bdots

Last compiled: Tuesday 14th February, 2023 at 19:28

Abstract

The Bootstrapped Differences of Timeseries (bdots) was first introduced by Oleson (and others) as a method for controlling type I error in a composite of serially correlated tests of differences between two time series curves in the context of eye tracking data. This methodology was originally implemented in R by Seedorff 2018. Here, we revist the original package, both expanding the underlying theoretical components and creating a more robust implementation.

1 Introduction

i hate introductions we will do this part last

In 2017, Oleson et al. introduced a method for detecting time-specific differences between (group? idk rewrite this sentence) functions in time. This largely centered around bootstrapping and computing a series of highly correlated t-statistics and using estimates of the autocorrelation as an adjustment for the family-wise error rate, presented in the context of the Visual World Paradigm (VWP), an experimental paradigm combining eyetracking with an interactive environment to measure dynamics in language processing. In 2018, `bdots` was introduced on CRAN to perform a limited version of the analysis proposed in Oleson. Here, we introduce `bdots` v2, an update to the CRAN package that broadly expands the capabilities of the original.

This paper is not intended to serve as a complete guide for using the `bdots` package. Rather, the purpose is to showcase major changes and improvements to the package, with those seeking a more comprehensive treatment directed to the package vignettes. Updates to `bdots` have been such that there is little resemblance to the original. Rather than taking a "compare and contrast" approach, we will first enumerate the major changes, followed by a general demonstration of the package use:

1. Major overhaul to underlying methodolgy
2. Simplified user interface

3. 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-homogenous sampling of data across subjects and groups
7. Introduce formula syntax for bootstrapping difference function
8. Interactive refitting process

Bootstrapped differences in time series The original motivation for `bdots` came from the problem of identifying when, in time, differences in the distributions of curves from two groups were statistically different. A full review of the context in which it was introduced is available in Seedorff et. al., 2018 [**?**].

Do I talk about how original bdots was strictly limited to assumption of homogeneity across means? Because that is going to be a big fucking deal when comparing the old bdots to the new.

Where am I going to say hey, something like

$$y_{it} = f_{\theta_i}(t) + \epsilon_{it} \tag{1}$$

and then note that we have a bunch of different assumptions we can put on that. (The critical thing here is θ_i since each subject is different).

My life will also be A LOT easier if I just say up front that right now, period, bdots assumes parametric functions. It would be nice if there was non-parameteric, and someday there will be AND the underlying theory does not *necessarily* require it, but I won't have to tiptoe with language if I can just put my stake in the ground. Which I will. bdots is parametric, **period**.

In summary, `bdots` has transitioned from a package focused exclusively on densly sampled time series assuming a limited number of functional forms to a robust framework for identifying time windows of significant difference across blah blah blah

2 Methodology and Overview

A standard analysis using `bdots` consists of two steps: fitting the observed data to a specified parametric function, f_θ, and then using the observed variability to construct estimates of the distributions of each groups' curves. Here, we briefly detail how this is implemented in practice and introduce the new methodologies in `bdots v2`. A more comprehensive treatment of these new methods, along with their justifications, is offered in Chapter 3.

Remember, I just need to physically be writing during this time. It can be trash. Trash can be edited, blank pages and regret cannot.

2.1 Establishing subject-level curves

We begin with the assumption that for subject i, the observed data is of the form

$$y_{it} = f_{\theta_i}(t) + \epsilon_{it}, \tag{2}$$

where ϵ_{it} can be specified to be either independent or have an AR(1) auto-correlation structure. Each subject is fit in `bdots` via `gnls`, returning an estimated set of parameters and their associated standard errors. Assuming large sample normality, we are able to construct a sampling distribution for each subject, accounting for within-subject variability. This gives us for each subject a distribution

$$\hat{\theta}_i \sim N(\theta_i, s_i^2). \tag{3}$$

2.2 Estimating Group Distributions

Once sampling distributions are created for each subject, we are prepared to begin estimating group distributions. We assume that the mean parameters for each subject come from a group level distribution, where for each subject i,

$$\theta_i \sim N(\mu_\theta, V_\theta). \tag{4}$$

This is notably in contrast to the original implementation of `bdots`; there, the authors proceeded with an assumption of homogeneity of means, with $\theta_i = \theta_j$ for each i, j. A more thorough investigation of this, along with justification for the current method, is presented in Chapter 3.

This in hand, we go about estimating the group distributions according to the following procedure: (not in love with the notation here).

1. For a group of size n, select n subjects from the group *with replacement*. This allows us to construct an estimate of V_θ.
2. For each selected subject i, draw a set of parameters from the distribution $\theta_i^* \sim N(\hat{\theta}_i, s_i^2)$. This gives us an accounting of the observed within-subject variability
3. For each resampled θ_i^*, find the bth bootstrap estimate for the group,.

$$\theta_b^* = \frac{1}{n}\sum_{i=1}^{n} \theta_i^*, \quad \text{where} \quad \theta_b^* \sim N\left(\mu_\theta, \frac{1}{n}V_\theta + \frac{1}{n^2}\sum s_i^2\right). \tag{5}$$

4. Perform steps (1.)-(3.) B times, using each θ_b^* to construct a distribution of population curves, $f_\theta(t)$.

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.

