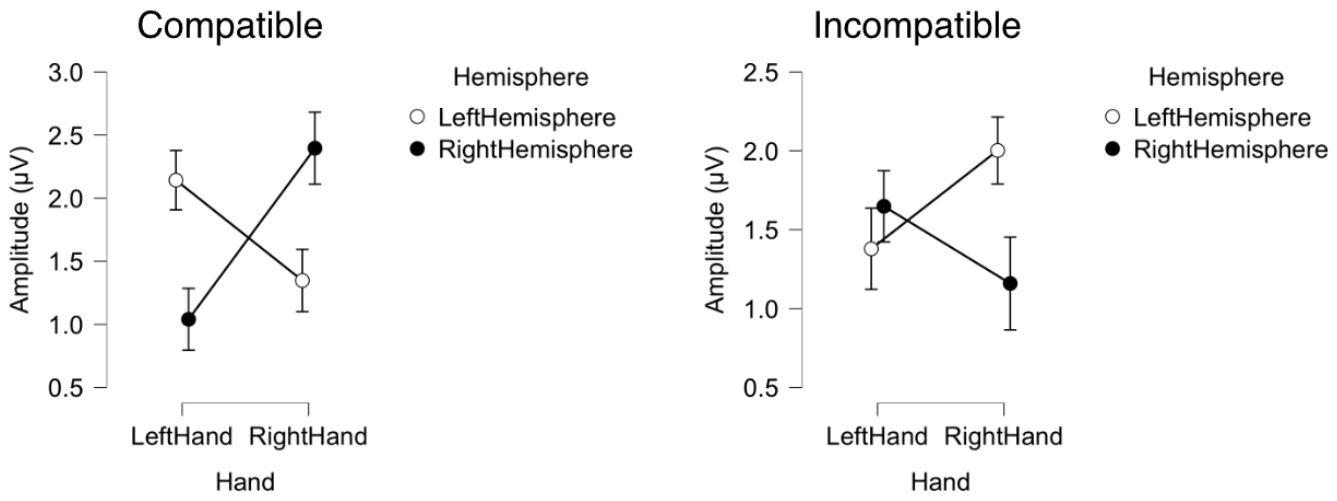


Figure 10.2. Means from each cell of the design, obtained with JASP. Error bars show the standard error of the mean in each cell.

But how do you know what the correct values should be? It turns out that with mean amplitude scores (but not most other scores), you get exactly the same values by averaging the single-subject scores or by obtaining the scores from the grand average waveforms (see the Appendix in Luck, 2014 for details). Load the grand average you created earlier (**grand.erp**) and run the Measurement Tool again, but specifying that it should measure only from this ERPset and save the results in a file named **mean_amplitude_grand.txt**. You can then compare those numbers to the values in the table or figure of descriptive statistics. Here are the values I obtained:

Example of amplitude scores

bin1_C3	bin1_C4	bin2_C3	bin2_C4	bin3_C3	bin3_C4	bin4_C3	bin4_C4	E
2.143	1.04	1.348	2.396	1.379	1.649	2.002	1.159	

These values exactly match the means shown in Figure 10.3. Success! Note that if you use some other scoring algorithm (e.g., peak amplitude) in your own studies, the values won't match exactly. However, you can at least make sure that the pattern is the same.

This verification process is very, *very,* **very** important! I estimate that you will find an error in at least 10% of analyses if you have three or more factors in your design.

Before we look at the inferential statistics, let's think about what main effects and interactions we would expect to see. First consider the Compatible condition, in which the voltage should be more negative for the contralateral hemisphere than for the ipsilateral hemisphere. This gives us a more negative voltage for left-hand responses than for right-hand responses over the right hemisphere, and the reverse pattern over the left hemisphere. In other words, the presence of the LRP is captured in the ANOVA as an interaction between Hemisphere and Hand. During this 200-250 ms time period, we expect to see an opposite effect for the Incompatible trials (because the voltage is more negative over the hemisphere contralateral to the incorrect response,

which makes it more positive contralateral to the correct response). Consequently, the difference between the Compatible and Incompatible trials should lead to a three-way interaction between Compatibility, Hemisphere, and Hand.

Table 10.1 shows the inferential statistics I obtained from JASP. You can see that the main effects of Hand and Hemisphere are not significant, consistent with the fact that you saw little or no overall difference between left-hand and right-hand responses or between the left and right hemispheres when you plotted the grand average ERPs (see Screenshot 10.2). The main effect of Compatibility is also not significant, consistent with the fact that the average voltage across cells for the Compatible condition was about the same as the average voltage across cells for the Incompatible condition.

By contrast, the interaction between Hemisphere and Hand was significant. This interaction is equivalent to asking about the contralaterality of the voltage if we averaged across Compatible and Incompatible trials. These two conditions yielded opposite-direction effects that partially cancel each other out. However, the contralateral negativity for the Compatible trials was larger than the contralateral positivity for the Incompatible trials, and this gives us an overall significant interaction. But this interaction is meaningless at best and misleading at worst, because the patterns were opposite for the Compatible and Incompatible trials, as indicated by the significant three-way interaction between Hemisphere, Hand, and Compatibility. This kind of complication is one of the reasons why many researchers have stopped using this approach and have shifted to the simpler approach described in the next exercise.

Within Subjects Effects

Cases	Sum of Squares	df	Mean Square	F	p
Hemisphere	1.970	1	1.970	0.380	0.541
Residuals	202.060	39	5.181		
Hand	2.414	1	2.414	1.612	0.212
Residuals	58.404	39	1.498		
Compatibility	2.715	1	2.715	0.517	0.477
Residuals	204.961	39	5.255		
Hemisphere ✻ Hand	5.388	1	5.388	8.224	0.007
Residuals	25.552	39	0.655		
Hemisphere ✻ Compatibility	1.349	1	1.349	3.193	0.082
Residuals	16.475	39	0.422		
Hand ✻ Compatibility	0.911	1	0.911	0.289	0.594
Residuals	123.078	39	3.156		
Hemisphere ✻ Hand ✻ Compatibility	53.246	1	53.246	34.623	< .001
Residuals	59.977	39	1.538		

Table 10.1. Inferential statistics from JASP.

The next step in our statistical analysis would be to perform specific contrasts so that we can see, for example, if the Hemisphere x Hand interaction is significant when the Compatible and Incompatible trials are analyzed separately. However, we're not going to take that next step, because this way of analyzing the data has several shortcomings. First, the size of the LRP is captured by the Hemisphere x Hand interaction rather than by a main effect, which makes things difficult to understand. Second, this approach generates a lot of *p* values, which means that the probability that we obtain one or more bogus-but-significant effects is quite high. If you run a three-way ANOVA, you get 7 *p* values (as shown in Table 10.1), and you'll have about a 30% chance of getting at least one bogus-but-significant effect (if the null hypothesis is actually true for all 7 effects). So, it's important to minimize the number of factors in your analyses (see Luck & Gaspelin, 2017 for a detailed discussion of this issue). The next exercise will show you a better approach.

Bogus Effects

When an effect in the data is just a result of random variation and does not reflect a true effect in the population, I like to refer to that effect as *bogus*. And if the effect is statistically significant, I refer to it as a *bogus-but-significant* effect. The technical term for this is a *Type I error*. But that's a dry, abstract, and hard-to-remember way of describing an incorrect conclusion that might be permanently etched into the scientific literature. I prefer the term *bogus*, which means *not genuine* or even *fake*. It's a much more emotionally negative term, as it should be for something that can be a major error.

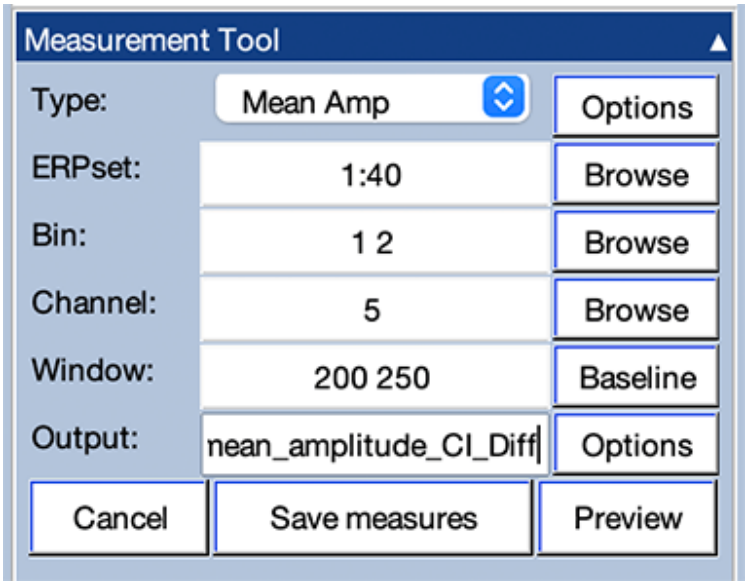
This page titled 10.4: Exercise- A First Pass at Scoring and Statistical Analysis is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 10.4: Exercise- A First Pass at Scoring and Statistical Analysis by Steven J Luck is licensed CC BY 4.0.

10.5: Exercise- Simplifying the Statistical Analysis

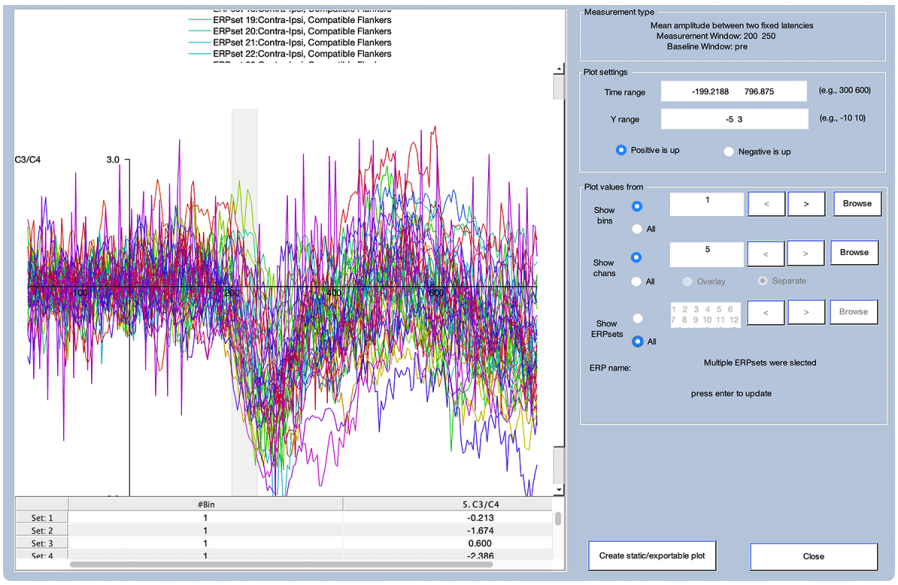
Most recent studies of the LRP (and other lateralized components, such as N2pc and CDA) obtain amplitude and latency scores from contralateral-minus-ipsilateral difference waves (like those shown in Screenshot 10.4). This has two advantages. First, it isolates the component of interest from all of the overlapping ERP components. Second, it reduces the number of factors in the statistical analysis.

Let's see how this works. Click the **Clear All** button in the **ERPsets** panel to clear all the ERPsets from memory, and then load the 40 ERPsets in the **Chapter_10 > Data > ERPsets_CI_Diff** folder. These ERPsets have been processed into contralateral-ipsilateral difference waves, one for compatible trials (bin 1) and one for incompatible trials (bin 2). Now go to the **Measurement Tool** panel, and set it up as shown in Screenshot 10.8. We're measuring from bins **1** and **2** (Compatible and Incompatible) in channel **5** (C3/C4), with **mean_amplitude_CI_Diff.txt** as the name of the output file.



Screenshot 10.8

Click **Preview** and make sure that the measurements look reasonable given the waveforms. You'll see that we have only two bins and one channel, and you can overlay the waveforms for all 40 participants (see Screenshot 10.9).



Screenshot 10.9

Once you've finished looking at the waveforms, click **Close** to go back to the **Measurement Tool** panel, and then click **Save measures** to save the scores. If you open **mean_amplitude_CI_Diff.txt**, you'll again see that we now have only two scores per participant, one for the Compatible condition and one for the Incompatible condition.

Now let's do a statistical analysis on these scores. Start by doing a paired t test comparing the Compatible and Incompatible conditions. You should get a statistically significant *t* value of -5.885 (or +5.885, depending on which condition came first). And here's something very important: This *t* test is exactly equivalent to the three-way interaction between Hemisphere, Hand, and Compatibility in the three-way ANOVA from the previous exercise. That is, it tests exactly the same null hypothesis, and it yields exactly the same *p* value (except for possible rounding error). The *t* value for this paired *t* test corresponds exactly to the *F* value from the three-way interaction once you realize that *F* is the same as t^2. If we square 5.885, we get 34.63, which is the same (except for rounding error) as the *F* value for the three-way interaction in Table 10.1. So, a *t* test on difference scores can be a simpler and more convenient way of testing for an interaction, and it doesn't cause a proliferation of *p* values.

Fixing a Problem

When I first loaded the data from **mean_amplitude_CI_Diff.txt** into JASP, all the scores ended up in a single column. After spending a few minutes trying to figure out what was causing this, I realized that the problem was that the column labels in the first row had commas in them, which confused JASP. When I replaced the commas with underscores, everything worked fine.

Now let's ask whether we have a statistically significant negativity for the Compatible condition and a statistically significant positivity for the Incompatible condition. This just involves performing separate one-sample *t* tests for each of these conditions (comparing the means to zero). When I did that, I obtained a significant negativity for the Compatible condition ($t(39) = -6.107$, $p < .001$) and a significant positivity for the Incompatible condition ($t(39) = 3.605$, $p < .001$).

Isn't this approach a lot simpler and more direct than a three-way ANOVA followed by a bunch of contrasts? We really have three primary hypotheses: 1) that the compatible and incompatible conditions are different; 2) that the contra-minus-ipsi voltage is negative for the compatible condition in this time range; and 3) that the contra-minus-ipsi voltage is positive for the incompatible condition in this time range. With this approach, we have one simple test for each hypothesis.

This page titled 10.5: Exercise- Simplifying the Statistical Analysis is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 10.5: Exercise- Simplifying the Statistical Analysis by Steven J Luck is licensed CC BY 4.0.

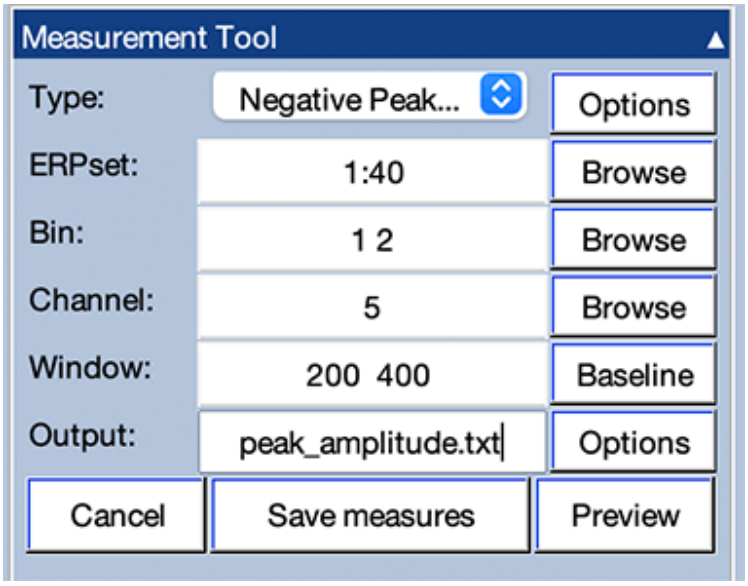
10.6: Exercise- Peak Amplitude

For the first couple decades of ERP research, the primary way of scoring ERP amplitudes was to find the peak voltage during the measurement window (either the most positive voltage for a positive peak or the most negative voltage for a negative peak). This approach was used initially because ERPs were processed using primitive computers that created a printout of the waveform, and researchers could easily determine the peak amplitude from the printout using a ruler (see Donchin & Heffley, 1978). This tradition persisted long after more sophisticated computers and software were available, but in many ways the peak voltage is a terrible way of scoring the amplitude of an ERP component. Mean amplitude is almost always superior. I provide a long list of the shortcomings of peak amplitude and the benefits of mean amplitude in Chapter 9 of Luck (2014). Here's one: what's so special about the point when the voltage reaches a maximum? Mean amplitude is now much more common than peak amplitude in most ERP research areas, but there are some areas where peak amplitude is still common.

In this exercise, we'll repeat the analyses from the previous exercise except that we'll measure peak amplitude instead of mean amplitude. And then you'll see for yourself some of the shortcomings of peak amplitude.

Make sure that the 40 ERPsets from the previous exercise are still loaded (from the **Chapter_10 > Data > ERPsets_CI_Diff** folder). Go to the **Measurement Tool** panel, and set it up as you had it for the previous exercise (Screenshot 10.8), except change the measurement **Type** to **Negative Peak Amplitude**. This algorithm will find the most negative point in the measurement window (200-250). This brings up an immediate problem: The contralateral-minus-ipsilateral voltage is negative in this time window for the Compatible condition, but it's positive for the Incompatible condition (at least in the grand averages). It would be completely invalid to search for a negative peak in one condition and a positive peak in the other, and then compare the two peak amplitudes, because they will inherently be more negative when we search for a negative peak than when we search for a positive peak. This means that it would be very difficult to use peak amplitude to capture the fact that the voltage was negative for the Compatible condition and positive for the Incompatible condition.

So, let's try a simpler analysis. The LRP eventually becomes negative for both the Compatible and Incompatible conditions (at least in the grand averages), so let's ask whether the negative peak over a broad measurement window differs in amplitude between these two conditions. The negative peaks in the grand average were at approximately 275 and 350 ms in the two conditions, so let's use a measurement window of 200-400 ms. To do this, change the **Window** in the **Measurement Tool** panel to **200 400** and set the **Output** filename to **peak_amplitude.txt** (see Screenshot 10.10).

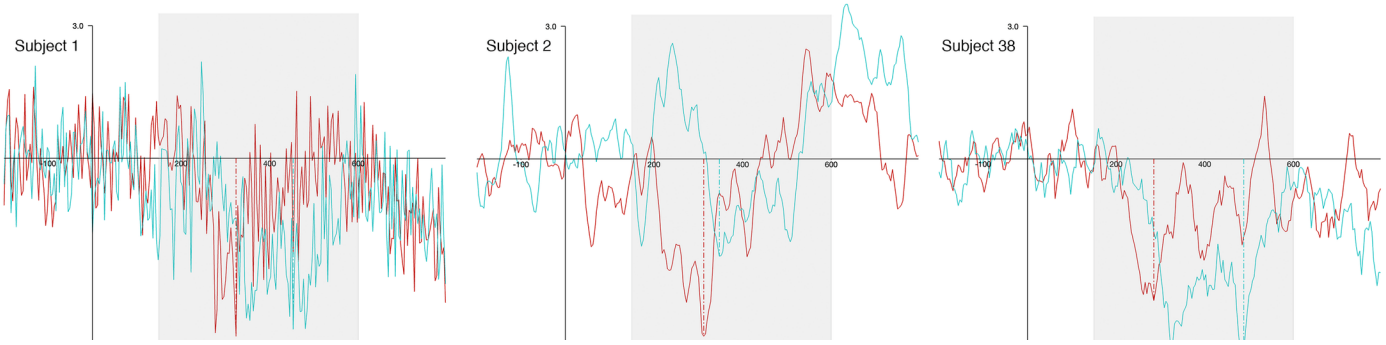


Screenshot 10.10

Now click the **Preview** button, and then set the **Preview** tool to show **All** bins. You should see something like the left side of Screenshot 10.11. This subject has a ton of high-frequency noise, and the peaks are driven mainly by this noise. This is one of the major shortcomings of peak amplitude: It is highly distorted by high-frequency noise. There's no reason to believe that the

amplitudes of the peaks in this waveform are a good reflection of the magnitude of this participant's LRP.

Another issue is that there is a peak for bin 2 that falls outside the measurement window (outside the gray region). If you start scrolling through the ERPsets, you'll see that this is an issue in many cases (e.g., Subjects 3 and 5).



Screenshot 10.11

We can fix this by specifying a broader window. Click **Close** in the **Preview** window to go back to the **Measurement Tool** panel, change the **Window** to **150 600**, and click **Preview** again. Set it to show **All** bins again, and then start scrolling through the ERPsets. For each ERPset, you should see a vertical dashed line going from the baseline to the peak of each waveform. You can see this in the center of Screenshot 10.12, which shows the data from Subject 2. I wish all participants had waveforms as nice as these. But if you scroll through the participants, you'll see that several of them have noisy or odd-looking waveforms. And keep in mind that the participants in this study were highly cooperative college students, and we used a very high-quality recording system. So, the poor quality of some of the waveforms is just something we have to live with in most ERP research.

Subject 38 (Screenshot 10.11, right) shows a way in which peak amplitude can be very problematic. You can see that the peak voltage is measured at approximately 275 ms for the Compatible condition but at nearly 500 ms in the Incompatible condition. What are the chances that we are measuring the same processes at 275 and 500 ms? Not very high.

Once you've scrolled through all the data, click **Close** in the **Preview** window to go back to the **Measurement Tool** panel, and then click Save measures to save the scores to the **peak_amplitude.txt** file. Now you can do a paired *t* test to see if the peak amplitudes for Compatible and Incompatible trials are significantly different. When I did this in JASP, I found that this difference was not significant ($t(39) = -0.81$, $p = .936$). That's not a surprise given that the LRP peak amplitudes looked pretty similar for the two conditions in the grand average (see Screenshot 10.4 in Section 10.3).

Now you can do one-sample *t* tests separately for the Compatible and Incompatible trials to see if the LRP is significantly different from zero. When I did this, I found that the amplitudes were significantly different from zero for both the Compatible trials ($t(39) = -19.15$, $p < .001$) and the Incompatible Trials ($t(39) = -18.27$, $p < .001$). But wait! These one-sample *t* tests are completely invalid, because chance is not zero for a peak amplitude measure. If you had pure noise during the measurement window, and you looked for the largest negative peak, you would virtually always find a value that was less than zero. In other words, peak amplitude is a *biased* scoring method. To make things even worse, the bias increases as the noise level increases: The noisier the data, the larger (on average) is the peak voltage. By contrast, mean amplitude is unbiased: It is equally likely to be larger or smaller than the true value, regardless of the noise level. This issue typically arises when you are comparing groups or conditions that differ in the number of trials (e.g., the rare and frequent conditions in an oddball experiment), because the noise level is typically greater when the number of trials is smaller. It's not valid to compare peak amplitudes in cases like this, but it's perfectly fine to compare mean amplitudes.

To see the bias in peak amplitude, let's repeat the analysis you did, but with a measurement window of **-100 0** (i.e., the last 100 ms of the prestimulus baseline period). There shouldn't be any real differences prior to the onset of the stimuli, and any differences we see must be a result of noise. When I did this, I found that there was still no significant difference between Compatible and Incompatible trials. However, when I did one-sample *t* tests against zero for each condition, I found that the amplitudes were still significantly different from zero for both the Compatible trials ($t(39) = -12.50$, $p < .001$) and the Incompatible Trials ($t(39) = -10.44$, $p < .001$).

Now try this again, but changing the measurement **Type** to **Mean Amp** (but leaving the measurement window at **-100 0**). You should now find that the mean across participants is very close to zero for both conditions, and not significantly different from zero for either Compatible trials ($t(39) = 0.896$, $p = .376$) or Incompatible Trials ($t(39) = 0.804$, $p = .426$). This shows you that

mean amplitude is an unbiased scoring method (see additional explanation in Chapter 9 of Luck, 2014).

The bottom line is that the peak voltage is not usually the best way to quantify the amplitude of an ERP component. Mean amplitude is much better in the vast majority of cases.

This page titled 10.6: Exercise- Peak Amplitude is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

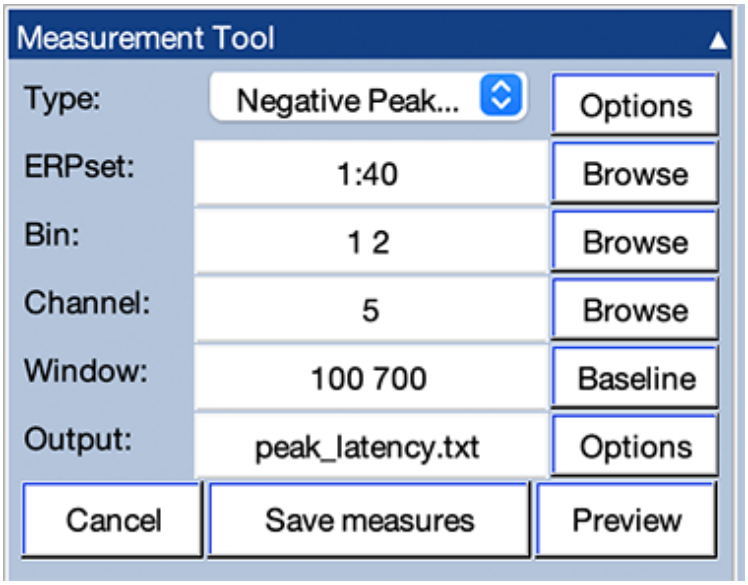
- 10.6: Exercise- Peak Amplitude by Steven J Luck is licensed CC BY 4.0.

10.7: Exercise- Peak Latency

Now we're going to switch from scoring amplitudes to scoring latencies. The traditional method for scoring latencies is to find the peak voltage in the measurement window (positive or negative) and record the latency at which this peak occurred. Like peak amplitude, peak latency is not usually the best scoring algorithm, but it's often a reasonable approach (see Chapter 9 in Luck, 2014). We'll look at some better alternatives in the following exercises.

In the present exercise, we're going to ask whether the peak latency of the LRP in the contralateral-minus-ipsilateral difference wave is later on Incompatible trials than on Compatible trials. Make sure that the 40 ERPsets from the previous exercise (from the **Chapter_10 > Data > ERPsets_CI_Diff** folder) are loaded in the **ERPsets** panel. Go to the **Measurement Tool** panel and set it up as shown in Screenshot 10.12. The measurement **Type** is **Negative Peak Lat**, and the **Window** is **100 700**. We're measuring from Bins 1 and 2 (Compatible and Incompatible) in Channel 5 (C3/C4), and we're saving the scores in an **Output** file named **peak_latency.txt**.

A key question in scoring ERP amplitudes and latencies is how to determine the time window. This is a complicated question, and you can read about several strategies in Chapter 9 of Luck (2014) and in Luck and Gaspelin (2017). As mentioned earlier, the most important thing is to avoid being biased by the data, which is best achieved by deciding on the measurement windows before you start the study. Of course, it's too late for that now with the ERP CORE experiments. However, if I were to choose a time window in advance for the LRP in a flankers paradigm, I'd assume that the LRP peak starts after 100 ms and ends before 700 ms. For this reason, we're using a measurement window of 150 to 600 ms in this exercise.



Screenshot 10.12

As always, the next step is to click the **Preview** button to see how well the algorithm is working. If you scroll through all 40 participants, you'll see that it has mixed success. It works reasonably well for waveforms that are clean and contain a large peak (e.g., Subjects 2 and 4), but the scores are distorted by high-frequency noise (e.g., Subjects 1 and 12), and the values are largely random for waveforms without a single distinct peak (e.g., Subjects 9, 10, and 38).

Now click **Close** in the **Preview** window to go back to the **Measurement Tool** panel, and click **Save measures** to save the scores to the file named **peak_latency.txt**. Load the data into your statistics package and perform a paired *t* test to compare the Compatible and Incompatible conditions. Verify that the means provided by the statistics package are reasonable. You should see a mean of 318.9 ms for Compatible and 375.6 ms for Incompatible. Unfortunately, the trick we used to verify that these means are correct when we analyzed mean amplitude—comparing the means from the statistical package with the values measured from the grand average—doesn't work with peak latency. If you measure the peak latency directly from the grand average ERP waveforms, you'll see a value of 285 ms for Compatible and 355 ms for Incompatible. The values from the grand average aren't the same as the mean of the single-subject values, but at least they show the same ordering (Compatible < Incompatible).

Now look at the actual *t* test results. You should see that the peak latency was significantly shorter for Compatible trials than for Incompatible trials ($t(39) = -3.647$, $p < .001$). Given the huge differences between Compatible and Incompatible trials in the grand average waveforms (Figure 10.2.C), it's not surprising that the difference in peak latency was significant, even if peak latency isn't an ideal scoring metric. You should also look at the effect size, measured as Cohen's d_z, which indicates how far apart the means are relative to the pooled standard deviation. You should see an effect size of -0.577 (or +0.577 depending on the order of conditions in your analysis), which is a medium effect size.

If you're familiar with effect sizes in ERP studies, you might be surprised that this effect size isn't bigger. After all, the peaks in the grand averages are very far apart in time. It therefore seems reasonable to suppose that we had a lot of measurement error when we computed the peak latency, which increased the standard deviation of the scores and therefore reduced the effect size. Given that peak latency scores are distorted by high-frequency noise, we should be able to reduce the measurement error and increase the effect size by applying a low-pass filter to the averaged ERPs prior to obtaining the peak latency scores.

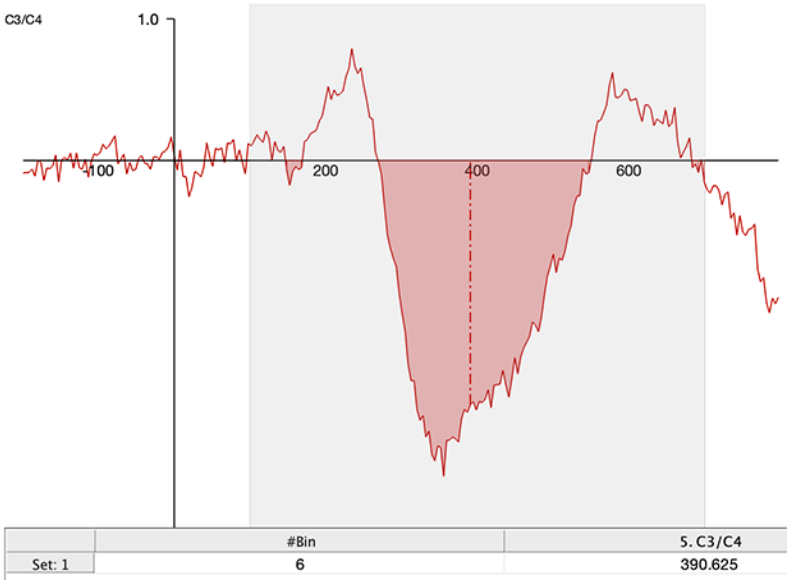
Let's try it. It would take quite a while for you to filter all 40 of the ERPsets using the GUI, so I've provided the ERPsets for you in **Chapter_10 > Data > ERPsets_CI_Diff_filt**. They've been low-pass filtered with a half-amplitude cutoff of 20 Hz and a slope of 12 dB/octave. Clear out the existing ERPsets from ERPLAB, load the filtered ERPsets, and repeat the measurement and analysis procedure (but change the name of the measurement file to **peak_latency_filt.txt**). You'll see that the effect size is now a little larger ($d = -0.630$). So, filtering helped, but only a little. Sometimes it helps a lot, especially when there is a lot of high-frequency noise in the data (which is not true for most of the waveforms in this experiment).

This page titled 10.7: Exercise- Peak Latency is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 10.7: Exercise- Peak Latency by Steven J Luck is licensed CC BY 4.0.

10.8: Exercise- Fractional Area Latency

In this exercise, we're going to look at a different latency scoring algorithm called *fractional area latency*, which is often superior to peak latency (especially when measured from difference waves). For a negative component like the LRP, this algorithm calculates the area of the waveform below the zero line and then finds the time point that divides the area into two areas at a particular percentage. If you want to estimate the *midpoint* of the waveform, you will look for the 50% point (the time that divides the area into two equal halves). This is then called the *50% area latency* (see Chapter 9 in Luck, 2014, for more details). Screenshot 10.13 shows what it looks like when I apply this algorithm to the grand average contra-minus-ipsi difference wave for the Incompatible trials, using a measurement window of 100 to 700 ms. The area under the curve in this measurement window is shaded in red, and the point that divides this area into two equal halves is indicated by the red vertical line. This region includes some little areas near the beginning and end of the measurement window, but that's just how it goes. It's difficult to perfectly quantify ERP amplitudes and latencies, and we have to live with some error.

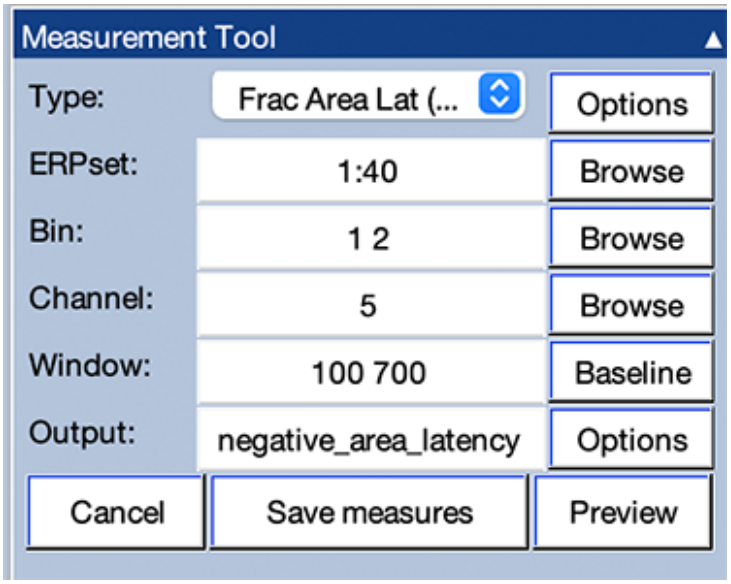


Screenshot 10.13

Don't Worry About High-Frequency Noise in Area-Based Measures

Area-based measures like fractional area latency are relatively insensitive to high-frequency noise, so we will apply this method to the unfiltered data. It's also usually unnecessary to filter out high-frequency noise when measuring mean amplitude.

Let's apply this scoring algorithm to the single-participant waveforms. If the filtered ERPsets from the previous exercise are still loaded in ERPLAB, clear them. Then load the 40 unfiltered difference wave ERPsets (from the **Chapter_10 > Data > ERPsets_CI_Diff** folder). Go to the **Measurement Tool** panel, and set it up as shown in Screenshot 10.14. The measurement **Type** is **Frac Area Lat (Neg Area)**, indicating that we're looking for a fractional area latency for a negative area. We're again measuring from Bins 1 and 2 (Compatible and Incompatible) in Channel 5 (C3/C4), and we're saving the scores in a file named **negative_area_latency.txt**. Click on the **Options** button next to the measurement type, and make sure it's set to **50% of area**.



Screenshot 10.14

Using the **Preview** tool, make sure that the scoring algorithm is working properly. Then click **Close** in the **Preview** window to go back to the **Measurement Tool** panel and click **Save measures** to save the measurements. Load the data into your statistics package and do a paired *t* test, as in the previous exercise. You'll see that the mean latency is ~45 ms shorter for the Compatible condition than for the Incompatible condition, which is actually a somewhat smaller difference than we saw for peak latency (a 57 ms difference). However, the Cohen's *d* value has increased substantially, from -0.577 for peak latency to -0.823 for the 50% area latency measure. And if you look at the descriptive statistics, you'll see that the standard deviations are now quite a bit lower. So, we now have a large effect size instead of a medium effect size, due to reduced variability (presumably owing to reduced measurement error).

The increased effect size we're seeing for 50% area latency relative to peak latency is consistent with what I've seen in many previous experiments. This is one of the reasons I recommend using 50% area latency, especially when the measurements are being obtained from difference waves.

When to Use Fractional Area Latency

The fractional area latency algorithm works well only if the waveform is dominated by a single component. This is usually true of difference waves, which are designed to isolate a single component. The 50% area latency measure also works well on parent waves when the component of interest is so large that it dominates everything else (e.g., the N400 for semantically deviant words or the P3b for rare targets).

A more direct way to compare measurement error for these two different scoring algorithms would be to look at the standardized measurement error (SME). Unfortunately, it's complicated to compute the SME for anything other than mean amplitudes. When some other scoring algorithm is used, or when the measurements are obtained from difference waves, a method called *bootstrapping* is necessary for calculating the SME. Currently, this can't be done from the ERPLAB GUI and instead requires scripting. I've provided a script for this at the end of the chapter. The script demonstrates that the SME was, in fact, much better (lower) for the 50% area latency measure than for the peak latency measure in this experiment.

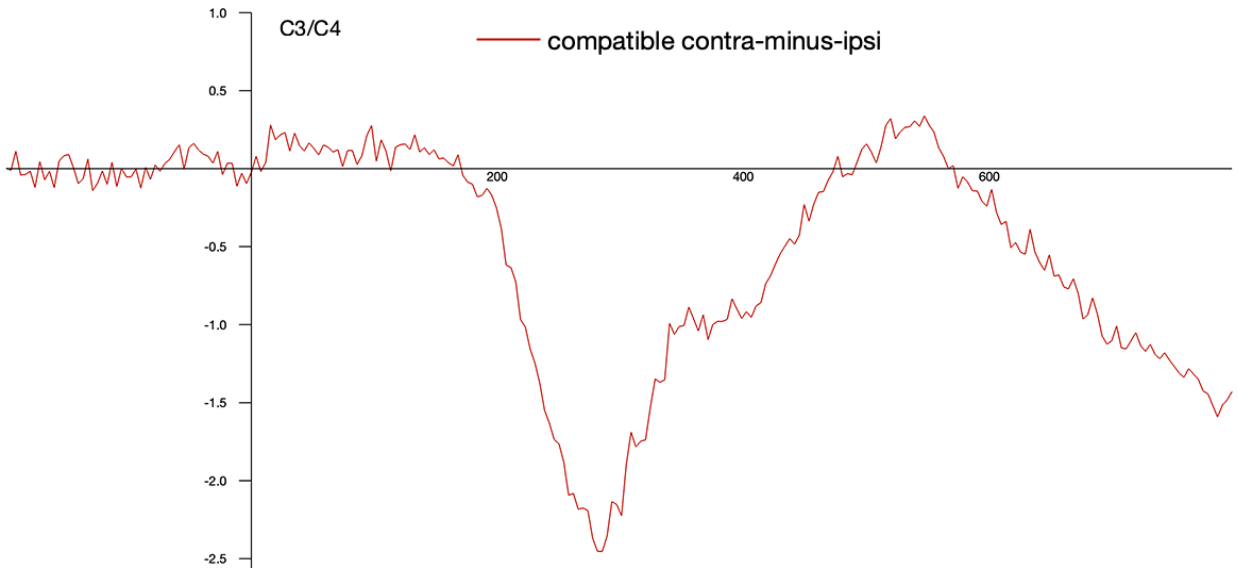
This page titled 10.8: Exercise- Fractional Area Latency is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 10.8: Exercise- Fractional Area Latency by Steven J Luck is licensed CC BY 4.0.

10.9: Exercise- Quantifying Onset Latency

The onset time of a difference between two conditions can be extremely informative. For example, the brain can't have a more negative response over the contralateral hemisphere than over the ipsilateral hemisphere until it has determined which hand should respond, so the onset latency of the LRP can be used as a marker of the time at which the brain has decided on a response hand (see Chapters 2 and 3 in Luck, 2014 [or in the newer 3rd edition] for a more detailed and nuanced discussion).

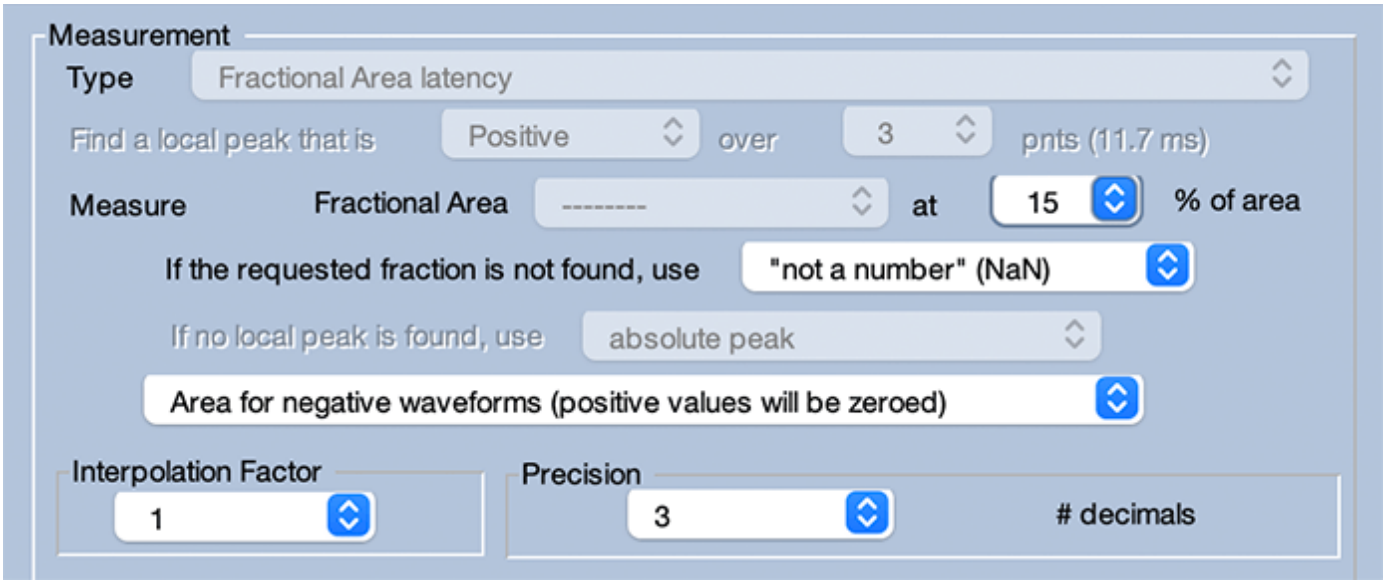
How can we quantify the onset time of a difference wave? Consider, for example, the Compatible waveform in Screenshot 10.15. The negativity of the LRP first falls below the zero line at 172 ms. However, the negativity between 172 and 200 ms is no larger than the noise level (as assessed, e.g., by the variations in voltage during the prestimulus period). The voltage at 172 ms may be negative simply because of noise, so we wouldn't want to consider 172 ms to be the onset of the LRP.



Screenshot 10.15

Early research attempted to solve this problem by using a statistical criterion such as the first of N consecutive points that are at least 2 standard deviations greater than the noise level (where the standard deviation is measured from the variation in voltage during the prestimulus period). However, this approach suffers from low power, and single-participant scores will vary according to the noise level as well as the true onset time. A terrific study by Kiesel et al. (2008) rigorously compared this technique with peak latency and two other measures that were not very widely used at the time—fractional area latency and fractional peak latency—and found that the two less widely used scoring methods were actually the best. These two methods are now more commonly used, and we'll focus on them here.

We already looked at fractional area latency in the previous exercise, but we used it to estimate the midpoint latency (the 50% area latency) rather than the onset latency. To estimate the onset latency, we simply need to use a lower percentage. In the present exercise, we'll calculate the time at which the area reaches the 15% point. To get started, make sure that the 40 ERPsets from the **Chapter_10 > Data > ERPsets_CI_Diff** folder are loaded. Go to the **Measurement Tool** panel, and set it up as in the previous exercise (Screenshot 10.14), except change the name of the output file to something like **FAL15_latency.txt**. Then click the **Options** button for the measurement type and change the **Fractional Area** percentage to **15% of area**, as shown in Screenshot 10.16. Click **OK** to go back to the Measurement Tool panel, and then take a look at the scores for the individual waveforms using the **Preview** tool. Finally, save the scores.



Screenshot 10.16

As before, load the resulting scores into your statistical package and compute the paired *t* test to compare the Compatible and Incompatible conditions. You should see that the difference in means across conditions is ~50 ms and that the effect is statistically significant ($t(39) = -5.964$, $p < .001$) with a very large effect size ($d = -0.943$).

Now let's try the other scoring algorithm, fractional peak latency, which is illustrated in Figure 10.3. This method finds the peak and then moves backward in time until the voltage reaches some fraction of the peak voltage (usually the 50% point). The latency of this point is then used as the estimate of onset latency. You might wonder why we usually choose the 50% point. Isn't the 15% point, for example, closer to the true onset? There are two reasons to choose the 50% point. First, it's less influenced by noise and therefore more reliable than lower percentages. Second, it actually does a better job of capturing the average onset time given that there is almost always significant trial-to-trial variation in onset times. As discussed in Chapter 9 of Luck (2014), the 50% peak latency point accurately captures the average of the single-trial onset times under some conditions.

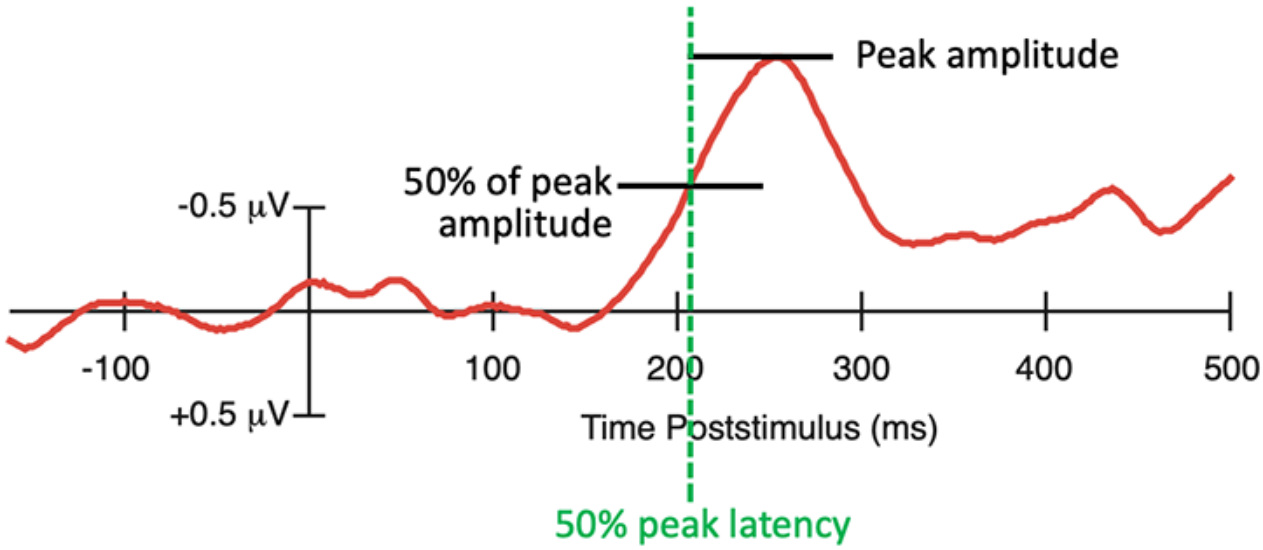
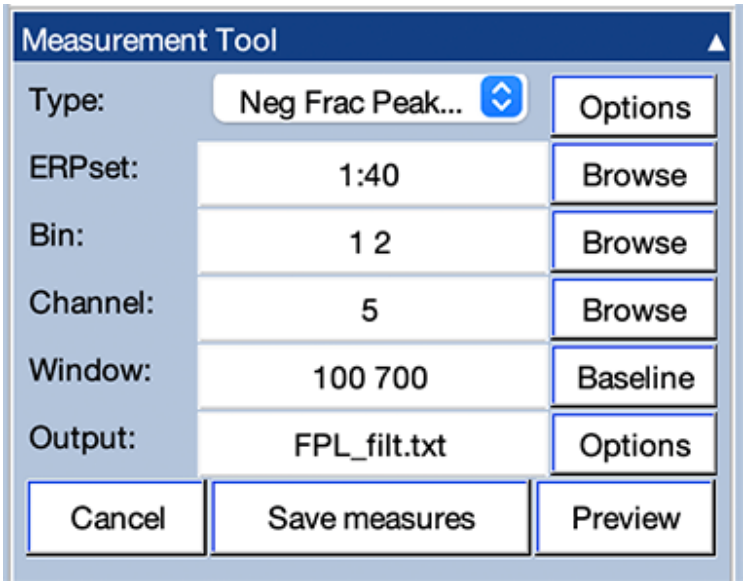


Figure 10.3. Example of the fractional peak latency method. In this example, we obtained the 50% peak latency (the latency at which the voltage reached 50% of the peak voltage).

Let's give it a try. First, clear the existing ERPsets out of ERPLAB and load the filtered ERPsets in the **Chapter_10 > Data > ERPsets_CI_Diff_filt** folder. This scoring method is, unfortunately, very sensitive to high-frequency noise, so we ordinarily apply fairly aggressive low-pass filtering (which, fortunately, has relatively little impact on the 50% peak latency). Go to the

Measurement Tool panel, and set it up as shown in Screenshot 10.17, using **FPL_filt.txt** as the **Output** file. Then click the **Options** button for the measurement **Type**, and set the **Fractional Peak (pre-peak onset)** percentage to **50% of peak**. Click OK to go back to the panel, and use the Preview tool to make sure that the scores look appropriate for the single-participant waveforms. Then save the scores to **FPL_filt.txt**.



Screenshot 10.17

Load the resulting scores into your statistical package and compute the paired *t* test. You should see that the difference in means across conditions is ~60 ms (a little larger than observed for the 15% area latency measure from the previous exercise). The effect is statistically significant ($t(39) = -4.376$, $p < .001$), but the effect size is smaller than observed in the previous exercise ($d = -0.692$ for 50% peak latency versus $d = -0.943$ for 15% area latency).

So, which of these two scoring methods is best? The effect size was larger for 15% area latency than for 50% peak latency in the analysis you just did. Also, Kiesel et al (2008) found that 50% area latency yielded less variability than 50% peak latency. Unfortunately, they didn't examine 15% area latency, and they didn't apply an aggressive low-pass filter prior to obtaining the 50% peak latency scores. We also found lower standard deviations for 50% area latency than for 50% peak latency for all of the basic difference waves in the six ERP CORE paradigms (see Table 3 in Kappenman et al., 2021). However, the 50% area latency captures the midpoint of the difference wave, not the onset, which is less sensitive to noise, so this really isn't a fair comparison. I think it's fair to say that this issue is unresolved at this point.

However, there is an important conceptual difference between these two scoring methods. Specifically, fractional area latency is impacted by voltages throughout the entire measurement window. For example, the negative voltage late in the waveform for the Incompatible trials (see Figure 10.2.C) will have an impact on the 15% fractional area latency score. By contrast, fractional peak latency is not influenced by anything that happens after the peak. This is particularly clear in the example shown in Figure 10.3, where there is a long "tail" to the difference wave that will have a large impact on the fractional area latency score but will have no impact on the fractional peak latency score. For this reason, I usually use the fractional peak latency score.

This exemplifies a broader issue: Although minimizing measurement error is important, it's also important to make sure that your scoring method is *valid* (i.e., measures what you are trying to measure with minimal influence from other factors).

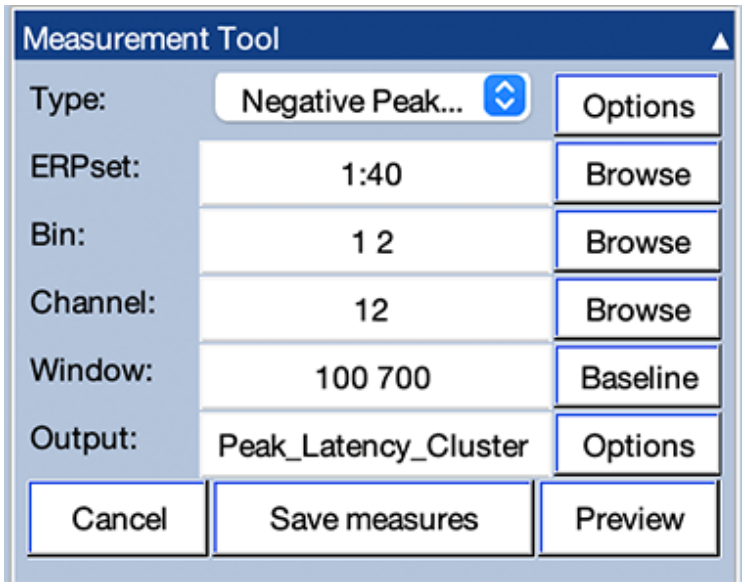
This page titled 10.9: Exercise- Quantifying Onset Latency is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 10.9: Exercise- Quantifying Onset Latency by Steven J Luck is licensed CC BY 4.0.
- 10.6: Exercise- Peak Amplitude by Steven J Luck is licensed CC BY 4.0.

