WHAT YOU SEE IS WHAT YOU GET: A CLOSER LOOK AT BIAS IN THE VISUAL WORLD PARADIGM

by

Collin Nolte

A thesis submitted in partial fulfillment of the
requirements for the Doctor of Philosophy
degree in Biostatistics in the
Graduate College of The
University of Iowa

May 2023

Thesis Committee: Patrick Breheny, Thesis Supervisor
Jacob Oleson
Bob McMurray
Grant Brown
Kristi Hendrickson

Copyright by
COLLIN NOLTE
2023
All Rights Reserved

ACKNOWLEDGMENTS

To my parents, Sean and Amy, whose enduring commitment to one another and to our family created a space of safety, encouragement, and exploration. To my mom, whose unwavering and unconditional support was a source of constant strength, even when it felt as if all were lost. And to my dad whose diligence and tireless commitment to attending to what needed to be done. Who showed me resolve in the face of difficulty

To Patrick, for suffering the unique privilege of serving as my advisor and mentor. Who challenged me to seek excellence and to dispassionately pursue the truth. And who showed me that no matter how much work there was to be done, that there was always time to talk about R.

To Pake, the iron from which I crafted the most intricate parts of my character. Who shared with the importance of family, community, and the fruits of brotherly love.

And finally, to Vixen. Of all the bitches, you will always be my favorite ♥

ABSTRACT

In 1995 the Visual World Paradigm was first introduced as an experimental paradigm relating eye-tracking data to lexical activation. This was done by measuring the location of a participant's visual fixation in real time in response to a spoken word. Shortly following this was the introduction of the "proportion of fixations" method, whereby indicators of fixations in the VWP were aggregated across trials at dense time slices, creating an ostensible curve demonstrating the proportion of trials in which participants were fixated on particular objects at each time. In 2017, methods were introduced to make temporal analysis of these curves tractable via bootstrapping and a novel α correction to counteract the multiple comparison problems implicit in densely sampled time series.

This dissertation improves upon the field in multiple ways. First, we reintroduce the `bdots` package with a dramatically simplified user interface. Most prominent among these changes include the ability for users to create and fit parametric functions independent the `bdots` software as well as the introduction of permutation testing for identifying temporal differences between groups. We also identify and correct methodological issues found in the original bootstrapping algorithm that, when unaccounted for, potentially lead to family-wise error rates of over 90%. Both this correction and the newly introduced permutation test demonstrate quality maintenance of the FWER across a robust collection of underlying assumptions without significant losses in power.

And finally, we propose a new generative model that links eye mechanics to lexical activation, along with a novel "look onset" method that seeks to replace the proportion of fixations method. We demonstrate asymptotic consistency between our generative model and the look onset method, both with regards to the recovery of subject-specific activation curves, as well as with the identification of systematic temporal differences between groups. We conclude by demonstrating this utility with data collected from prior studies as well as by providing a number of avenues for future research.

PUBLIC ABSTRACT

In 1995 the Visual World Paradigm was first introduced as an experimental paradigm relating eye-tracking data to lexical activation. This was done by measuring the location of a participant's visual fixation in real time in response to a spoken word. Shortly following this was the introduction of the "proportion of fixations" method, whereby indicators of fixations in the VWP were aggregated across trials at dense time slices, creating an ostensible curve to help visualize and quantify the time course of lexical activation. In 2017, methods were introduced to make temporal analysis of these curves tractable via bootstrapping and a novel α correction to counteract the multiple comparison problems implicit in densely sampled time series.

This dissertation contributes to the field in multiple ways. First, we reintroduce the `bdots` package that significantly enhances researchers' ability to draw conclusions from VWP studies. Accompanying this are drastic changes to the underlying methodology to accommodate a far more robust set of assumptions regarding data generation. And finally, we introduce a generative model for relating eye mechanics to lexical activation alongside a novel look onset method that seeks to replace the proportion of fixation method currently in use.

TABLE OF CONTENTS

LIST OF TABLES

Table

LIST OF FIGURES

Figure

CHAPTER 1
INTRODUCTION

This dissertation is made up of three chapters, each addressing some aspect of eye tracking in the Visual World Paradigm (VWP) or `bdots`, the R software created to accompany the analysis. Each of these chapters is also written as a separate manuscript. As such, there will be some overlap and redundancy between chapters, though this is intentional as they are intended to stand alone. Although they are presented necessarily in a linear fashion, they can be read in any order.

The first chapter addresses major changes in the `bdots` package, an implementation of the "bootstrapped differences in time series" methodology first introduced in **?**. The package offers a user-friendly way for fitting parametric functions to subject-specific time series data along with a bootstrapping procedure for identifying temporal differences between experimental groups. Major updates in this reiterated version include a simplification to the user interface, increased functionality, and general quality-of-life improvements. Included also is a use-case example performing an analysis with non-VWP data: an analysis of tumor growth in mice across a number of experimental treatment groups. This chapter also includes important changes to the underlying methodology, including a change to the way the original bootstrapping algorithm is constructed, as well as the introduction of a permutation test to control the family wise error rate. These methods are included both for completeness for new users and to demonstrate important changes for existing users. Justification for these changes is provided in Chapter 4.

The second chapter addresses the Visual World Paradigm itself. It begins with a general review of the VWP, eye tracking, and how these are used in relation to one another to assess lexical activation in time. This chapter examines critical issues in the current "proportion of fixation" method, presents a more comprehensive model for relating the underlying cognitive mechanisms of interest with the observed physiological behavior, and introduces a novel "look onset" method for using data in the VWP. We justify our approach both with a theoretical argument as to why this method would be superior for the recovery of the underlying cognitive mechanisms and verify

the veracity of our claims with a number of simulations. We conclude this chapter by suggesting a number of ways this new method can be incorporated into current theories of lexical activation.

Finally, the third chapter of this dissertation addresses the underlying methodology presented in the the motivating methodological paper for the `bdots` package [**?**]. Specifically, this paper introduced a novel correction to highly correlated test statistics constructed via bootstrapping from densely sampled time series data. We identify a critical issue in the underlying assumptions as they relate to empirically observed data and the impacts on the family-wise error rate when these assumptions do not hold. Presented in this chapter is a modification of the original bootstrapping algorithm that still uses the modified Bonferonni correction based on the autocorrelation of the test statistics in time, as well as the introduction of a permutation test for identifying temporal differences. We demonstrate that both the modified bootstrap and the permutation test maintain a FWER close to the nominal rate across a number of differing assumptions and conclude with a comparison of power for each of the methods discussed. The results of Chapter 4 provide justification for the underlying changes in Chapter 2.

All of the code used in this dissertation is available at github.com/collinn.

CHAPTER 2
REINTRODUCTION OF THE BDOTS PACKAGE

2.1 Introduction

In 2017, **?** introduced a method for detecting time-specific differences in the trajectory of outcomes between experimental groups. Accompanying this was a novel method for controlling the family-wise error rate, particularly in the case of densely sampled time series where constructed test statistics exhibit high degrees of autocorrelation. This was followed up with in 2018 with the introduction of the `bdots` package to CRAN [**?**]. Here we reintroduce the `bdots` package as a significant upgrade that broadly expands the capabilities of the original.

This manuscript is not intended to serve as a complete guide for using the `bdots` package. Instead, the purpose is to showcase major changes and improvements, with those seeking a more comprehensive treatment directed to the package vignettes which are included in the appendix. Rather than taking a "compare and contrast" approach, we will first enumerate the major changes, followed by a general demonstration of package use:

1. Major changes to underlying methodology with implications for prior users of the package
2. Simplified user interface
3. Introduction of user defined curves
4. Permit fitting for arbitrary number of groups
5. Automatic detection of paired tests based on subject identifier
6. Allows for non-homogeneous sampling of data across subjects and groups
7. Introduce formula syntax for bootstrapping difference function
8. Interactive refitting process

We start by clearly describing the type of problem that `bdots` has been created to solve.

Bootstrapped differences in time series

A typical problem in the analysis of differences in time series, and the kind that `bdots` is intended to solve, involves that of two or more experimental groups containing subjects whose

responses are measured over time. This may include the growth of tumors in mice or the change in the proportion of fixations over time in the context of the VWP. In either case, we assume that each of the subjects $i = 1, \ldots, n$ has observed data of the following form:

$$y_{it} = f(t|\theta_i) + \epsilon_{it} \tag{2.1}$$

where f represents a functional mean structure while the error structure of ϵ_{it} is open to being either IID or possess an AR(1) structure. At present, `bdots` requires that each of the subjects being compared have the same parametric function f. This is not strictly necessary at the theoretical level, and future goals of the package include accommodating non-parametric functions. While the mean structures for each of the subjects is required to be of the same parametric form $f(\cdot|\theta)$, each differs in their instance of their own subject-specific parameters, θ_i.

An explicit assumption of the current iteration of `bdots` is that each subject *i*s' parameters within a group $g = 1, \ldots, G$ is drawn from a group level distribution

$$\theta_i \sim N(\mu_g, V_g). \tag{2.2}$$

Bootstrapping these parameters gives an estimate of this distribution, which is then in turn used to construct a distribution of functions f. This in turn gives a representation of the temporal changes in group characteristics. It is precisely the identification of if and when these temporal changes differ between groups that `bdots` seeks to accomplish.

Homogeneous Means Assumption

The assumption presented in Equation 2.2 differs from the original iteration of `bdots` in a critical way. In **?**, there was no assumption of variability between subject parameters, congruent with the assumption that $\theta_i = \theta_j$ for all subjects i, j within a group. The most relevant consequence of the homogeneous means assumption is that there is no estimate of between-subject variability, resulting in a drastic inflation of the type I error. A detailed treatment of this issue and how it is

resolved is handled in Chapter 4. For now, we will give a methodological overview of the process used by `bdots` along with a presentation of an updated bootstrapping process and the introduction of a permutation test.

2.2 Methodology and Overview

A standard analysis using `bdots` consists of two steps: fitting the observed data to a specified parametric function, $f(\cdot|\theta)$, and then using the observed variability to construct estimates of the distributions of functions for groups whose differences we wish to investigate. Here, we briefly detail how this is implemented in practice and introduce the new methodologies in `bdots`.

2.2.1 Estimating Subject-Specific Curves

We begin with the assumption that for subject i in group g, we have collected observed data of the form given in Equation 2.1, with the subject specific parameter θ_i following the distribution in Equation 2.2. Each subject is then fit in `bdots` via the nonlinear curve fitting function `nlme::gnls`, returning for each set of observed data an estimated set of parameters $\hat{\theta}_i$ and their associated standard errors. From these estimates we are able to construct a sampling distribution for each subject:

$$\hat{\theta}_i \sim N(\theta_i, s_i^2). \tag{2.3}$$

Just as in **?**, this distribution provides an estimate of the within-subject uncertainty in the estimate of subject-specific function parameters.

2.2.2 Estimating Group Distributions

To help clarify the discussion that follows, we begin by reiterating a few important points. First, each subject that we consider in a `bdots` analysis has a set of subject-specific parameters, θ_i, which define the shape of their time series, itself assumed to be a parametric function. The

collection of parameters of all subjects within a particular group make up a distribution of group parameters, the distribution given in Equation 2.2. And although this distribution is specifically a distribution of *parameters*, any sample of parameters from this distribution can be used to construct a distribution of *functions* that define a time series. In other words, we begin by observing a time series for each subject. We then fit parametric functions to these observed time series. These estimated parameters themselves make up a distribution of parameters. This distribution of parameters can then be used to construct a distribution of time series.

In addition to the distribution of group parameters (Equation 2.2), we have also described a distribution of subject-specific parameters (Equation 2.3). The motivation for the distribution given in Equation 2.3 is as follows: for each individual subject, we wish to recover an estimate of the parameters that would be used to fit a parametric function to the observed time series data. In doing so, there is some degree of uncertainty that comes from the curve fitting process as evidenced by the standard errors. This accounts for within-subject variability. The insight of **?** was that this *within-subject variability* should be accounted for when estimating the *between-subject variability* present in the distribution of group parameters. That is, if each individual subjects' estimate was associated with a high degree of uncertainty, it would be appropriate for this uncertainty to be reflected in the total uncertainty representing the entire group.

With this in mind, we now propose the following algorithm for creating bootstrapped estimates of the *group level distributions*:

1. For a group of size n, select n subjects from the group *with replacement*. This allows us to construct an estimate of V_g from Equation 2.2.

2. For each selected subject i in bootstrap b, draw a set of parameters from the *subject-specific* distribution

$$\theta^*_{ib} \sim N(\hat{\theta}_i, s_i^2). \tag{2.4}$$

3. Find the mean of each of the bootstrapped θ^*_{ib} in group g to construct the bth group bootstrap,

θ^*_{gb} where

$$\theta^*_{gb} = \frac{1}{n}\sum \theta^*_{ib}, \qquad \theta^*_{gb} \sim N\left(\mu_g, \frac{1}{n}V_g + \frac{1}{n^2}\sum s_i^2\right). \tag{2.5}$$

That is, the bootstrapped estimate follows a *group level* distribution

4. Perform steps (1)-(3) B times, using each θ^*_{gb} to construct a sample of population curves, $f(\cdot|\mu_g)$.

Note that the group level distribution in Equation 2.5 differs from that under the homogeneous means assumption by the factor of V_g in the variance term on account of the fact that subjects were originally sampled *without* replacement

The final population curves from (4) can be used to create estimates of the mean response and an associated standard deviation at each time point for each of the groups bootstrapped. These estimates are used both for plotting and in the construction of confidence intervals. They also can be, but do not necessarily have to be, used to construct a test statistic, which is the topic of our next section.

2.2.3 Hypothesis Testing for Statistically Significant Differences in Time Series

We now turn our attention to the primary goal of an analysis in `bdots`, the identification of time windows in which the distribution of curves of two groups differ significantly. A problem unique to the ones addressed by `bdots` is that of multiple testing; and especially in densely sampled time series, we must account for multiple testing while controlling the family-wise error rate (FWER). There are primarily two ways by which we are able to do this which we detail below.

2.2.3.1 α Adjustment

The first method whereby we control the FWER involves making adjustments to the nominal alpha value such as the case with a Bonferonni correction. Just as in the original iteration of `bdots`, we are able to first construct test statistics from the bootstrapped estimates described in the previous

section. These bootstrapped test statistics $T_t^{(b)}$ can be written as

$$T_t^{(b)} = \frac{(\overline{p}_{1t} - \overline{p}_{2t})}{\sqrt{s_{1t}^2 + s_{2t}^2}}, \tag{2.6}$$

where $\overline{p}_{gt}$ and s_{gt}^2 are mean and standard deviation estimates of the estimated functions at each time point t and for groups 1 and 2, respectively. As was demonstrated in **?**, these test statistics can be highly correlated in the presence of densely sampled test statics, leading to an inflated type I error. The FWER in this case can be controlled with the adjustment proposed in **?**. In addition to this, adjustments to the nominal alpha can also be made using all of the adjustments present in `p.adjust` from the R `stats` package.

2.2.3.2 Permutation Testing

In addition to modified correction based on the bootstrapped test statistics, `bdots` provides a permutation test for controlling the FWER without any additional assumptions of autocorrelation.

In doing so, we begin by creating an observed test statistic in the following way: first, taking each subject's estimated parameter $\hat{\theta}_i$, we find the subject's corresponding parametric curve $f(t|\hat{\theta}_i)$. Within each group, we use *these* curves to create estimates of the mean population curves and associated standard errors at each point[1]. Letting p_{gt} and s_{gt}^2 represent the mean population curve and standard error for group g at time t, we define our observed permutation test statistic,

$$T_t^{(p)} = \frac{|\overline{p}_{1t} - \overline{p}_{2t}|}{\sqrt{s_{1t}^2 + s_{2t}^2}}. \tag{2.7}$$

.

We then going about using permutations to construct a null distribution against which to compare the observed statistics from Equation 2.7. We do so with the following algorithm:

[1]This differs from the bootstrapped test statistic in which the mean of the subjects' parameters was used to fit a population curve, i.e., $\frac{1}{n}\sum f(t|\theta_i)$ compared with $f\left(t|\frac{1}{n}\sum \theta_i\right)$. The implications of this have not been investigated any further

1. Assign to each subject a label indicating group membership

2. Randomly shuffle the labels assigned in (1.), creating two new groups

3. Recalculate the test statistic $T_t^{(p)}$, recording the maximum value from each permutation

4. Repeat (2.)-(3.) P times. The collection of P statistics will serve as our null distribution, denoted $\widetilde{T}$. Let $\widetilde{T}_\alpha$ be the $1-\alpha$ quantile of $\widetilde{T}$. Areas where the observed $T_t^{(p)} > \widetilde{T}_\alpha$ are designated significant.

Paired statistics can also be constructed in both the bootstrap and permutation methods. This is implemented by ensuring that at each bootstrap the same subjects are selected for each group or by ensuring that each permuted group contains one observation from each subject.

A demonstration of power and FWER control for both the heterogeneous bootstrap and permutation test are given in Chapter 4.

In the following sections, we detail the two major components of an analysis using the `bdots` package, curve fitting and the identification of statistically significant differences in time series. Within each section, we will begin with a high level explanation of the major changes that have been made, along with a detailed description of the function syntax. We then concretize this discussion with a real world example along with illustrations of function calls and return objects.

2.3 Curve Fitting

The first step in performing an analysis with `bdots` involves specifying the parametric function which defines the mean structure from Equation 2.1 and fitting curves to the observed data for each subject. Throughout this discussion and into the next section, we will use as our real world example a comparison of tumor growth for the 451LuBr cell line in mice with repeated measures across five treatment groups. A depiction of this data is given in Figure 2.1.

```
> head(mouse, n = 10)
      Volume Day Treatment ID
 1:   47.432   0         A  1
 2:   98.315   5         A  1
 3:  593.028  15         A  1
 4:  565.000  19         A  1
 5: 1041.880  26         A  1
 6: 1555.200  30         A  1
 7:   36.000   0         B  2
 8:   34.222   4         B  2
 9:   45.600  10         B  2
10:   87.500  16         B  2
```

Figure 2.1: Illustration of Mouse data in long format

A new feature of `bdots` is the ability to fit and analyze subjects with non-homogeneous time samples, which we see in the `Day` variable values in the mouse data in Figure 2.1.

A new feature of `bdots` is the ability to fit and analyze subjects with non-homogeneous time samples. For example, consider the `Day` column for our mouse data shown in Figure 2.1, where the first four observations for ID 1 are all different than those for ID 2. For the present analysis, we will be interested in determining if and when the trajectory of tumor growth (measured in volume) changes between any two treatment groups.

There are two primary functions in the `bdots` package: one for fitting the observed data to a parametric function and another for estimating group distributions and identifying time windows where they differ significantly. The first of these, `bfit`, is the topic of this section.

The `bfit` Function

The curve fitting process is performed with the function `bfit` , taking the following arguments:

```
bfit(data, subject, time, y, group, curveType, ar, ...)
```

Figure 2.2: Main arguments to `bfit`, though see `help(bfit)` for additional options

The data argument takes the name of the dataset being used. subject is the subject identifier column in the data and should be passed as a character. It is important to note here that the identification of paired data is now done automatically; in determining if two experimental groups are paired, bdots checks that the intersection of subjects in each of the groups are identical with the subjects in each of the groups individually. As such, it will be important for the user to be sure that if the data is paired that the subject identifiers are the same between subject groups.

The time and y arguments are column names of the time variable and outcome, respectively. Similarly, group takes as an argument a character vector of each of the group columns that are meant to be fit, accommodating the fact that bdots is now able to fit an arbitrary number of groups at once provided that the outcomes in each group adopt the same parametric form. And lastly, the curveType argument, which is used to specify details of the parametric function to which the data will be fit. As this argument is more involved, we will address it separately in the next section.

curveType functions

Whereas the previous iteration of bdots had a separate fitting function for each parametric form (i.e., logistic.fit for fitting data to a four-parameter logistic), we are now able to specify the curves we wish to fit independent of the fitting function, passed as an argument to bfit. This is done with the argument curveType. Unlike the previous arguments which took either a data.frame or character vector, curveType takes as an argument a function call, for example, logistic(). In short, this allows the user to pass additional arguments to further specify the curve. For example, among the parametric functions now included in bdots is the polynomial function, taking as an additional argument the number of degrees we wish to use. To fit the observed data with a five parameter polynomial in bfit, one would then pass the argument curveType = polynomial(degree = 5). Curve functions currently included in bdots are logistic(), doubleGauss(), expCurve(), and polynomial(). In addition to the functions provided by default in the bdots package, bfit can also accept user-created curves; a detailed explanation of how this is done is provided in the appendix as well as with vignette("bdots").

Using our mouse data with the columns shown in Figure 2.1, we are ready to fit curves to

each of the subjects. For this analysis we will fit data to an exponential curve of the form

$$f(t|\theta) = x_0 e^{tk} \tag{2.8}$$

where $\theta = [x_0, k]$. This form is specified in the `expCurve()` provided by the `bdots` package.

```
mouse_fit <- bfit(data = mouse, subject = "ID", time = "Day",
                  y = "Volume", group = "Treatment", curveType = expCurve())
```

Having successfully fit curves to our data, we now consider the return object and provided summaries.

Return Object and Generics

The function `bfit` returns an object of class `bdotsObj`, inheriting from class `data.table`. As such, each row uniquely identifies one combination of subject and group values. Included in this row are the subject identifier, group classification, summary statistics regarding the curves, and a nested `gnls` object. Inheriting from `data.table` also permits us to use `data.table` syntax to subset the object as in Figure 2.5 where we elect to only plot the first four subjects.

```
> class(mouse_fit)
[1] "bdotsObj"   "data.table" "data.frame"

> head(mouse_fit)
   ID Treatment        fit      R2   AR1 fitCode
1:  1         A <gnls[18]> 0.97349 FALSE       3
2:  2         B <gnls[18]> 0.83620 FALSE       4
3:  3         E <gnls[18]> 0.96249 FALSE       3
4:  4         C <gnls[18]> 0.96720 FALSE       3
5:  5         D <gnls[18]> 0.76156 FALSE       5
6:  7         B <gnls[18]> 0.96361 FALSE       3
```

Figure 2.3: A `bfit` object inheriting from `data.frame`

The number of columns will depend on the total number of groups specified, with the subject and group identifiers always being the first columns. Following this is the `fit` column, which

contains the fitted object returned from `gnls`, as well as `R2` indicating the R^2 statistic. The `AR1` column indicates whether or not the observed data was able to be fit with an AR(1) error assumption. Finally, there is the `fitCode` column, which provides a numerical summary on the quality of fits.

Several methods exist for this object, including `plot`, `summary`, and `coef`, returning a matrix of fitted coefficients obtained from `gnls`.

Fitting Diagnostics

The `bdots` package was originally introduced to address a very narrow scope of problems, and the `fitCode` designation is an artifact of this original intent. Specifically, it assumed that all of the observed data was of the form given in Equation 2.1 where the observed time series was dense and the errors were autocorrelated. Autocorrelated errors can be specified in the `gnls` package (used internally by `bdots`) when generating subject fits, though there were times when the fitter would be incapable of converging on a solution. In that instance, the autocorrelation assumption was dropped and constructing a fit was reattempted.

R^2 proved a reliable metric for this kind of data, and preference was given to fits with an autocorrelated error structure over those without. From this, the hierarchy given in Table 2.1 was born. `fitCode` is a numeric summary statistic ranked from 0 to 6 detailing information about the quality of the fitted curve, constructed with the following pseudo-code:

```
AR1 <- # boolean, determines AR1 status of fit
fitCode <- 3*(!AR1) + 1*(R2 < 0.95)*(R2 > 0.8) + 2*(R2 < 0.8)
```

A fit code of 6 indicates that `gnls` was unable to successfully fit the subject's data.

`bdots` today stands to accommodate a far broader range of data for which the original `fitCode` standard may no longer be relevant. The presence of autocorrelation cannot always be assumed, and users may opt for a metric other than R^2 for assessing the quality of the fits. Even the assessments of fits on a discretized scale may be something of only passing interest. Even then, however, this is how the current implementation of `bdots` categorizes the quality of its fits, with the creation of greater flexibility in this regard being a priority for future directions.

`fitCode`	AR(1)	R^2
0	TRUE	$R^2 > 0.95$
1	TRUE	$0.8 < R^2 < 0.95$
2	TRUE	$R^2 < 0.8$
3	FALSE	$R^2 > 0.95$
4	FALSE	$0.8 < R^2 < 0.95$
5	FALSE	$R^2 < 0.8$
6	NA	NA

Table 2.1: Description of the `fitCode` statistic

While the fit code offers a simple diagnostic for assessing quality of individual subjects, it will often be useful to consider broader summaries for reporting on the quality of fits for groups as a whole. This is done most simply using the `summary` and `plot` functions.

Summaries and Plots

Users are able to quickly summarize the quality of the fits with the `summary` method now provided. For example, we may consider a summary of the fitted mouse data:

```
> summary(mouse_fit)

bdotsFit Summary

Curve Type: expCurve
Formula: Volume ~ x0 * exp(Day * k)
Time Range: (0, 106) [31 points]

Treatment: A
Num Obs:  10
Parameter Values:
        x0          k
172.232953   0.056843
########################################
############### FITS ###################
########################################
AR1,       0.95 <= R2        -- 2
AR1,       0.80 < R2 <= 0.95 -- 1
AR1,       R2 < 0.8          -- 0
Non-AR1,   0.95 <= R2        -- 0
Non-AR1,   0.8 < R2 <= 0.95  -- 3
Non-AR1,   R2 < 0.8          -- 4
No Fit                       -- 0

[...]

All Fits
Num Obs:  42
Parameter Values:
        x0          k
102.487118   0.053662
########################################
############### FITS ###################
########################################
AR1,       0.95 <= R2        -- 4
AR1,       0.80 < R2 <= 0.95 -- 2
AR1,       R2 < 0.8          -- 0
Non-AR1,   0.95 <= R2        -- 9
Non-AR1,   0.8 < R2 <= 0.95  -- 16
Non-AR1,   R2 < 0.8          -- 11
No Fit                       -- 0
```

Figure 2.4: Abridged output from the summary function. Note that this includes data on the formula used, the quality of fits and mean parameter estimates by group, and a summary of all fits combined

It is also recommended that users visually inspect the quality of fits for their subjects, which includes a plot of both the observed and fit data. There are a number of options available in `?plot.bdotsObj`, including the option to fit the plots in base R rather than `ggplot2`. This is especially helpful when looking to quickly assess the quality of fits as `ggplot2` can be notoriously slow with large data sets. Figure 2.5 includes a plot of the first four fitted subjects.

```
plot(mouse_fit[1:4, ])
```

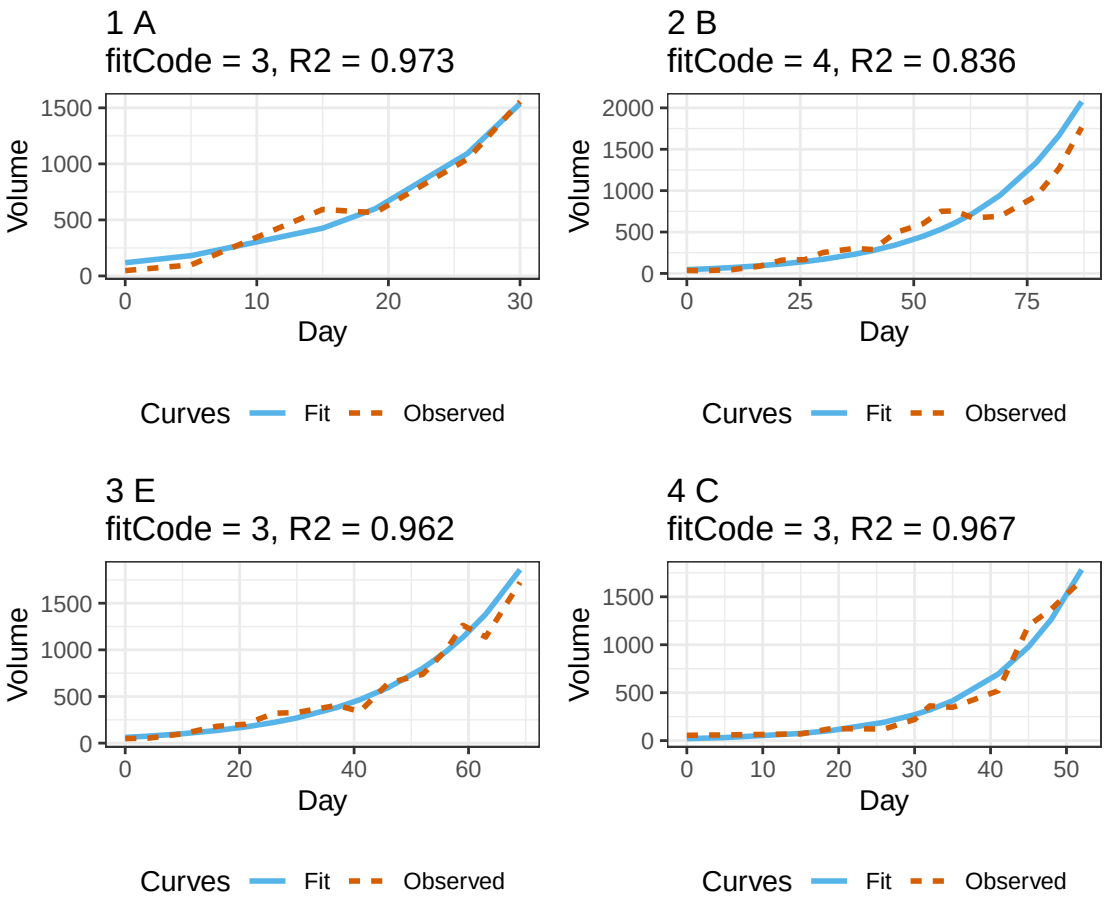


Figure 2.5: Plot of `mouse_fit` using `data.table` syntax to subset to only the first four observations

`bdots` provides now an interactive refitting function, `brefit`, which provides a number of

options to help users recalibrate low quality fits. Details on this function and how it is used are provided in Section 2.5.

2.4 Identification of Group Differences

Having satisfactorily fit subject-specific parametric curves to the observed data, we are ready to begin estimating the group distributions and investigating temporal differences. This section details the function for doing so, `bboot`, along with the introduction of a formula syntax that is new and unique to `bdots`. As before, we will follow each of our descriptions with an illustration of use with the mouse tumor data. Following this, we summarize new generics that are available to the object returned by `bboot`, including those for the `summary` and `plot` functions.

The `bboot` Function

This is done with the bootstrapping[2] function, `bboot`. The number of options included in the `bboot` function have expanded to include a new formula syntax for specifying the analysis of interest as well as to include options for permutation testing. A call to `bboot` takes the following form:

```
bboot(formula, bdObj, B, alpha, permutation = TRUE, padj = "oleson", ...)
```

The `formula` argument is new to `bdots` and will be discussed in the next section. As for the remaining arguments, `bdObj` is simply the object returned from `bfit` that we wish to investigate, and `B` serves the dual role of indicating the number of bootstraps/permutations we wish to perform; `alpha` is the rate at which we wish to control the FWER. `permutation` and `padj` work in contrast to one another: when `permutation = TRUE`, the argument to `padj` is ignored. Otherwise, `padj` indicates the method to be used in adjusting the nominal `alpha` to control the FWER. By default, `padj = "oleson"`. Finally, as previously mentioned, there is no longer a need to specify if the groups are paired, and `bboot` determines this automatically based on the subject identifiers in each

[2] Although both bootstrapping and permutation testing are available for statistically testing temporal differences, bootstrapping is still used for the construction of confidences intervals given in the `plot` function, hence the description as a "bootstrapping" function

of the groups.

Formula

As the `bfit` function is now able to create fits for an arbitrary number of groups at once, we rely on a formula syntax in `bboot` to specify precisely which groups differences we wish to compare. Let `y` designate the outcome variable indicated in the `bfit` function and let `group` be one of the group column names to which our functions were fit. Further, let `val1` and `val2` be two values within the `group` column. The general syntax for the `bboot` function takes the following form:

```
y ~ group(val1, val2)
```

Note that this is an *expression* in R and is written without quotation marks. To give a more concrete example, suppose we wished to compare the difference in tumor growth curves for A and B from the `Treatment` column in our mouse data (Figure 2.1). We would do so with the following syntax:

```
Volume ~ Treatment(A, B)
```

There are two special cases to consider when writing this syntax. The first is the situation that arises in the case of multiple or nested groups, the second when a difference of difference analysis is conducted. We will describe these in the next section, though it is important to note here that the mouse data being used to provide illustration to the package use does not naturally accommodate either of these extensions. As such, we will begin by briefly introducing a mock data structure and then using it to illustrate the extensions of the syntax.

Formula Syntax for Nested Groups and the Difference of Differences

The formula syntax introduced in Section 2.4 is straightforward enough in the case in which we are interested in comparing two groups within a single category, as is the case when we compare two treatment groups, both within the `Treatment` column. As `bdots` now allows multiple groups to be fit at once, there may be situations in which we need more precision in specifying what exactly we wish to compare. Consider for example an artificial dataset that contains some outcome `y` for a collection of vehicles, consisting of eight distinct groups, nested in order of vehicle origin (foreign

or domestic), vehicle class (car or truck), and vehicle color (red or blue). A table detailing the relationship of the groups is given in Table 2.2.