2.3 Hypothesis testing for statistically significant differences

We now turn our attention to the primary goal of an analysis in `bdots`, identifying time windows in which the distribution of curves of two groups differ significantly. A problem unique to the ones address 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). Version 2 of `bdots` provides two ways with which this can be accomplished.

2.3.1 Oleson Adjustment

Just as in the original iteration of `bdots`, we are able to construct test statistics from the bootstrapped estimates described in the previous section. These test statistics $T(t)$ can be written as

$$T(t) = \frac{(\overline{f}_1(t) - \overline{f}_2(t))}{\sqrt{\frac{1}{n_1}\text{Var}(f_1(t)) + \frac{1}{n_2}\text{Var}(f_2(t))}}, \tag{6}$$

where [...] I actually don't like that notation at all, but I can't use $\overline{p}_{1t}$ and s^2_{1t}, really, because I already used s_i^2 for within-subject variance. Whatever, pin in this for now, pretend I carefully described what all of the components of this test statistic is for now.

To account for the multiple testing problem with autocorrelated test statics, we construct a nominal significance level α^* to produce the desired effective FWER α. For now I won't give any more details, it basically finds some autocorrelation parameter and uses this to create a new alpha. We then use this to determine which $T(t) < z1 - \alpha^*/2$, giving us our significant regions. neato.

2.3.2 Permutation testing

Alternatively, `bdots` provides a modified permutation test for controlling the FWER without any additional assumptions on the autocorrelated status of the errors or test statistics.

We begin by creating test statistics at each time point, similar to Equation 6. Using the fitted parameters $\hat{\theta}_i$ for each subject i, we construct subject-specific curves $f_{\hat{\theta}_i}$ and use *these* to construct population mean

and standard deviations at each time point, giving population curves and standard deviations [...] (i still don't have notation i really like for this, especially in contrast to the bootstrapping step). We use these to create the observed test statistic

$$T_p(t) = [\dots] \tag{7}$$

(this is basically the same formula as Equation 6 but with absolute value and having $\overline{f}$ mean different things. whatever notation is used)

When then going about using permutations to construct a null distribution against which to compare the observed statics. We do so with the following algorithm:

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 $T_p(t)$, recording the maximum value
4. Repeate (2.)-(3.) P times

The collection of maximum values for $T(t)$ will serve as the null distribution against which to compare our observed $T(t)$. Regions in which the observed t statistic are beyond the specified α in the null distribution are then considered significant.

2.3.3 Odds and Ends

Both of the methods presented are able to accommodate paired assumptions with minor adjustments to their algorithms. In the case of the bootstrap, we simply must take care to ensure that for each iteration, the collection of subjects sampled in one group with replacement are also sampled in the other, ensuring that each bootstrapped estimate comes from the same distribution. Likewise, paired testing is implemented through permutation testing by modifying the shuffling process so that each subject has one set of observations in each of the permuted groups.

Finally, it is of note that other adjustments for FWER are offered here as were in the original implementation of `bdots`, including all of the adjustments present in the `p.adjust` function from the `stats` R package. I say this for completeness, but idk that anybody cares. Okie dokie, then, thems the methods! On to an example analysis.

```
> head(mouse)

     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
```

Figure 1: Illustration of Mouse data

3 SAmple Analysis

We are going to use this to demonstrate an example use-case of the bdots package with all kinds of sexy bells and whistles

3.1 Example Data

The `bdots` pacakge now has support for dat with non-homogenous time sampling across subjects and trials. We illustrate this with a worked example where we will consider a study (source?) comparing tumor growth for the 451LuBr cell line in mice with repeated measures in five treatment groups.

A standard analysis using `bdots` consists of two steps: fitting subject specific curves and determining the group distributions through bootstrapping.

3.2 Fitting Curves

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

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

(need to add/talk about an AR1 argument that fits data with this assumption or not)

The dataset being analyzed is passed into the `data` argument and should be in long format. The experimental conditions or groups being analyzed are passed into the `group` argument: unlike previous iterations of `bdots`, an arbitrary number of groups can be fit at once, assuming that they all adopt the same parametric form (for example, the four parameter logistic). The `subject` argument is the subject identifier column. Here, it is important to note that if a paired design is being conducted that the subject identifiers are the same between groups. That is, when determining the regions of significant differences in the bootstrapping step, the paired status between two groups is determined by comparing the intersection of two groups with

themselves. `time`, of course, is the name of the column containing time, and `cores` tells the fitter how many computer cores to use in a parallel enviroment (the default is half). Each of `subject`, `time`, and `y`, are length one character vectors representing columns of the dataset used in `data`, while `group` is a character vector (of varying length), also column(s) found in `data`. Finally, there is the `curveType` argument which is discussed in the next section.