10.10: Exercise- Collapsing Across Channels and Correlating Latencies with Response Times

So far, we've been measuring and analyzing the data only from the C3 and C4 channels, where the LRP is largest. However, it's often valuable to include the data from multiple channels. One way to do that is to measure from multiple different channels and include channel as a factor in the statistical analysis. However, that adds another factor to the analysis, which increases the number of *p* values and therefore increases the probability of getting bogus-but-significant effects. Also, interactions between an experimental manipulation and electrode site are difficult to interpret (Urbach & Kutas, 2002, 2006). In most cases, I therefore recommend averaging the waveforms across the channels where the component is large, creating a *cluster*, and then obtaining the amplitude or latency scores from the cluster waveform. Averaging across channels in this manner tends to produce a cleaner waveform, which decreases the measurement error (as long as we don't include channels where the ERP effect is substantially smaller). Also, it avoids the temptation to "cherry-pick" the channel with the largest effect. A very careful analysis of the ERP CORE by Zhang and Kappenman (2024) provides quantitative support for this advice. For the ERP CORE flankers experiment, they found that a cluster including FC3/4, C3/4, and C5/6 yielded the best signal-to-noise ratio. They also found that this cluster yielded the best statistical power when they simulated a comparison of two groups (e.g., a patient group and a control group) that differed in LRP amplitude.

I've already created this cluster in the difference waves in the **Chapter_10** folder. They're in Channel 12, which is labeled **cluster**. Let's try measuring the peak latency from this channel in the filtered data, which you can load from the **Chapter_10 > Data > ERPsets_CI_Diff_filt** folder (if they're not already loaded). You can just repeat the measurement and analysis procedures from the earlier exercise where we measured the peak latency from the filtered data, but changing the **Channel** from **5** to **12** as shown in Screenshot 10.18. As usual, take a look at the measures using the **Preview** tool, and then save them to a file named **Peak_Latency_Cluster.txt**.



Screenshot 10.18

An example of why previewing the measures is important

Many times in this chapter, I've asked you to use the Preview tool to look at the measures along with the actual ERP waveforms. Did you do it every time? Probably not. But you really should. After I wrote the text in the preceding paragraph, I got everything set up in the **Measurement Tool** panel, and then I clicked the **Preview** button to make sure everything looked OK, and I noticed something strange.

> We're now measuring from the new **cluster** channel, and in the **Preview** window I saw that some of the ERP waveforms had a ton of high-frequency noise. I double-checked to make sure that I was in the folder with the filtered ERPsets (**ERPsets_CI_Diff_filt**), and I was. Then I looked at the other channels, and they didn't have the high-frequency noise. It was only in the new **cluster** channel. I then went back and looked at the script that created the ERPsets for this exercise, and I saw that the filtering was applied only to channels 1-11 (the original channels), not channel 12 (the cluster channel). I then fixed the script (which required changing only one number), ran it again, and everything was fine.
>
> This shows the value of previewing the measures along with the ERP waveforms. If I hadn't clicked the **Preview** button, I would have been scoring noisy data and not getting optimal results. So, you should believe me when I tell you to preview the measures with the waveforms every time.

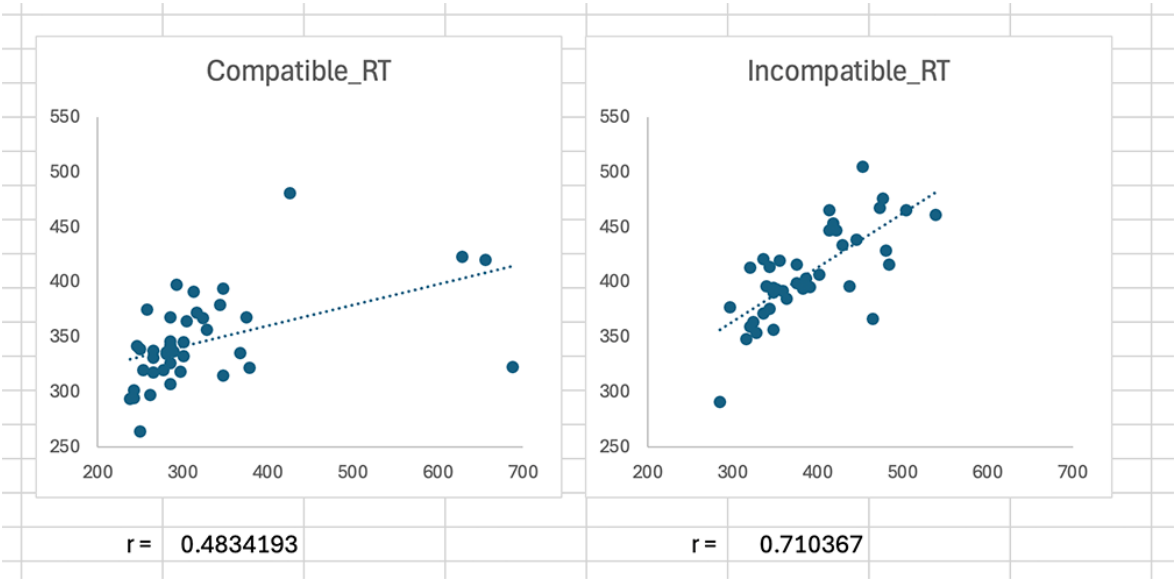
Now that you have the peak latency scores, you can perform the usual paired *t* test. You should see that the effect size is about the same as before (-0.613 for the cluster analysis relative to -0.630 when we measured from the C3/C4 channel). Collapsing across channels doesn't always increase the effect size, but it doesn't usually hurt, and it avoids the need to include channel as a factor in the analysis or the need to select one channel to use. My lab now measures from a cluster in virtually all of our studies.

Our last step will be to ask whether the peak latency values are correlated with response times (RTs). That is, do participants with later LRP peaks also have slower RTs? Unfortunately, it takes some additional work to extract RTs using ERPLAB. The ERPLAB documentation has a whole section devoted to behavioral analysis, and it's not super difficult, but it's too much for me to explain here. For the present exercise, I wrote an ERPLAB script named **LRP_RTs.m** to obtain the mean RTs for Compatible and Incompatible trials for each participant. You can look at the script (in the **Chapter_10** folder) to see how I extracted the RTs from the EEGsets.

This script saves mean RTs for each participant in each condition to an Excel spreadsheet named **RT.xlsx**. These mean RTs were calculated after excluding trials that were rejected because of artifacts, so the ERP data and the behavioral data are based on exactly the same set of trials.

Let's look at the correlations. You'll need to combine the RT values from the spreadsheet with the peak latency values that you just created from the waveforms that were collapsed across the three pairs of electrode sites. And then you'll need to read them into your statistical package or into a spreadsheet program. I turned them into an Excel spreadsheet, with scatterplots and correlations (see **Peak_Latency_Cluster_with_RTs.xlsx** in the **Chapter_10** folder). The spreadsheet computes the correlation between peak latency and RT, separately for the Compatible condition and the Incompatible condition.

Screenshot 10.19 is a screenshot from the Excel file. The peak latency scores were correlated fairly well with the RTs, especially for the incompatible trials. Some outliers in the LRP peak latency scores for the Compatible trials clearly reduced the correlation in that condition. Nonetheless, these correlations show that the timing of the brain activity is related to the timing of the behavioral response.



Screenshot 10.19

This page titled 10.10: Exercise- Collapsing Across Channels and Correlating Latencies with Response Times is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 10.10: Exercise- Collapsing Across Channels and Correlating Latencies with Response Times by Steven J Luck is licensed CC BY 4.0.

10.11: Matlab Scripts For This Chapter

I've created three scripts for this chapter. The first script is named **LRP_RTs.m**, and it shows you how to get the single-trial RTs for the trials without artifacts flagged, compute the mean RT for each condition, and save the results to an Excel spreadsheet.

The second script, **LRP_scoring.m**, shows you how to obtain several of the amplitude and latency scores described in this chapter.

The third script, **LRP_bSME.m**, demonstrates how to get the SME values for several of the scores. For scores other than mean amplitude, bootstrapping is required (Luck et al., 2021), and we call the result the *bootstrapped SME* or *bSME*. The script demonstrates how to implement the bootstrapping procedure and compute the bSME values. Bootstrapping requires re-averaging the data for a given participant many times, and it can be slow. The script is set to do only 100 iterations per participant so that it runs reasonably quickly. There is a variable you can change to a larger value (e.g., 10,000) to get more robust SME estimates.

This page titled 10.11: Matlab Scripts For This Chapter is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 10.11: Matlab Scripts For This Chapter by Steven J Luck is licensed CC BY 4.0.

10.12: Key Takeaways and References

Key Takeaways

- Many methods exist to quantify the timing or magnitude of an ERP component or experimental effect. The traditional methods—peak amplitude and peak latency—have many shortcomings, and better methods are available.
- In many cases, it is advantageous to obtain scores from a difference wave that isolates the component or experimental effect of interest.
- You should always visually verify that the scores are being calculated appropriately for each individual ERP waveform.
- Most scoring methods require specifying a measurement window, and this needs to be done in an unbiased manner. If you decide on the measurement window after seeing the waveforms, you may consciously or unconsciously choose a window that increases the probability of bogus-but-significant effects.
- When you have many conditions and/or channels, it's easy to accidentally put the cells of the design into the wrong order in the statistical analysis. You should always check the table of means produced by your statistical package and make sure it matches what you are seeing in the grand average ERP waveforms.
    - For mean amplitude, but not most other scoring methods, taking the score from the grand average waveforms gives you the same result as measuring from the single-participant waveforms and then averaging. This makes it easy to compare the table of means with the grand averages.
    - For other measures, you can still make sure that the table of means shows the same pattern as the grand averages, even if the individual values are not identical.
- To reduce the number of *p* values and the likelihood of bogus-but-significant effects, you should use the smallest possible number of factors in your statistical analyses. This can often be achieved by collapsing across channels and obtaining scores from difference waves.

References

Donchin, E., & Heffley, E. F. (1978). Multivariate analysis of event-related potential data: A tutorial review. In D. Otto (Ed.), *Multidisciplinary Perspectives in Event-Related Brain Potential Research* (pp. 555–572). U.S. Government Printing Office.

Eriksen, C. W. (1995). The flankers task and response competition: A useful tool for investigating a variety of cognitive problems. *Visual Cognition*, *2*, 101–118.

Gehring, W. J., Liu, Y., Orr, J. M., & Carp, J. (2012). The error-related negativity (ERN/Ne). In S. J. Luck & E. S. Kappenman (Eds.), *The Oxford Handbook of Event-Related Potential Components* (pp. 231–292). Oxford University Press.

Gratton, G., Coles, M. G. H., Sirevaag, E. J., Eriksen, C. W., & Donchin, E. (1988). Pre- and post-stimulus activation of response channels: A psychophysiological analysis. *Journal of Experimental Psychology: Human Perception and Performance*, *14*, 331–344.

Kappenman, E. S., Farrens, J. L., Zhang, W., Stewart, A. X., & Luck, S. J. (2021). ERP CORE: An Open Resource for Human Event-Related Potential Research. *NeuroImage*, *225*, 117465. https://doi.org/10.1016/j.neuroimage.2020.117465

Kiesel, A., Miller, J., Jolicoeur, P., & Brisson, B. (2008). Measurement of ERP latency differences: A comparison of single-participant and jackknife-based scoring methods. *Psychophysiology*, *45*, 250–274. https://doi.org/10.1111/j.1469-8986.2007.00618.x

Love, J., Selker, R., Marsman, M., Jamil, T., Dropmann, D., Verhagen, J., Ly, A., Gronau, Q. F., Šmíra, M., Epskamp, S., Matzke, D., Wild, A., Knight, P., Rouder, J. N., Morey, R. D., & Wagenmakers, E.-J. (2019). JASP: Graphical Statistical Software for Common Statistical Designs. *Journal of Statistical Software*, *88*(1), 1–17. https://doi.org/10.18637/jss.v088.i02

Luck, S. J. (2014). *An Introduction to the Event-Related Potential Technique, Second Edition*. MIT Press.

Luck, S. J., & Gaspelin, N. (2017). How to get statistically significant effects in any ERP experiment (and why you shouldn't). *Psychophysiology*, *54*, 146–157. https://doi.org/10.1111/psyp.12639

Luck, S. J., Stewart, A. X., Simmons, A. M., & Rhemtulla, M. (2021). Standardized measurement error: A universal metric of data

quality for averaged event-related potentials. *Psychophysiology*, *58*, e13793. https://doi.org/10.1111/psyp.13793

R Core Team. (2016). *R: A Language and Environment for Statistical Computing*. Vienna, Austria.

Smulders, F. T. Y., & Miller, J. O. (2012). The Lateralized Readiness Potential. In S. J. Luck & E. S. Kappenman (Eds.), *The Oxford Handbook of Event-Related Potential Components* (pp. 209–229). Oxford University Press.

Urbach, T. P., & Kutas, M. (2002). The intractability of scaling scalp distributions to infer neuroelectric sources. *Psychophysiology*, *39*, 791–808. https://doi.org/10.1017/S0048577202010648

Urbach, T. P., & Kutas, M. (2006). Interpreting event-related brain potential (ERP) distributions: Implications of baseline potentials and variability with application to amplitude normalization by vector scaling. *Biological Psychology*, *72*(3), 333–343. https://doi.org/10.1016/j.biopsycho.2005.11.012

Zhang, W., & Kappenman, E. S. (2024). Maximizing signal-to-noise ratio and statistical power in ERP measurement: Single sites versus multi-site average clusters. *Psychophysiology*, *61*, e14440. https://doi.org/10.1111/psyp.14440

This page titled 10.12: Key Takeaways and References is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 10.12: Key Takeaways and References by Steven J Luck is licensed CC BY 4.0.

11: EEGLAB AND ERPLAB SCRIPTING

Learning Objectives

In this chapter, you will learn to:

- Use the Matlab command line to execute EEGLAB and ERPLAB routines.
- Efficiently move back and forth between scripts and the GUI to have the best of both worlds.
- Use the EEG and ERP histories to see the commands that correspond to the EEGLAB and ERPLAB procedures that you run from the GUI.
- Create simple scripts by copying commands from the EEG and ERP histories into a text file.
- Use variables to define paths in a way that avoids problems and makes it easy to move your scripts and data to new locations.
- Use loops so that you can efficiently repeat a set of processing steps on the data from multiple participants.
- Read from and write to spreadsheets and text files to increase the power and flexibility of your scripts.
- Create an entire processing pipeline that begins with the raw EEG and ends with amplitude and latency scores that are ready for statistical analysis.
- Implement good programming practices that will minimize errors and increase the readability of your code.

You can do a lot with the ERPLAB Studio GUI. However, you will eventually grow tired of all the pointing and clicking, especially the seventh time you reanalyze the data from an experiment (and believe me, you will be lucky if it's only seven times). It's straightforward to write Matlab scripts that automate almost every processing step. Scripts can also help you avoid the errors that inevitably arise in the thousands of clicks required to conduct every processing step for every participant. Better yet, scripts can allow you to implement new or modified processing steps, making your research more innovative. Scripts also play an important role in open science: When you publish a paper, you can make your data and scripts available (e.g., using the Open Science Framework), and then the world can see exactly how you processed the data and can exactly reproduce your analysis methods. These are the reasons why this whole chapter is devoted to scripting.

This chapter is written so that it will be valuable both for people who don't know anything about Matlab scripting, people who are experts at Matlab programming, and everyone in between. Over the various exercises in this chapter, you will gradually build up a very sophisticated script. You will start with a script that simply loads one EEGset, and then you will add more and more elements until you have a script with a whole processing pipeline.

11.1: Data for This Chapter

11.2: Expected Background Knowledge

11.3: Bugs as an Opportunity for Growth

11.4: Design of the N170 Experiment

11.5: Exercise- The Matlab Command Line and the EEG Variable

11.6: Exercise- The ALLEEG Variable and Refreshing the ERPLAB Studio GUI

11.7: Exercise- EEG.history and eegh

11.8: Exercise- From the Command Line to a Script

11.9: Exercise- Using a Variable for the Path

11.10: Exercise- Loops

11.11: Exercise- Looping Through Data from Multiple Participants

11.12: Rapid Cycling Between Coding and Testing

11.13: Exercise- Using a Script to Reference EEGsets

11.14: Exercise- Improving the Script and Adding 3D Channel Locations
11.15: Exercise- Preprocessing the EEG and Using a Spreadsheet to Store Subject-Specific Information
11.16: Exercise- Building an Entire EEG Processing Pipeline
11.17: Exercise- Averaging, Quantifying Data Quality, and Plotting
11.18: Exercise- Scoring Amplitudes and Latencies and Performing Statistical Analyses
11.19: Key Takeaways and References

This page titled 11: EEGLAB and ERPLAB Scripting is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 11: EEGLAB and ERPLAB Scripting by Steven J Luck is licensed CC BY 4.0.

11.1: Data for This Chapter

The data we'll be using for the exercises in this chapter can be found in the Chapter_11 folder in the master folder: https://doi.org/10.18115/D5G045.

This page titled 11.1: Data for This Chapter is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 11.1: Data for This Chapter by Steven J Luck is licensed CC BY 4.0.

11.2: Expected Background Knowledge

Learning how to write scripts may seem daunting if you've never done any serious computer programming before. Fortunately, EEGLAB and ERPLAB have a *history* feature that makes it fairly easy to write simple scripts for automating your analyses. Every time you run a routine from ERPLAB Studio or from the EEGLAB GUI, the equivalent script command is saved in the history. To begin writing a script, you simply go through all the steps in the GUI, and then copy the commands from the history into a script file. You'll usually need to make a few minor modifications to the commands, and then you need to add a little bit of general Matlab code so that your script can automatically loop through all your participants. But that's the essence of creating automated EEGLAB/ERPLAB scripts. In fact, that's exactly the process I used to create the example scripts for this book.

The scripts you'll write to automate your EEGLAB/ERPLAB analyses will be pretty simple, but you do need to understand some basic programming concepts, especially *variables*, *arrays*, and *loops*. I'm not going to explain these concepts, so you will need to do a little preparation before starting this chapter if you don't already have some computer programming background. When someone without a lot of programming experience joins my lab, I typically have them take the first half of the online Introduction to Programming with MATLAB course offered on Coursera. If you already have a lot of experience with other programming languages, but you haven't programmed in Matlab, you probably don't need that course; instead, you can get one of the many books that provide a general introduction to Matlab and use that book to learn Matlab's syntax and its unusual but powerful approach to matrix operations.

This page titled 11.2: Expected Background Knowledge is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 11.2: Expected Background Knowledge by Steven J Luck is licensed CC BY 4.0.

11.3: Bugs as an Opportunity for Growth

No matter how much prior experience you have, you need to be prepared for some frustration in this chapter. First, mistakes and bugs are inevitable in programming. As I've said before, you need to think of mistakes and problems as opportunities for learning. In fact, I encourage you to play with the example scripts in this chapter, changing them to see how they work or to try new things. You'll get lots of error messages, but that's actually part of the reason for playing. And that's why I wrote the Troubleshooting Guide in Appendix 2.

I made literally hundreds of mistakes while creating the example scripts in this chapter, and I can't count how many times I heard Matlab's error sound. Most of these mistakes were small and easily fixed. But others were more conceptual and took some time to understand and remedy. In this chapter, I'll describe some of my mistakes, how I discovered them, and how I solved them. I want to both show you that making mistakes is a normal part of the process and give you some examples of the troubleshooting process. There are probably still some mistakes in the scripts for this chapter. If you see something that doesn't quite make sense, it could well be an error on my part.

As you go through the chapter, keep in mind that there are millions of small details about Matlab and EEGLAB that I don't mention. If I provided every possible detail, this chapter would have turned into an entire book. I wanted to keep the chapter reasonably brief, so I'm counting on you to draw inferences from what you see and to figure some things out for yourself. This will be a little frustrating at times, but you'll learn a lot more this way. When you have questions, do what I did hundreds of times while writing this chapter: Google it. Or ask someone with more experience. Or ask ChatGPT, Copilot, Claude, or some other AI chatbot. Or post a question via email to the EEGLAB or ERPLAB listservs. And don't forget about the Troubleshooting Guide in Appendix 2.

Once you've completed the exercises in this chapter, I recommend going back and looking at the example scripts at the end of the previous chapters. They include examples of specific processing steps and options that aren't covered in the present chapter. You can also find lots of examples of EEGLAB/ERPLAB scripts online, and you may have labmates or colleagues who can provide example scripts. However, I have an important rule about this: you should never apply someone else's script to your own data unless you fully understand every line of code in that script. People often violate this rule, and they end up with garbage (and often don't realize it).

This page titled 11.3: Bugs as an Opportunity for Growth is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 11.3: Bugs as an Opportunity for Growth by Steven J Luck is licensed CC BY 4.0.

11.4: Design of the N170 Experiment

This section will provide a brief overview of the experimental design and main results from the ERP CORE N170 experiment, which was based on a prior study by Rossion and Caharel (2011). The N170 is typically found to be larger for faces than for almost any class of non-face stimuli. In this experiment, cars were used as the non-face stimuli. We also presented phase-scrambled faces and cars, which contain the same low-level information as the faces and the cars but without the higher-level features that are essential for discriminating between the broader classes of faces and cars. The stimuli and results are summarized in Figure 11.1.

The N170 effect is largely independent of what task the participants are performing, but it can be helpful to have participants perform some kind of task to keep them alert and attentive. In this experiment, participants were instructed to discriminate whether a given stimulus was an intact image or a scrambled image. Specifically, they pressed one button for faces and cars and another button for scrambled faces and scrambled cars. These four stimulus classes were presented in random order. Stimulus duration was 300 ms, and a blank interstimulus interval of 1100-1300 ms occurred between stimuli.

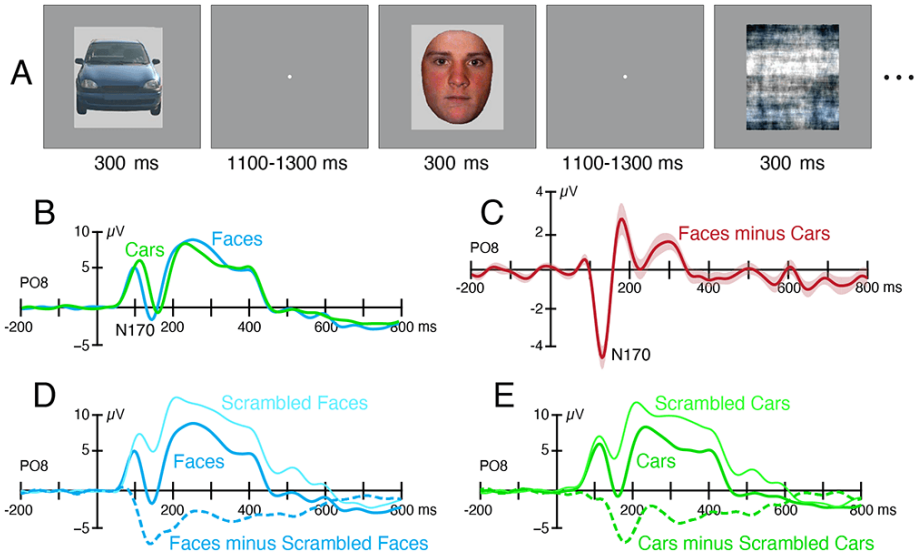


Figure 11.1. Experimental paradigm and results from the ERP CORE N170 experiment. (A) Example sequence of stimuli. Participants were instructed to press one of two buttons to indicate whether a given stimulus was intact (face or car) or scrambled (scrambled face or scrambled car). (B) Grand average ERP waveforms for car and face stimuli, averaged across all 37 participants. (C) Grand average N170 difference wave, which was created by subtracting the car ERP waveform from the face ERP waveform. (D) Grand average ERP waveforms for faces, scrambled faces, and the difference between faces and scrambled faces. (E) Grand average ERP waveforms for cars, scrambled cars, and the difference between cars and scrambled cars. All waveforms show data from PO8 referenced to the average of all channels.

We used 40 different face images, 40 different car images, and scrambled versions of each of these images. Each of the 160 images was presented once. A different event code was used for each of the 160 images, as indicated in Table 11.1.

Table 11.1. Event codes for the ERP CORE N170 experiment.

		Event Code
Stimuli	Faces	1–40
	Cars	41–80

		Event Code
	Scrambled Faces	101–140
	Scrambled Cars	141–180
	Accuracy	**Event Code**
Responses	correct	201
	incorrect	202

Of the 40 participants who were tested, three had to be excluded because of artifacts. The grand average waveforms from the remaining 37 participants are shown for the face and car stimuli in Figure 11.1.B. Note that, for the sake of simplicity, the exercises in the present chapter include only Subjects 1-10. As you will see, Subject 5 was one of the excluded subjects, so the final analyses in this chapter are based on the data from only 9 participants.

As illustrated in Figure 11.1.B, the peak of the N170 was slightly greater (more negative) for faces than for cars. The main difference between the waveforms appeared to be a faster onset latency for the faces. However, this means that the amplitude was greater for faces than for cars between approximately 110 and 150 ms. This is illustrated in the faces-minus-cars difference wave shown in Figure 11.1.C. When I first saw what looked more like a latency difference than an amplitude difference, I thought we had done something wrong. However, this pattern is actually quite common in N170 experiments.

When you compare the ERPs elicited by faces and cars, it's possible that any differences in the waveforms could be a result of differences in low-level features (e.g., luminance, spatial frequency) that are not actually important in perceiving whether a stimulus is a face or a car. This is why the experiment included phase-scrambled face and car images, which contain the same low-level features as the faces and cars but do not contain the higher-level features that we used to determine whether an image is a face or a car.

Figure 11.1.D shows the ERPs elicited by the faces and the scrambled faces, along with the faces-minus-scrambled-faces difference wave. This difference wave is designed to subtract out any brain activity related to the low-level features shared by faces and scrambled faces. You can see a nice N170 in this difference wave. Figure 11.1.E shows the corresponding waveforms for cars, scrambled cars, and the cars-minus-scrambled-cars difference. You can again see an N170, but it's smaller and later than the N170 in the faces-minus-scrambled-faces difference wave.

This page titled 11.4: Design of the N170 Experiment is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 11.4: Design of the N170 Experiment by Steven J Luck is licensed CC BY 4.0.

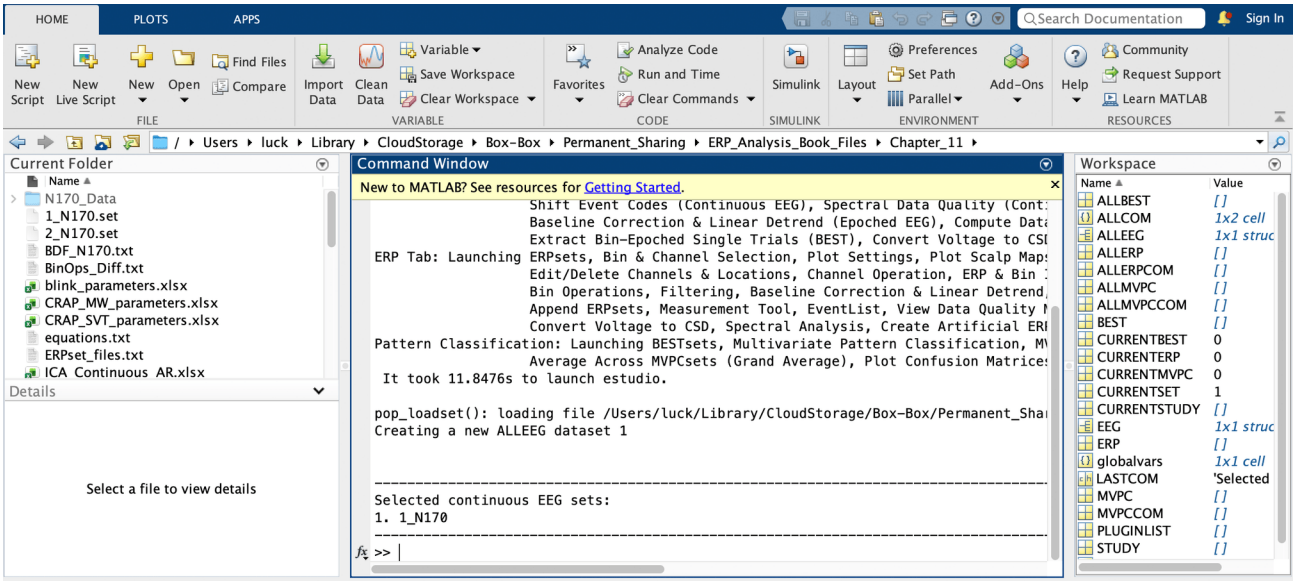
11.5: Exercise- The Matlab Command Line and the EEG Variable

A Matlab *script* is simply a series of Matlab commands that are stored in a text file. Running a script is equivalent to typing the commands on the Matlab command line. So, we're going to start by running some commands from the command line.

To start, quit ERPLAB Studio if it's already running. Then, type **clear all** on the Matlab command line (i.e., at the prompt in the Matlab Command Window pane; see Screenshot 11.1). This clears everything out of Matlab's memory, which is a good thing to do when you're first getting started on a new task. When you clear the variables, anything that was in Matlab's Workspace pane should disappear. Another nice housekeeping command is **clc**, which clears the command window so that you're not distracted by what happened earlier.

By the way, Matlab doesn't do anything with a command until you hit the **Return** key (which may instead be labeled **Enter** on your keyboard). When I say that you should type something on the Matlab command line, you should follow it with **Return** or **Enter**. If you didn't already press **Return/Enter** after **clear all**, do it now.

Now launch ERPLAB Studio by typing **estudio** on the command line, go to the **EEGsets** panel, set the **Chapter_11** folder to be the current folder, and load the EEGset named **1_N170.set**. You should now see a set of variables in the Workspace pane, as shown along the right side of Screenshot 11.1. This includes **EEG**, which is used to store the current EEGset, and **ALLEEG**, which is used to store all of the EEGsets that are available in memory. You should also see the corresponding **ERP** and **ALLERP** variables, which hold the current ERPset and all available ERPsets.



Screenshot 11.1

You can see the contents of a variable by typing its name on the command line. Let's try it! Type **EEG** on the command line (followed by the **Return** key, of course). Variable names in Matlab are case-sensitive, so make sure you type **EEG** and not **eeg** or **Eeg**. Once you type this, the contents of the **EEG** variable will be shown in the Command Window. Screenshot 11.2 shows the first few lines.

```
EEG =
  struct with fields:
            setname: '1_N170'
           filename: '1_N170.set'
           filepath: '/Users/luck/Library/CloudStorage/Box-Box/Permanent_Sharing/ERP_Analysis_Book_Files/Chapter_11/'
            subject: ''
              group: ''
          condition: ''
```

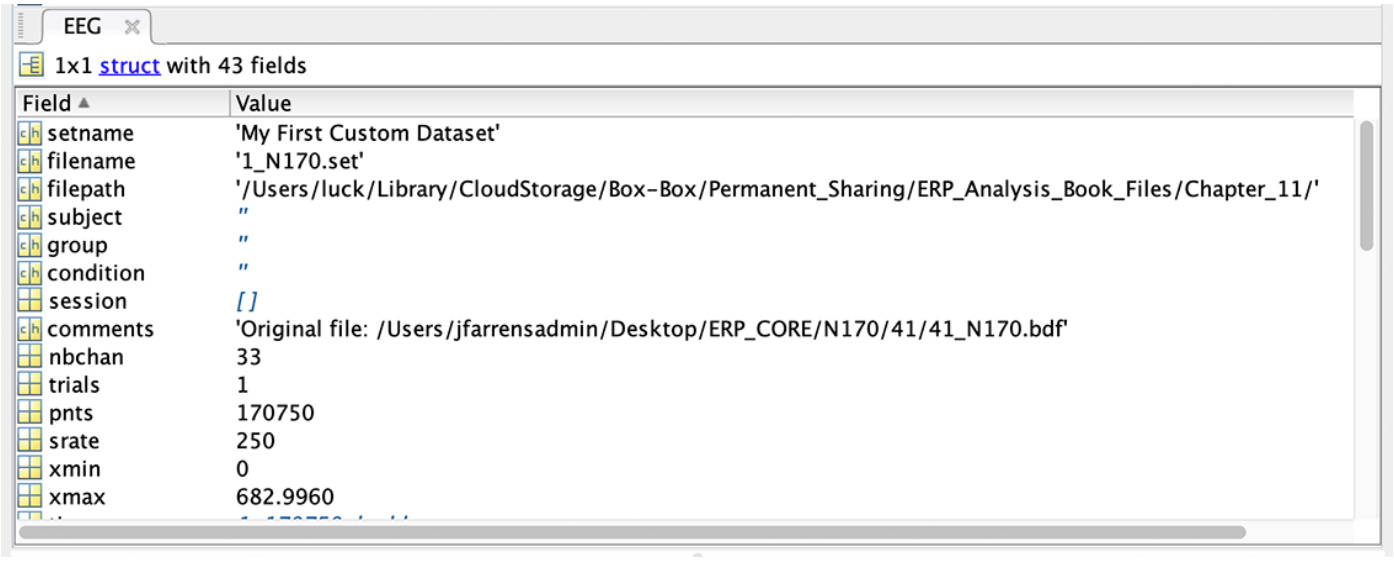
Screenshot 11.2

EEG is a complicated variable that contains many individual fields (you can learn about the details by typing **help eeg_checkset** on the command line). For example, the field named **EEG.setname** stores the name of the EEGset, which is the name shown in the **EEGets** panel (and not necessarily the same as the filename, which is stored in E**EG.filename**). Let's change the name of the dataset. To do this, type **EEG.setname = 'My First Custom Dataset'**. Note that the period at the end of the sentence is not part of the command you should type. I use **boldface** to indicate the exact text you should type. In addition, make sure you include the single quote marks (which are in boldface). Once you type this and press **Return/Enter**, Matlab will print out the whole EEG variable again in the Command Window, and you'll be able to see that the name has changed.

Here are a couple important things to note about the command you just entered:

- Matlab uses single quote marks to indicate literal text. If you didn't use the quote marks and had instead typed **EEG.setname = My First Custom Dataset**, Matlab would have assumed that **My First Custom Dataset** was a sequence of four variable names (**My**, **First**, **Custom**, and **Dataset**). See the text box below for a hint about single quote marks.
- Most Matlab commands return one or more variables, and the value of the returned variables is ordinarily printed in the Command Window. You can suppress this by placing a semicolon at the end of the command. To see this in action, type **x = 1** (followed by **Return/Enter**), and then type **x = 2;** (again followed by **Return/Enter**).
- When you change the EEGset name using the command line, you won't see the new set name in the **EEGsets** panel right away. The reason for this will be explained later in this section.

You can also see the contents of a variable by double-clicking on the name of the variable in Matlab's Workspace pane. Try double-clicking the **EEG** variable in this pane. A new Variables pane should appear in Matlab, showing you the fields of the **EEG** variable (see Screenshot 11.3). If you look at the **EEG.setname** field, you will see the new EEGset name you created.



Screenshot 11.3

Another of the fields in **EEG** is named **times**, and it contains the latency in milliseconds of each time point in the dataset. Double-click it within the Variables pane to see its contents; a new tab will open labeled **EEG.times**, and you'll see a very wide list of latency values. This EEGet has a sampling rate of 250 Hz, so the first point is 0 ms, the second point is 4 ms, the third point is 8 ms, etc.

Single Quotes

Click on the tab for the **EEG** structure and take another look at the **times** field. Next to the **times** name, you should see **1 x 170750 double**. The term **double** is used by Matlab (and many other programming languages) to refer to a number that is stored in scientific notation (e.g., X times 10^{Y}) using twice the traditional precision (and therefore double the amount of storage space). The **1 x 170750** part indicates that **EEG.times** is an array of these double-precision numbers with 1 row and 170750 columns. If you go back to the tab for **EEG.times**, you'll see that it has one row and 170750 columns (one column for each data point in the dataset).

In the Matlab Variables pane, you'll see a field within the **EEG** variable named **data**, which is listed as **33 x 170750 single**. This variable stores the actual voltages in the dataset. It has 33 rows (one for each channel) and 170750 columns (one for each time point). That's a pretty natural way to store EEG data, isn't it? To save space, EEGLAB stores it using single-precision numbers.

When you start writing scripts, it's easy to get confused about the rows versus the columns of an array. I find it helpful to look at the array in the Variables pane to remind myself which dimension is the rows and which is the columns.

This page titled 11.5: Exercise- The Matlab Command Line and the EEG Variable is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 11.5: Exercise- The Matlab Command Line and the EEG Variable by Steven J Luck is licensed CC BY 4.0.

11.6: Exercise- The ALLEEG Variable and Refreshing the ERPLAB Studio GUI

When you have multiple EEGsets in the **EEGsets** panel, this means that those EEGsets have been loaded into memory. The current EEGset is stored in the **EEG** variable. All of the EEGsets (including a copy of the current EEGset) are stored in a variable named **ALLEEG**, which is just an array of **EEG** variables. When you go to the **EEGsets** panel, you're seeing a list of the EEGsets stored in **ALLEEG**. This is one reason why changing the name of **EEG.setname** didn't cause a change in the **EEGsets** panel. That is, we changed the name of this field in the **EEG** variable, but not in the **ALLEEG** variable that is used for the **EEGsets** panel.

Let's fix that by typing **ALLEEG(1) = EEG;** on the command line (including the semicolon so that it doesn't print the new value in the Command Window). Now type **ALLEEG(1).setname** (without a semicolon), and you'll see that the first (and only) **EEG** structure in **ALLEEG** has the new name ('My First Custom Dataset').

However, if you look in the **EEGsets** panel, you still won't see the new name for the EEGset. This is because the ERPLAB Studio GUI doesn't "know" that the **ALLEEG** variable has changed. That is, the GUI only updates the names listed in the panel when it thinks that something has changed. You can tell ERPLAB Studio that there has been a change in the EEG or ALLEEG data by clicking the **Refresh** button in the **EEGsets** panel. Try this, and then look at the **EEGsets** panel. Now you should see the updated name (**My First Custom Dataset**). Note that there's also a **Refresh** button in the **ERPsets** panel for updating the names of the ERPsets in this panel.

Now let's see what happens when we have two EEGsets loaded in ERPLAB Studio. To make the second dataset, filter the EEGset you already have loaded by going to the **Filtering** panel (in the EEG tab) and specifying a high-pass cutoff at **0.1** Hz, a low-pass cutoff at **30** Hz, and a roll-off of **12** dB/octave. Name the resulting EEGset **01_N170_filt**. You should now see two EEGsets in the **EEGsets** panel.

If you look at the **EEG** tab in the Variables pane, you will see that **EEG.setname** is now **01_N170_filt**. This is because the EEG variable always holds the currently active EEGset (which is the filtered EEGset you just created). If you double-click on **ALLEEG** in the Workspace pane to see it in the Variables pane, you'll see that **ALLEEG** now has a size of 1 x 2 because we have two datasets loaded. If you type **ALLEEG(1).setname**, you'll see the name of the first dataset. If you type **ALLEEG(2).setname**, you'll see the name of the second dataset.

Leave these two datasets loaded for the next exercise.

This page titled 11.6: Exercise- The ALLEEG Variable and Refreshing the ERPLAB Studio GUI is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 11.6: Exercise- The ALLEEG Variable and Redrawing the GUI by Steven J Luck is licensed CC BY 4.0.

11.7: Exercise- EEG.history and eegh

EEGLAB has a super convenient *history* feature, which we shamelessly copied when we created ERPLAB. Whenever you run a routine from the ERPLAB Studio GUI, the equivalent Matlab command text is saved in the history. The **history** field of the **EEG** variable (i.e., **EEG.history**) stores the history of the routines that were applied to that specific EEGset. You can see the history by typing **EEG.history** on the Matlab command line. In addition, you can get a history of everything that was done since the last time you launched eStudio (which might include routines applied to many different EEGsets) by typing **eegh** on the command line.

You can also see the history in ERPLAB Studio's **History** panel (which allows you to select the **Current EEGset** history or the **Current session** history).

Go to the **EEGsets** panel and select the first EEGset (which should be named **My First Custom Dataset**). This EEGset should now be stored in the **EEG** variable, and if you type **EEG.history** you should see the history of the routines that were applied to this EEGset:

```
>> EEG.history

ans =
    EEG.etc.eeglabvers = '2025.0.0'; % this tracks which version of EEGLAB is being used, you may ignore it
        EEG = pop_loadset('filename','1_N170.set','filepath','/Users/luck/Library/CloudStorage/Box-Box/Permanent_Sharing/ERP_Analysis_Book_Files/Chapter_11/');
```

The first line (beginning **EEG.etc.eeglabvers**) was automatically added by the software to help you track what version of EEGLAB was used. The second line (beginning **pop_loadset**) is the command you used to load the EEGset from a file. Notice that the Matlab command that you used to change the name of the EEGset isn't in the history. Only EEGLAB and ERPLAB operations are automatically saved in the history.

I want to pause for a minute and encourage you to appreciate what just happened. You used your mouse to load the EEGset using the ERPLAB Studio GUI, which should have been simple and natural now that you've spent a lot of time using ERPLAB Studio. And then the history allowed you to see the actual Matlab command that was used to load the EEGset. This is the same command you'll use to load EEGsets when you start writing scripts. So, you don't need to read some opaque documentation to figure out what command to use and how to use it. You just need to run the routine from the GUI and look at the history.

This makes scripting so much easier! And it's not just for beginners: when I created the scripts for each chapter of this book, I first ran the routines from the GUI so that I could get the history, which I then copied into the scripts that I was writing. I'm not sure which of the original EEGLAB developers came up with this scheme, but it's brilliant!

Now that I've gushed about how great this is, I want to note a small complication. If you look closely at the **pop_loadset** command that was used to load the EEGset, you'll see that it specifies the filename (**'filename', '1_N170.set'**) and the location of the file on your hard drive (the *path* to the file, listed in the string following **'filepath'**). However, the path that you can see on your computer is not the same as the path shown above, which shows the path on my computer (**'/Users/luck/Library/CloudStorage/Box-Box/Permanent_Sharing/ERP_Analysis_Book_Files/Chapter_11/'**).

Keep in mind that the paths you'll see in this book reflect the organization of files on my computer, so they won't be the same as the paths you'll see on your computer. Paths can be pretty confusing. One of the most common problems people have when learning to write scripts is getting the paths wrong. I often get them wrong myself. Later, I'll demonstrate a strategy you can use to avoid path problems.

Now select the second EEGset in the **EEGsets** panel (**01_N170_filt**), and type **EEG.history** to see the history for that EEGset:

```
EEG.etc.eeglabvers = '2025.0.0'; % this tracks which version of EEGLAB is being used; you may ignore it
EEG = pop_loadset('filename', '1_N170.set', 'filepath','/Users/luck/Library/CloudStorage/Box-Box/Permanent_Sharing/ERP_Analysis_Book_Files/Chapter_11/');
EEG = pop_basicfilter( EEG, 1:33, 'Boundary', 'boundary', 'Cutoff', [ 0.1 30], 'Design', 'butter', 'Filter', 'bandpass', 'Order', 2, 'RemoveDC', 'on' ); % GUI: 25-Oct-2025 06:26:07'
```

You'll see that the first two lines are the same as the history for the first EEGset. That's because those commands are part of

the sequence of operations that was applied to the second EEGset. Following those lines, you can see the command for filtering the EEG (beginning with **pop_basicfilter**). When you're writing a script that includes filtering, you can see what text to write by filtering a file in the GUI, looking at the history, and then copying this line of code into your script.

Now go back to the **EEGsets** panel and select the first EEGset (**My First Custom EEGset**). Then delete channel 33 from this EEGset by going to the **Edit/Delete Channels & Locations** panel, entering **33** in the **Chan** field, and clicking the **Delete chan** button. We've now modified both of the EEGsets, deleting channel 33 from EEGset 1 and filtering EEGset 2. If you select one of these two EEGsets, which causes it to be stored in the **EEG** variable, you can see the Matlab commands for these operations.

If you want to see everything you've done since launching eStudio, you can use type **eegh** in the Command window. If you do that now, you should see something like this:

```
EEG = pop_loadset('filename','1_N170.set','filepath','/Users/luck/Library/CloudStorage/Box-Box/Permanent_Sharing/ERP_Analysis_Book_Files/Chapter_11/');
% Selected continuous EEG sets: [1. 1_N170]
EEG = pop_basicfilter( EEG, 1:33 , 'Boundary', 'boundary', 'Cutoff', [ 0.1 30], 'Design', 'butter', 'Filter', 'bandpass', 'Order', 2, 'RemoveDC', 'on' ); % GUI: 25-Oct-2025 06:26:07
% Selected continuous EEGsets: [2. 01_N170_filt]
% Selected continuous EEGsets: [1. My First Custom Dataset]
% Selected continuous EEGsets: [2. 01_N170_filt]
% Selected continuous EEGsets: [1. My First Custom Dataset]
EEG = pop_select( EEG, 'rmchannel',{'VEOG_lower'});
% Selected continuous EEG sets: [1. My First Custom Dataset]
```

Note that several lines begin with a percent symbol (**%**). Matlab uses this symbol to denote that a line is a comment rather than code to be executed. ERPLAB Studio adds comments to the session history to make it clear which EEGset or ERPset has been selected.

The first line in the history above is the command for loading the first EEGset from your hard drive into the **EEG** variable. The next line is a comment indicating that EEGset 1 is now selected. The next line of code is for the filtering routine, which operates on the **EEG** variable. The resulting EEGset was saved as the second EEGset in **ALLEEG**, and became the selected EEGset, so the following comment indicates that the second EEGset is now selected.

The next several lines are comments resulting from the fact that the previous exercises involved switching between the two EEGsets in the **EEGsets** panel.

The next line of Matlab code (i.e., the next line that does not begin with a **%**) shows the operation for removing channel 33 from the first EEGset. You can tell that it operated on the first EEGset because the comment right above it indicates that EEGset 1 was selected.

As a reminder, you can also see the history in ERPLAB Studio's **History** panel, selecting either the **Current EEGset** history (like typing **EEG.history**) or **Current session** history (like typing **eegh**). It's hard to see the history very well in a small panel, so there is a button for showing the history in the command window. There's also a button that lets you save the history in a file, which is useful when you are ready to turn the history into a script.

You now know how to see the Matlab commands for the routines you run in the ERPLAB Studio GUI. This will make your life much easier in the following exercises, where you'll start putting together scripts.

This page titled 11.7: Exercise- EEG.history and eegh is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 11.7: Exercise- EEG.history and eegh by Steven J Luck is licensed CC BY 4.0.

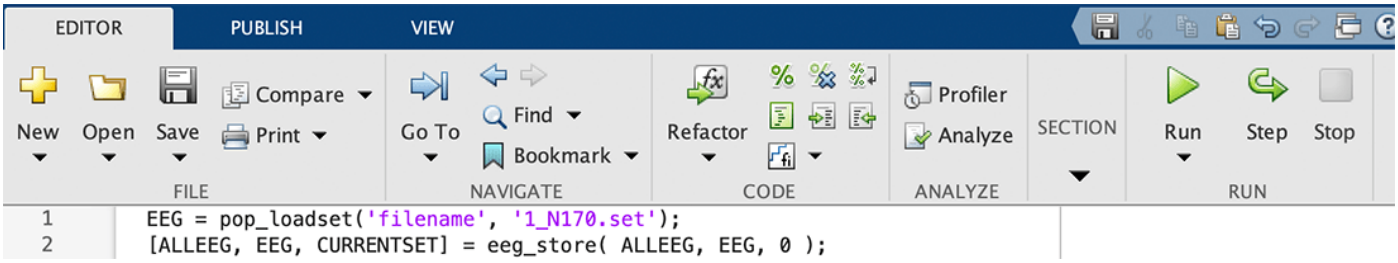
11.8: Exercise- From the Command Line to a Script

In this exercise, we'll see how commands that you type on the command line can be run from a script file. To start from a blank slate, you should quit eStudio, type **clear all** on the command line, and relaunch eStudio. Make sure that **Chapter_11** is the current folder in Matlab, and open the file named **Script1.m** by double-clicking the name from Matlab's **Current Folder** pane. This script consists of three lines, which open a dataset, add it to **ALLEEG**, and redraw the Matlab GUI.

Copy the first line of the file (beginning **EEG = pop_loadset...**) into your computer's clipboard, and then paste it onto the command line to execute the command. This command loads the EEGset from Subject 1 in the N170 ERP CORE experiment. You can determine whether it has successfully loaded the EEGset into the **EEG** variable by looking at this variable in the Matlab Workspace pane and seeing that **EEG.setname** now has a value of **'1_N170'**. Now copy and paste the second line of **Script1.m** onto the command line to add the dataset to **ALLEEG** (which you can verify by looking at **ALLEEG** in the Workspace pane). Now go to the **EEGsets** panel in eStudio and click the **Refresh** button so that eStudio knows about the EEGset you loaded. You should now see it in the **EEGsets** panel.

Let's take a closer look at the first line: **EEG = pop_loadset('filename', '1_N170.set');** This is just like the first line of the **EEG.history** in the previous exercise, except that it doesn't specify the path. That's because the **1_N170.set** file is in the current folder (which you can see in Matlab's Current Folder pane). When you try to open a file but don't specify a path, Matlab will first look in the current folder. If it doesn't find the file there, it will search the entire Matlab path. When you save a file but don't specify a path, Matlab will save it in the current folder. We'll look at a robust way to specify the path in the next exercise.

First, though, we'll see how to run an actual script. In theory, you could just put a set of commands in any text file and then copy-and-paste them onto the command line. The file would just be a way to store the commands so that you can easily reuse them. In fact, I sometimes use script files for this purpose. But there are easier ways to execute the commands in a script file. The simplest is just to click the **Run** button in the Matlab script editor. In my version of Matlab, it's a green triangle (see Screenshot 11.4). To see this in action, quit eStudio, type **clear all** in the Command Window, restart eStudio, and click the **Run** button in the editor window for **Script1.m**. If you look at the **EEG** variable and the **ALLEEG** variable, you can confirm that each line of the script has executed. To see this EEGset in eStudio, go to the **EEGsets** panel and click the **Refresh** button.



Screenshot 11.4

Now let's look at three ways that we can execute a portion of a script rather than the whole thing. The first way is simple: Copy the set of lines that you'd like to execute into the clipboard and paste them onto the command line. You've already done this with single lines of code, but you can do this with multiple lines.

The other two ways are easier to demonstrate with a script that has several distinct parts, so load the script named **Script2.m** (e.g., by double-clicking on this name in the Current Folder pane). This script loads the data from Subjects 1 and 2. I'll explain how this script works in the next exercise. For now, we'll just see how to execute parts of it.

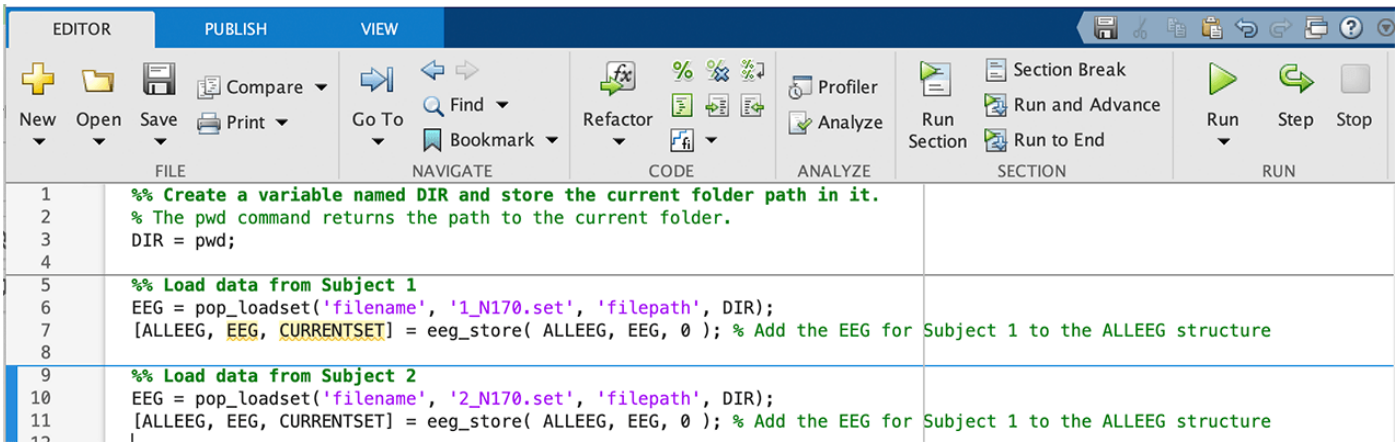
Start by quitting eStudio, typing **clear all**, and restarting eStudio. Now, go into the text editor for **Script2.m**, and select the first 7 lines (by dragging your mouse over those lines; note that you can see the line numbers along the left side of the window). Now right-click (or control-click) on the selected text, and select **Evaluate Selection in Command Window** from the menu that pops up. (You can also see a keyboard equivalent, which will be more convenient for using this approach in the future.) This should load the dataset for Subject 1, which you can verify by looking at the contents of the **EEG** variable.