Origin	Class	Color
foreign	car	red
		blue
	truck	red
		blue
domestic	car	red
		blue
	truck	red
		blue

Table 2.2: Example of nested vehicle classes

Beginning with a simple case, suppose we want to investigate the difference in outcome between all foreign and domestic vehicles. Notionally, we would write

```
y ~ Origin(foreign, domestic)
```

just as we did in the mouse data example: here, the name of the group variable `Origin`, followed by the values we are interested in comparing, `domestic` and `foreign`. Alternatively, if we wanted to limit our investigation to only foreign and domestic *trucks*, we would do this by including an extra term specifying the group and the desired value. In this case:

```
y ~ Origin(foreign, domestic) + Class(truck).
```

Similarly, to compare only foreign and domestic *red* trucks, we would add an additional term for color:

```
y ~ Origin(foreign, domestic) + Class(truck) + Color(red)
```

There are also instances in which we might be considered in the interaction between two groups. Although there is no native way to handle interactions in `bdots`, this can be done indirectly through the difference of differences [**?**]. To illustrate, suppose we are interested in understanding

how the color of the vehicle differentially impacts outcome based on the vehicle class. In such a case, we might look at the difference in outcome between red cars and red trucks and then compare this against the difference between blue cars and blue trucks. Any difference between these two differences would give information regarding the differential impact of color between each of the two classes. This is done in `bdots` using the `diffs` syntax in the formula:

```
diffs(y, Class(car, truck)) ~ Color(red, blue)
```

Here, the *outcome* that we are considering is the difference between vehicle classes, with the groups we are interested in comparing being color. This is helpful in remembering which term goes on the left hand side of the formula.

Similar as to the case before, if we wanted to limit this difference of differences investigation to only include domestic vehicles, we can do so by including an additional term:

```
diffs(y, Class(car, truck)) ~ Color(red, blue) + Origin(domestic).
```

As before, this can be further subset with an arbitrary number of nested groups.

We now return to a description of the process of estimating group differences with the mouse tumor dataset.

Summary and Analysis

Let's begin first by running `bboot` using bootstrapping to compare the difference in tumor growth between treatment groups A and E in our mouse data using permutations to test for regions of significant difference.

```
mouse_boot <- bboot(Volume ~ Treatment(A, E), bdObj = mouse_fit,
                    permutation = TRUE)
```

This returns an object of class `bdotsBootObj`. A summary method is included to display relevant information:

```
> summary(mouse_boot)

bdotsBoot Summary

Curve Type: expCurve
Formula: Volume ~ x0 * exp(Day * k)
Time Range: (0, 59) [21 points]

Difference of difference: FALSE
Paired t-test: FALSE
Difference: Treatment

FWER adjust method: Permutation
Alpha: 0.05
Significant Intervals:
     [,1] [,2]
[1,]   15   32
```

There are a few components of the summary that are worth identifying when reporting the results. In particular, note the time range provided, an indicator of if the test was paired, and which groups were being considered. The last section of the summary indicates the testing method used, an adjusted `alphastar` if `permutation = FALSE`, and a matrix of regions identified as being significantly different. This matrix is `NULL` is no differences were identified at the specified alpha; otherwise there is one row included for each disjointed region of significant difference.

In addition to the provided summary output, a `plot` method is available, with a list of additional options included in `help(plot.bdotsBootObj)`.

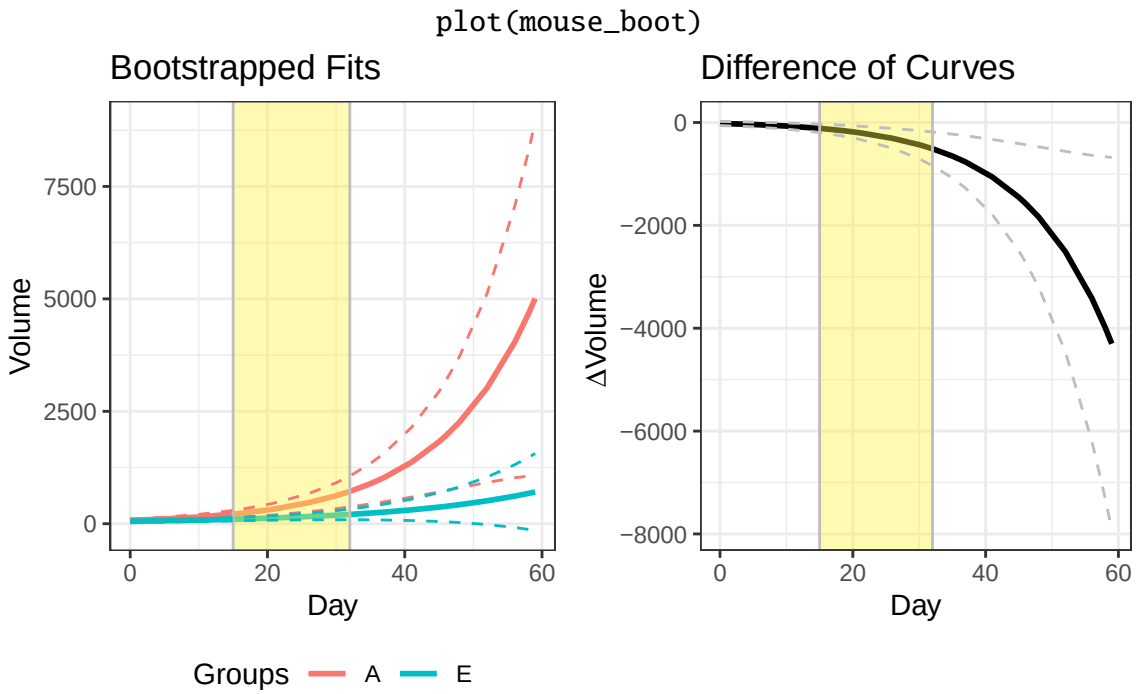


Figure 2.6: Bootstrapped distributions with regions of significant difference determined via permutation testing

2.5 Ancillary Functions

Outside of a standard analysis using the `bdots` package, there are a suite of functions that users may find helpful. Brief descriptions of these additional functions are given here.

2.5.1 Refitting

The nonlinear curve fitting algorithm used by `nlme::gnls` in `bfit` can be sensitive to the starting parameters. Sensible starting parameters are computed from the observed data as part of the curve fitting functions (i.e., within the `logistic()` function), though these can often be improved upon. The quality of the fits can often be evidenced by the `fitCode` or via visual inspections of the fitted functions against the observed data. There are sometimes, however, when the quality of fits are poor. When this occurs, there are several options available to the user, all of which are provided through the function `brefit`. `brefit` takes the following arguments:

```
brefit(bdObj, fitCode = 1L, subset = NULL, quickRefit = FALSE, paramDT = NULL)
```

The first of these arguments outside of the `bdObj` is `fitCode`, indicating the minimum fit code to be included in the refitting process. As discussed in Section 2.3, this can be a sub-optimal way to specify data to subset. To add flexibility to which subjects are fit there is now the `subset` argument taking either a logical expression, a collection of indices that would be used to subset an object of class `data.table`, or a numeric vector with indices that the user wishes to refit. For example, we could elect to refit only the first 10 subjects or refit subjects with $R^2 < 0.9$:

```
refit <- brefit(fit, subset = 1:10) # refit the first 10 subjects
refit <- brefit(fit, subset = R2 < 0.9) # refit subjects with R2 < 0.9
```

When an argument is passed to subset, the `fitCode` argument is completely ignored.

To assist with the refitting process is the argument `quickRefit`. When set to `TRUE`, `brefit` will take the average coefficients of accepted fits within a group and use those as new starting parameters for poor fits. The new fits will be retained if they have a larger R^2 value by default. When set to `quickRefit = FALSE`, the user will be guided through a set of prompts to refit each of the curves manually.

Finally, the `paramDT` argument allows for a `data.table` with columns for subject, group identifiers, and parameters to be passed in as a new set of starting parameters. This `data.table` requires the same format as that returned by `bdots::coefWriteout`. The use of this functionality is covered in more detail in the `bdots` vignettes and is a useful way for reproducing a `bdotsObj` from a plain text file.

When `quickRefit = FALSE`, the user is put through a series of prompts along with a series of diagnostics for each of the subjects to be refit. Here, for example, is the option to refit subject 11 from the mouse data:

```
Subject: 11
R2: 0.837
AR1: FALSE
rho: 0.9
fitCode: 4

 Model Parameters:
       x0         k
53.186497  0.051749

Actions:
1) Keep original fit
2) Jitter parameters
3) Adjust starting parameters manually
4) Remove AR1 assumption
5) See original fit metrics
6) Delete subject
99) Save and exit refitter
Choose (1-6):
```

There are a number of options provided in this list. The first, of course, keeps the original fit of the presented subject and moves on to the next subject in the list. The second option takes the values of the fitted parameter and "jitters" them, changing each of the values by a prespecified magnitude. Given the sensitivity of `nlme::gnls` to starting parameters, this is sometimes enough for the fitter to converge on a better fit for the observed data. Alternatively, the third option gives the user the ability to select the starting parameters manually. The third option gives the user the ability to attempt refitting the observed data without an AR(1) error assumption, though this is only relevant if such an assumption exists. Option (5) reprints summary information and option (6) allows the user to delete the subject all together.

When any attempt to refit the observed under new conditions is presented (options (2)-(4)), a plot is rendered comparing the original fit side-by-side with the new alternative, Figure 2.7.

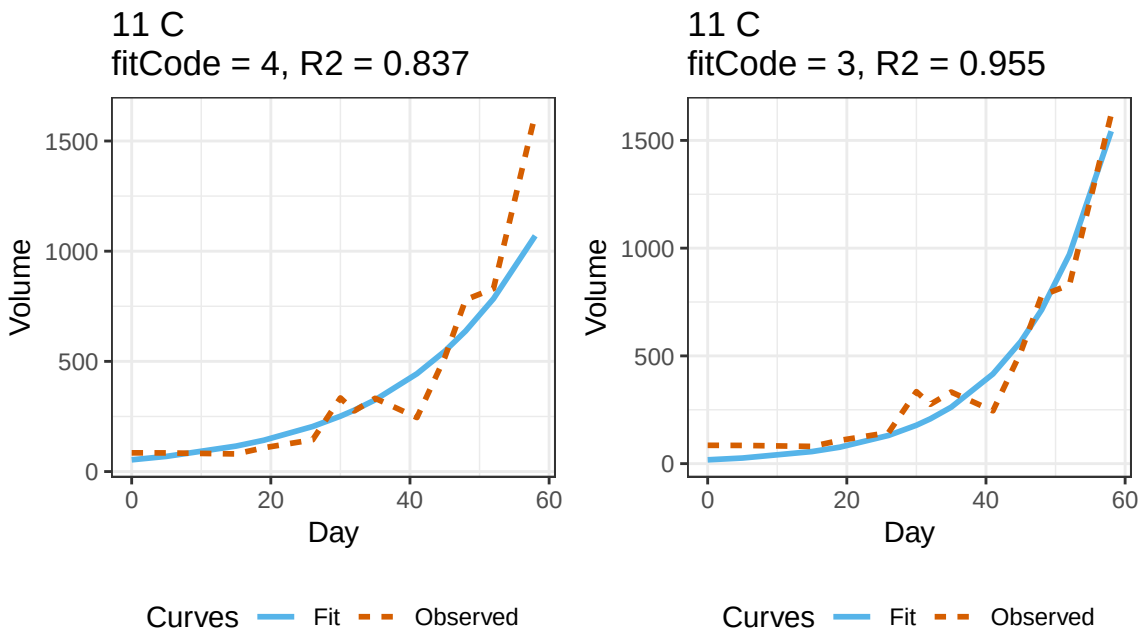


Figure 2.7: before and after refit

As the menu item suggests, users have the ability to end the manually refitting process early and save where they had left off. To retain previously refit items and start again at a later time, pass the first refitted object back into the refitter as such:

```
refit <- brefit(fit, ...)
refit <- brefit(refit, ...) # pass in the refitted object
```

A final note should be said regarding the option to delete a subject. As `bdots` now automatically determines if subjects are paired based on subject identifiers (necessary for calculations in significance testing), it is critical that if a subject has a poor fit in one group and must be removed that they are also removed from all additional groups in order to retain paired status. This can be overwritten with a final prompt in the `brefit` function before they are removed. The removal of subjects can also be done with the ancillary function, `bdRemove`, useful for removing subjects without undergoing the entire refitting process. See `help(bdRemove)` for details.

2.5.2 Correlations

There are sometimes cases in which we are interested in determining the correlation of a fixed attribute with group outcome responses across time . This can be done with the `bcorr` function (previously `bdotsCorr`), which takes as an argument an object of class `bdotsObj` as well as a character vector representing a column from the original dataset used in `bfit`

```
bcorr(fit, "value", ciBands, method = "pearson")
```

See the vignettes included in the appendix for a more detailed example of how this function is used.

2.5.3 α Adjustment

There may also be situations in which users wish to make an adjustment to autorcorrelated test statistics using the modified Bonferonni adjustment provided in **?**, though in a different context than what is done in `bdots`. To facilitate this, we introduce an extension to the `p.adjust` function, `p_adjust`, identical to `p.adjust` except that it accepts method `"oleson"` and takes additional arguments `rho`, and `df`. `rho` determines the autocorrelation estimate for the oleson adjustment while `df` returns the degrees of freedom used to compute the original vector of t-statistics. If an estimate of `rho` isn't available, one can be computed on a vector of t-statistics using the `ar1Solver` function in `bdots`:

```
t       <- diffinv(rnorm(5))
rho     <- ar1Solver(t)
unadj_p <- pt(t, df = 10)
adj_p   <- p_adjust(unadj_p, method = "oleson",
                    df = 10, rho = rho, alpha = 0.05)
```

The `p_adjust` function returns both adjusted p-values, which can be compared against the specified alpha (in this case, 0.05) along with an estimate of alphastar, a nominal alpha at which one can compare the original p-values:

```
> unadj_p
[1] 0.5000000 0.0849965 0.0381715 0.1601033 0.0247453 0.0013016
> adj_p
[1] 0.9201915 0.1564261 0.0702501 0.2946514 0.0455408 0.0023954
attr(,"alphastar")
[1] 0.027168
```

Here, for example, we see that the last two positions of `unadj_p` have values less than `alphastar`, identifying them as significant; alternatively, we see these same two indices in `adj_p` significant when compared to `alpha = 0.05`

2.6 Discussion

The original implementation of `bdots` set out to address a narrow set of problems. Previous solutions beget new opportunities, however, and it is in this space that the second iteration of `bdots` has sought to expand. Since then, the interface between user and application has been significantly revamped, creating an intuitive, reproducible workflow that is able to quickly and simply address a broader range of problems. The underlying methodology has also been improved and expanded upon, offering better control of the family-wise error rate.

While significant improvements have been made, there is room for further expansion. The most obvious of these is the need to include support for non-parametric functions, the utility of which cannot be overstated. Not only would this alleviate the need for the researcher to specify in advance a functional form for the data, it would implicitly accommodate more heterogeneity of functional forms within a group. Along with this, the current implementation is also limited in the quality-of-fit statistics used in the fitting steps to assess performance. R^2 and the presence of autocorrelation are relevant to only a subset of the types of data that can be fit, and allowing users more flexibility in specifying this metric is an active goal for future work. In all, future directions of this package will be primarily focused on user interface, non-parametric functions, and greater flexibility in defining metrics for fitted objects.

Appendix

2.7 Writing Custom Curve Functions

One of the most significant changes in the newest version of `bdots` is the ability to specify the parametric curve independently of the fitting function. Not only does this simplify a typical analysis, reducing all fitting operations to the single function `bfit`, it also provides users with a way to modify this function to meet their own needs. In this section we will detail how the curve function is used in `bdots` and how users can write their own. Finally, we will conclude with an example of how this was used in Chapter 3 to create adequate fits with the look onset method when the typical method for constructing estimated starting parameters based on the proportion of fixation method failed.

To begin, it is important to understand a little of how `bdots` works internally. In the curve fitting steps using `bfit`, the data is split by subject and group, creating a list whereby each element is the set of all observations for a single subject (i.e., in a paired setting, an individual subject would have two separate elements in this list). Ultimately, the data in each element will be used to construct a set of estimated parameters and standard errors for each subject provided by the function `lmer::gnls`. Doing so requires both (1) a formula to which we fit the data and (2) starting parameter estimates. Providing the both of these is the role of the curve function. Figure 2.8 provides an illustration of this process.

We turn our attention now to the curve function itself, using as an example a curve function for sitting a straight line to the observed data, given in Figure 2.9. In describing the purpose for each, we also include enough detail so that this may be used as a template in constructing a new one all together. We do this in an enumerated list, each item corresponding to the numbered portion in Figure 2.9.

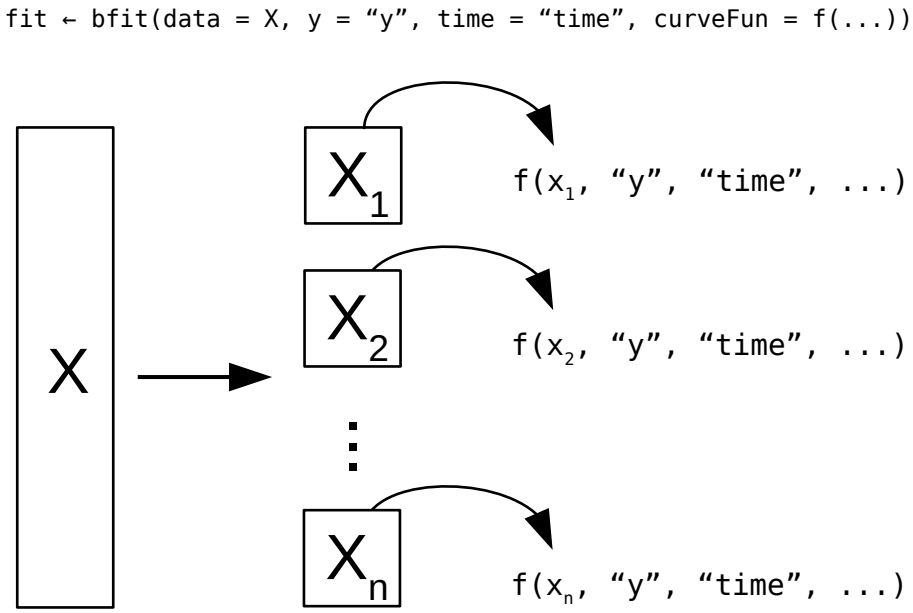


Figure 2.8: A call to the function `bfit` with data `X` and outcome and time variables `"y"` and `"time"`. `bfit` splits the dataset `X` by subject/group and passes each individual `data.frame` into the curve function `f()`, along with time and outcome character vectors as well as any other arguments passed into `‘...‘`. In particular, the `‘...‘` argument allows the user to specify characteristics of the curve function that apply to all instances, as would be the case, for example, if `curveFun = polynomial(degree = 5)`. Finally, each instance of `f(...)` returns both a formula for `lmer::gnls` as well as subject-specific starting parameters.

```
① linear <- function (dat, y, time, params = NULL, ...) {

      linearPars <- function(dat, y, time) {
          time <- dat[[time]]
②         y <- dat[[y]]
          if (var(y) == 0) {
              return(NULL)
          }
          mm <- (max(y) - min(y))/max(time)
          bb <- mean(y) - mm * mean(time)
          return(c(intercept = bb, slope = mm))
      }

③     if (is.null(params)) {
          params <- linearPars(dat, y, time)
      }
④     if (is.null(params)) {
          return(NULL)
      }
      y <- str2lang(y)
      time <- str2lang(time)
⑤     ff <- bquote(.(y) ~ slope * .(time) + intercept)
      attr(ff, "parnames") <- names(params)
      return(list(formula = ff, params = params))
  }
```

Figure 2.9: An example curve function with its constituent parts

1. The first part of the curve function is the collection of arguments to be passed, also known as formals. Each curve function should have an argument `dat`, which takes a `data.frame` as described in Figure 2.8, as well as arguments `y` and `time` which will take character strings indicating which columns of `dat` represent the outcome and time variables, respectively. Following this is the prespecified argument `params = NULL`, which is used by `bdots` during the refitting process, where the estimated starting parameters for the function are retrieved from outside the curve fitting function. During the initial fitting process, however, these parameters are generally constructed from the observed data. The only exception to this would be if the user decided to specify the initial starting parameters for *all* subjects when calling `bfit`, as in the call

```
fit <- bfit(dat, "y", "time", curveFun = linear(intercept = 0, slope = 1).
```

This should not be common. Following the `params` argument, any other arguments specific to the curve function could be included. Although there are none for `linear`, an example of when they might be used would be for `polynomial`, in which the degree of the polynomial to be fit would be included. Finally, there is the `...` argument, which is needed to accommodate the passing of any additional arguments from `bfit` that are not a part of the curve function. Generally, this is not needed by the users but should be included nonetheless.

2. Also included in a curve function is a second function to estimate starting parameters from the observed data. While not strictly necessary that it be included *within* the curve function, it is useful for keeping the curve function self contained; parameter estimating functions defined outside of the curve function will otherwise still be used if they exist in the users calling environment. For estimating starting parameters for a linear function we see here the function `linearPars`, taking as its arguments `dat`, `y`, and `time`. In this example, we check in case `var(y) == 0`, which causes issues for `lmer::gnls`, though in general it is a good idea to check for any other potential issues when estimating starting parameters (negative values for a logistic, for example). Importantly, this function returns a named vector, with the names of the parameters needing to match the parameter names in the formula given in (5).

3. As detailed in (1), with the argument `params = NULL`, the curve function should begin by estimating starting parameters. When different parameters are passed into `params`, this is skipped.

4. This is a quick check on the result from (3). Had `linearPars` returned a `NULL` object, the curve function itself should return a `NULL` object so that it is not passed to the fitter.

5. Finally, we have the most intricate part of the curve function, which is the construction of the formula object to be used by `lmer::gnls`. The first two lines of this use the base R function `str2lang` which turns a character string into an R language object (specifically, an unevaluated expression), making the names of the outcome and time variable suitable for a

formula. The next line using the base R function `bquote`. The function `quote` returns its argument exactly as it was passed as an unevaluated expression; `bquote` does the same but first substituting any of its elements wrapped in `.()`. As it is written here, this will return a formula object using `slope` and `intercept` as is, but while replacing `.(y)` and `.(time)` with the appropriate names based on the column names in `dat`. Finally, the names of the parameters are included as attributes to the formula object and the curve function concludes by returning a named list including both the formula object, as well as the named vector of parameters.

The object returned by the curve function is not limited to just providing starting parameters for observed data; the formula itself is converted by `bdots` into a function proper, capable of evaluating and bootstrapping values from that function in `bboot`. And so long as a user is able to recreate the steps provided, they should be able to construct any sort of nonlinear function to be fit to their data, even if it is not included in `bdots`.

While there is obvious utility in being able to specify *new* curves for `bdots` to fit, we describe a case here in which the flexibility of the curve function was used to recreate the `doubleGauss()` function for use with our simulated data. In short, Chapter 3 details a proposed method for fitting data in the Visual World Paradigm relying *not* on a densely sampled function in time, but rather as a collection of unordered binary observations. When the `doubleGauss` function was originally introduced to `bdots`, the empirically observed data was a relatively close match for its parametric form:

$$f(t|\theta) = \begin{cases} \exp\left(\frac{(t-\mu)^2}{-2\sigma_1^2}\right)(p - b_1) + b_1 & \text{if } t \leq \mu \\ \exp\left(\frac{(t-\mu)^2}{-2\sigma_2^2}\right)(p - b_2) + b_2 & \text{if } t > \mu \end{cases} \tag{2.9}$$

An example of how the observed data matched the proposed functional form is taken from an example given in Figure 2.10.

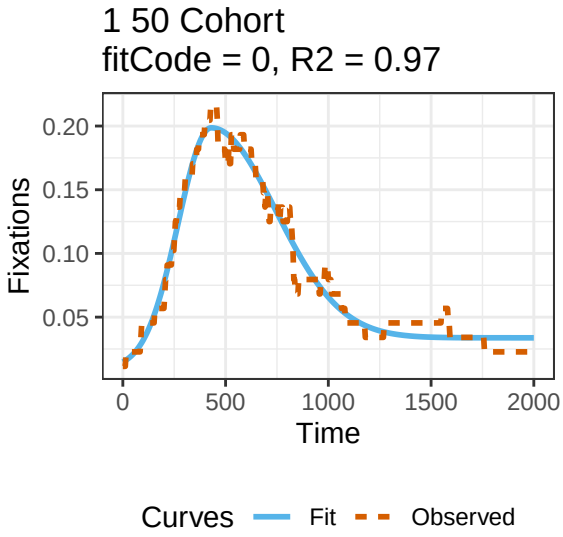


Figure 2.10: Observed data matching the parametric form of the asymmetrical Gaussian

Accordingly, an appropriate function for estimating the starting parameters took the form given in Figure 2.11.

```
dGaussPars <- function (dat, y, time) {
    time <- dat[[time]]
    y <- dat[[y]]
    if (var(y) == 0) {
        return(NULL)
    }
    mu <- time[which.max(y)]
    ht <- max(y)
    base1 <- min(y[time < mu])
    base2 <- min(y[time > mu])
    y1 <- y - base1
    y1 <- rev(y1[time <= mu])
    time1 <- rev(time[time <= mu])
    totalY1 <- sum(y1)
    sigma1 <- mu - time1[which.min(abs((pnorm(1) - pnorm(-1)) *
        totalY1 - cumsum(y1)))]
    y2 <- y - base2
    y2 <- y2[time >= mu]
    time2 <- time[time >= mu]
    totalY2 <- sum(y2)
    sigma2 <- time2[which.min(abs((pnorm(1) - pnorm(-1)) * totalY2 -
        cumsum(y2)))] - mu
    return(c(mu = mu, ht = ht, sig1 = sigma1, sig2 = sigma2,
        base1 = base1, base2 = base2))
}
```

Figure 2.11: Estimate starting parameters for empirical asymmetric Gaussian

This is appropriate, for example, when considering that the `mu` parameter is estimated by finding *when* the observed data is at its peak, while the `ht` parameter is found to be the peak itself. In the case of the look onset method, however, data consists of non-ordered binary observations $\{0, 1\}$ in time, making estimates for even these two parameters completely unreliable. An immediate consequence of this was that in a simulation of 1,000 subjects fit with asymmetric Gaussian data was conducted, less than half were able to return adequate fits from `bdots`, proving problematic for any kind of systematic analysis in assessing the proposed method.

The solution was to provide a new curve function utilizing a different internal function for estimating starting parameters. The particular interest in presenting this here is how broad of a solution this may be to any number of problems: rather than attempting to construct parameters

directly from the data (which may be misbehaved), we provide a reasonable distribution of starting parameters, drawing any number of samples from it, and retaining those which best fit the observed data:

```
dgaussPars_dist <- function(dat, y, time, startSamp = 8) {
        time <- dat[[time]]
        y <- dat[[y]]

        if (var(y) == 0) {
            return(NULL)
        }

        spars <- data.table(param = c("mu", "ht", "sig1", "sig2", "base1", "base2"),
                            mean = c(630, 0.18, 130, 250, 0.05, 0.05),
                            sd = c(77, 0.05, 30, 120, 0.015, 0.015),
                            min = c(300, 0.05, 50, 50, 0, 0),
                            max = c(1300, 0.35, 250, 400, 0.15, 0.15))

        fn <- function(p, t) {
            lhs <- (t < p[1]) * ((p[2] - p[5]) *
                       exp((t - p[1])^2/(-2 * p[3]^2)) + p[5])
            rhs <- (t >= p[1]) * ((p[2] - p[6]) *
                       exp((t - p[1])^2/(-2 * p[4]^2)) + p[6])
            lhs + rhs
        }
        npars <- vector("list", length = startSamp)
        for (i in seq_len(startSamp)) {
            maxFix <- Inf
            while (maxFix > 1) {
                npars[[i]] <- Inf
                while (any(spars[, npars[[i]] <= min | npars[[i]] >= max])) {
                  npars[[i]] <- spars[, rnorm(length(npars[[i]])) * sd + mean]
                }
                maxFix <- max(fn(npars[[i]], time))
            }
        }
        r2 <- vector("numeric", length = startSamp)
        for (i in seq_len(startSamp)) {
            yhat <- fn(npars[[i]], time)
            r2[i] <- mean((y - yhat)^2)
        }
        finalPars <- npars[[which.min(r2)]]
        names(finalPars) <- c("mu", "ht", "sig1", "sig2", "base1", "base2")
        return(finalPars)
    }
```

Figure 2.12: Using random distribution to estimate starting parameters

By utilizing an existing fitting function while substituting the method by which the starting parameters are estimated, we were able to go from recovering less than half of the simulated starting parameters successfully to well over 80%.

CHAPTER 3
THE LOOK ONSET METHOD

3.1 Introduction

Spoken words create analog signals that are processed by the brain in real time. That is, as spoken word unfolds, a collection of candidate words are considered until the target word is recognized [**?**]. The degree to which a particular candidate word is being considered is known as activation. An important part of this process involves not only correctly identifying the word but also eliminating competitors. For example, we might consider a discrete unfolding of the word "elephant" as "el-e-phant". At the onset of "el", a listener may activate a cohort of potential resolutions such as "elephant", "electricity", or "elder", all of which may be considered competitors. With the subsequent "el-e", words consistent with the received signal, such as "elephant" and "electricity" remain active competitors, while incompatible words, such as "elder", are eliminated. Such is a rough description of this process, continuing until the ambiguity is resolved and a single word remains.

Our interest is in measuring the degree of activation of a target word, relative to competitors. Activation, however, is not measured directly, and we instead rely on what can be observed with physiological behavior. Although there are a number of relevant indices of lexical activation [**?**], we concern ourselves here with eye tracking data collected in the context of the Visual World Paradigm (VWP) [**?**], an experimental model in which a participant's eye movements are tracked as they respond to spoken language. Recently, however, researchers have begun to reexamine some of the underlying assumptions associated with the VWP, calling into question the validity or interpretation of current methods ([**?**], [**?**]). We present here a brief history of word recognition in the context of the VWP, along with an examination of contemporary concerns. In response, we then propose a new generative model for understanding eye mechanics, accompanied by an alternative method for relating eye-tracking data to lexical activation. We follow this with a series of simulations, both in the context of recovering the trajectory of eye mechanics for individual subjects,

as well as for identifying differences in trajectories between groups of individuals. Finally, we demonstrate the performance of our model with an application to data collected in prior studies.

Visual World Paradigm

The Visual World Paradigm was first introduced in 1995, marking the initial link between the mental processes associated with language comprehension and eye movements [**?**]. A typical experiment in the VWP involves situating a subject in front of a "visual world", commonly a computer screen today, and asking them to identify and select an object corresponding to a spoken word. The initiation of eye movements and subsequent fixations are recorded as this process unfolds, with the location of the participants' eyes serving as a proxy for which words or images are being considered. This association was first demonstrated by comparing how the mean time to initiate an eye movement to the correct object was mediated by the presence of phonological competitors ("candy" and "candle", sharing auditory signal at word onset) and situations containing syntactic ambiguity ("Put the apple on the towel in the box" and "Put the apple *that's* on the towel in the box" in ambiguous scenarios with one or more apples). It is by comparing the trajectory of these mechanics across trials in the presence of auditory or semantic competitors that researchers have used the VWP in their investigation of spoken word recognition. For a review of the Visual World Paradigm and its applications, see **?**, **?**.

Proportion of fixation

The interactive activation model of word recognition was first introduced in **?**, positing that word recognition takes place in a system with multiple interacting hierarchies of processing, including those made up of features, phonemes, and words. This model was later instantiated as a computer model known as TRACE [**?**]. Briefly, TRACE is structured as a hierarchical network model in which input is fed into the system (typically as a string of phonemes) with potential resolutions (here, words) being represented as nodes. The TRACE model processes this input over time and in parallel, with activation incrementally cascading from features (voicing, aspiration, etc.,) to phonemes to words. Multiple words can be activated at once, graded by consistency of the word with the received signal. Finally, the TRACE model exhibits competition whereby the activation

of a particular word functions to inhibit its competitors. This entire process continues as input is received until a single word is most active. An example of this is demonstrated in Figure 3.1 using the current implementation of TRACE, jTRACE [**?**], in which the word "party" was provided as input as a string of phonetic characters. Figure 3.1 shows the activation of the target word, along with a number of competitors.