Curve functions New here is `curveType`, where we specify the (parametric) curve that we wish to fit to the dataset provided in `bfit`. Unlike the previous arguments taking either a `data.frame` or character vector, this argument takes a function call, for example, `logistic()`. The motivation for this is detailed in the appendix, but in short, it allows one to provide additional arguments to the fitting curve. For example, provided with `bdots` are a number of common parameteric functions (with their default arguments) including `logistic()`, `doubleGauss(concave = TRUE)`, `expCurve()`, and `polynomial(degree = 5)`. `bfit` will also accept user-created curves, and a detailed vignettes for writing yoru own can be found with `vignette("bdots")`.

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

Fit Codes (assumption of autocorrelation may change)

One column included in `bdotsObj` is `fitCode`, a numeric summary statistic ranked from 0 to 6 detailing information about the quality of the fitted curve. By default, `bfit` assumes autocorrelation in the fitting of individual subject curves. The `gnls` function is not always able to accomodate this, and in situations in which the fitting algorithm fails, it will try again, dropping the AR(1) assumption. An indication of whether an AR(1) model was fit or not, in conjunction with a range of R^2 values, gives rise to the value for `fitCode`, presented in Table 1. The method for determining the `fitCode` can be seen in the following lines:

```
AR1 <- # determines if AR1 status
fitCode <- 3L*(!AR1) + 1L*(R2 < 0.95)*(R2 > 0.8) + 2L*(R2 < 0.8)
```

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

Future work on the package will seek to modularize the evaluation of fit quality, allowing users more freedom in creating a hierarchy of fitted values.

`fitCode`	AR(1)	R^2
0	TRUE	$R^2 > 0.95$
1	TRUE	$0.8 < R2 < 0.95$
2	TRUE	$R^2 < 0.8$
3	FALSE	$R^2 > 0.95$
4	FALSE	$0.8 < R2 < 0.95$
5	FALSE	$R^2 < 0.8$
6	NA	NA

Table 1: fit codes, though less relevant for other types of data so idk really what to do about it. nothing for the dissertation, at least

3.2.1 Worked example (fitting)

We begin by fitting our mouse data using the `bdots` fitting function, `bfit`

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

As the object `fit` is of class `data.table`, the default print option simply prints it as it would any other `data.table` or `data.frame`. A summary method is included, providing information on the type of fit, diagnostics for each group, and diagnostics for fits overall. An example of this summary for a single treatment group is included here:

```
> summary(mouse_fit[Treatment == "A", ])

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: Abridged summary of mouse fit data

The default plotting method plots each of the fitted subjects, including observed and fit data. See Figure 3 for a plot of the first four fitted subjects.

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

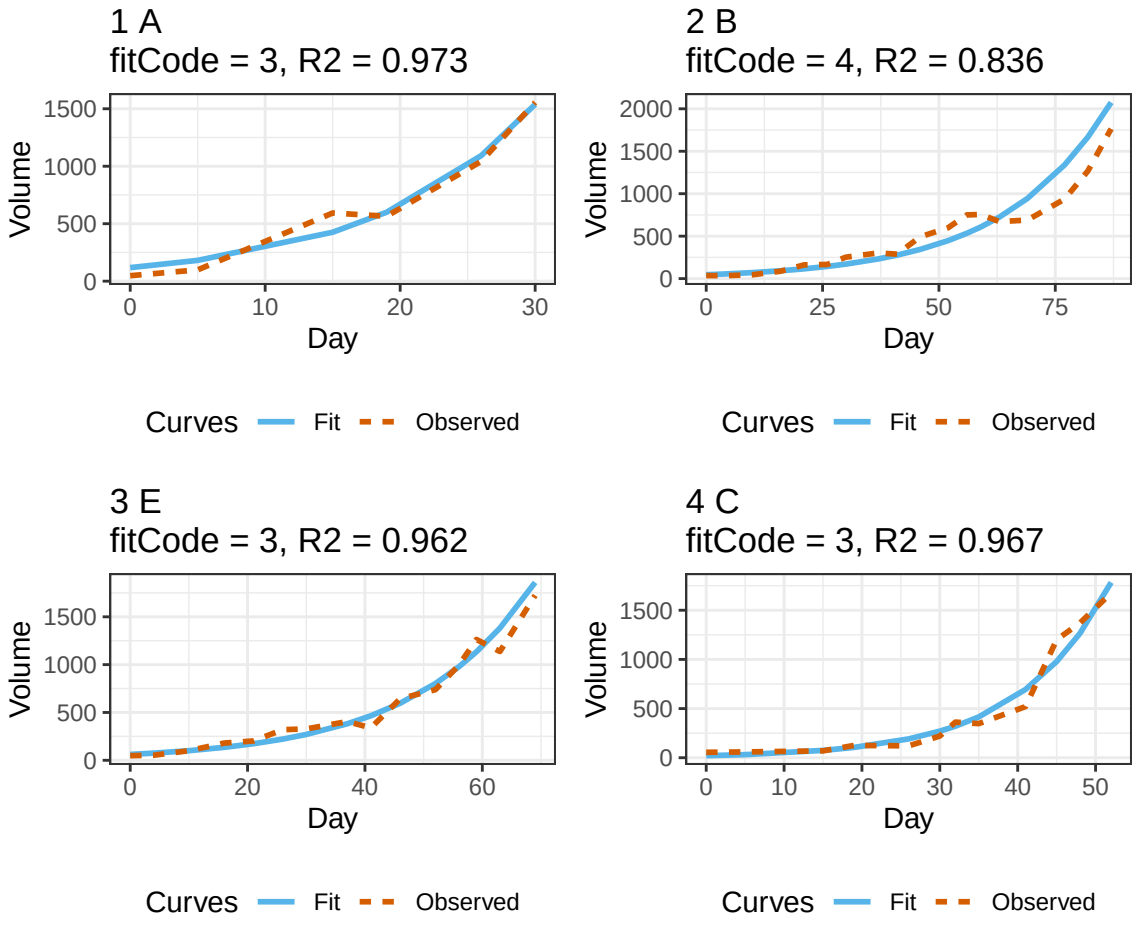

Figure 3: No longer using scale image saweet. But i would like to include a generic/no ggplot2 in the final version

3.3 Bootstrapping

Once fits have been made, we are ready to begin estimating the group distributions. This is done with the bootstrapping funciton, `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