Now select and execute lines 9-11 to load the EEGset for Subject 2, and then click the **Refresh** button in the **EEGsets** panel to update the GUI. You should now see the EEGsets from both Subject 1 and Subject 2 in the **Datasets** menu. Note that you didn't

execute all the lines of the script at once. You can select any set of lines from a script and run them, in any order you want. This approach gives you a lot of flexibility. The lines in a script usually need to go in a particular order to give you the desired result. For example, you can't filter a file until after you've loaded it. But sometimes you're just using the script as a convenient place to store a bunch of commands that you might execute in some other order.

Now we're going to look at one more way to execute commands from a script, in which we divide the script into sections and run one section at a time. If you look at **Script2.m**, you'll see that some lines begin with a **%** symbol. This symbol indicates that the line is just a comment, not code that will be executed. You can also put a **%** symbol after a command to provide a comment about that command (see examples in Screenshot 11.5).

You should also note that some of the lines begin with two consecutive **%** symbols. This indicates the start of a new *section*. If you click on the line that begins with two consecutive **%** symbols, you can run that section of code by clicking the **Run Section** button in the toolbar near the top of the script editor window.



Screenshot 11.5

This page titled 11.8: Exercise- From the Command Line to a Script is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 11.8: Exercise- From the Command Line to a Script by Steven J Luck is licensed CC BY 4.0.

11.9: Exercise- Using a Variable for the Path

In this exercise, we're going to take a closer look at **Script2.m**, which demonstrates how to use variables to specify the paths to your files, which is much more robust. Make sure that **Chapter_11** is still the current folder, and type **pwd** on the command line (*pwd* stands for *print working directory*, where *directory* is another name for *folder*). You should see the path for your current folder. Line 3 of **Script2.m** uses the **pwd** command to get the path to the current folder and store it in a variable named **DIR** (short for *directory*, but we could have named it almost anything). Run line 3 of the script (using any of the methods described in the previous exercise) and then look at the value of **DIR** (by typing **DIR** on the command line or by double-clicking it in the Workspace pane). You'll see that **DIR** is a string that holds that path to the current folder.

Now we can use **DIR** as the starting point for providing paths to our Matlab commands. This is much more robust than specifying something like **C:/ERP_Analysis_Book/Ch_10_Scripting** as the path. For example, if you move your data and scripts to a new location on your computer or a new computer, using **DIR** as the path will still work (because the **pwd** command will return the new location of the script), but specifying the path directly will now fail. You just need to make sure that the script you are running is in Matlab's current directory when you run it.

We didn't really need to specify the paths in **Script2.m**, because the files were located in the current folder. Close **Script2.m** and open **Script2b.m**, which shows a more realistic example. This new script assumes a more complicated but better organization for your data files, in which each participant has a separate folder inside of a general data folder.

This is the organization that I generally recommend for EEG and ERP data (whether or not you use eStudio). When you are analyzing your data, you will typically end up with a lot of files for each participant, and this just keeps them well organized. There's some redundancy, because I also recommend including the Subject ID number in each filename. But this redundancy is useful because it minimizes the likelihood of errors. Nothing about EEGLAB or eStudio requires this organization, but it's used for all the scripts in this book. You can choose a different organization, but please don't just put all the files for all the participants into a single folder. It will seem simpler at first, but it will make your life much more difficult in the long run. You just don't want to have 3478 files in the same folder!

This organization means that you need a different path for each participant. In **Script2b.m**, we implement this by using **DIR** as a base folder (the folder that contains the script) and then use this to create a variable named **Subject_DIR** that holds the path for the participant we're currently processing.

Important note: This approach assumes that Matlab's current folder is set to be the folder that contains the script. You should make sure this is true when you run the scripts in this book.

Line 6 of **Script2b.m** sets the value of **Subject_DIR** like this:

```
Subject_DIR = [DIR '/N170_Data/1/']; % Folder holding data for this subject
```

Putting multiple character strings inside square brackets causes Matlab to concatenate the strings together. For example, **['ERPs' 'Are' 'Great']** is equivalent to '**ERPsAreGreat**', and **['ERPs ' 'are ' 'great']** is equivalent to '**ERPs are great**'. If **DIR** has a value of **'C:/ERP_Analysis_Book/Ch_10_Scripting**', then **[DIR '/N170_Data/1/']** has a value of **'C:/ERP_Analysis_Book/Ch_10_Scripting/N170_Data/1/**'.

To see this in action, run lines 3 and 6 of **Script2b.m** and then look at the value of **Subject_DIR**. This is what the value of **Subject_DIR** looks like on my computer:

```
Subject_DIR =

    '/Users/luck/Library/CloudStorage/Box-Box/Permanent_Sharing/ERP_Analysis_Book_Files/Chapter_11/N170_Data/1/'
```

The **Subject_DIR** variable is then used on line 7 to load the EEGset for subject 1 from **N170_Data/1/**. Line 11 will update Subject_DIR so that it will load the data for Subject 2 from **N170_Data/2/**.

This may seem like a lot of work, but I guarantee that it will make your life easier in the long run. For example, if you develop a script for one experiment and want to use it for a second experiment, you can place a copy of the script in the folder for the second experiment, and the script will automatically look for the data files in the correct place.

If you're using a Windows machine, **Script2b.m** may not work properly because it uses slashes between folder names rather than backslashes. You can just replace the slashes in the paths with backslashes. In a later section, I'll show you a better way to do this that will allow your scripts to work on any operating system that can run Matlab.

This page titled 11.9: Exercise- Using a Variable for the Path is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 11.9: Exercise- Using a Variable for the Path by Steven J Luck is licensed CC BY 4.0.

11.10: Exercise- Loops

In this exercise, we'll introduce the concept of *loops*, which will make your life much easier by allowing you to automate the processing of multiple participants. There are several kinds of loops, but the most common is called a *for loop*. It allows you to repeat the same sequence of steps multiple times, but with one variable changing (e.g., a variable that indicates which participant should be processed). We're going to go through several examples in this exercise, starting simple and working our way up to a script that processes the data from multiple participants.

Open **Script3.m** by double-clicking on it in the Current Folder pane and then run it (e.g., by clicking the **Run** button). You should see that it prints a list of numbers between 1 and 10, formatted like this:

```
subject =

    1
```

In a later script, these will be the ID numbers of our participants. Note that the number 5 is missing. This is meant to indicate that Subject 5 is being excluded (because that subject had too many artifacts and was excluded from the final analyses).

Now take a look at the script. The main body of the script is this:

```
for subject = [ 1 2 3 4 6 7 8 9 10 ]

   display(subject);

end
```

In Matlab, a **for** loop begins with **for** and ends with **end**. The lines of code between the **for** and **end** lines are the *body* of the loop. This body will be executed multiple times, once for each element specified in the array on the **for** line. The **for** line defines a variable (which we've named **subject** in this example) and specifies an array of values that will be stored in this variable each time go through the loop (**[1 2 3 4 6 7 8 9 10]** in this example).

In **Script3.m**, the body of the loop is a single line consisting of the **display(subject);** command, which just prints the value of the variable named **subject** that we specified on the **for** line. Note that it's conventional to indent the body of a loop using tabs. The tabs are ignored by Matlab when a script is executed, but they make it easier to see the structure of a script. Not required, but highly recommended!

Each time we go through the loop, the variable named **subject** is set to a new value in the array of values following the equals sign on the **for** line. When the loop starts, **subject** will be set to 1 (because 1 is the first value in the array). The **display(subject)** line will then execute, and it will display a value of 1 because that's the value of the **subject** variable at this moment. Then the **end** line occurs, telling Matlab to go back to the start of the loop and set **subject** to the next value in the array (2). The **display(subject)** line will then execute, but this time it will display a value of 2 because that's now the value of the **subject** variable.

Matlab will keep repeating the loop, setting **subject** to 3, then to 4, then to 6, etc. There is no 5 in our array of values, so **subject** will never take on that value. The array is just a list of values, and any sequence of values will work. For example, we could use **[3 1 5 9]**, and then **subject** would take on that set of values in that order (3, then 1, then 5, then 9). We could even use non-integer numbers (e.g., **[5.23 6.1 -5.442 10021.2]**) or character strings (e.g., **['S1' 'S2' 'S3' 'S5']**). Matlab is much more flexible than most programming languages in this regard. Spend some time playing with the array in the script so that you get a good sense of how it works.

Now close **Script3.m** and open **Script3b.m**, which is a slightly more sophisticated version of the same set of ideas. Run the script to see what it does. You should see a set of lines starting with this:

```
Processing Subject 1
Processing Subject 2
Processing Subject 3
```

Now look at the script. You'll notice two main changes from the first script. First, instead of providing an explicit list of subject

IDs in the **for** line, we've defined a variable named **SUB** that stores this array:

SUB = [1 2 3 4 6 7 8 9 10]; %Array of subject IDs

We then specify this variable as our array in the **for** statement:

for subject = SUB

This approach embodies my #1 principle of writing good code: All values used by a script should be defined as variables at the top of the script. I'll say more about this principle later in the chapter.

The second change to the script is that it uses the **fprintf** command instead of the **display** command to print the value of the **subject** variable. The **fprintf** command is much more powerful and flexible. It takes a little time to learn to use it, but it's well worth the time. Here's the specific version used in our script:

fprintf('Processing Subject %d\n', subject);

The first parameter in the **fprintf** command is a *formatting string*. It contains plain text that is printed by the routine (e.g., **Processing Subject**), and it also includes formatting statements for variables that appear as subsequent parameters. The **%d** tells the command to print a whole number (the value of the **subject** variable), and the **\n** tells it to print a newline (a return). You can do much more than this with **fprintf**, but we're keeping it simple for now (see Matlab's documentation for **fprintf** for details).

Once you understand how this script works, open and run **Script3c.m**. You'll see that the output has more information than provided by the previous script. For each iteration of the loop, it indicates how many times we've gone through the loop, plus the subject's ID. This allows us to see that the fifth subject has an ID of 6, not 5.

Look at the script to see how it works. Near the top, you'll see that we use a Matlab function called **length** to define a variable named **num_subjects** that stores the number of subjects in the **SUB** array:

num_subjects = length(SUB);

We've then used this new variable to define the array of values for the loop, which we've defined as **1:num_subjects**. Type **1:num_subjects** on the command line. You'll see that it is equivalent to a list of integers between **1** and **num_subjects** (i.e., 1 through 9). As a result, we're no longer looping through the subject IDs. As a result, I've changed the name of the variable in the **for** statement to **subject_index**.

Each time through the loop, we get the subject ID by finding the element in **SUB** that corresponds to **subject_index** and store it as a text string in a variable named **ID**. For example, when **subject_index** is 5, we get the 5th element of **SUB**, which is 6 (because **SUB** skips subject 5). **SUB** is an array of numbers, but as you'll see in the next script, it's useful to store the ID as a text string. We therefore use a Matlab function called **num2str** to convert the number to a string before storing it in **ID**. Note that the format string for the **fprintf** command uses **%s** to indicate that this command should print a string variable for **ID**.

Why it Pays to Include Good Comments and Meaningful Variable Names in Your Scripts

When you're in the middle of writing a script to process the data for an experiment, you will get very focused on *getting the job done*. That is, you just want the script to work so that you can get to the next step of the project (and ultimately to the point of submitting a paper for publication). However, the fastest route to a goal is not always the straightest: If you focus too much on the immediate goal of getting the script to work, you may actually slow your progress toward the final goal of getting the paper submitted. It really pays to spend a little extra time initially to write the code in a way that will be optimal in the long run.

In practice, this means following good coding practices that reduce the likelihood of errors, like defining all important values as variables at the top of the script. Errors can really slow you down if you don't realize the error until you're near the point of submitting the paper and now need to repeat all the analyses, change all the figures, and update the text of your paper. It's also important to realize that you will probably need to come back to your script many months after you've written it (e.g., when you're writing your Method section or you realize you need to reanalyze your data), and you will save yourself a lot of time if you write your code in a way that's easy to read later.

There are two straightforward ways of making your code more readable. The first is to use variable names that have an obvious meaning. For example, I could have used something like **ns** as the name of the variable that holds the number of subjects, but instead I used **num_subjects**. The second is to add lots of comments. For example, take a look at the example scripts I created for this chapter. Of course, these scripts were designed to be read by other people, so I put more a bit more work into those comments than I might have for a regular script that I wasn't planning to share. But I still include tons of comments in scripts that I don't plan to share. It's a gift to my future self, because I know I will probably need to come back to those scripts after months or even years, and the comments will make my life much easier then. And when I come back to a script with good comments, I always try to thank my past self for the gift.

You should also keep in mind that it's becoming more and more common to make your data and scripts available online when you publish a paper. This means that you're never really writing scripts just for yourself. Other researchers are going to be looking at your scripts, so you want the other researchers to be able to understand your code (and you don't want the scripts to be embarrassing!). If you make the code easy to understand, this increases the likelihood that the other researchers will follow up on your research, which means that your research will have a larger impact. And aren't you doing research so that it has an impact? I know that I trust a paper a lot more if I see that the analysis scripts are well-written and well-documented.

I've found that people often spend a huge amount of time polishing their scripts (making them more logical and adding lots of comments) right before they're going to submit the paper for publication. They often find mistakes, and then they end up having to change their figures and the statistics in their Results sections. It's really inefficient. It makes much more sense to write your scripts right from the beginning with the intent of sharing them, making sure that the logic is clear, the variables have meaningful names, and everything is clearly commented. This takes a lot of discipline, because when you're writing a script, you just want to get the job done. But this approach will save you a lot of time and agony in the long run.

This page titled 11.10: Exercise- Loops is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 11.10: Exercise- Loops by Steven J Luck is licensed CC BY 4.0.

11.11: Exercise- Looping Through Data from Multiple Participants

In this exercise, we'll get to see how loops are used to load the datasets from a set of participants. We'll add some processing steps that make the script more useful in the next exercise, but I wanted to keep things simple for now. To get started, quit EEGLAB, close any open scripts, type **clear all**, and open **Script4.m**. But don't launch EEGLAB—we'll have the script do that!

Go ahead and run **Script4.m** to see what it does. It should launch eStudio and then load the EEGsets for Subjects 1-10 (except Subject 5). After it runs, you'll need to click the **Refresh** button in the **EEGsets** panel to see the loaded EEGset.

Now let's look at the script and see how it works. The first line of code launches eStudio, which creates several variables that you will find useful (e.g., **EEG** and **ALLEEG**). After the script runs, you can see these variables in Matlab's **Workspace** pane.

The next line of code creates the **DIR** variable, as in the previous scripts, which holds the location of the script (and should be the **Chapter_11** folder). Then the script creates a new variable named **Data_DIR**, which appends **'/N170_Data/'** onto the **DIR** variable. This gives us a path to the folder containing the single-participant data folders.

The next step is to define a variable named **Dataset_filename,** which has a value of **'_N170.set'**. We'll eventually combine this variable with the subject ID to get the entire filename for a given participant (e.g., **1_N170.set**).

Then we define variables for the list of subjects and the number of subjects, just as in the previous example. Note that these steps embody the principle that all values used by a script should be defined as variables at the top of the script. It's a little extra up-front work to do this, but it dramatically reduces the likelihood of bugs later (especially when you take a previous script and modify it for a new purpose).

The next step is to loop through the subjects. The first part of this is just like what we did in the previous script, including setting **ID** to be a string with the current subject's ID. Then the script creates a variable named **Subject_DIR**, which specifies the folder that holds that data for the subject currently being processed by the loop (e.g., **.../Chapter_11/N170/1/** for the first subject). We do this by concatenating the **Data_DIR** variable with the **ID** variable and then a **/** character. We also create a variable named **Subject_filename** by concatenating the **ID** variable with the **Dataset_filename** variable. This gives us a value of **1_N170.set** for the first subject.

We then load the EEGset for this subject, using **Subject_filename** as the filename and **Subject_DIR** as the path. The EEGset is stored in the **EEG** variable, and our last step in the body of the loop is to add this EEGset to the **ALLEEG** variable using the **eeg_store** routine. The zero we specify as the last parameter for this routine tells it to add the new dataset to the end of **ALLEEG**.

Although this script loads the EEGsets for only 9 participants, there are actually 40 participants in this experiment, each with an EEGset. This script is a much faster way of loading these 40 EEGsets than using the GUI to separately load each one. Because all the key values are specified as variables at the top of the script, you can easily find them and modify them so that you can use the same script with another experiment, assuming that the data are organized in the same way on your computer. You'd just need to modify the list of subject IDs (the **SUB** variable), the name of the folder holding the data (**Data_DIR**), and the base dataset name (**Dataset_filename**). This will be much faster and easier if you're consistent in how you organize the data for each experiment (see the text box below).

Consistency

There is a famous line from the poet Ralph Waldo Emerson that is frequently misquoted as "Consistency is the hobgoblin of little minds." People sometimes use this incorrect version of the quote to belittle people for being consistent. However, the actual quote is "A foolish consistency is the hobgoblin of little minds" (Emerson, 1841, p. 14; my emphasis). It's not the least bit foolish to be consistent about your data organization, your filenames, your variable names, etc. You will save yourself huge amounts of time and grief by developing a good organizational strategy early in your career and then sticking to it (but with thoughtful changes when necessary).

This page titled 11.11: Exercise- Looping Through Data from Multiple Participants is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 11.11: Exercise- Looping Through Data from Multiple Participants by Steven J Luck is licensed CC BY 4.0.

11.12: Rapid Cycling Between Coding and Testing

If you think about the exercises we've done so far, you'll realize that we started with an extremely simple script (**Script1.m**) that simply loaded a file. We then improved the script slightly (**Script2.m** and **Script2b.m**), and then added a bit more functionality (**Script3.m** and **Script4.m**). This is a good way to learn, but it's also the way you should write your own scripts. Don't try to write 30 lines of code (or an entire script) without doing any testing along the way. If you write a 30-line script, you will probably have 8 different errors in the script, and it will be really hard to figure out what's going wrong.

Instead, the best approach is to write a small amount of code, test it, fix it if you find errors, and then add more code. At this point, you should be adding only 1-3 lines of code at a time. As you gain experience, you can write more lines before each test, but even an experienced programmer usually does some testing after every 20-40 new lines of code.

I almost put this piece of advice in a text box, but I decided that it's so important it deserves its own section!

Many studies have found that even experienced programmers spend 30-50% of their time finding and fixing errors in their code (i.e., debugging) rather than generating new code. And the proportion is even higher in beginning programmers. If you follow the advice I give in this chapter (like testing your code after adding 1-3 lines and defining all of your variables at the top of each script), you will spend a little more time generating new code but will save yourself a ton of time (and a lot of hair-pulling aggravation) on debugging.

This page titled 11.12: Rapid Cycling Between Coding and Testing is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

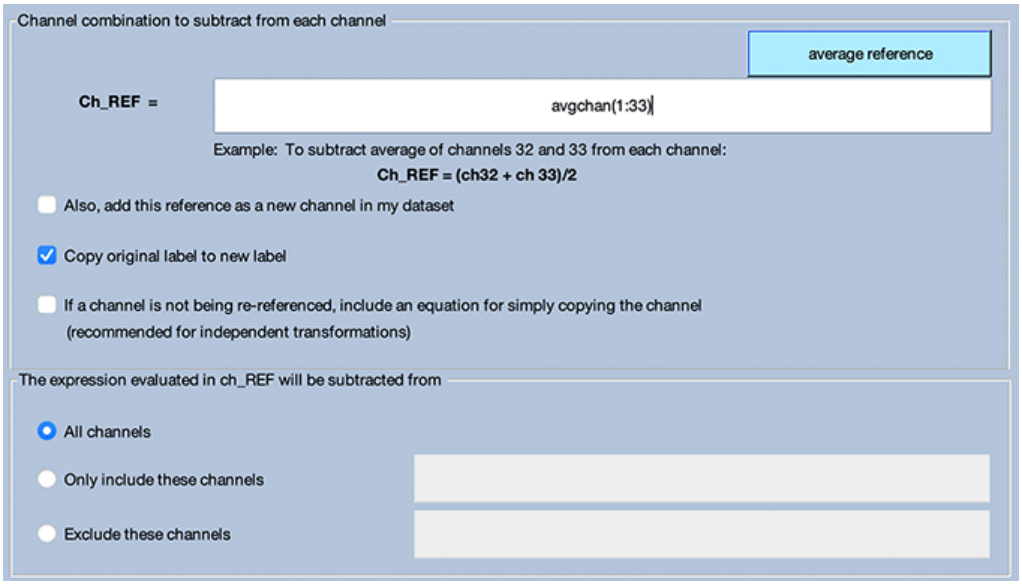
- 11.12: Rapid Cycling Between Coding and Testing by Steven J Luck is licensed CC BY 4.0.

11.13: Exercise- Using a Script to Reference EEGsets

In this exercise, we'll add referencing to the script from the previous exercise so that the new script is more useful. You'll also learn some important strategies along the way.

As you may recall, the ERP CORE data were recorded without a reference (which is possible in only a few recording systems), so we need to use EEG Channel Operations to reference the data. We'll also add bipolar VEOG and HEOG channels. The N170 is traditionally referenced to the average of all sites (see Chapter 5), and that's what we'll do in this example.

You should already have 9 datasets loaded from the previous exercise. If you don't, run **Script4.m** again and click **Refresh** in the **EEGsets** panel. With the **10_N170** dataset active, go to the **Channel Operations** panel. Clear out any equations that are already set in the panel's text box, set the **Mode** to **Create new dataset**, and make sure the box labeled **Try to Preserve Channel Locations** is checked. Click the **Reference Asst** button, and enter the parameters shown in Screenshot 11.6.



Screenshot 11.6

The key is specifying **avgchan(1:33)** in the text box at the top, which tells it to use the average of all 33 channels as the reference. Then click **OK**, which should fill the equations window with an equation for referencing each of the 33 channels to the average of Channels 1–33. Then add these two equations to create bipolar HEOG and VEOG channels:

nch34 = ch31 - ch32 Label HEOG-bipolar

nch35 = ch33 - ch16 Label VEOG-bipolar

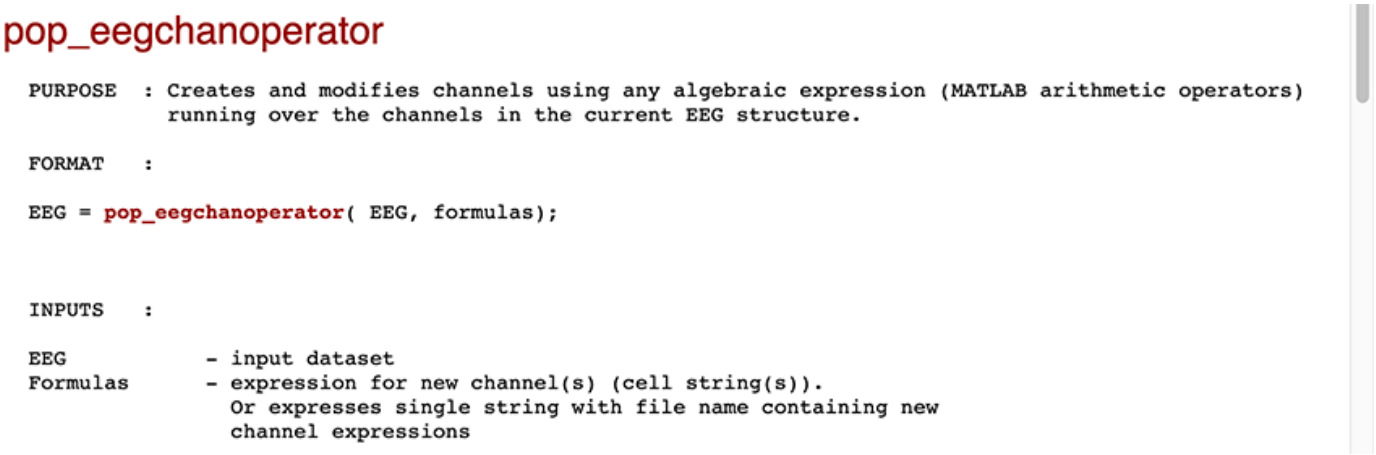
Then click **Run** to execute the routine and name the new EEGset **10_N170_ref**. Now type **eegh** in the Command Window. Near the end, you will see the script command for the Channel Operations function, which begins with **EEG = pop_eegchanoperator**. This command is extremely long because it contains all 35 equations in it. There's a better way to do this: Instead of having this huge command, we can put all the equations in a file and pass the filename instead of the list of equations.

To see how this works, select the original version of **10_N170** from the **EEGsets** menu, go back to the **Channel operations** panel, and click the **Advanced** button. You should see the 35 equations from before. Near the bottom of the window, click the **Save list as** button to save the list of equations as a file. Name the file **equations.txt**. In addition to saving the file, this command puts the filename (including the entire path) in the **File:** text box. Now, select the check box labeled **Send file rather than individual equations**, and click **OK** to go back to the panel. Then click **Run** to execute the routine. Now when you type **eegh** on the command line, you should see a much shorter line of code that looks something like this:

EEG = pop_eegchanoperator(EEG, '/Users/luck/Chapter_11/equations.txt' , 'ErrorMsg', 'cw', 'KeepChLoc', 'on', 'Saveas', 'off');

Let's take a closer look at this command. We send the **EEG** variable as the first parameter. The routine does the channel operation computations and then sends back a new version of this variable as its output. The second parameter is the filename for the equations file (including the whole path, which will be different on your computer). Then we have a sequence of pairs of parameters, which occur in the order *parameter name*, *parameter value*. For example, the sequence of parameters **'ErrorMsg', 'cw'** tells the routine to set any error messages to the Command Window. The next pair of parameters is **'KeepChLoc', 'on',** which tells the routine that the option for keeping the channel locations should be on (which you had set from the GUI by checking the option labeled **Try to preserve channel locations**). The use of pairs of parameters—one for the name of the parameter and one for the value—makes it easier to understand what the parameters are doing. This approach is common in Matlab scripts and functions, and it's used extensively in EEGLAB and eStudio routines.

How do you know what options are available for a command like **pop_eegchanoperator**? You can use Matlab's **help** feature. There are several ways to access this feature. One common way is to type **help pop_eegchanoperator** on the command line. Another way is to select the name of the routine in the Command Window (or in a script) and then right-click or control-click on it to get a contextual menu, which contains a **Help on Selection** item. Try one of these methods to see the help for **pop_eegchanoperator**. When you do this, you should see something like Screenshot 11.7. When displaying the help for a function. Matlab simply shows you all of the comments (lines of text beginning with the **%** character) at the top of the **.m** file containing that function (e.g., the **pop_eegchanoperator.m** file, which contains the code for this routine and is located in a subfolder deep inside the EEGLAB folder that was created when you installed EEGLAB and ERPLAB). This works with any command, including built-in Matlab commands like **pwd** and EEGLAB and ERPLAB commands like **pop_eegchanoperator**.



pop_eegchanoperator

```
PURPOSE  : Creates and modifies channels using any algebraic expression (MATLAB arithmetic operators)
           running over the channels in the current EEG structure.

FORMAT   :

EEG = pop_eegchanoperator( EEG, formulas);


INPUTS   :

EEG           - input dataset
Formulas      - expression for new channel(s) (cell string(s)).
                Or expresses single string with file name containing new
                channel expressions
```

Screenshot 11.7

Now that you've learned about the command for EEG Channel Operations, let's add it to the script we started in the previous exercise. Open **Script4.m** (if it's not already open), and save it as **MyScript4.m**. Copy the **pop_eegchanoperator** line from **eegh** history in the Command Window to the clipboard and then paste it into **MyScript4.m** right after the line that begins with **EEG = pop_loadset** (right before the line with **eeg_store** on it). Make sure you've pasted the version that uses the **equations.txt** file. Then save the updated **MyScript4.m** file. (I always save a script before running it—if there's an error that crashes Matlab, I don't want to lose the new code that I just added.)

Now quit eStudio, type **clear all**, and run **MyScript4.m** to see it work. You should see EEGLAB launch and then lots of text appear in the Command Window (mainly from **Channel Operations**), including a line that says something like **Processing Subject 9 of 9, ID = 10** as each individual subject is being processed. Once the script finishes, you should click the Refresh button in the EEGsets panel. If you look at the waveforms, you should see that you now have bipolar VEOG and HEOG channels.

If you run into an error, make sure that you followed the above instructions carefully, and think about the logic of what you're doing. You can also consult the Troubleshooting Tips in Appendix 2. The next several sections of this chapter will require you to make various additions to **MyScript4.m**, and you'll almost certainly make at least one error at some point. When you do, feel free to make a few unkind comments about my ancestors, and then remember that errors are a big part of scripting and that one of the goals of this book is for you to learn how to find and fix problems. Appendix 2 ends with some recommendations for writing code in a way that reduces the likelihood of errors.

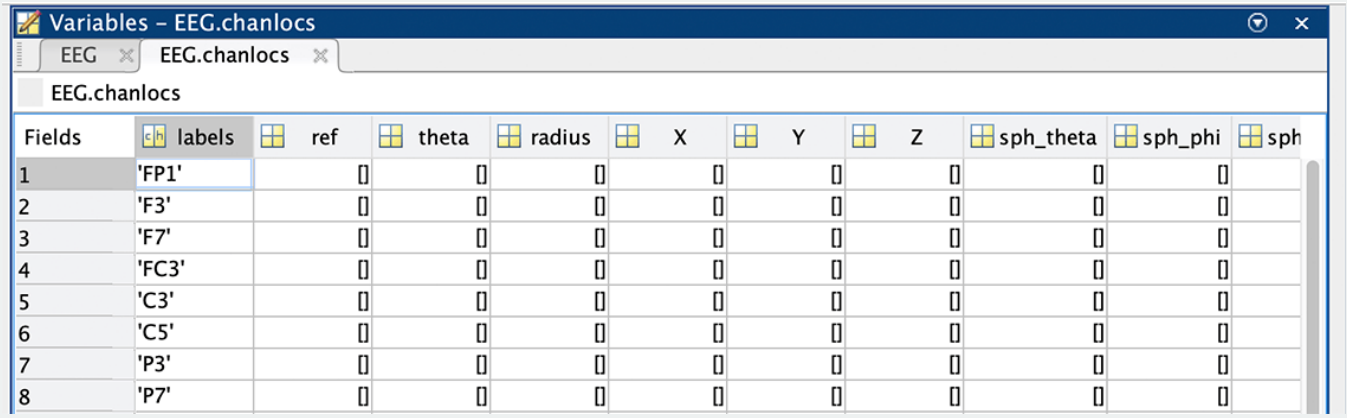
After you get the script to run correctly, I want you to spend a moment thinking about how easy it was to add the step for referencing the data to the prior script. Once you had the basic script for looping through the participants and opening their

datasets, you just had to run EEG Channel Operations from the GUI, get the history, and copy the command from the history to the appropriate point in the script. Done!

Well, not so fast. You'll usually want to make a few changes to the command that you copied from the history. For example, the new line of code you added to the script broke my rule about values (such as filenames) being defined as variables at the top of the script. If you moved this script and the **equations.txt** file to a new folder, the script would break. The next exercise will solve this problem and make some additional improvements.

Seeing a List of the Channels in a Dataset or ERPset

When you're using the eStudio GUI, you can easily see what channels are in an EEGset by looking at the **Channel and IC Selection** panel. However, this information isn't so obvious when you're writing scripts. Here's a little trick that I use quite often: Double-click on the **EEG** variable in Matlab's Workspace pane to open it in the Variables pane, and then double-click on the **chanlocs** (channel locations) field in the Variables pane. This also works for ERPsets if you double-click on the **ERP** variable.



Fields	labels	ref	theta	radius	X	Y	Z	sph_theta	sph_phi	sph
1	'FP1'	[]	[]	[]	[]	[]	[]	[]	[]	
2	'F3'	[]	[]	[]	[]	[]	[]	[]	[]	
3	'F7'	[]	[]	[]	[]	[]	[]	[]	[]	
4	'FC3'	[]	[]	[]	[]	[]	[]	[]	[]	
5	'C3'	[]	[]	[]	[]	[]	[]	[]	[]	
6	'C5'	[]	[]	[]	[]	[]	[]	[]	[]	
7	'P3'	[]	[]	[]	[]	[]	[]	[]	[]	
8	'P7'	[]	[]	[]	[]	[]	[]	[]	[]	

This page titled 11.13: Exercise- Using a Script to Reference EEGsets is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 11.13: Exercise- Referencing with a Script by Steven J Luck is licensed CC BY 4.0.

11.14: Exercise- Improving the Script and Adding 3D Channel Locations

In this exercise, we're going to make some improvements to the script from the previous exercise, including defining a variable for the name of the equations file, adding 3D electrode locations to each channel, and saving the EEGset to your hard drive.

Do You Need to Launch eStudio?

Before we get into making improvements to the script, I'm going to show you something that can save you a lot of time when you're testing a script. The first thing that **MyScript4.m** does when you run it is launch eStudio. But if eStudio is already running, do you really want to spend the time to launch it again? This can get very tiring. I know, because my original version of the script launched eStudio every time, even if it was already running, and it got very annoying after the 10th time or so. I therefore decided to show you a trick for seeing if it is already running.

Make sure that **MyScript4.m** is still open. Find the line near the beginning that launches eStudio with the **estudio** command, and replace that line with the following set of lines:

```
h = findobj('Tag', 'EStudio'); % Find the eStudio window (if it exists)
if isempty(h) % Check to see if the eStudio already exists
   disp('Launching eStudio');
   estudio;
else % Do the following steps if the eStudio window does not exist
   disp('eStudio is already running -- resetting EEG, ALLEEG, and CURRENTSET.');
   EEG = [];
   ALLEEG = [];
   CURRENTSET = 0;
   % You will still see the previous EEGsets in eStudio until you click the Refresh button in the EEGsets panel
end
```

The first of these lines (beginning **h = findobj**) takes advantage of the fact that, when eStudio launches, it creates a window with the tag **EStudio**. The **findobj** function in this line looks for a window with this tag, storing information about the window in the variable **h**. The next line uses the **isempty** function to see if **h** actually contains anything (i.e., if it is an empty variable). If it's empty, this implies that eStudio is not running, so the script launches it with the **estudio** command (after informing you with the **disp** command). Otherwise ("else"), it informs you that eStudio is already running and then resets the **EEG**, **ALLEEG**, and **CURRENTSET** variables. **EEG** and **ALLEEG** are set to empty arrays. **CURRENTSET**, which indicates which EEGset is currently selected, is set to zero. If we didn't reset **EEG**, **ALLEEG**, and **CURRENTSET**, the script would add onto any EEGsets that were already loaded, which is not what we want in this situation.

To test this, quit from eStudio and type **clear all**. Make sure you have saved your changes to **MyScript4.m** to disk, and then run the script. Because eStudio is not running, the first thing you should see is the message **Launching eStudio**, which indicates that **h** was empty (meaning that Matlab couldn't find the eStudio window) and the line **disp('Launching eStudio');** was therefore executed. You will then see some text indicating that eStudio is launching. When eStudio finishes launching, the rest of the script will run and load all the EEGsets. If you click **Refresh** in the **EEGsets** panel, you'll be able to see the EEGsets.

Now go back to your script, select the first 16 lines (up through the **end** statement in the text you added to the script), and execute these lines (e.g., by control-clicking and telling the popup menu to evaluate the selection in the Command Window). You should see this message in the Command Window: **eStudio is already running -- resetting EEG, ALLEEG, and CURRENTSET.** And if you look at the EEG, ALLEEG, and CURRENTSET variables in Matlab's Workspace pane, you can see that they've been reset.

Now clear all the EEGsets from the **EEGsets** panel and run the whole script. It will print that same message again in the Command Window and then load all the EEGsets. As usual, you need to click **Refresh** in the **EEGsets** panel to see the EEGsets.

Using a Variable for the Equations Filename

Our next step will be to store the name of the file with the Channel Operations equations in a variable at the top of the script. This will make the script more robust.

Make sure that **MyScript4.m** is still open. After the set of lines that define the various folders and filenames, add this line:

```
Equations_filename = [DIR 'equations.txt'];
```

That line will create a variable that holds the path and filename of the file with the equations. It's really easy to make an error when you create a variable like this. In fact, I initially made an error when I wrote that line of code. When you create a new variable for a path and filename, you should check it first before running your script. To do this, run that one line of code. (The **DIR** variable should already be set from when you ran the script earlier.) Then, type **dir(Equations_filename**) on the command line. The **dir** command lists any files that match the variable you specify inside the parentheses. When you run this command, you should get an error message saying that the file is not found.

Now type **Equations_filename** on the command line to see the value of this variable. Do you see the problem? We're missing the slash between the path and the filename (or a backslash on a Windows computer; see the text box below). Fix this by changing the code to:

```
Equations_filename = [DIR filesep '/equations.txt'];
```

filesep is a Matlab function that returns whatever file separator is appropriate for the operating system you are using (backslash for Windows, forward slash for Mac OS, UNIX, and Linux). You can see this by typing **filesep** in the Command Window. You'll see that it returns '/' if you're running Mac OS, UNIX, or Linux, or '\' if you're running Windows.

Now run the new line of code for defining the **Equations_filename** variable, and then run the **dir(Equations_filename**) command. If **Equations_filename** is set correctly, the **dir** command should just print the name of the file (**equations.txt**).

Now replace the string holding the path and filename in the **pop_eegchanoperator** command with **Equations_filename** so that the command looks like this:

```
EEG = pop_eegchanoperator( EEG, Equations_filename , 'ErrorMsg', 'cw', 'KeepChLoc', 'on', 'Warning', 'on', 'Saveas', 'off'  );
```

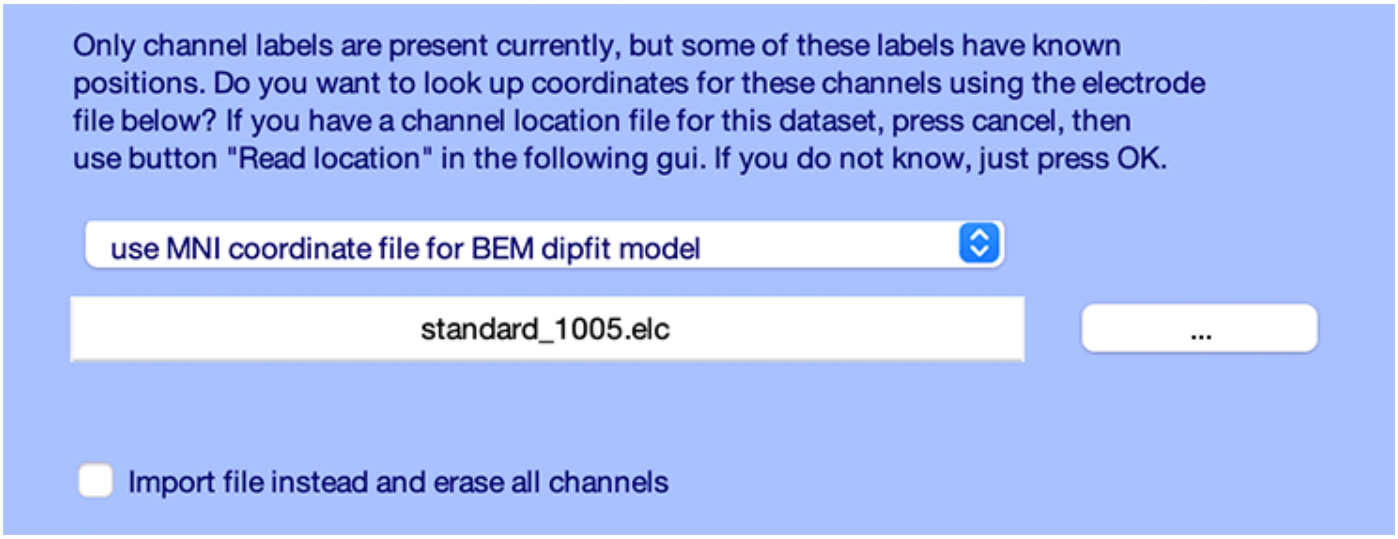
Now run the updated script and click **Refresh** in the **EEGsets** panel. It should lead to the same result as the previous version of the script that didn't use the **Equations_filename** variable, but the script is now more robust.

Now we're going to make a small improvement that will make your life easier someday. Here's the problem we want to prevent: when a script is running, you don't usually want error and warning messages to appear as new windows that wait for you to respond. If you're running a script overnight because it takes a long time, you don't want the script to be waiting for a response to a warning message when you arrive the next morning. So, let's change **'Warning', 'on'** to **'Warning', 'off'** so that warning messages aren't printed. (But note that error messages will still be printed.) Go ahead and save the updated script.

Adding 3D Electrode Locations

Now you are going to see how to add 3D electrode location information to each channel. This information will be essential when you want to interpolate bad electrodes (because interpolation creates a new value at the bad electrode from a distance-weighted average of the other electrodes, and electrode locations are needed to get the distance information). It will also be essential if you want to plot scalp maps of your EEG or ERP data.

Select **1_N170** in the **EEGsets** panel, go to the **EEGLAB Tools** panel, and click the **Chan locations** button. This will launch a window like the one shown in Screenshot 11.8. The general idea is that EEGLAB has a file that contains a list of all the standard electrode names (from a modified version of the International 10/20 system) and the expected 3D location of each location on an average head. The routine will go through all of the electrode names in your EEGset, try to find the corresponding names in its file, and then add the 3D location information to the channel for each electrode name that it can find. If it can't find one or more of your electrode names, those channels won't get a 3D location.



Screenshot 11.8

The default file with the name-location pairs is **standard_1005.elc**, which you can see in the window shown in Screenshot 11.8. I prefer to use a different file, which is named **standard-10-5-cap385.elp** and is located in the **Chapter_11** folder. To specify this file, click on the button with the three dots in it, and select **standard-10-5-cap385.elp**. Then click the **Ok** button, which will cause a very complicated new window to appear. This window allows you to adjust the locations. No adjustments are necessary, so just click the **Ok** button for this window. You'll then see the standard eStudio window for naming and saving the new EEGset that is being created. You can just accept the default name.

Now type **eegh** in the Command Window to see the history. The last few lines should contain the relevant line of code, which should look something like this (but with a different path to the file):

```
EEG=pop_chanedit(EEG, {'lookup','/Users/luck/Chapter_11/standard-10-5-cap385.elp'});
```

You can copy this line from the history into your script, right after the line that begins **EEG = pop_eegchanoperator**. However, this violates our rule about having information like paths and filenames in the middle of a script. In addition, because this line of code includes the path to the file, it will break if you ever move your experiment to a different location on your computer. To make it more robust, we want to use the **DIR** variable we created earlier that represents the current folder (even if you later shift everything to a new folder). To solve these problems, add the following near the top of your script (under the line defining the **Equations_filename** variable):

```
ElectrodeLocation_file = [DIR filesep 'standard-10-5-cap385.elp'];
```

Then, change the **pop_chanedit** line to this:

```
EEG = pop_chanedit(EEG, {'lookup', ElectrodeLocation_file});
```

Now your script has defined the name of the file in a variable at the top, with the location being whatever your current folder is. If you move everything to a new folder, the script will still work as long as the folder holding the script and data is the current folder when you run the script.

Save your script, run it, and then click **Refresh** in the **EEGsets** panel. To see the channel locations, you can go to the **Edit/Delete Channels & Locations** panel and click **Add or edit chan locations**. You should see a new window with an image of the electrode locations superimposed on a head, with the location information in a table on the right side of the window. Once you've verified that everything looks okay, you can close this window.

Naming and Saving the EEGset

Our next step will be to update the name of the EEGset that we've modified to indicate that it has been referenced and that channel locations have been added. Then we'll save the EEGset to disk.

First, type **EEG.setname** in the Command Window so that you can see the name of the current EEGset. If Subject 10 is the active EEGset, you should see that the setname is **'10_N170'**. To update the setname, add the following line to your script after the line with the **pop_chanedit** command:

```
EEG.setname = [EEG.setname '_ref_chanlocs'];
```

This takes the current setname and appends **_ref_chanlocs** to the end, storing the result in the **EEG.setname** variable. That was pretty easy. Note that you should have added this line right before the line with the **eeg_store** command, so the new setname will be stored in **ALLEEG** and therefore appear in the **EEGsets** panel once you click the **Refresh** button. This line of code violates my rule about using variables to define values, because I didn't define a variable for **'_ref_chanlocs'**. Later, in the script named **Script4b.m**, you'll see that I have fixed this problem by using a variable named Setname_suffix.

Now let's add some code for saving the updated EEGset to your hard drive, in the same folder as the original EEGset. You should first use the GUI to save the current EEGset on your hard drive so that you can use the history to see the command for saving an EEGset file. To do this, go to the **EEGsets** panel, click the **Save a Copy** button, and use the suggested name for the EEGset and filename (**10_N170_ref_copy**). Now type **eegh** to see the history. You should see something like this:

```
EEG = pop_saveset( EEG, 'filename','10_N170_ref_copy.set','filepath','/Users/luck/Chapter_11');
```

Copy that line from the Command Window and paste it into your script after the line with the **eeg_store** command. But we want to make some changes before we use it. First, we don't want the file to be named **10_N170_ref_copy.set**. I like to have the filename be the same as the setname, but with **.set** at the end. To accomplish this, change the new line of code by replacing **'tmp.set'** with **[EEG.setname '.set']** (which creates a string with the setname followed by **.set**).

We also want to change the path so that it uses the **Subject_DIR** variable we previously created to hold the path for this subject's data. To do this, change the string that currently lists the path (which is **'/Users/luck/Chapter_11'** on my computer but will be something else on your computer) to **Subject_DIR**. This line of code should now look like this:

```
EEG = pop_saveset( EEG, 'filename', [EEG.setname '.set'], 'filepath', Subject_DIR);
```

Before you run the code, I want to show you one more little trick for dealing with bugs and other kinds of errors. I probably made an error at least 50% of the time when I added code to one of the scripts for this book. If you have a script that loops through the data from 40 participants, it can be really annoying to have the script show an error message 40 times before it finishes. You can usually type ctrl-C to stop the code after the first error, but that doesn't always work. So, the first time I try out a new script, I just try it with the data from the first participant. I do that by following the line that defines the subject IDs with another line that lists only the first subject ID. For example, near the top of MyScript4.m, I have these two lines:

```
SUB = [ 1 2 3 5 6 7 8 9 10 ]; %Array of subject numbers

%SUB = [ 1  ]; %Just the first subject
```

The first time I run the script after making a change, I simply remove the **%** symbol on the second of the two lines. This overwrites the original contents of **SUB** and replaces it with only the ID for Subject 1, and the loop runs only for this one subject. Once I get the script to work for that one subject, I then put the **%** symbol back and run the script again.

To see this in action, add the line with **SUB = [1];** (without the **%** symbol) right after the line that currently defines the **SUB** variable. Now try running your updated version of **MyScript4.m**. If you get an error, you can fix it and try again.

Once you get it to run, you will notice that nothing seems to have changed in the EEGsets panel. That's because eStudio doesn't know that you've updated the contents of **ALLEEG**. To get eStudio synced with the current contents of **ALLEEG**, you need to click **Refresh** in the **EEGsets** panel. Then you should see an EEGset only for Subject 1, and it should be named **1_N170_ref_chanlocs**.

Now that the script is working with Subject 1, you can comment out the **SUB = [1];** line (i.e., add the **%** symbol at the beginning of the line) and run the script again to process all the subjects. Once the script finishes, you'll need to click **Refresh** in the **EEGsets** panel to see the EEGsets for all the subjects.

Now you have a robust, well-designed script that launches eStudio if necessary, loads the EEGsets, references the data, adds the 3D channel locations, and saves the updated EEGsets to your hard drive. You can now open **Script4b.m** to see my version of this script. It has comments that explain each line of code.

Script4b.m also uses a slightly different version of the equations file, named **Reference_Equations.txt**, which reorders the channels in addition to referencing them. The order of the channels in the original data files reflects the order of electrodes in the BioSemi recording system we used to collect the data. This ordering is convenient for the process of placing the electrodes on the head, but not for viewing the data. I reordered the channels so that they are in sets that go from left to right for a given line of electrodes (e.g., F7, F3, Fz, F4, F8 for the Frontal channels).

This page titled 11.14: Exercise- Improving the Script and Adding 3D Channel Locations is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 11.14: Exercise- Improving the Referencing Script by Steven J Luck is licensed CC BY 4.0.
- 11.15: Exercise- Preprocessing the EEG and Using a Spreadsheet to Store Subject-Specific Information by Steven J Luck is licensed CC BY 4.0.
- 11.13: Exercise- Referencing with a Script by Steven J Luck is licensed CC BY 4.0.

11.15: Exercise- Preprocessing the EEG and Using a Spreadsheet to Store Subject-Specific Information

As described in earlier chapters, I strongly recommend that you go through each step of EEG processing manually, looking at the data from each step, before running a script. Actually, I don't merely recommend it—I insist on it! This will allow you to catch problems that would otherwise contaminate your data. It will also let you determine some subject-specific parameters, such as which channels should be interpolated and what artifact detection parameters should be used.

But if some parameters differ across subjects, how can you use one script that processes the data from all the subjects? The answer is to store the subject-specific parameters in a spreadsheet, and then have your script read the parameters from that spreadsheet. **Script5.m** will show you how this works in the context of interpolating bad channels and other preprocessing steps. The interpolation process won't work well if there are big voltage offsets in the channels used to interpolate the bad channels, so the script also applies a high-pass filter that eliminates these offsets prior to interpolation.

Let's take a look at **Script5.m** before running it. In the list of variable assignments near the top of the script, you'll see a variable named **Interpolation_filename** that holds the name of an Excel spreadsheet, **interpolate.xlsx**. This spreadsheet contains information about which channels are bad and should be interpolated. Take a look at the file in Excel (or import it into Google Sheets or some other spreadsheet program). Here's what you should see:

Table 11.2. Spreadsheet of information for interpolation (interpolate.xlsx).

ID	Bad_Channels	Ignored_Channels	Channel_Names
1	[6 13]	[31 32 33]	C5, Oz
2		[31 32 33]	
3	[25]	[31 32 33]	P8
4		[31 32 33]	
6		[31 32 33]	
7		[31 32 33]	
8		[31 32 33]	
9		[31 32 33]	
10		[31 32 33]	

The first column contains the Subject ID values (without Subject 5, whom we're still excluding). The second column indicates the bad channels (if any) corresponding to the Subject ID values. They're specified using square brackets so that they will be interpreted as arrays by Matlab. You'll see why that's important in a bit. The next column indicates which channels should be excluded when we compute the interpolated values (the EOG channels). Those are the same for each participant, so they could be listed in the script, but I like to put them in the spreadsheet in case they need to vary across participants. The last column shows the names of the bad channels. This column isn't used by the script, but it's nice to have that information when you're looking at the spreadsheet.

If you look closely at the contents of the cells (e.g., by clicking on a cell and looking at its contents in the Formula Bar), you'll see that the value for each cell begins with a single quote (except for the ID values). This tells Excel that the contents of that cell should be treated as a text string and never interpreted as a number. I've found that this can avoid problems when we read the values into Matlab, because we want every cell in a column to be the same data type. You should also note that the column labels don't have any spaces or special characters in them (except for the underscore character). The Matlab routine we'll use to read the spreadsheet will use the column labels as variable names, so they must be legal Matlab variable names.

Now look at the script again and find the line with the **readtable** command, which should look like this:

```
interpolation_parameters = readtable(Interpolation_filename);
```

This is a Matlab function that reads data from a file into a special *Table* data structure. It's a very powerful function that can read from many different file types. It uses the filename to determine what kind of file is being read (e.g., the **.xlsx** filename extension is used to indicate that it's an Excel XML file). It creates a Table structure from the data file, and we've told it to store this Table in a variable named **interpolation_parameters**.