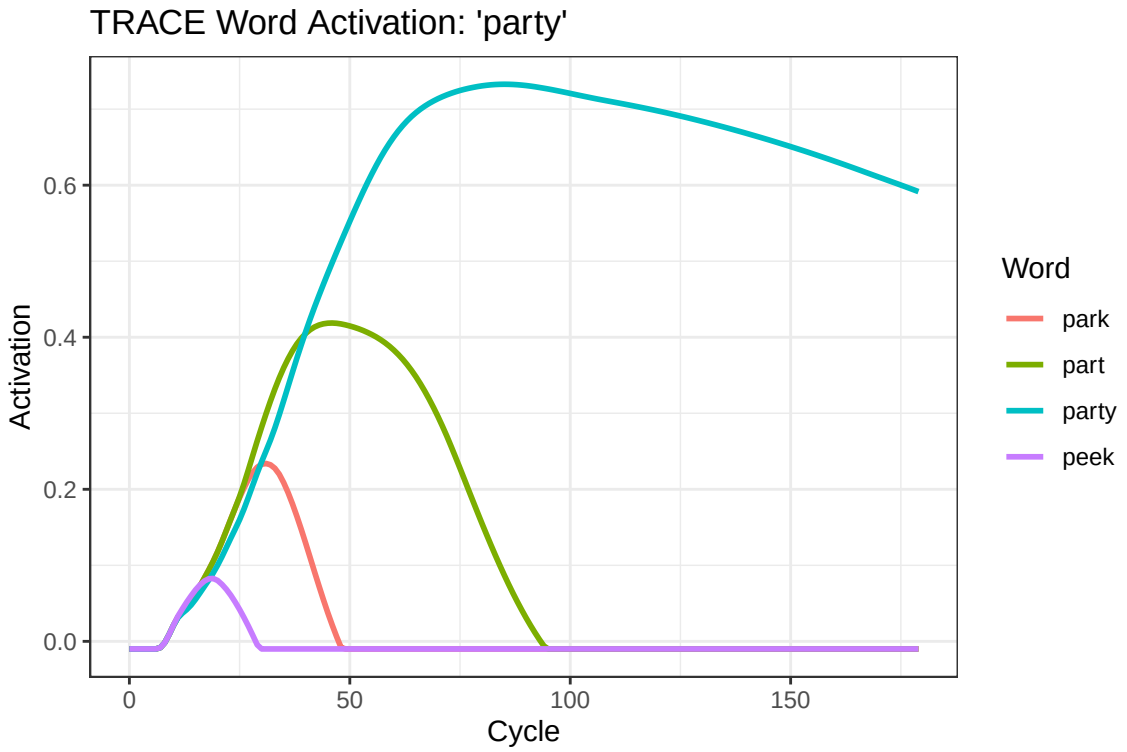


Figure 3.1: TRACE activation of the word "party", with competitors

It was against simulated TRACE data that **?** found a tractable way of analyzing eye-tracking data collected in the VWP. During each trial, the location of an eye fixation was recorded at intervals of 33ms and was marked as either a 0 or 1 as an indicator of whether or not the subject was fixated on the referent or not. By taking the average of fixations towards a referent at each time point, Allopenna was able to create a "fixation proportion" curve that largely reflected the shape and competitive dynamics of word activation suggested by TRACE, both for the target object as well as for competitors, giving researchers a method to both visualize and quantify the time course

of lexical activation. This also provided an empirical grounding for an explicit linking hypothesis, relating activation to eye mechanics (emphasis added):

> We made the general assumption that the probability of initiating an eye movement to fixate on a target object *o* at time *t* is a direct function of the probability that *o* is the target given the speech input and where the probability of fixating *o* is determined by the activation level of its lexical entry relative to the activation of other potential targets. [...] Note that this hypothesis does *not* require stronger and less defensible assumptions about the relationship between eye movements and attention. For example, we are not committed to the assumption that scan patterns in and of themselves reveal underlying cognitive processes. [**?**]

The utility in this linking hypothesis comes from its simplicity, asserting only that the probability of fixating an object increases as the likelihood it has been referred to also increases. This is in contrast to a number of more involved linking hypotheses presented in **?**.

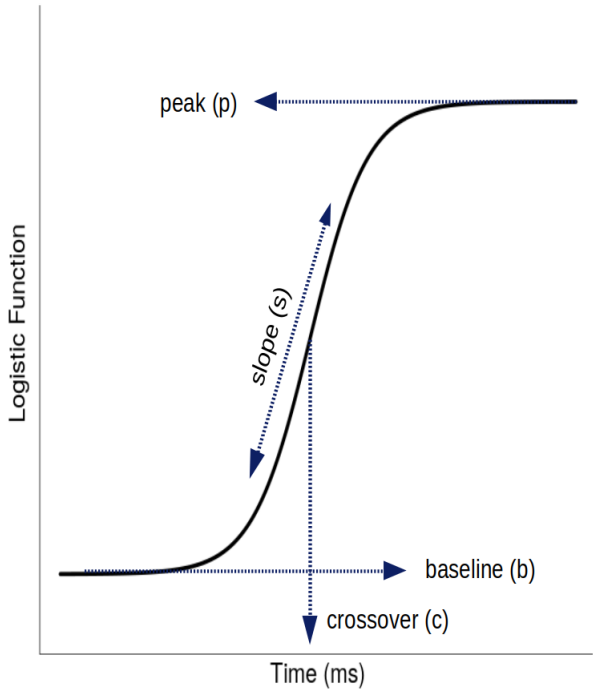


Figure 3.2: An illustration of the four-parameter logistic (Equation 3.4) and its associated parameters, introduced as a parametric function for fixations to target objects

Parametric Methods and Individual Curves

While many studies have addressed qualitative aspects of word recognition such as feedback [**?**], or priming [**?**], few had offered consideration to individual differences in activation. A first attempt was provided by **?** using mixed effects models to capture subject-specific effects by fitting observed data to polynomial functions in time. While this addressed the problem of capturing individual differences, it was burdened by the fact that polynomials are often a poor fit for VWP data, often requiring a high degree to be appropriately fit, resulting in potentially poor asymptotic behavior. Further, polynomial coefficients are often unable to be intuitively mapped onto clinically relevant properties of the functions they are intended to emulate. This was the position taken by **?**, who first introduced non-linear functions to the observed data, equipped with readily interpretable and clinically relevant properties. This includes the four parameter logistic function provided in

Equation 3.4 as well as the asymmetric Gaussian function given in Equation 3.5. Outside of serving to address psycholinguistic concerns relating to individual differences in word recognition, this advancement has been critical in shaping current statistical models used in the context of the VWP. The specification of subject-specific parameters itself implies a distribution of parameters within an experimental group, serving as an impetus for investigating group differences in word activation through the use of bootstrapped differences in time series [**?**] and the subsequent development of the `bdots` software in R for analyzing such differences [**?**].

3.2 Analysis with VWP Data

We now shift focus to the physiological mechanics of eye movements in the context of the Visual World Paradigm, including a mathematical description of how these mechanics relate to the proportion of fixation method first introduced by **?**.

3.2.1 Anatomy of Eye Mechanics

While a number of experimental methods are used as real-time indices of lexical access [**?**], we concern ourselves here with the use of eye tracking, given that "eye movements to objects in the workspace are closely time-locked to referring expressions in the unfolding speech stream, providing a sensitive and non-disruptive measure of spoken language comprehension during continuous speech [**?**]." This process in its entirety is made up of several interrelated components, including activation and the visual-motor system, oculomotor delay, and mechanics of eye movement itself. We briefly introduce each of these components in turn, while providing a visual summary in Figure 3.3.

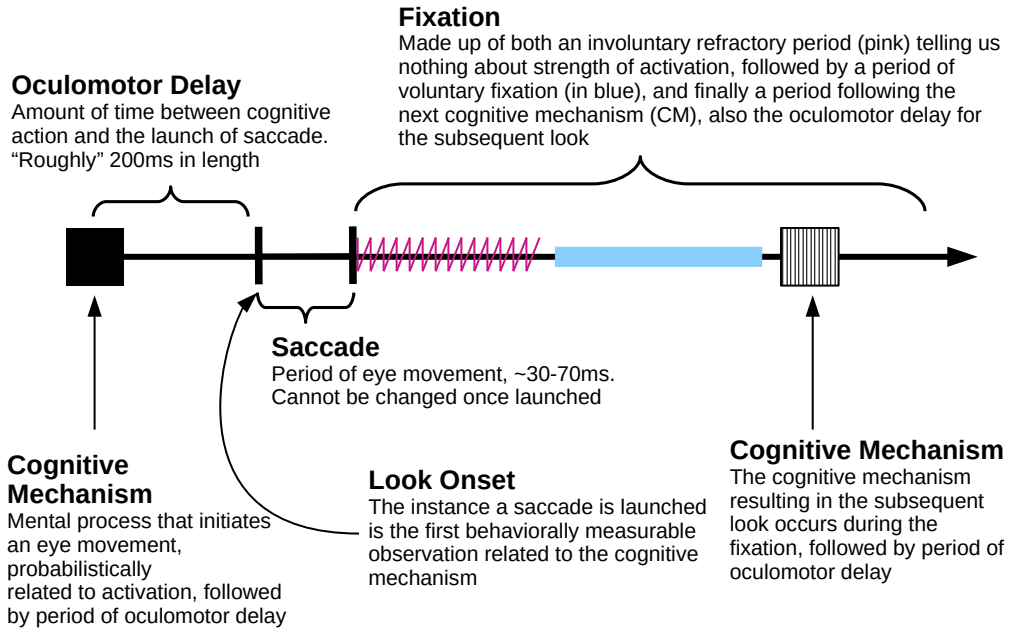


Figure 3.3: Anatomy of a look

3.2.1.1 Saccades and Fixation

Rather than acting in a continuous sweeping motion as our perceived vision might suggest, our eyes themselves move about in a series of short, ballistic movements, followed by brief periods of stagnation. These periods of movement and stagnation, respectively, are the saccades and fixations.

Saccades are short, ballistic movements lasting between 20ms-60ms, during which time we are effectively blind. Once in motion, saccades are unable to change trajectory from their intended destination. Following this movement is a period known as a fixation, itself made up of a necessary refraction period (during which time the eye is incapable of movement) followed by a period of voluntary fixation which may include planning time for deciding the destination of the next eye movement. Additionally, the duration of fixations are typically longer and more variable. It will be

convenient to follow previous convention and consider a saccade followed by its adjacent fixation as a single concept called a "look" [**?**]. We take particular care here to note that the beginning of a look, or "look onset", starts the instance that a previous look ends or, said another way, the instant an eye movement is launched.

The mechanical aspects of eye movements are not themselves what we are interested in. Rather, they serve as an index of the underlying cognitive process, lexical activation, which we discuss next.

3.2.1.2 Activation and Visual-Motor system

The concept of activation, as it relates to the discussion here, arises from the metaphor in which word perception is made up of a network made up of hierarchical levels (letter, phoneme, word, etc.,) acting as an interactive process unfolding in time [**?**]. Under this *interactive activation model*, greater activation is associated with a greater excitatory action for a network node (specifically here a word) resulting from consistency with the received auditory signal. The interactive activation model allows for both excitatory and inhibiting activations, resulting in the "competing" activation curves between candidate resolutions as the auditory signal unfolds. A theoretical example of activation is given in Figure 3.1.

Eye movements themselves are not a direct read-out of activation, and instead are downstream of a much larger process that also involves the visual-motor system, including general stimulus processing and environment scanning [**?**]. It is this peripheral system, in conjunction with lexical activation, that leads to the *probabilistic* association of activation and fixations. Any proposed relation of each of these in detail to saccades and fixations is known as a *linking hypothesis* tying the observed to the theoretical [**?**].

For our purposes here, we take no stronger a position than acknowledging the interplay between these processes and acknowledging their culmination as a "cognitive mechanism" (CM) that ultimately is responsible for initiating the launching of a saccade, a mechanism itself that is probabilistically related to activation.

3.2.1.3 Oculomotor Delay

In between the cognitive mechanism that initiates the decision to launch an eye movement and the movement itself is a period known as oculomotor delay. It is typically estimated to take around 200ms to plan and launch an eye movement [**?**], and this is usually accounted for by subtracting 200ms from any observed behavior. As oculomotor delay is only "roughly" estimated to be around 200ms, we suggest that accounting for randomness will be critical in correctly recovering the cognitive mechanism of interest or at very least in identifying possible sources of bias or error.

3.2.2 Proportion of Fixation Method

We now consider how the aforementioned mechanics relate to the Visual World Paradigm. In a typical implementation of the VWP, a participant is asked to complete a series of trials during each of which they are presented with a number of competing images on screen (typically four). A verbal cue is given, and the participants are asked to select the image corresponding to the spoken word. All the while, participants are wearing (generally) a head-mounted eye tracking system recording where on screen they were fixated.

An individual trial of the VWP may be short, lasting anywhere from 1000ms to 2500ms before the correct image is selected. Prior to selecting the correct image, the participant's eyes scan the environment considering images as potential candidates to the spoken word. As this process unfolds, a snapshot of the eye is taken at a series of discrete steps (typically every 4ms) indicating where on the screen the participant is fixated. A single trial of the VWP typically contains no more than four to eight total "looks" before the correct image is clicked, resulting in a paucity of data in any given trial.

To be clear, eye trackers themselves only record x and y coordinates of the eye at any given time, and it is only after the fact that "psycho-physical" attributes are mapped onto the data (saccades, fixations, blinks, etc.,). We adopt the strategy of prior work in discussing eye tracking data in terms of their physiological mapping, as this will be crucial in constructing a physiologically

relevant understanding of the problem at hand [**?**].

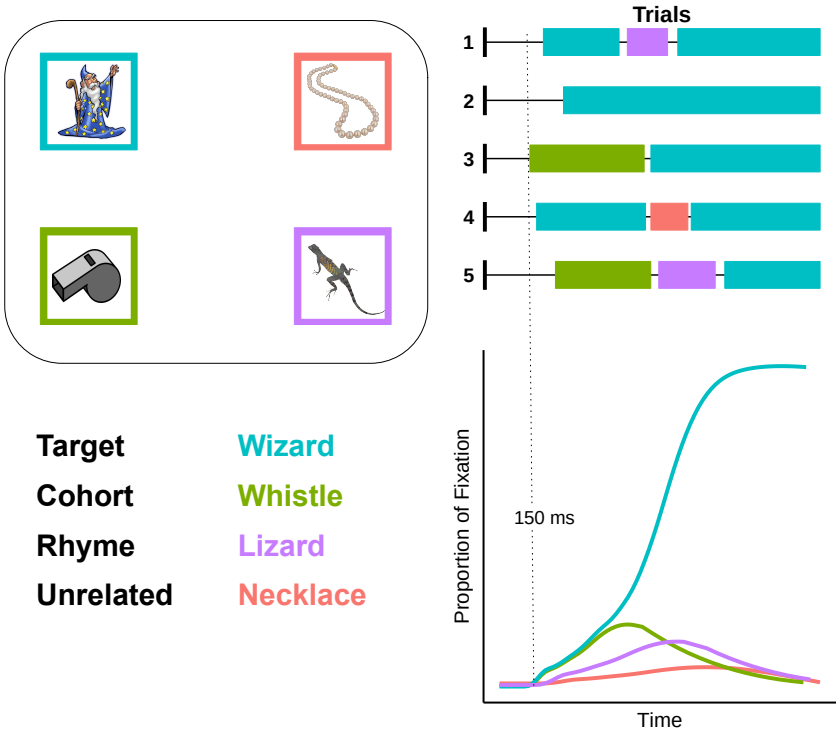


Figure 3.4: Trials in the Visual World Paradigm

To create a visual summary of this process aggregated over all of the trials, a la Allopena, a "proportion of fixations" curve is created, aggregating at each discrete time point the average of indicators of whether or not a participant is fixated on a particular image. A resulting curve is created for each of the competing categories (target, cohort, rhyme, unrelated), creating an empirical estimate of the activation curve, $f(t|\theta)$. See Figure 3.4. For any subject $i = 1, \ldots, n$, across times $t = 0, \ldots, T$ and trials $j = 1, \ldots, J$, a construction of this curves can be expressed as:

$$y_{it} = \frac{1}{J} \sum_{j=1}^{J} z_{ijt} \tag{3.1}$$

where z_{ijt} is an indicator $\{0, 1\}$ towards a particular object in trial j at time t. Following the contributions of **?**, we model the empirical curve given in Equation 3.1 with a parametric function $f(t|\theta)$, identifying subject-specific parameters θ_i which describe clinically relevant attributes of the curve,

$$f(t|\theta_i) \equiv y_{it}. \tag{3.2}$$

The recovery of subject-specific parameters occurs by subjecting the observed data to a simple loss function,

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}}\, \mathcal{L}(f_\theta, y). \tag{3.3}$$

As mentioned previously, fixations to the Target in a VWP experiment are often modeled with the four parameter logistic function (Figure 3.2):

$$f(t|\theta) = \frac{p - b}{1 + \exp\left(\frac{4s}{\mathrm{p}-b}(x - t)\right)} + b. \tag{3.4}$$

Similarly, a six parameter asymmetric Gaussian function , often used for fixations to competitors, is given as

$$f(t|\theta) = \begin{cases} \exp\left(\frac{(t-\mu)^2}{-2\sigma_1^2}\right)(p - b_1) + b_1 & \text{if } t \leq \mu \\ \exp\left(\frac{(t-\mu)^2}{-2\sigma_2^2}\right)(p - b_2) + b_2 & \text{if } t > \mu \end{cases} \tag{3.5}$$

In a typical VWP study, the collection of subject-specific curves is used to estimate temporal characteristics of a larger group, for example, identifying how the process of lexical activation differentiates itself between groups of typically developed children and those with specific cognitive disabilities. As such, it is critical that data collected in the VWP be able to not only accurately reflect relevant characteristics of individual subjects, but also that these characteristics are preserved in the aggregate such that meaningful differences can be correctly identified. This is facilitated by

having accurate and tractable models relating lexical activation to physiologically observable data.

To this end, and in light of the discussion of the mechanics presented here, we propose an alternative model linking data recorded in the VWP with lexical activation, centered around the cognitive mechanism initiating eye movements. We call this the look onset method, which we present in the next section.

3.3 Look Onset Method

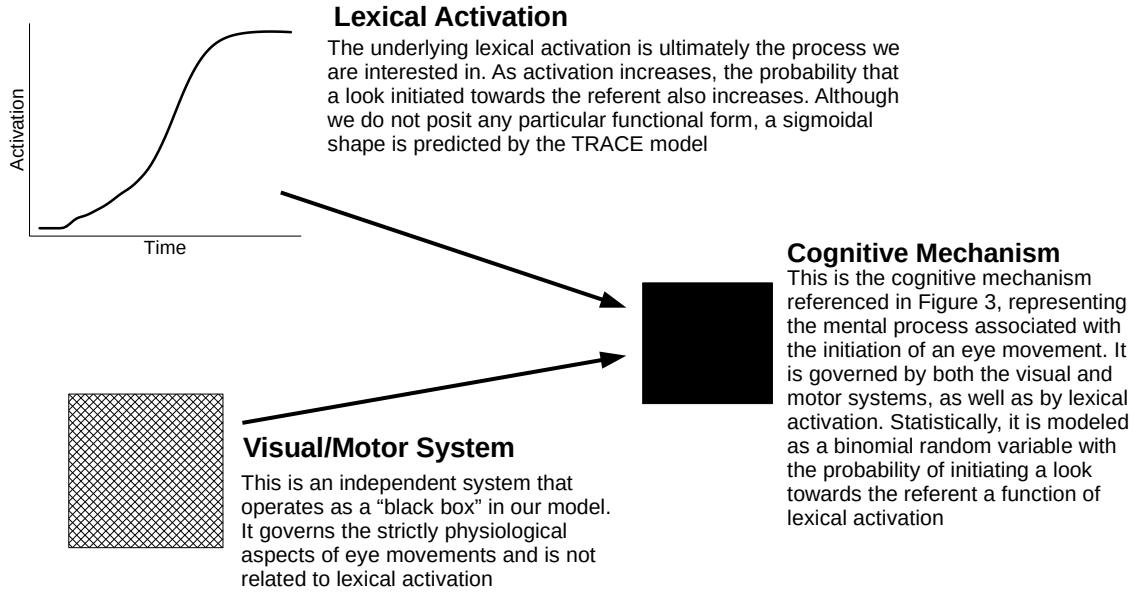


Figure 3.5: The cognitive mechanism that determines the destination of a look is a function of both lexical activation and the underlying visual/motor system

In this section, we present a new method for utilizing data in the Visual World Paradigm in pursuit of the recovery of latent activation. From the physiologically grounded model presented in Figure 3.3, we argue for delineating the several related but ultimately *distinct* processes associated with eye mechanics. The primary of these, and the focus of the method we present here, is that of

the cognitive mechanism responsible for initiating eye movements. Additionally, we consider the processes responsible for the duration of a fixation, as well as that governing oculomotor delay.

3.3.1 Look Onset Method

Outside of the implicit acknowledgement that the destination of saccadic eye movements are probabilistically related to lexical activation, we identify the cognitive mechanism initiating this movement as the first event downstream this cognitive process that culminates as a physiologically measurable phenomenon in the context of eye tracking data, despite the period of oculomotor delay separating this mechanism from the subsequent saccade.

Nonetheless, in the context of a Target object in the VWP (or any other specific referent), we can frame this mechanism as a Bernoulli event whereby the probability of this event resulting in a fixation to the Target object is determined by the activation level of that particular referent. In the case of a Target object, for example, this probability increases concomitant with the amount of auditory stimulus received.

From this observation, we propose here what we call the *look onset method*, which holds that the destination of a saccade launched at some time t, s_t, is probabilistically determined by its lexical activation at time t. Letting $f(t|\theta)$ denote the lexical activation to the Target at time t, for example, gives

$$s_t \sim Bern[f(t|\theta)]. \tag{3.6}$$

Under the look onset method, the *only* relevant data in the recovery of the underlying activation is that of the look onset, marked by the initiation of a saccade. That is, rather than subject specific data being recorded as an array of proportions for each observed time, the look onset method captures a set of ordered pairs, $\mathcal{S} = \{(s_t, t)\}$. Recovery of subject-specific parameters proceeds just as in Equation 3.3 with a simple loss function,

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}}\, \mathcal{L}(f_\theta, \mathcal{S}). \tag{3.7}$$

Just as **?** noted a consistency between activation levels predicted by the TRACE model (as in Figure 3.1) and the proportion of fixations, we also observe a general consistency between the activation levels predicted by TRACE and the probability of initiating an eye movement to a particular location at some point in time. Accordingly, we adopt as our specification of $f(t|\theta)$ the same parametric models introduced in **?** and given in Equations 3.4 and 3.5. This allows for a natural comparison in modeling VWP data using either the proportion of fixation or look onset methods.

In anticipation of the observation that the look onset method discards relevant information regarding the strength of activation by not implicitly including the length of fixation (which is captured indirectly in the proportion of fixation method through the aggregation of all fixations over time), we acknowledge this and reserve further comment for the discussion.

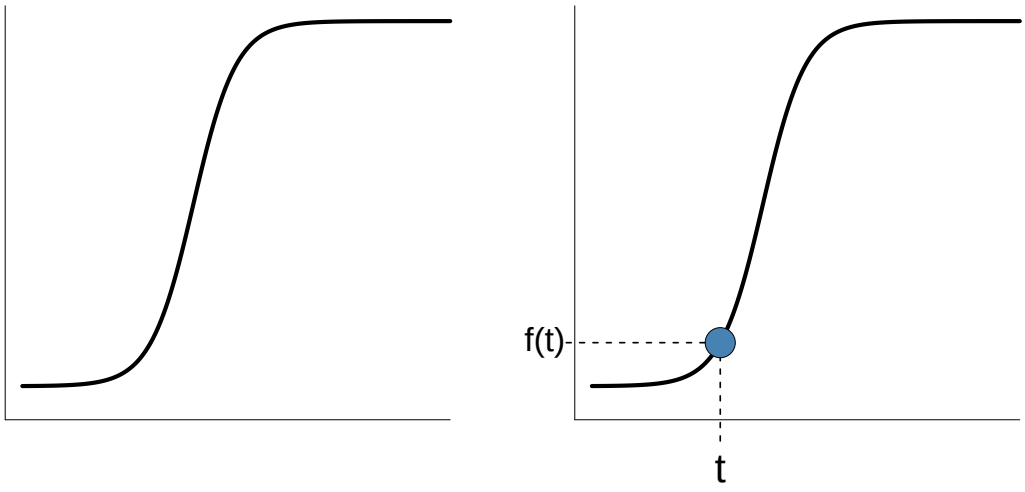
To explicitly summarize our position, we contend that the mechanism downstream the lexical process responsible for initiating an eye movement represents an unambiguous binomial event that is probabilistically related to activation. By modeling this event as such, we can more closely characterize how the underlying probability of initiating a look to a particular referent changes and, by extension, more closely approximate this process of lexical activation in time. Under these assumptions, we next detail a prominent source of bias present under the proportion of fixation method, as well as a second source of bias that has remained otherwise unattended in VWP studies.

3.3.2 Added Observation Bias

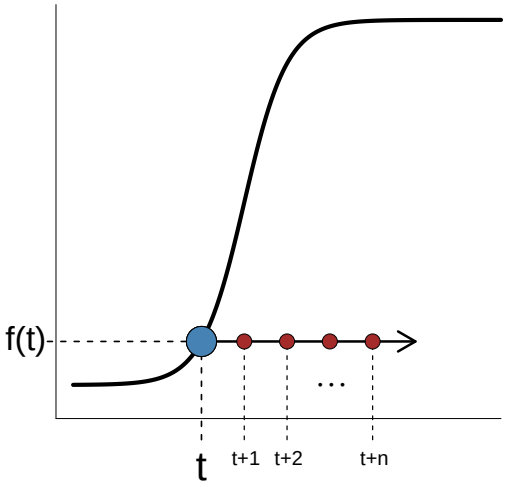
The look onset method begins with the assumption that relevant mechanism in the recovery of the underlying activation is the binomially distributed mechanism determining the location of a look, where the probability of initiating a look to e.g., the Target is determined by the strength of the underlying lexical activation. The look onset method accommodates this assumption by

recording as the only relevant data the initial moment a saccade is launched along with its subsequent destination. In contrast, the proportion of fixation method includes the *entire* duration of a fixation with an indicator $\{0, 1\}$, despite not having observed any additional behavior associated with the cognitive mechanism responsible for initiating an eye movement at that time. By using the entire fixation, it both obscures data that is relevant to the mechanism of interest (onset) while also conflating it with data generated by a fundamentally different process. This introduces what I call *added observation bias*. This bias is specifically with regards to the estimation of the underlying probability that a look initiated at some time t will be directed towards the referent, as illustrated in Figure 3.5.

To illustrate, consider a situation in which a participant in a VWP trial initiates a look to the Target object at time t with probability $f(t|\theta)$. Under both the look onset and proportion of fixation methods, we would mark with an indicator at time t the location of the look. For a fixation with a duration of length n, however, the proportion of fixation method would also mark with *the same indicator* values at $t+1, t+2, \ldots, t+n$ the outcome of the random variable drawn with probability $f(t|\theta)$. The consequences of this are twofold: first, by not distinguishing between the initial onset and subsequent fixation, we are "adding" observations to our observed data by including points in which no additional observations regarding the underlying process had been made. And second, these added observations are inherently biased: at time $t + n$, we are still recording outcomes drawn with probabilities $f(t|\theta)$, despite the fact that at $t + n$, the underlying activation would then be $f(t + n|\theta)$. The result of these combined is a distorted estimate of the underlying activation. A depiction of this phenomenon is given in Figure 3.6.



(a) (b)



(c)

Figure 3.6: **(a.)** Example of a nonlinear activation curve $f(t|\theta)$ **(b.)** At some time, t, a saccade is launched with its destination probabilistically determined by $f(t|\theta)$ **(c.)** For a look persisting over n time points, $t+1,\ldots,t+n$, we are recording "observed" data, adding to the proportion of fixations at each time but without having gathered any additional observed data at $f(t+1|\theta),\ldots,f(t+n|\theta)$, providing a biased estimate the true probability.

It would be constructive to comment briefly on how the added observation bias would manifest when modeling looks to the Target object, as this is something we will see in a number of simulations following this. As the logistic function modeling the Target is monotonic, it holds

that $f(t+1|\theta) > f(t|\theta)$ for all t. From the depiction given in Figure 3.6, we should anticipate that each of the added observations acts to deflate the true probability. Accordingly, we expect two see two distinct phenomenon: first, on account of the true probability being deflated at each time, we should expect the estimate of the slope to be negatively biased (i.e., a flatter slope), and second, we should expect the crossover parameter to be positively biased, on account of the flatter slope and consequent later inflection point. This deflation of values on the y-axis should result in what appears to be a horizontal shift along the x-axis, and indeed, this is what we will find in each of the cases that follow.

As a final comment here and to reiterate a point previously made, our intent here is not to say that the length of a fixation is irrelevant in consideration of the strength of underlying activation. Rather, we argue that by clearly delineating separate processes we can more concisely recover those that we are interested in.

3.3.3 Delayed Observation Bias

The final portion of the look onset model is that associated with oculomotor delay. Unlike the situation with the added observation bias, oculomotor delay affects both methods in that the delay between the mechanism of interest and the observed outcome is the same. Previously, we noted that this delay is typically acknowledged to be "roughly" 200ms, and in a standard VWP analysis it is usually accounted for with a horizontal shift of 200ms to the observed data. The ability to correctly specify and correct for this delay is critical, particularly in situations such as the look onset method in which the total data is much sparser and potentially much more sensitive to misspecification.

We facilitate this by specifically accounting for the phenomenon of oculomotor delay as a random process, denoted with the variable ρ. Under this specification, observe that a look initiated at time t is towards the target not with probability $f(t|\theta)$ but rather $f(t-\rho|\theta)$. The difference between these is what we call the *delayed observation bias*. There are two qualities of delayed observation bias that we consider here. The first is the simple observation that if there is any difference between

the true mean duration of oculomotor delay and 200ms, then the typical adjustment made to correct for it will itself be biased. Assuming that the mean oculomotor delay is 200ms (and there is no observed bias), our second consideration relates to the degree of randomness present and how this variability impacts the recovery of the underlying activation function.

While we don't here provide any specific solutions to this problem (though see future directions in the discussion, as well as the discussion on total variation in the appendix), it will be important to identify the potential impact of this phenomenon and demonstrate the need for further investigation.

3.4 Recovery of Individual Curves

In this section we construct a number of simulations to investigate the impact of added observation bias in the recovery of the underlying activation, as well as highlight the influence of randomness in the oculomotor delay in recovery. The simulations are constructed to emulate a typical study in the Visual World Paradigm, in which individual subjects are tasked with undergoing a series of trials, during each of which subjects make a series of fixations whose locations are probabilistically associated with lexical activation. For brevity, we consider in this section only those fixations associated with the Target object, modeled with the four parameter logistic as given in Equation 3.4; simulations according to looks to competitors is treated in the index, though the phenomenon we detail here are ultimately agnostic to any particular generating function. We will begin by detailing the process of simulating a single subject.

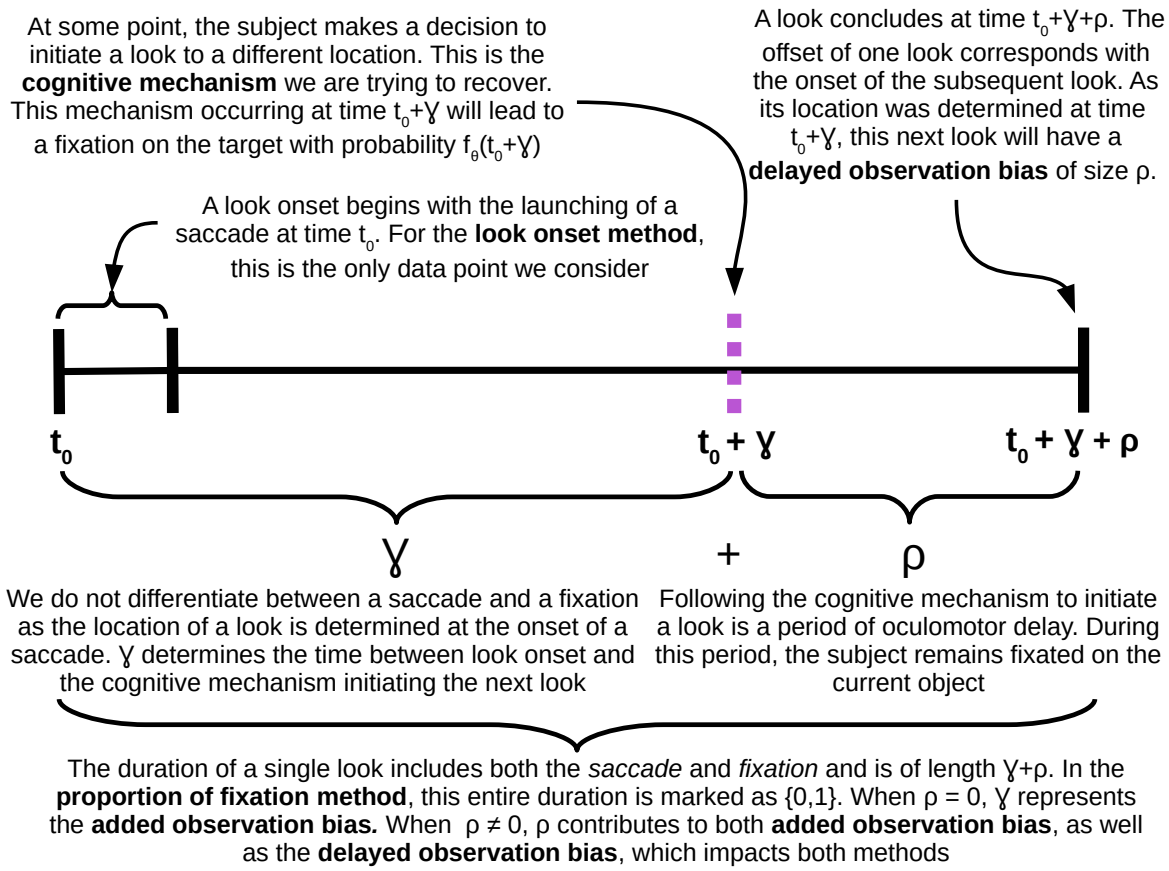


Figure 3.7: The Look Onset model

First, each subject i randomly draws a set of parameters θ_i from an empirically determined distribution based on normal hearing participants in the VWP [**?**] to construct a subject specific generating curve, $f(\cdot|\theta_i)$, where f here is assumed to be the logistic given in Equation 3.4. It is according to this function that the decision to initiate a look at time t will subsequently direct itself to the Target with probability $f(t|\theta_i)$. We then go about simulating trials according to the following method: At some time t_0, a subject initiates a look. This look persists for at least a duration of γ, drawn from a gamma distribution with mean and standard deviation independent of time and previous fixations. At time $t_0 + \gamma$, the subject determines the location of its next look, with the next look being directed towards the target with probability $f(t + \gamma|\theta_i)$. The decision to initiate a look is

followed by a period of oculomotor delay, ρ, during which time the subject remains fixated in the current location. Finally, at time $t_0 + \gamma + \rho$, the subject ends the look initiated at t_0 and immediately begins its next look to the location determined at time $t_0 + \gamma$. For the look onset method, the only data recorded are the times of a look onset and their location: in this case, at times t_0 and $t_0 + \gamma + \rho$. By contrast, the proportion of fixation method records the object of fixation at 4ms intervals for the entire period of length $\gamma + \rho$. A single trial begins at $t = 0$ and continues constructing looks as described until the total duration of looks exceeds 2000ms. Each subject undergoes 300 trials, and 1,000 subjects are included in each simulation.