```
(singleMeans we should discuss)
bboot(formula, bdObj, Niter, alpha, padj, permutation, singleMeans, cores, ...)
```

By default, the method for determining significance when comparing the difference of two time series is the permutation method, as this makes the fewest assumptions while maintaining an appropriate type I error rate and adequate power (see chapter 3). While `permutation = TRUE`, the argument to `padj` is ignored;

otherwise, adjustments can be made to the specified `alpha` to control FWER as was presented in the original `bdots` package. Regardless of what method is used for determining significance, bootstrapping as detailed in Chapter 3 is still used for determing group distributions. Finally, in contrast to the previous `bdots`, there is no longer a need to specify if the groups are paired; `bboot` determines this automatically based on the subject identifiers in each of the groups.

A key component of the bootstrapping function is specifying which groups in our dataset we are wishing to analyze and how. This is done with a formula syntax unique to `bdots` explained in the next section.

3.3.1 Bootstrapping Formula

As the `bfit` function is now able to create fits for an arbitrary number of groups, 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 values of two of the groups in that same 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 used in a character vector. To give a more concrete example, suppose we wished to compare the difference in tumor growth curves for treatments A and B in our mouse data (see Figure 1 for column names and values). 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 nested groups, the second when a difference of difference analysis is conducted. Details on both of these cases are handled in the appendix.

3.4 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, 106) [31 points]

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

FWER adjust method: Permutation
Alpha: 0.05
Significant Intervals:
     [,1] [,2]
[1,]   15   46
```

There are a few components of this worth identifying when reporting the results. First, included at the top is the name of the function used, its expression in R, and the range of time points considered. Below this is information related to the provided formula, namely: is this a difference of difference, are the elements of the groups paired, and what grouping was used in determining the differences. The final section includes information on the FWER, including the method used and the level at which significance was deteremined. When using one of the autocorrelated alpha-adjusted parameters, estimated of the autocorrelation and the adjusted alpha are also presented. Finally included is a matrix of significant intervals. This is NULL if no significant differences were found at the specified alpha; otherwise one row is included for each disjointed region of significance.