To see how this works, run the first part of the script, starting with the **DIR = pwd** line and going through the **interpolation_parameters = readtable(Interpolation_filename)** line. Then double-click on the **interpolation_parameters** variable in the Workspace pane so that you can see the contents of this variable. You'll see that it contains the same rows and columns that were in the spreadsheet. Now type **interpolation_parameters.Bad_Channels** on the command line. You'll see a list of the bad channels for each subject:

```
{'[6 13]'}
{0×0 char}
{'[25]'  }
{0×0 char}
{0×0 char}
{0×0 char}
{0×0 char}
{0×0 char}
{0×0 char}
```

You can see that this list is in a special Matlab-specific format called a *cell array*. Cell arrays are a little difficult to understand and tricky to use correctly. This is especially true for beginners, but I still often make mistakes when I try to use them (and then I often ask an AI system like ChatGPT for help). You'll need to learn about them someday, but for now you can rely on code in **Script5.m** that extracts the contents of the **interpolation_parameters** table into a set of simple numeric arrays.

Let's execute the code, but without actually going through the whole loop. To do this, first type **subject = 1** on the command line so that the looping variable has the correct value for the first subject. Then execute the **ID = num2str(SUB(subject))** line in the body of the loop, because we're going to need this variable. Now execute the three lines of code beginning with **table_row =**. The first of these lines determines which row of the **interpolation_parameters** table contains the values for this subject. The second line gets the array of bad channels for this subject and stores it in a variable named **bad_channels** (which has zero elements if there are no bad channels). The third line gets the array of to-be-ignored channels and stores it in a variable named **ignored_channels**. If you're not already an experienced Matlab programmer, the code on those lines probably looks like hieroglyphics—as I said, cell arrays are a little complicated. Once you're more familiar with Matlab coding, and you've wrapped your brain around cell arrays, you should come back to this code and figure out how it works. But for now, you can treat it like a bit of magic that gets you the information you need.

Now you should inspect the contents of **bad_channels** and **ignored_channels**, either by typing the variable names on the command line or looking at them in the Workspace. You'll see that **bad_channels** is an array with the values 6 and 13 (the two bad channels for Subject 1), and **ignored_channels** is an array with the values 31 through 33 (the three channels we will be ignoring

when computing interpolated values for the bad channels).

The next line of code does the interpolation using the **pop_erplabInterpolateElectrodes** function. You can see that we send the **bad_channels** and **ignored_channels** variables to this function. But don't run this line of code yet, because we haven't run all of the preceding lines in the body of the loop. Let's take a look at those lines before we run them.

The body of the loop begins by setting some variables and loading the EEGset, just as in the previous script. Then it runs a routine called **pop_erplabShiftEventCodes**, which shifts all the stimulus event codes to be 26 ms later. This is necessary because there is a fairly substantial delay between when an LCD display receives an image from the computer's video card and when it actually displays that image. We measured that delay using a photosensor and found that it was 26 ms in our system (but it might be quite different in your system). We therefore shift the event codes so that they occur at the actual time that the stimulus appeared instead of at the time when the image was sent to the display. If you're using an LCD display, you must measure the delay and shift the stimulus event codes. If you don't know how to measure the delay, contact the manufacturer of your EEG recording system.

The next step uses the **pop_basicfilter** function to run a bandpass filter with a bandpass of 0.1 to 30 Hz (12 dB/octave, which corresponds to a *filter order* of 2 in the code). Filtering out the low frequencies is essential prior to interpolation, because otherwise the random voltage offsets in each channel will produce a bizarre scalp distribution and the interpolation algorithm (which assumes a smooth scalp distribution) will produce bizarre results. Note that it is a good idea to remove the DC offset before filtering continuous EEG data, and this is implemented by specifying **'RemoveDC', 'on'** when we call the **pop_basicfilter** function.

The next step runs the **pop_chanedit** function to add information about the 3D location of each electrode site based on the electrode names, as described in the preceding section.

The next few lines extract the information from the **interpolation_parameters** table and run the interpolation routine.

Finally, we reference the data, name the EEGset, and save it to disk (just as in the previous exercise).

Note that, for each of these processing operations, we sent the EEGset to the processing function in the **EEG** variable, and then the routine returned a modified dataset that we stored in the **EEG** variable. In other words, the new EEGset overwrites the old EEGset in the **EEG** variable. That's much more efficient than storing the result of each new operation in **ALLEEG**. But note that keeping the individual EEGsets makes sense when you're processing the data in the GUI, because you want the flexibility of going back to a previous EEGset.

Now that we've looked at the code, let's run the script, but only for the first subject. To limit it to the first subject, make sure that the **SUB = [1]** line near the top isn't commented out. Then run the script. You can now see that the script has created a new EEGset file named **1_N170_shift_filt_chanlocs_interp_ref.set** in the **Chapter_11 > N170_Data > 1** folder.

However, if you have eStudio running, this EEGset isn't visible in the **EEGsets** menu. **Script5.m** differs from the previous example scripts in that it doesn't make the EEGsets available in the eStudio GUI. It just does the processing and saves the results for each participant in an EEGset file. As you'll see later in the chapter, you'll often have a series of scripts for processing a given experiment (e.g., one for the initial preprocessing, another for ICA, another for post-ICA EEG processing, and another for dealing with averages). Each of these scripts will read in the files created by the previous script, so there's often no need to make the EEGsets available in the GUI.

If you do want to look at the results of a given script, you'll want a convenient way of reading in the files that were just created. **Script5.m** accomplishes this by including code at the end for loading the new EEGset files into **ALLEEG** so that you can view them in eStudio. This code is preceded by a **return** command, which causes the script to terminate, so the code at the end won't ordinarily execute when you run the script. But after you run the script, you can just select the code at the end and run it manually. Give it a try, and then verify that you can see the new EEGset in the **EEGsets** panel (after you click the **Refresh** button).

Once you have that EEGset loaded into eStudio, also load the original EEGset (**1_N170.set**) for this participant from the **Chapter_11 > N170_Data > 1** folder. You'll see that the new EEGset is much smoother and doesn't have large DC offsets (because it has been bandpass filtered). And you can see some huge artifacts in the C5 and Oz channels in the original data that are gone in the new data (because the script interpolated those channels). You should also notice that the channels are in a more convenient order in the new EEGset than in the original EEGset as a result of the EEG Channel Operations step near the end of the loop.

Now comment out the **SUB = [1]** line near the top of the script and run the script again to process the data from all 9 participants.

You can then select and execute the code at the end of the script for loading the new EEGsets into eStudio. (Don't forget to click the **Refresh** button in the **EEGsets** panel.)

At this point, I'd like to remind you of some advice I gave you at the beginning of the chapter: Play! The best way to understand how these scripts actually work is to play around with the code. If you're not sure you understand one of the steps, try changing it to see what happens. But don't just do this randomly. Come up with hypotheses and test them by means of experimental manipulations (i.e., by changing the code and seeing if the results confirm or disconfirm your hypotheses). You can also try modifying the scripts to process your own data. Unless you're a very experienced programmer, you probably won't actually understand the key points from this chapter unless you engage in this kind of active exploration. And now is a great time to play around with the code, because the rest of the chapter assumes that you fully understand the basics and are ready to put them together into a complete data processing pipeline.

This page titled 11.15: Exercise- Preprocessing the EEG and Using a Spreadsheet to Store Subject-Specific Information is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 11.15: Exercise- Preprocessing the EEG and Using a Spreadsheet to Store Subject-Specific Information by Steven J Luck is licensed CC BY 4.0.

11.16: Exercise- Building an Entire EEG Processing Pipeline

You've now learned the basics of scripting, so we're ready to build an entire EEG preprocessing pipeline. This pipeline will execute all the steps prior to averaging. It closely matches the example pipeline described in Appendix 3. Here, we've divided the pipeline into five separate scripts:

- **Step1_pre_ICA_processing.m**- Performs the initial preprocessing steps prior to ICA-based artifact correction
- **Step2_ICA_phase1.m**- Creates an optimized EEGset for the ICA decomposition
- **Step3_ICA_phase2.m**- Performs the ICA decomposition
- **Step4_ICA_phase3.m**- Transfers the ICA weights from the optimized EEGset to the original EEGset and reconstructs the data without the artifactual ICs
- **Step5_post_ICA_processing.m**- Performs the steps following artifact correction that are necessary prior to averaging

I encourage you to divide your EEG and ERP processing into multiple scripts in this manner. A general principle of good programming is to divide a complex job into a set of small, independent modules. This makes each part of the job simpler and less error-prone. It also makes it easier for you to find and fix problems. And it makes it easier for you to reuse your code for future experiments. Ideally, none of your scripts should be more than about 200 lines long (including comments). If you find a script is getting a little long, try to figure out how to break it up into a sequence of smaller scripts.

Step1_pre_ICA_processing.m is similar to **Script5.m** from the previous exercise, but with two major changes. First, interpolation has been removed from this script and moved to a later stage, following artifact correction. Second, the new script references the data to O2 rather than to the average of all sites. This is because using the average of all sites as a reference makes ICA complicated (see the chapter on artifact correction). After the ICA step, we'll re-reference to the average of all sites. By the way, I could have used any site as the reference at this initial stage. See the following text box for more information.

Recognizing a Conceptual Error

When I first started writing these scripts, I referenced the data to the average of all sites at the beginning of the pipeline (because this is the standard reference for the N170). I had gotten through the stage of performing the ICA decomposition, and I started going through the data to determine which ICs should be removed (using the process described in the chapter on artifact correction). The ICs from the first participant looked okay but not great. The time course of one of the top ICs had a lot of weird high-frequency noise that I wasn't seeing in the EEG. The ICs from the second participant were even worse, with the top two ICs showing lots of high-frequency noise, and the eye movements distributed across three ICs. I was starting to get suspicious. When I looked at the ICs for the third participant, the blinks were spread across the top four ICs, and two of them again had a ton of weird high-frequency noise.

I then asked myself what I was doing differently from before, and then I realized that I was now referencing to the average of all sites prior to the decomposition. I then changed the scripts to use O2 as the reference, re-ran the ICA decomposition, and then everything worked better.

The moral of the story is that you may get occasional participants for whom the ICs don't look great, but if you see more than one or two, you need to think through your process and figure out what's going wrong. The reference is one possible problem. Another common problem is an insufficient recording duration (especially if you have >64 channels). A third common problem is huge C.R.A.P. that hasn't been eliminated prior to the decomposition.

Step2_ICA_phase1.m implements the procedures described in Chapter 9 for creating EEGsets that are optimized for the ICA decomposition, including downsampling to 100 Hz, eliminating breaks and other periods of huge C.R.A.P., and implementing an aggressive high-pass filter. It's similar to one of the example scripts at the end of Chapter 9 (**MMN_artifiact_correction_phase3.m**). Open **Step2_ICA_phase1.m** and take a look at it.

One very important element of this script is that it assumes you've already gone through the EEG to determine the parameters that you will use for finding huge C.R.A.P. with the **Artifact rejection (continuous EEG)** routine and stored these parameters in a spreadsheet named **ICA_Continuous_AR.xlsx**. I've already done this for you. To determine these parameters, I first commented out the part of the script that performs the continuous artifact rejection, and then I ran the code at the end of the script for loading the EEGsets into the eStudio GUI. These EEGsets have been downsampled and aggressively filtered, and I wanted to see

what the artifacts looked like in these EEGsets because they will be used for the continuous artifact rejection. Two of the subjects had some large C.R.A.P., and I ran the **Artifact rejection (continuous EEG)** routine from the GUI for these subjects to figure out the best rejection criteria. As discussed in the chapter on artifact correction, you really need to look carefully at the data when setting these parameters if you want ICA to work well. The spreadsheet also indicates which channels to include. I've left out the EOG channels and Fp1/Fp2 so that ordinary ocular artifacts don't get deleted. I've also left out any channels that will be left out of the ICA decomposition and interpolated later. We don't need to delete crazy periods in these channels, because they're being left out of the decomposition.

If you look at the code for reading these parameters from the spreadsheet, you'll see that it's much like the code for reading the parameters for interpolating bad channels in the previous exercise, except that we have different column labels in the spreadsheet.

You'll also see that the script calls the **pop_erplabDeleteTimeSegments** routine to delete the periods of time during the breaks, which also helps get rid of large C.R.A.P. Note that the parameters that control this routine are defined as variables at the top of the script, following good programming practice.

Once you've looked through the script to see how it works, go ahead and run it. You'll see that it creates a new EEGset file for each participant with **_optimized** at the end of the filename. You can also load the new EEGsets into the eStudio GUI by running the bit of code at the end of the script and then clicking **Refresh** in the **EEGsets** panel. This allows you to see what the optimized EEGsets look like and make sure everything worked properly.

Step3_ICA_phase2.m runs the ICA decomposition process on the EEGset created by **Step2_ICA_phase1.m**. It assumes that an Excel spreadsheet named **interpolate.xlsx** has already been created to indicate which channels will be interpolated after correction has been performed and should therefore be excluded from the ICA decomposition. I've already created this file for you. Note that the decomposition process is quite slow, so this script takes a long time to run.

Step4_ICA_phase3.m takes the ICA weights in the optimized EEGset and transfers them back to the pre-optimization EEGset. It then removes the artifactual ICs, which are listed in a file named **ICs_to_Remove.xlsx**. I had to determine which ICs were artifactual by looking at the IC scalp maps and by comparing the IC time courses with the EEG/EOG time courses (as described in the chapter on artifact correction). To make this easier, I used the bit of code at the end of **Step3_ICA_phase2.m** to load the EEGsets into the eStudio GUI.

I didn't spend a lot of time making careful decisions about which ICs to remove (and making sure that the ICA decomposition was truly optimal). For example, Subject 1 still has some blink activity remaining in F4 after the correction. It's really boring to spend many hours in a row getting the ICA perfect for a large set of participants! This is one more reason why you should do the initial preprocessing of each participant within 48 hours of data collection. It's a lot easier to spend the time required to optimize the ICA when you're only doing it for one participant at a time and don't have to spend an entire day processing the data from 20 participants.

The next script is **Step5_post_ICA_processing.m,** which performs the steps following artifact correction that must be executed prior to averaging. This includes re-referencing to the average of all sites (and putting the channels into a more useful order), performing interpolation for any bad channels, adding an EventList, assigning events to bins with BINLISTER, epoching the data, and performing artifact detection. The script also prints a summary of the overall proportion of trials marked for rejection in each participant and creates an Excel file with this information broken down by bin.

Open the script and take a look. You'll see that the first part of the script (prior to the loop) defines and opens a set of files with the artifact detection parameters. We've already corrected for blinks, so we only want to flag epochs with blinks that occurred at a time that might interfere with the perception of the stimulus. Eye movements aren't typically an issue in this paradigm because the stimuli are presented briefly in the center of the display, but we flag trials with eye movements that might interfere with the perception of the stimulus (which were quite rare). We use the uncorrected bipolar channels for the blink and eye movement detection. We also flag trials with large C.R.A.P. in any of the EEG channels (using both an absolute voltage threshold and a moving window peak-to-peak amplitude algorithm). Each of these artifact detection routines uses a different flag so that we can keep track of the number of trials flagged for each type of artifact.

Prefiltering prior to artifact detection

These scripts were written when I wrote the first edition of this book, and I made only a few changes for this edition that were necessary for the scripts to be suitable for ERPLAB Studio. In the time since I first created those scripts, I have realized that the prefiltering option is very helpful for artifact detection. This option makes a copy of the epoched EEG, applies a low-pass filter, performs artifact detection on this filtered copy, and then copies the artifact flags back to the original data. This avoids marking trials simply because they have some high-frequency noise (which isn't a problem for most research).

I thought about modifying **Step5_post_ICA_processing.m** to include this prefiltering. However, prefiltering means that you can use a lower rejection threshold, and I was simply too lazy to go back and figure out new custom thresholds for each participant. But I recommend that you use this option in your own scripts. It simply involves adding **'LowPass', 30,** to the **pop_artmwppth** and **pop_artextval** function calls, like this:

EEG = pop_artextval(EEG, 'Channel', channels, 'LowPass', 30, 'Twindow', test_period, 'Threshold', [-threshold threshold], 'Flag', flag_CRAP1);

Note that prefiltering is not necessary when using the step function (**pop_artstep**).

The top portion of the script also pre-allocates a set of arrays that will be used to store the number of accepted and rejected trials for each participant. This pre-allocation isn't strictly necessary, but it's good programming practice—it makes it clear what the dimensions of the arrays are.

The body of the subject loop loads the EEGset created by the previous script, interpolates any bad channels, re-references the data, creates the EventList, runs BINLISTER, and epochs the data. These steps are pretty straightforward.

The next set of lines reads the parameters for blink detection from a spreadsheet and then runs the step function algorithm to find blinks that occurred between -200 and +300 ms. These lines use the same "magic code" that I used to grab parameters from spreadsheets in the previous scripts. I didn't spend a lot of time setting the artifact detection parameters—you could do a better job if you spent some time using the strategies described in the chapter on artifact rejection. Once the parameters have been extracted from the spreadsheet, the artifact detection routine is called. We then repeat this process for the eye movement and C.R.A.P. artifacts.

Each of these artifact detection steps adds to the flags set by the previous step. At the end, we save the EEGset to the hard drive. It's now ready for averaging, which will be covered in the next exercise.

After the end of the loop, I added some code to grab information about the number of trials that were flagged for each participant. The code demonstrates how to save this information in Matlab's special Table format, which then makes it easy to save the information as a spreadsheet. This is useful because you should report this information in your journal articles (e.g., the proportion of trials marked for rejection, averaged across participants, along with the range across participants).

As usual, the end of the script has some code after the **return** statement that you can use to load the EEGsets into the eStudio GUI.

Whew! That's a lot of code. But I hope it shows you how to break a complex sequence of processing steps into a set of relatively simple modules. And I hope it also demonstrates the process of going back and forth between the eStudio GUI (to set various participant-specific parameters and make sure the data look okay) and scripts (which are much faster, especially when you need to reprocess the data multiple times).

One last thing: Once you've created a set of scripts that perform the different processing stages for your experiment, you can create a "super script" that simply calls the scripts for each stage. Then you can execute all the stages by calling this one script.

This page titled 11.16: Exercise- Building an Entire EEG Processing Pipeline is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 11.16: Exercise- Building an Entire EEG Processing Pipeline by Steven J Luck is licensed CC BY 4.0.
- 11.15: Exercise- Preprocessing the EEG and Using a Spreadsheet to Store Subject-Specific Information by Steven J Luck is licensed CC BY 4.0.

11.17: Exercise- Averaging, Quantifying Data Quality, and Plotting

The series of scripts in the previous exercise combine to produce an EEGset for each participant that is ready for averaging. In the present exercise, we'll compute the averaged ERPs, including calculating aSME for a custom time window of 110-150 ms (the measurement window for the N170). We'll then create a grand average across participants and plot it. Finally, we'll print a summary of the aSME values to the Command Window.

Go ahead and open the script (**Step6_averaging_and_SME.m**) and take a look at it. As usual, the script begins by defining a set of variables that will be used later in the script. It also opens a file named **ERPset_files.txt**. This file will be used when we make the grand average. When we made grand averages in previous chapters, we loaded all the single-participant ERPsets into ALLERP and then told the grand averaging routine which of these ERPsets to include in the grand average. However, it's sometimes more convenient just to work with one ERPSet at a time, save the ERPsets as files on the hard drive, and not keep all the ERPsets loaded in memory. In this case, we can send the grand averaging routine a list of the filenames of the ERPsets, which are themselves stored in a text file. This is what **ERPset_files.txt** is used for in the present script. I could have just loaded all the ERPsets into ALLERP, but I wanted to demonstrate this alternative approach and show you how to open a text file and write to it.

The main loop in the script begins by loading the EEGset that was created by the final preprocessing script (**Step5_post_ICA_processing.m**), which is all ready for averaging. The next few lines define the custom aSME time window. This information is stored in a data structure called a **DQ_spec** (*data quality specification*). This structure defines the time windows prior to obtaining the data quality measures and stores the data quality measures once they've been calculated. It's a little complicated, so ERPLAB provides a routine called **make_DQ_spec** to create it.

We use this routine to create a variable named **custom_aSME_spec**. There are potentially many different types of data quality metrics that can be stored in a **DQ_spec** variable. The first metric stored in a DQ_spec structure is a measure of the baseline noise. The second is the standard error of the mean at each individual time point. The third is the aSME. It's possible for you to define additional types, the most common of which is the bSME (bootstrapped SME). Here, we're going to use aSME, but we're going to specific a custom time window for the aSME calculation so that we get an aSME value that corresponds to our N170 measurement window (110–150 ms). We do this by finding out how many time windows have already been defined by default when we called **make_DQ_spec** and then adding a new one.

The next step is to call the averaging routine (**pop_averager**), sending it the **custom_aSME_spec** variable that we just created so that it will compute the aSME for our custom time window (along with the default time windows). This routine returns an ERPset that we store in the **ERP** variable. We then create a name for the ERPset and save the ERPset to a file on your disk. We also save the name of this file (including the path) in the **ERPset_files.txt** file so that we have it when we make the grand average later.

The **ERP** variable includes a field named **ERP.dataquality** that stores the data quality metrics that were calculated during the averaging process. The script shows how you can grab the aSME values from **ERP.dataquality** for the channel we will ultimately use to score the N170 amplitude (PO8, Channel 27). For each subject, we get an array of four aSME values for this channel, one for each of the four bins. We store this in a two-dimensional array named **aSME_custom_values,** which has one dimension for subjects and another dimension for bins. The aSME data inside **ERP.dataquality** are stored in a 3D array with dimensions of channels, time ranges, and bins, and we use a Matlab function called **squeeze** to convert this 3D array into the 1D array of values for each bin for the current subject:

```
aSME_custom_values(subject,:) = squeeze(ERP.dataquality(where_aSME).data(measurement_channel,custom_entry_index,:));
```

When you're first learning to write scripts in Matlab, you'll probably find that you frequently get confused about how arrays work and when you need to use a function like **squeeze** to obtain the desired results. I still sometimes get confused, and I often make mistakes when writing code that operates on complicated arrays. But I've learned how to interpret Matlab's error messages, and I often search the Internet for solutions (or ask an AI like ChatGPT). I also recommend getting a good Matlab book and spending some time learning the logic behind how Matlab operates on arrays. As I mentioned earlier, my lab uses a book called *Matlab for Behavioral Scientists* (Rosenbaum et al., 2014).

The aSME quantifies the data quality for the mean voltage within a given time window, so it's most directly useful when you're scoring amplitude using the mean voltage (which is how we scored N170 amplitude in the ERP CORE paper). If you're using

some other scoring method (e.g., peak amplitude, fractional peak latency), you need to use a more complicated method called *bootstrapping* to obtain the SME values. You'll also need to use bootstrapping if you'll be obtaining scores from difference waves or if you apply any other kind of processing to the ERP data after averaging but before scoring (e.g., filtering or channel operations). ERPLAB currently requires scripting to compute bootstrapped SME values, and the example scripts at the end of Chapter 10 demonstrate how to do this. Chapter 10 also contains a script that demonstrates how to obtain behavioral data.

After the main loop finishes, the script prints out the custom aSME values that we saved in the variable named **aSME_custom_values** for each participant. It also prints the mean across participants for each bin, along with the RMS (root mean square) across participants. I recommend taking a look at the aSME values for each participant and then looking at their EEG and ERPs to see if you can understand why some participants have worse (larger) aSME values than others.

Finally, the script makes a grand average using the filenames stored in **ERPset_files.txt**. It then plots the grand average using the **pop_plotERPwaviewer()** routine. This is the routine that is used to take whatever is plotted in the main ERP plotting region and plot it in a separate window (by going to the **Plotting Options** popup menu at the lower right of the plotting region and selecting **Create Static/Exportable Plot**). **pop_plotERPwaviewer()** is incredibly complicated, because there are zillions of options for how to plot an ERPset. In general, you can see how to use it by plotting things the way you want them in the main plotting region and then selecting **Show Command** in the **Plotting Options** popup menu. This will show the appropriate **pop_plotERPwaviewer()** command in the Command Window, as illustrated in Screenshot 11.9. You can then copy it into your script and modify it as needed. But this is still a bit challenging given the huge number of parameters.

```
%EStudio:Equivalent command:
ALLERP = pop_plotERPwaviewer( ALLERP, 1,  1, 1:4,  1:32, 'Blc', 'none', 'Box', [ 6 6], 'FigbgColor', [ 1 1 1], 'FigOutpos', [ 921.6 595.733],...
 'FigureName', 'grand_N170', 'GridposArray', [1  2  3  4  5  6;7   8   9  10  11  12;13  14  15  16  17  18;19  20  21  22  23  24;25  26  27  28  29  30;31  32   0   0   0   0],...
 'GridSpace', [1  20;1  20], 'History', 'gui', 'Labelcolor', [ 0 0 0], 'Labelfontsize',  12, 'Labeloc',...
 [ 21.0803 100 1], 'LabelsName', {'FP1' , 'FP2' , 'F7' , 'F3' , 'Fz' , 'F4' , 'F8' , 'FC3' , 'FCz' , 'FC4' , 'C5' , 'C3' , 'Cz' , 'C4' ,...
 'C6' , 'CPz' , 'P9' , 'P7' , 'P3' , 'Pz' , 'P4' , 'P8' , 'P10' , 'PO7' , 'PO3' , 'PO4' , 'PO8' , 'O1' , 'Oz' , 'HEOG-left' , 'HEOG-right' ,...
 'VEOG-lower' }, 'Legcolumns',  2, 'LegendFontsize',  12, 'LegendName', {'Faces, Correct' , 'Cars, Correct ' , 'Scrambled Faces, Correct' ,...
 'Scrambled Cars, Correct ' }, 'LegtextColor',  1, 'LineColor', [0.75           0           0;0.375       0.75           0;0       0.75       0.75;0.375          0       0.75],...
 'LineMarker', {'none' , 'none' , 'none' , 'none' }, 'LineStyle', {'-' , '-' , '-' , '-' }, 'LineWidth',...
 [ 1 1 1 1]', 'PLOTORG',  1:3, 'SEM',  0, 'TimeRange', [ -200 796], 'Transparency',  0, 'Xlabelcolor', [ 0 0 0], 'Xlabelfontsize',  12, 'Xtickdecimal',  0, 'XtickdisFlag',  1, 'Xticklabel',
  -16:4:8 );
```

Screenshot 11.9

If you take a look at the section of **Step6_averaging_and_SME.m** that contains the **pop_plotERPwaviewer()** function call, you'll see that I replaced most of the parameters in the function call with variables, which are defined in the lines right above the function call. Here are four of those variable definitions for parameters that control the scale and ticks for the X and Y scales:

Time_range = [-200 800]; % Time range to plot
Time_ticks = [-200 0 200 400 600 800]; % Time points for ticks along the X axis
Y_scale = [-10.0 15.0]; % Y-axis scale
Y_ticks = [-10 -5 5 10 15]; % Voltage values for ticks along the Y axis

These variables then replace the numbers and strings used as parameters in the **pop_plotERPwaviewer()** function call. This makes it much easier to understand how the function is working and to make changes. If you want your own scripts to plot ERP waveforms, I recommend starting with the code I wrote in this part of **Step6_averaging_and_SME.m** and modifying it as necessary for your own needs. Note that I defined these variables right above the **pop_plotERPwaviewer()** function call rather than at the top of the script. For a function this complicated, it really helps to see the variable definitions at the same time you're looking at the function call, so this is one of the rare times I violate my rule about defining all variables at the top of the script.

I recommend running the script and then playing around with the plotting parameters. Once you've run the script once, you can just re-run the lines of code for defining the relevant variables and running **pop_plotERPwaviewer()**.

By the way, it's also possible for a script to plot a period of EEG data using the **pop_ploteegset()** function. You can see how this function works by plotting an EEGset in the **EEG** tab and then selecting **Show Command** in the **Plotting Options** popup menu. This will show the appropriate **pop_ploteegset()** command in the Command Window. Fortunately, this function is much simpler than the **pop_plotERPwaviewer()** function.

This page titled 11.17: Exercise- Averaging, Quantifying Data Quality, and Plotting is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 11.17: Exercise- Averaging with a Custom aSME Time Window by Steven J Luck is licensed CC BY 4.0.
- 11.13: Exercise- Referencing with a Script by Steven J Luck is licensed CC BY 4.0.

11.18: Exercise- Scoring Amplitudes and Latencies and Performing Statistical Analyses

This exercise demonstrates the final stages of processing using a script named **Step7_scoring.m**. This script obtains amplitude and latency scores from each participant and creates the files you'll need to perform a statistical analyses. There are many ways we could analyze the N170 data, and here we will look at three simple approaches.

In the first approach, we'll measure the mean amplitude in the N170 latency range (110-150 ms) for the faces and cars in the left-hemisphere and right-hemisphere electrode clusters. We'll then perform a 2 x 2 ANOVA with factors of stimulus type (face vs. car) and electrode hemisphere (left vs. right). The N170 is typically larger for faces than for cars, and this effect is usually larger in the right hemisphere than in the left hemisphere. As a result, we would expect a main effect of stimulus type and a stimulus type x electrode hemisphere interaction. You'll need to perform this statistical analysis using your own statistics package. I used JASP.

Any differences between the faces and the cars in the first analysis could reflect differences in low-level features (e.g., luminance, spatial frequency) rather than differences between the face and car categories per se. To address this possibility, the experiment also presented phase-scrambled face and car images, which contain the same low-level features as the faces and cars (i.e., the same spatial frequency power spectrum) but are unrecognizable as faces or cars. In our second analysis, we'll make a face-minus-scrambled-face difference wave and a car-minus-scrambled-car difference wave. The ERP activity directly attributable to the low-level features should be subtracted away in these difference waves, and any differences between the two difference waves can be attributed to higher-level features.

The third analysis will be just like the second analysis, except that it will be performed on the peak latency of the N170 rather than the mean amplitude. Because peaks are easily distorted by high-frequency noise, we'll apply a stronger low-pass filter to the data prior to measuring the peak latency.

Statistics Toolbox

The script **Step7_scoring.m** uses the Matlab **ttest()** function, which is part of the Statistics and Machine Learning Toolbox. You can see what toolboxes are installed by typing **ver** on the Matlab command line. If you do not have this toolbox, I recommend consulting your IT department to see if you can get it for free as part of your license. You can also purchase it from The Mathworks.

If you cannot get this toolbox, the script has information about how to comment out the lines that use the **ttest()** function.

Go ahead and open the script (**Step7_scoring.m**) and take a look at it. As usual, the script begins by defining a set of variables that will be used later in the script. It also opens a couple files that we'll use to store the names of the ERPset files. We'll send these files to the Measurement Tool so that it knows which ERPsets to use for scoring the ERPs (just like we did with the grand averaging routine in the previous exercise).

The main loop loads the ERPsets created by the script in the previous exercise. It then uses ERP Channel Operations (the **pop_erpchanoperator** routine) to create a left-hemisphere cluster channel and a right-hemisphere cluster channel. This is pretty simple, so we just send the equations directly in the script rather than saving them in a file.

The next step is to create the difference waves using ERP Bin Operations (the **pop_binoperator** routine). It sends a file named **BinOps_Diff.txt** that contains the equations for making the difference waves. The channel and bin operations create updated versions of the **ERP** variable, and we save this ERPset to the hard drive. We also save the name of the ERPset in a file named **Measurement_files.txt**.

Then we apply a low-pass filter with a half-amplitude cutoff at 15 Hz and a slope of 48 dB/octave, which helps us measure the peak latency more precisely. The resulting ERPset is saved to the hard drive, and the name of the ERPset is saved in a file named **Measurement_15Hz_files.txt**. Note that this filter might distort the onset and offset times of the N170, and might even produce artifactual deflections before and after the N170, but these distortions will not impact our ability to score the peak latency.

After we loop through all the subjects, we close the two files that are used to store the ERPset filenames. Then we start the first analysis stage, in which we obtain the mean amplitude scores from the parent waveforms. This is achieved by calling the **pop_geterpvalues** routine, which is the script equivalent of the **Measurement Tool** panel. We send it the name of the file that holds the names of all the unfiltered ERPsets that we created in the loop so that it knows which ERPsets should be measured. We also send it the start and end times of the measurement window (110 and 150 ms, which are the values recommended in the ERP CORE paper). We also send two arrays, one containing a list of the bins that we want to measure (Bins 1-4, which contain the parent waveforms) and one containing a list of the channels that we want to measure (35 and 36, the left- and right-hemisphere cluster channels). There are also some parameters that you should recognize from using the **Measurement Tool** panel. Finally, we tell it the name of the text file that it should use for saving the amplitude scores (**MeanAmp.txt**).

Go ahead and run the script, and you'll see that it creates the **MeanAmp.txt** file, with one line for each subject and one column for each of our 8 measurements (4 bins x 2 channels). Load these data into a statistical package. We're going to ignore the bins for the scrambled stimuli at this point and focus on the unscrambled faces and cars, performing a 2 x 2 ANOVA with factors of stimulus type (face vs. car) and electrode hemisphere (left vs. right). If you run the analysis, you should get something like the ANOVA table shown in Table 11.3.

Table 11.3. ANOVA table for the first N170 analysis (from JASP).

Cases	Sum of Squares	df	Mean Square	F	p
Hemisphere	2.961	1	2.961	1.674	0.232
Residuals	14.535	8	1.769		
StimType	35.537	1	35.537	32.767	< .001
Residuals	8.676	8	1.085		
Hemisphere x StimType	1.047	1	1.047	2.048	0.190
Residuals	4.091	8	0.511		

As predicted, the greater N170 amplitude for the faces than for the cars led to a significant main effect of stimulus type. The effect was somewhat greater in the right-hemisphere cluster than in the left-hemisphere cluster, but the stimulus type x electrode hemisphere interaction was not significant. We have only 9 participants, so this probably just reflects low power.

The next part of the script obtains the mean amplitude scores from the difference waves. This time, however, it doesn't save the scores in a file. Instead, we use the **'SendtoWorkspace', 'on'** option to save the scores in a variable named **ERP_MEASURES** in the Matlab workspace. This variable is a 3-dimensional array with dimensions of bin, channel, and subject. For example, ERP_MEASURES(2, 1, 7) is the score for the second bin, the first channel, and the seventh subject. The script grabs these values and stores them in two one-dimensional arrays, one for the faces-minus-scrambled-faces bin and one for the cars-minus-scrambled-cars bin. The one dimension is subject, so each of these arrays has 9 values.

We then send these two arrays to the **ttest** function like this:

```
[h,p,ci,stats] = ttest(faces_minus_scrambled_meanamp, cars_minus_scrambled_meanamp);
```

The function returns four variables: **h** is 1 if the effect was significant and 0 otherwise (assuming an alpha of .05); **p** is the *p* value from the test; **ci** is the 95% confidence interval for the difference between the two means; and **stats** holds the actual *t* value along with the degrees of freedom. This routine can be used to perform a paired *t* test or a one-sample *t* test. The **ttest2** function can be used for an independent-samples *t* test, which you would use to compare two groups of subjects. After the script calls the **ttest** function, it uses **fprintf** to print the results in the Command Window. As you can see by looking at your Command Window, there was a significant difference in amplitude between the faces-minus-scrambled-faces and cars-minus-scrambled-cars difference waves (t(8) = -4.068, p = 0.004).

The last part of the script measures the peak latency of the N170 instead of the mean amplitude. It uses a wider measurement window (which is often needed for latency measures), and it performs the measurements from the more aggressively filtered ERPsets. The script then calls the **ttest** function and prints the results in the Command Window. The peak latency was significantly earlier for the faces-minus-scrambled-faces waveform than for the cars-minus-scrambled-cars waveform (t(8) = -2.438, p = 0.041, consistent with Figures 11.1D and 11.1E).

This is the last script for this chapter. You've now gone through every major step of EEG and ERP processing, all the way from reading in the raw EEG to conducting a statistical analysis. Congratulations!

But remember, you should go back and forth between scripts and the GUI rather than relying solely on scripts. For example, you should use the **Preview** button in the **Measurement Tool** panel to look at the scores alongside each averaged ERP waveform to verify that the measurement process is working properly. However, the script is also useful, because it makes it easier to repeat the processing if you need to make a change somewhere earlier in the pipeline. Also, if you make your scripts and data available when you publish a paper, other researchers can see exactly how you implemented each step of processing rather than relying on the relatively brief and vague description of the processing that is typically provided in a Method section.

Scripting is a skill that takes a long time to master, and you may initially wonder if it's worthwhile. It may seem like it's faster to do everything in the GUI than to spend hours debugging scripts that do the same thing. But you will get faster with experience, and you'll save a huge amount of time in the long run.

This page titled 11.18: Exercise- Scoring Amplitudes and Latencies and Performing Statistical Analyses is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 11.18: Exercise- Scoring Amplitudes and Latencies and Performing Statistical Analyses by Steven J Luck is licensed CC BY 4.0.

11.19: Key Takeaways and References

Key Takeaways

- The process of creating a script typically involves running the processing steps in the eStudio GUI, copying the commands from the history into a script, and adding a loop.
- You will save yourself a lot of time and pain in the long run by using intrinsically meaningful variable names, including lots of comments, and defining all key values as variables at the top of the script.
- It's almost always better to write a sequence of relatively simple scripts rather than one large script that does everything. This makes errors less likely, makes errors easier to find, and gives you more flexibility.
- You should make extensive use of the GUI the first time you process the data from a given participant. This allows you to detect problems in the data, set participant-specific parameters (e.g., for artifact rejection and correction), and make sure that everything is working properly. Once you've gone through the data in this manner, you should reprocess the data with your scripts (which helps avoid errors that can occur in manual processing).
- You can learn a lot by getting example scripts from other people, but don't apply those scripts to your own data unless you fully understand every line of code.

References

Emerson, R. W. (1841). *Self-Reliance*. Lulu Press.

Luck, S. J., Stewart, A. X., Simmons, A. M., & Rhemtulla, M. (2021). Standardized measurement error: A universal metric of data quality for averaged event-related potentials. *Psychophysiology*, *58*, e13793. https://doi.org/10.1111/psyp.13793

Rosenbaum, D. A., Vaughan, J., & Wyble, B. (2014). *MATLAB for Behavioral Scientists* (2nd Edition). Routledge.

Rossion, B., & Caharel, S. (2011). ERP evidence for the speed of face categorization in the human brain: Disentangling the contribution of low-level visual cues from face perception. *Vision Research*, *51*(12), 1297–1311. https://doi.org/10.1016/j.visres.2011.04.003

This page titled 11.19: Key Takeaways and References is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 11.19: Key Takeaways and References by Steven J Luck is licensed CC BY 4.0.

12: MULTIVARIATE PATTERN CLASSIFICATION (DECODING)

Learning Objectives

In this chapter, you will learn to:

- Apply decoding to a simple binary case of distinguishing between pictures of two different faces
- Apply decoding to the data from multiple participants at once, using multiple parallel processing cores to increase the speed of decoding
- Apply decoding to a more complex multiclass task, distinguishing among four different face identities or among four different emotional expressions
- Use decoding in place of conventional mean amplitude analyses to improve sensitivity, effect sizes, and statistical power (using data from the ERP CORE N170 experiment)
- Perform temporal generalization analyses, in which you can assess the extent to which similar neural representations are active at different time points
- Perform decoding on frequency-domain data as well as time-domain data
- Prevent the artificial inflation of decoding accuracy that typically occurs when there are more trials in one class than another (using both simulated data and real data)
- Quantify decoding performance in terms of both proportion correct and area under the curve (AUC) using data from the ERP CORE P3b experiment

This chapter focuses on ERPLAB's tools for multivariate pattern classification (decoding). Remarkably, you can decode subtle brain representations with ERPs, allowing you to determine whose face a participant just saw, the emotional content of a scene they are holding in working memory, and their planned motor responses. You can also apply decoding to traditional ERP paradigms to increase your statistical power for determining things like whether the N400 is larger for one class of words than for another.

Decoding isn't terribly difficult to understand, but it's pretty different from traditional approaches to ERP analysis. I'm going to assume that you already know the basics (e.g., how a support vector machine is applied to ERP data, how cross-validation works in the context of ERPs), so I will provide only a brief review of the essence of decoding. If you don't already understand the basics, I recommend reading Chapter 11 in the third edition of *An Introduction to the Event-Related Potential Technique* (Luck, in preparation). If you don't have access to that chapter, you can instead read the first three parts of the Decoding Tutorial in the ERPLAB documentation. However, I strongly recommend that you read the book chapter before you start applying decoding to your own data, because it discusses many potential problems that you will need to avoid. (And I'm not recommending it just to earn more royalties!)

12.1: Data for This Chapter

12.2: Example- Decoding Face Identity and Emotional Expression

12.3: The Essence of Decoding

12.4: Exercise- Binary Decoding of ID 1 versus ID 2

12.5: Exercise- Decoding Multiple BESTsets and Making a Grand Average

12.6: Exercise- 4-way Decoding of Face Identity and Emotional Expression

12.7: Exercise- Exporting the Data for Statistical Analysis

12.8: Exercise- Using Decoding in Place of Conventional Mean Amplitude Analyses

12.9: Exercise- Temporal Generalization

12.10: Demonstration- Decoding in the Frequency-Domain

12.11: Exercise- Artificially Inflated Decoding with Unbalanced Classes

12.12: Exercise- Methods for Appropriate Decoding with Unbalanced Classes
12.13: Exercise- Using Area Under the Curve (AUC) with the ERP CORE P3b Oddball Paradigm
12.14: Matlab Script For This Chapter
12.15: Key Takeaways and References

This page titled 12: Multivariate Pattern Classification (Decoding) is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck.

12.1: Data for This Chapter

The data we'll be using for the exercises in this chapter can be found in the Chapter_12 folder in the master folder: https://doi.org/10.18115/D5G045.

This page titled 12.1: Data for This Chapter is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck.

12.2: Example- Decoding Face Identity and Emotional Expression

I always like to start my explanations of decoding with an example from a face decoding study published by Gi-Yeul Bae (2021). He presented the participants with a sequence of photographs of faces and asked them to perform a modified 1-back task to ensure that they paid attention to the faces and stored them briefly in working memory. He used 16 different photographs of faces, which combined four different face identities (4 different people) with four different emotions (neutral, happy, angry, fearful), and each of the 16 photographs was presented 40 times. In one set of analyses, he decoded which of the four face identities was shown in the photograph, collapsed across the four different emotions for each face. In another set of analyses, he decoded which of the four emotional expressions was shown, collapsing across the four identities. In both cases, there were four classes of stimuli (four identities or four expressions), so chance was 25% (1 out of 4).

Figure 12.1 shows the grand average decoding accuracy, with the shaded region around each line indicating the standard error of the mean. As you can see, decoding accuracy was well above chance for both face identity and emotional expression, but the timing was faster for identity than for emotion.

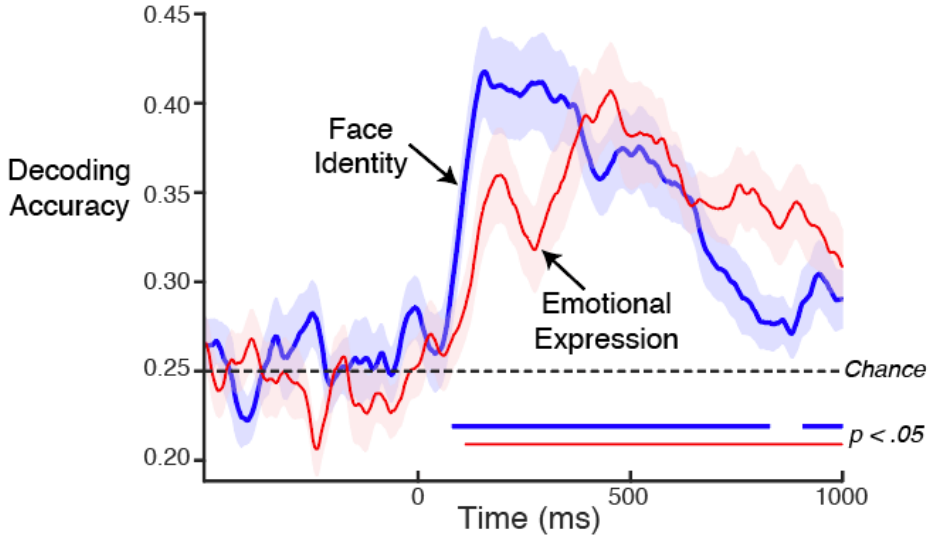


Figure 12.1.Grand average decoding accuracy from the study of Bae (2021). The shaded region shows the standard error of the mean. The lines along the bottom indicate time points at which decoding accuracy was significantly different from chance.

In the first part of this chapter, you will go through a set of exercises with the data from this experiment (which Gi-Yeul has made publicly available) so you can see how to implement these decoding analyses in ERPLAB Studio. If you've already gone through the previous chapters of this book (especially Chapters 1, 2, and 6), you'll see that decoding the data from this experiment is a straightforward extension of ERPLAB's approach to sorting trials into bins. Note that the original study of Bae (2021) used somewhat unusual filter settings, and here we will use more conventional settings.

In the second part of the chapter, we'll do some decoding exercises using the data from the ERP CORE N170 and P3b experiments. You'll see that you can use decoding accuracy as a replacement for conventional mean amplitude measures, which often yields increased statistical power. We'll also cover some important technical details that you absolutely must get right in order to have valid decoding results.

This page titled 12.2: Example- Decoding Face Identity and Emotional Expression is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck.

12.3: The Essence of Decoding

To understand the essence of decoding, start by imagining that Bae (2021) had only two face identities (ID 1 vs. ID 2), and we are trying to decode which one of them was presented on each trial. The decoder is given the set of voltage values across electrode sites for a trial (or the average of a set of trials) and attempts to guess whether that set of voltage values was produced by ID 1 or ID 2. A more formal term for decoding is *pattern classification*: a classifier (decoder) attempts to determine which class was present (ID 1 or ID 2) given a pattern of activity (the set of voltage values across electrode sites). The classifier is trained with several examples of patterns for each class, and it learns an optimal way to classify test cases that were not used for training. In a two-class problem such as distinguishing between ID 1 and ID 2, chance accuracy is 50% (i.e., this is the accuracy that would be expected by chance if the voltage patterns are the same for ID 1 and ID 2).

Decoding is conducted separately for each participant. It is also typically performed independently at each time point (e.g., 300 ms), giving us a waveform with decoding accuracy at each time point for each participant. These decoding waveforms are analogous to conventional ERP waveforms, and the statistical procedures used to analyze conventional ERPs can be applied in a straightforward manner to the decoding data.

To visualize how decoding works, we are going to start with a simplified situation in which we have only two electrodes (E1 and E2) and we are decoding from the single-trial data. As shown in Figure 12.2, these two electrode sites form a two-dimensional space, and the voltage at E1 and E2 for a single trial is represented as a location in this space. We have one set of points for ID 1 (drawn as blue dots) and another set for ID 2 (drawn as red dots). There is some overlap between these two classes, but the dots for ID 1 are mainly clustered in the upper right part of the space, whereas the dots for ID 2 are mainly clustered in the lower left part of the space. The goal of decoding is to find a *decision line* that optimally separates the two classes.

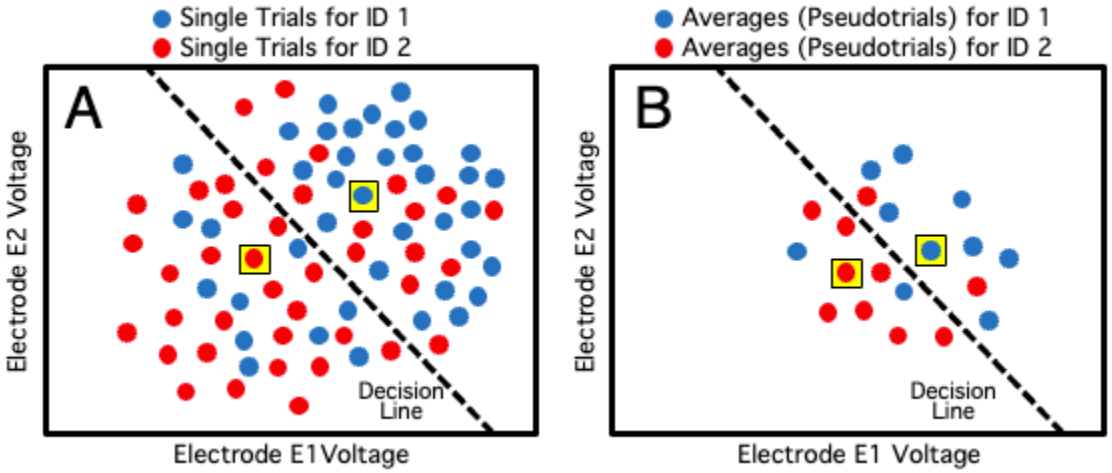


Figure 12.2. (A) Simplified representation of the pattern classification process when we have two classes (face ID 1 and face ID 2) and two electrode sites (E1 and E2). Note that these are artificial data, not the actual data from Bae (2021). Two test trials (the dots surrounded by squares) are held out from training and used to test the classifier. (B) Extension to classification from *pseudotrials* created by averaging multiple trials together. Each dot represents the scalp distribution measured from one pseudotrial. One pseudotrial from each class is held out for testing (marked by a square around the dot).

The classifier is first given a large number of training cases from each class, and it uses these cases to find a good decision line. However, it would not be fair to test the classifier on the same trials used for training, because the classifier might be picking up on random noise that produces small differences between the training cases for the two classes. Two *test trials* are therefore held out from training (marked by boxes surrounding the dots in Figure 12.2).

Once the classifier has finished using the training data to determine a decision line, the data from the test trials are fed into the classifier, and these trials are classified according to which side of the decision line they fall on. A trial is correctly classified if it falls on the correct side of the decision line for that class. Decoding accuracy is quantified as the proportion of test trials that were correctly classified. This use of separate trials for training and testing is called *cross-validation*.

With just two test cases, decoding accuracy could be 0% correct, 50% correct, or 100% correct. To get more fine-grained percent

correct values, we repeat the process *N* times (where *N* is the number of stimuli per class), leaving out a different pair of test trials on each repetition. For each repetition, a new decoder is trained and tested. We can then quantify the decoder as the proportion of trials that were correctly classified across all the repetitions. Each repetition is called a fold, and the whole process is called *N-fold cross-validation*.

Single-trial EEG epochs are usually too noisy to yield good decoding accuracy. As a result, we often create *pseudotrials* by averaging together multiple single trials. For example, Figure 12.2A shows single-trial data from 40 trials per class, and we could randomly divide the 40 trials for a given class into 10 sets of 4 trials, and then create an average from each set of 4 trials. This would give us 10 pseudotrials for each class, as shown in Figure 12.2B. We would train the classifier on 9 of the 10 pseudotrials for each class and then test the classifier with the 1 remaining pseudotrial for each class. We would then repeat this process a total of 10 times, each time leaving out a different pair of pseudotrials for testing. This would be a 10-fold cross-validation.

To minimize the influence of randomness, we iterate this process multiple times. On each iteration, we re-randomize which stimuli are assigned to a given pseudotrial. For each iteration, we get a proportion correct from the N-fold cross-validation, and the final decoding accuracy is the average of these proportion correct values across all the iterations. We typically run at least 10 iterations, and sometimes as many as 100.

So far, we have been assuming that you have only 2 electrode sites, giving us a simple 2-dimensional space. Fortunately, the procedure scales up in a straightforward way to larger numbers of electrodes. If you have, for example, 30 electrode sites, you will have a 30-dimensional space. Instead of finding a simple decision line in a 2-dimensional space, the classifier will find a *hyperplane* that optimally divides the actual 30-dimensional space, and test cases will be classified according to which side of the hyperplane they fall on. The more general term *decision boundary* can be used whether it is a line in a 2-D space or a hyperplane in a large-dimensional space.

We have also been assuming that you have only 2 classes, but you can also perform *multiclass decoding* when you have 3 or more classes. Multiclass decoding is achieved by training several individual decoders. For example, you can train one decoder for each possible pair of classes (called *one-versus-one* decoding). Alternatively, you can train one decoder for each class, distinguishing between that class and all of the other classes (called *one-versus-all* decoding). My lab has generally found that one-versus-all decoding works best with ERP data. After you've trained the several decoders, you can combine them with the *error-correcting output codes* approach (Dietterich & Bakiri, 1995). In this approach, each test case is fed into all of the individual decoders, each individual decoder indicates its confidence that the test case comes from a given class, and a single best guess is determined from the outputs of all the individual decoders.

There are many different algorithms that can be used to find the decision hyperplane. For the kinds of decoding that are common in cognitive and affective neuroscience, the support vector machine (SVM) algorithm seems to work best in most cases. It works particularly well when there are not a lot of training cases. This means that the SVM algorithm allows us to average many trials together into each pseudotrial, which creates a small number of relatively clean pseudotrials. ERPLAB implements the SVM algorithm and also allows you to use an older classification technique called *linear discriminant analysis* (LDA) and a newer method called the *cross-validated Mahalanobis distance* (Walther et al., 2016).