Three total simulations were performed to investigate the biases identified in the previous section, each differing only in the random distribution of the oculomotor delay parameter, ρ. In the first simulation we set $\rho = 0$ to remove any oculomotor delay. Under this scenario, a look initiated at time t by subject i will be directed towards the target with probability $f(t|\theta_i)$. Doing so removes any potential bias from delayed observation and allows us to identify the effects of the added observation bias in isolation. In the remaining simulations we probe the effects of randomness in oculomotor delay, investigating what effect uncertainty may have in our recovery of the generating function. We do this assigning ρ to follow either a normal or Weibull distribution, each with a mean value of 200ms. As is standard in a VWP analysis, we subtracted 200ms from each observed point prior to fitting the data. A consequence of this is that in these simulations, the bias itself is accurately accounted for by subtracting the correct mean, with the resulting error in the curve fitting process the result of the inherent variability. This does not detract from the argument being made, however, and any true bias in the mean of the oculomotor delay would asymptotically result in a horizontal shift of the observed data according to the direction and magnitude of the bias.

Simulated data was fit to the four parameter logistic function using `bdots v2.0.0`.

As all of the data could not be individually inspected prior to being included in the analysis, subjects were excluded from consideration if fitted parameters from either the look onset method or the proportion of fixation method resulted in a peak less than the slope, or if the crossover or slope were negative. In the settings in which there was no delay, normally distributed delay, or

Weibull distributed delay, 981, 973, and 981 subjects were retained, respectively.

3.4.1 Results

Presented together are results from each of the simulations, beginning first with a collection of visual summaries. Simulation settings under each of the assumptions for oculomotor delay are displayed as a panel in Figure 3.8, 3.9, and 3.10. Each panel starts with a box plot showing the observed bias ($E(T)-\theta$) for each of the retained subjects. Following this is a representative sample of subject-specific curves under each method, alongside error curves demonstrating the difference between the generating and recovered curves.

The results presented for no oculomotor delay in Figure 3.8 serve to highlight the impact of the added observation bias in isolation, and from this we observe a few things. As predicted, we see positive bias in the crossover parameter for the look onset method, consistent with the observation that the majority of "observations" are underestimates of the true probability (as would be true in all monotone functions). As more time passes before the inflection point is reached, this has the secondary effect of "flattening" the estimated curve, an effect that we also see with the slight negative bias present in the slope parameter. In contrast, the recovery of each of the parameters under the look onset method appear to be unbiased.

These observations are again reflected in the presentation of representative curves, in all cases resulting in a curve estimated from the proportion of fixation method that is horizontally shifted right of the generating curve and also less steep. There is an interesting phenomenon whereby the horizontal shift in the representative curves visually appears to be a small difference – however, the bias between the true and recovered curves is not horizontal but vertical, and the magnitude of error between these is more obviously demonstrated in the representative error curves making up the last portion of the panel. This is especially important considering the fact that it is precisely this portion of the curve that is most sensitive to bias in recovery. The consequences of this are discussed briefly in the appendix.

For the remaining panels, we see that the primary effects of oculomotor delay are that of increased variability in the recovery of parameters. We also see evidence of bias in the recovery of all parameters for *both* methods in the case of Weibull delay, demonstrating that even when we make unbiased adjustments for oculomotor delay as we did here, the highly non-linear nature of these curves can result in downstream difficulties. This especially highlights the need for effectively addressing the effects of variability in oculomotor delay.

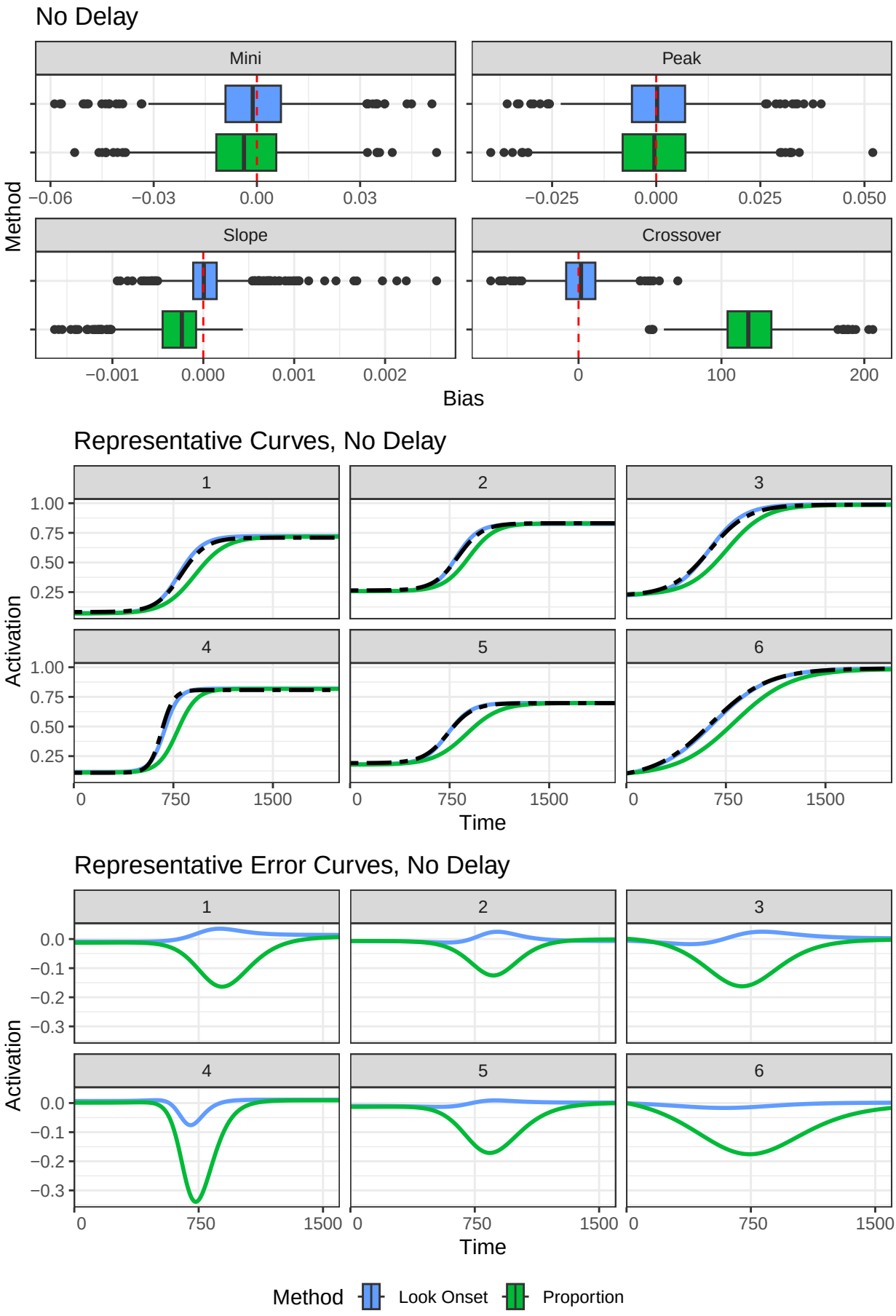


Figure 3.8: Summary of simulation results in the recovery of subject-specific curves generated by the logistic function with no oculomotor delay

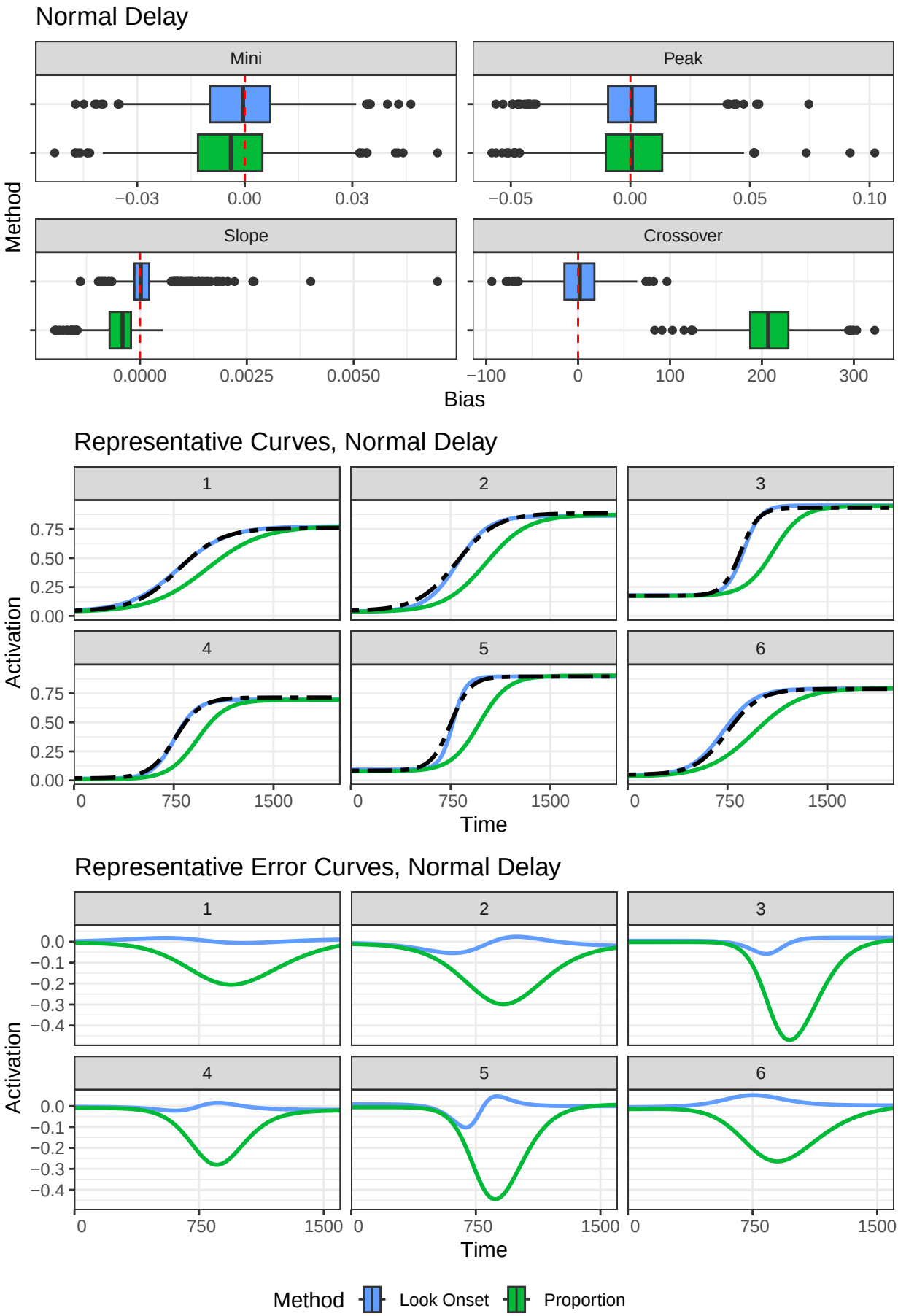


Figure 3.9: Summary of simulation results in the recovery of subject-specific curves generated by the logistic function with normally distributed oculomotor delay

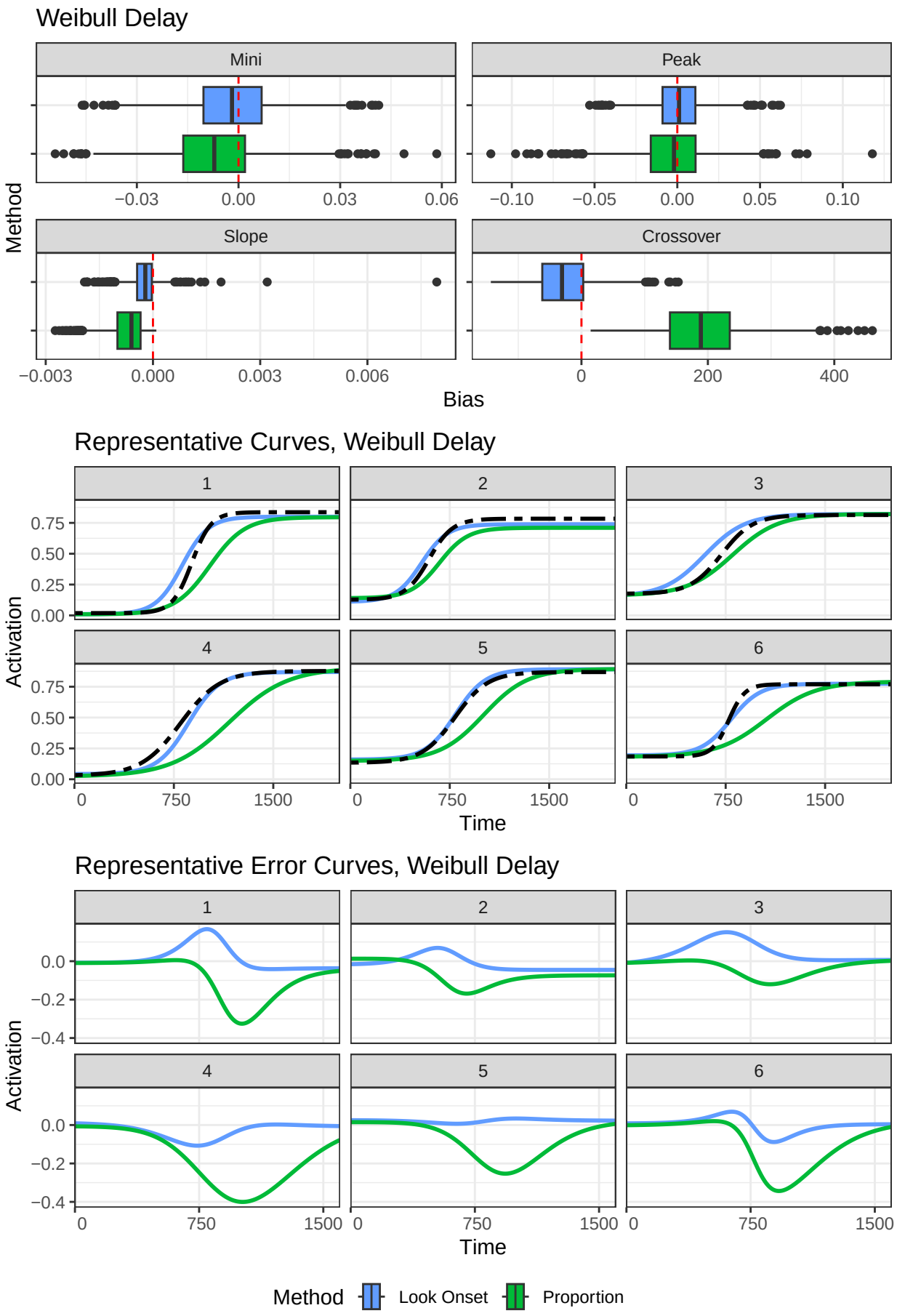


Figure 3.10: Summary of simulation results in the recovery of subject-specific curves generated by the logistic function with Weibull distributed oculomotor delay

In addition to the visual summaries, we present in Table 3.1 a summary of the mean integrated squared error (MISE) between the generating and recovered curves using each of the methods

We begin by noting the magnitude of difference between the look onset method and the proportion of fixation method in the case of $\rho = 0$, or No Delay, demonstrating the amount of bias introduced in the proportion method. This alone demonstrates how critical of an issue the added observation bias is in the recovery of the underlying activation.

To assess the effects of randomness in the oculomotor delay, it seems prudent to limit the comparisons within each method. Considering first the look onset method, we see that as the degree of variability increases, so does the difficulty in correctly recovering the underlying curve. It is important to note that these magnitudes are meant to be relative rather than absolute: the particular values observed are a function of the relationship between the generating γ distribution and that of ρ. Nonetheless, this does suggest a need to further investigate ways to control for the added uncertainty. To quickly comment on the apparently "flipped" MISE for the proportion of fixation method as it relates to the normal and Weibull distributed oculomotor delay, it would seem as if the skew of the Weibull distribution acted in such a way as to actually offset some of the observed added observation bias and seems more an artifact of the simulation conditions rather than an inherent statement relating OM bias to the proportion of fixation method in general.

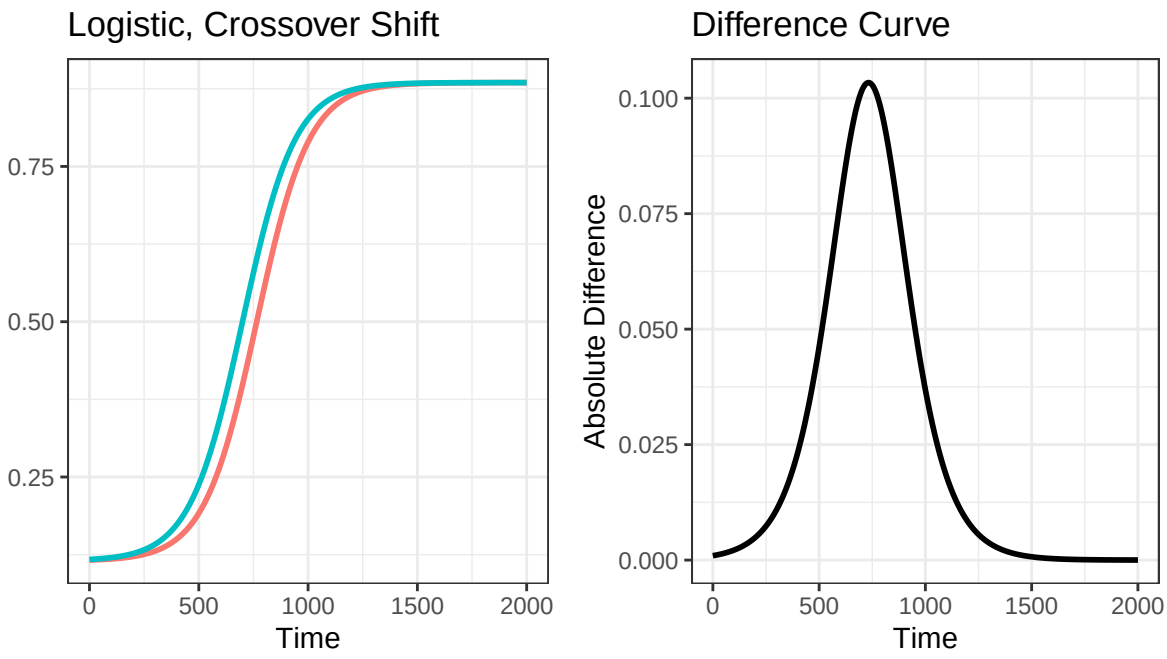
Method	Delay	1st Qu.	Median	3rd Qu.
Look Onset	No Delay	0.17	0.32	0.56
Look Onset	Normal Delay	0.37	0.71	1.24
Look Onset	Weibull Delay	1.05	2.16	4.23
Proportion	No Delay	8.21	11.33	16.01
Proportion	Normal Delay	22.90	30.65	39.37
Proportion	Weibull Delay	15.27	24.75	38.14

Table 3.1: Summary of mean integrated squared error across simulations

Overall, these results conclusively demonstrate both the devastating effect of added observation under the assumptions illustrated in Figure 3.7 as well as the need to consider the effects of variability associated with oculomotor delay, even when unbiased adjustments are able to be made.

3.5 Recovery of Group Differences

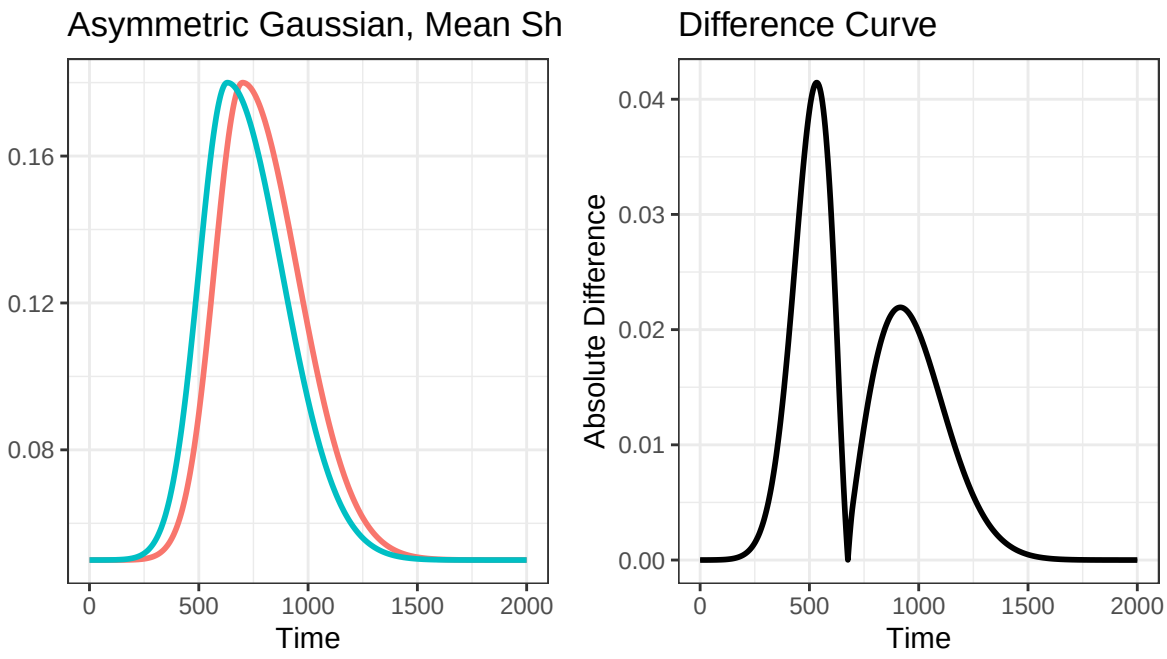
In Section 3.4, we demonstrated the utility of the look onset method in the asymptotic recovery of individual curves under our generative model. Often, however, the need to specify subject-specific differences is in pursuit of a higher goal, namely the identification of clinically relevant differences between groups. This problem is elaborated upon in **?**, where it is noted, for example, that simply averaging across participants data has the ability to either create or mask group-level effects (i.e., large variability in the crossover parameter in the logistic function in Equation 3.4 could manifest in the aggregate as a difference in slope). Having demonstrated the efficacy in recovering the subject-specific curves, we now consider if differences at the subject level are preserved in the aggregate through a power analysis using each of the methods discussed for identifying clinically relevant group differences.



(a) (b)

Figure 3.11: Plot of logistic groups with horizontal shift, alongside difference curve

We proceed just as we did in Section 3.4, generating data according to the process previously described and detailed in Figure 3.7, recovering individual curves via each the look onset and proportion of fixation methods. What differs here, however, is that each simulation is constructed to simulate drawing two groups with 25 subjects each from empirically estimated distributions that differ from one another as a horizontal shift of 65 in the location parameters (i.e., the crossover parameter in the logistic and the mean parameter in the asymmetric Gaussian). An illustration of the mean structure of each group and its shifted counterpart is depicted in Figures 3.11 and 3.12 for the logistic and asymmetric Gaussian, respectively.



(a) (b)

Figure 3.12: Plot of asymmetric Gaussian groups with horizontal shift, alongside difference curve

Just as in the case of the recovery of subject-specific curves, what we are interested in here is the identification of vertical differences between groups, the magnitude of which at each time point is given in Figures 3.11 and 3.12 as well. As greater differences should be easier to detect, we should anticipate a strong temporal relationship between the magnitudes of difference and the observed power.

Power for these methods is estimated in the following way: first, we simulate two groups of 25 subjects each by drawing function parameters from either the empirically determined multivariate normal distribution used in Section 3.4 or from that same distribution with the location parameter shifted horizontally by 65. We proceed exactly the same way, both in recovering the curves under the look onset and proportion of fixation method, and under conditions with no oculomotor delay, as well as Normally distributed and Weibull distributed delay. The fitted subject-specific curves are then used to create an estimate of the group distribution using the bootstrapping procedure in the `bdots` package. Using the `p_adjust = "oleson"` adjustment for the nominal alpha,

temporal regions identified as statistically significant are recorded. We repeat this process 1000 times, taking at each 4ms time point the proportion of instances in which statistically significant differences were identified.

3.5.1 Results

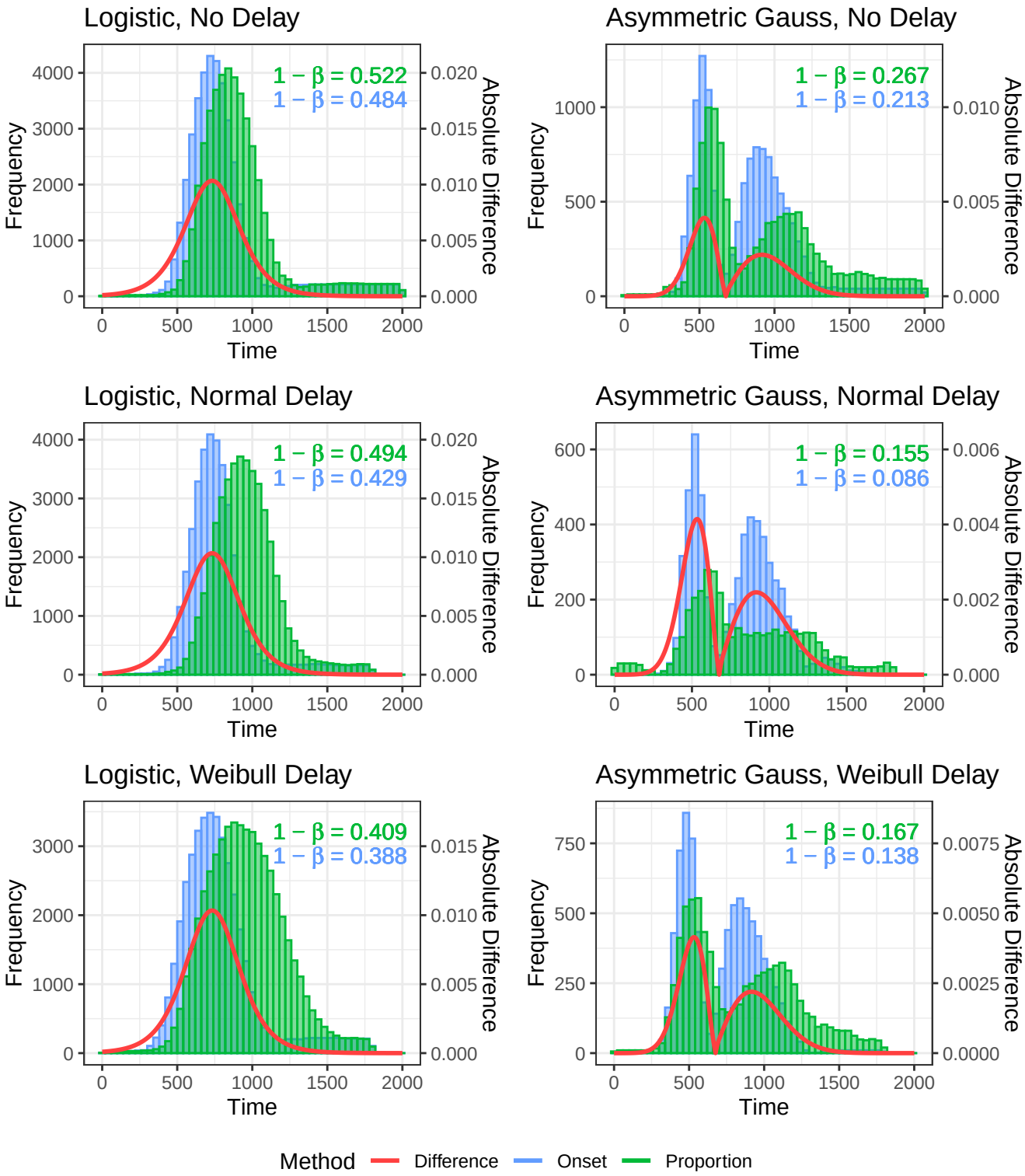


Figure 3.13: Histograms of observed power and overlaid error function

Results of the simulations are presented visually in Figure 3.13. Within each tile is a histogram indicating the proportion of trials in which a statistically significant difference was identified[1]. Laid over each histogram are plots of the error curves from Figures 3.11 and 3.12, helping to coordinate observed power in time. In the top right of each histogram is an estimate of total power, indicating the proportion of simulations in which *any* difference was identified. Finally, the correlation between the error curve and power density for each of the setting simulations is provided in Table 3.2.

Curve	Delay	Look Onset	Proportion of Fixations
Logistic	No Delay	0.9527	0.7789
Logistic	Normal Delay	0.9503	0.5390
Logistic	Weibull Delay	0.9861	0.6404
Asymmetric Gauss	No Delay	0.9520	0.6554
Asymmetric Gauss	Normal Delay	0.9359	0.6448
Asymmetric Gauss	Weibull Delay	0.9413	0.8681

Table 3.2: Correlation of power density with difference function between methods

There are several conclusions that can be readily drawn from the results presented. First, Figure 3.13 demonstrates that in each case, the observed overall power was greater for the look onset method. In conjunction with this is the observation that the magnitude of temporal power in the look onset method closely matches the magnitude of absolute difference, which is corroborated by the correlations presented in Table 3.2. A corollary of this observation is that clinically relevance differences that are detected by the proportion of fixation method may be temporally misaligned.

[1]Binwidth for each histogram is 40, which is why some proportions are greater than 1000

3.6 Application to Real Data

While improved performance in the theoretical domain is a necessary condition for the adoption of the look onset method, it is not sufficient, and towards that end we turn our attention now to an application with real data. We revisit here a study conducted by **?**, which collected VWP data on 93 children differing in language and cognitive abilities. This included children who were typically developed (N, $n = 40$), those with specific language impairment (SLI, $n = 20$) and non-specific language impairment (NLI, $n = 17$), and those with specific cognitive impairment (SCI, $n = 16$). Though the primary goals of this study involved both the investigation of individual differences and the mapping of particular physiological characteristics to TRACE parameters, we limit our consideration here to a standard analysis in `bdots` whereby we seek to identify temporal-specific differences in the trajectory of fixations with looks to the Target, modeled with the four parameter logistic function. We do this by structuring the original data according to both the proportion of fixations and look onset methods, fitting subject-specific curves to the data, and then identifying statistically significant differences via permutation testing in `bdots`. As there are four total groups, we consider each of the pairwise differences, resulting in six total differences for investigation.