In addition to the provided summary output, plotting methods are available

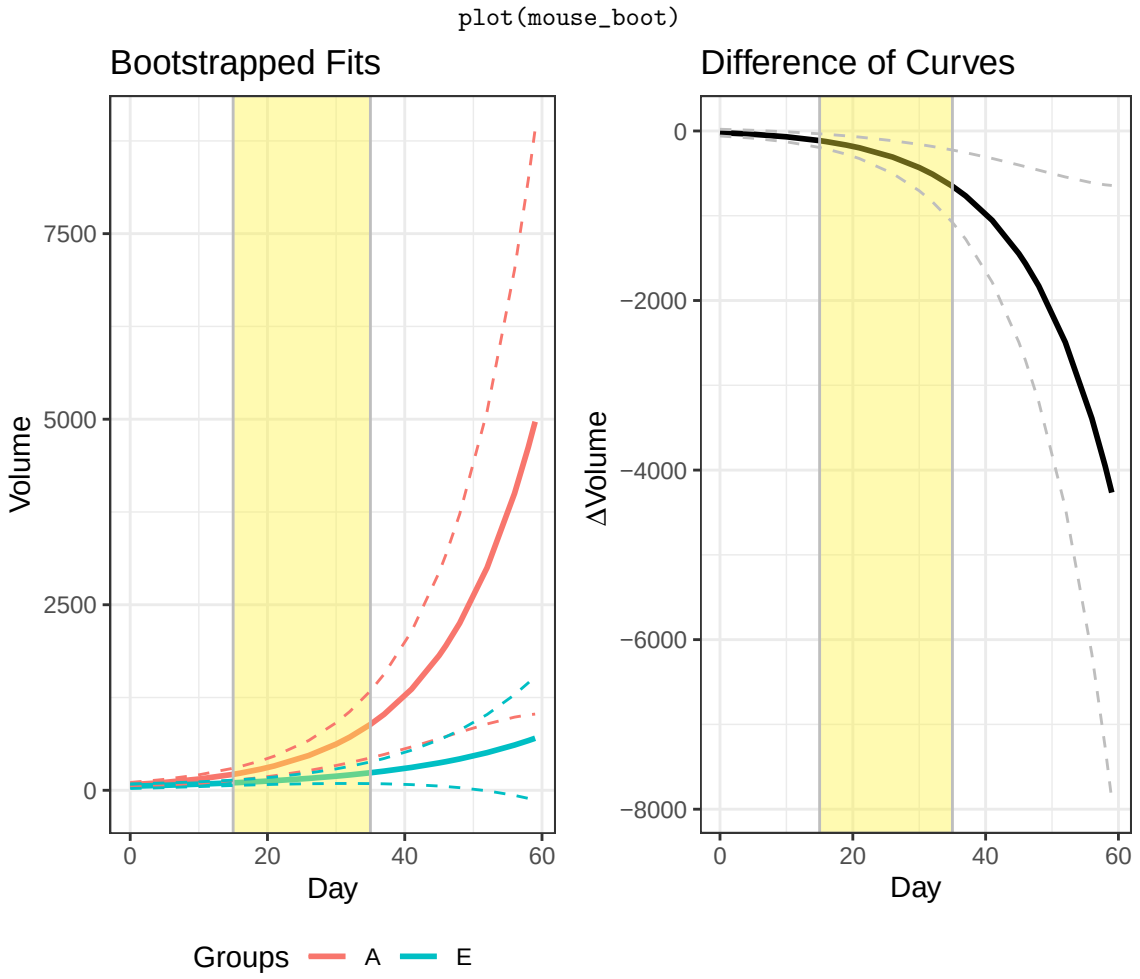

Figure 4: These confidence intervals are based on the bootstrap (permutation only does significance testing). I will go through and investigate this more closely later, my guess is this is related to the fact that we are using the intersection of time points to fit curves while most data (maybe) only goes out to half way. will investigate later. Because obviously values of 100,000 are way the eff out of what is even remotely reasonable. UPDATE: I cheated and cut time to max out at 60 instead of 105. Can consider how else to resolve this. Also, NOTE: significant region did change somewhat considerably when cutting max time with less time being evaluting resulting in larger regions being significant

Depending on user needs, these plots can be recreated both without confidence bands or without the additional difference curve

```
plot(mouse_boot, ciBands = FALSE, plotDiffs = FALSE)
```