When applying decoding to ERPs, you will need to make several decisions:

- You will need to choose a classification algorithm. In most cases, I recommend using an SVM with one-versus-all decoding.
- You will need to choose which electrode sites to use. Researchers typically use all sites because the classifier will naturally give the greatest weight to the most informative sites.
- You will need to choose the number of folds and the number of trials to average together for each pseudotrial. In most cases, the number of trials per pseudotrial is simply the total number of trials per class divided by the number of folds.
- If the number of trials differs from class to class (e.g., because one class is more frequent than the other), you will need to choose a method for dealing with this variability.
- You will need to choose the number of iterations.

As usual, it is important to make these selections a priori so that you don't bias the results. We will discuss principles for making these choices as we go through the exercises in this chapter.

This page titled 12.3: The Essence of Decoding is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck.

12.4: Exercise- Binary Decoding of ID 1 versus ID 2

In our first exercise, we're going to do a simple binary decoding of ID 1 versus ID 2 for one participant in the study of Bae (2021). But first let's take a look at the EEG and event codes and do some preprocessing.

Reset everything with the **Reset** button and set the **Current Folder** to the **Chapter_12** folder. In the **EEG** tab, click **Load EEG** in the **EEGsets** panel, navigate to **Bae_Data > 302**, and select the file named **302_preprocessed.set**. The first subject, 301, had a few extra event codes, so we're going to focus on the data from Subject 302 in this exercise. We'll look at the data from Subject 301 later.

Now scroll through the data from **302_preprocessed** as you normally would when beginning to process a file.

All channels in this EEGset have been referenced to the average of the left and right mastoids, and the data have been filtered from 0.1–20 Hz. An EventList has been added. ICA has already been applied to this EEGset to correct blinks and eye movements. The EOG channels have not been corrected, so you will see blinks only in the VEOG L channel (VEOG-lower referenced to average mastoids) and the VEOG L-U channel (VEOG-lower minus Fp2), which were not corrected. You'll also see eye movements only in the HEOG L and HEOG R channels (HEOG-left and -right referenced to average mastoids) and the HEOG R-L channel (HEOG-right minus HEOG-left), which were also left uncorrected. We will exclude these uncorrected channels from the decoding process.

If you scroll far enough, you'll start seeing event codes. Here are the event codes for the 16 different combinations of face ID and emotional expression:

Event codes for the 16 different combinations of face ID and emotional expression

	Fearful	Happy	Neutral	Angry
ID 1	211	221	231	241
ID 2	212	222	232	242
ID 3	213	223	233	243
ID 4	214	224	234	244

ERPLAB uses its standard *bin* approach to define the classes for decoding. To decode a set of classes, you start by using BINLISTER to assign each event to the appropriate bin for each class. If you double-click on **BDF_ID_Expression.txt** in Matlab's **Current Folder** pane, you can see that 8 bins have been defined. Bins 1 through 4 correspond to ID 1 through ID 4, including all four emotional expressions for each ID. For example, Bin 1 corresponds to ID 1 and includes event codes 211, 221, 231, and 241. Bins 5-8 correspond to the four emotional expressions, including all four IDs in the bin for a given expression. When I decoded face identity (Figure 12.1), I did this by telling ERPLAB to use Bins 1-4 to define the four classes. When I decoded emotion expression, I told ERPLAB to use Bins 5-8 to define the four classes.

Now that you've seen how the bins are defined, let's run BINLISTER with this set of bin definitions. Go to the **Assign Events to Bins** panel, click the **Browse** button, and select the file named **BDF_ID_Expression.txt**. Then click **Run** and use the default EEGset name (**302_preprocessed_bins**). The panel will show you that there are 160 occurrences of each of the 8 bins. There were 40 trials for each of the 16 ID/emotion combinations, so this gives us 160 trials for each ID (collapsed across emotions) and 160 trials for each emotion (collapsed across IDs).

The next step is to epoch the data. Go to the **Extract Bin-Based Epochs** panel, make sure the epoch is set to **-200 800**

and that the baseline period is set to **Pre**, and click **Apply**. When the naming/saving window appears, use the default EEGset name (**302_preprocessed_bins_be**). Click **Run**. In the main eStudio plotting region, you should see that you now have epochs of 200-800 ms for each trial, with each epoch labeled with one of the eight bins.

We're now ready to begin the specific steps needed for decoding. The first step is to convert this EEGset into something called a *bin-epoched single-trial set* (BESTset). A BESTset contains bin-epoched data in a super-efficient organization. Each bin is stored as a 3-dimensional matrix of channels x time points x trials. We designed this data structure for decoding, but if you ever write custom scripts to apply some kind of new algorithm to single-trial EEG epochs, this format could be very useful.

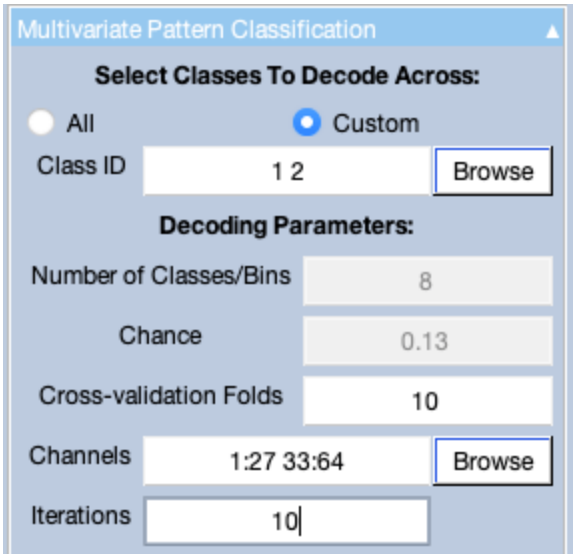
To convert the current EEGset into a BESTset, make sure that **302_preprocessed_bins_be** is still selected in the **EEGsets** panel, and go to the panel named **Extract Bin-Epoched Single Trials**. The option labeled **Only epochs without flagged artifacts** should be set (so that epochs flagged with artifacts are excluded), as should the option for **Exclude epochs with boundary events**. Although we're only going to decode ID 1 and ID 2 (bins 1 and 2) in this exercise, make sure that all 8 bins are selected (so we can use the same BESTset for later exercises). Do not check the box labeled **Transform into phase-independent power**; this option allows you to decode on the basis of the power in a given frequency band, which we will do in a later exercise. Click **Run**, change the name of the BESTset to **302**, check the boxes for **Save BESTsets to disk** and **Use bestname as filename**, and click **Okay**.

After a moment, you'll be automatically taken out of the **EEG** tab and into the **Pattern Classification** tab. This is where the magic happens! You should see that the BESTset you just created, named **302**, is in the **BESTsets** panel. This panel is analogous to the **EEGsets** and **ERPsets** panels, with options for loading, renaming, saving, and clearing BESTsets. Note that nothing is plotted when a BESTset is loaded. The plotting area in the **Pattern Classification** tab is used for plotting decoding accuracy, as we will see soon.

Make sure that **302** is selected in the **BESTsets** panel, and then go to the **Multivariate Pattern Classification** panel. This is a large and complicated panel, but we'll walk through it bit by bit. The first step is to select the classes used for decoding. By default, it will decode among all of the bins, so **All** is probably set. We want to decode only ID 1 versus ID 2 (bins 1 and 2), so click the **Custom** button and enter **1 2** in the **Class ID** text box (and press the Tab key to make sure that your choice was entered).

Now let's set up the **Decoding Parameters** section of the panel. The **Number of Classes/Bins** field should be set to **2** because you entered 2 bins in the **Class ID** field. The **Chance** value should therefore be set to **0.5** (1 out of 2). You can't control these parameters directly; they are determined by the number of classes/bins you selected in the **Class ID** field.

A value that you *can* select is the number of **Cross-validation Folds**. This determines what *N* to use for the *N*-fold cross-validation procedures. It also indirectly determines the number of trials that will be averaged together for each pseudotrial, which is the total number of available trials per class divided by the number of folds. Set this value to **10**. Set the channels panel to **1:27 33:64**, which will select just the EEG channels (and exclude the mastoid and EOG channels). Finally, set the number of **Iterations** to **10**. We usually use more than 10 iterations, but that can make the decoding process very slow, so we'll keep it at 10 for these exercises. The top portion of the panel should now look like Screenshot 12.1.



Screenshot 12.1

Now go to the **Number of trials and averages** section and select the options for **Equal Trials, Equal AVGs** and **Across Classes** (and don't check the box for **Across BESTsets**, which I will explain later).

Now click the button labeled **Show summary of trials and averages** near the bottom of the panel. It will pop up a window like the one shown in Screenshot 12.2, summarizing which bins have been selected as the classes for decoding. You can see that Class 1 is Face ID 1 and Class 2 is Face ID 2. You can also see that there are 160 trials available for each of these bins (the **N(trials)** column), that the number of pseudotrials/averages per bin is set to 10 (the **B(ERPs)** column), and that each ERP (pseudotrial) will be the average of 16 trials (the **N(per ERP)** column). With 160 trials and 10 pseudotrials/averages per bin (which is the same as the number of folds), we automatically get 16 trials per average.

BEST File	Class ID	Class/Label	N(trials)	B(ERPs)	N(per ERP)
302	**1**	Face ID 1	160	10	16
302	**2**	Face ID 2	160	10	16
302	NOT USED	Face ID 3	NOT USED	NOT USED	NOT USED
302	NOT USED	Face ID 4	NOT USED	NOT USED	NOT USED
302	NOT USED	Fearful	NOT USED	NOT USED	NOT USED
302	NOT USED	Happy	NOT USED	NOT USED	NOT USED
302	NOT USED	Neutral	NOT USED	NOT USED	NOT USED
302	NOT USED	Angry	NOT USED	NOT USED	NOT USED

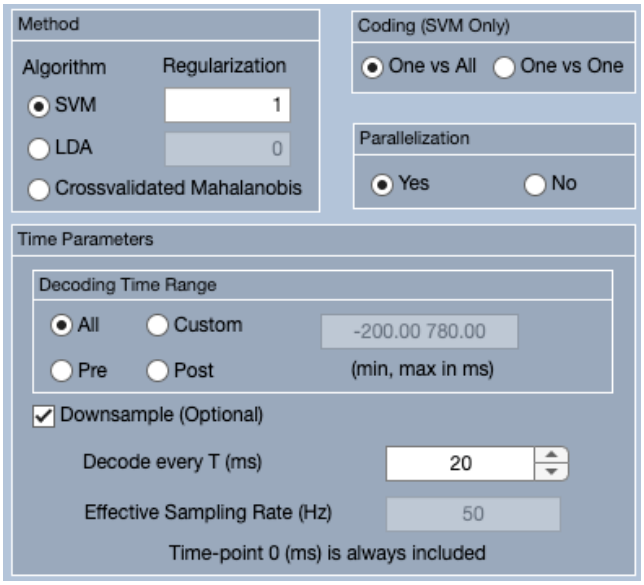
Screenshot 12.2

Close this summary window and look at the options near the bottom of the **Multivariate Pattern Classification** panel to make sure that **Metric** is set to **ACC** and that **Normalization** and **Temporal generalization matrix** are set to **Off**. I will explain these options later.

Now click the **Options** button near the bottom of the panel, which will pop up a new window, and set everything in that window to match Screenshot 12.3. Make sure that the **Algorithm** is set to **SVM** with a **Regularization** value of **1** and **One vs All** coding. [See Zhang et al. (2026) if you want to know more about the regularization parameter and why 1 is a good value.]

I recommend setting **Parallelization** to **Yes** if your computer has a multicore processor with at least 8 cores. Decoding can take a very long time to run, but it can easily run in parallel across multiple processing cores. If you have a multicore processor, turning on the parallelization can make the decoding run much, much faster. Note that the decoding routine leaves one core unused so

that you can still do other things while the decoding is running.

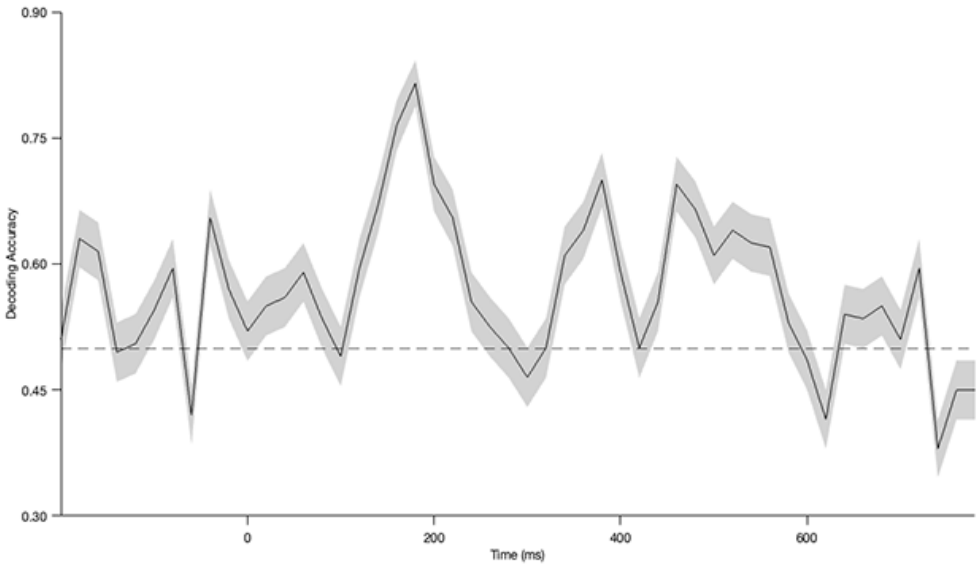


Screenshot 12.3

For the exercises in this chapter, we are also going to check the **Downsample** box and set the **Decode every T (ms)** value to **20**. Rather than decoding at every time point (every 4 ms with this dataset), this option sets it to decode every 5th time point (every 20 ms). This will allow it to run in 1/5 the amount of time required to decode every single time point. In my lab, we typically run our preliminary decoding analyses with this downsampling to make the decoding run faster. Once we've finalized the analysis pipeline, we set it to decode every time point (and then let it run overnight or even for a couple days).

Once you have everything set, click **Okay** to go back to the main decoding panel, and then click **Run** to actually perform the decoding. A window will appear that allows you to name the output. The output of the decoding process is called a *multivariate pattern classification set* or *MVPCset*. You can use the default name (which is simply **302**).

It will take some time for the decoding to finish running (somewhere between 15 seconds and 10 minutes). Nothing will seem to happen at first, because it takes some time for Matlab to get the parallel processing running. Then you will see some messages along the bottom of the eStudio window showing which iteration is running. When it is done, the new MVPCset will appear in the **MVPCsets** panel, and you should see a decoding waveform like that shown in Screenshot 12.4. It won't look exactly the same, however, because the trials are randomly assigned to the pseudotrials, and this creates some variability in the decoding. That variability will be small if you run a lot of iterations, but we are only running 10 iterations here, so the results will vary a bit each time you run the decoding. [You can prevent this by doing the decoding from a script and setting the seed of the random number generator to the same value each time the script runs.]



Screenshot 12.4

The first thing to note about the decoding waveform is that the chance level (0.5) is shown as a dashed line. Decoding accuracy should hover around this value during the prestimulus interval. If the decoding is consistently above or below chance during the prestimulus interval, that almost always indicates that there is a problem with the decoding or the experimental design. In Screenshot 12.4, you can see that the values bounce up and down around the chance line until approximately 100 ms. At this point, the neural signal contains some information that can distinguish between ID 1 and ID 2, and decoding accuracy rises quickly. However, this is a single participant, and there is still a lot of noise, so we shouldn't try to interpret the variations in decoding accuracy between 100 and 800 ms.

You've done it! You've taken the EEG data from a single participant and decoded whether the participant was viewing ID 1 or ID 2 at each time point. Spend a moment considering how amazing this is. The data were recorded from electrodes on the skin, a long distance from the parts of the brain involved in face processing, and yet we can decode which of two faces the participant was viewing. And we can do this separately at each moment in time. In my view, this is a minor miracle! I never would have thought this was possible 10 years ago. Not only is it possible, but it's also pretty easy to do with ERPLAB Studio.

Of course, there are some limitations to this decoding. The first is that we decoded from averages of 16 trials, not from individual trials. Single-trial decoding is possible when the classes being decoded produce very different patterns of voltage on the scalp, but it does not usually work well for subtle classifications like different faces. Another limitation is that the decoding accuracy is far from perfect. The decoding of ID 1 versus ID 2 is far too low to be used for engineering applications such as brain-computer interfaces. However, it is good enough for answering a lot of interesting scientific questions (see Hebart & Baker, 2018, for an excellent discussion of the differing needs of scientific versus engineering applications of decoding).

This page titled 12.4: Exercise- Binary Decoding of ID 1 versus ID 2 is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck.

- 6.8: Exercise - Response-Locked Averaging by Steven J Luck is licensed CC BY 4.0.

12.5: Exercise- Decoding Multiple BESTsets and Making a Grand Average

Now that you've seen how to decode the data from a single participant, we're going to decode multiple participants in one shot and then make a grand average. To keep things simple and fast, we'll just work with the data from subjects 301-305. I will assume that you have already decoded subject 302 and that the MVPCset for this subject is still loaded. Before getting started, click **Clear All** in the **BESTsets** panel and also in the **MVPCsets** panel.

I've already made the BESTsets for you. In the **BESTsets** panel, click **Load**, navigate to the **BESTsets** folder (inside the **Chapter_12** folder), and open the files named **301.best**, **302.best**, **303.best**, **304.best**, and **305.best**. Now select all five of these BESTsets in the **BESTsets** panel and go to the **Multivariate Pattern Classification** panel. Make sure the parameters are still set as shown in Screenshots 12.1 and 12.3 from the previous exercise.

Click the button near the bottom of the decoding panel labeled **Show summary of trials and averages**. You will now see information about all five of the BESTsets. One thing you should note is that, whereas there should be 160 trials in each class/bin, subject 301 has 190 trials in Bin 1 and 172 trials in Bin 2. This is the result of a small problem with the stimulus presentation script for this subject.

Because the trials for each bin are subdivided into 10 sets for making averages/pseudotrials (indicated in the **B(ERPs)** column), we have enough trials to have more than 16 trials per average in subject 301. As you will see in the **N(per ERP)** column, which indicates the number of trials per average, the subjects with 160 trials per class have 16 trials per ERP. However, subject 301 has 17 trials per ERP. That makes sense for Bin 2, which has 172 trials, but shouldn't there be 19 trials per ERP for Bin 1, which has 190 trials?

The reason for this is that *it's very important to have the same number of trials per average* for all bins/classes being decoded. As you will see in a later exercise, decoding accuracy tends to be artificially inflated when the number of trials per average differs across bins/classes. When you set up the main decoding panel, you selected the **Equal Trials, Equal AVGs** and **Across Classes** options, which told eStudio to subsample from the available trials for each bin/class to equate the number of trials per average. That is why both Bins 1 and 2 have 17 trials per average for subject 301.

Do you also need to equate the number of trials per average across participants? As long as you've equated the number of trials per average within each individual participant, you will not artificially inflate the decoding accuracy (i.e., produce decoding accuracy that is greater than it should be given the information in the EEG). Consequently, it is not always necessary to equate the number of trials per average across participants.

However, decoding accuracy is greater when more trials are available for averaging (all else being equal). Thus, you will need to equate the number of trials across participants if you are comparing decoding accuracy between two experimental conditions (e.g., comparing letter decoding accuracy for attended versus unattended stimuli) or between two groups (e.g., comparing letter decoding accuracy for a patient group and a control group) or if you are looking at individual differences (e.g., assessing whether letter decoding accuracy is correlated with scores on a reading test). This can be achieved by loading all the BESTsets into eStudio and checking the box labeled **Across BESTsets** in the decoding panel. This will equate the number of trials per average for all the loaded BESTsets, subsampling as necessary from the available trials. [Alternatively, you can manually determine the appropriate number of trials per average and select the **Manual Floor** option to specify this number.]

Let's try it! Close the window showing the summary, check the box labeled **Across BESTsets**, and click **Show summary of trials and averages**. You will now see that all four subjects have 16 trials per average. Now close the window showing the summary and uncheck the box labeled **Across BESTsets**. It's not necessary to equate across participants in the present study, because we are not comparing across groups or experimental conditions with different numbers of trials. There's no point in throwing away trials unnecessarily, and it would just reduce our statistical power in the present study.

Box 12.1 describes some common situations in which the number of trials will vary across groups or conditions, as well as a trick that can sometimes avoid the need to throw away large numbers of trials.

Box 12.1 Why the Number of Trials per Average May Vary (and a Trick)

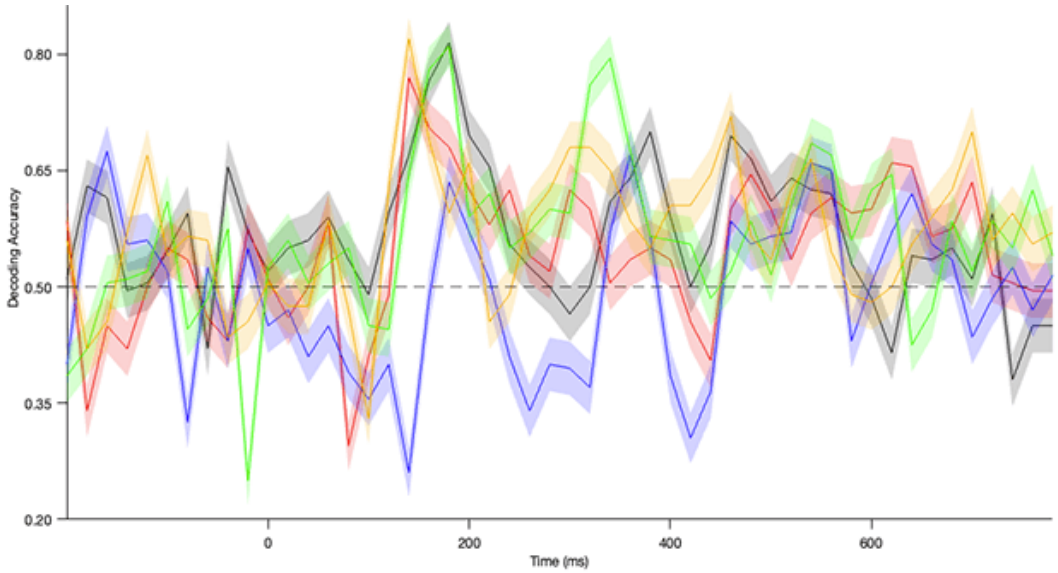
Sometimes the number of trials per bin/class differs widely because of the experimental design. For example, my lab ran an oddball experiment in which there were 4 stimulus locations that appeared on 87.5% of trials and 4 stimulus locations that appeared on 12.5% of trials (Carrasco et al., 2025). We decoded the location of the stimulus separately in the frequent and rare conditions, and then we compared decoding accuracy for the frequent and rare conditions. To avoid having greater decoding accuracy for the frequent condition simply because more trials were available for decoding, we equated the number of trials used for decoding in the rare and frequent conditions. We found that location decoding accuracy was slightly better for the rare set of locations than for the frequent set of locations, especially after 800 ms poststimulus.

Artifact rejection is another common cause of differences in the number of trials per bin/class. In many cases, this leads to modest random variations in the number of trials per bin/class, in which case the **Equal Trials, Equal AVGs** and **Across Classes** options will solve the problem. However, if you are comparing decoding accuracy across experimental conditions, across groups, or across participants, there may be systematic (nonrandom) differences in the number of trials marked for rejection. For example, blinks might be more common in a more difficult experimental condition, in a cognitively impaired patient group, or in individuals who score high on some individual differences measure. In these cases, you will probably need to equate across conditions or participants as well as across bins/classes (e.g., by checking the box labeled **Across BESTsets**).

This can sometimes require excluding a very large number of trials. For example, imagine that you ran a study to compare location decoding accuracy for Group A and Group B, and 48% of trials were rejected for one participant in Group A, but fewer than 5% were rejected in each of the other participants. If you simply loaded all the BESTsets and checked the box labeled **Across BESTsets**, eStudio would exclude 48% of the trials for every participant in both groups. This would make you very unhappy, because throwing out so many trials will reduce your statistical power. Here's a trick to make you happier: You could instead equate the number of trials across individual pairs of participants. That is, you would randomly pair together one participant from Group A with one participant from Group B, load their BESTsets and check the box for equating Across BESTsets, and decode those two participants. You'd then repeat this for every other pair of participants. That way, there would be only one pair where 48% of the trials would need to be excluded, and yet the number of trials per bin/class would be equated on average for the two groups.

We're now ready to perform the decoding of participants 301–305. If you have a reasonably fast computer or you are willing to wait a while for the decoding to finish, go ahead and click the **Run** button in the decoding panel. If you don't want to wait, you can go to the **MVPCsets** panel, click **Load**, navigate to the **MVPCsets** folder, and load the MVPCsets that I've already created for these participants (which are named **30#_ID1vsID2.mvpc**).

When you have all five MVPCsets loaded (301-305), make sure they are all selected in the **MVPCsets** panel. You should see them all overlaid as in Screenshot 12.5 (but they may look a little different given the randomization that is used in the process of assigning trials to averages/pseudotrials). You can see that there is a lot of variability, mainly due to the noise in the single-participant EEG data.



Screenshot 12.5

Now let's make a grand average of these five MVPCsets. Make sure they are all selected in the **MVPCsets** panel and then open the **Average Across MVPCsets** panel. You should see that the **MVPCsets** text box contains **1:5**, which indicates that MVPCsets 1-5 will be averaged together. Make sure that the option for computing the point-by-point standard error is selected, and click **Run**. Name the resulting MVPCset whatever you'd like. You should now see the grand average decoding waveform in the main plotting window, with the standard error indicated by the shaded regions around the lines.

This page titled 12.5: Exercise- Decoding Multiple BESTsets and Making a Grand Average is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck.

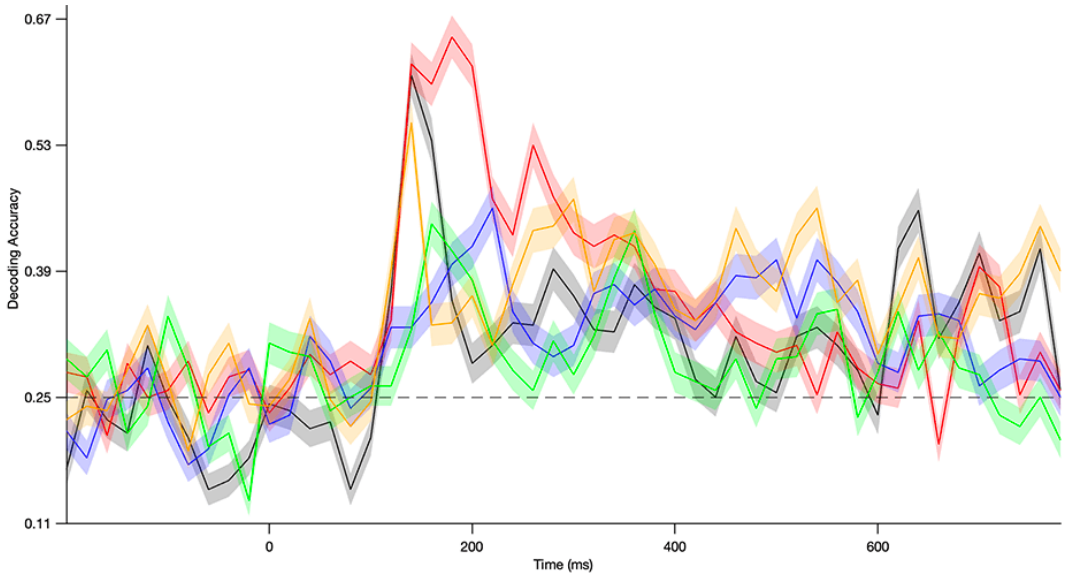
- 6.8: Exercise - Response-Locked Averaging by Steven J Luck is licensed CC BY 4.0.

12.6: Exercise- 4-way Decoding of Face Identity and Emotional Expression

Now that we've seen how decoding works when there are only two classes, let's do the 4-way decoding of face identity and the 4-way decoding of emotional expression. First, click the **Clear All** button in the **MVPCsets** panels to clear the decoding results from the previous exercise. Now go to the **BESTsets** panel in the **Pattern Classification** tab and make sure that the **BESTsets** for subjects 301-305 are still loaded (if you need to reload them, they are located in the **BESTsets** folder). Then select all five BESTsets.

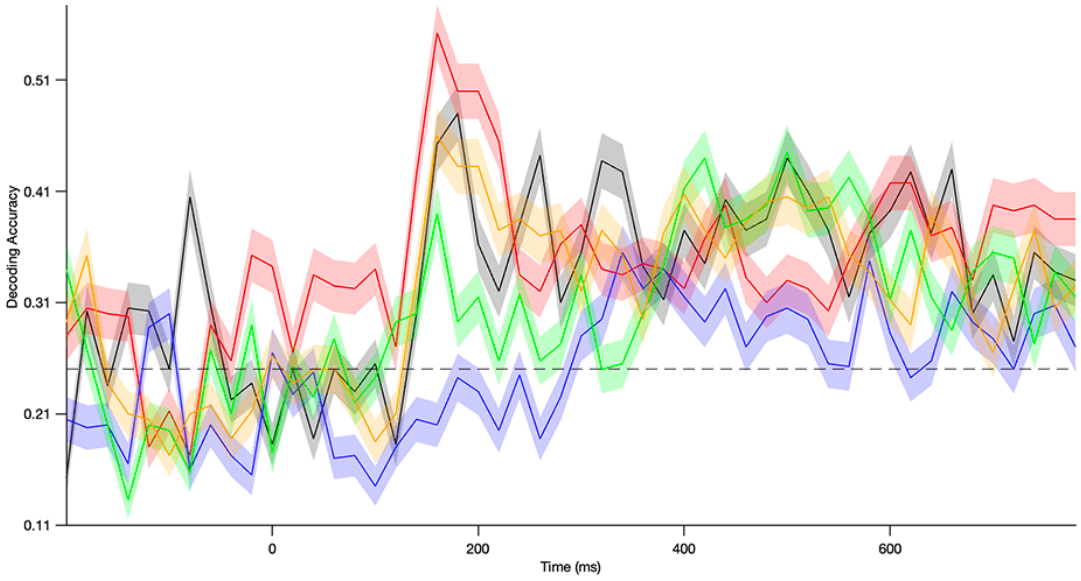
Now go to the **Multivariate Pattern Classification** panel and set it up just like before (see Screenshots 12.1 and Screenshot 12.3), except change the **Class ID** field to **1 2 3 4**. This will allow us to do the 4-way decoding of face identity. Click the **Show summary of trials and averages** button to make sure the number of trials per average seems reasonable (17 for subject 301, 16 for the other subjects). You can then click **Run** to perform the decoding or go to the **MVPCsets** panel and load the MVPCsets I've already created for these 5 subjects (located in the **MVPCsets** folder, named **30#_ID.mvpc**).

When you have all five MVPCsets loaded, select all of them in the **MVPCsets** panel so you can see them all overlaid as in Screenshot 12.6 (note that they may look a little different given the randomization of trials). One thing to note is that chance is now 25% rather than 50% (because there are now four classes, and pure guessing would lead to 25% correct).



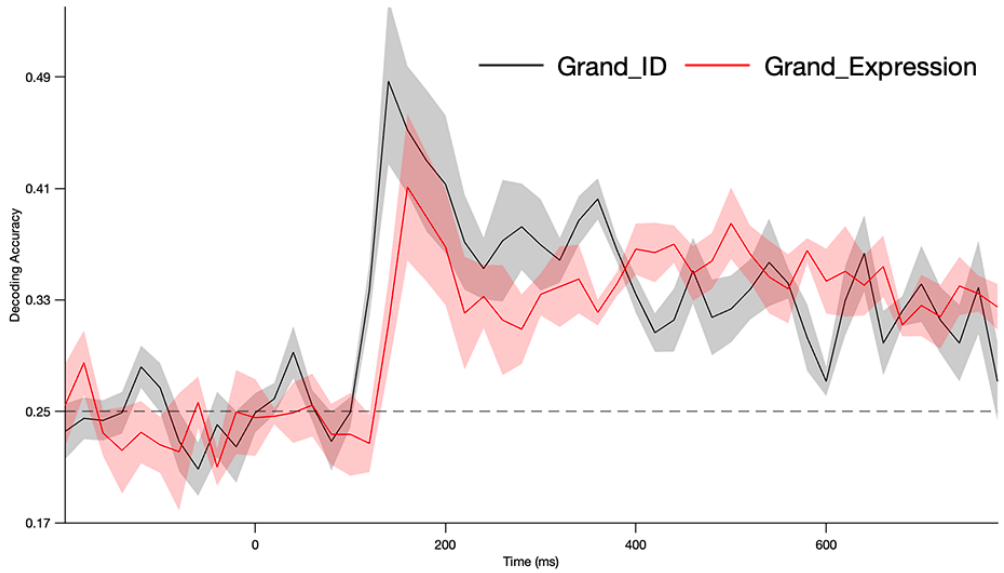
Screenshot 12.6

Now repeat the process, but specifying **5 6 7 8** in the **Class ID** field so that we can do the 4-way decoding of emotional expression. If you want to load the **MVPCsets** I've already created for these 5 subjects, they are in the **MPVCsets** folder (named **30#_Expression.mvpc**). The decoding waveforms for the five participants should look something like those in Screenshot 12.7. Like single-subject ERPs, the single-subject decoding waveforms are pretty noisy.



Screenshot 12.7

Now let's make grand averages using the **Average Across MVPCsets** panel. MVPCsets 1-5 should correspond to the ID decoding, so specify those 5 MPVCsets at the top of this panel. Create the grand average and name the resulting MVPCset **Grand_ID**. Now repeat the process for MVPCsets 6-10, naming the result **Grand_Expression**. If you go back to the MVPCsets panel and select the two grand averages, you should see something like Screenshot 12.8. Even though we're looking at only 5 participants, these grand averages look quite a lot like the grand averages of all 22 participants we saw in Figure 12.1, with clearly above-chance for both ID and emotional expression and an earlier onset of decoding for ID. I find that decoding waveforms are just as consistent across participants—or even more consistent—compared to ERP difference waves, assuming you have enough trials.



Screenshot 12.8

Now that you've seen how decoding works, I want to mention something that has been lurking in the background. Specifically, with the exception of subject 301, the number of trials per class was the same for each participant. Shouldn't there be some variation as a result of artifact rejection? When Gi-Yeul Bae first started doing decoding in my lab, he claimed that artifact rejection was not helpful, so no artifacts were rejected in this dataset. I was always a little skeptical of this claim, so Guanghui Zhang looked at this issue systematically across a large number of datasets (Zhang & Luck, 2025). He found that Gi-Yeul was correct: Artifact rejection is not usually beneficial for decoding, and it can actually make decoding worse because the reduction in trials directly reduces decoding accuracy. Moreover, artifacts are not as harmful as you might expect, because the SVM algorithm

will automatically downweight any electrode sites that contain wild voltage swings. However, if you have artifacts that differ systematically across conditions, those artifacts could artificially inflate your decoding accuracy, and that would be a problem. In most cases, I recommend using artifact correction to minimize activity related to blinks and eye movements, which does not degrade decoding accuracy but does minimize the potential for confounds. However, I recommend not using artifact rejection unless you are concerned about systematic differences in other artifacts.

Here's another minor background issue. When you go to the **Options** window in the decoding panel, you will see a parameter named **Regularization** with a default value of **1**. This parameter (called the *box constraint* parameter in Matlab) balances the trade-off between minimizing error in the training set and maintaining a simpler decision boundary that is more likely to generalize to new cases. A value of 1 gives equal weight to these two goals, and we have found that it is a good setting across a wide range of experimental paradigms (Zhang et al., 2026).

This page titled 12.6: Exercise- 4-way Decoding of Face Identity and Emotional Expression is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck.

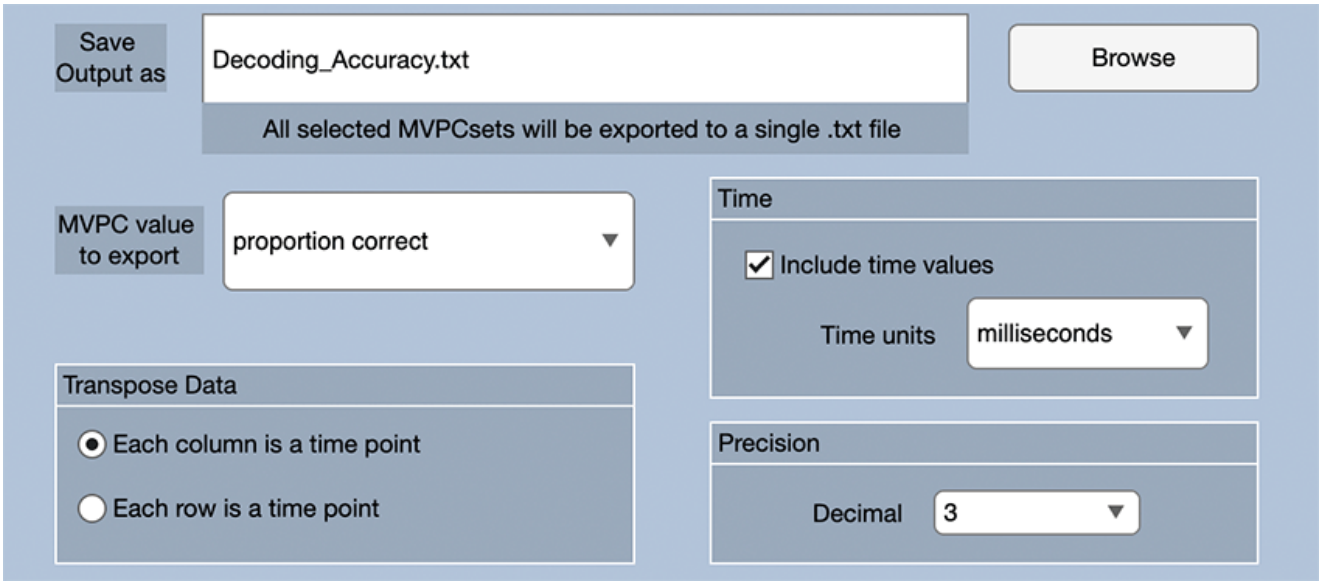
- 6.8: Exercise - Response-Locked Averaging by Steven J Luck is licensed CC BY 4.0.

12.7: Exercise- Exporting the Data for Statistical Analysis

A general design feature of ERPLAB is that we do not have any built-in statistical analyses. If we included a few simple but easy-to-use statistical functions, many people would use those functions whether or not they were appropriate. The alternative would be to provide a huge suite of statistical functions, but then it would be complicated to use, and we would be duplicating the functions available in packages like R. Thus, our approach is to make it easy for people to export the data in formats that can be read by general-purpose statistical packages. This isn't as easy for you as built-in statistics would be, but I think it's actually better if the statistical analysis isn't too easy. Appropriate statistical analysis takes some real thought!

In this exercise, we'll see how the decoding accuracy values can be exported from eStudio for statistical analysis. Make sure you still have the 10 MVPCsets loaded from the previous exercise (1-5 for the ID decoding, 6-10 for the emotional expression decoding). Select all 10 of them in the **MVPCsets** panel (but don't include the grand averages) and click **Export**. You should see a window like the one shown in Screenshot 12.9.

Change the name in the **Save Output as** field to **Decoding_Accuracy.txt**. This file will be saved in Matlab's current folder. If you want to save it elsewhere, you can type in the full path or click the **Browse** button. **MVPC value to export** should be set to proportion correct, which is the only available option for these MVPCsets. Later, I'll explain how you can instead quantify decoding accuracy as the area under the receiver operating characteristic curve (AUC).



Screenshot 12.9

In the **Transpose Data** section, you can either have one row for each participant and a separate column for each time point, or one row for each time point and a separate column for each participant. For this exercise, select **Each column is a time point**.

Make sure that the box is checked for **Include time values** with units of **milliseconds** so that the output file contains the timing information. And make sure the **Precision** is set to **3** (meaning that the accuracy values will have 3 digits to the right of the decimal point). Click **Export**.

You will now have a file named **Decoding_Accuracy.txt** in the **Chapter_12** folder. You can see it by double-clicking on it in Matlab's **Current Folder** pane. Alternatively, you can import it into a spreadsheet or a statistical analysis program.

You'll see that the top row of the file indicates the latency in milliseconds for each time point (from -200 to +780 ms). And you'll see that the left column indicates the name of the MVPCset (which contains the participant number and whether ID or emotional expression was being decoded). Each of the other cells of the file contains the proportion correct for a given combination of latency and MVPCset.

In general, you can analyze these data in the same way as regular ERP data. For example, you can compute the mean decoding accuracy across the points in a given time range (like mean amplitude in conventional ERP analyses) and compare these values against chance with a one-sample *t* test (to see if the decoding was above chance). You can also compare these values across groups or conditions with an *F* or *t* test (to see if the amount of decodable information differed across groups or conditions). Alternatively, you can use the *mass univariate* approach in which you perform statistical comparisons at each individual time point and then apply a correction for multiple comparisons. For more information, see Chapter 11 in Luck (in preparation).

This page titled 12.7: Exercise- Exporting the Data for Statistical Analysis is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck.

12.8: Exercise- Using Decoding in Place of Conventional Mean Amplitude Analyses

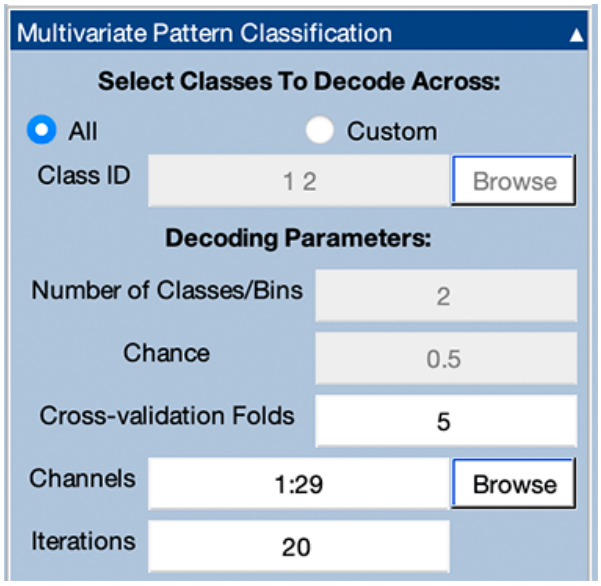
After my lab had spent several years decoding perceptual and working memory representations (and answering questions that we never could have asked with traditional univariate methods!), we realized that decoding might also be more sensitive for answering traditional ERP questions. If we could decode which of four faces was presented, surely the decoding approach would have excellent sensitivity for detecting the larger N170 component elicited by faces than by cars. Carlos Carrasco, then a grad student in my lab, did a formal study of this using each of the 7 ERP components of the ERP CORE (Carrasco et al., 2024).

As we expected, Carlos found that decoding was more sensitive than traditional methods for detecting differences in component amplitudes, especially for the N170, P3b, N400, and ERN components. Decoding was no better or worse than traditional methods for the N2pc, LRP, and MMN components. However, we later developed an improved decoding method for components such as MMN that come from paradigms in which one class is more common than the other, and this improved decoding performance for the MMN (Zhang et al., in preparation). My guess is that decoding didn't help with the N2pc and LRP paradigms because the conventional analysis in these paradigms already makes use of an optimized difference between electrode sites (contralateral versus ipsilateral), and decoding just can't do much better.

Let's see how decoding works as a substitute for conventional mean amplitude measurements with the ERP CORE N170 paradigm (described in detail in Chapter 11). We'll focus on the faces and cars, leaving out the random-phase faces and random-phase cars (see Carrasco et al., 2024 for analyses of those trial types). As in Chapter 11, we'll focus on subjects 1-10 (leaving out subject 5, who had too many trials rejected). You can find the EEGsets for these 9 subjects in the **N170_EEGsets** folder within the **Chapter_12** folder. These EEGsets were just copied from the output of the analysis pipeline script described in Chapter 11, and they have been preprocessed with filtering, referencing, artifact correction, artifact detection, and epoching.

I've already created BESTsets from these EEGsets, but with only Bin 1 (faces) and Bin 2 (cars). You can find them in the **N170_BEST** folder. Clear the existing BESTsets and MVPCsets from eStudio, go to the **BESTsets** panel in the **Pattern Classification** tab, and load all 9 of the BESTsets.

Now go to the **Multivariate Pattern Classification** panel and set it up as shown in Screenshot 12.10. Note that we have only two classes (faces and cars). There were 50-80 trials per bin (after artifact rejection), which means that we'd have only 5-8 trials per average/pseudotrial if we kept the number of folds at 10. We find that it's usually best to have 10-20 trials per average/pseudotrial (see Box 12.2), so set the number of **Cross-validation Folds** to **5**. If you click on **Show Summary of trials and averages**, you'll see that this gives us the desired number of trials per average/pseudotrial.



Screenshot 12.10

Box 12.2: Choosing the Number of Folds

If you have 50 trials per class, how many folds should you use? For example, is it better to select 5 folds (i.e., 5 averages/pseudotrials per class, resulting in 10 trials per average/pseudotrial) or 10 folds (resulting in 5 trials per average/pseudotrial)? Guanghui Zhang examined this question across a wide variety of datasets (Zhang et al., Under Review). He systematically varied the balance between the number of folds/averages/pseudotrials and the number of trials per average/pseudotrial (ensuring that the number of averages x the number of trials per average was always the total number of trials), and he calculated decoding accuracy and the Cohen's d_z metric of effect size for every combination.

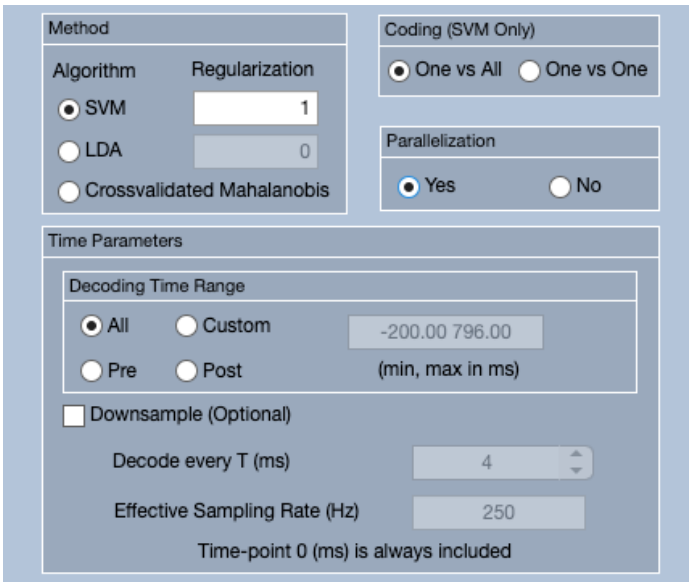
Guanghui found that the number of folds doesn't matter very much as long as you end up with at least ~5 trials per average/pseudotrial. In most cases, the highest decoding accuracy and largest d_z were found with 3-5 folds and at least 10 trials per average. However, as long as you avoid the extremes (fewer than 5 trials per average or fewer than 3 folds), the differences in decoding accuracy and d_z were modest. So, it probably won't matter much how many folds you choose as long as you avoid the extremes.

Note, however, that all of the datasets in Guanghui's analysis had a reasonably large total number of trials per class (at least 30), and all of them had relatively clean EEG recorded from highly cooperative participants. We can't be sure that the same conclusions would apply to situations where the number of trials is small and/or the EEG is very noisy.

If you click the **Browse** button for selecting the **Channels**, you can select all the EEG channels (channels 1-29). We're going to set the number of **Iterations** to **20** for this exercise, because our goal is to compare decoding accuracy against conventional mean amplitude measures, and we want the decoding accuracy values to be reasonably precise. In his paper, Carlos actually used 100 iterations, but that would take too long to run for this exercise.

Make sure that the **Equal Trials, Equal AVGs** and **Across Classes** options are set, and that the **Metric** is set to **ACC**. Also, set the **Temporal generalization matrix** option to **On**. We'll take a look at the results of that option in the next exercise.

Now click the **Options** button and set the options as shown in Screenshot 12.11. We're not going to downsample for this exercise, because that would give us only two time points in the N170 latency range (110-150 ms for the ERP CORE data). Click **Okay** to confirm the options and go back to the main decoding panel.

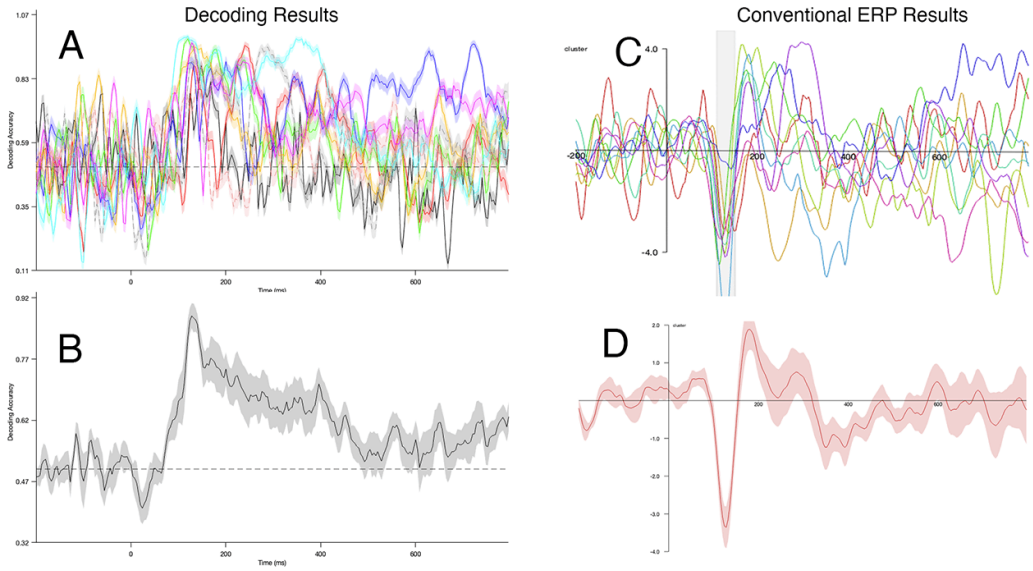


Screenshot 12.11

If you have a fast computer (or you're not in a hurry), you can click the **Run** button to start the decoding process. It took about 10 minutes to run the decoding on my laptop computer, which has a 14-core Apple M4 Pro processor, but it might take a lot longer if you have fewer cores or a slower processor. If you don't want to spend the time running it yourself, you can go to the **MVPCsets** panel and load the MVPCsets I obtained when running the decoding, which you can find in the **N170_MVPC** folder.

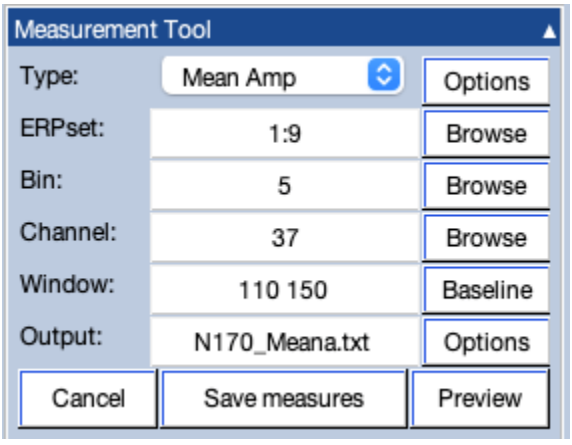
Once you have all the MVPCsets selected, you should see something like Screenshot 12.12A. The single-subject decoding waveforms are noisy (as is typical), but you can see that the decoding accuracy is well above chance for all 9 subjects at around 150 ms.

Now make a grand average of these 9 MVPCsets (by selecting them all, going to the **Average Across MVPCsets** panel, and clicking **Run**). You should now see something like Screenshot 12.12B. The mean decoding accuracy is well above chance at around 150 ms, especially relative to the standard error of the mean.



Screenshot 12.12

Now let's see how this compares with the conventional N170 effect, which is typically quantified as the difference between the face-elicited ERPs and the car-elicited ERPs. I've already made the ERPs for you. To load them, go to the **ERPsets** panel in the **ERP** tab, click **Load**, navigate to the **N170_ERPs** folder, and load all 9 ERPsets. I've already created a faces-minus-cars difference wave (Bin 5) and a cluster channel that averages across the recommended measurement channels for the N170 (Channel 37). The easiest way to create a plot that overlays these difference waves is to use the **Measurement Tool** panel. This will also allow us to measure N170 amplitude as the mean amplitude between 110 and 150 ms. Set up the **Measurement Tool** panel as shown in Screenshot 12.13 and then click the **Preview** button.