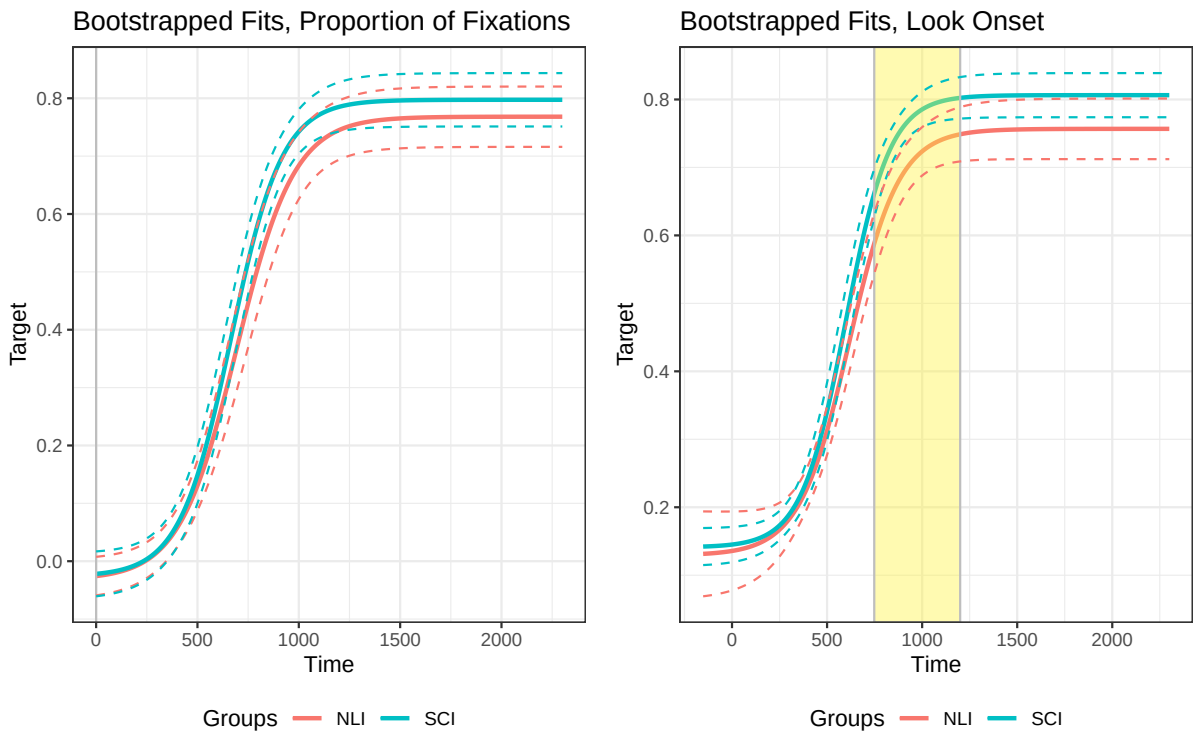
3.6.1 Data Preparation

There are a number of decisions that must be made when transforming raw eye tracking data to a format suitable for the look onset method. Briefly, we detail here the decisions made in processing data for both the proportion of fixations and for look onset. First, in all cases we removed trial data in any trials for which the subject made no fixations or selected the incorrect object in response to the auditory signal. We recorded no fixations or saccade movements (for look onset) that occurred past 2300ms, and we truncated data in each trial to end once the participant had selected the correct response.

For look onset data specifically, there are a number of additional decisions to be made, particularly regarding the beginning and end of an individual trial. At the beginning of a trial when $t = 0$, for example, the eyes are already fixated on some location (typically the center), yet these

are accounted for in the proportion of fixation. Regarding look onset, we must then decide whether we choose to take the first saccade launched after the beginning of the trial, resulting in a paucity of data near the beginning, or to choose to include the first saccade launched prior to the beginning of the trial which would be unrelated to the auditory stimulus (as it had not yet been received), but would offer more consistency with the data provided via the proportion of fixations. Similarly, one notes that at the end of a trial, a fixation will persist until the response has been collected (resulting in "observed" data), whereas the final saccade launched prior to response will often be a few hundred milliseconds sooner, resulting in a similar paucity of data at the end of the trial. Ultimately, we elected to only include saccades launched after the onset of auditory stimulus and made no further accommodation for the end of the trial, and although our results were not sensitive to either of these decisions, we believe they are worth consideration in future work.

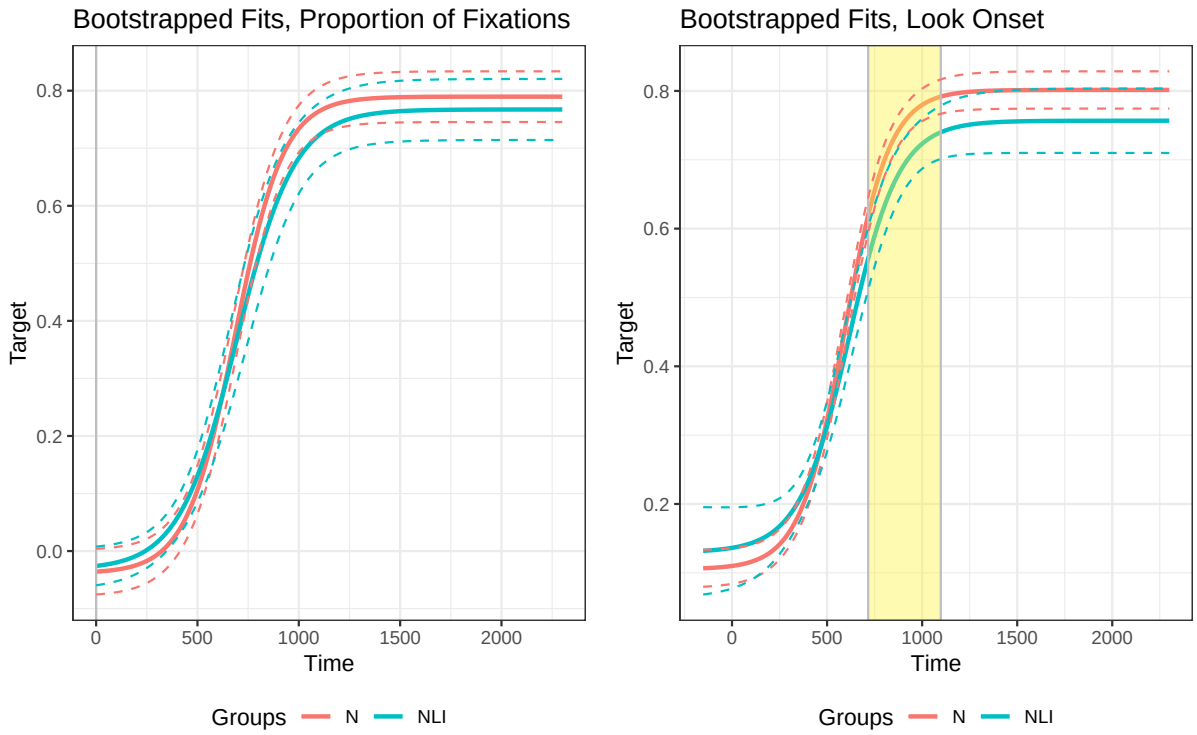
3.6.2 Results



(a) (b)

Figure 3.14: Estimated mean curves with confidence intervals of NLI and SCI, with significant region identified using the look onset method

Temporal differences between groups were made using permutation testing in `bdots`. Of the six comparisons made, only two were found to have significant differences between group curves, those being the comparisons made between specific cognitive impairment (SCI) and non-specific language impairment (NLI), as well as between NLI and typically developed adolescents (N). Further, these differences were only identified with the look onset method, with the proportion of fixation method in both cases offering null results. Plots of the mean curves and bootstrapped confidence intervals, along with highlighted regions of identified differences, are provided in Figure 3.14 and Figure 3.15.



(a) (b)

Figure 3.15: Estimated mean curves with confidence intervals of NLI and N, with significant region identified using the look onset method

3.7 Discussion

Through our investigation, we have presented a physiologically grounded model relating eye tracking data to underlying lexical access by placing emphasis singularly on the first instance of a look. Under the assumptions of this model, we further demonstrated a significant source of bias present under the standard "proportion of fixations" method typical of VWP data. We have proposed an alternative in response, the look onset method, which limits relevant data to the initial launching of a saccade. Under our generative model, we not only demonstrated superior recovery of individual subject-specific curves but also in the unbiased identification of temporal-specific differences between groups. The utility of this additional power was made evident in our re-examination of an existing study, in which the look onset method identified several statistically

significant differences between groups, whereas the proportion of fixation method did not. And finally, the look onset method can be implemented immediately, utilizing the same analytic approach provided by the `bdots` package in R.

The look onset method has the further benefit of giving rise to a much sparser dataset, providing a more computationally tractable basis for algorithms with complexity that scale in the number of observations. This may be of considerable advantage in fitting mixed models to the data, potentially providing the ability to control for trial-specific variability, an option that has hitherto been infeasible.

Of likely concern to many readers will be the fact that, as the look onset method retains only the initial onset of a fixation, a great deal of information is lost concerning the *duration* of the fixation. It is well established [**?**] that the length of a fixation is itself meaningful, with longer fixations generally associated with stronger activation. The length of fixation may also be important when attempting to differentiate fixations associated with searching patterns (i.e., what images exist on screen?) against those associated with consideration (is this the image I've just heard?). It would be incorrect to suggest that the look onset method considers the length of fixation *irrelevant*; rather, it makes no statement about it at all. In other words, by clearly delineating two separate processes, one initiating an eye movement, the other determining the duration, we free ourselves to construct far more nuanced models. For example, the length of a fixation may suggest a weighting to the proceeding look onset, with a longer duration being indicative of intentional consideration and shorter ones being suggestive of searching behavior. These questions suggest a number of experimental designs to differentiate eye tracking behavior, particularly with regards to scanning/search behavior and experimental conditions, similar to what is demonstrated in **?**.

It is also critical to specify a present shortcoming in how the duration of fixations are treated. Implicit in the proportion of fixations method is a crucially overlooked assumption of a linear relationship between the fixation length and the activation. That is, insofar as the construction of the fixation curve is considered, a fixation persisting at 20ms after look onset (and well within the refraction period in which no new information regarding the cognitive mechanism or voluntary

fixation could be obtained, see Figure 3.3) is considered identical to a fixation persisting at 500ms after onset: both are undifferentiated in being recorded as either a 0 or 1. In other words, the specific duration of a particular fixation does not directly change how the data is recorded. This is in contrast to the possibility offered by the look onset method, whereby the duration of the fixation could weight the look onset by importance.

A second consideration necessary for using the length of a fixation is the composition of the fixation itself, given in Figure 3.3. Suppose, for example, that the refractory period following each fixation was exactly 100ms and that the oculomotor delay before initiating the subsequent fixation is exactly 200ms. From this, we could conclude that 300ms of any fixation are "built in" and have no necessary information regarding the strength of activation motivating the fixation. Without considering this information, we may determine a fixation of 500ms to be 25% greater than a fixation of 400ms. However, once we account for the "built in" portion of a fixation, we see that the first is associated with an intentional fixation period of 200ms, whereas the second has an intentional period of only 100ms. In this light, the first fixation might be considered 100% greater than the latter. And while we offer no more than speculation here, this phenomenon should be taken into consideration in future research.

In conclusion, we have presented a statistically defensible generative model for eye movements in the context of the Visual World Paradigm, accompanied by a novel method for aggregating and analyzing collected data. While the methods proposed are not a drastic alteration to the motivating assumptions of **?**, we believe them to be a critical step forward towards a statistically sound treatment of eye tracking data in the realm of lexical activation.

Appendices

3.8 Misc OM Discussion

Outside of a demonstration of its existence and potential consequence, little more has been said about addressing the delayed observation bias. Further, the consequences of the delayed observation (under the assumption that the mean value is correctly accounted for) seem almost

trivial in comparison to the differences between it and the added observation bias. That being said, we believe there are still critical reasons for considering its significance.

As mentioned earlier, the particular values observed in these simulations are both a function of the relationship between the distribution generating γ and that of ρ. However, they are also a function of the generating function itself. In particular, we draw attention to the degree of total variation f over the interval $[a, b]$, defined as

$$V(f) = \sup_{\mathcal{P}} \sum_{i=0}^{n_p-1} |f(t_{i+1}) - f(t_i)|, \tag{3.8}$$

where $\mathcal{P} = \{P = \{t_0, \ldots, t_{n_p}\}\}$ is the set of all possible partitions of $[a, b]$. Despite appearances, this is a relatively straightforward metric in the case of monotone functions such as the logistic, where the total variation is simply $|f(b) - f(a)|$. To illustrate the relevance of this, consider a hypothetical situation in which the underlying activation we are wishing to recover is a constant function, $f(t) = c$, where the probability of fixating on a target is independent of time. In such a situation, a delayed observation would be of no issue; despite changes in time t, the probability c remains unchanged. In contrast, consider a second hypothetical situation in which activation is defined exponentially, $f(t) = \exp(t)$. In this case, the impact of delayed observation depends drastically on time, when the delay in observation in the range of small values of t have a drastically smaller impact than delayed observations when t is large ($\exp(1) - \exp(0) = 1.7183$ while $\exp(11) - \exp(10) = 37848$, despite both cases having $\Delta t = 1$).

In short, these hypothetical situations detail how the magnitude of total variation can have differential effects on the delay in observation. Now consider again the logistic function in Figure 3.2 and imagine its domain partitioned into three equally sized portions. Both the first and third, near the asymptotes, have relatively low total variation, resulting in a relatively benign effect from oculomotor delay. In contrast, the middle third contains nearly all of the variation of the function, indicating the delayed observation here will have a disproportionate impact on the successful

recovery of the function. Given the clinical relevance of both the slope and crossover parameters, as well as acknowledging the impact that these have on the overall shape of the function, we demonstrate a need in a accounting for this delay precisely where it will impact function recovery the most. This, of course, is not unique to the logistic, with the effects of the delayed observation bias compounded in the asymmetric Gaussian functions (appendix) which has a more complicated variation structure and, accordingly, more difficulty in recovering the generating curves.

3.9 Recovery of Individual Curves – Asymmetric Gaussian

Presented here are the results of the simulations for the recovery of subject-specific curves generated with the asymmetric Gaussian function, the parametric function typically associated with looks to competitors in the VWP. As in the section fitted with the logistic function, simulations include settings in which there is no oculomotor delay, as well as delay that is normally and Weibull distributed. Again, as all fits could not be individual examined, an automated criterion was used to determine which fits were considered adequate. Here, this stipulated that the estimated sigma parameters be positive and that the height parameter be larger than either of the base parameters. The number of fits retained for the no delay, normal delay, and Weibull delay were 855, 786, and 816, respectively.

3.9.1 Results

As might be expected, the more complicated mean structure provided by the asymmetric Gaussian led to a generalized increase in the difficulty of recovery for both the look onset and proportion methods, relative to those generated with a logistic mean structure. However, we do still find that in the case of no delay, given in Figure 3.16, that the recovery of individual parameters is still unbiased and, as with the logistic, the location parameter (here, mu), is right shifted.

The results for the median integrated squared error are given in Table 3.3. We again see results similar to those with the logistic in that the look onset method outperforms the proportion of fixation method in all cases.

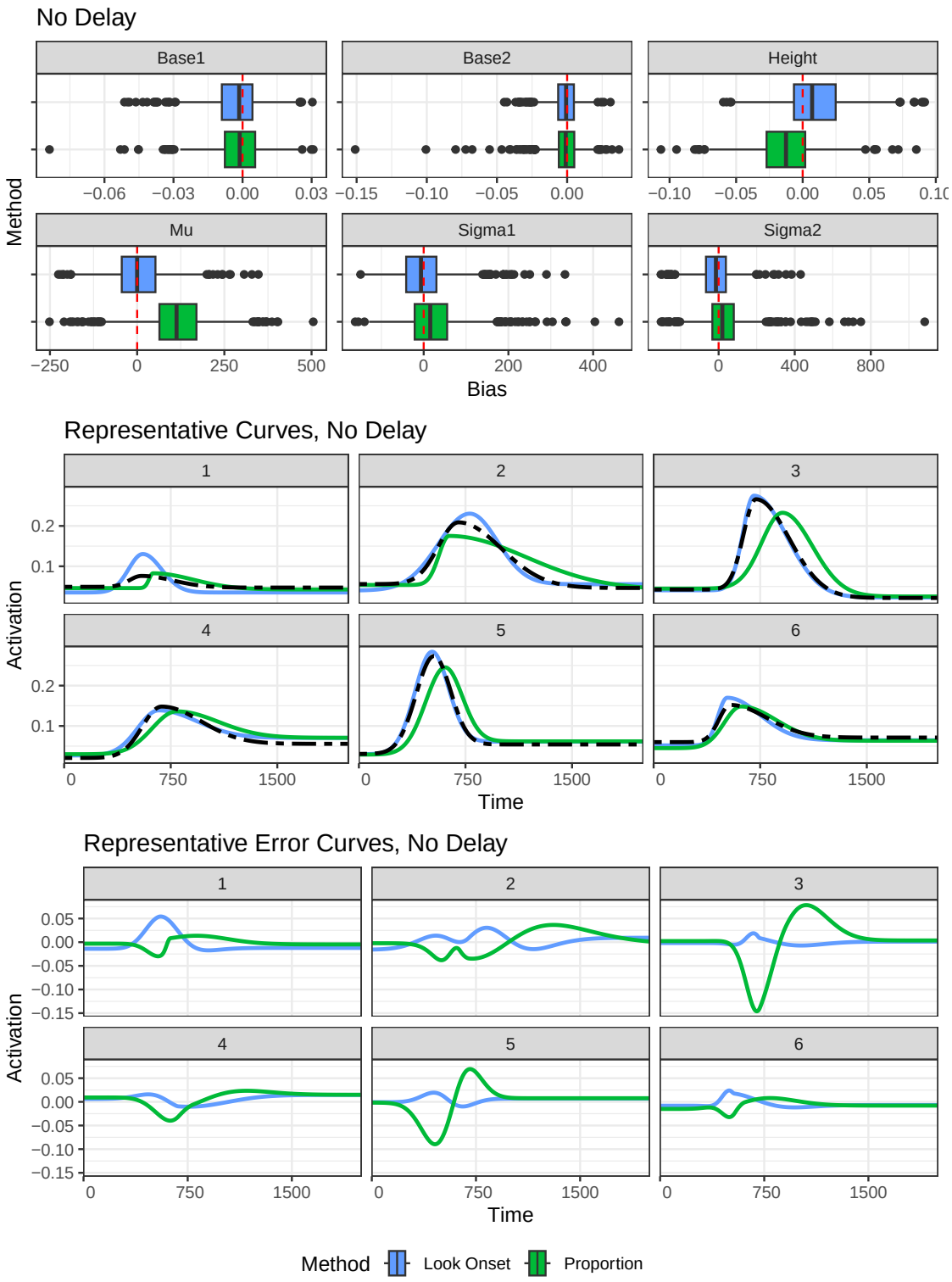


Figure 3.16: Summary of simulation results in the recovery of subject-specific curves generated by the asymmetric Gauss with no oculomotor delay

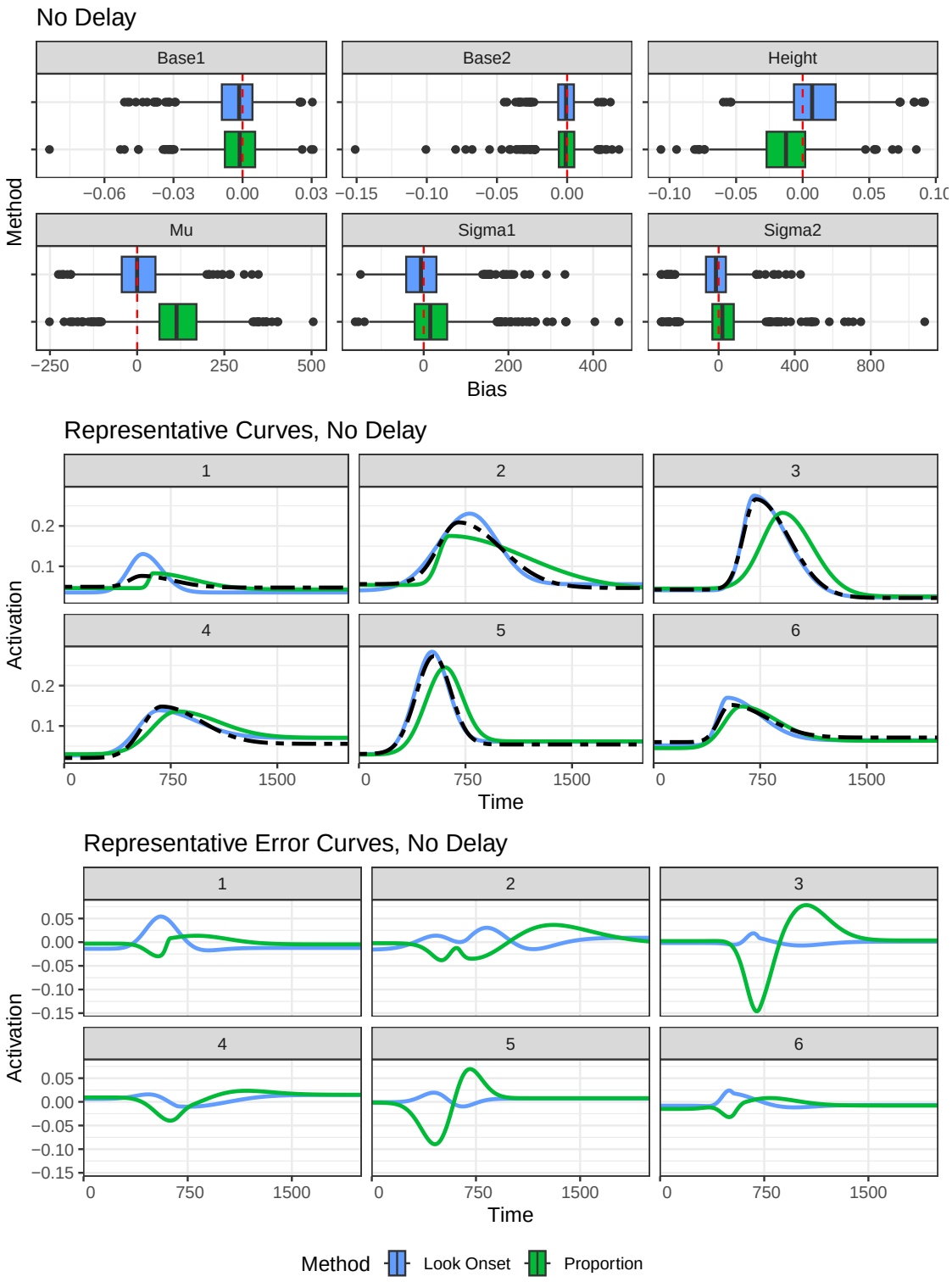


Figure 3.17: Summary of simulation results in the recovery of subject-specific curves generated by the asymmetric Gauss with normally distributed oculomotor delay

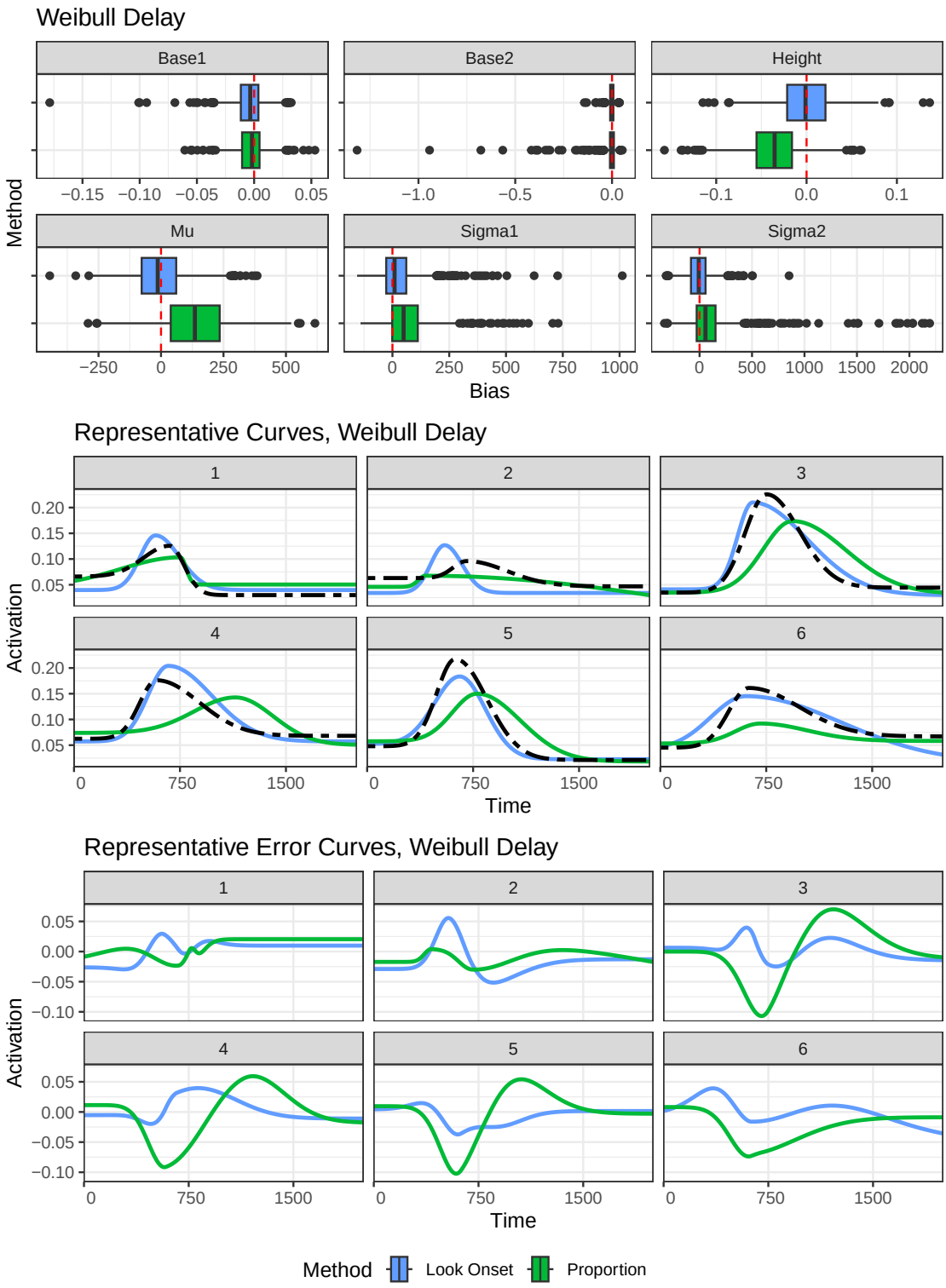


Figure 3.18: Summary of simulation results in the recovery of subject-specific curves generated by the asymmetric Gauss with Weibull distributed oculomotor delay

Curve	Delay	1st Qu.	Median	3rd Qu.
Look Onset	No Delay	0.22	0.36	0.63
Look Onset	Normal Delay	0.38	0.70	1.15
Look Onset	Weibull Delay	0.52	0.84	1.39
Proportion	No Delay	0.75	1.29	2.08
Proportion	Normal Delay	1.38	2.44	3.96
Proportion	Weibull Delay	1.00	1.98	3.43

Table 3.3: Median integrated squared error for recovery of individual curves generated with asymmetric Gaussian

3.9.2 R^2 for Recovery of Individual Curves

Here, we provide an alternative summary of the recovery of subject specific curves fit with both the logistic and asymmetric Gauss.

3.9.2.1 Logistic

Curve	Delay	1st Qu.	Median	3rd Qu.
Look Onset	No Delay	1.00	1.00	1.00
Look Onset	Normal Delay	0.99	1.00	1.00
Look Onset	Weibull Delay	0.98	0.99	0.99
Proportion	No Delay	0.92	0.94	0.95
Proportion	Normal Delay	0.80	0.83	0.86
Proportion	Weibull Delay	0.80	0.86	0.91

Table 3.4: R^2 for Logistic

3.9.2.2 Asymmetric Gaussian

Curve	Delay	1st Qu.	Median	3rd Qu.
Look Onset	No Delay	0.80	0.91	0.95
Look Onset	Normal Delay	0.63	0.82	0.91
Look Onset	Weibull Delay	0.57	0.77	0.87
Proportion	No Delay	0.48	0.65	0.75
Proportion	Normal Delay	0.10	0.33	0.52
Proportion	Weibull Delay	0.20	0.46	0.64

Table 3.5: R^2 for Asymmetric Gaussian

CHAPTER 4
METHODOLOGICAL CHANGES IN THE BOOTSTRAPPED DIFFERENCES OF TIME SERIES

4.1 Introduction

A problem ubiquitous throughout the sciences, though in the cognitive sciences especially, is that of statistically analyzing a process unfolding in time. In particular, we consider the problem of comparing a process in time as it evolves differentially between two or more experimental groups. And while there are many techniques for demonstrating *that* a difference exists, few offer any insight into *when*. Testing for temporal differences is complicated by the fact that when this process is continuous, there are often an arbitrarily large number of time points that could be compared. This is essentially a problem of multiple comparisons.

Various approaches have been proposed for addressing this issue, the most rudimentary of which involve binary tests for identifying the existence of any difference, such as the area under the curve (AUC). More sophisticated techniques involve the used of cluster-based permutation testing [**?**], whereby test statistics are computed at each observed time point with adjacent significant tests being combined into clusters. This controls the family-wise error rate (FWER) by reducing adjacent test statistics into a single cluster, thereby reducing the number of total tests. More recently, **?** introduced a modified Bonferroni correction to a series of test statistics, using estimates of autocorrelation between statistics to make the appropriate adjustments to the significance levels to control FWER. This approach, which they named bootstrapped differences in time series, was introduced in the R package, `bdots` [**?**].

The modified Bonferonni correction used by `bdots` relies on the construction of estimated distributions of time series via bootstrapping for each experimental group. These distributions, in turn, are used to construct test statistics at each observed time point. A closer look at the original iteration presents concerns as it involves quite restrictive assumptions on the data that are unlikely to be met in many, if not most situations.

This includes data typically collected in the context of the Visual World Paradigm (VWP), a widely used experimental paradigm in language research that involves tracking eye movements from participants in response to spoken language. This is notable in that it was data collected in VWP research that motivated the `bdots` methodology. There, it was maintained that the process of interest in each experimental group assumed a homogeneous mean structure, with no between-subject variability to be accounted for. Empirical data collected in a variety of contexts suggests that this assumption is unlikely to be true, the consequence of which is a type I error rate that is unacceptably high.

Here, we present two alternative methods that accommodate flexibility in the assumptions made by the original bootstrapped differences in time series algorithm. First, we propose a modified bootstrapping procedure that adequately accounts for observed between-subject variability while retaining the FWER adjustment that addresses the autocorrelated test statistics. In addition, we offer a permutation test for identifying temporal differences between groups, borrowing from the insight of the original bdots in that it also captures within-subject variability as demonstrated in the standard errors in the model fits.

We begin with a mathematical description of the problem along with the proposed alternatives to the original bootstrapping algorithm. This is followed by a simulation estimating the FWER across a number of experimental conditions. Finally, we consider two separate simulations for the assessment of power among the competing methods: this includes one simulation with a simple piecewise-linear function relating power to effect sizes; the second simulation interrogates power when the magnitudes of between-subject and within-subject variability differ.

4.2 Methods

In each of the methods to be described, we begin with the observation of y_{it} for subjects $i = 1, \ldots, n$ over times $t = 1, \ldots, T$. Typically, these subjects fall into different groups $g = 1, \ldots, G$, with each group containing n_g subjects. We further assume that the empirically observed data follows from a parametric function f with associated error:

$$y_{it} = f(t|\theta_i) + \epsilon_{it} \tag{4.1}$$

where θ_i is the subject-specific parameterization of f with

$$\epsilon_{it} = \phi\epsilon_{i,t-1} + w_{it}, \quad w_{it} \sim N(0, \sigma). \tag{4.2}$$

Under this paradigm, the errors are permitted to be either iid normal (with $\phi = 0$) or have an AR(1) structure, with $0 < \phi < 1$. It is generally assumed that the observed data across subjects make up a distribution for each group, specified as either a multivariate distribution of the parameters θ_i or a distribution of resulting curves, $f(t|\theta_i)$. Ultimately, it will be from a distribution of curves that we determine the temporal characteristics of each group, with the differences in these temporal characteristics being what we are interested in identifying. With the general notation addressed, we now move to the particulars of each of the methods considered.

4.2.1 Homogeneous Bootstrap

The original bootstrapping algorithm presented in **?** follows what we will call the *homogeneous means* assumption. Accordingly, we will call this bootstrapping algorithm the *homogeneous bootstrap*. Under the homogeneous means assumption, it is still assumed that observed data for each subject retains the mean structure given in Equation 4.1, but with the additional assumption that $\theta_i = \theta_j$ for all subjects i, j within the same group. In other words, there is assumed to be no variability in the mean structure between subjects within the same group. This is evidenced in the original bootstrapping differences in time series algorithm, which samples without replacement at each bootstrapping step:

1. For each subject, fit a nonlinear regression model to obtain $\hat{\theta}_i$. **?** recommends specifying an AR(1) autocorrelation structure for model errors. Assuming large sample normality, the sampling distribution of each estimator can be approximated by a multivariate normal distribution with mean for subject i, $\hat{\theta}_i$ corresponding to the point estimate and standard deviations

corresponding to the standard errors, s_i.

2. Using the approximate sampling distributions in (1), randomly draw one bootstrap estimate for each of the model parameters on every subject

$$\hat{\theta}_i^{(b)} \sim N(\hat{\theta}_i, s_i^2) \tag{4.3}$$

3. Once a bootstrap estimate has been collected for each parameter and for every subject, for each parameter, find the mean of the bootstrap estimates across n_g individuals for the bth bootstrap in group g,

$$\theta_g^{(b)} = \frac{1}{n_g} \sum_{i=1}^{n_g} \hat{\theta}_i^{(b)} \tag{4.4}$$

4. Use the mean parameter estimates to determine a bootstrapped population level curve, which provides the average population response at each time point, $f(t|\theta_g^{(b)})$.