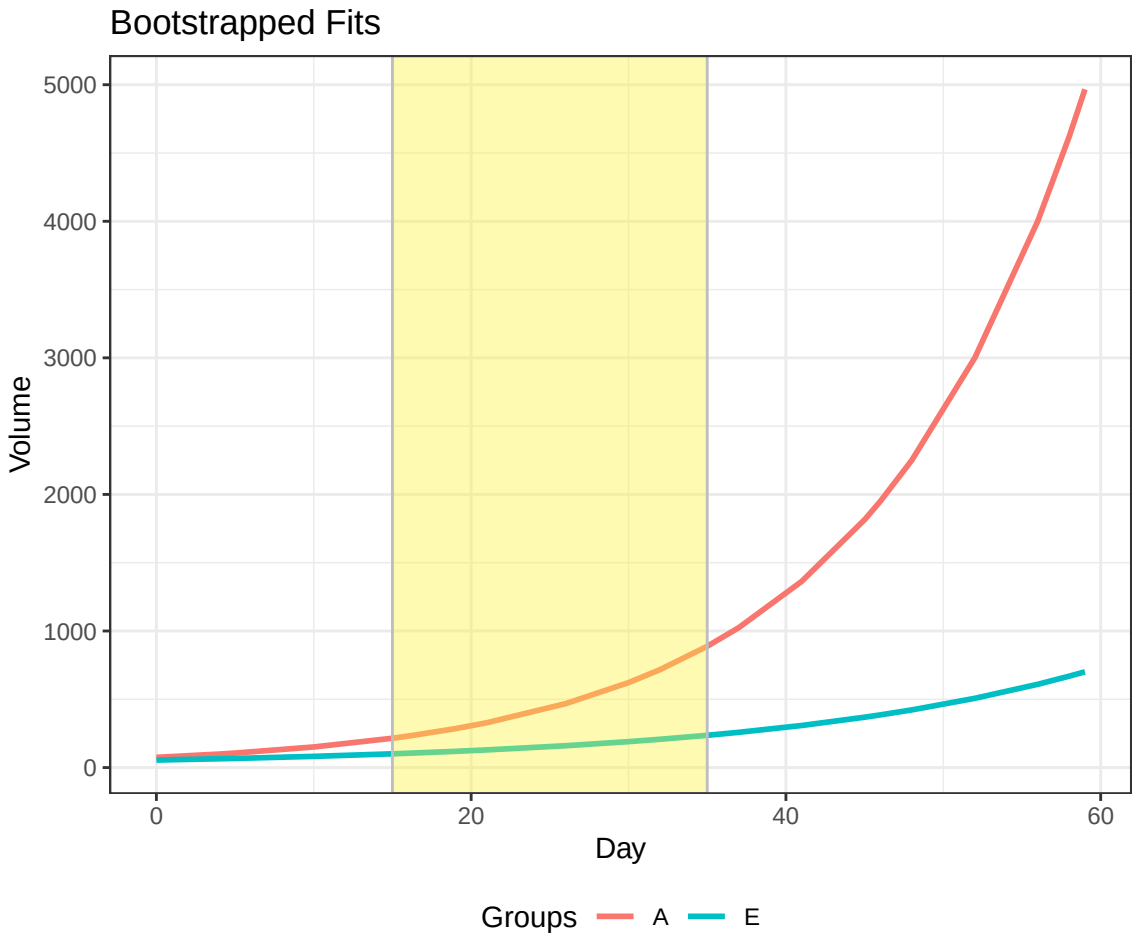

Figure 5: obviously this has same issue with previous.

4 Extras

Let's do a brief tour of some of the other additions to bdots that probably doesn't warrant its own section for use

Do I mention non-homogenous sampling elsewhere? could also illustarte with saccades, idk

4.1 Non-homogenous sampling

[not sure if i need this section or if i should elaborate]

The `bdots` package now has support for data with non-homogenous time sampling across subjects or trials. For example, here is data collected comparing tumor growth for 451LuBr cell line in mice with repeated measures and five treatment groups

It is not a problem to fit these groups and perform our bootstrapping analysis either on the union of

```
with(tumrdata[subjects 1-4, ], plot(observations as points))
```