Screenshot 12.13

In the Preview window that appears, set **Show ERPsets** to **All**. You should see something like Screenshot 12.12C, which displays the faces-minus-cars difference wave from the cluster channel for all 9 subjects. Just as the decoding accuracy around 150 ms was above chance for every subject (Screenshot 12.12A), the difference wave was negative for every subject at this time. However, the amount of variability in the ERP voltage at this time was substantially greater than the amount of variability in decoding

accuracy at this time. As I said earlier, decoding accuracy waveforms are as consistent or more consistent than the corresponding ERP difference waves in most cases.

Close the Preview window and click **Save measures** in the **Measurement Tool** panel. This creates a text file with the mean amplitude for each of the 9 subjects. We'll come back to that file in a moment.

Now let's make a grand average of the ERPs for the 9 subjects. Make sure that all 9 ERPsets are selected in the **ERPsets** panel, go to the **Average Across ERPsets** panel, make sure that **Compute point-by-point SEM** is checked, and click **Run**. Now go to the **Bin & Channel Selection** panel and select Channel **37** and Bin **5** (and click **Apply**). Then go to the **Plot Settings** panel and check the box for **Show standard error** (and click **Apply**). The plotting area should now show something like Screenshot 12.12D.

Compare the grand average ERP waveform (Screenshot 12.12D) with the grand average decoding waveform (Screenshot 12.12B). Note that whereas the ERP waveform shows a large and robust negative effect peaking around 130 ms, followed by a weak positive effect from approximately 170-200 ms and then not much else, the decoding accuracy stays well above chance for hundreds of milliseconds. The decoder is picking up on information that is present in the single-subject EEG scalp distributions after 220 ms that is not visible in the ERP waveforms once we average across a cluster of electrode sites and across subjects. This shows how decoding can reveal information that is invisible in conventional ERP analyses. For an explanation of why this is, see Chapter 11 in Luck (in preparation).

Now let's get back to the question of whether decoding is more sensitive than conventional mean amplitude for detecting the main N170 effect (which is best measured from 110-150 ms in this dataset; see Kappenman et al., 2021). Given that decoding accuracy and ERP amplitudes are on different scales, how can we compare the decoding effect with the ERP amplitude effect? The answer is to put them on a common scale. And a really good scale for this is the Cohen's d_z metric of effect size. This is just like the standard Cohen's *d*, except designed for comparing values against chance or comparing two within-subjects conditions (see Lakens, 2013). It is a good choice for the present situation because it is directly related to statistical power: A larger d_z means you will have a greater probability of getting $p < .05$ if an effect is actually present.

In the present case, we want to ask whether the mean voltage from 110-150 in the faces-minus-cars difference wave is significantly different from zero (which is equivalent to asking whether the voltage in this time period is significantly different for faces versus cars). We could test this with a one-sample *t* test against zero, and the d_z value is the corresponding effect size for this *t* test. For the decoding analysis, we can take the average decoding accuracy from 110-150 and compare this against chance (0.5) with a one-sample *t* test, and we can also compute the corresponding d_z value. If the d_z value is greater for the mean decoding accuracy from 110-150 ms than for the mean voltage from 110-150 ms, then we can conclude that decoding gives us more statistical power for detecting the presence of difference in brain activity between faces and cars.

In the **Chapter_12** folder, you will find a spreadsheet named **N170_meana.xlsx**. It contains the single-subject N170 mean amplitude measures that we obtained earlier in this exercise, along with Excel formulas for calculating the mean across subjects, the standard deviation (SD) across subjects, and the d_z value (which is simply the mean minus chance (0) divided by the SD). You can see that the d_z metric of effect size was -2.50. [It's a negative value because we did faces-minus-cars, which gives us a negative value.]

The **Chapter_12** folder also contains a spreadsheet named **N170_decoding_accuracy.xlsx**, which contains the decoding accuracy values for each subject at each time point. I created this spreadsheet by selecting the 9 MVPCsets in the **MVPCsets** panel and clicking **Export**. Using Excel, I added formulas to compute the mean decoding accuracy between 110 and 150 ms for each subject (column B). And for the resulting values, I added the mean across subjects, the SD across subjects, and the d_z value (which is simply the mean minus chance (0.5) divided by the SD). The d_z for decoding accuracy was 3.46, which is quite a bit larger than the d_z for mean amplitude.

These results demonstrate that, at least in this study, mean decoding accuracy over the time window for the N170 component provides us with a larger effect size (and therefore greater statistical power) than we get when using the conventional mean amplitude over that same time window. Larger effect sizes are almost always a good thing, because they mean that we are better able to determine whether an effect is truly present or absent. Thus, you may want to consider using decoding accuracy rather than mean amplitude when you are analyzing a study that was designed to look at conventional ERP components. However, you should first read Chapter 11 in Luck (in preparation) to understand the potential downsides to this approach. Unfortunately, there are always downsides to whatever approach we take, and it's up to us to choose the approach with the most upside potential. I guess that's why they pay us the big bucks...

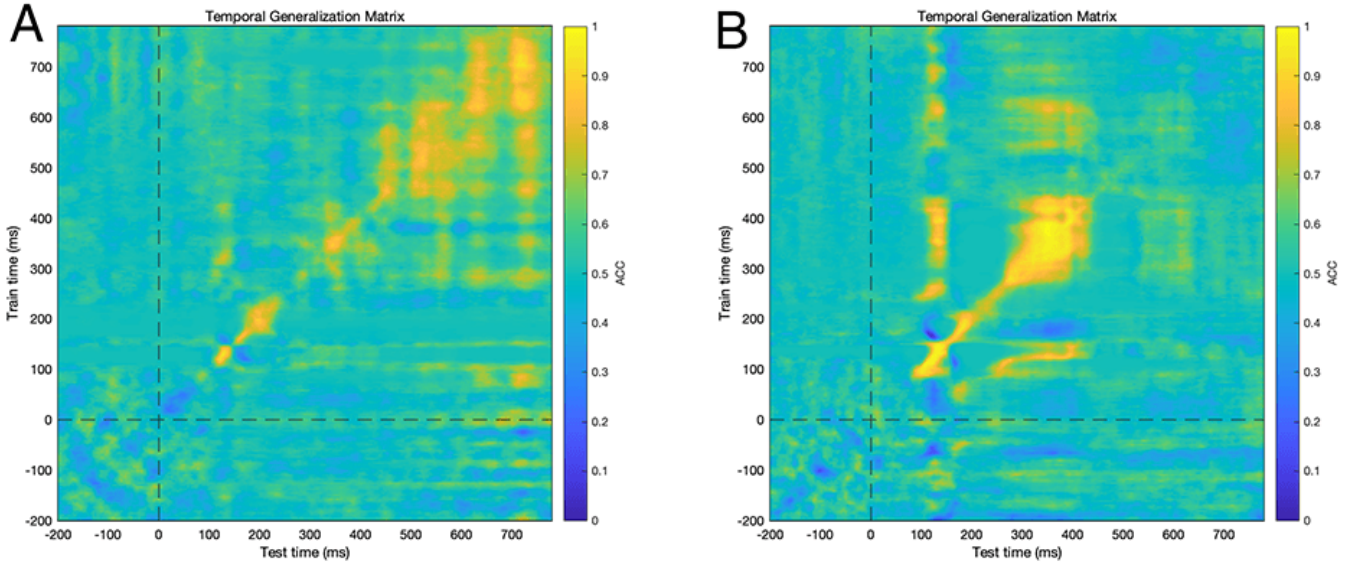
This page titled 12.8: Exercise- Using Decoding in Place of Conventional Mean Amplitude Analyses is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck.

12.9: Exercise- Temporal Generalization

You can learn a lot about how specific neural representations come and go over time by looking at what's called *temporal generalization* of decoding (King & Dehaene, 2014). The basic idea is that you train a decoder at one time point and then you feed it test cases from another time point. If the decoding accuracy is above chance, this suggests that similar brain representations are used at the different time points. For example, if you train the decoder at 150 ms and test it at 600 ms, and you find significant decoding, this suggests that the sensory representation at 150 ms is still active at 600 ms (or, to be more precise, that the representation at 600 ms is similar to the representation at 150 ms).

You can get information about temporal generalization in eStudio by simply checking the box labeled **Temporal general matrix** when setting up the decoding in the **Multivariate Pattern Classification** panel. In fact, you already did this in the previous exercise. When a decoder is trained with the data from a given time point, this option will cause it to test that decoder with the data from every time point. This gives you a decoding accuracy value for every combination of training time point and testing time point.

Let's see what this looks like. Make sure you still have the 9 MVPCsets from the previous exercise loaded. Then select the MPVCset labeled **3_N170**, go to the **Plot/Export Temporal Generalization Matrix** panel, and click the **Plot** button. You should see something like Screenshot 12.14A. The vertical (Y) axis indicates the time point used to train the decoder, and the horizontal (X) axis indicates the time point used to test the decoder. Every combination of training time and testing time is shown. The color scale shows the decoding accuracy, with lighter values indicating higher decoding accuracy.



Screenshot 12.14

From approximately 100-400 ms, decoding accuracy is well above chance (marked in bright yellow) mainly along the diagonal. This means that training at a given time point (e.g., 200 ms along the Y axis) produces good decoding only for that point and the surrounding time points (e.g., 190-210 ms along the X axis). In other words, there was very little temporal generalization in this time range, which means that the representational format changed quickly from moment to moment. After 400 ms, however, there was more generalization. For example, when the decoder was trained at 460 ms (along the Y axis), there was good decoding (yellow values) when the decoder was tested at most points between 450 and 700 ms (along the X axis). This suggests that the later representations tended to persist for hundreds of milliseconds. The pattern shown by this participant is fairly common.

You can see another common pattern by selecting the data for **7_N170** from the **MVPCsets** panel and plotting the temporal generalization matrix for this subject (see Screenshot 12.14B). This subject also shows a line of bright yellow along the diagonal from approximately 100-400 ms (but with a little more generalization from 300-400 ms than was seen for **3_N170**). However, this was accompanied by horizontal and vertical lines of bright yellow at approximately 100 ms. That is, when the decoder was trained at 100-120 ms (along the Y axis), good decoding (yellow values) could be seen from approximately 250 to 400 ms (along

the X axis). Conversely, when the decoder was tested at 100-120 ms (along the X axis), good decoding could be seen for decoders trained at many time points from approximately 250-700 ms (along the Y axis). In other words, the representations present from approximately 100-120 ms were similar to those present at much later time points for this subject.

Although I am showing you these two patterns in two subjects from the same experiment, these two patterns are usually observed in grand averages from different experiments. You can look at the temporal generalization matrix from this experiment by selecting the grand average MVPCset and plotting it using the **Plot/Export Temporal Generalization Matrix** panel. To perform statistical analyses, you can export the single-subject temporal generalization matrices using the **Export** button in this panel. The **Export** tool allows you to save the plots of the matrices in files on your computer drive, the matrices themselves in spreadsheets on your computer drive, or both.

This page titled 12.9: Exercise- Temporal Generalization is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck.

12.10: Demonstration- Decoding in the Frequency-Domain

Although eStudio focuses mainly on time-domain ERPs rather than frequency-domain analyses, we've included a simple feature in our decoding tools that allows you to apply decoding to frequency-domain data. For example, the scalp distribution of alpha-band EEG oscillations varies according to where the participant is attending (Rihs et al., 2007), and the algorithms implemented in eStudio make it possible to decode the direction of attention from these oscillations (Bae & Luck, 2018).

To see how this is done, go to the **EEG** tab and open the **Extract Bin-Epoched Single Trials (BEST)** panel. If you don't have any EEGsets loaded, everything in the panel will be grayed out, but that's okay for now. If you look near the bottom of the panel, you will see a checkbox labeled **Transform into phase-independent power** along with two text boxes. When you are creating a BESTset from an EEGset, you can check this box and enter the desired frequency range in the two text boxes. To extract the alpha band, for example, you could specify 8 to 12 Hz as the frequency band.

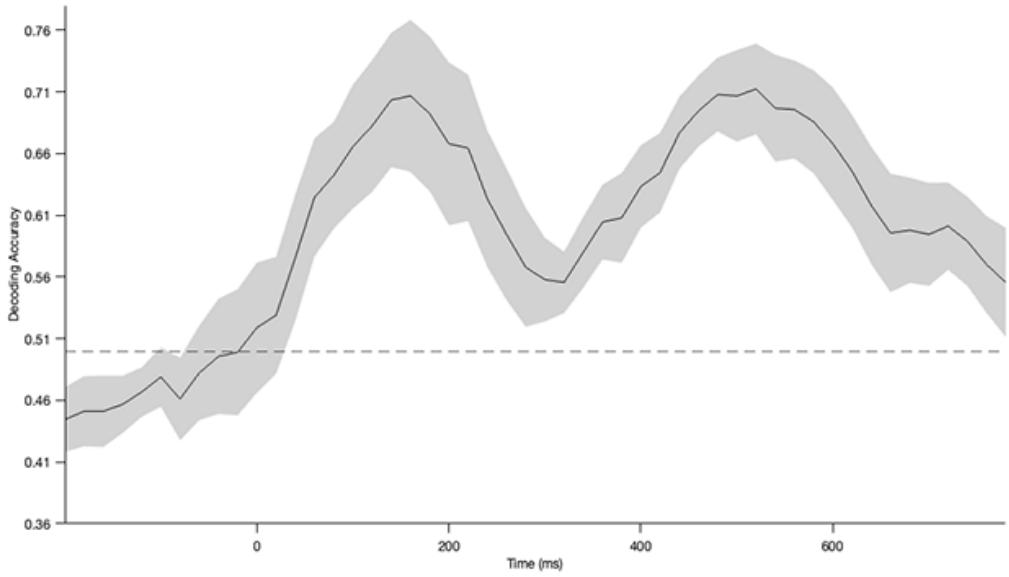
When you do this, each EEG epoch is passed through a bandpass filter with the specified lower and upper bounds, and then the filtered data are passed through the *Hilbert transform* before being saved in the BESTset. The filtering step extracts the desired frequency band, and the Hilbert transform converts the data from a voltage value at each time point to a phase-independent power value at each time point. Consequently, the single-trial epochs in the BESTset will represent the power in the desired frequency band at each time point. You can then just apply the regular decoding process to the resulting BESTset to decode the classes on the basis of the phase-independent power information. In other words, you just need to check the box labeled **Transform into phase-independent power** and specify the frequency range to switch from decoding voltage values to decoding in the frequency domain. It couldn't be much simpler!

As a demonstration, I took the data from the EEG CORE N170 experiment and created new BESTsets with this option set to extract the phase-independent power from 8 to 12 Hz (you can find these BESTsets in the **N170_BEST_Alpha** folder). I then performed the decoding of faces versus cars (you can find the results in the **N170_MVPC_Alpha** folder). I'm not going to have you go through all the steps yourself (although you can if you want).

I did encounter one small problem the first time I tried to perform this analysis. As discussed in Chapter 7 of Luck (in preparation), you lose a lot of temporal information when you apply a narrow bandpass filter like the 8–12 Hz filter I used to extract alpha, and you need a relatively long epoch length to apply the filter. The epoch of -200 to 800 ms used in the ERP CORE N170 experiment was not long enough for the filter to work. I therefore re-epoched the data from -2000 to +2000 Hz prior to creating the BESTsets (you can find the re-epoched data in the **N170_Data_Alpha** folder). Also, when I performed the decoding using the **Multivariate Pattern Classification** panel, I clicked the **Options** button and set the time range for decoding to be **-200 800** (to match the decoding time window in the previous exercise).

After I ran the decoding for each of the 9 participants, I made a grand average, which you can see in Screenshot 12.15. To tell the truth, I wasn't quite sure what to expect. However, alpha is suppressed when you focus on an interesting visual object (as first demonstrated by my intellectual great great grandfather, Lord Adrian, in a paper by Adrian & Matthews, 1934). Faces tend to automatically capture attention, which presumably impacts alpha, so I thought the alpha signal might differentiate between faces and cars. As you can see in the grand average, the alpha signal could in fact be used to decode whether the stimulus was a face or a car, with a peak around 150 ms.

Note that the decoding accuracy first starts rising above chance at approximately 0 ms. This is a result of the temporal smearing that occurs when you isolate a specific frequency band, and it does not mean that the brain had information about whether the stimulus was a face or a car at that time point.



Screenshot 12.15

This page titled 12.10: Demonstration- Decoding in the Frequency-Domain is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck.

12.11: Exercise- Artificially Inflated Decoding with Unbalanced Classes

In most experiments designed for decoding, each of the classes has the same probability. For example, the first experiment described in this chapter had 40 trials for each combination of face ID and emotional expression. If you throw out trials with behavioral errors (or perhaps trials with artifacts), you might not have exactly the same number of trials per class. But if you've excluded only a few trials, it's not a big deal if you have to equate the number of trials per class (by selecting the **Equal Trials, Equal AVGs** and **Across Classes** options).

Sometimes, however, the number of trials differs greatly across classes. This might happen if many more trials were excluded in one class than the others (e.g., because of more behavioral errors in one class). However, large imbalances mainly occur when the experimental design involves presenting one class less often than the others. For example, imagine that you've run an oddball experiment in which there are 100 targets and 900 nontargets, and you want to use decoding to see if the brain activity differs between targets and nontargets. If you decode the data using 10 folds, you would have 10 trials per average/pseudotrial for the targets and 90 trials per average/pseudotrial for the nontargets.

Although this kind of imbalance is not a problem with conventional mean amplitude measures (see Chapter 9 in Luck, in preparation), it produces artificially inflated decoding accuracy. The reason for this artificial inflation is illustrated in Figure 12.3A. This is a simulation of what would happen if the single-trial brain activity was actually the same for targets and nontargets, but the averages for targets were noisier than the averages for the nontargets (because fewer trials were averaged together for the targets than for the nontargets). As you can see from the figure, the center of mass in the electrode space is the same for targets and nontargets, but the greater noise for the averaged targets tends to cause more extreme values. The decoder can then find a decision line that separates more extreme values from less extreme values, which will allow it to decode the classes better than chance. Note that there are no consistent differences between the voltage patterns for targets and nontargets on the individual trials in this imaginary example (except for random noise); the problem arises when we combine differences in the number of trials with averaging, which leads to different amounts of noise in the averages for targets and nontargets and therefore more extreme values for target averages than for nontarget averages.

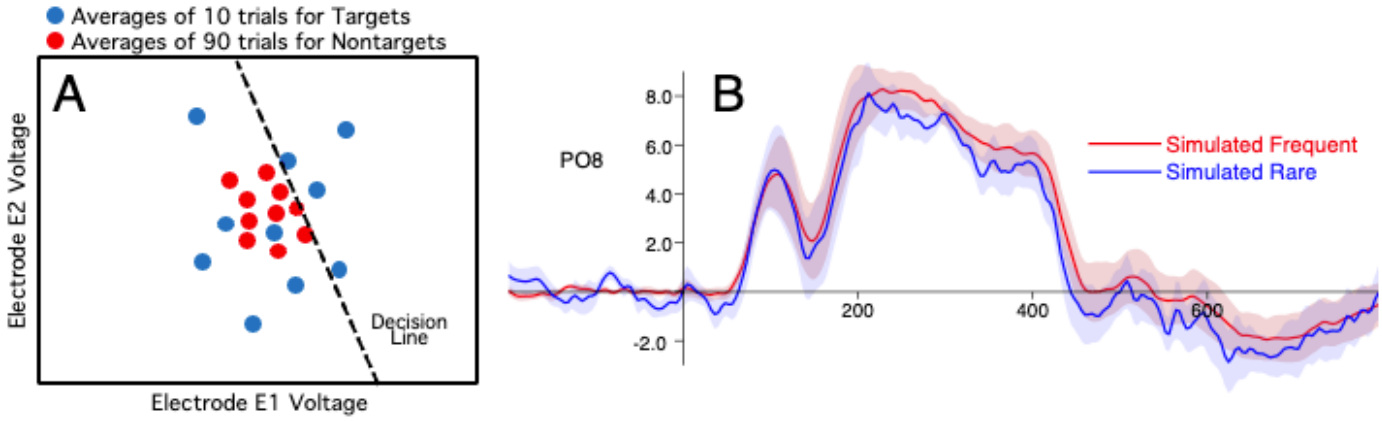


Figure 12.3. (A) Examples of voltage pairs when 10 trials versus 90 trials are averaged together, with no difference between the single-trial values for the two classes. (B) Grand average decoding accuracy from the simulation.

Before I discuss a couple possible solutions for this problem, I'd like you to see the problem for yourself. To make that possible, I created a dataset in which there are no consistent differences between two classes in the single trials, but one class occurs on 10% of trials and the other class occurs on 90% of trials. Specifically, I took the data from the ERP CORE N170 experiment (including face trials, car trials, randomized-phase face trials, and randomized-phase car trials) and I replaced each event code with a 1 for a random 10% of the trials and with a 2 for the other 90% of trials. I then binned the trials according to these randomized event codes, with Bin 1 being the Simulated Frequent condition and Bin 2 being the Simulated Rare condition. The original datasets had 320 trials, and the simulated datasets had an average of 288 trials in Bin 1 and 32 trials in Bin 2. The actual stimuli in a given bin were just as likely to be faces, cars, randomized-phase faces, and randomized-phase cars, so the only differences between the two bins are random noise. In other words, this is a very realistic simulation of a null effect, in which the trials in the two bins are random samples from the same population of actual single trials of EEG data. If we see above-chance

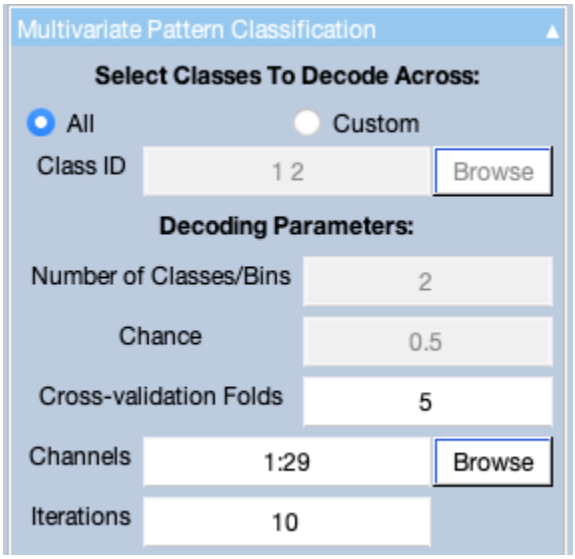
decoding in this null effect situation, this would indicate a problem with the decoding.

The EEGsets with the scrambled event codes are in the folder **N170_Data_Scrambled**. I made averages for each of the 9 participants and put them in the folder **N170_ERPs_Scrambled**. The grand average across participants (for the PO8 electrode site, where the N170 was largest) is shown in Figure 12.3B. As you can see, the ERP for the Simulated Rare bin is much like the ERP for the Simulated Frequent bin, with each of the two waveforms falling well within the standard error of the other waveform. That's what we'd expect given that trials were randomly assigned to these two bins. You can also see that the waveform for the Simulated Rare bin is noisier, with more up-and-down deflections in the baseline (and throughout the waveform). That's what we'd expect given that only 10% of the stimuli were assigned to the Simulated Rare bin.

Given that the trials were randomly assigned to the two bins, there was no consistent information in the single-trial EEG signal that could distinguish between the two bins. However, the fact that the averaged ERPs were noisier for the simulated rare bin than for the simulated frequent bin means that it should be possible, in theory, to determine whether an average came from the simulated rare bin/class or the simulated rare bin/class (as illustrated in Figure 12.3A).

Let's apply decoding to these datasets and see for ourselves. First, clear the existing BESTsets and MVPCsets from eStudio. I've already made the BESTsets for you, so go to the **BESTsets** panel in the **Pattern Classification** tab, click **Load**, navigate to the **N170_BEST_Scrambled** folder, and load all 9 BESTsets. Note that I did not apply any artifact rejection to these BESTsets.

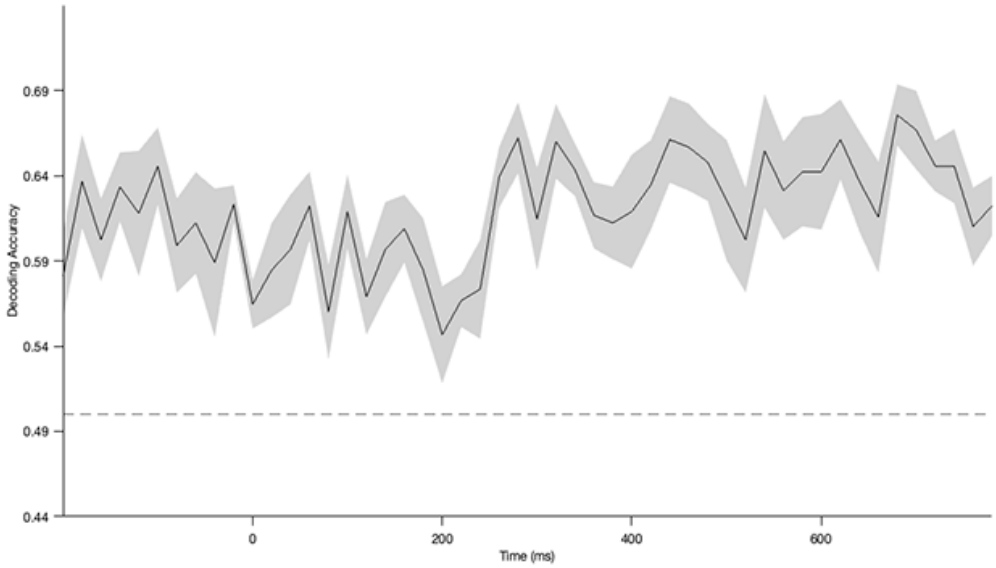
Select all 9 of the BESTsets in the **BESTsets** panel and open the **Multivariate Pattern Classification** panel. Set up the top portion of the panel as shown in Screenshot 12.16. In the bottom portion, set the **Number of trials and averages** section to **Max Trials, Equal AVGs**. This sets up the cross-validation process to use all of the trials in each bin/class, with 5 averages per class (because the number of cross-validation folds is set to 5). You can see this by clicking **Show Summary of trials and averages**. For example, Subject 1 has 279 total trials in the Simulated Frequent bin/class, yielding 55 trials in each of the 5 averaged ERPs for that bin; by contrast, there are only 39 total trials in the Simulated Rare bin/class, yielding 7 trials in each of the 5 averaged ERPs for that bin. This is exactly the situation in which you are likely to get bogus above-chance decoding.



Screenshot 12.16

Now close the summary window and go back to the decoding panel. Near the bottom of the panel, notice that the **Metric** is set to **AUC** (which stands for *area under the curve*). In the previous exercises, we used the **ACC** metric (accuracy, or proportion correct), but now that metric is grayed out and you can't select it. This is because the **Max Trials, Equal AVGs** option is likely to produce artificially inflated decoding accuracy, and when we put together this software package, we wanted to make sure that people don't use this inappropriate option. [The AUC metric is not inflated with this option, so we allow it.]

I therefore created a special script to do the decoding in this invalid way (as if you had used the **Max Trials, Equal AVGs** option with the **ACC** metric). To see the results, go to the MVPCsets panel and load all 9 MVPCsets from the **N170_MVPC_Scrambled_MaxTrials** folder. Select all 9 of them and look at the plots of decoding accuracy in eStudio's main plotting area. Notice that the decoding accuracy is above chance for almost every MVPCset at almost every time point. Now make a grand average using the **Average Across MVPCsets** panel. You should see something like Screenshot 12.17.



Screenshot 12.17

The decoding accuracy in the grand average is obviously well above chance at every time point, even during the prestimulus baseline. That's obviously bogus given that the trials were randomly assigned to the two bins/classes. This demonstrates that decoding is artificially inflated if the averages/pseudotrials are noisier for one class than the other(s). The usual cause of this is having fewer trials in one class, but anything that makes the data noisier for one class will inflate the decoding accuracy.

By the way, just as I always look at the prestimulus baseline prior to the rest of the waveform when I am looking at conventional ERP plots, I always look at decoding accuracy during the prestimulus period first when looking at ERP decoding waveforms. If the prestimulus decoding is above chance, then something is wrong. Looking at the prestimulus decoding has allowed me to detect errors in the decoding methods of the trainees in my lab several times. And it has also allowed me to detect problems in the decoding methods of researchers from other labs. You should also pay close attention to decoding accuracy in the prestimulus period, whether you are looking at your own data or someone else's data.

This page titled 12.11: Exercise- Artificially Inflated Decoding with Unbalanced Classes is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck.

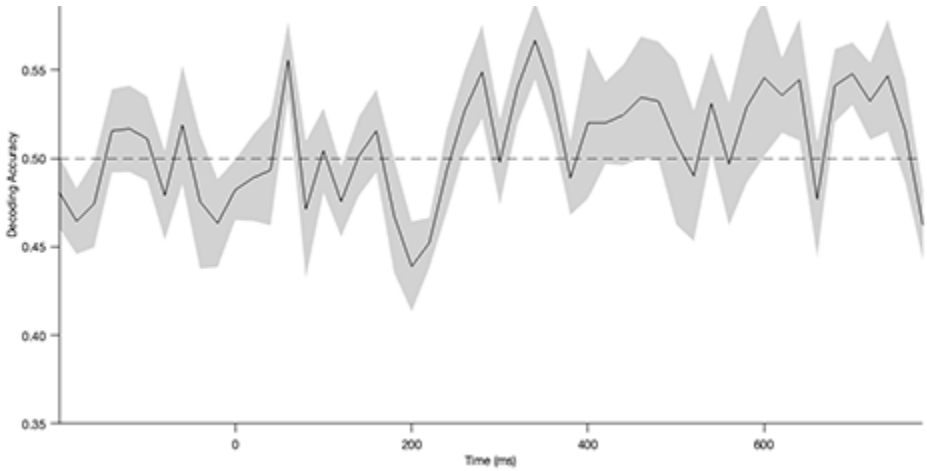
12.12: Exercise- Methods for Appropriate Decoding with Unbalanced Classes

Now that we have seen that differences in the number of trials per class can be a problem, let's look at two solutions. The first is to subsample from the available trials to equate the number of trials per average across classes. In fact, this is what you have done in all the previous exercises, where there were small differences in the number of trials per class. Let's try that with the simulated rare and frequent classes, which have a huge difference in the number of trials per class.

Make sure that the BESTsets from the previous exercise are still loaded and that the top portion of the **Multivariate Pattern Classification** panel is still set up as shown in Screenshot 12.16. In the bottom portion of the panel, set the **#Trials & AVGs** section to **Equal Trials, Equal AVGs** and **Across Classes**. If you click **Show Summary of AVGs and Trials**, you will now see that the number of trials per average/class (the **N(per ERP)** column) is the same for the Simulated Frequent and Simulated Rare classes (e.g., 7 for Subject 1).

Set the **Metric** to **ACC** and make sure that **Temporal generalization matrix** is off. In the **Options**, turn on **Parallelization**, and set **Downsampling** to decode every 20 ms. Then you can either click **Run** or load the MVPCsets that I've already created in the **N170_MVPC_Scrambled_EqualTrials** folder.

Once you have the 9 MVPCsets, select them all in the **MVPCsets** panel. You should see that the decoding accuracy hovers around chance. If you make a grand average, you'll see even more clearly that the values hover around chance, especially in the prestimulus period. This is shown in Screenshot 12.18.



Screenshot 12.18

Although equating the number of trials via subsampling eliminates the inflation of decoding accuracy, it means that we are not using all of the available trials to do the decoding. That seems like a waste. Guanghui Zhang and I therefore developed and validated a very simple approach that equates the number of trials without throwing out any trials (Zhang et al., in preparation). In this approach, we use the same number of trials per average for both classes, but we create more averages/pseudotrials (and therefore more training cases) for the class with more trials.

To see how this works, go to the decoding panel and set the **Number of trials and averages** section to **Equal Trials, Max AVGs** and **Across Classes**. Then click **Show Summary of AVGs and Trials**. You will see that the number of trials per average/class (the **N(per ERP)** column) is still the same for the Simulated Frequent and Simulated Rare classes (e.g., 7 for Subject 1), but there are now more ERPs/pseudotrials (the **B(ERPs)** column) for the Simulated Frequent class (e.g., 39 averages for Subject 1) than for the Simulated Rare class (e.g., 5 averages for Subject 1).

If you want to see the results, you can load the MVPCsets from the **N170_MVPC_Scrambled_EqualTrialsMaxAVGs** folder. Once again, decoding accuracy hovered around chance, showing that this method does not lead to inflated decoding accuracy even though it uses all the trials. When there are real effects present, we find that the **Equal Trials, Max AVGs** approach leads to larger

effect sizes than the **Equal Trials, Equal AVGs** approach, because it uses all the trials. We therefore recommend this approach when you have large differences in the number of trials per class. The **Equal Trials, Equal AVGs** option is fine if the number of trials per class varies only a little (and it is easier to describe in a Methods section).

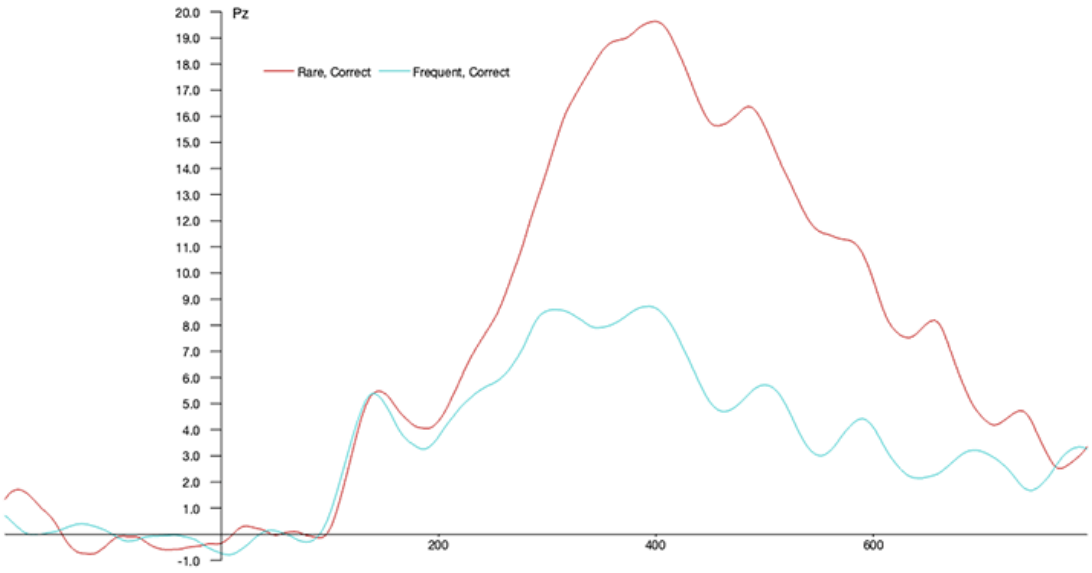
In the **Equal Trials, Max AVGs** approach, there are more training cases for the class with more trials than for the class with fewer trials. You might be concerned that this would bias the decoder to guess the more frequent class. In practice, we have found that this is not a problem when looking at the overall proportion correct. The standard way to deal with this kind of bias is to quantify decoding performance as the area under the receiver operating characteristic (the AUC metric mentioned earlier). Although we have not found it necessary to use AUC with this method to avoid inflation of decoding accuracy, we have found that AUC is still useful because it leads to larger effect sizes (Zhang et al., in preparation). Therefore, we recommend using this metric instead of proportion correct (ACC). AUC will be described in more detail in the next exercise.

This page titled 12.12: Exercise- Methods for Appropriate Decoding with Unbalanced Classes is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck.

12.13: Exercise- Using Area Under the Curve (AUC) with the ERP CORE P3b Oddball Paradigm

In this exercise, you'll take what you learned about decoding with different numbers of trials per class and apply this knowledge to classifying rare targets and frequent standards in the data from the ERP CORE P3b paradigm. The paradigm was described in detail in Chapter 6. To keep things simple, we'll do the decoding only for Subjects 1–5. The preprocessed EEGsets for all 34 subjects can be found in the **P3b_Data** folder. The necessary preprocessing steps have already been applied (e.g., filtering, referencing, artifact correction, bin assignment, epoching). Bin 1 consists of the Rare Targets, and Bin 2 consists of the Frequent Standards. I've excluded trials with behavioral errors (because the P3b depends on how the subject actually responds), but I haven't excluded trials with artifacts (because, as described earlier, this tends to hurt decoding more than it helps).

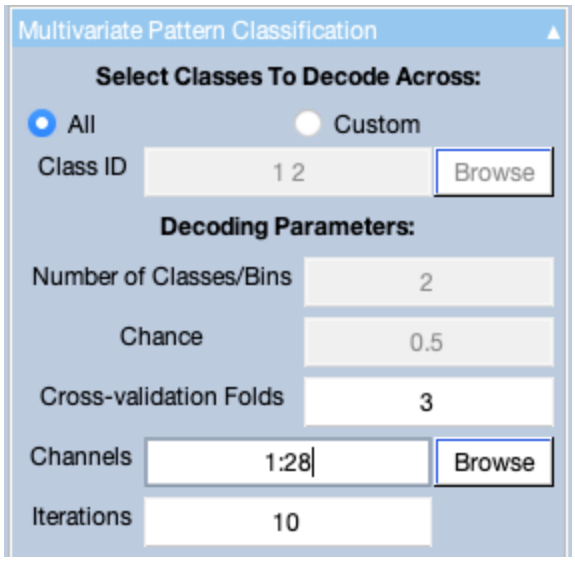
I've created conventional ERPs for Subjects 1-5 (in the **P3b_Data** folder), and Screenshot 12.19 shows the grand average of these 5 participants at the Pz electrode site (where the P3b was largest). You can see that the P3b component was much larger for the Rare Targets than for the Frequent Standards.



Screenshot 12.19

We can potentially gain sensitivity and statistical power by using decoding rather than conventional methods to quantify the extent to which the ERPs are different for the Rare Targets versus the Frequent Standards. However, we need to do the decoding in a way that is not distorted by the fact that there were approximately four times as many Frequent Standards as Rare Targets. As recommended at the end of the previous section, we're going to use the **Equal Trials, Max AVGs** approach to equate the number of trials while using the maximum number of averages/pseudotrials per class.

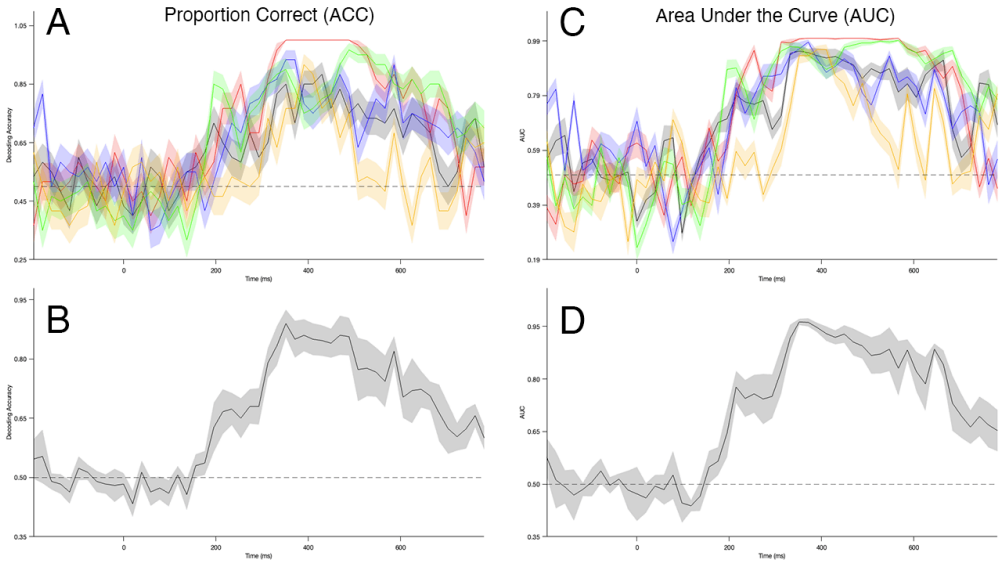
Begin by clearing any loaded BESTsets and MVPCsets from eStudio. I've already converted the EEGsets into BESTsets, so go to the **BESTsets** panel in the **Pattern Classification** tab, click **Load**, navigate to the **P3b_BEST** folder, and load the 5 BESTsets. Select all 5 BESTsets in the BESTsets panel, and then set up the **Multivariate Pattern Classification** panel as shown in Screenshot 12.20. We have only 30-40 trials in the Rare Targets bin, and we want to have 10-20 trials per average/pseudotrial, so we're setting **Cross-validation Folds** to **3**. In the bottom portion of the panel, set the **Number of trials and averages** section to **Equal Trials, Max AVGs** and **Across Classes**, and click **Show summary of trials and averages**. For Subject 1, you should see that we have 11 trials per average and 3 averages for the Rare class, and 11 trials per average and 14 averages for the Frequent class. Thus, we've equated the number of trials per average, but we're using more averages for the Frequent class than for the Rare class so that we don't have to throw away a bunch of trials from the Frequent class.



Screenshot 12.20

Close the summary window and then click on the **Options** button in the decoding panel. Check the **Downsample** box and set it to decode every 19.53 ms. The sampling rate for this experiment was 256 Hz, which means that there was one sample every 3.91 ms. This is why we are decoding every 19.53 ms instead of every 20 ms.

Click **Okay** to go back to the decoding panel, and make sure that the **Metric** is set to **ACC**. Now you can either click **Run** or load the MVPCsets that I've already created in the **P3b_MVPC** folder. If you plot all five of the MVPCsets, you should see something like Screenshot 12.21A. And if you make a grand average of those MVPCsets, you should see something like Screenshot 12.21B.



Screenshot 12.21

Now we're going to repeat the decoding using area under the curve (AUC) as the metric of decoding performance. You can learn exactly what AUC means and how it's calculated by reading Chapter 11 of the third edition of An Introduction to the Event-Related Potential Technique (Luck, in preparation). Briefly, AUC captures the ability of the decoder to distinguish between the classes in a manner that controls for any biases toward one or the other class. Chance is always an AUC of 0.5, and perfect decoding would be an AUC of 1.0.

Let's try it. Keep everything the same as before, except change the **Metric** to **AUC**, and then run the decoding (or load the MVPCsets that I've already created in the **P3b_MVPC_AUC** folder). If you plot all five of the MVPCsets, you should see something

like Screenshot 12.21C. And if you make a grand average of those MVPCsets, you should see something like Screenshot 12.21D. Note that the Y axis in these plots is now **AUC** instead of **Decoding Accuracy**.

The plots of decoding accuracy (ACC) and area under the curve (AUC) look pretty similar. However, the standard errors are a little lower for the AUC metric, indicating that the variability among participants was a little lower. That should mean that the effect size (Cohen's d_z) should be greater for AUC than for ACC. There isn't a direct way to compute this with eStudio, so I wrote a little script for it. The results are shown in Figure 12.4. As expected, the effect size was at least a little larger for AUC than for ACC at most points, and it was much larger for AUC than for ACC from approximately 300-500 ms. This is consistent with a study conducted by Guanghui Zhang, in which he compared ACC and AUC across a large number of decoding studies and found that the effect sizes were typically larger for AUC (Zhang et al., in preparation).

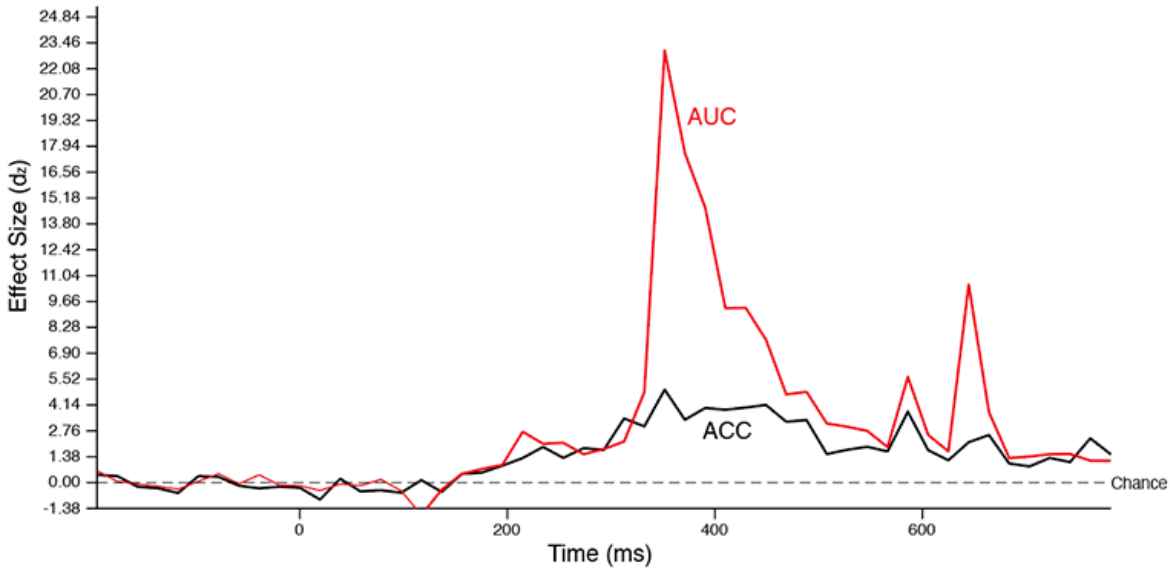


Figure 12.4. Effect size (Cohen's d_z) for decoding between Rare Targets and Frequent Standards in the ERP CORE P3b oddball experiment. The red line shows the effect size when decoding performance was quantified as area under the curve (AUC), and the black line shows the effect size when decoding performance was quantified as proportion correct (ACC). Chance is zero for d_z.

This page titled 12.13: Exercise- Using Area Under the Curve (AUC) with the ERP CORE P3b Oddball Paradigm is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck.

12.14: Matlab Script For This Chapter

To demonstrate how you can run decoding with a script, I've provided a script named **Example_Decoding_Script.m**. It uses the same preprocessed data from the Bae (2021) study that we used in the first few exercises of this chapter. For each participant's data, the script assigns trials to bins, epochs the data, and creates a BESTset. It then performs the face ID decoding and the emotional expression decoding.

The decoding in this script uses the **Equal Trials, MAX AVGs** approach to cross-validation, in which the number of trials is equated across classes (separately for each participant), but the number of averages/pseudotrials per class can vary to use as many of the trials as possible. The number of trials is similar across classes in this dataset, so the number of averages/pseudotrials doesn't actually end up varying across classes, but there is no harm in specifying that option (and it will help in cases where the number of trials is quite different across classes).

This page titled 12.14: Matlab Script For This Chapter is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck.

12.15: Key Takeaways and References

Key Takeaways

- Decoding compares the pattern of voltage across electrode sites for different classes of trials to quantify the extent to which the classes produce consistently different voltage patterns
- If there are differences in the number of trials in each class being decoded, it is important to equate the number of trials used in each average/pseudotrial.
- If there are differences in the number of trials across experimental conditions, and you want to compare decoding accuracy across conditions, you must also equate the number of trials used per condition
- If there are differences in the number of trials across participants, and you want to compare decoding accuracy across different groups of participants (or you are looking at individual differences), you must equate the number of trials used per participant
- Decoding can be used instead of conventional mean amplitude analyses to improve sensitivity, effect sizes, and statistical power for assessing differences between experimental conditions
- Temporal generalization analyses allow you to assess the extent to which similar neural representations are used at different time points
- Decoding can be applied to frequency-domain data as well as time-domain data
- When the number of trials differs across classes, this can artificially inflate decoding accuracy, leading to above-chance decoding even during the prestimulus baseline period
- Differences in the number of trials across classes can be addressed by equating the number of trials per average across classes, but the number of averages can be greater in the classes with more trials to avoid throwing out trials unnecessarily
- Decoding performance can be quantified in terms of both proportion correct and area under the curve (AUC), with AUC often producing larger effect sizes

References

Adrian, E. D., & Matthews, B. H. C. (1934). The Berger rhythm: Potential changes from the occipital lobes in man. *Brain, 57*, 355–385. https://doi.org/https://doi.org/10.1093/brain/57.4.355

Bae, G. Y. (2021). The Time Course of Face Representations during Perception and Working Memory Maintenance. *Cerebral Cortex Communications, 2*. https://doi.org/10.1093/texcom/tgaa093

Bae, G. Y., & Luck, S. J. (2018). Dissociable decoding of working memory and spatial attention from EEG oscillations and sustained potentials. *The Journal of Neuroscience, 38*, 409–422. https://doi.org/https://doi.org/10.1523/JNEUROSCI.2860-17.2017

Carrasco, C. D., Bahle, B., Simmons, A. M., & Luck, S. J. (2024). Using multivariate pattern analysis to increase effect sizes for event-related potential analyses. *Psychophysiology, 61*, e14570. https://doi.org/http://doi.org/10.1111/psyp.14570

Carrasco, C. D., Simmons, A. M., Kiat, J. E., & Luck, S. J. (2025). Enhanced working memory representations for rare events. *Psychophysiology, 62*, e70038. https://doi.org/https://doi.org/10.1111/psyp.70038

Dietterich, T. G., & Bakiri, G. (1995). Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research, 2*, 263–286. https://doi.org/https://doi.org/10.1613/jair.105

Hebart, M. N., & Baker, C. I. (2018). Deconstructing multivariate decoding for the study of brain function. *NeuroImage, 180*, 4–18. https://doi.org/10.1016/j.neuroimage.2017.08.005

Kappenman, E. S., Farrens, J. L., Zhang, W., Stewart, A. X., & Luck, S. J. (2021). ERP CORE: An open resource for human event-related potential research. *NeuroImage, 225*, 117465. https://doi.org/https://doi.org/10.1016/j.neuroimage.2020.117465

King, J.-R., & Dehaene, S. (2014). Characterizing the dynamics of mental representations: The temporal generalization method. *Trends in Cognitive Sciences, 18*, 203–210. https://doi.org/10.1016/j.tics.2014.01.002

Lakens, D. (2013). Calculating and reporting effect sizes to facilitate cumulative science: A practical primer for t-tests and

ANOVAs. *Frontiers in Psychology, 4*, 863. https://doi.org/10.3389/fpsyg.2013.00863

Luck, S. J. (in preparation). *An Introduction to the Event-Related Potential Technique, Third Edition*. MIT Press.

Rihs, T. A., Michel, C. M., & Thut, G. (2007). Mechanisms of selective inhibition in visual spatial attention are indexed by α-band EEG synchronization. *European Journal of Neuroscience, 25*, 603–610. https://doi.org/10.1111/j.1460-9568.2007.05278.x

Walther, A., Nili, H., Ejaz, N., Alink, A., Kriegeskorte, N., & Diedrichsen, J. (2016). Reliability of dissimilarity measures for multi-voxel pattern analysis. *NeuroImage, 137*, 188–200. https://doi.org/10.1016/j.neuroimage.2015.12.012

Zhang, G., & Luck, S. J. (2025). Assessing the impact of artifact correction and artifact rejection on the performance of SVM- and LDA-based decoding of EEG signals. *NeuroImage, 316*, 121304. https://doi.org/https://doi.org/10.1016/j.neuroimage.2025.121304

Zhang, G., Wang, X., & Luck, S. J. (2026). Evaluating the effects of regularization and cross-validation parameters on the performance of SVM-based decoding of EEG data. *Cortex, 201*, 171–189. https://doi.org/10.1016/j.cortex.2026.05.005

Zhang, G., Wang, X., & Luck, S. J. (in preparation). Maximizing EEG decoding for experimental designs with different numbers of trials per class. Manuscript in Preparation.

This page titled 12.15: Key Takeaways and References is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck.

13: Appendix 1- A Very Brief Introduction to EEG and ERPs

This appendix provides a very brief introduction to the EEG and ERPs (and necessarily oversimplifies some complex issues). It is meant to provide enough background for you to understand the big picture of the analysis procedures described in this book, but it is just a start.

The EEG

The neurons in your brain produce two main kinds of electrical potentials (voltages), called *postsynaptic potentials* and *action potentials*. We can't ordinarily pick up action potentials from the scalp, but the postsynaptic potentials produced by individual neurons can propagate through the brain, skull, and scalp. If we put electrodes on the scalp, as shown in Figure A1.1, we can pick up the propagated postsynaptic potentials. These propagated voltages are what we call the *electroencephalogram* or *EEG*. This is a simplification; if you want to know the full story, I recommend Jackson and Bolger (2014) or Buzsáki et al. (2012).