5. Perform steps (2)-(4) B times to obtain estimates of the population curves. Use these to create estimates of the mean response and standard deviation at each of the time points. For each group $g = 1, \ldots, G$, this gives

$$\overline{p}_{gt} = \frac{1}{B} \sum_{b=1}^{B} f(t|\theta_g^{(b)}), \qquad s_{gt}^2 = \frac{1}{B-1} \sum_{b=1}^{B} f(t|\theta_g^{(b)}) - \overline{p}_{gt}, \tag{4.5}$$

where $\overline{p}_{gt}$ and s_{gt}^2 are mean and standard deviation estimates at each time point for group g.

Population means and standard deviations at each time point for each of the groups were used to construct a series of (correlated) test statistics, where the family-wise error rate was controlled by using the modified Bonferonni correction introduced in **?** to test for significance. As this correction is also used for the heterogeneous bootstrap presented next, we offer a more comprehensive review of this adjustment at the end of this section.

4.2.2 Heterogeneous Bootstrap

Typically, subjects within a group demonstrate considerable variability in their mean parameter estimates. In this case, we should avoid the presumption that $\theta_i = \theta_j$, as accounting for between-subject variability within a group will be critical for obtaining a reasonable distribution of the population curves. More likely, we may assume that the distribution of parameters for subjects $i = 1, \ldots, n_g$ in group $g = 1, \ldots, G$ follows the distribution

$$\theta_i \sim N(\mu_g, V_g), \tag{4.6}$$

where μ_g and V_g are the group-specific mean and variance values, respectively. In contrast to the previous set of assumptions, we call this the *heterogeneous means* assumption Similar to what was presented in the homogeneous bootstrap algorithm, we can further account for uncertainty in our estimation of θ_i by $\hat{\theta}_i$ by treating the standard errors derived when fitting the observed data to the mean structure suggested in Equation 4.1 as estimates of their standard deviations. This gives us a multivariate normal distribution for each subject's estimated parameter,

$$\hat{\theta}_i \sim N(\theta_i, s_i^2). \tag{4.7}$$

As our goal remains as being able to obtain reasonable estimates of the population curves for each group, it is necessary to estimate both the observed within-subject variability found in each of the $\{s_i^2\}$ terms, *as well as* the between-subject variability present in V_g. For example, let θ_{ib}^* represent a bootstrapped sample for subject i in bootstrap $b = 1, \ldots, B$, where

$$\theta_{ib}^* \sim N(\hat{\theta}_i, s_i^2), \tag{4.8}$$

as was done in Step (2.) of the homogeneous bootstrapping algorithm. If we were to sample *without replacement*, we would obtain a homogeneous mean value from the bth bootstrap for

group g, $\theta_{bg}^{(hom)}$, where

$$\theta_{bg}^{(hom)} = \frac{1}{n_g}\sum_{i=1}^{n_g}\theta_{ib}^{*}, \quad \theta_{bg}^{(hom)} \sim N\left(\mu_g, \frac{1}{n_g^2}\sum_{i=1}^{n_g} s_i^2\right). \tag{4.9}$$

Such an estimate captures the totality of the within-subject variability with each draw but fails to account for the variability in the group overall. For this reason, we sample the subjects *with* replacement, creating the heterogeneous bootstrap mean $\theta_{bg}^{(het)}$, where again each θ_{ib}^{*} follows the distribution in Equation 4.8, but the heterogeneous bootstrapped group mean now follows

$$\theta_{bg}^{(het)} \sim N\left(\mu_g, \frac{1}{n_g}V_g + \frac{1}{n_g^2}\sum s_i^2\right). \tag{4.10}$$

The estimated mean value remains unchanged, but the variability is now fully accounted for. We therefore present a modified version of the bootstrap which we call the *heterogeneous bootstrap*, making the following changes to the original:

1. In step (1), the specification of AR(1) structure is *optional* and can be modified with arguments to functions in `bdots`. Our simulations show that while failing to include it slightly inflates the type I error in the heterogeneous bootstrap when the data truly is autocorrelated, specifying an AR(1) structure can lead to overly conservative estimates when it is not.

2. In step (2), we sample subjects *with replacement* and then for each drawn subject, randomly draw one bootstrap estimate for each of their model parameters based on the mean and standard errors derived from the `gnls` estimate.

Just as with the homogeneous bootstrap, these bootstrap estimates are used to create test statistics T_t at each time point, written

$$T_t^{(b)} = \frac{(\overline{p}_{1t} - \overline{p}_{2t})}{\sqrt{s_{1t}^2 + s_{2t}^2}}, \tag{4.11}$$

where $\overline{p}_{gt}$ and s^2_{gt} are mean and standard deviation estimates at each time point for groups 1 and 2, respectively. Finally, just as in **?**, one can use the autocorrelation of the $T^{(b)}_t$ statistics to create a modified α for controlling the FWER.

4.2.3 Permutation Testing

In addition to the heterogeneous bootstrap, we also introduce a permutation method for hypothesis testing. The permutation method proposed is analogous to a traditional permutation method, but with an added step mirroring that of the previous in capturing the within-subject variability. For a specified FWER of α, the proposed permutation algorithm is as follows:

1. For each subject, fit the nonlinear function with *optional* AR(1) autocorrelation structure for model errors. Assuming large sample normality, the sampling distribution of each estimator can be approximated by a normal distribution with mean corresponding to the point estimate and standard deviation corresponding to the standard error

2. Using the mean parameter estimates derived in (1), find each subject's corresponding fixation curve. Within each group, use these to derive the mean and standard deviations of the population level curves at each time point, denoted $\overline{p}_{gt}$ and s^2_{gt} for $g = 1, 2$. Use these values to compute a permutation test statistic $T^{(p)}_t$ at each time point,

$$T^{(p)}_t = \frac{|\overline{p}_{1t} - \overline{p}_{2t}|}{\sqrt{s^2_{1t} + s^2_{2t}}}. \tag{4.12}$$

   This will be our observed test statistic.

3. Repeat (2) P additional times, each time shuffling the group membership between subjects. This time, when constructing each subject's corresponding fixation curve, draw a new set of parameter estimates using the distribution found in (1). Recalculate the test statistics $T^{(p)}_t$, retaining the maximum value from each permutation. This collection of P statistics will serve as our null distribution which we denote $\widetilde{T}$. Let $\widetilde{T}_\alpha$ be the 1 - α quantile of $\widetilde{T}$

4. Compare each of the observed $T_t^{(p)}$ with $\widetilde{T}_\alpha$. Areas where $T_t^{(p)} > \widetilde{T}_\alpha$ are designated significant.

4.2.4 Paired Data

Briefly, we attend to the issue of paired data for each of the discussed methods, as this is critical for a proper assessment of both FWER and power. Specifically, in a paired setting we note that our interest is in determining the distribution of paired differences rather than of the respective groups.

For the homogeneous bootstrap, this is done by default: as each subject is sampled without replacement, we can be sure that each bootstrap estimate between groups contains the same subjects. For the heterogeneous bootstrap, this is done by ensuring that the same subjects sampled in each bootstrap for one group is matched identically with sampling subjects in the other. Put differently, this states that we begin by determining which subjects will be included in each bootstrap *for both groups at the same time*. Lastly, in the case of permutation testing, paired data is addressed by ensuring that each permuted group contains one observation for each subjects, so that each paired subject in one permuted group has its corresponding observation in the other.

4.2.5 Modified Bonferonni Correction

While the permutation method determines significance via comparison to an estimated null distribution, both the homogeneous and heterogeneous bootstrap construct estimates of the observed distribution against which the null hypothesis is tested. This results in a series of often highly correlated test statistics, raising concerns that are typically associated with multiple testing.

The method for controlling FWER presented in **?** begins on the assumption that adjacent test statistics in a densely sampled time series will be highly correlated. That is, it will be assumed that the test statistics have null standard normal distribution $T_t \sim N(0, 1)$, but that the sequence of statistics themselves unfolds with an AR(1) process

$$T_t = \rho T_{t-1} + \epsilon_t, \tag{4.13}$$

where $\epsilon_t \sim N(0, (1-\rho^2))$. This gives the conditional distribution

$$T_t | T_{t-1} \sim N(\rho T_{t-1}, 1-\rho^2), \tag{4.14}$$

with the joint distribution of T_t and T_{t-1} being bivariate normal with mean 0, a variance of 1, and correlation ρ. From this, they show that for some α^*, the true FWER α under the null hypothesis can be expressed as

$$1 - P\left(\bigcap_{t=1}^{T} I_t\right) = 1 - P(I_1)\prod_{t=2}^{T} P(I_t|I_{t-1}) = 1 - P(I_1)P(I_t|I_{t-1})^{T-1}, \tag{4.15}$$

where I_t is the event that $|T_t| \leq z_{(1-\frac{\alpha^*}{2})}$. The correction is made by finding the nominal significance α^* that produces the desired FWER of α. Of note here, no adjustment to the nominal α is needed when the tests are perfectly correlated. Conversely, when the tests are perfectly independent, Equation 4.15 reduces to the standard Bonferonni correction. This correction is implemented in `bdots` and is also provided outside of the bootstrapping process with the function `bdots::p_adjust` using `method = "oleson"`.

4.3 FWER Simulations

We now go about comparing the family-wise error rate of the three methods just described. In doing so, we will consider several conditions under which the observed subject data may have been generated or fit. This includes generating data with both a homogeneous and heterogeneous means assumption, generating data with and without autocorrelated errors, and fitting data with and without an AR(1) assumption. In addressing performance under paired and unpaired conditions, we have included two distinct but reasonable instances in which the data may be paired, which we will elaborate further on shortly. However, as paired data becomes difficult to define under the homogeneous means assumption (in which all subjects are, in a sense, "paired"), we will omit these settings from our final simulations. In all, this gives us sixteen different arrangements which we will examine for their family-wise error rates using each of the three methods previously described.

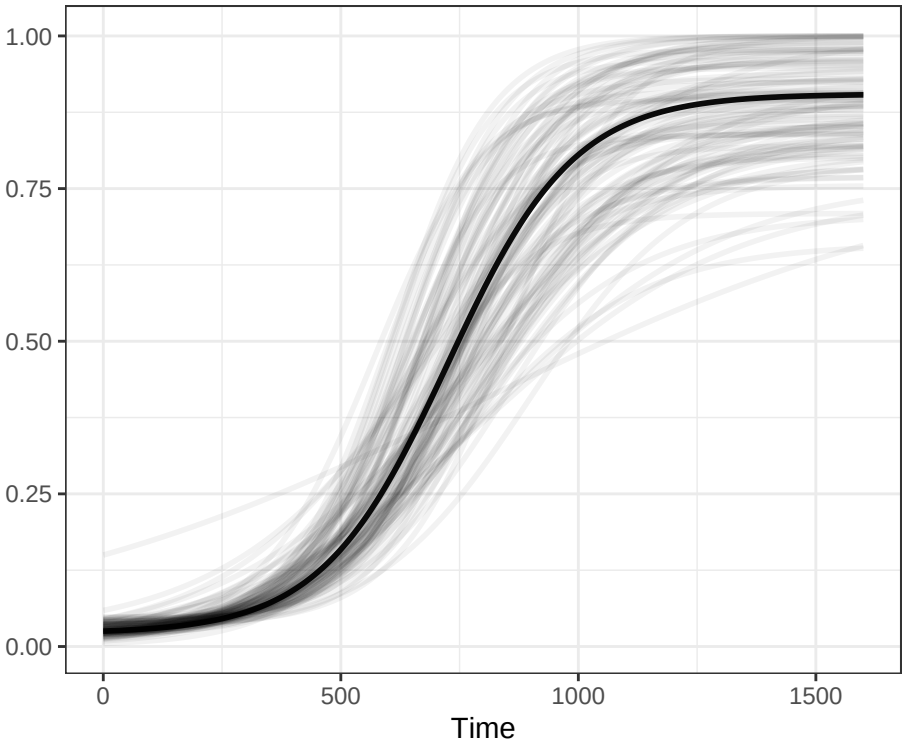


Figure 4.1: 50 samples from the generating distribution of the four-parameter logistic in Equation 4.16 used for testing FWER

4.3.1 Data Generation

Data was generated according to Equation 4.1, with the parametric function $f(t|\theta)$ belonging to the family of four-parameter logistic curves defined:

$$f(t|\theta) = \frac{p-b}{1+\exp\left(\frac{4s}{\text{p}-b}(x-t)\right)} + b \tag{4.16}$$

where $\theta = (p, b, s, x)$, the peak, baseline, slope, and crossover parameters, respectively.

We further assume that each group drew subject-specific parameters from a multivariate normal distribution, with subject $i = 1, \ldots, N$ in group $g = 1, \ldots, G$ following the distribution in Equation 4.6. These parameters are then used to simulate empirical data according the the mean and error structures for each simulation setting, which we detail next.

Mean Structure

In all of the simulations presented, the distribution of parameters used in Equation 4.6 was empirically determined from data on normal hearing subjects in the VWP [**?**]. Parameters used were those fit to fixations on the Target, following the functional form of Equation 4.16. A visual depiction of the distribution of these curves is given in Figure 4.1.

Under the homogeneous means assumption, we set $\theta_i = \theta_j$ for all subjects i, j, assuring that each of the subjects' observations is derived from the same mean structure, differing only in their observed error structure.

Error Structure

The error structure is of the form

$$e_{it} = \phi e_{i,t-1} + w_{it}, \quad w_{it} \sim N(0, \sigma) \tag{4.17}$$

where the w_{it} are iid with $\sigma = 0.025$. ϕ corresponds to an autocorrelation parameter and is set to $\phi = 0.8$ when the generated data is to be autocorrelated and set to $\phi = 0$ when we assume the errors are all independent and identically distributed.

Paired Data

As we previously noted, paired data is only a sensible condition under the assumption of heterogeneous means, and we limit our consideration to that case. Further, we observe that in the construction of paired data, there are two methods that seem reasonable, and we employ both of them here.

In considering the construction of paired data for subject i, the first method proceeds as follows: we begin by drawing parameters θ_i from Equation 4.6. Denote this θ_{i1} to indicate that this is the parameter estimate for subject i in group 1. We then simulate observed data according to Equation 4.1. To create the paired data, we then set $\theta_{i2} = \theta_{i1}$ and again simulate observed data according to Equation 4.1. Under this first method, the generating parameters between groups are *identical*, with the only differences between the simulated data being that contributed by the error

term.

In the second method of obtaining paired data, we proceed as in the first, except now letting

$$\theta_{i2} = \theta_{i1} + N(0, 0.05 \cdot V_g). \tag{4.18}$$

This adds a small amount of random noise between paired parameters, simulating the degree of variability that may normally be found between conditions, even when there is no true effect. Accommodating this phenomenon is relevant in situations in which the data gathering mechanism has imperfect reliability, as in the case of the VWP. It is also relevant because, as we will show, the homogeneous bootstrap is highly sensitive to these assumptions, with potentially disastrous results when these conditions are not precisely met. To avoid potential confusion, the results for each of these will be presented separately.

Each set of conditions generates two groups, with $n = 25$ subjects in each group, with time points $t = 0, 4, 8, \ldots, 1600$ in each trial and with 100 simulated trials for each subject. Columns in the tables indicate homogeneity of means assumption, whether or not an AR(1) error structure was used in constructing the data, and if autocorrelation was specified in the fitting function. The last conditions help assess the impact of correctly or incorrectly identifying the type of error when conducting an analysis in `bdots`. Finally, results will be separated by paired status. Each simulation was conducted 1000 times, with the proportion of simulations in which a significance difference was incorrectly identified used to determine the family-wise error rate.

4.3.2 Results

We consider the efficacy the methods under each of the simulation settings with an analysis of the family-wise error rate (FWER) and the median per-comparison error rate. The first of these details the proportion of simulations under each condition that marked *at least* one time point as being significantly different between the two groups. This is critical is understanding each method's ability to correct adjust for the multiple testing problem associated with testing each of the observed time points. The results for the unpaired data are presented in Table 4.1, while those

for each of the paired variations are presented in Tables 4.2 and 4.3.

Complimenting the FWER estimate is an estimate of the median per-comparison rate. For each time point across each of the simulations, we computed the proportion of instances in which that time was incorrectly determined to have a significant difference. The median of these values across all time points is what is considered. This metric gives a sense of magnitude to the otherwise binary FWER; for example, a situation in which there was a high FWER and low median per-comparison rate would indicate that the type I error within a particular time series would be sporadic and impact limited regions. Large median per-comparison rates indicate that large swaths of a time series frequently sustain type I errors. The median per-comparison rates for unpaired simulations are presented in Table 4.4 and each of the paired simulations in Table 4.5 and Table 4.6.

4.3.2.1 FWER

There are a few things of immediate note when considering the results of Table 4.1. First, we see from the first two settings of the unpaired simulations that the FWER for the homogeneous bootstrap are consistent with those presented in **?**, confirming the importance of specifying the existence of autocorrelation in the `bdots` fitting function when autocorrelated error is present. By contrast, this is far less of a concern when using the heterogeneous bootstrap or permutation testing, both of which maintain a FWER near the nominal alpha, regardless of whether or not the error structure was correctly identified. This continues to be true under the homogeneous mean assumption when the true error structure is not autocorrelated.

The most striking results of this, however, appear when the data generation assumes a heterogeneous mean structure. While both the heterogeneous bootstrap and the permutation test maintain a FWER near the nominal alpha, the homogeneous bootstrap fails entirely, with a FWER > 0.9 in all cases.

Het. Means	AR(1) Error	AR(1) Specified	Hom. Boot	Het. Boot	Perm.
No	Yes	Yes	0.09	0.00	0.06
No	Yes	No	0.84	0.06	0.14
No	No	Yes	0.12	0.01	0.08
No	No	No	0.14	0.00	0.05
Yes	Yes	Yes	0.94	0.05	0.05
Yes	Yes	No	0.99	0.07	0.07
Yes	No	Yes	1.00	0.08	0.05
Yes	No	No	0.99	0.05	0.04

Table 4.1: FWER for logistic function (unpaired)

We turn our attention now to simulations with paired data. Table 4.2 gives the results under the construction assuming *identical* parameters between groups. Note that despite the simulation having a heterogeneous mean structure, the FWER associated with the homogeneous bootstrap is comparable to that of the permutation test (though both with elevated FWER rates). Because these data are paired, the observed variability within each group has no effect on the distribution of paired differences. This is in stark contrast to the results presented in Table 4.3, where a small amount of variability was added to the paired set of parameters. What is most relevant here is that both the heterogeneous bootstrap as well as the permutation test are robust to these small differences in the paired setting, while the results associated with the homogeneous bootstrap are radically different. This serves to demonstrate how sensitive the homogeneous bootstrap is to such a rigid set of underlying assumptions.

Het. Means	AR(1) Error	AR(1) Specified	Hom. Boot	Het. Boot	Perm.
Yes	Yes	Yes	0.10	0.00	0.12
Yes	Yes	No	0.75	0.06	0.12
Yes	No	Yes	0.12	0.00	0.11
Yes	No	No	0.11	0.01	0.13

Table 4.2: FWER for logistic function (paired, identical parameters)

Het. Means	AR(1) Error	AR(1) Specified	Hom. Boot	Het. Boot	Perm.
Yes	Yes	Yes	0.48	0.04	0.10
Yes	Yes	No	0.93	0.07	0.12
Yes	No	Yes	0.81	0.04	0.08
Yes	No	No	0.81	0.07	0.09

Table 4.3: FWER for logistic function (paired, added noise)

4.3.2.2 Median per-comparison error rate

We next consider the median per-comparison error rate, which offers some insight into the FWER. In particular, consider the situation in which in Table 4.4, in the fourth row we see a median per-comparison error rate of 0.00 for the homogeneous bootstrap, despite Table 4.1 indicating a FWER of 0.15. This is a consequence of the majority of the type I errors occuring in a relatively limited region. In contrast, the median per-comparison error rate of the homogeneous bootstrap under the assumption of heterogeneity suggests that the type I errors are widespread and not limited to any particular area.

It is also worth commenting on the permutation test median per-comparison error rate in

Table 4.4; combined with a FWER near the nominal 0.05, that these values are not identically 0 suggests that errors are likely distributed across the entire range rather than limited to a small area.

Het. Means	AR(1) Error	AR(1) Specified	Hom. Boot	Het. Boot	Perm.
No	Yes	Yes	0.02	0.00	0.01
No	Yes	No	0.31	0.01	0.02
No	No	Yes	0.01	0.00	0.01
No	No	No	0.00	0.00	0.01
Yes	Yes	Yes	0.59	0.01	0.01
Yes	Yes	No	0.83	0.02	0.01
Yes	No	Yes	0.84	0.02	0.01
Yes	No	No	0.82	0.01	0.01

Table 4.4: Median per-comparison error rate for unpaired data

Regarding paired data, we see again a similar situation play out as we did with the FWER. Table 4.5 treats the paired setting with identical parameters, and we see median per-comparison error rates consistent with the lower FWER from Table 4.2. And again, Table 4.6 demonstrates a more robust performance for both the heterogeneous bootstrap and permutation test, relative to that of the homogeneous bootstrap.

Het. Means	AR(1) Error	AR(1) Specified	Hom. Boot	Het. Boot	Perm.
Yes	Yes	Yes	0.03	0.00	0.02
Yes	Yes	No	0.27	0.01	0.02
Yes	No	Yes	0.01	0.00	0.02
Yes	No	No	0.01	0.00	0.02

Table 4.5: Median per-comparison error rate for paired data (identical parameters)

Het. Means	AR(1) Error	AR(1) Specified	Hom. Boot	Het. Boot	Perm.
Yes	Yes	Yes	0.14	0.01	0.02
Yes	Yes	No	0.46	0.01	0.03
Yes	No	Yes	0.44	0.01	0.01
Yes	No	No	0.41	0.02	0.02

Table 4.6: Median per-comparison error rate for paired data (added noise)

4.3.3 Discussion

It was the control of the family-wise error rate in the context of densely sampled, highly correlated tests that first motivated the development of `bdots`. And while the results presented in this section indeed demonstrate control of the FWER under a strict set of assumptions, they also highlight the consequences when these conditions are not met. With the introduction of both the heterogeneous bootstrap and the permutation test, we have offered two alternatives that are robust to a wide variety of situations while maintaining performance similar to that of the homogeneous bootstrap in the best of cases. This is but one half of the question, however. In the following section, we present two separate simulations to determine if the robustness acquired is as the cost

of significant power.

4.4 Power Simulations

In our assessment of power among the presented methods, we examine two distinct simulations, each associated with a different aspect of the question of identifying temporal differences. The first of these seeks to identify the relationship between power and effect size, which is done by comparing two simple piecewise linear functions as we will describe shortly. The goals of the second simulation, which is done in the context of VWP data, are twofold: first, to assess and verify the power of the new methods in paired and unpaired settings, and second, to investigate the relationship between power and effect size in light of between-subject variability.

4.4.1 Piecewise Linear Power

To better understand the relationship between effect size and power, we have created a simple example consisting of two simulated experimental groups whose mean structure is a simple piecewise linear function defined on the interval $(-1, 1)$ as follows:

$$y = \begin{cases} b & x < 0 \\ mx + b & x \geq 0 \end{cases} \tag{4.19}$$

The set of parameters drawn from each subject include a baseline parameter, b, as well as a slope parameter, m, which were each drawn from a univariate normal distribution. To distinguish experimental groups, one designated as being the "Effect" group, other other being "No Effect", we set the slope parameter of the "No Effect" group to be identically $m = 0$, while the "Effect" group had a slope parameter that was normally distributed with a mean value $\mu_m = 0.25$. Both groups drew their baseline parameter, b, from a normal distribution with mean $\mu_b = 0$. As a consequence of this, and following from Equation 4.19, there should be no difference between groups when $x < 0$, with an effect size for the "Effect" group being mx for all $x \geq 0$. This was done for two reasons. First, by intentionally including an interval in which there is no difference, we are able

to observe power in the context of type I error as well. That is, achieving a higher power when $x \geq 0$ will be of little value if differences are equally (and incorrectly) identified when $x < 0$. The second reason for having set our problem up this way was to give us the ability to observe changes in power in conjunction with changes in effect size, mediated by increasing values in x. A visual depiction of each of these groups is given in Figure 4.2.

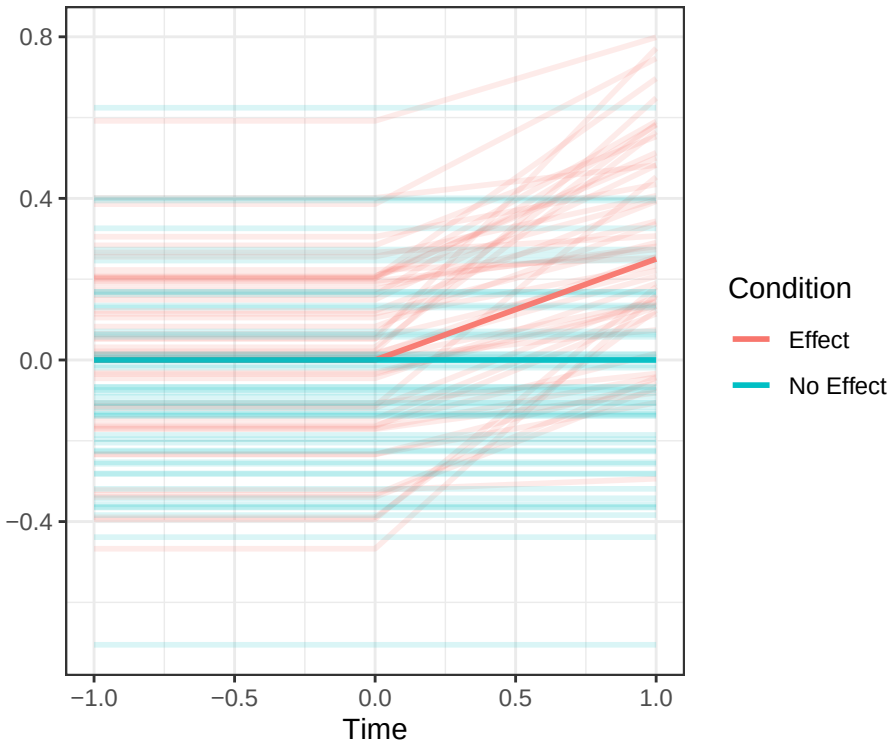


Figure 4.2: 50 samples from the generating distributions of each group in Equation 4.19. The distribution of values is the same for each group for all $t < 0$

For these simulations, we limited consideration to three possible scenarios: first, we assumed the conditions presented in **?**, assuming homogeneity between subject parameters and an AR(1) error structure, with the model fitting performed assuming autocorrelated errors. For the remaining scenarios, we assumed heterogeneity in the distribution of subject parameters, simulated with and without an AR(1) error structure. In both of these last two scenarios, we elected to *not* fit the model assuming autocorrelated errors. This was for two reasons: first, simulations exploring the type I

error rate suggested that models fit with the autocorrelation assumption tended to be conservative. Second, and given the results of the first, this makes setting the assumption of autocorrelation to FALSE in `bdots` seem like a sensible default, and as such, it would be of interest to see how the model performs in cases in which there is autocorrelated error that is not accounted for. Simulation results for a number of other settings are handled in the appendix.

For each subject, parameters for their mean structure given in Equation 4.19 were drawn according to their group membership and fit using `bdots` on the interval (-1,1). Time windows in which the groups differed were identified using the homogeneous bootstrap, heterogeneous bootstrap, and permutation testing. By including the interval (-1,0) in which the null hypothesis was true, we are able to mitigate the effects of over-zealous methods in determining power, and we present the results in the following way: any tests in which a difference was detected in (-1,0) was marked as having a type I error, and the proportion of simulations in which this occurred for each method is reported as the FWER in the column labeled α. The next column, β, is the type II error rate, indicating the proportion of trials in which no differences were identified over the entire region. The last Greek-letter column is $1 - \beta - \alpha$, a modified power statistic indicating the proportion of tests in which a difference was correctly identified. The remaining columns relate to this modified power column, giving a partial summary of the earliest onset time of detection. As a true difference occurs on the interval $t > 0$, smaller values indicate greater power in detecting differences. Finally, a plot giving the power at each time point is given in Figure 4.3. This plot represents the true power, though note that it does not take into account the rate at which these regions were identified in conjunction with a type I error rate.

4.4.1.1 Results

The results of the power simulation are presented in Table 4.7. We begin by considering the case in which we assumed a homogeneous mean structure with autocorrelated errors, matching the conditions in which the homogeneous bootstrap was first presented. Notably, we find that the permutation method demonstrates the greater power, with the median onset time just under

that of the homogeneous bootstrap. This is at the expense of a larger FWER, though still below the nominal level. Alternatively, the heterogeneous bootstrap maintains a similar FWER as the homogeneous bootstrap at the cost of power. The remaining settings tell a similar story with the exception of the homogeneous bootstrap which continues to demonstrate unacceptable FWER under the heterogeneous means assumption. As to the effect of incorrectly specifying an AR(1) error structure when comparing the last two settings, note that there tends to be very little effect between the heterogeneous bootstrap and permutation, with neither performing consistently better or worse at any of the quantiles given.

Method	Heterogeneity	AR(1)	α	β	1 - α - β	1st Qu.	Median	3rd Qu.
Hom. Boot	No	Yes	0.00	0.00	1.00	0.025	0.030	0.035
Het. Boot	No	Yes	0.00	0.00	1.00	0.035	0.040	0.045
Perm	No	Yes	0.03	0.00	0.97	0.020	0.025	0.030
Hom. Boot	Yes	No	0.95	0.00	0.05	0.005	0.008	0.010
Het. Boot	Yes	No	0.00	0.01	0.98	0.260	0.330	0.480
Perm	Yes	No	0.04	0.00	0.95	0.245	0.325	0.452
Hom. Boot	Yes	Yes	0.94	0.00	0.06	0.005	0.013	0.015
Het. Boot	Yes	Yes	0.01	0.01	0.98	0.270	0.370	0.465
Perm	Yes	Yes	0.04	0.00	0.96	0.245	0.365	0.440

Table 4.7: caption

The results in Table 4.8 are a summary of all of the methods found by taking the mean of each of the results presented. This is intended to interrogate the performance of each of these methods when underlying assumptions are unknown or unspecified. We find a robust performance for each of the new methods presented, maintaining a reasonable relationship between FWER and power. The metrics associated with homogeneous bootstrap are perhaps a bit misleading here as they appear to demonstrate exceptional power, though at the cost of unacceptable type I error.