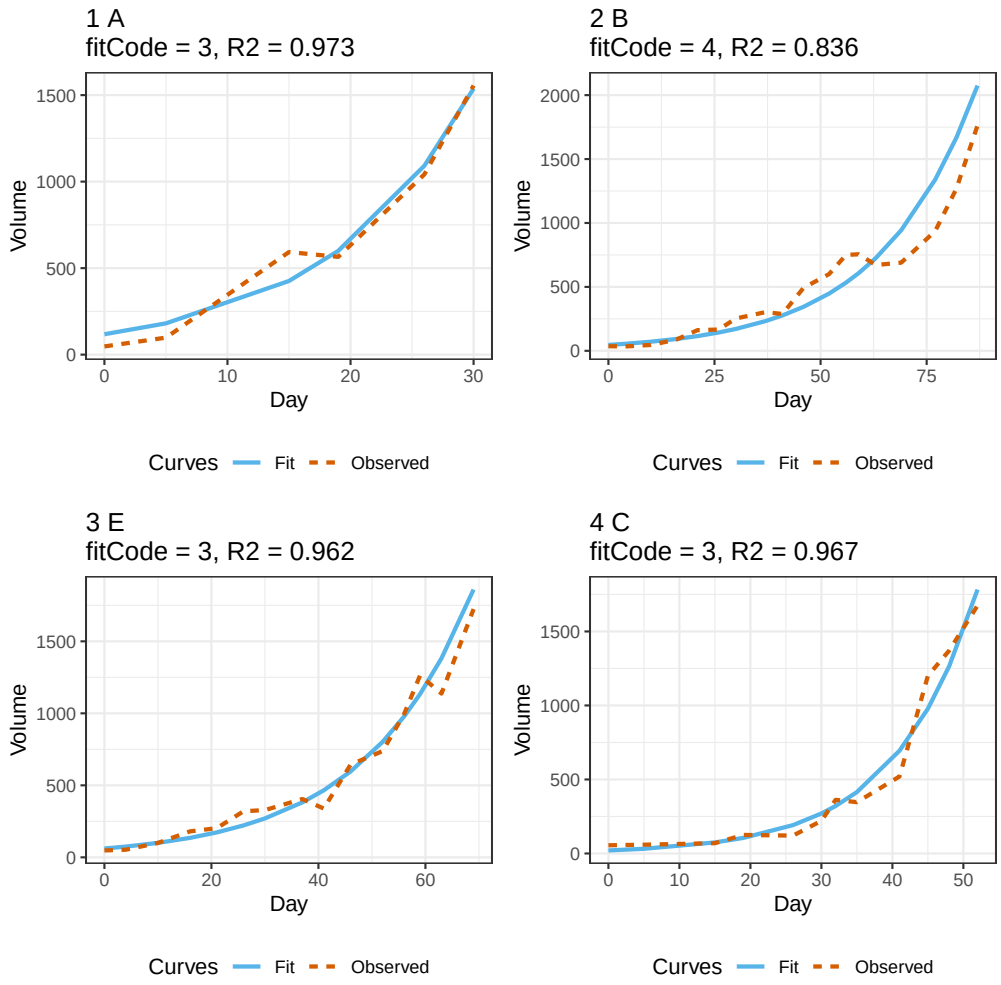

Figure 6: mousey data

observed time, or some custom range in between

`example?`

`bdots` also allows for repeated observations, as is the case with saccade data from the VWP. Here, an individual subject has 30 trials with saccades taken at the trial level. That is, rather than taking a sequence of observations for each subject, `bfit` allows for an unordered set with observations and associated time, $\mathcal{S}_i = \{(y_j, t_j)\}$ across j observations. As this relates to the VWP, you can read more about his development in my dope ass other paper called chapter 2.

4.2 Refitting

There are sometimes situations in which the fitted function returned by `bfit` is a poor fit. This is largely a consequence of the sensitivity of the `nlme::gnls` function used in `bfit` for fitting the non-linear curves. Sensible starting parameters are computed as a part of the curve fitting functions (i.e., `logistic()`, but see the vignettes for more details), though these can often be improved upon. The quality of the fit can be evidenced by the `fitCode` or via a visual inspection of the fitted functions against the observations for each subject. When this occurs, there are several options available to the user, all of which are provided through the function `brefit` (previously `bdotsRefit`). `brefit` takes the following arguments:

```
brefit(bdObj, fitCode = 1L, quickRefit = FALSE, numRefits = 2L, paramDT = NULL)
```

The first of these arguments, outside of the object itself, is `fitCode`, indicating the minimum fit code to be included in the refitting process. This is a convenient way to limit the refitting process to those of a particular quality. Given the sensitivity of `nlme::gnls` to the starting parameters, one can often try automatically refitting all of the specified subject by simply jittering the previous set of parameters and comparing the updated fit to the original; this is done by setting `quickRefit = TRUE`, with `numRefits` indicating how many attempts the fitter should make in doing so.

Finally, `paramDT` 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 whereby for each subject to be refit, in addition to being given a series of diagnostics:

```
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):
```

Along with this is given a plot of the original fit, side-by-side with the suggested alternative.

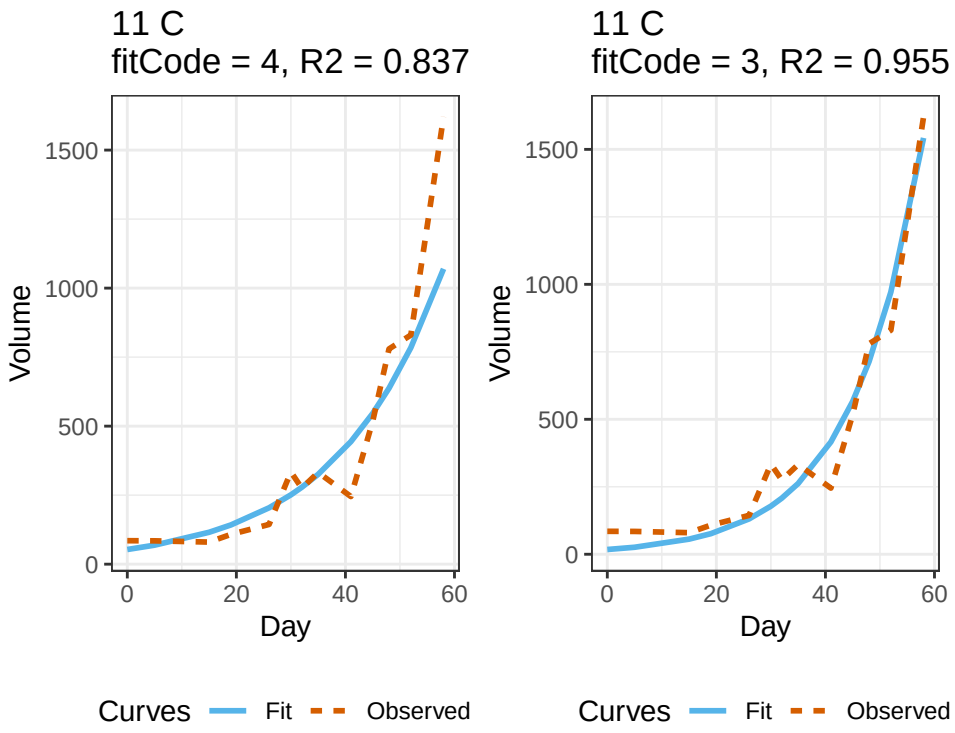

Figure 7: before and after refit: man i am good at picking new parameters

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 the significance testing step), it is critical that if a subject has a poor fit in one group and must be removed that he or she is 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.