The postsynaptic potentials produced by individual neurons sum together and spread widely before reaching the scalp, so any given electrode is picking up voltages generated by thousands or millions of neurons in a broad set of brain regions. As a result, a voltage picked up at a given electrode site may not represent activity coming from the cortex directly under the electrode.

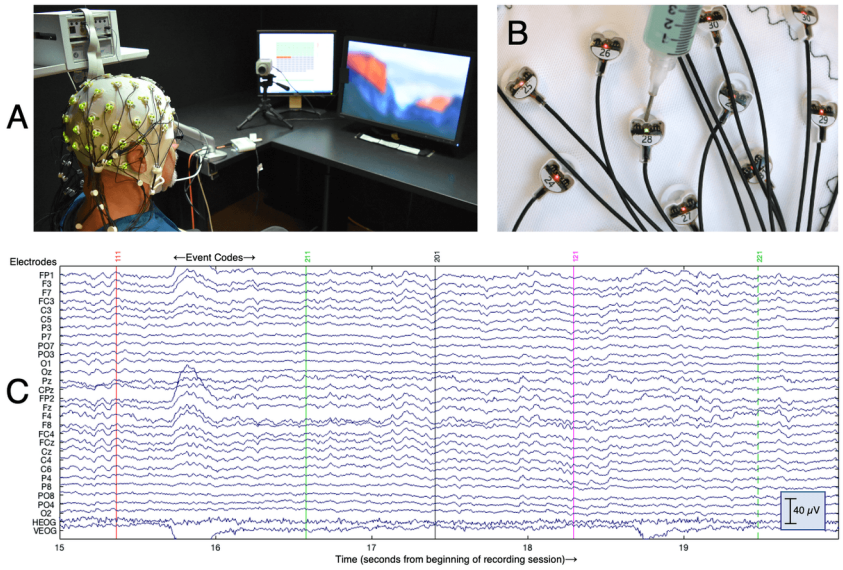


Figure A1.1. Typical EEG recording. (A) The subject is in the lab, wearing an electrode cap and facing the computer monitor where the stimuli will be presented. (B) Each electrode is a metal pellet encased in plastic. A conductive gel is squirted into the electrode holder to make an electrical contact between the electrode pellet and the skin. (C) A 5-second period of EEG. Each waveform is the signal from one of the 30 electrodes that was used in this experiment. The X axis is time (in seconds), and the Y axis is voltage (in microvolts, μV). The colored vertical lines indicate event codes, which are numeric codes that mark the time and identity of the stimuli that were presented to the subject. The EEG signals are referenced to the right mastoid. HEOG (horizontal electrooculogram) is the potential between two electrodes placed just lateral to the eyes, and VEOG (vertical electrooculogram) is the potential between electrodes placed under and over the right eye.

The EEG may be recorded from just a few electrodes or from as many as 256 electrodes, but most studies include between 16 and 64 electrodes. As shown in Figure A1.2, there is a widely used electrode naming system (the International 10/20 System) in which each electrode is given a 1-2 letters to indicate the general region of the head and a number to indicate the left-right position (or a "z" for the midline). For example, electrode P1 is over parietal cortex, just slightly to the right of the midline, and electrode FCz is on the midline, between the frontal and central regions.

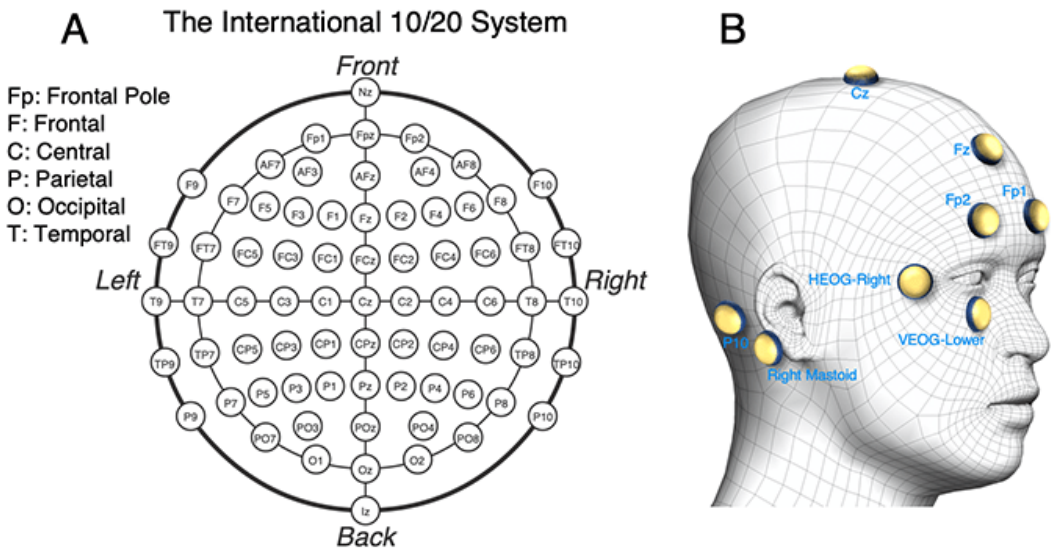


Figure A1.2. (A) The International 10-20 system. In this system, each electrode is named with 1-2 letters indicating the general region of the scalp and a number indicating the left-right location. The numbers are odd for the left hemisphere, even for the right hemisphere, and get larger as we move away from the midline (which is labeled with a "z" to indicate "zero"). For example, Fp2 is over the frontal pole, just to the right of the midline, and Cz is over the central sulcus on the midline. (B) Additional common electrode sites. Electrodes are often placed over the left and right mastoid processes (the bony protrusions just behind the earlobes) as reference electrodes. The horizontal electrooculogram (HEOG) is often recorded from electrodes just lateral to the left and right eyes to detect horizontal eye movements. The vertical electrooculogram (VEOG) is often recorded from an electrode just under the left or right eye (referenced to the corresponding Fp1 or Fp2 electrode over the eye) to detect blinks and vertical eye movements.

The Reference Electrode

A voltage is the *potential* for electrical charges to move from one place to another (e.g., from one terminal of a car battery to the other). As a result, the EEG is always a voltage between two electrodes, which we call the *active electrode* and the *reference electrode*. Typically, most or all of the active electrodes share the same reference electrode. Consider, for example, the EEG waveforms in Figure A1.1.C. Each waveform has a label, which indicates the active electrode for that waveform, and all of these channels except the bottom two used the same reference electrode, which was located on the *mastoid process* (the bony protrusion behind the ear, shown in Figure A1.2.B). This means that the waveform labeled F7 shows the electrical potential between the F7 electrode site and the mastoid. In theory, we would like a reference that is electrically neutral and does not impact the observed waveform. In practice, however, there is no neutral site, and the waveform for a given channel is equally impacted by activity arising from the active and reference electrodes.

The horizontal and vertical electrooculogram signals at the bottom of Figure A1.1.C (labeled HEOG and VEOG) use different reference electrodes. The HEOG signal uses HEOG-right as the active electrode (see Figure A1.2.B) and the mirror-image HEOG-left electrode as the reference. The VEOG signal uses the VEOG-lower electrode as the active electrode and the FP2 electrode as the reference. These *bipolar* configurations make it easier to detect eye movements and blinks, which are large artifacts that propagate across the scalp.

Choosing the best reference electrode can be challenging. Fortunately, you can re-reference your data offline to any electrode (or combination of electrodes) that was in the original recording.

Artifact Rejection and Correction

Some portions of the EEG are contaminated by very large artifacts arising from sources such as eye blinks, eye movements, and muscle activity. In some studies, these epochs are excluded from all analyses, which is called *artifact rejection*. In other studies, a mathematical procedure is used to estimate the contribution of the artifacts and remove them from the epochs, which is called *artifact correction*. Many studies use correction for some artifacts and rejection for others.

Signal Averaging and Event-Related Potentials

The EEG is an incredibly complex signal that includes activity generated throughout the cortex from hundreds or even thousands of processes, all of which are summed together in our scalp electrodes. As a result, you can't get a lot of information about specific neurocognitive processes from looking at the raw EEG. The raw EEG signal can tell you if someone is asleep, awake but zoned out, or highly alert and attentive, and you can sometimes see evidence of pathology (e.g., epileptic spikes). Signal processing methods must be applied to the EEG data to isolate specific processes. Perhaps the simplest of these methods is *signal averaging*, which we use to extract ERPs from the EEG.

Signal averaging is illustrated in Figure A1.3. The goal of signal averaging is to isolate the electrical *potentials* (voltages) that are *related* to specific *events* (i.e., the *event-related potentials*). Signal averaging accomplishes this by assuming that an event such as a stimulus produces a similar waveform on each trial, and random noise is added to this waveform to produce the EEG that we record. If we average together the recorded EEG following many events, then the noise will "average out" and the signal will remain in the average. Figure A1.3.A illustrates the EEG from a single electrode. Whenever an event such as a stimulus or response occurs, an *event code* is inserted into the EEG data file to mark the time of the event. In this example, the events are stimuli (e.g., shapes flashed on a computer monitor).

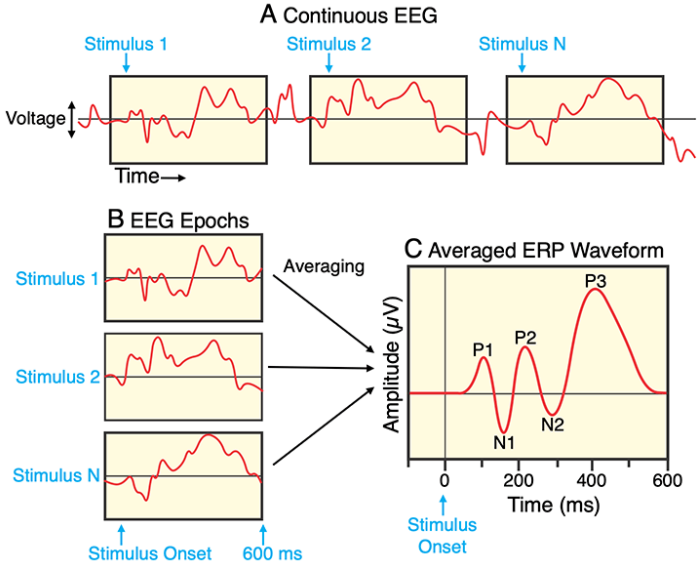


Figure A1.3. The signal averaging process. The EEG is recorded as a continuous signal (A) while stimuli are presented. An event code marks the onset of each stimulus. An epoch of EEG data is extracted for each stimulus (B), beginning slightly before stimulus onset (e.g., 100 ms) and extending for a fixed period after stimulus onset (e.g., 600 ms). The single-trial epochs are lined up and then averaged together to create an averaged ERP waveform (C). That is, the averaged value at time *t* is the average voltage at time *t* across all *N* epochs. The X axis is time in thousandths of a second (milliseconds, ms), and the Y axis is voltage in millionths of a volt (microvolts, μV). In a typical experiment, there would be several different types of stimuli, and a separate averaged ERP waveform would be created for each stimulus type. We would also have a separate waveform at each electrode site.

For each event, we extract an *epoch* of EEG data surrounding that event. In this example, the epochs begin 100 ms before stimulus onset (giving us a 100-ms *prestimulus baseline* period) and extend until 600 ms after stimulus onset. As shown in Figure A1.3.B, all of the epochs for a given type of stimulus are then lined up with respect to time zero. The voltage at a given time for a given stimulus is assumed to be the consistent ERP at that time plus random noise. If we simply average all the epochs together, the noise will largely cancel out, and the averaged ERP waveform will mainly consist of the consistent ERP response (plus some residual noise because we always have a finite number of trials). The number of trials required to produce an acceptable average varies widely across experiments and depends on factors such as how large the ERP is and how much noise is in the EEG. Note that artifact rejection and/or correction are applied prior to averaging.

What Do We Mean by "Noise"?

In ERP research, we use the term *noise* to refer to any uncontrolled variability in the EEG or in the averaged ERP waveform. We are assuming that the EEG consists of a consistent ERP signal on each trial plus random variability. Anything that causes the EEG to deviate from the consistent ERP signal is considered noise. If we repeated an experiment multiple times for a given participant (assuming no fatigue, no learning, etc.), any differences in the averaged ERP waveforms across repetitions would be a result of this noise. Some of this noise comes from electrical devices in the recording environment and is unintentionally picked up by the recording electrodes. Some of it comes from nonneural biological sources, such as blinks and muscle contractions. Some of it comes from brain activity that is unrelated to the stimuli. All else being equal, an averaged ERP waveform will have less residual noise if more trials are averaged together.

ERP Peaks and Components

Figure A1.3.C shows a hypothetical averaged ERP waveform for a visual stimulus recorded at an occipital electrode site (with no noise). The X axis is time, with stimulus onset at time zero. The Y axis is amplitude. The prestimulus period is flat because there should be no consistent stimulus-related activity prior to the stimulus. The waveform begins to deviate from zero at approximately 50-60 ms, which is when visual information typically reaches visual cortex. There is no delay between the postsynaptic potentials generated in the brain and the voltage deflections that we see in an ERP waveform. As a result, the sequence of voltages over time in the ERP waveform corresponds to the sequence of sensory, cognitive, affective, and motor processes that follow the onset of a stimulus.

The ERP waveform consists of a set of positive-going and negative-going *waves* or *peaks*, which are related in a complicated way to a set of underlying *components* in the brain that reflect specific neurocognitive processes. We can't directly see these components, but we try to draw inferences about them from the observed scalp ERP waveforms (see Chapter 2 in Luck, 2014, for a detailed discussion).

Most ERP components are named with a *P* or an *N* to indicate whether they are positive-going or negative-going, followed by a number to indicate the timing. For example, *N1* is the first major negative wave, and *P3* is the third major positive wave. If the number is large (>10), it instead indicates the approximately latency of the peak in milliseconds (e.g., *N170* for a negative-going wave that peaks at 170 ms). Sometimes, a component is given a name that reflects the conditions under which it is observed (e.g., the *error-related negativity* for a negative voltage that is present when the participant makes an incorrect response). Note that the relationship between an underlying component and the observed peaks and waves is complex, and the component names can sometimes be misleading. For example, the P1 elicited by a visual stimulus is completely unrelated to the P1 elicited by an auditory stimulus, but the P3 is the same for auditory and visual stimuli. A detailed discussion is provided in Chapters 2 and 3 of Luck (2014) and by Kappenman and Luck (2012).

To statistically analyze ERP data, researchers typically start by obtaining an amplitude or latency *score* from each individual participant's averaged ERP waveform in each group or condition. Then these scores are entered into a statistical analysis that is much like the analysis of a behavioral variable such as response time. For example, you might find the *peak latency* (the time point when the voltage reaches its maximum value) for the P3 wave in each participant's ERP waveform and then determine whether the latencies are significantly later in a patient group than in a control group using a *t* test.

References

Buzsáki, G., Anastassiou, C. A., & Koch, C. (2012). The origin of extracellular fields and currents—EEG, ECoG, LFP and spikes. *Nature Reviews Neuroscience*, *13*, 407–420. https://doi.org/10.1038/nrn3241

Jackson, A. F., & Bolger, D. J. (2014). The neurophysiological bases of EEG and EEG measurement: A review for the rest of us. *Psychophysiology*, *51*(11), 1061–1071. https://doi.org/10.1111/psyp.12283

Kappenman, E. S., & Luck, S. J. (2012). ERP components: The ups and downs of brainwave recordings. In S. J. Luck & E. S. Kappenman (Eds.), *The Oxford Handbook of ERP Components* (pp. 3–30). Oxford University Press.

This page titled 13: Appendix 1- A Very Brief Introduction to EEG and ERPs is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 12: Appendix 1: A Very Brief Introduction to EEG and ERPs by Steven J Luck is licensed CC BY 4.0.

14: APPENDIX 2- TROUBLESHOOTING GUIDE

Overview

Computers can be maddening. No matter how much we test ERPLAB, we still get occasional bug reports. There can also be bugs in EEGLAB, in Matlab, or in your operating system. Even if there isn't a bug, you might encounter a problem because you specify an incorrect setting or leave out a necessary step. And if you're writing your own scripts, you're likely to introduce your own bugs. I can't tell you how much time I spent troubleshooting problems when I was writing this book!

As I mentioned in Chapter 1, you should look at these problems as an opportunity for working on your general problem-solving skills. Indeed, I find that troubleshooting computer problems is like a microcosm of science. You generate hypotheses, you collect data to test those hypotheses, and you make sure that you can replicate both the problem and the solution.

In this Troubleshooting Guide, I've provided a variety of hints and strategies for solving the most common types of problems you're like to encounter. This includes both specific strategies for issues that often arise in Matlab, EEGLAB, and ERPLAB, along with general strategies for solving computer problems and debugging computer programs. I've also provided some exercises to help you learn to interpret Matlab's error messages and to monitor the operation of a script while it's running.

If you're just using the ERPLAB Studio GUI, you only need to read the first two sections (A2.1 and A2.2). The remaining sections are primarily for scripting.

14.1: Data for This Appendix

14.2: A2.1 The First Step

14.3: A2.2 Some Basic Solutions

14.4: A2.3 Taking a Scientific Approach

14.5: A2.4 Deciphering Matlab's Error Messages

14.6: A2.5 Debugging Scripts by Performing Experiments and Collecting Data

14.7: A2.6 Avoiding Bugs in Your Scripts with Good Programming Practices

14.8: A2.7 References

This page titled 14: Appendix 2- Troubleshooting Guide is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 13: Appendix 2: Troubleshooting Guide by Steven J Luck is licensed CC BY 4.0.

14.1: Data for This Appendix

The data we'll be using for the exercises in this appendix can be found in the Appendix_2 folder in the master folder: https://doi.org/10.18115/D5G045.

This page titled 14.1: Data for This Appendix is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck.

14.2: A2.1 The First Step

Here's something that should be obvious but often isn't: **When you run into a problem you can't easily solve, you should read the documentation and online help**. Over half of the "bug reports" we get for ERPLAB aren't bugs at all but are simple misunderstandings. These misunderstandings typically arise when someone starts using ERPLAB without spending the time to go through our tutorial, when someone tries to use a complicated feature without first reading the documentation for that feature, or when someone encounters a common issue but hasn't consulted our Frequently Asked Questions page. In tech support, these are called **RTFM** problems. Google **RTFM** if this is an unfamiliar term and you'd like a quick smile.

Here are the main sources of EEGLAB and ERPLAB information:

- EEGLAB: Online Documentation, Frequently Asked Questions
- ERPLAB: Online documentation, Frequently Asked Questions, Tutorial

You can also often find solutions by typing something like **ERPLAB binlister** or **EEGLAB ICA** into Google or some other search engine. You can also post questions to the email list for EEGLAB or the email list for ERPLAB. But please don't send me a personal email—I don't have the bandwidth to provide individualized tech support for everyone who uses ERPLAB. However, if you find a problem with this book (e.g., an error, a missing file), please send me an email so I can fix it.

This page titled 14.2: A2.1 The First Step is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 13.1: A2.1 The First Step by Steven J Luck is licensed CC BY 4.0.

14.3: A2.2 Some Basic Solutions

Resetting and Restarting

If you ask someone who works at an IT help desk, they'll tell you that a huge proportion of computer problems can be solved by restarting the computer. Computers are insanely complicated, and there are millions of idiosyncratic problems that can accumulate as the computer carries out more and more operations. A simple restart returns everything to its default status. A restart is an especially good strategy if you start having "weird" problems (e.g., when things that worked fine a few minutes ago are no longer working).

However, restarting your computer can take some time, so when I run into a weird problem when analyzing ERP data, my first step is just to quit and relaunch eStudio. In fact, I try to prevent those problems by quitting and restarting eStudio whenever I'm about to start something new. I also like to type **clear all** in the Matlab command window prior to restarting eStudio, which clears out any variables that have been set (and I also sometimes type **close all** to close any windows that might have been opened previously).

It's also possible that some option has been set that you've forgotten about, which causes eStudio to do something unexpected. You can reset all the options to their defaults by clicking the **Reset** button (in the lower right corner of the plotting region).

If that doesn't work, I try resetting ERPLAB's *working memory*. This is a file that ERPLAB uses to store your current settings. The problem you're having might be a side effect of one of your settings, or the settings may have become corrupted, so clearing the settings sometimes helps. To do this, go to the **Settings** menu in the top left corner of the eStudio GUI and select **Memory Settings > Reset Working Memory**. If resetting the working memory doesn't work, I try restarting Matlab. Again, this is especially useful when I'm encountering a weird problem.

If none of the above steps solve the problem, and the problem is really odd, I try restarting my computer. But that's very rare. And just like I restart eStudio on a regular basis to prevent problems, I recommend restarting your computer once a week or so. If the weird problem persists, you might try moving to a different computer to see if that solves the problem.

Finally, you can try uninstalling and reinstalling EEGLAB and ERPLAB. Although rare, it is possible that the code has been corrupted. I once encountered a case of a user who reported that one of the ERPLAB windows was missing some buttons, and she found that reinstalling ERPLAB solved the problem.

Missing Channel Locations or Data Quality Metrics in Datasets or ERPsets

Are your channel locations or data quality metrics missing from an EEGset or ERPset? The most common reason for this is that you have applied **Channel Operations** or **Bin Operations** to your data.

When you apply Channel Operations, ERPLAB may not be able to figure out the location information for your new channel. For example, when you create a new channel by averaging together a set of existing channels, how can ERPLAB know the 3-D location of this new channel? Indeed, this average across channels doesn't really have a location. There is a box you can check in the GUI labeled **Try to preserve location information**, and it can preserve the channel locations in many cases (e.g., when you re-reference your data). But it uses a fairly simple algorithm, and it may strip out your previous channel location information if the algorithm can't figure out a sensible set of locations.

A similar issue arises for the data quality metrics that are computed during averaging. If you modify the channels using **Channel Operations** or modify the bins using **Bin Operations** to your data, ERPLAB has no way of knowing whether the data quality metrics are still valid. For example, if you create a new bin with a difference wave or an average of two prior bins, what is the data quality for this new bin? There may be analytic solutions for some cases, but these solutions would require many assumptions. As a result, ERPLAB strips out the data quality information from the ERPset that is created when you run **Channel Operations** or **Bin Operations**. Our logic in stripping out the channel location information or data quality metrics in these situations is that we'd rather eliminate the information than provide you with incorrect or misleading information.

Solutions to Miscellaneous Common Problems

- Make sure the MATLAB PATH is set correctly (especially if you're having problems right after installing a new version). See the section describing the Matlab PATH near the end of Chapter 1 for more details.
- If you can't get eStudio to launch, make sure that ERPLAB is installed inside the **plugins** folder within the **EEGLAB** folder (e.g., eeglab2024.0 > plugins >erplab). You should see a folder named **studio_functions** inside that folder.
- Make sure you're using the right version of Matlab, EEGLAB, and ERPLAB.
- Some ERPLAB functions (e.g., filtering) require the Matlab Signal Processing Toolbox. You may need to install this toolbox. If you purchased Matlab yourself, you may need to buy this toolbox from The Mathworks. If you obtained Matlab through your institution, ask your IT group how to get this toolbox.
- If eStudio or Matlab doesn't seem to respond to the mouse or keyboard, you may have a window hidden somewhere that's waiting for input. For example, if you are performing manual artifact detection in one window, and then you look at another window, the artifact detection GUI may be waiting for input, and nothing else will work right until you close that GUI.
- If you load an EEGset but some or all of the channels appear to be missing in the plot region, the DC offset of the EEG signal is probably shifting the EEG outside of the range that is visible in the plotting window. To solve this, select **Remove DC** in the **Plot Settings** panel.
- EEGLAB has its own routine for extracting epochs from the continuous EEG (**EEGLAB > Tools > Extract epochs**), but do not use it!!! Instead, use the **Extract bin-based epochs** panel.
- EEGLAB and ERPLAB often print information in the command window that can help you figure out why things aren't working properly.
- Don't forget to read the Frequently Asked Questions (FAQ) pages for both EEGLAB and ERPLAB.

This page titled 14.3: A2.2 Some Basic Solutions is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 13.2: A2.2 Some Basic Solutions by Steven J Luck is licensed CC BY 4.0.

14.4: A2.3 Taking a Scientific Approach

You're a scientist, right? This means that you've been trained to develop and test hypotheses. That's exactly what you do when you're troubleshooting and debugging. That is, you need to come up with a hypothesis about the nature of the problem and then figure out how to test that hypothesis. This section will flesh out the scientific approach to finding and fixing problems with ERPLAB and EEGLAB.

The Importance of Replication

If you really want to understand a problem, you first need to make sure that you can replicate it. Does it persist after you restart Matlab? Do you get the same problem on a different computer?

Also, once you think you've found the solution, you should make sure that you can make the problem reappear and then disappear again. For example, imagine that a part of ERPLAB keeps crashing when you try to run a particular processing step, and you try 12 different things to make it stop crashing. And you finally get it to work after installing a new version of EEGLAB. Was the EEGLAB version really the problem? After all, you probably updated the PATH when you installed the new version of EEGLAB, so perhaps the PATH was the problem. If you want to be sure that it was a version problem, you should try going back to the previous version and seeing if ERPLAB crashes again. And then you should verify that updating the version again eliminates the crash. In experimental psychology, this is often called the A-B-A-B experimental design.

In many cases, you don't actually care whether you fully understand the problem, and this A-B-A-B approach is unnecessary. Once the program is no longer crashing, you may not care what was causing the crash. But in other cases it's really important to understand the problem. For example, imagine that you're writing a script, and you find that you can get your script to run by sending some parameter that you don't really understand when calling an ERPLAB function. Do you really trust that you've now solved the problem? In this kind of situation, the A-B-A-B approach is usually worth the time and effort.

Start with a Literature Review

When you're designing an ERP experiment, you already know that it's important to read the relevant literature so that you know what has already been done in your research area and so that you can learn about useful methods for answering your scientific question. Without the necessary background knowledge, you probably won't have a good hypothesis and probably won't design a good experiment.

When troubleshooting or debugging, the analog is to read the documentation, do the tutorials, and scan the frequently asked questions page. If you don't take these steps first, you're likely to waste a lot of time pursuing bogus hypotheses about the source of the problem and trying solutions that are unlikely to work.

Carefully Observe the World and then Develop a Hypothesis

You were probably taught the "scientific method" when you were in elementary school. In the usual version, a scientist observes the world, develops a hypothesis, and then conducts an experiment to test the hypothesis. This is a gross mischaracterization of how science actually works (see, e.g., Feyerabend, 1993). However, this oversimplification is useful for troubleshooting and debugging, especially insofar as it proposes that you should carefully observe the world in the process of developing your hypothesis about the source of the problem.

In the context of software troubleshooting, this means that you should look carefully at the inputs and outputs. The main inputs are the data that are being processed and the parameters that are specified (either the settings in the GUI or the values that are being sent to a function in a script). The main outputs are the output data created by eStudio (e.g., EEGsets and ERPsets), error messages, and the messages that are printed by EEGLAB and ERPLAB in Matlab's command window during every processing step. I particularly recommend taking a careful look at the parameters in your script and the messages printed to the command window. You can also take a look at the values of the various fields within the EEG and ERP variables for clues.

Matlab error messages often appear to be written in a language that you don't speak (*Programmerese*). But if you spend enough time looking at the messages, you may find that you know enough of the words to glean some valuable information. You may be

able to get some insight by typing the key part of the message into a search engine, enclosed in double quotes and preceded by **matlab**. For example, if you get the error message **character Vector Is Not Terminated Properly**, you can type this into Google: **matlab "character Vector Is Not Terminated Properly"**. A later section of this appendix provides more detailed information about how to decipher Matlab's error messages.

The error message will tell you which lines of code generated the error message, both in your script and inside a function you were calling when the problem occurred. This is a major clue. You can then open the code for the function and see what was happening in that function. But keep this in mind: The actual problem in the code may have been several lines before the problem was detected, so you should look at that whole section of code.

Once you've done your careful observation, you should be able to develop a hypothesis.

Run Experiments and Collect New Data to Refine and Test Your Hypothesis

Now it's time to test your hypothesis. Sometimes this is trivial: You hypothesize that you need to update your version of EEGLAB, and you then see if updating EEGLAB solves the problem. But in other cases, your hypothesis does not specify the solution, but is instead a more general hypothesis, such as which part of the code is the problem or which variable might not be set properly. In these cases, you need to figure out how to test your hypothesis.

The method for testing your hypothesis will, of course, depend on the nature of your hypothesis. However, there are two general strategies, which are analogous to recording versus manipulating brain activity. That is, you can examine the representations that your code produces (e.g., values of variables, output files) at various steps in processing, or you can attempt to modify the operation of the code (e.g., by changing the inputs or by turning off various parts of the code). I'll say more about these options in a later section.

Science is Social!

One of the main shortcomings of the elementary school version of the scientific method is that it does not treat science as a deeply human, social activity. Much of science is driven by vanity, competition, and cooperation. I wouldn't recommend vanity or competition when you're troubleshooting or debugging, but cooperation is extremely valuable. More specifically, if you get stuck, get advice from other people. This could be a more experienced EEGLAB/ERPLAB/Matlab person in your lab or your department. Or it could be some stranger on the Internet. (The part of the Internet inhabited by programmers and scientists is much kinder and more helpful than most other parts.)

Often, your question has already been answered. EEGLAB has an active listserv, which you can join. You can then search the archive to see if your question has already come up. ERPLAB also has a listserv with a searchable archive. If you don't see an answer to your question, post it to the listserv.

If you're writing a script and having a problem with a built-in Matlab function, or you're having a problem figuring out the right syntax, you can probably find an answer on the Internet. The two best sources of answers are the resources provided by Matlab and the StackOverflow forum. I usually just type something like this into Google: **Matlab "sorting a list"**. I probably did searches like that about 300 times while writing this book!

You can also try asking an AI system such as Claude or Gemini for help, especially for help with the basics of Matlab programming. But don't rely too heavily on AI when you're first learning to program and debug; AI systems will be much more powerful once you have some solid programming and debugging skills.

Rubber Duck Debugging

If you can't find the solution to your problem online, it might be time to ask a colleague. However, you might not have an appropriate colleague, or you might not want to bother anyone. In these cases, you can try something called *rubber duck debugging*. This concept comes from the observation that when you go to ask someone for help, the process of explaining the problem to that person often leads you to realize the answer. So, you really just need to try explaining the problem. Instead of taking up someone else's time with this, you can just explain the problem to a rubber duck (Thomas & Hunt, 2019). Of course, it doesn't need to be a rubber duck. It can be a plant, your dog, a photo of Einstein, etc. You can even try writing an email to an

imaginary friend explaining the problem. The key is to describe the problem carefully in natural language. I like this approach so much that I bought rubber ducks for everyone in my lab (Figure A2.1).



Figure A2.1. Rubber ducks for debugging. They're always happy to listen to you talk about your code, and they never judge you.

This page titled 14.4: A2.3 Taking a Scientific Approach is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 13.3: A2.3 Taking a Scientific Approach by Steven J Luck is licensed CC BY 4.0.

14.5: A2.4 Deciphering Matlab's Error Messages

As noted in the previous section, Matlab's error messages are often helpful when you're developing a hypothesis about the cause of your problem. In this section, we'll do some exercises that are designed to help you learn how to decipher these messages.

Before we do the exercises, we need to divide software problems into three categories:

- A **syntax error** occurs when the script contains information that is not legal Matlab code. Often, this is a result of a typo (e.g., a missing comma). When you run a script, Matlab first checks for syntax errors, and the script won't even start running if a syntax error is detected. Instead, an error message is printed in the command window.
- A **runtime error** occurs when the code, while legal in its syntax, produces some kind of problem that causes the script to terminate with an error message in the command window. For example, if the script tries to open a file with a specific name, but no file with that name exists in the PATH, the script will terminate with an error message.
- A **logical error** occurs when the program runs to completion without an error message, but the result is incorrect. This usually means that the code in the script does not correctly implement the desired processing steps, but it sometimes means that the input violates the assumptions of the script.

Note that only syntax errors and runtime errors produce error messages. Logical errors are usually the most difficult to diagnose and solve.

Now it's time to try some exercises. These exercises require scripts and data in the Appendix_2 folder in the master folder: https://doi.org/10.18115/D5G045.

Exercise A2.1: A Syntax Error

Launch eStudio if it isn't already running, make Appendix_2 the current folder, and double-click on **Test_Case_1.m** in the Current Folder panel of the Matlab GUI. This should open the script in the Matlab editor. It's a very simple script, but it has a bug. Click the **Run** button in the editor window to run the script. You should see the following error message in the Command Window:

Error: File: Test_Case_1.m Line: 3 Column: 30
Character vector is not terminated properly.

This message tells you where in the script it detected a problem (Line 3, Column 30), and it tells you the nature of the problem ("Character vector is not terminated properly"). However, the nature of the problem is described in Programmerese, so you might not understand what the error message means.

However, at least it tells you what line was running when the error message was generated (Line 3). Take a look at Line 3 of the **Test_Case_1.m** script. This line is trying to open a dataset named **1_N170.set**. Notice that the text **'1_N170.set);** is underlined in red. This is Matlab's way of telling you that it thinks this part of the code is a syntax error. If you hover your mouse over this part of the code, some more Programmerese pops up, saying "A quoted character vector is unterminated." Even if you don't fully understand this message, you should be able to infer that there is some problem with the termination of the filename, **1_N170.set**.

This filename is specified as a text string, and Matlab text strings need to be enclosed in single quote characters (e.g., '**1_N170.set**'). Otherwise Matlab doesn't know where the string starts and stops. In this script, we are missing the single quote that terminates the string. That's what Matlab means when it tells you "Character vector is not terminated properly". This is a very common error that I made at least a dozen times when preparing the scripts in this book.

Try adding a single quote mark after **.set** in the script. You'll see that the red underline disappears from under this string. Now try running the script. Voila! No more error message. [However, this may have messed up the EEGsets in eStudio, so you should probably quit and restart eStudio before you try to do any real work.]

If you can't figure out an error message in this way, try Googling it (e.g., with the search phrase **matlab "character Vector Is Not Terminated Properly"**).

Exercise A2.2: A More Subtle Syntax Error

Close **Test_Case_1.m** and double-click on **Test_Case_2.m** to load it in the Matlab editor. Click the **Run** button in the editor window to run the script. You should see the following error message in the Command Window:

File: Test_Case_2.m Line: 4 Column: 26
Invalid expression. When calling a function or indexing a variable, use parentheses.
Otherwise, check for mismatched delimiters.

Take a look at Line 4 of the script. The error message says that this is an "Invalid expression" — not very informative.

Do you see anything marked as problematic by Matlab in the editor window? If you look very closely, you'll see that a right square bracket near the middle of the line is underlined in red, as is a right parenthesis near the end of the line. Matlab isn't sure which one of these is the source of the problem, so you now have two hypotheses to examine.

When there are problems with square brackets or parentheses (which Matlab calls "delimiters"), this usually means that they're not paired properly. That is, every left bracket needs a corresponding right bracket. To fix a problem like this, you need to make sure that each delimiter has a pair and that both delimiters are in the correct places. If you look closely at Line 4, you'll see that there is a left parenthesis that is correctly paired with the underlined right parenthesis.

But if you look at the square brackets, you'll see that there are two right brackets and only one left bracket. This means you need to figure out if we are missing a left bracket or if we have a right bracket that shouldn't be there. In this case, the problem is an extra right bracket. Try deleting the underlined right bracket. You should see that the red underlines disappear, including the one under the right parenthesis. Now try running the script. It should now run properly.

Exercise A2.3: A Simple Runtime Error

Close **Test_Case_2.m**, double-click on **Test_Case_3.m,** and click the **Run** button to run the script. You should see the following error message in the Command Window:

Unrecognized function or variable 'EGG'.
Error in Test_Case_3 (line 4)
[ALLEEG, EEG, CURRENTSET] = eeg_store(ALLEEG, EGG, 0);

This error message is actually reasonably easy to understand: There is a variable named **EGG** on Line 4 that isn't recognized. There are usually two possible explanations for this kind of problem. The first is that the variable is defined in the wrong part of the program, so it hasn't yet been defined when Matlab tries to execute Line 4. You can just search for the variable to see if it's defined later. The second common explanation is a simple typo. In this case, it's pretty clear that the variable is supposed to be named **EEG**, not **EGG**. If you change the name to EEG and run the script, you'll find that the problem has been eliminated.

Note that Matlab doesn't underline any of the code in the script when it detects this error. That's because it's a "runtime error" (an error detected while the script is running) rather than a syntax error. In other words, Line 4 is perfectly legitimate Matlab code when considered in isolation. It's only a problem in the context of the rest of the code (i.e., because no variable named **EGG** has been defined on lines that execute before Line 4), and Matlab doesn't realize there is a problem until it tries to run the script.

Exercise A2.4: A Slightly More Complicated Runtime Error

Close **Test_Case3.m**, double-click on **Test_Case_4.m,** and run the script. You should see something like the following error message in the Command Window:

Error using load
Unable to find file or directory
'/Users/luck/Dropbox/Research/Manuscripts/ERP_Analysis_Book/Appendix_2_Troubleshooting/Exercises1_N170.set'.

Error in pop_loadset (line 155)
TMPVAR = load('-mat', filename);

```
Error in Test_Case_4 (line 7)
EEG = pop_loadset('filename', Dataset_filename );
```

This error message is a little more complicated. The script calls an EEGLAB function named **pop_loadset**, and Matlab actually detected the error while **pop_loadset** was executing. The error message therefore tells you what line of **pop_loadset** was running when the error was detected, along with the line of **Test_Case_4.m** that called **pop_loadset**.

Note that Matlab has detected an error on line 155 in an EEGLAB script named **pop_loadset** and an error on line 7 of our test script (**Test_Case_4**). It's pretty unlikely that there is a bug in the pop_loadset script, so you should focus on line 7 of **Test_Case_4.m**.

When you get an error message like this, it probably means that you have sent some kind of invalid data to the **pop_loadset** function. However, if you can't figure out the problem by looking at your script, you might want to open the script for the function that your script is calling. You can do that by selecting the name of the function within your script, right-clicking on it, and selecting **Open "pop_loadset"**.

I'm not actually going to reveal the problem with **Test_Case_4.m** here. That will be revealed in the next section.

This page titled 14.5: A2.4 Deciphering Matlab's Error Messages is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 13.4: A2.4 Deciphering Matlab's Error Messages by Steven J Luck is licensed CC BY 4.0.

14.6: A2.5 Debugging Scripts by Performing Experiments and Collecting Data

Earlier, I said that you should act like a scientist when debugging code, which includes performing experiments and collecting data. In this section, I'll provide some simple examples of this.

Scripting is all about storing information in variables and applying operations to those variables (e.g., addition, subtraction). When there is a runtime error or logical error, some variable usually has an incorrect value. A key aspect of debugging is therefore assessing the values of the variables at different points in the execution of the script. This is much like monitoring the representations of the brain at different time points following the onset of a stimulus while a participant performs a task.

There are a couple ways to monitor variables. The simplest is to print their values to the command window at key points during the execution of the script. To see how this works, let's use this approach to understand what is wrong with the **Test_Case_4.m** script.

Exercise A2.5: Displaying Variable Values

Open the script (if it's not already open) and take a look at it. The goal of the script is to load a dataset named **1_N170.set** that is located in the current folder (the same folder as the script). To achieve this, we start by getting the path to the current folder with the **pwd** (*print working directory*) function and store the result in a variable named **DIR**. We then concatenate this value with **'1_N170.set'** to create the full path to the file, and we store the result in a variable named **Dataset_filename**. We then send **Dataset_filename** to the **pop_loadset** function. But we're getting an error message indicating that Matlab can't find the file that we've specified in this manner.

To figure out the source of the error, let's examine the values of the **DIR** and **Dataset_filename** variables. To accomplish this, add the following two lines to the script, right after line 5 (i.e., after the lines of code that set the values of these variables):

```
display(DIR);
display(Dataset_filename);
```

These commands will display the values of the **DIR** and **Dataset_filename** variables to the Command Window. To see this, run the script. You should see the same error message in the Command Window as before, but prior to that you should see something like this:

```
DIR = '/Users/luck/Dropbox/Research/Manuscripts/ERP_Analysis_Book/Appendix_2_Troubleshooting/Exercises'
Dataset_filename = '/Users/luck/Dropbox/Research/Manuscripts/ERP_Analysis_Book/Appendix_2_Troubleshooting/
Exercises1_N170.set'
```

The value of **DIR** will be different on your computer because your current folder is not the same as mine. However, if you look closely at it, it should be the correct folder for your computer.

If you look at the value of **Dataset_filename**, you should see the error: there is no slash (or backslash) between **Exercises** and **1_N170.set**. In other words, the script is telling Matlab to look for a file named Exercises1_N170.set instead of a file named **1_N170.set** inside the **Exercises** folder (i.e., **Exercises/1_N170.set**).

To fix this problem, change Line 5 from this:

```
Dataset_filename = [DIR '1_N170.set'];
```

to this:

```
Dataset_filename = [DIR filesep '1_N170.set'];
```

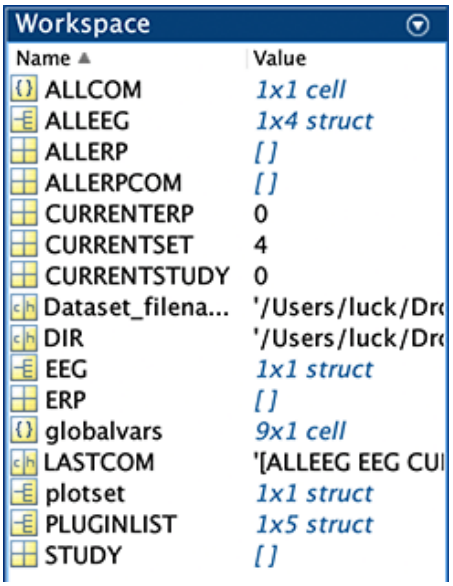
filesep is a Matlab function that returns the file separator character for the current operating system (slash for Mac or Linux, backslash for Windows). Now run the code. You will see that the **Dataset_filename** variable now has a slash (or backslash)

between the folder name and the file name, and everything now works correctly. [But keep in mind that eStudio does not know that you've loaded a new EEGset; you'll need to click the **Refresh** button in the **EEGsets** panel to see the new EEGset in eStudio.]

By the way, I recommend learning how to use the **fprintf** command to display the values of variables. It's much more powerful than the **display** command. It's explained briefly in Chapter 10, and you can Google it to find more details. Very worth knowing!

Exercise A2.6: The Workspace Pane and the Variables Pane

Once a script has stopped running, you can see the valuable of a variable by simply typing the name of the variable in the Matlab command window. Try this by typing **Dataset_filename** in the command window. You can also see the value of a variable by looking at the *Workspace* pane in the Matlab GUI (which is probably on the right side of the GUI). This pane should look something like Screenshot A2.1.

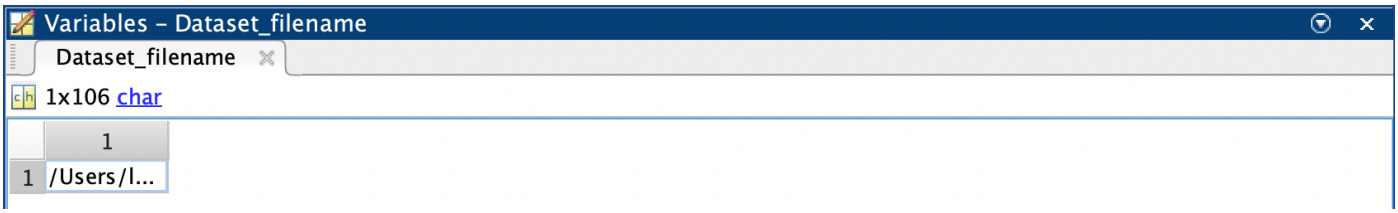


Screenshot A2.1

This pane shows all the variables that Matlab knows about. For example, I currently have four datasets loaded, and the fourth dataset is currently active, so **CURRENTSET** has a value of 4. For more complicated variables, the dimensions of the variable are shown instead of the value. For example, **ALLEEG** is a data structure that holds all of the datasets, and it is listed as a **1x4 struct** to indicate that it is a structure that is 1 row high by 4 columns wide (because I have 4 datasets loaded).

You can see that the Workspace pane in Screenshot A2.1 also contains the **DIR** and **Dataset_filename** variables that were created by the last script I ran, **Test_Case_4.m**. If you don't see these variables on your computer, run **Test_Case_4.m** again (after fixing the bug). The strings stored in these variables are too long to be seen well in the Workspace pane, but Matlab has a *Variables* pane that you can use to inspect variables more carefully.

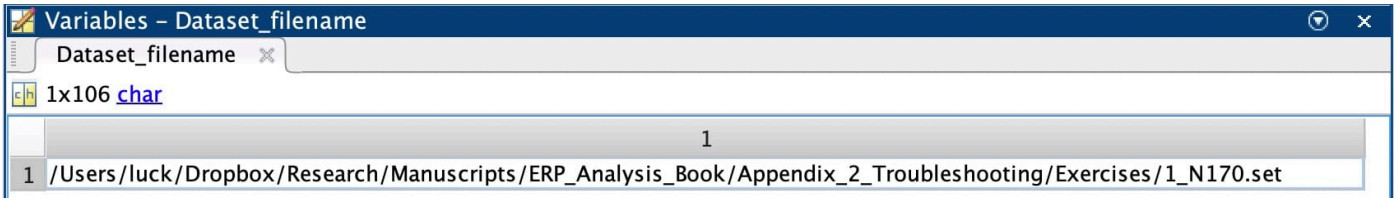
To see this, double-click on the **Dataset_filename** variable in the Workspace pane. You should see the Variables pane open up in the Matlab GUI (probably above the command window). It should look something like Screenshot A2.2.



Screenshot A2.2

The value of the **Dataset_filename** variable is now shown (in the little box with "1" to the left and "1" to the right). However, the

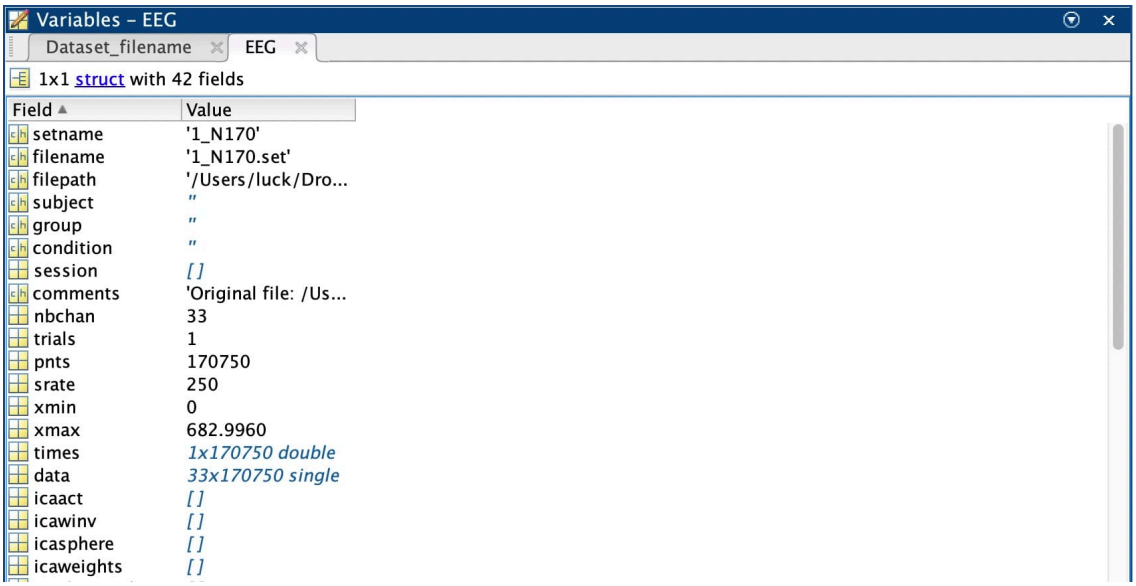
box is too small to see much of the variable. If you place your mouse over the right edge of the box with the "1" above the variable value and drag rightward, the box will expand, and you'll be able to see the whole variable (as in Screenshot A2.3).



Screenshot A2.3

If you've fixed the bug in **Test_Case_4.m**, you should be able to see the slash (or backslash) between **Exercises** and **1_N170.set**. This might seem like a lot of work to see the contents of a variable, but once you're practiced at this approach, it will become very efficient. Also, it's a great way to look at more complex variables.

For example, double-click on the **EEG** variable in the Workspace. The Variables pane should now look something like Screenshot A2.4. You can easily see the various fields of this complex data structure, such as the number of channels (the **nbchan** field) and the number of time points (the **pnts** field). The actual voltage values are stored in the **data** field, and you can see that this field is 33 rows high (one row for each of the 33 channels) and 170750 columns wide (one column for each of the 170750 time points). Double-click on the **data** field, and now you can see the actual voltage values for each combination of channel and time point. Try double-clicking on other fields as well to get a sense of what information is held in the **EEG** variable.



Screenshot A2.4

Exercise A2.7: Interrupting a Script to View Variable Values

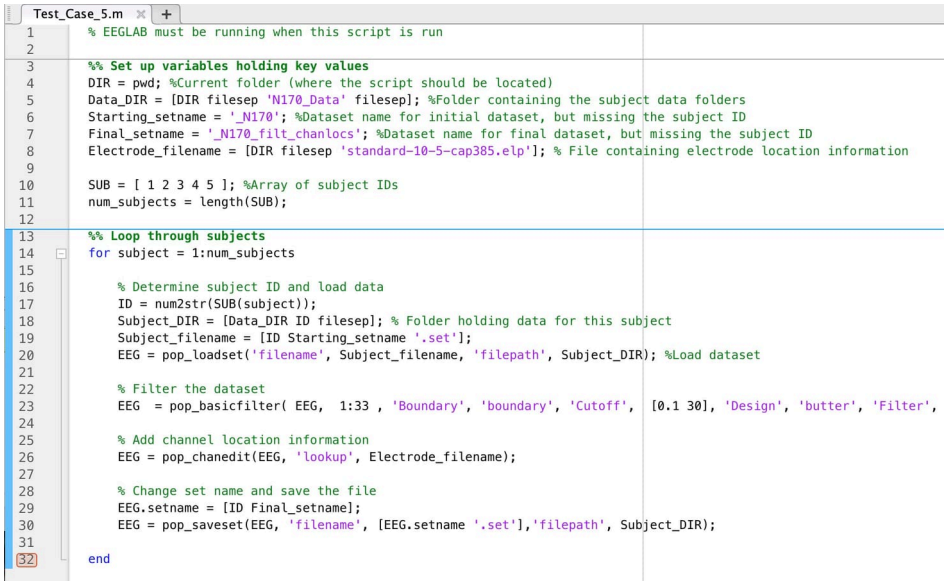
The Workspace and Variables panes are great ways to see the values of variables, but you can ordinarily use this approach only after a script is finished. But Matlab also contains a *debugger* that allows you to pause the operation of a script at key times so that you can examine the values of the variables at those times.

To see this in action, load the **Test_Case_5.m** script. This script is much more complicated than the previous test cases, but it is actually a simplified version of the **Step1_pre_ICA_processing.m** script from Chapter 10. It loops through the data from the first five subjects in the N170 experiment, loading each subject's dataset, filtering it, adding channel location information, renaming it, and then saving the new dataset to the disk.

There isn't a bug in this script, but let's imagine that there was a problem and that you suspected that the problem arose in the loop that starts on Line 14 and ends on Line 32. If you just start running the script, you won't be able to inspect the values of

the variables using the Workspace and Variable panes until the script ends. However, you can set a *breakpoint* that causes the program to pause at the end of each time the program runs through the loop.

To set a breakpoint at a given line, you simply click the corresponding line number in the editor window for the script (but make sure that the script has been saved first if you've made any changes). For example, Screenshot A2.5 shows what happened when I clicked on the **32** corresponding to the line number for the **end** statement at the end of the loop (Line 32). This line number is now highlighted in a red box.



Screenshot A2.5

Go ahead and click on the line number for Line 32 in the **Test_Case_5.m** script. It should now have a red box around it, as in Screenshot A2.5. Now run the script. You should see that the script runs, printing a bunch of information in the command window. However, it has only run through the loop once and has paused. You can tell that it has paused because the command window prompt is now **K>>** instead of just **>>**. You can also see a green arrow next to the **end** statement in the editor window for the script. And if you look at the Workspace, you can see that the **subject** variable (which loops from 1 to 5 in this script) is set to **1**.