Method	α	β	$1-\alpha-\beta$	1st Qu.	Median	3rd Qu.
Hom. Bootstrap	0.772	0.000	0.228	0.011	0.016	0.021
Het. Bootstrap	0.002	0.097	0.901	0.328	0.419	0.546
Permtuation	0.027	0.038	0.935	0.294	0.424	0.556

Table 4.8: Summary of methods for Type II error

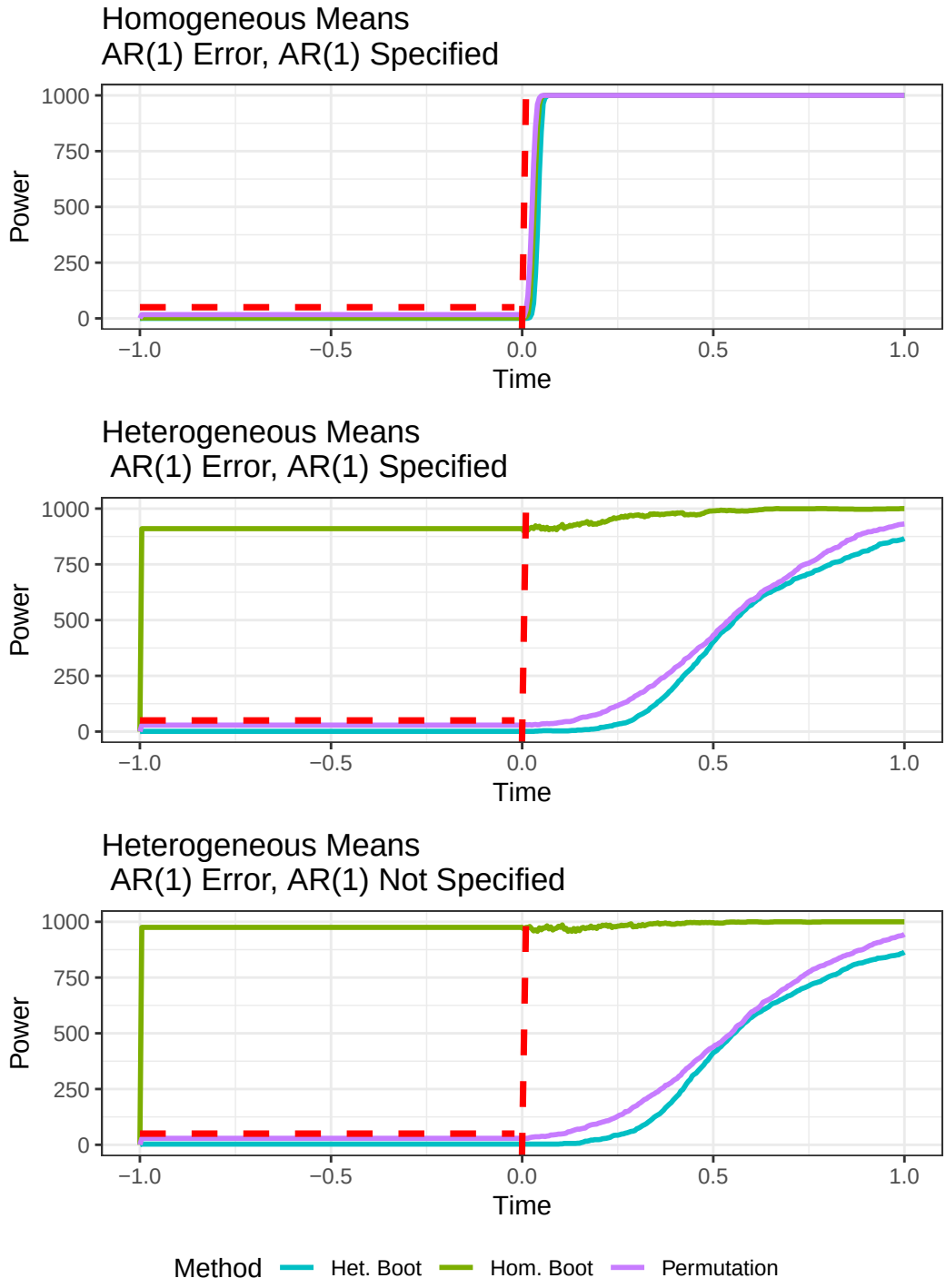


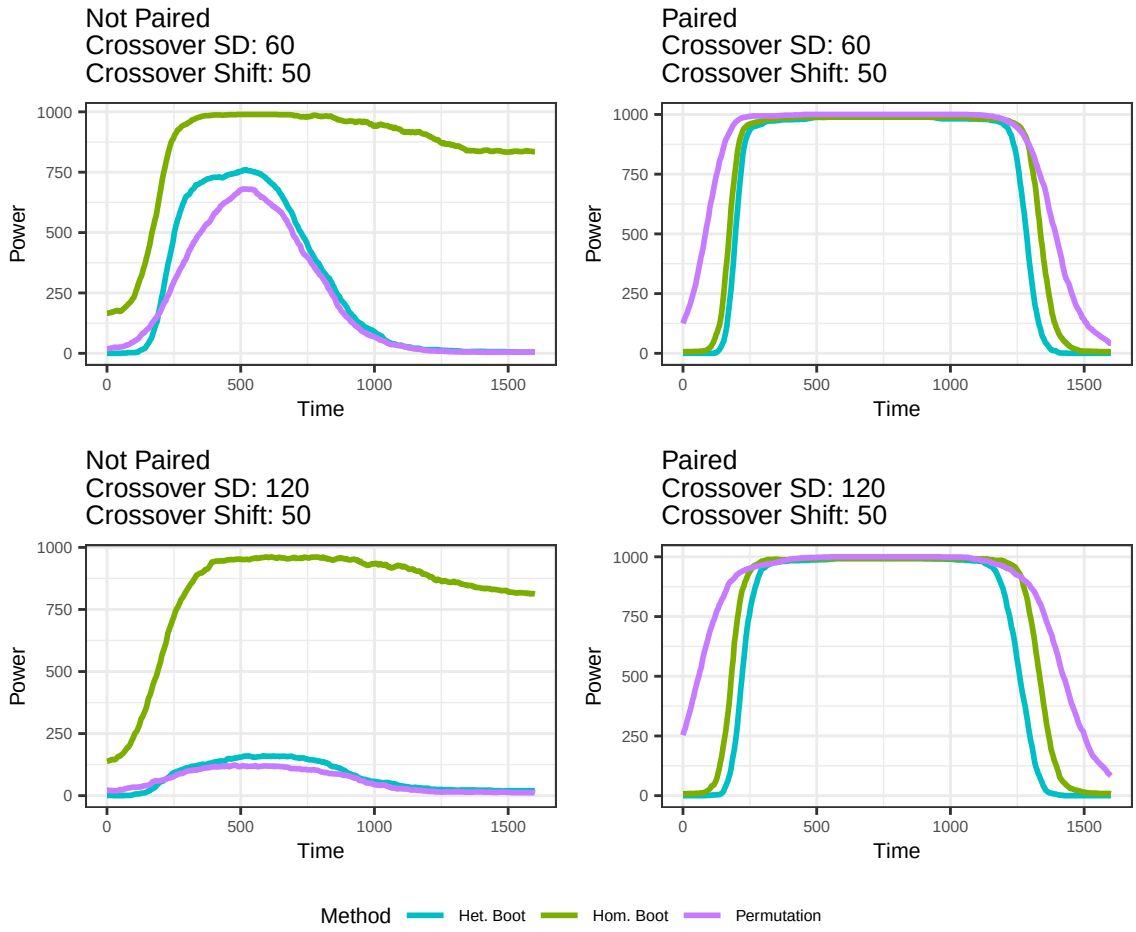
Figure 4.3: Observed power of each of the methods at each time in (-1,1)

4.4.2 Logistic Shift Power

The final set of simulations we consider seek to address concerns related to power in a paired setting, as well as to investigate the relationship between effect size and variability. We proceed with simulation settings similar to those in our investigation of the type I error in that each of the two groups under consideration draw from an empirical distribution of parameters associated with the four parameter logistic function. Unlike the situation in determining the FWER, here differences are introduced between groups by changing the crossover parameter, indicating the inflection point of the function. To simulate various effect sizes, we investigate changes in crossover of 50 and 150. Further, to determine the effect of between subject variability on the identification of differences, we have set the standard deviation of the crossover parameter from the empirical distribution to take values of either 60 or 120. Thus, the settings we will be presenting include small and large changes in light of small or large variability. Data was generated with iid errors and fit without specifying AR(1) correlation in `bdots`. Finally, each of these settings will be run as paired and unpaired data, with parameters for the paired data being identical between groups (i.e., no added variability).

Consider first, for example, the results presented in Figure 4.4, each of which demonstrate the observed power at each time point when the crossover parameter between group was shifted by 50. Predictably, we see greater power in the unpaired setting when the standard deviation of the crossover parameter is smaller, relative to the shift. By contrast, in the paired setting we observe little difference in power as a function of parameter variability. This is also to be expected as here we are concerned with the variability of the *difference* rather than of each of the groups, and these results confirm that this is indeed the case.

A similar set of conclusions can be seen by considering the observed power presented in Figure 4.5, where we observe the results when the crossover parameter is shifted by 150. Similar to the prior case, we see greater power associated with the smaller standard deviation in the unpaired case, with power in the paired case performing similarly. And finally, by comparing the results between Figures 4.4 and 4.5 we confirm in all cases that a greater effect size is associated with



.

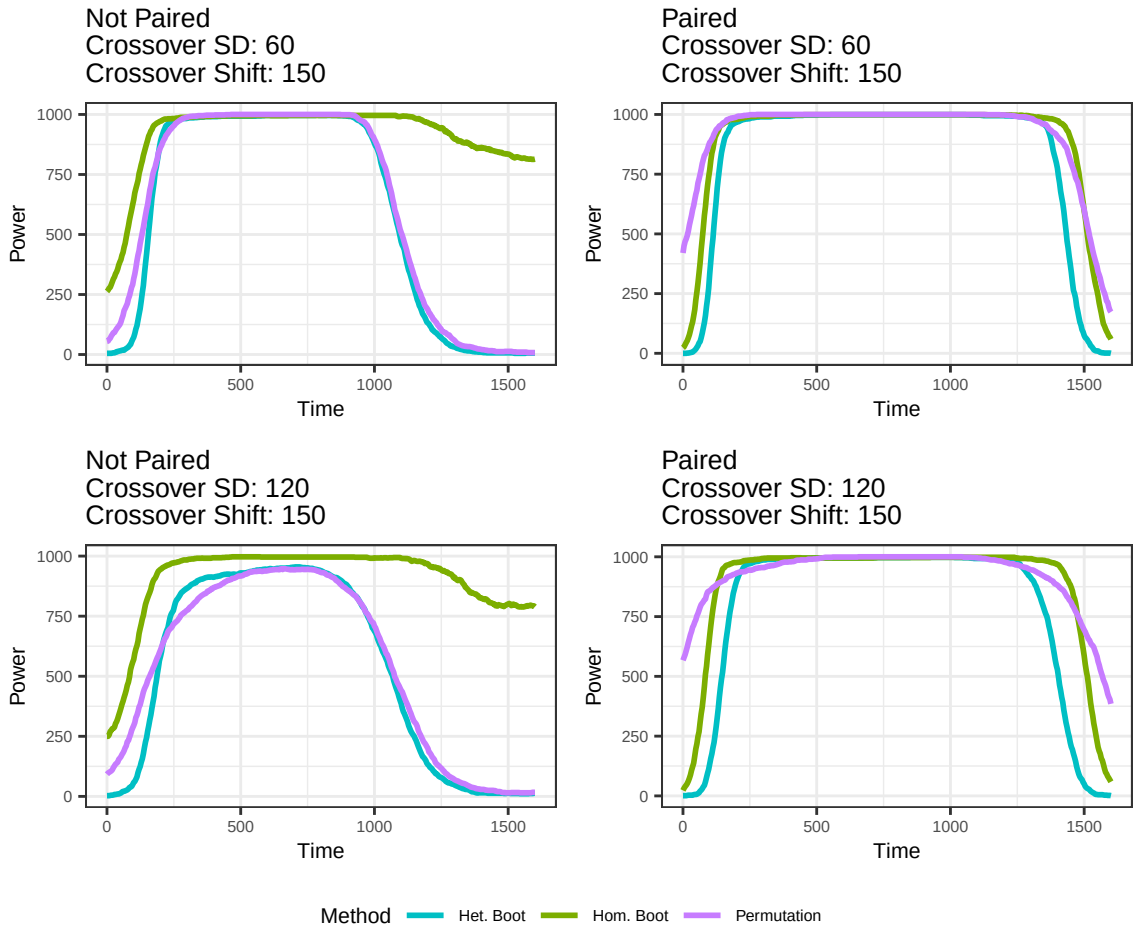
Figure 4.4: Observed power following a small shift in the crossover parameter

greater power.

4.5 Discussion

We set out both to interrogate the validity of the homogeneous bootstrap assumptions and to propose two alternative methods that would be more robust under a greater variety of assumptions. In doing so, we demonstrated conclusively the utility of the heterogeneous bootstrap and permutation tests while also highlighting a major shortcoming of the original. It's worth noting, however, that the FWER adjustment proposed in **?** is still valid, if not slightly conservative, and with power similar to that of the permutation method.

In light of the results presented, one issue of concern is addressing the fact that a version of



.

Figure 4.5: Observed power following a large shift in the crossover parameter

`bdots` with the homogeneous mean assumption was presented in 2018 and remained accessible on CRAN until the end of 2022. This has implications for the number of papers in which `bdots` may have demonstrated significance between groups when the underlying assumptions of homogeneous mean structure did not hold, as is likely the case in all instances related to the VWP. Concurrent with this issue is the issue of identifying current users of `bdots` of this change, as results found only a month ago will be profoundly different than what is seen today. At present, I am not sure the best way to address either of these. In either case, however, it will be prudent to remove this option from the `bdots` package all together, as there appears to be no obvious advantage to the homogeneous bootstrap over the others in terms of either controlling the FWER or obtaining power, even when the homogeneous mean structure assumption is met.

There are several limitations of the current paper that are worthy of further investigation. First, limited consideration was given to the effect of sample density on the observed type I error rate or power. As the fitting function in `bdots` simply returns a set of parameters, one could conceivably perform any of the methods presented on any arbitrary collection of points, whether or not any data were observed there. This extends itself to the condition in which subjects were sampled at heterogeneous time points, as may be the case in many clinical settings. What impact this may have on how to best handle these cases remains open for exploration. It is also worth investigating in greater detail what impact the re-drawing of subject specific parameters from their respective distributions has on both the FWER and power, as in several of the simulations the observed FWER was much lower than the nominal level. Particularly in the case of the permutation method which is *not* seeking to estimate the group distributions, it may be worthwhile to see if a favorable trade can be made to increase the resulting power.

We conclude by noting that `bdots` is now equipped with two methods to effectively control the FWER when assessing the differences in time series under a greater set of underlying assumptions, including those involving the presence of highly correlated test statistics. Further, both methods presented are robust to misspecification of the error structure while maintaining an acceptable FWER and adequate power.

Appendix

4.6 Full Piecewise Power Simulations

Here we present the full collection of the linear piecewise power simulations, which, in addition to those given in Table 4.7 includes cases for heterogeneous means where autocorrelation is specified when fitting with `bdots`. This is indicated in the "AR(1) Specified" column. Notably from this table, we see that in the case in which the true errors are iid, there is no measurable effect on power when an autocorrelated structure is incorrectly specified. This is similar to the opposite situation, in which the true error does have an AR(1) structure. In this case, we observe a marginal benefit to correctly specifying an AR(1) structure. This may in fact make retaining the AR(1)

assumption a reasonable default in the `bdots` package.

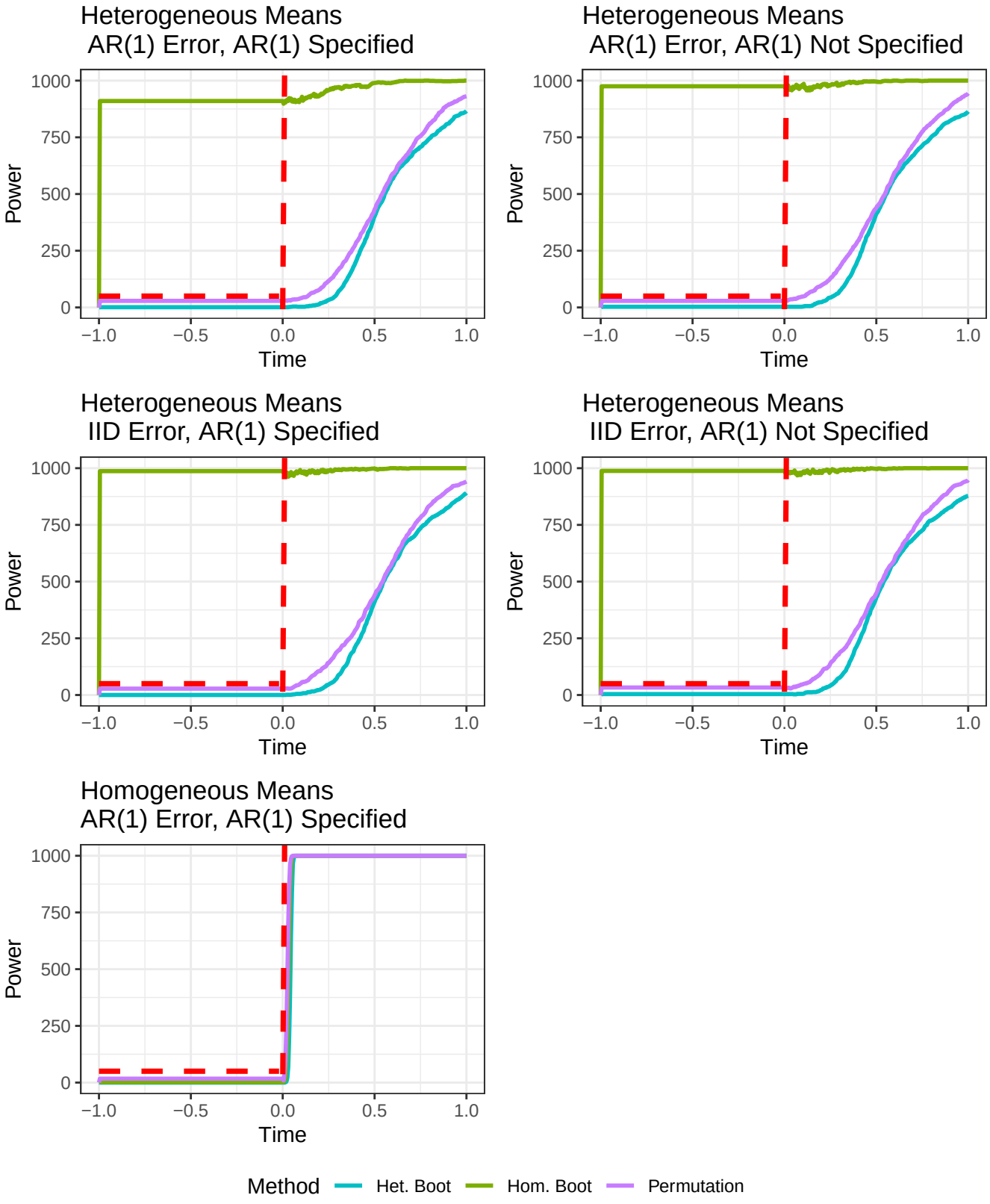


Figure 4.6: Power plots in time for each of the simulation settings. Note that in the heterogeneous means case, there is little difference when AR(1) is incorrectly specified

Method	Heterogeneity	AR(1) Error	AR(1) Specified	α	β	1 - α - β	1st Qu.	Median	3rd Qu.
Hom. Boot	No	Yes	Yes	0.00	0.00	1.00	0.025	0.030	0.035
Het. Boot	No	Yes	Yes	0.00	0.00	1.00	0.035	0.040	0.045
Perm	No	Yes	Yes	0.03	0.00	0.97	0.020	0.025	0.030
Hom. Boot	Yes	No	No	0.95	0.00	0.05	0.005	0.008	0.010
Het. Boot	Yes	No	No	0.00	0.01	0.98	0.260	0.330	0.480
Perm	Yes	No	No	0.04	0.00	0.95	0.245	0.325	0.452
Hom. Boot	Yes	No	Yes	0.98	0.00	0.02	0.005	0.008	0.010
Het. Boot	Yes	No	Yes	0.00	0.01	0.99	0.261	0.350	0.475
Perm	Yes	No	Yes	0.04	0.00	0.96	0.225	0.335	0.440
Hom. Boot	Yes	Yes	No	0.94	0.00	0.06	0.005	0.013	0.015
Het. Boot	Yes	Yes	No	0.01	0.01	0.98	0.270	0.370	0.465
Perm	Yes	Yes	No	0.04	0.00	0.96	0.245	0.365	0.440
Hom. Boot	Yes	Yes	Yes	0.83	0.00	0.17	0.021	0.032	0.040
Het. Boot	Yes	Yes	Yes	0.00	0.01	0.98	0.250	0.330	0.450
Perm	Yes	Yes	Yes	0.03	0.00	0.97	0.223	0.335	0.428

Table 4.9: Power for full piecewise linear simulation

CHAPTER 5
CRAN VIGNETTES

Included here are the collection of vignettes that are included with the `bdots` packages. In order, these are:

1. **`bdots`** This is the primary vignette used to introduce new users to the package. It includes information on fitting parametric functions to subject data, a brief tutorial on refitting, and a description of the bootstrapping process as well as the `bdots` formula syntax

2. **Refitting with Saved Parameters** This vignette illustrates how to save the fitted coefficients from the fitting step (or after refitting) and how to load them back into R to recreate a fitted `bdotsObj` object. This has been primarily used by users who have created subject-specific parameters in other software (i.e., MATLAB) who wish to import them to `bdots`

3. **Correlations** This vignette details the `bdotsCor` function to find the correlation of a fixed value (i.e., vocabulary scores or IQ tests) with the group fitted curves at each time point

4. **User Curve Functions** This vignette offers detailed instructions for users who wish to create and import their own custom parametric curve functions

bdots

```
library(bdots)
#> Loading required package: data.table
#>
#> Attaching package: 'bdots'
#> The following object is masked from 'package:data.table':
#>
#>     rbindlist
```

Overview

This vignette walks through the use of the bdots package for analyzing the bootstrapped differences of time series data. The general workflow will follow three steps:

1. **Curve Fitting** During this step, we define the type of curve that will be used to fit our data along with variables to be used in the analysis
2. **Curve Refitting** Often, some of the curves returned from the first step have room for improvement. This step allows the user to either quickly attempting refitting a subset of the curves from step one or to manually make adjustments themselves
3. **Bootstrap** Having an adequate collection of curves, this function determines the bootstrapped difference, along with computing an adjusted alpha to account for AR1 correlation

This process is represented with three main functions, `bdotsFit -> bdotsRefit -> bdotsBoot`

This package is under active development. The most recent version can be installed with `devtools::install_github("collinn`

Fitting Step

For our example, we are going to be using eye tracking data from normal hearing individuals and those with cochlear implants using data from the Visual Word Paradigm (VWP).

```
head(cohort_unrelated)
#>    Subject Time DB_cond Fixations LookType Group
#> 1:       1    0      50  0.011364   Cohort    50
#> 2:       1    4      50  0.011364   Cohort    50
#> 3:       1    8      50  0.011364   Cohort    50
#> 4:       1   12      50  0.011364   Cohort    50
#> 5:       1   16      50  0.022727   Cohort    50
#> 6:       1   20      50  0.022727   Cohort    50
```

The `bdotsFit` function will create a curve for each unique permutation of `subject`/`group` variables. Here, we will let `LookType` and `DB_cond` be our grouping variables, though we may include as many as we wish (or only a single group assuming that it has multiple values). See `?bdotsFit` for argument information.

```
fit <- bdotsFit(data = cohort_unrelated,
                subject = "Subject",
                time = "Time",
                y = "Fixations",
                group = c("DB_cond", "LookType"),
```

```
                curveType = doubleGauss(concave = TRUE),
                cores = 2)
```

A key thing to note here is the argument for `curveType` is passed as a function call with arguments that further specify the curve. Currently within the `bdots` package, the available curves are `doubleGauss(concave = TRUE/FALSE)`, `logistic()` (no arguments), and `polynomial(degree = n)`. While more curves will be added going forward, users can also specify their own curves, as shown here.

The `bdotsFit` function returns an object of class `bdotsObj`, which inherits from `data.table`. As such, this object can be manipulated and explored with standard `data.table` syntax. In addition to the subject and the grouping columns, we also have a `fit` column, containing the fit from the `gnls` package, a value for `R2`, a boolean indicating `AR1` status, and a final column for `fitCode`. The fit code is a numeric quantity representing the quality of the fit as such:

fitCode	AR1	R2
0	TRUE	$R2 > 0.95$
1	TRUE	$0.8 < R2 < 0.95$
2	TRUE	$R2 < 0.8$
3	FALSE	$R2 > 0.95$
4	FALSE	$0.8 < R2 < 0.95$
5	FALSE	$R2 < 0.8$
6	NA	NA

A `fitCode` of 6 indicates that a fit was not able to be made.

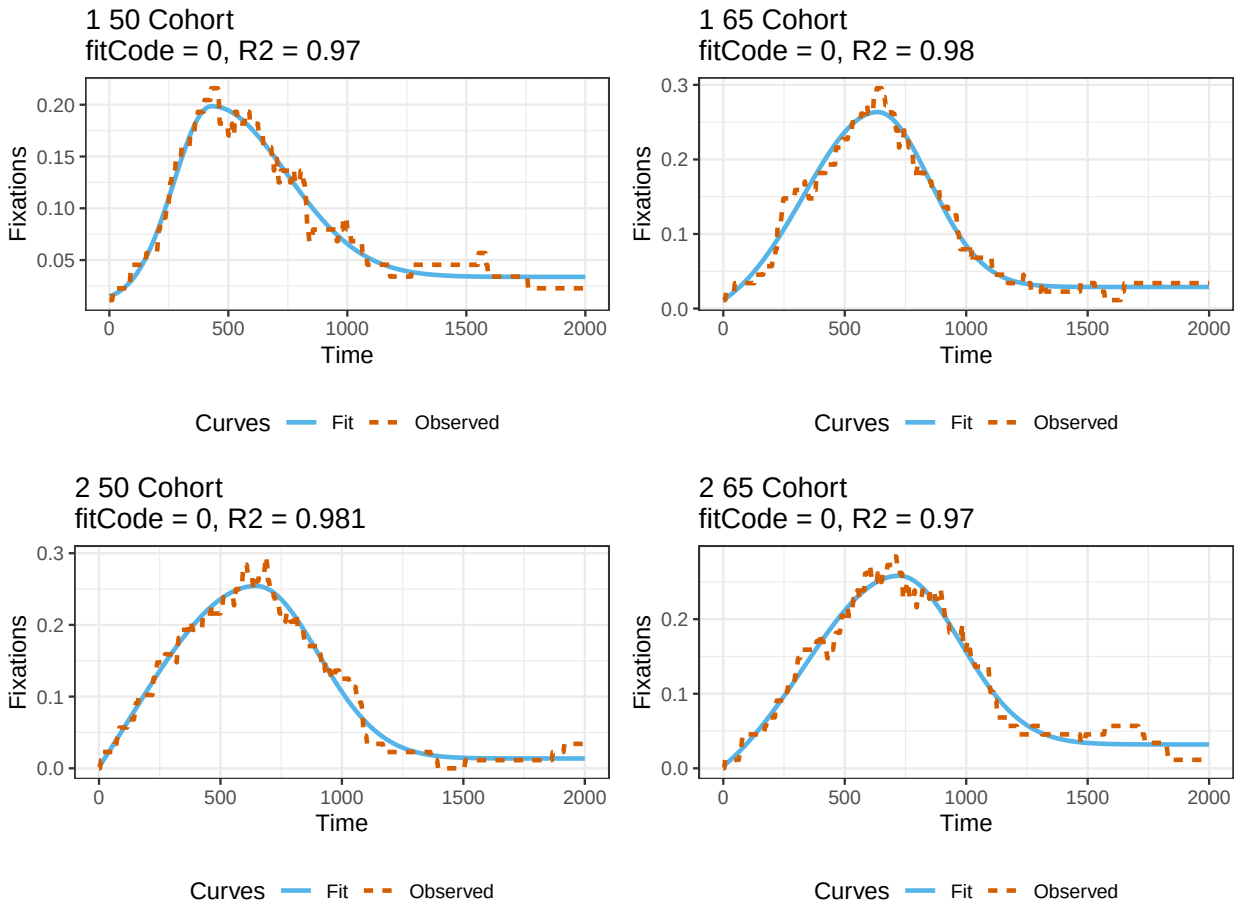
In addition to `plot` and `summary` functions, we also have a method to return a matrix of coefficients from the model fits. Because of the `data.table` syntax, we can examine subsets of this object as well

```
head(coef(fit))
#>          mu      ht   sig1   sig2      base1    base2
#> [1,] 429.76 0.19860 159.89 314.64  0.0097098 0.033761
#> [2,] 634.93 0.26350 303.81 215.38 -0.0206361 0.028924
#> [3,] 647.07 0.25438 518.96 255.99 -0.2130875 0.013682
#> [4,] 723.05 0.25821 392.95 252.94 -0.0548262 0.031973
#> [5,] 501.48 0.22477 500.85 158.42 -0.3316790 0.025227
#> [6,] 460.72 0.30677 382.73 166.08 -0.2433086 0.039922

head(coef(fit[DB_cond == 50, ]))
#>          mu      ht   sig1   sig2      base1     base2
#> [1,] 429.76 0.19860 159.89 314.64  0.0097098 0.0337611
#> [2,] 647.07 0.25438 518.96 255.99 -0.2130875 0.0136820
#> [3,] 501.48 0.22477 500.85 158.42 -0.3316790 0.0252268
#> [4,] 521.68 0.24838 270.74 209.39 -0.0385777 0.1045933
#> [5,] 553.19 0.22727 207.44 226.72 -0.0101197 0.0286633
#> [6,] 615.90 0.15877 286.21 392.57 -0.0105632 0.0076619
```

The plots for this object will compare the observed data with the fitted curve. Here is an example of the first four:

```
plot(fit[1:4, ])
```



Refitting Step

Depending on the curve type and the nature of the data, we might find that a collection of our fits aren't very good, which may impact the quality of the bootstrapping step. Using the `bdotsRefit` function, users have the option to either quickly attempt to automatically refit specified curves or to manually review each one and offer alternative starting parameters. The `fitCode` argument provides a lower bound for the fit codes to attempt refitting. The default is `fitCode = 1`, indicating that we wish to attempt refitting all curves that did not have `fitCode == 0`. The object returned is the same as that returned by `bdotsFit`.

```
## Quickly auto-refit (not run)
refit <- bdotsRefit(fit, fitCode = 1L, quickRefit = TRUE)

## Manual refit (not run)
refit <- bdotsRefit(fit, fitCode = 1L)
```

For whatever reason, there are some data will will not submit nicely to a curve of the specfied type. One can quickly remove all observations with a fit code equal to or greater than the one provided in `bdRemove`

```
table(fit$fitCode)
#>
#>  0  1  3  4  5
#> 18 14  1  1  2

## Remove all failed curve fits
refit <- bdRemove(fit, fitCode = 6L)
```

```
table(refit$fitCode)
#>
#>  0  1  3  4  5
#> 18 14  1  1  2
```

There is an additional option, `removePairs` which is `TRUE` by default. This indicates that if an observation is removed, all observations for the same subject should also be removed, regardless of fit. This ensures that all subjects have their corresponding pairs in the bootstrapping function for the use of the paired t-test. If the data are not paired, this can be set to `FALSE`.

Bootstrap

The final step is the bootstrapping process, performed with `bdotsBoot`. First, let's examine the set of curves that we have available from the first step

1. **Difference of Curves** Here, we are interested specifically in the difference between two fitted curves. For our example case here, this may be the difference between curves for `DB_cond == 50` and `DB_cond == 65` nested within either the `Cohort` or `Unrelated_Cohort` `LookTypes` (but not both).
2. **Difference of Difference Curves** In this case, we are considering the difference of two difference curves similar to the one found above. For example, we may denote the difference between `DB_cond` `50` and `65` within the `Corhort` group as $\text{diff}_{\text{Cohort}}$ and the differences between `DB_cond` `50` and `65` within `Unrelated_Corhort` as $\text{diff}_{\text{UnrelatedCohort}}$. The difference of difference function will then return an analysis of $\text{diff}_{\text{Cohort}}$ - $\text{diff}_{\text{UnrelatedCohort}}$

We can express the type of curve that we wish to fit with a modified formula syntax. It's helpful to read as "the difference of LHS between elements of RHS"

For the first type, we have

```
## Only one grouping variable in dataset, take bootstrapped difference
Outcome ~ Group1(value1, value2)

## More than one grouping variable in difference, must specify unique value
Outcome ~ Group1(value1, value2) + Group2(value3)
```

That is, we might read this as "difference of Outcome for value1 and value2 within Group1."