4.3 User created curves

Continue to ponder if this worth elaborating on

4.4 Correlations

There are sometimes cases in which we are interested in determining the correlation of a fixed attribute with group outcome responses across time (what such a case may be, I have no idea). 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")
```

This returns a thing that can be plotted. Idk, it really doesn't seem that important

4.5 α Adjustment

Finally, we consider an extension to the `p.adjust` function, `p_adjust`, identical to `p.adjust` except that it accepts method `"oleson"` and takes additional arguments `rho`, `df`, and `cores`. `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:

```
t <- diffinv(rnorm(100))
rho <- ar1Solver(t)
unadj_p <- pt(t, df = 10)
adj_p <- p_adjust(unadj_p, method = "oleson", df = 10, rho = rho)
```

5 Discussion

First paragraph of conclusion. Maybe say things like here are the problems bdots has tried to solve, etc., idk it just needs to be reconciled with the last paragraph, which i kinda like.

While significant improvements have been made, there is of course room for further expansion, and it is this area that we are most excited about future directions. 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 flexibility in fit metrics (this last sentence kind of sucks).

The original implementation of `bdots` set out to address a very narrow set of problems which it succeeded in doing. 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 a simple, reproducible workflow that is able to quickly and simply address a far broader range of problems. This includes not only introducing support for a far wider variety of types of data but also expanding the methods by which data can be analyzed through the introduction of user-specified parametric curves. Further, the implementation of the underlying methodology has been improved and expanded upon, offering far better coverage of the estimated distributions, as well as increasing the methods by which significance testing is conducted, accommodating a broader range of underlying assumptions. Finally, a full suite of ancillary functions have been added, ranging from simple quality of life additions (methods, refitting) to those that add (can't say expand again) (?) analytical questions (?) (correlation function, etc.,). Concluding sentence. The end.

Appendix A - custom curves

From an R programming perspective, this is perhaps the most novel and interesting portion of the new package update. Worked use-case examples are included in the pacakge vignettes, so here we will limit discussion to the theoretical considerations when implementing it since it's actually pretty neat (I think). plus it adds length to my dissertation and everybody knows longer == more intelligent

Appendix B - Fitting non-nested groups

(currently just copy pasted from the body of document, not editted so no need to really review)

First, there would be some function of sorts, something like `makeUniqueGroups` which would create a new group column with each permutation of previous groups being given a unique identifier. Doing this on the vehicle example would look something like `fit <- makeuniquewhatever` resulting in the following grouping structure (for example) (and maybe you could specify group name and values who knows, kinda like factor this is just a working thought example)

Origin	Class	Color	bgroup
foreign	car	red	A
		blue	B
	truck	red	C
		blue	D
domestic	car	red	E
		blue	F
	truck	red	G
		blue	H

To then investigate differences in outcome between a foreign red car and a domestic blue truck would simply then be

$$\texttt{y} \sim \texttt{bgroup(A, H)}$$

yeah not like sexy or anything but whatever it would work.

Appendix X

Copy and paste hard data example here

We will illustrate use of the updated `bdots` package with a worked example, using an artificial dataset to help detail some of the newer aspects of the package. The dataset will consist of outcomes for a collection of vehicles, consisting of eight distinct groups. These groups will be nested in order of vehicle origin (foreign or domesetic), vehicle class (car or truck), and vehicle color (red or blue). Further, vehicles of different color but within the same origin and class groups will be considered paired observations. A table detailing the relationship of the groups is shown here:

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

Table 2: table of stuff

The outcome here is simply `y` due to a lack of creativity, but the functional form assumed (and used in data generation) follows the four parameter logistic,

$$f_\theta(t) = b + \frac{p-b}{1+\exp\left(\frac{4s}{p-b}(x-t)\right)}, \tag{8}$$

where b, p, s, and x represent the baseline, peak, slope, and crossover points, respectively

The formula argument serves two functions in `bboot`: first, it specifies the collection of curves we wish to investigate the difference between, and second, it determines if we are interested in directly comparing the differences or the difference of differences between curves.

To begin, let's reintroduce the structure of the groups we have in our dataset. Recall that we have foreign and domestic cars and trucks, and each of these vehicles comes in red and blue. Recall also that the different colors of each vehicle are considered paired observations.

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

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

Note that this involves the grouping variable, `Origin`, with the two values we are interested in comparing, `domestic` and `foreign`. With this specification, the distribution of functions considered in `domestic` include all red and blue domestic cars and trucks.

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).
```

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 of two groups. Although there is no native way to handle interactions in `bdots`, this can be done indirectly through the difference of differences (McMurray et al 2019, though truthfully I still don't understand why). 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 again 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` synatx 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 outcome of interest being color. This is helpful in remembering which term goes on the LHS 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).
```

The formula syntax was originally contrived to make comparisons within groups or within nested groups. Conceivably, however, one could be interested in making the comparison between domestic red trucks and foreign blue cars. Doing so requires a bit of a work around. Examples detailing how one might go about doing this are included in appendix B.