Now let's look at the current value of the **EEG** data structure. You may already be showing **EEG** in the Variables pane, in which case you are looking at the current value of this variable. If it's not already showing, you can double-click **EEG** in the Workspace pane to look at it in the Variables pane. You can see that the set name has been correctly updated to **1_N170_filt_chanlocs** (although seeing the full name of this field may require double-clicking on the **setname** field in the **Variables** pane.

You can resume execution of the script by clicking the Continue button in the toolbar along the top of the editor window for **Test_Case_5.m**. When you do that, the script resumes and runs through the loop one more time. If you now look at the value of the **setname** field of **EEG** in the Variables pane, you'll see that it has been updated to **2_N170_filt_chanlocs**. You can click the **Stop** button in the editor window's toolbar to quit from the script.

This feature of Matlab is very useful for debugging. You can read more about it in the Matlab documentation for breakpoints. Other related tools are described in Matlab's general debugging documentation.

This page titled 14.6: A2.5 Debugging Scripts by Performing Experiments and Collecting Data is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 13.5: A2.5 Debugging Scripts by Performing Experiments and Collecting Data by Steven J Luck is licensed CC BY 4.0.

14.7: A2.6 Avoiding Bugs in Your Scripts with Good Programming Practices

The first step in debugging a script is to get in a time machine, go back to the moment when you started writing the script, and tell yourself that you're now wasting a huge amount of time trying to debug the script. "Please," you should tell your earlier self, "follow good programming practices while writing this script so that I won't need to waste so much time. I'm even busier now than I was when I first wrote the script."

If you don't have a time machine, you should resolve to follow good programming practices from now on. It will take a little more time now, but you will be giving a great gift to your future self. As Ben Franklin famously said, "An ounce of prevention is worth a pound of cure" (which could be updated to "A gram of prevention is worth a kilogram of cure").

This section contains a set of good programming practices that I find to be particularly relevant for scientists who are analyzing EEG/ERP data in Matlab and are not already expert programmers.

Rapid cycling between coding and testing

Perhaps the most common mistake that novice coders make is trying to write an entire script without doing any testing along the way. If you write a 30-line script, you will probably have 8 different errors in the script, and it will be really hard to figure out what's going wrong.

The best approach is to write a small amount of code, test it, debug it if necessary, and then add more code. When you're new to programming, this might be only 1-3 lines of code at a time. As you gain experience, you can write more lines before testing, but even an experienced programmer usually does some testing after every 20-40 new lines of code.

Define all values as variables at the top of the script

If you've already read the chapter on scripting, you'll know that "defining all values as variables at the top of the script" is my #1 principle of writing good code. For example, in the N170 experiment that is the focus of the scripting chapter, we analyzed the data from subjects 1-10, but we left out subject 5. When we looped through the subjects, we needed a line of code like this:

for subject = [1 2 3 4 6 7 8 9 10] # Note that 5 is missing from this list

Imagine that we need to loop through the subjects in three different parts of the script (e.g., once for pre-ICA EEG processing, once for post-ICA EEG processing, and once for ERP processing). We could just repeat that same loop in each of these three different parts of the script. But now imagine that, a year after we've analyzed the data, we get reviews back from a journal and a reviewer wants us to reanalyze the data without excluding subject 5. Now we need to find all the parts of the script with this loop and modify them. Will we remember that we had three loops and add subject 5 back to all of them? There's a good chance that we will have forgotten and won't find all three of them. As a result, we will have a bug. And we will either end up with the wrong result or waste hours of time trying to find the problem.

To avoid this problem, you should always, **always**, **ALWAYS** use a variable at the top of the script to define a list like this. **ALWAYS!!!**

Here's an example:

% This line is in the top section of the script
SUB = [1 2 3 4 6 7 8 9 10]; % Array of subject IDs, excluding subject 5

The same principle applies to individual numbers (e.g., the number of subjects) and strings (e.g., a filename).

Even if you understand and appreciate this advice, it's easy to ignore it by saying to yourself, "This script is just a few lines. I don't need to worry about putting the values into variables at the top." Most long scripts start as short scripts, so you are being shortsighted if you don't do the right thing for short scripts. So, at the risk of repeating myself, you should always, **always**, **ALWAYS**

use a variable at the top of the script to define values. **ALWAYS!!!**

Note that zeros and ones can be an exception to this rule when they are being used more conceptually. For example, zero and one are sometimes used to mean TRUE and FALSE. Or you might do something like this:

```
% This line is in the top section of the script
SUB = [ 1 2 3 4 6 7 8 9 10 ]; % Array of subject IDs, excluding subject 5
num_subjects = length(SUB); % Number of subjects

% This is how we use the list later in the script
for subject_num = 1:num_subjects
    subject = SUB(subject_num);
    % More code here to process the data from this subject
end
```

Make your code readable

The reality of science is that you will often start a script, come back to it a few weeks later to finish it, and then modify it 18 months later (after you get the reviews for a manuscript). And someone else may get a copy of your script and modify it for their own studies. If the code isn't easily readable, bugs are likely to be introduced at these times. Here are a few simple things you can do to make your code more readable:

- Include lots of internal documentation in your scripts. It's a great gift to your future self and to the scientists who want to follow up on your research
- Define all values as variables at the top, as noted before, but also make sure there is a comment indicating the purpose of each variable
- Divide your code into small, modular sections (or separate functions), with a comment at the beginning of each section or function that explains what that section or function does
- Use intrinsically meaningful variable names (e.g., **num_subjects** instead of **ns**) and function names (e.g., **ploterps** instead of **npbd**). [I wasted a couple hours one night in my first year of graduate school because someone had used the name **npbd** for a function that plotted ERP waveforms, and I'm still bitter...]

You can find more discussion of the importance of readability in the scripting chapter.

Make your code modular

If you have a single script that is more than 200 lines long, it should probably be broken into a sequence of multiple scripts. It's a lot harder to find problems in a long script than in a short script. And it's a lot easier to introduce problems into a long script (e.g., by adding code to the wrong section). For example, the EEG/ERP processing pipeline in Chapter 10 consists of a series of 7 scripts. In the ERP CORE experiments, we had about 20 different scripts for processing each individual experiment.

Make your code portable by using relative paths

Almost all EEG/ERP processing scripts need to access files via a path. The worst way to handle this is something like this (for loading a dataset):

```
EEG = pop_loadset('filename', '/Users/luck/ERP_Analysis_Book/Appendix_2_Troubleshooting/Exercises/1_N170.set');
```

This violates the principle of defining all values at the top of the script. A better, but still problematic, approach is this:

```
% Variables defined at the top of the script
Data_DIR = '/Users/luck/ERP_Analysis_Book/Appendix_2_Troubleshooting/Exercises/';
setname = '1_N170.set';

% Loading the data later in the script
EEG = pop_loadset('filename', setname, 'filepath', Data_DIR);
```

The problem with this approach is that it will break if you switch to a different computer, move your data to a different location, or share your script with someone else.

A better approach is to determine the path from the location of the script (assuming that the script is kept with the data):

Data_DIR = pwd; %Current folder (where the script should be located)

Chapter 10 describes this in much more detail.

Code review

It is becoming very common (and sometimes required) for researchers to post their data analysis code online along with their data when publishing a paper. That way, other researchers can verify that they get the same results with your data and can use your code in their own studies. I think this is a wonderful trend.

When you realize that other people will be looking at and running your code, this tends to increase the pressure to make sure that the code actually works correctly. In theory, you should already be highly motivated to make sure that your code works, because your findings depend on code that works correctly. But public scrutiny is often an even stronger motivator.

A really good way to make sure that your code works correctly is to use *code review*. This is just a fancy term for having someone else go through your code to make sure it's correct. Of course, code review is a lot easier and more effective if you've made your code readable and portable. The person who reviews your code will also likely have suggestions for making your code even more readable and will provide a good test of whether your code is portable (i.e., whether it works on the reviewer's computer).

You can also use an AI system such as Claude or Gemini to review your code. It won't understand the logic behind your code, but it might still find problems or make useful suggestions for improvement. Just make sure that the suggestions actually work. And as I said before, don't rely on AI very much until you've gained considerable expertise.

This page titled 14.7: A2.6 Avoiding Bugs in Your Scripts with Good Programming Practices is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 13.6: A2.6 Avoiding Bugs in Your Scripts with Good Programming Practices by Steven J Luck is licensed CC BY 4.0.

14.8: A2.7 References

Feyerabend, P. (1993). *Against Method* (3rd ed.). Verso.

Thomas, D., & Hunt, A. (2019). *The Pragmatic Programmer: Your journey to mastery, 20th Anniversary Edition*. Addison-Wesley Professional.

This page titled 14.8: A2.7 References is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 13.7: A2.7 References by Steven J Luck is licensed CC BY 4.0.

15: Appendix 3- Example Processing Pipeline

This appendix provides an example of a typical EEG preprocessing pipeline for a single participant, including all the steps prior to averaging. It is nearly identical to the pipeline scripts provided in the second half of the chapter on Scripting (Chapter 10). Much of the logic for the ordering of the steps is spelled out in Luck (2014, especially Appendix 1).

This pipeline is optimized for the kinds of experiments typically conducted in my lab, such as the ERP CORE experiments, in which highly cooperative college student participants are tested with relatively simple cognitive tasks. Changes will often be necessary for different types of tasks and/or participants. In other words, don't blindly follow this pipeline!

Step 1: Preprocessing Prior to ICA-Based Artifact Correction

- Import the participant's original EEG data file(s) into EEGLAB
  - If there are multiple data files (e.g., one per trial block), combine them into a single dataset.
    - If the cap was removed and replaced between data files (e.g., because you had recording sessions on multiple days), you should merge the data files after the ICA step (because the electrodes might not be in exactly the same places for the different files, which will mess up ICA but isn't a big deal for most other kinds of analyses).
    - If a given channel is bad in a subset of the blocks, you could interpolate that channel for those blocks at this stage. If you do this, you should apply the filtering step described below at this stage, which will make the interpolation work better. (And then you will skip the filtering at the later stage.)
  - If your data were recorded without a reference (usually this is only for the BioSemi ActiveTwo system), you may receive a warning that strongly encourages you to reference the data now. Do not do it. You will reference the data at a later step.
  - Recommended: Save this dataset as a file.
- If appropriate, shift the stimulus event codes in time to account for any consistent mismatch between the time of the event code and the actual stimulus onset time.
  - All LCD displays interpose a constant delay between the time that the video information arrives from the computer's video card and the time that the image is presented on the screen. This delay cannot be detected by the computer and cannot be determined reliably from the manufacturer's specifications. It must be measured with some kind of photosensor. This delay is often greater than 25 ms, so it must be accounted for.
  - Almost all video displays (LCD and CRT) have a delay that depends on the vertical position of the stimulus on the display. This delay is typically ~15 ms for the bottom of the display relative to the top (assuming a refresh rate of 60 Hz). If you are using stimuli at very different vertical locations in your different experimental conditions, you should measure the delay separately for the different locations and adjust accordingly. For large stimuli that are vertically centered, we measure the delay in the vertical center of the display.
  - Delays may also occur for stimuli in other modalities. The delay should be measured and, if substantial, accounted for by shifting the event codes.
  - Make sure that you shift only the stimulus event codes and not the response event codes (unless there is also a delay for the responses). Use a positive shift value because the event code needs to be shifted later in time (because the stimulus occurred after the event code).
  - ERPLAB's function for shifting event codes creates a CSV file showing how each event code was shifted. You should look at this file to verify that the shifting was correct.
  - It's also important to verify that the delay is consistent (±1 sample period). If it is not consistent, there is probably a problem with the stimulus presentation script.
- If desired, downsample your data to a lower sampling rate
  - My lab typically records at 500 Hz and then downsamples to 250 Hz. This gives us a higher sampling rate if we ever need it, but downsampling to 250 Hz makes the subsequent data processing faster and the data files half as large.
  - Note that EEGLAB's pop_resample routine will automatically apply an appropriate antialiasing filter prior to the resampling process.
- Apply a bandpass filter to the EEG and EOG channels
  - We ordinarily use a non-causal Butterworth impulse response function, 0.1–30 Hz half-amplitude cut-off, 12 dB/oct roll-off.
    - Some researchers prefer a lower cutoff for the high-pass filter (e.g., .05 Hz or .01 Hz), especially when looking at slow components like the late positive potential. However, we find that 0.1 Hz works best in most cases. Do not use anything higher than 0.1 Hz unless you really know what you're doing!
    - You should apply the option for removing the DC offset in the data prior to filtering. This can reduce the edge

artifacts that occur at the beginning and end of each recording period.
    - If extensive 60 Hz noise is present, you can apply the cleanline plugin (Mitra & Pesaran, 1999; see Bigdely-Shamlo et al., 2015 for important details about implementing this tool). If cleanline doesn't work well for you, you can try the newer Zapline method (de Cheveigné, 2020; Klug & Kloosterman, 2022). Alternatively, you could decrease the low-pass frequency to 20 Hz and/or increase the slope of the low-pass filter to 48 dB/octave (which requires doing the low-pass and high-pass filtering as separate steps).
    - If you have also recorded other signals (e.g., a photosensor), you may not want to filter those channels (or you might want different filtering parameters for those channels).
- Add channel location information specifying the 3-D locations of the electrodes.
- Perform channel operations to create bipolar channels (and to reference the data if they were not referenced during the recording).
    - Create a bipolar HEOG channel (HEOG-left minus HEOG-right or vice versa)
    - Create a bipolar VEOG channel (VEOG-lower minus the closest electrode that is above the eyes). For example, if the VEOG-lower electrode is under the left eye, the FP1 signal would ordinarily be subtracted from the VEOG-lower signal.
    - Keep the original, monopolar EOG channels, which will be important for ICA.
    - Make sure that the channel location information is preserved by the Channel Operations routine.
    - If your data were not recorded with a reference, reference the data now.
        - In some systems (e.g., BioSemi ActiveTwo), the data saved to disk are not referenced, and you should perform referencing now. However, if your data were referenced during recording, do not re-reference now. The re-referencing will be performed later, after artifact correction.
        - In most labs that use the Brain Products ActiCHamp system, the data are already referenced at this point. However, our lab has a custom version of the data acquisition system in which the data are not referenced. Our lab therefore references at this point (and we re-reference later, after artifact correction).
        - If you are referencing now, use a single electrode site as the reference. If a combination of sites is desired in the long run (e.g., the average of all sites, the average of the mastoids), you will re-reference to that combination later.
        - It doesn't really matter what site is used as the reference at this point. My lab uses P9.
    - Recommended: Save the resulting dataset as a file. We call this dataset the **pre-ICA dataset**.
- View the EEG to make sure everything looks okay and to identify channels that require interpolation. These channels will be left out of the ICA decomposition process. The actual interpolation will be done after the ICA correction has been performed.
    - To identify channels that should be interpolated:
        - Visually inspect the EEG on both short (e.g., 5-second) and long (e.g., 60-second) time scales. This video demonstrates how to perform an initial visual inspection of continuous EEG data: https://doi.org/10.18115/D5V638.
        - If the voltage drifts around or there are many sudden changes in voltage, that channel should be interpolated.
        - If a channel is fine for most of the session but shows occasional periods of large deflections, this channel can be interpolated after epoching but limited to epochs in which the large deflections occur (as determined with the artifact detection procedures).
        - If a channel shows a lot of high-frequency noise, it may not actually need to be interpolated.
            - If the main dependent variable will be the mean voltage over some time range (e.g., 200-300 ms for N2pc or 300-500 ms for N400), high-frequency noise is not usually a problem. The best way to tell if it will be a problem is to look at the standardized measurement error (SME). If the SME for that channel is >2 SD beyond the mean SME of the other channels (excluding EOG, Fp1, and Fp2), then the channel should be interpolated. If the channel is the main channel for data analysis, you could interpolate if it is >1.5 SD beyond the mean.
            - If there is a lot of high-frequency noise, the signal may also be corrupted in this channel. To see this, make a note to check the averaged ERP for that channel and surrounding channels after the averaging process. If the channel looks quite different from the surrounding channels, the signal is probably corrupted, and the channel should be interpolated (which will require going back to this stage and repeating all the subsequent processes.
            - If the main dependent variable will be a latency value and/or a peak-related variable, high-frequency noise is more likely to be a problem. You can confirm this by computing the bootstrapped SME when you get to the averaging step.
    - Save information about channels that should be interpolated in an Excel file. That makes it possible to repeat the processing with a script.
    - If a given channel misbehaves only occasionally, save this information in the Excel file that controls artifact detection.

Step 2: ICA-Based Artifact Correction

- Make a copy of the **pre-ICA dataset**. We will use this new dataset for the ICA decomposition. This new dataset is called the **ICA decomposition dataset**.

- Apply the following operations to the **ICA decomposition dataset**.
  - Bandpass filter with half-amplitude cutoffs at 1 and 30 Hz, 48 dB/octave
    - It's OK that we've double-filtered the data. The original filtering is so much milder that it will be dwarfed by the new filter.
    - If your previous low-pass cutoff was at 20 Hz, use 20 Hz here as well
    - If your previous low-pass slope was 48 dB/octave, just do a high-pass filter at this stage (1 Hz, 48 dB/octave).
    - Note: It is important that this filtering is done before the following steps. Some of the following steps introduce discontinuities in the EEG data, and those can lead to edge artifacts if the filtering is applied later.
  - Resample the data at 100 Hz.
    - This is not strictly necessary, but it makes the ICA decomposition faster.
    - Don't resample if the recording was brief. You need enough data to train the ICA decomposition routine. See the chapter on artifact correction for details on how much data is necessary.
  - Delete break periods, because the EEG is often "crazy" during breaks, which degrades the ICA decomposition.
    - A break is defined as a period of at least X ms without an event code, where X is some reasonable value (e.g., 2000 ms, assuming that you never have 2000 ms between event codes except during breaks).
    - You can tell it to ignore certain event codes. You should always exclude boundary events. You may also want to exclude response events (in case the subject makes button presses during the breaks).
    - You need a buffer at the edges of the break periods so that you do not cut into your eventual epochs (e.g., 1500 ms at the beginning of the break and 500 ms at the end of the break).
  - Delete periods of "crazy" EEG (wild deflections, beyond what you see with ordinary artifacts).
    - This can ordinarily be done using ERPLAB's **Artifact rejection (continuous data)** routine.
    - When setting the parameters, make sure that you are mainly rejecting segments of data with "crazy" EEG and are not rejecting segments with blinks or other ordinary artifacts.
    - You should ordinarily exclude the EOG channels. If you include these channels, you may need a rejection threshold that is too high for the EEG channels. For subjects with unusually large blinks, you may also need to exclude Fp1, Fp2, or any other channels that are very close to the eyes.
    - For most subjects, a threshold of 500 μV and a window size of 1000 ms works well. However, you should visually inspect the results and adjust these parameters if necessary. If blinks exceed the threshold, you will need to increase the threshold (or perhaps exclude Fp1, Fp2, or other channels with huge blink activity). The final parameters should be stored in an Excel spreadsheet for use in scripting.
    - Check the box labeled **Join artifactual segments separated by less than** and put 1000 in the corresponding text box.
    - Note that this deletion of segments with "crazy" EEG is designed only to improve the ICA decomposition. Once the ICA weights are transferred back to the pre-ICA dataset to create the post-ICA dataset (see below), these segments will be present in the data. You will later epoch the post-ICA data and apply ordinary artifact detection to mark and eventually exclude any epochs that contain wild voltage deflections. This way, you will have an accurate count of the number of trials excluded because of artifacts.
  - Recommended: Save the final version of the **ICA decomposition dataset** as a file.
- Perform the ICA decomposition process on the **ICA decomposition dataset**.
  - Make sure that '**extended', 1** is set (it should be set by default). This allows ICA to detect sub-Gaussian components, such as line noise and slow drifts.
  - Exclude the bipolar EOG channels (because we will use these later to see the blinks and eye movements).
  - Exclude "bad channels" that will be interpolated.
  - If the ICA crashes or fails to converge, check the dataset to see if you have a lot of very short segments between boundary events. If so, you may need to alter your procedures for deleting periods of "crazy" EEG (e.g., by increasing the value for **Join artifactual segments separated by less than**).
  - Recommended: Save the result as a new file.
- Examine the components (especially the scalp maps) to make sure that the decomposition worked correctly. You should have 1-2 ICs corresponding to blinks (and possibly vertical eye movements) and 1-2 ICs corresponding to horizontal eye movements.
  - My lab doesn't ordinarily remove components corresponding to other artifacts, but this could be done when necessary for EKG artifacts and perhaps others.
- Determine which components correspond to artifacts that should be removed. This is done by both examining the scalp maps and comparing the time course of the components with the time course of the EOG signals. This information should be stored in a spreadsheet so that it can be used in scripts.
- Transfer the ICA weights from the **ICA decomposition dataset** to the **pre-ICA dataset**. The result is called the **post-ICA dataset.**
- Remove the independent components corresponding to the artifacts.

    - Recommended: Save the result as a file.
- Visually inspect the corrected data to make sure that the correction worked properly.

Step 3: Post-ICA EEG Processing (starting with the post-ICA dataset)

- Re-reference the data
    - My lab's current preference is the average of P9 and P10, with the average of the left and right mastoids as a second-best alternative. It is usually best to use whatever is most common in your area of ERP research. For example, we use the average of all sites as the reference when we look at the N170 elicited by faces, because that is the most common reference in that research area. This makes it easier to compare ERPs across studies (because a different reference can make the waveforms and scalp distributions look radically different).
    - Also create bipolar HEOG and VEOG from the ICA-corrected EOG channels and look at them to assess the effectiveness of the artifact correction.
    - Keep the uncorrected bipolar HEOG and VEOG signals, which we will use for detecting blinks and eye movements that occurred near the time of stimulus onset and may have interfered with perception of the stimulus.
- Perform the interpolation for the "bad channels" that were identified earlier.
    - Non-EEG channels (e.g., EOG channels, a photosensor channel, a skin conductance channel) should be ignored in the process of computing interpolated values.
    - Note that interpolation should ordinarily be performed after high-pass filtering.
    - If you are using the average of all sites as the reference, you should either perform the interpolation before referencing or exclude any to-be-interpolated sites from the reference.
- Add an EventList to the dataset
- Run BINLISTER to assign events to bins
- Extract bin-based epochs
    - Important: Use ERPLAB's tool (Extract bin-based epochs), not EEGLAB's tool (Extract epochs). The subsequent ERPLAB operations will not work correctly if you use EEGLAB's tool.
    - Our standard epoch is from -200 to +800 ms. A longer prestimulus is used if we will be doing frequency-based analyses or if we want to maximize trial-to-trial stability of the signal (e.g., for decoding, which is extraordinarily sensitive to trial-to-trial variability).
    - Baseline correction is ordinarily applied at this step
    - Recommended: Save the epoched dataset as a file

Step 4: Artifact Detection

- General procedure for a given type of artifact
    - Start with default detection parameters
    - Apply to data
    - Check number of trials detected
    - Scroll through data to determine whether the parameters were effective
    - If the parameters were not effective, update them and try again until effective parameters are found
    - The parameters should be stored in a spreadsheet for future scripting.
- Standard artifacts to detect
    - Blinks and eye movements that might interfere with the perception of the stimulus
        - Test for blinks in the uncorrected bipolar VEOG channel and test for eye movements in the uncorrected HEOG channel
        - Use the step function with a window size of 200 ms and a step size of 10 ms
        - The test period should start at -200 ms (because vision is functionally suppressed for ~100 ms after the blink is complete, so we want to detect blinks even if they ended shortly before stimulus onset)
        - The test period should be at least 200 ms and should terminate at or after the offset of the stimulus
        - Start with a threshold of 50 μV for blinks and 32 μV for eye movements (which corresponds to a 2° rotation of the eyes)
        - You can often look at the averaged EOG signals to see if you've successfully removed these artifacts.
    - General C.R.A.P. that increases measurement error
        - Apply to all channels except for the uncorrected HEOG and VEOG channels
        - Use both the absolute voltage threshold and moving window peak-to-peak algorithms; they occasionally catch different trials
        - The test period should typically encompass the whole epoch.

        - 150 μV is a good starting threshold.
        - The goal is to reduce measurement error, which you can quantify by looking at the SME. When you discard a trial because of C.R.A.P., you reduce trial-to-trial variability, which decreases measurement error. However, you also reduce the number of trials, which increases measurement error. The SME takes both into account and tells you whether, when both factors are considered, the measurement error is better or worse.
        - You can tolerate more C.R.A.P. in a given channel if that channel will not contribute to your main analyses (e.g., if your analyses will be limited to other channels). In many cases, most of the channels are used only when plotting scalp maps. However, if you will be re-referencing after artifact detection, any channels that are part of the reference are important.
    - Note: Use a different flag for each type of artifact. This makes it possible to track how many artifacts of each type were flagged.
- The number of trials with versus without artifacts should be recorded. We do this aggregated across conditions (i.e., collapsed across bins). If the percentage of rejected trials is greater than 25%, my lab always excludes the subject from the final analyses. (We raise this threshold to 50% for studies of psychiatric and neurological conditions.) A different threshold for exclusion may be appropriate for other kinds of research. However, it is important for the exclusion criterion to be set prior to looking at the data.
- After a participant's data have been fully processed in this way (including visualizing the data and saving all the participant-specific analysis parameters in one or more spreadsheets), the participant's data should be reprocessed with a script
    - This should yield the same result, but it can avoid errors that may occur in manual processing
    - The ICA decomposition process does not need to be repeated when the data are reprocessed with a script. That is, the weights can be transferred from the **ICA decomposition dataset** created during the manual processing.

References

Bigdely-Shamlo, N., Mullen, T., Kothe, C., Su, K.-M., & Robbins, K. A. (2015). The PREP pipeline: Standardized preprocessing for large-scale EEG analysis. *Frontiers in Neuroinformatics*, *9*. https://doi.org/10.3389/fninf.2015.00016

de Cheveigné, A. (2020). ZapLine: A simple and effective method to remove power line artifacts. *NeuroImage, 207,* 116356. https://doi.org/10.1016/j.neuroimage.2019.116356

Klug, M., & Kloosterman, N. A. (2022). Zapline-plus: A Zapline extension for automatic and adaptive removal of frequency-specific noise artifacts in M/EEG. *Human Brain Mapping, 43(9)*, 2743–2758. https://doi.org/10.1002/hbm.25832

Luck, S. J. (2014). *An Introduction to the Event-Related Potential Technique, Second Edition*. MIT Press.

Mitra, P. P., & Pesaran, B. (1999). Analysis of Dynamic Brain Imaging Data. *Biophysical Journal*, *76*(2), 691–708. https://doi.org/10.1016/S0006-3495(99)77236-X

This page titled 15: Appendix 3- Example Processing Pipeline is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck directly on the LibreTexts platform.

- 14: Appendix 3: Example Processing Pipeline by Steven J Luck is licensed CC BY 4.0.

INDEX

A

analytic standardized measurement error
6.7: Exercise - The Signal-to-Noise Ratio

C

current density
5.8: Exercise- Current Density

G

global field power
5.9: Exercise- Global Field Power

H

Hillyard Principle
6.2: Design of the ERP CORE Visual Oddball P3b Experiment

O

oddballs
6.2: Design of the ERP CORE Visual Oddball P3b Experiment

This page titled Index is shared under a Public Domain license and was authored, remixed, and/or curated by Steven J Luck.

GLOSSARY

Accuracy- The similarity of a measured value to its actual, true value.

Alpha- In statistics, the criterion chosen for statistical significance (typically .05).

Alternative hypothesis- The hypothesis that a population parameter (e.g., a mean difference, deviation from a reference value, or correlation) differs from the value specified by the null hypothesis. Informally, the hypothesis that a true effect exists.

Antialiasing filter- A low-pass filter used in hardware prior to digitization to prevent aliasing artifacts.

Auditory brainstem response (ABR)- Rapid ERPs elicited by auditory stimuli within 10 ms of stimulus onset, originating from the ascending auditory brainstem pathway.

Bandpass filter- A filter that passes frequencies within an intermediate range while attenuating frequencies above and below that range. Equivalent to applying both a low-pass and a high-pass filter.

Bogus- Fake or untrue. In statistics, a "bogus-but-significant effect" refers to a Type I error (a statistically significant result where no true population effect exists).

CDA- See Contralateral Delay Activity.

Cohen's d- A metric of effect size for cases where means are being compared for two groups or conditions (or for one group relative to a reference value). When the means from two conditions are being compared within a single group, the effect size is termed dz instead of d.

Conductance- The ease with which electrical current passes through a conductor (the inverse of resistance).

Contralateral- On the opposite side. The brain is organized largely contralaterally; one hemisphere processes input from and controls the opposite side of space/body.

Contralateral delay activity (CDA)- A broad ERP component observed during the retention period of visual working memory tasks, calculated as the voltage difference between electrodes contralateral and ipsilateral to the remembered items.

Cosine wave- A sinusoidal wave representing the X-value of a unit circle at a given angle. Identical to a sine wave shifted leftward by 90° in phase.

Current- The flow of electrical charge through a conductor.

Decibel (dB)- A logarithmic scale for signal power or amplitude. A doubling of power corresponds to a 3 dB increase (a 10-fold increase is 10 dB). Because power is proportional to amplitude squared, a doubling of amplitude corresponds to a 6 dB increase (a 10-fold increase is 20 dB).

Dipole- A pair of equal and opposite electrical charges separated by a small distance.

Effect size- A quantitative measure of the magnitude of an experimental effect or the strength of a relationship, typically standardized relative to data variance. See also Cohen's d.

Electrical potential- See Voltage.

Electromyogram (EMG)- Electrical activity generated by muscle contraction, recorded via electrodes on the skin.

Event-related potential (ERP)- An electrical potential (voltage) time-locked to a specific event (e.g., a stimulus or response). Replaced the earlier term evoked potential for non-obligatory, cognitive or affective brain responses.

Event-related magnetic field (ERMF)- The magnetic field accompanying an ERP, recorded via magnetoencephalography (MEG).

Evoked potential (EP)- An older term for ERP, mainly now used for sensory responses. Equivalent to evoked response.

Expected value- The theoretical long-run average of a random variable over an infinite number of repetitions.

Experimentwise error rate- The probability of obtaining at least one false positive across all statistical tests in an experiment.

Familywise error rate- The probability of obtaining at least one false positive across all statistical tests in a family of related tests.

Fast Fourier transform (FFT)- An efficient method for performing a discrete Fourier transform.

Filter- A device or algorithm that attenuates specific signal components, such as specific frequency bands.

Fourier transform- A mathematical procedure for decomposing a continuous time-series waveform into a sum of sinusoids of varying frequencies, amplitudes, and phases.

Grand average- An ERP waveform created by averaging single-subject averaged ERPs across all participants in a study or group.

High-pass filter- A filter that passes high frequencies and attenuates low frequencies, commonly used to remove slow drift and skin potentials.

Impedance- Total opposition to alternating electrical current flow, combining resistance, capacitance, and inductance.

Ipsilateral- On the same side. See also Contralateral.

Lateralized readiness potential (LRP)- A motor preparation ERP component calculated as the difference in voltage between electrodes contralateral and ipsilateral to the responding hand.

Latency jitter- Trial-to-trial variability in the timing of a neural or behavioral event relative to a reference point.

Line noise- Electrical interference from the building's AC power system, occurring at 50 Hz or 60 Hz depending on the region.

Low-pass filter- A filter that passes low frequencies and attenuates high frequencies, used to eliminate high-frequency noise or prevent aliasing before digitization.

Magnetoencephalogram (MEG)- Magnetic fields that accompany activity.

Noise- Trial-to-trial voltage variations unrelated to the specific ERP component or feature being scored. Can originate from non-biological sources (e.g., equipment drift), non-neural biological sources (e.g., skin potentials), or background neural activity (e.g., ongoing alpha oscillations).

Notch filter- A filter that attenuates a narrow frequency band (e.g., 50/60 Hz line noise) while passing higher and lower frequencies.

Null hypothesis- The baseline hypothesis that no effect or difference exists in the underlying population (e.g., condition means are identical, or a correlation is zero).

One-tailed test- A directional statistical test that rejects the null hypothesis only if the observed effect is in a particular direction.

P value- The probability of obtaining a test statistic (or effect size) at least as extreme as the observed value if the null hypothesis were true. If $p < \text{alpha}$, the null hypothesis is provisionally rejected. See also alpha, effect size, Cohen's d, one-tailed test, two-tailed test.

Phase- The horizontal shift in a sinusoidal waveform.

Potential- See Voltage.

Power- In physics, electrical power equals voltage multiplied by current (or voltage squared for constant resistance). In statistics, the probability of correctly rejecting a false null hypothesis.

Precision- The degree of agreement among repeated measurements of the same quantity under identical conditions.

Reliability- In psychometrics, the ratio of true score variance to total observed variance across participants. A score is reliable to the extent that the observed variance is a result of true differences across subjects rather than measurement error or other

sources of noise.

Resistance- Opposition to direct electrical current flow.

Sine wave- A sinusoidal wave representing the Y-value of a unit circle at a given angle. Identical to a cosine wave shifted rightward by 90° in phase.

Sinusoidal waveform- A waveform shaped like a sine or cosine wave with a specific frequency, amplitude, and phase.

Signal-to-noise ratio (SNR)- The size of the signal is divided by the size of the noise. The signal can be quantified as an amplitude score, and the noise can be quantified as the standardized measurement error for that score.

Standardized measurement error (SME)- The standard error of measurement for an individual subject's ERP amplitude or latency score, quantifying precision of the score across trials. See also accuracy, precision, noise.

Statistical power- The probability that a true effect will be statistically significant. See also p value, Type I error, Type II error.

Two-tailed test- A nondirectional statistical test that rejects the null hypothesis no matter the direction of the observed effect.

Type I error- A false positive (bogus) result (rejecting the null hypothesis when it is actually true).

Type II error- A false negative result (accepting the null hypothesis when an effect actually exists).

Voltage- The electrical potential difference between two points that drives charges to flow through a conductor.

This page titled Glossary is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck.

DETAILED LICENSING

Overview

Title: Applied Event-Related Potential Data Analysis 2e

Webpages: 201

Applicable Restrictions: Noncommercial

All licenses found:

- CC BY-NC 4.0: 97% (195 pages)
- Public Domain: 2.5% (5 pages)
- Undeclared: 0.5% (1 page)

By Page

- Applied Event-Related Potential Data Analysis 2e - *CC BY-NC 4.0*
  - Front Matter - *CC BY-NC 4.0*
    - TitlePage - *CC BY-NC 4.0*
    - InfoPage - *Public Domain*
    - Table of Contents - *Public Domain*
    - Licensing - *Public Domain*
    - Hardware and Software Requirements - *CC BY-NC 4.0*
    - Preface - *CC BY-NC 4.0*
    - Acknowledgments - *CC BY-NC 4.0*
    - How to Cite This Book - *CC BY-NC 4.0*
  - 1: First Steps - *CC BY-NC 4.0*
    - 1.1: Getting Started - *CC BY-NC 4.0*
    - 1.2: Installing the Software and Downloading the Data - *CC BY-NC 4.0*
    - 1.3: Exercise- Loading and Viewing an EEG File - *CC BY-NC 4.0*
    - 1.4: Exercise - Viewing EEG Spectra - *CC BY-NC 4.0*
    - 1.5: Exercise- Loading and Plotting ERPs - *CC BY-NC 4.0*
    - 1.6: Understanding the Matlab Path - *CC BY-NC 4.0*
    - 1.7: Key Takeaways and References - *CC BY-NC 4.0*
  - 2: Processing the Data from One Participant in the ERP CORE N400 Experiment - *CC BY-NC 4.0*
    - 2.1: Data For This Chapter - *CC BY-NC 4.0*
    - 2.2: Design of the N400 Experiment - *CC BY-NC 4.0*
    - 2.3: Exercise - Looking at the EEG and the Event Codes - *CC BY-NC 4.0*
    - 2.4: Exercise- Filtering Out Low-Frequency Drifts from the EEG - *CC BY-NC 4.0*
    - 2.5: Exercise- Creating an EventList - *CC BY-NC 4.0*
    - 2.6: Exercise- Assigning Events to Bins with BINLISTER - *CC BY-NC 4.0*
    - 2.7: Exercise- Epoching and Baseline Correction - *CC BY-NC 4.0*
    - 2.8: Exercise- Artifact Detection - *CC BY-NC 4.0*
    - 2.9: Exercise- Averaged ERPs - *CC BY-NC 4.0*
    - 2.10: Exercise- Data Quality - *CC BY-NC 4.0*
    - 2.11: Review of Processing Steps - *CC BY-NC 4.0*
    - 2.12: A Simple Matlab Script - *CC BY-NC 4.0*
    - 2.13: Key Takeaways and References - *CC BY-NC 4.0*
  - 3: Processing Multiple Participants in the ERP CORE N400 Experiment - *CC BY-NC 4.0*
    - 3.1: Data for This Chapter - *CC BY-NC 4.0*
    - 3.2: Exercise- Organizing and Loading the Data from 10 Participants - *CC BY-NC 4.0*
    - 3.3: Exercise- Examining the Single-Participant ERPsets - *CC BY-NC 4.0*
    - 3.4: Exercise- “Bad” Data - *CC BY-NC 4.0*
    - 3.5: Exercise- Making a Grand Average - *CC BY-NC 4.0*
    - 3.6: Exercise- Low-Pass Filtering - *CC BY-NC 4.0*
    - 3.7: Exercise - Scoring N400 Amplitude - *CC BY-NC 4.0*
    - 3.8: Exercise- Simple Statistical Analysis of N400 Data - *CC BY-NC 4.0*
    - 3.9: Exercise- A More Complex Analysis - *CC BY-NC 4.0*
    - 3.10: Exercise- ERP Channel Operations - *CC BY-NC 4.0*
    - 3.11: Exercise- ERP Bin Operations - *CC BY-NC 4.0*
    - 3.12: Review of Processing Steps - *CC BY-NC 4.0*
    - 3.13: Matlab Scripts For This Chapter - *CC BY-*

NC 4.0
- 3.14: Key Takeaways and References - *CC BY-NC 4.0*

- 4: Filtering the EEG and ERPs - *CC BY-NC 4.0*
  - 4.1: Data for this Chapter - *CC BY-NC 4.0*
  - 4.2: Classes of Filters - *CC BY-NC 4.0*
  - 4.3: Exercise- Assessing the Frequency Content of the Noise - *CC BY-NC 4.0*
  - 4.4: Exercise- Filtering the 60 Hz Noise - *CC BY-NC 4.0*
  - 4.5: Practical Advice About Filtering - *CC BY-NC 4.0*
  - 4.6: Exercise- Filtering as a Linear Operation - *CC BY-NC 4.0*
  - 4.7: Exercise- The Impulse Response Function - *CC BY-NC 4.0*
  - 4.8: Exercise- Filtering a Series of Impulses (and a Sneaky Introduction to Convolution) - *CC BY-NC 4.0*
  - 4.9: Predicting Waveform Distortion from the Impulse Response Function - *CC BY-NC 4.0*
  - 4.10: Exercise- High-Pass Filtering - *CC BY-NC 4.0*
  - 4.11: Background- How Should We Define the Optimal Filter? - *CC BY-NC 4.0*
  - 4.12: Background- Quantifying Signal, Noise, and Waveform Distortion - *CC BY-NC 4.0*
  - 4.13: Exercise- Creating artificial waveforms - *CC BY-NC 4.0*
  - 4.14: Exercise- Quantifying Waveform Distortion in the Artificial Waveforms - *CC BY-NC 4.0*
  - 4.15: Exercise- Quantifying Data Quality - *CC BY-NC 4.0*
  - 4.16: Selecting the Optimal Filter - *CC BY-NC 4.0*
  - 4.17: Matlab Scripts for this Chapter - *CC BY-NC 4.0*
  - 4.18: Key Takeaways and References - *CC BY-NC 4.0*
- 5: Referencing and Other Channel Operations - *CC BY-NC 4.0*
  - 5.1: Data for This Chapter - *CC BY-NC 4.0*
  - 5.2: Background- Understanding Active, Reference, and Ground Electrodes - *CC BY-NC 4.0*
  - 5.3: Exercise- Working with the Artificial Data - *CC BY-NC 4.0*
  - 5.4: Exercise- Average Mastoids as the Reference - *CC BY-NC 4.0*
  - 5.5: Exercise- Re-Referencing the N400 ERP CORE Data - *CC BY-NC 4.0*
  - 5.6: Exercise- The Average Reference - *CC BY-NC 4.0*
  - 5.7: What is the Best Reference Site? - *CC BY-NC 4.0*
  - 5.8: Exercise- Current Density - *CC BY-NC 4.0*
  - 5.9: Exercise- Global Field Power - *CC BY-NC 4.0*
  - 5.10: Exercise- Referencing the EEG Data from the ERP CORE N400 Experiment - *CC BY-NC 4.0*
  - 5.11: Exercise- Other Common Re-Referencing Scenarios - *CC BY-NC 4.0*
  - 5.12: Matlab Script For This Chapter - *CC BY-NC 4.0*
  - 5.13: Key Takeaways and References - *CC BY-NC 4.0*
- 6: Assigning Events to Bins, Averaging, Baseline Correction, and Assessing Data Quality - *CC BY-NC 4.0*
  - 6.1: Data for This Chapter - *CC BY-NC 4.0*
  - 6.2: Design of the ERP CORE Visual Oddball P3b Experiment - *CC BY-NC 4.0*
  - 6.3: The Event Code Scheme - *CC BY-NC 4.0*
  - 6.4: Overview of Bin Descriptor Files - *CC BY-NC 4.0*
  - 6.5: Exercise - A Basic Assignment of Events to Bins - *CC BY-NC 4.0*
  - 6.6: Exercise - Looking at the Averaged ERPs - *CC BY-NC 4.0*
  - 6.7: Exercise - The Signal-to-Noise Ratio - *CC BY-NC 4.0*
  - 6.8: Exercise - Response-Locked Averaging - *CC BY-NC 4.0*
  - 6.9: Exercise - Comparing Correct and Error Trials to See the Error-Related Negativity - *CC BY-NC 4.0*
  - 6.10: Exercise - Sequential Analysis of the P3b - *CC BY-NC 4.0*
  - 6.11: Exercise - Combining Bins - *CC BY-NC 4.0*
  - 6.12: Exercise - Overlap - *CC BY-NC 4.0*
  - 6.13: Matlab Script For This Chapter - *CC BY-NC 4.0*
  - 6.14: Key Takeaways and References - *CC BY-NC 4.0*
- 7: Inspecting the EEG and Interpolating Bad Channels - *CC BY-NC 4.0*
  - 7.1: Data for This Chapter - *CC BY-NC 4.0*
  - 7.2: Design of the Mismatch Negativity (MMN) Experiment - *CC BY-NC 4.0*
  - 7.3: Video Demonstration- Performing an Initial Inspection of a Participant's EEG - *CC BY-NC 4.0*
  - 7.4: Background- Interpolating Bad Channels - *CC BY-NC 4.0*
  - 7.5: Exercise - Interpolating Bad Channels - *CC BY-NC 4.0*
  - 7.6: Matlab Script For This Chapter - *CC BY-NC 4.0*
  - 7.7: Key Takeaways and References - *CC BY-NC 4.0*
- 8: Artifact Detection and Rejection - *CC BY-NC 4.0*
  - 8.1: Data for This Chapter - *CC BY-NC 4.0*
  - 8.2: Overview - *CC BY-NC 4.0*
  - 8.3: Background- Why Do We Reject Artifacts? - *CC BY-NC 4.0*

- 8.4: Background- The General Approach - *CC BY-NC 4.0*
- 8.5: Exercise- Simple Blink Detection - *CC BY-NC 4.0*
- 8.6: Exercise- Adjusting the Threshold - *CC BY-NC 4.0*
- 8.7: Exercise- Manually Marking and Unmarking Artifacts - *CC BY-NC 4.0*
- 8.8: An Iterative Approach to Setting Parameters - *CC BY-NC 4.0*
- 8.9: Exercise- Effects of Artifact Rejection on Data Quality - *CC BY-NC 4.0*
- 8.10: Exercise- Better Blink Detection - *CC BY-NC 4.0*
- 8.11: Exercise- Commonly Recorded Artifactual Potentials (C.R.A.P.) - *CC BY-NC 4.0*
- 8.12: Exercise- Detecting Eye Movements - *CC BY-NC 4.0*
- 8.13: Using Artifact Detection to Avoid Changes to Visual Inputs - *CC BY-NC 4.0*
- 8.14: The ERP CORE N2pc Experiment - *CC BY-NC 4.0*
- 8.15: Exercise- Visualizing the Eye Movements - *CC BY-NC 4.0*
- 8.16: Exercise- Using the Averaged HEOG to Visualize Consistent Eye Movements - *CC BY-NC 4.0*
- 8.17: Exercise- A Two-Stage Strategy for Eliminating Small But Consistent Eye Movements - *CC BY-NC 4.0*
- 8.18: Matlab Script For This Chapter - *CC BY-NC 4.0*
- 8.19: Key Takeaways and References - *CC BY-NC 4.0*

◦ 9: Artifact Correction with Independent Component Analysis - *CC BY-NC 4.0*
- 9.1: Data for this Chapter - *CC BY-NC 4.0*
- 9.2: Exercise- A First Pass at ICA-Based Blink Correction - *CC BY-NC 4.0*
- 9.3: Background- A Quick Conceptual Overview of ICA - *CC BY-NC 4.0*
- 9.4: Exercise- Making ICA Work Better - *CC BY-NC 4.0*
- 9.5: Exercise- Transferring the Weights, Identifying the ICs, Correcting the EEG, and Interpolating - *CC BY-NC 4.0*
- 9.6: Exercise- Deleting C.R.A.P. Prior to ICA - *CC BY-NC 4.0*
- 9.7: General Recommendations for ICA-Based Artifact Correction - *CC BY-NC 4.0*
- 9.8: Background- Evaluating the Effectiveness of Artifact Correction+Rejection - *CC BY-NC 4.0*
- 9.9: Exercise- Determining Whether Blinking is a Confound - *CC BY-NC 4.0*
- 9.10: Exercise- Was Blink Correction Effective? - *CC BY-NC 4.0*
- 9.11: Exercise- Does Artifact Correction+Rejection Improve Data Quality? - *CC BY-NC 4.0*
- 9.12: Matlab Scripts For This Chapter - *CC BY-NC 4.0*
- 9.13: Key Takeaways and References - *CC BY-NC 4.0*

◦ 10: Scoring and Statistical Analysis of ERP Amplitudes and Latencies - *CC BY-NC 4.0*
- 10.1: Data for This Chapter - *CC BY-NC 4.0*
- 10.2: Design of the Flankers Experiment - *CC BY-NC 4.0*
- 10.3: Exercise- Examining the Grand Averages - *CC BY-NC 4.0*
- 10.4: Exercise- A First Pass at Scoring and Statistical Analysis - *CC BY-NC 4.0*
- 10.5: Exercise- Simplifying the Statistical Analysis - *CC BY-NC 4.0*
- 10.6: Exercise- Peak Amplitude - *CC BY-NC 4.0*
- 10.7: Exercise- Peak Latency - *CC BY-NC 4.0*
- 10.8: Exercise- Fractional Area Latency - *CC BY-NC 4.0*
- 10.9: Exercise- Quantifying Onset Latency - *CC BY-NC 4.0*
- 10.10: Exercise- Collapsing Across Channels and Correlating Latencies with Response Times - *CC BY-NC 4.0*
- 10.11: Matlab Scripts For This Chapter - *CC BY-NC 4.0*
- 10.12: Key Takeaways and References - *CC BY-NC 4.0*

◦ 11: EEGLAB and ERPLAB Scripting - *CC BY-NC 4.0*
- 11.1: Data for This Chapter - *CC BY-NC 4.0*
- 11.2: Expected Background Knowledge - *CC BY-NC 4.0*
- 11.3: Bugs as an Opportunity for Growth - *CC BY-NC 4.0*
- 11.4: Design of the N170 Experiment - *CC BY-NC 4.0*
- 11.5: Exercise- The Matlab Command Line and the EEG Variable - *CC BY-NC 4.0*
- 11.6: Exercise- The ALLEEG Variable and Refreshing the ERPLAB Studio GUI - *CC BY-NC 4.0*
- 11.7: Exercise- EEG.history and eegh - *CC BY-NC 4.0*
- 11.8: Exercise- From the Command Line to a Script - *CC BY-NC 4.0*
- 11.9: Exercise- Using a Variable for the Path - *CC BY-NC 4.0*
- 11.10: Exercise- Loops - *CC BY-NC 4.0*
- 11.11: Exercise- Looping Through Data from Multiple Participants - *CC BY-NC 4.0*
- 11.12: Rapid Cycling Between Coding and Testing - *CC BY-NC 4.0*
- 11.13: Exercise- Using a Script to Reference EEGsets - *CC BY-NC 4.0*
- 11.14: Exercise- Improving the Script and Adding 3D Channel Locations - *CC BY-NC 4.0*

- 11.15: Exercise- Preprocessing the EEG and Using a Spreadsheet to Store Subject-Specific Information - *CC BY-NC 4.0*
- 11.16: Exercise- Building an Entire EEG Processing Pipeline - *CC BY-NC 4.0*
- 11.17: Exercise- Averaging, Quantifying Data Quality, and Plotting - *CC BY-NC 4.0*
- 11.18: Exercise- Scoring Amplitudes and Latencies and Performing Statistical Analyses - *CC BY-NC 4.0*
- 11.19: Key Takeaways and References - *CC BY-NC 4.0*

◦ 12: Multivariate Pattern Classification (Decoding) - *CC BY-NC 4.0*
- 12.1: Data for This Chapter - *CC BY-NC 4.0*
- 12.2: Example- Decoding Face Identity and Emotional Expression - *CC BY-NC 4.0*
- 12.3: The Essence of Decoding - *CC BY-NC 4.0*
- 12.4: Exercise- Binary Decoding of ID 1 versus ID 2 - *CC BY-NC 4.0*
- 12.5: Exercise- Decoding Multiple BESTsets and Making a Grand Average - *CC BY-NC 4.0*
- 12.6: Exercise- 4-way Decoding of Face Identity and Emotional Expression - *CC BY-NC 4.0*
- 12.7: Exercise- Exporting the Data for Statistical Analysis - *CC BY-NC 4.0*
- 12.8: Exercise- Using Decoding in Place of Conventional Mean Amplitude Analyses - *CC BY-NC 4.0*
- 12.9: Exercise- Temporal Generalization - *CC BY-NC 4.0*
- 12.10: Demonstration- Decoding in the Frequency-Domain - *CC BY-NC 4.0*
- 12.11: Exercise- Artificially Inflated Decoding with Unbalanced Classes - *CC BY-NC 4.0*
- 12.12: Exercise- Methods for Appropriate Decoding with Unbalanced Classes - *CC BY-NC 4.0*
- 12.13: Exercise- Using Area Under the Curve (AUC) with the ERP CORE P3b Oddball Paradigm - *CC BY-NC 4.0*
- 12.14: Matlab Script For This Chapter - *CC BY-NC 4.0*
- 12.15: Key Takeaways and References - *CC BY-NC 4.0*

◦ 13: Appendix 1- A Very Brief Introduction to EEG and ERPs - *CC BY-NC 4.0*

◦ 14: Appendix 2- Troubleshooting Guide - *CC BY-NC 4.0*
- 14.1: Data for This Appendix - *CC BY-NC 4.0*
- 14.2: A2.1 The First Step - *CC BY-NC 4.0*
- 14.3: A2.2 Some Basic Solutions - *CC BY-NC 4.0*
- 14.4: A2.3 Taking a Scientific Approach - *CC BY-NC 4.0*
- 14.5: A2.4 Deciphering Matlab's Error Messages - *CC BY-NC 4.0*
- 14.6: A2.5 Debugging Scripts by Performing Experiments and Collecting Data - *CC BY-NC 4.0*
- 14.7: A2.6 Avoiding Bugs in Your Scripts with Good Programming Practices - *CC BY-NC 4.0*
- 14.8: A2.7 References - *CC BY-NC 4.0*

◦ 15: Appendix 3- Example Processing Pipeline - *CC BY-NC 4.0*

◦ Back Matter - *Undeclared*
- Index - *Public Domain*
- Glossary - *CC BY-NC 4.0*
- Detailed Licensing - *Public Domain*

This page titled Applied Event-Related Potential Data Analysis 2e is shared under a CC BY-NC 4.0 license and was authored, remixed, and/or curated by Steven J Luck.