With our working example, we would find the difference of '$DB_cond == 50$' and `DB_cond == 65` within `LookType == "Cohort"` with

```
## Must add LookType(Cohort) to specify
Fixations ~ DB_cond(50, 65) + LookType(Cohort)
```

For this second type of curve, we specify an "inner difference" to be the difference of groups for which we are taking the difference of. The syntax for this case uses a `diffs` function in the formula:

```
## Difference of difference. Here, outer difference is Group1, inner is Group2
diffs(Outcome, Group2(value3, value4)) ~ Group1(value1, value2)

## Same as above if three or more grouping variables
diffs(Outcome, Group2(value3, value4)) ~ Group1(value1, value2) + Group3(value5)
```

For the example illustrated in (2) above, the difference $\text{diff}_{50} - \text{diff}_{65}$ represents our inner difference, each nested within one of the values for `LookType`. The "outer difference" is then difference of these between `LookTypes`. The syntax here would be

```
diffs(Fixations, DB_cond(50, 65)) ~ LookType(Cohort, Unrelated_Cohort)
```

Here, we show a fit for each

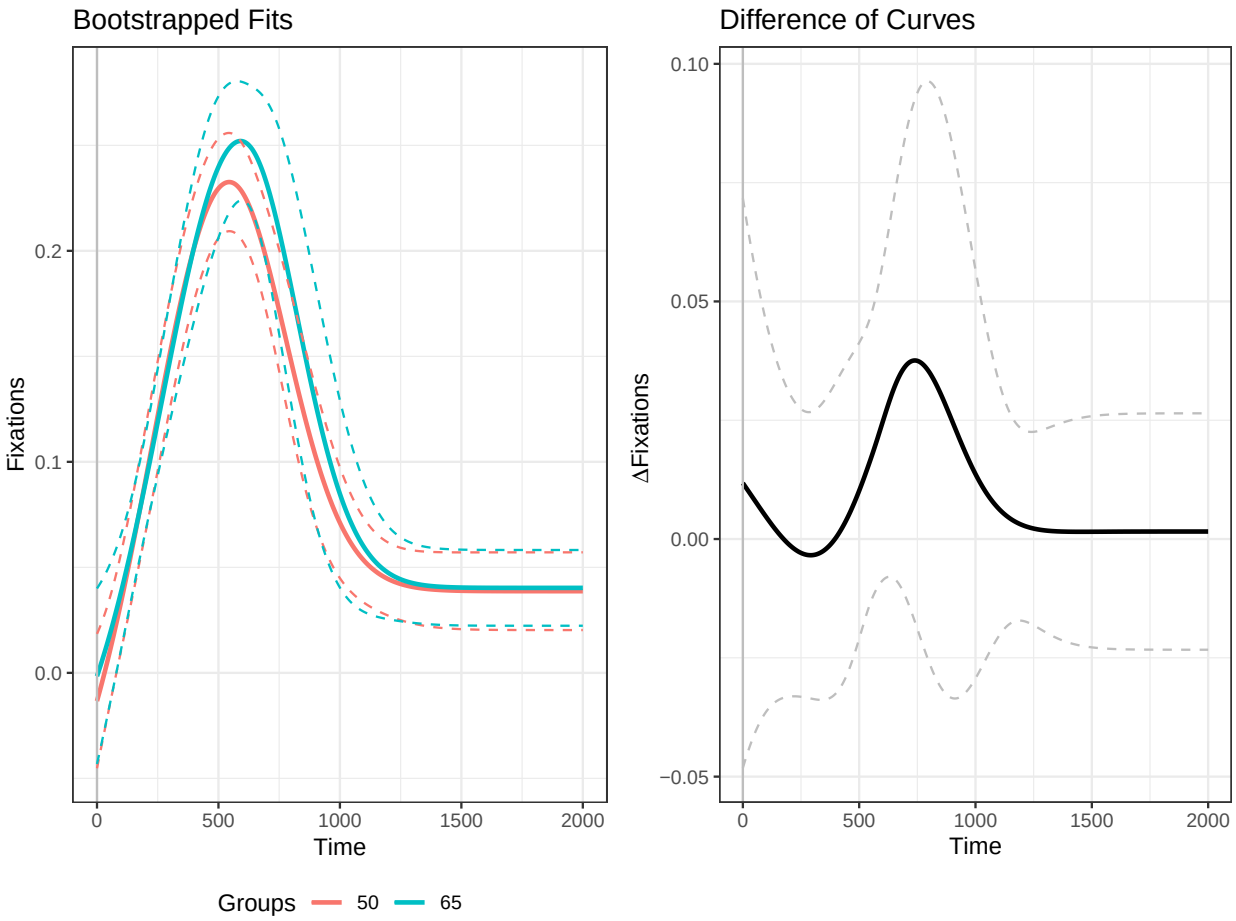
```
boot1 <- bdotsBoot(formula = Fixation ~ DB_cond(50, 65) + LookType(Cohort),
                   bdObj = refit,
                   Niter = 1000,
                   alpha = 0.05,
                   padj = "oleson",
                   cores = 2)

boot2 <- bdotsBoot(formula = diffs(Fixation, LookType(Cohort, Unrelated_Cohort)) ~ DB_cond(50, 65),
                   bdObj = refit,
                   Niter = 1000,
                   alpha = 0.05,
                   padj = "oleson",
                   cores = 2)
```

For each, we can then produce a model summary, as well as a plot of difference curves

```
summary(boot1)
#>
#> bdotsBoot Summary
#>
#> Curve Type: doubleGauss
#> Formula: Fixations ~ (Time < mu) * (exp(-1 * (Time - mu)^2/(2 * sig1^2)) * (ht - base1) + base1) + (
#> Time Range: (0, 2000) [501 points]
#>
#> Difference of difference: FALSE
#> Paired t-test: TRUE
#> Difference: DB_cond -- 50 65
#>
#> Autocorrelation Estimate: 0.99969
#> FWER adjust method: oleson
#> Alpha: 0.05
#> Adjusted alpha: 0.021774
#> Significant Intervals:
#> NULL

plot(boot1)
```


Bootstrapped Fits
Fixations
0.0
0.1
0.2
0
500
1000
1500
2000
Time
Groups
50
65
Difference of Curves
ΔFixations
0.10
0.05
0.00
−0.05
0
500
1000
1500
2000
Time

Refit with Saved Parameters

Overview

This vignette walks through using a text file of previously fit model parameters to use in the `bdotsRefit` function. This is convenient if you have already gone through the refitting process and would like to save/load the refitted parameters in a new session.

To demonstate this process, we start with fitting a set of curves to our data

```
library(bdots)

fit <- bdotsFit(data = cohort_unrelated,
                subject = "Subject",
                time = "Time",
                y = "Fixations",
                group = c("Group", "LookType"),
                curveType = doubleGauss(concave = TRUE),
                cor = TRUE,
                numRefits = 2,
                cores = 2,
                verbose = FALSE)

refit <- bdotsRefit(fit, quickRefit = TRUE, fitCode = 5)
```

From this, we can create an appropriate `data.table` that can be used in a later session

```
parDT <- coefWriteout(refit)
head(parDT)
#>    Subject Group LookType     mu      ht   sig1   sig2      base1    base2
#> 1:       1    50   Cohort 429.76 0.19860 159.89 314.64  0.0097098 0.033761
#> 2:       1    65   Cohort 634.93 0.26350 303.81 215.38 -0.0206361 0.028924
#> 3:       2    50   Cohort 647.07 0.25438 518.96 255.99 -0.2130875 0.013682
#> 4:       2    65   Cohort 723.05 0.25821 392.95 252.94 -0.0548262 0.031973
#> 5:       3    50   Cohort 501.48 0.22477 500.85 158.42 -0.3316790 0.025227
#> 6:       3    65   Cohort 460.72 0.30677 382.73 166.08 -0.2433086 0.039922
```

It's important that columns are included that match the unique identifying columns in our `bdotsObj`, and that the parameters match the coefficients used from `bdotsFit`

```
## Subject, Group, and LookType
head(refit)
#>    Subject Group LookType        fit      R2   AR1 fitCode
#> 1:       1    50   Cohort <gnls[18]> 0.96972  TRUE       0
#> 2:       1    65   Cohort <gnls[18]> 0.98049  TRUE       0
#> 3:       2    50   Cohort <gnls[18]> 0.98117  TRUE       0
#> 4:       2    65   Cohort <gnls[18]> 0.96975  TRUE       0
#> 5:       3    50   Cohort <gnls[18]> 0.97619  TRUE       0
#> 6:       3    65   Cohort <gnls[18]> 0.95349 FALSE       3

## doubleGauss pars
```

```
colnames(coef(refit))
#> [1] "mu"    "ht"    "sig1"  "sig2"  "base1" "base2"
```

We can save our parameter `data.table` for later use, or read in any other appropriately formatted `data.frame`

```
## Save this for later using data.table::fwrite
fwrite(parDT, file = "mypars.csv")
parDT <- fread("mypars.csv")
```

Once we have this, we can pass it as an argument to the `bdotsRefit` function. Doing so will ignore the remaining arguments

```
new_refit <- bdotsRefit(refit, paramDT = parDT)
```

We end up with a `bdotsObj` that matches what we had previously. As seeds have not yet been implemented, the resulting parameters may not be exact. It will, however, assist with not having to go through the entire refitting process again manually (although, there is always the option to save the entire object with `save(refit, file = "refit.RData)`)

```
head(new_refit)
#>    Subject Group         LookType        fit      R2  AR1 fitCode
#> 1:       1    50           Cohort <gnls[18]> 0.96972 TRUE       0
#> 2:       1    50 Unrelated_Cohort <gnls[18]> 0.97900 TRUE       0
#> 3:       1    65           Cohort <gnls[18]> 0.98049 TRUE       0
#> 4:       1    65 Unrelated_Cohort <gnls[18]> 0.87164 TRUE       1
#> 5:       2    50           Cohort <gnls[18]> 0.98117 TRUE       0
#> 6:       2    50 Unrelated_Cohort <gnls[18]> 0.95612 TRUE       0
```

Correlation Function

Correlations in `bdots`

This vignette is created to illustrate the use of the `bdotsCorr` function, which finds the correlation between a fixed value in our dataset and the collection of fitted curves at each time points for each of the groups fit in `bdotsFit`.

First, let's take an existing dataset and add a fixed value for each of the subjects

```
library(bdots)
library(data.table)

## Let's work with cohort_unrelated dataset, as it has multiple groups
dat <- as.data.table(cohort_unrelated)

## And add a fixed value for which we want to find a correlation
dat[, val := rnorm(1), by = Subject]

head(dat)
```

```
##    Subject Time DB_cond Fixations LookType Group      val
## 1:       1    0      50  0.011364   Cohort    50 -0.58118
## 2:       1    4      50  0.011364   Cohort    50 -0.58118
## 3:       1    8      50  0.011364   Cohort    50 -0.58118
## 4:       1   12      50  0.011364   Cohort    50 -0.58118
## 5:       1   16      50  0.022727   Cohort    50 -0.58118
## 6:       1   20      50  0.022727   Cohort    50 -0.58118
```

Now, we go about creating our fitted object as usual

```
## Create regular fit in bdots
fit <- bdotsFit(data = dat,
                subject = "Subject",
                time = "Time",
                group = c("LookType", "Group"),
                y = "Fixations", curveType = doubleGauss2(),
                cores = 2)
```

Using this fit object, we now introduce the `bdotsCorr` function, taking four arguments:

1. `bdObj`, any object returned from a `bdotsFit` call
2. `val`, a length one character vector of the value with which we want to correlate. `val` should be a column in our original dataset, and it should be numeric
3. `ciBands`, a boolean indicating whether or not we want to return 95% confidence intervals. Default is `FALSE`
4. `method`, paralleling the `method` argument in `cor` and `cor.test`. The default is `pearson`.

```
## Returns a data.table of class bdotsCorrObj
corr_ci <- bdotsCorr(fit, val = "val", ciBands = TRUE)
head(corr_ci)
```

```
##    time Correlation    lower   upper     Group Group1 Group2
```

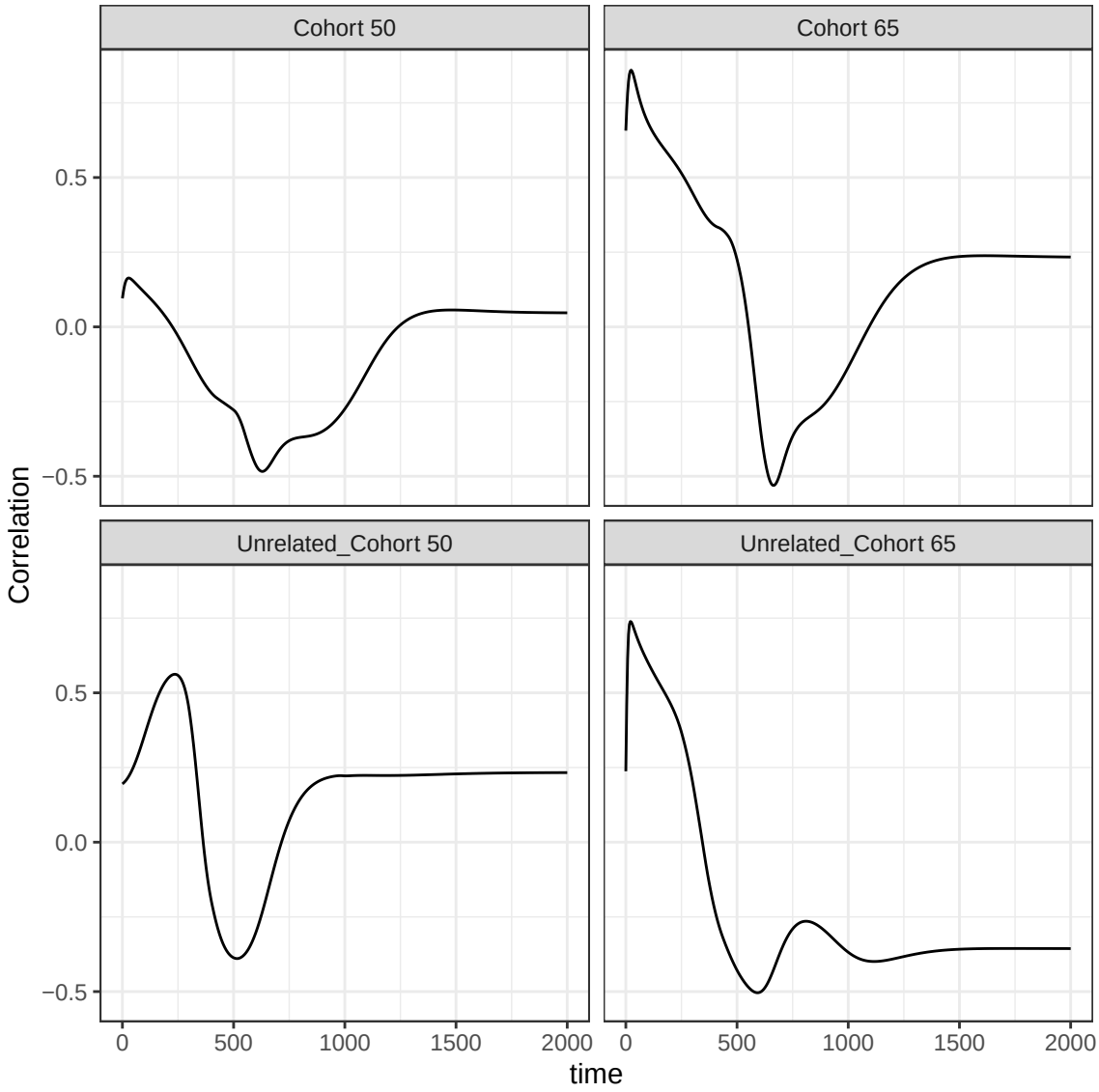
```
## 1:    0    0.095552 -0.60709 0.71434 Cohort 50 Cohort     50
## 2:    4    0.115780 -0.59402 0.72422 Cohort 50 Cohort     50
## 3:    8    0.132659 -0.58281 0.73227 Cohort 50 Cohort     50
## 4:   12    0.145523 -0.57408 0.73829 Cohort 50 Cohort     50
## 5:   16    0.154420 -0.56795 0.74241 Cohort 50 Cohort     50
## 6:   20    0.159896 -0.56413 0.74491 Cohort 50 Cohort     50
```

```
## Same, without confidence intervals
corr_noci <- bdotsCorr(fit, val = "val")
head(corr_noci)
```

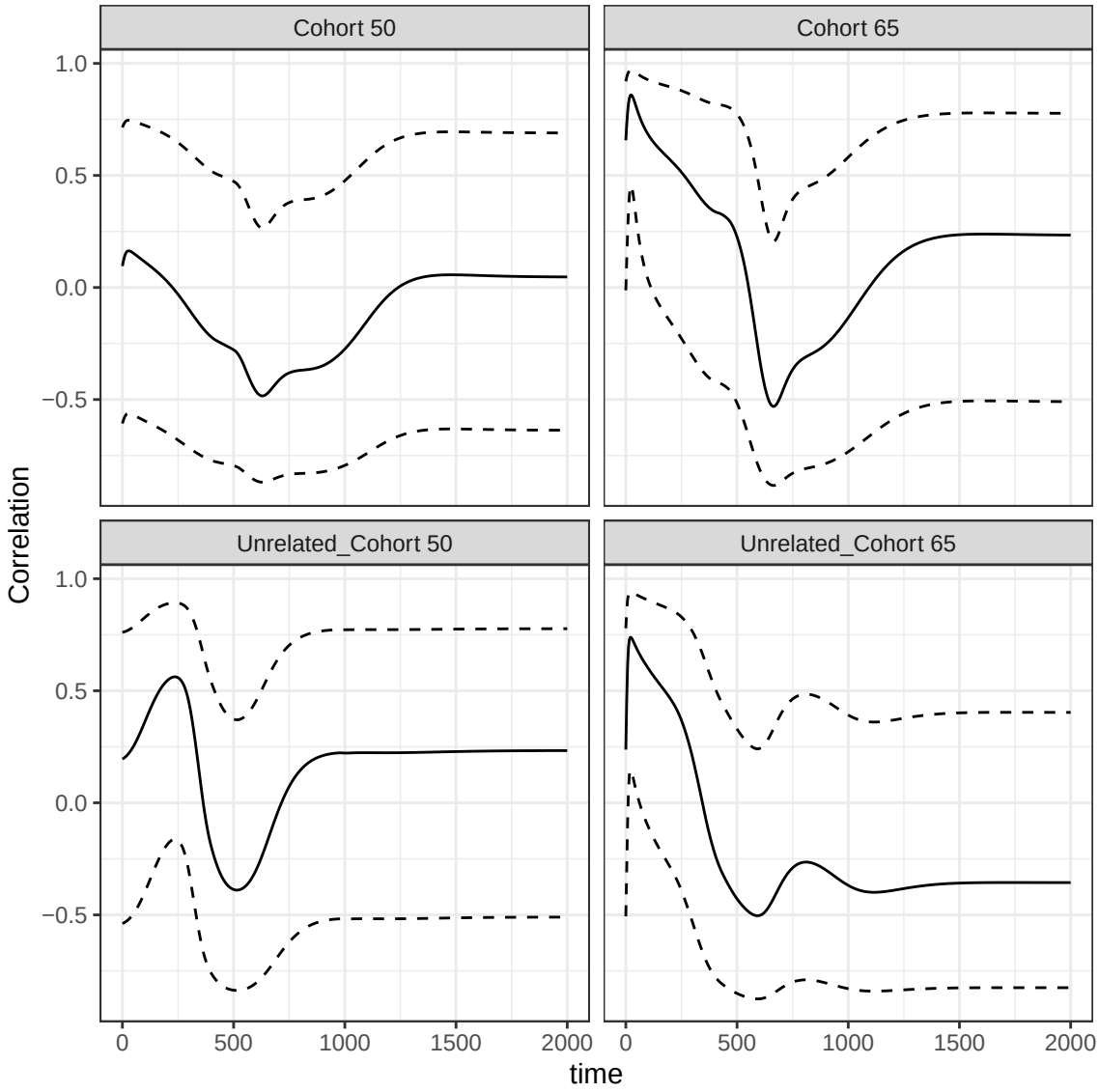
```
##    time Correlation     Group Group1 Group2
## 1:    0    0.095552 Cohort 50 Cohort     50
## 2:    4    0.115780 Cohort 50 Cohort     50
## 3:    8    0.132659 Cohort 50 Cohort     50
## 4:   12    0.145523 Cohort 50 Cohort     50
## 5:   16    0.154420 Cohort 50 Cohort     50
## 6:   20    0.159896 Cohort 50 Cohort     50
```

From here, we are able to use the `data.tables` themselves for whatever we may be interested in. We also have a plotting method associated with this object

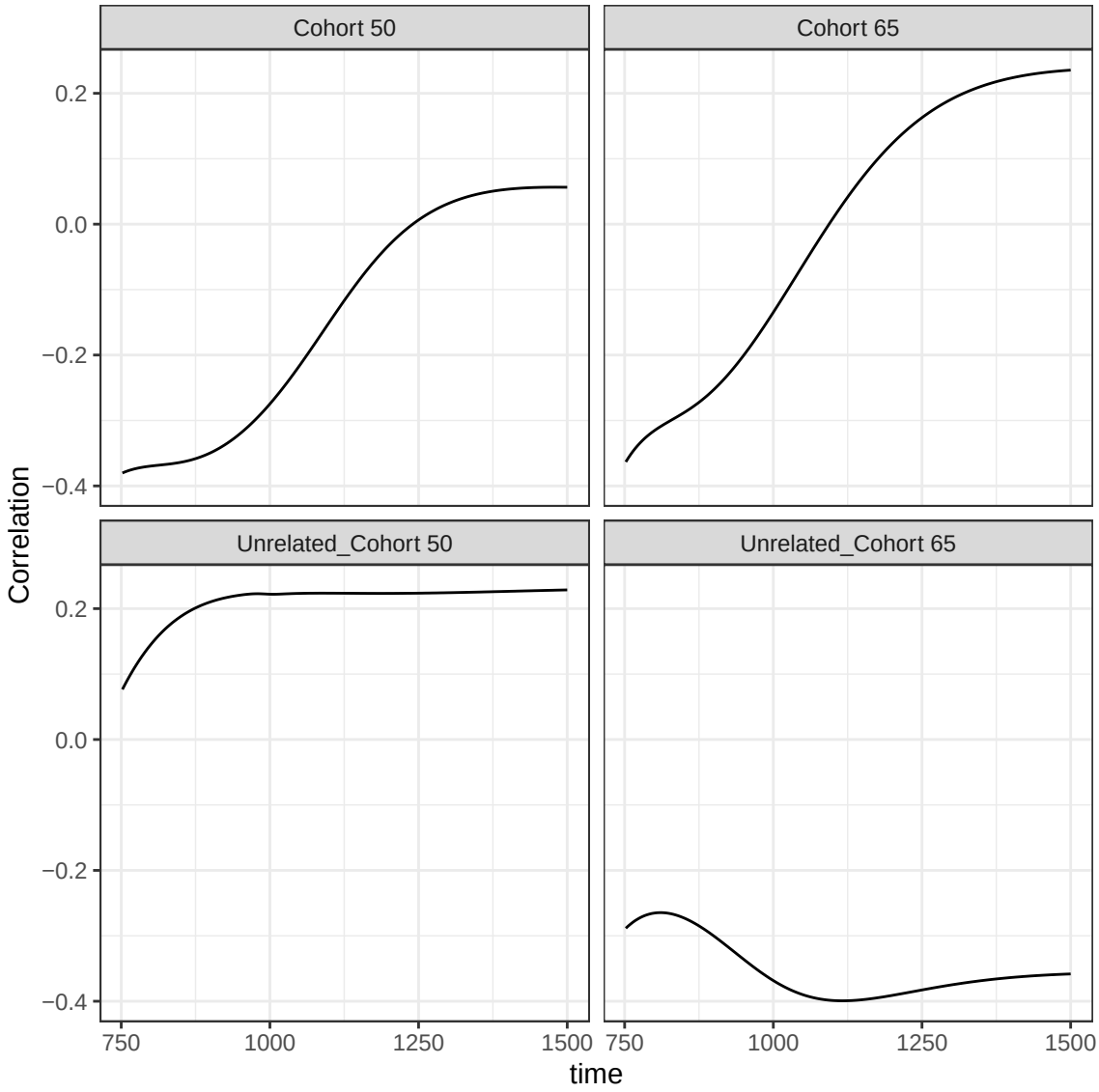
```
## Default is no bands
plot(corr_ci)
```




```
## Try again with bands
plot(corr_ci, ciBands = TRUE)
```

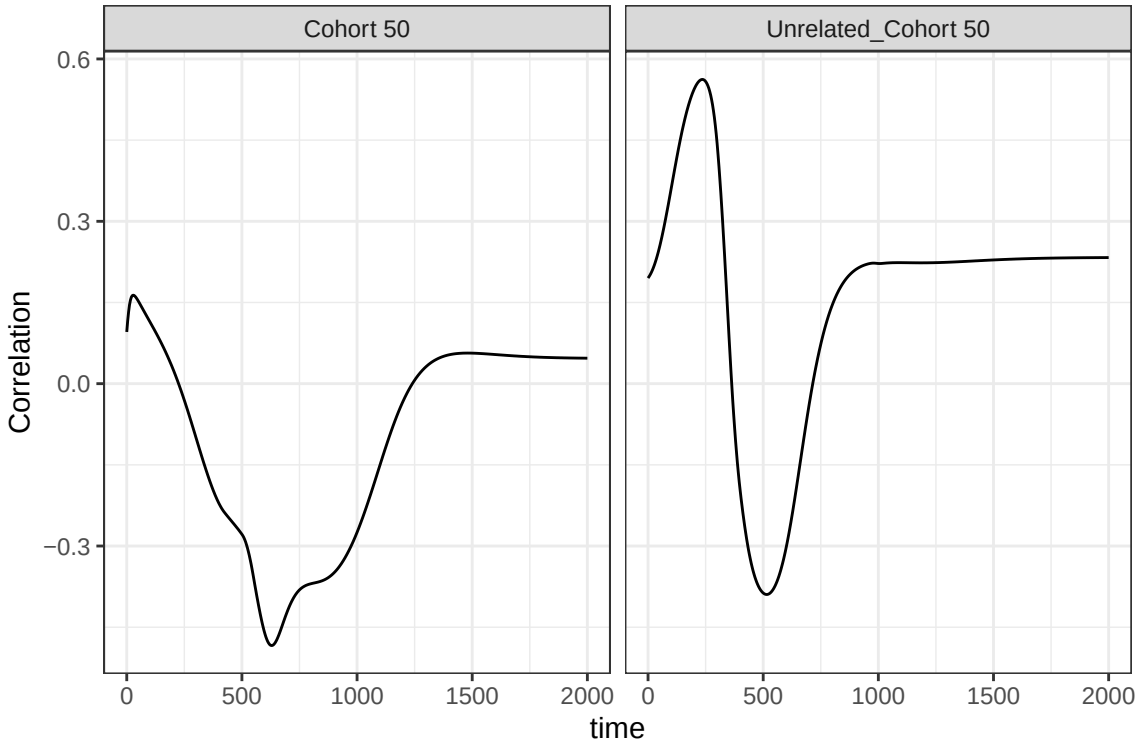



```
## Narrow in on a particular window
plot(corr_ci, window = c(750, 1500))
```



Because this object is a `data.table`, we have full use of subsetting capabilities for our plots

```
plot(corr_ci[Group2 == "50", ])
```


Cohort 50
Unrelated_Cohort 50
Correlation
0.6
0.3
0.0
−0.3
0
500
1000
1500
2000
time

User Curve Functions

We saw in the general overview when first generating our model fits with `bdotsFit` that we we could specify the curve with the argument `curveType`. Presently, the `bdots` package contains three options for this: `doubleGauss`, `logistic`, and `polynomial`. Documentation is included for each of these curves.

```
library(bdots)

fit <- bdotsFit(data = cohort_unrelated,
                subject = "Subject",
                time = "Time",
                y = "Fixations",
                group = c("DB_cond", "LookType"),
                curveType = doubleGauss(concave = TRUE),
                cores = 2)
```

Note that each of these is a function in their own right and must be passed in as a `call` object. Curve functions that include arguments further specifying the type of curve, i.e., `doubleGauss(concave = TRUE)` and `polynomial(degree = n)`, should include these when the call is passed into `bdotsFit` as seen in the example above.

Because each of the functions exists independently of `bdotsFit`, users can specify their own curve functions for the fitting and bootstrapping process. The purpose of this vignette is to demonstrate how to do so. If you find that you have a curve function that is especially useful, please create a request to have it added to the `bdots` package here.

We will examine the `doubleGauss` function in more detail to see how we might go about creating our own. First, let's identify the components of this function

```
doubleGauss
#> function (dat, y, time, params = NULL, concave = TRUE, ...)
#> {
#>     if (is.null(params)) {
#>         params <- dgaussPars(dat, y, time, concave)
#>     }
#>     else {
#>         if (length(params) != 6)
#>             stop("doubleGauss requires 6 parameters be specified for refitting")
#>         if (!all(names(params) %in% c("mu", "ht", "sig1", "sig2",
#>             "base1", "base2"))) {
#>             stop("doubleGauss parameters for refitting must be correctly labeled")
#>         }
#>     }
#>     if (is.null(params)) {
#>         return(NULL)
#>     }
#>     y <- str2lang(y)
#>     time <- str2lang(time)
#>     ff <- bquote(.(y) ~ (.(time) < mu) * (exp(-1 * (.(time) -
#>         mu)^2/(2 * sig1^2)) * (ht - base1) + base1) + (mu <=
#>         .(time)) * (exp(-1 * (.(time) - mu)^2/(2 * sig2^2)) *
```

```
#>         (ht - base2) + base2))
#>     attr(ff, "parnames") <- names(params)
#>     return(list(formula = ff, params = params))
#> }
#> <bytecode: 0x564a31bd8580>
#> <environment: namespace:bdots>
```

There are four things to note:

1. **Arguments** In addition to the argument `concave = TRUE`, which specifies the curve, we also have `dat`, `y`, `time`, `params = NULL`, and `....` These are the names that must be used for the function to be called correctly. The first represents a `data.frame` or `data.table` subset from the `data` argument to `bdotsFit`, while `y` and `time` correspond to their respective arguments in `bdotsFit` and should assume that the arguments are passed in as `character`. It's important to remember to set `params = NULL`, as this is only used during the refitting step.

2. **Body** As can be seen here, when `params = NULL`, the body of the function computes the necessary starting parameters to be used with the `gnls` fitting function. In this case, the function `dgaussPars` handles the initial parameter estimation and returns a named `numeric`. When `params` is not `NULL`, it's usually a good idea to verify that it is the correct length and has the correct parameter names.

3. **Formula** Care must be exercised when creating the `formula` object, as it must be quoted. One may use `bquote` and `str2lang` to substitute in the `character` values for `y` and `time`. Alternatively, if this is to only be used for a particular data set, one can simply use `quote` with the appropriate values used for `y` and `time`, as we will demonstrate below. Finally, the quoted `formula` should contain a single attribute `parnames` which has the names of the parameters used.

4. **Return Value** All of the curve functions should return a named list with two elements: a quoted `formula` and `params`, a named `numeric` with the parameters.

Briefly, we can see how this function is used by subsetting the data to a single subject and calling it directly.

```
## Return a unique subject/group permutation
dat <- cohort_unrelated[Subject == 1 & DB_cond == 50 & LookType == "Cohort", ]
dat
#>      Subject Time DB_cond Fixations LookType Group
#>   1:       1    0      50  0.011364   Cohort    50
#>   2:       1    4      50  0.011364   Cohort    50
#>   3:       1    8      50  0.011364   Cohort    50
#>   4:       1   12      50  0.011364   Cohort    50
#>   5:       1   16      50  0.022727   Cohort    50
#>  ---
#> 497:       1 1984      50  0.022727   Cohort    50
#> 498:       1 1988      50  0.022727   Cohort    50
#> 499:       1 1992      50  0.022727   Cohort    50
#> 500:       1 1996      50  0.022727   Cohort    50
#> 501:       1 2000      50  0.022727   Cohort    50
```

```
## See return value
doubleGauss(dat = dat, y = "Fixations", time = "Time", concave = TRUE)
#> $formula
#> Fixations ~ (Time < mu) * (exp(-1 * (Time - mu)^2/(2 * sig1^2)) *
#>     (ht - base1) + base1) + (mu <= Time) * (exp(-1 * (Time -
#>     mu)^2/(2 * sig2^2)) * (ht - base2) + base2)
#> attr(,"parnames")
#> [1] "mu"    "ht"    "sig1"  "sig2"  "base1" "base2"
#>
```

```
#> $params
#>         mu         ht       sig1       sig2      base1      base2
#> 428.000000   0.215909 152.000000 396.000000   0.011364   0.022727
```

We will now create an entirely new function that is not included in `bdots` to demonstrate that it works the same; the only change we will make is to substitute in the values for `y` and `time` without using `str2lang`. For our data set here, the corresponding values to `y` and `time` are `"Fixations"` and `"Time"`, respectively

```
doubleGauss2 <- function (dat, y, time, params = NULL, concave = TRUE, ...) {

  if (is.null(params)) {
    ## Instead of defining our own, just reuse the one in bdots
    params <- bdots:::dgaussPars(dat, y, time, concave)
  }
  else {
    if (length(params) != 6)
      stop("doubleGauss requires 6 parameters be specified for refitting")
    if (!all(names(params) %in% c("mu", "ht", "sig1", "sig2",
                                  "base1", "base2"))) {
      stop("doubleGauss parameters for refitting must be correctly labeled")
    }
  }

    ## Here, we use Fixations and Time directly
    ff <- bquote(Fixations ~ (Time < mu) * (exp(-1 * (Time - mu)^2 /
                  (2 * sig1^2)) * (ht - base1) + base1) + (mu <= Time) *
                  (exp(-1 * (Time - mu)^2/(2 * sig2^2)) * (ht - base2) + base2))
    return(list(formula = ff, params = params))
}

same_fit_different_day <- bdotsFit(data = cohort_unrelated,
                                   subject = "Subject",
                                   time = "Time",
                                   y = "Fixations",
                                   group = c("DB_cond", "LookType"),
                                   curveType = doubleGauss2(concave = TRUE),
                                   cores = 2)
```

Seeds have not yet been implemented, so there is some possibility that the resulting parameters are slightly different; however, using the `coef` function, we can roughly confirm their equivalence

```
## Original fit
head(coef(fit))
#>          mu      ht   sig1   sig2      base1    base2
#> [1,] 429.76 0.19860 159.89 314.64  0.0097098 0.033761
#> [2,] 634.93 0.26350 303.81 215.38 -0.0206361 0.028924
#> [3,] 647.07 0.25438 518.96 255.99 -0.2130875 0.013682
#> [4,] 723.05 0.25821 392.95 252.94 -0.0548262 0.031973
#> [5,] 501.48 0.22477 500.85 158.42 -0.3316790 0.025227
#> [6,] 460.72 0.30677 382.73 166.08 -0.2433086 0.039922

## "New" fit
head(coef(same_fit_different_day))
#>          mu      ht   sig1   sig2     base1    base2
#> [1,] 424.04 0.19850 154.21 319.46  0.011368 0.033637
```

```
#> [2,] 634.81 0.26351 303.66 215.43 -0.020586 0.028933
#> [3,] 646.94 0.25444 517.65 256.00 -0.211652 0.013571
#> [4,] 723.09 0.25821 393.00 252.90 -0.054853 0.031975
#> [5,] 501.61 0.22479 500.95 158.31 -0.332169 0.025229
#> [6,] 460.74 0.30680 382.52 165.99 -0.242859 0.039933
```