bdots

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 note

perhaps obvious but i have made no adjustments to account for spacing or plot location on page

2 Introduction

In 2017, Oleson et al. introduced a method for detecting time-specific differences in densley sampled time series. 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. This paper was 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, R software was introdued on CRAN to perform a limited version of the analysis proposed in Oleson. In this paper, 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 use guide to updates in 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 the bdots package 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. Simplified user interface

2. User defined curves
3. Permit fitting for arbitrary number of groups
4. Updates to bootstrapping algorithm and introduction of permutation test
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. The fitted bdots object inherits from data.table class
9. bdots is now stylized "bdots"

Bootstrapped differences in time series The high level motivation, abstracted from the particulars, is more or less as follows: we are often interested in generally comparing time series between two or more groups and in particular, identifying a window of time in which they differ significantly without a priori specification of any regions. A full(er) review of previous methods can be found in Seedorff, though here we limit the scope of interest to specifying that we are interested in "developing a statistical tool to (1) detect differences in two time series (such as the VWP) and (2) to offer a precise characterization of the time window in which a difference occurs [**?**]." This is subsequently done in two steps. First, we use a bootstrapping procedure to estimate the group distributions of two time series; and second, we use either a FWER correction or permutation testing to identify time windows in which differences occur.

A typical instantiation of this problem occurs when we have two groups (or experimental conditions, etc.,) in which subjects have an associated time series. For example, we may be interested in comparing the growth of a particular tumor in mice over time between a control group and several candidate treatment groups. It's assumed that each group has some distribution of associated functions in time, and we are interested in identifying windows in time in which these distributions of functions are significantly different.

need transition here to next paragraph better

Although there is no native way to handle interactions between groups in `bdots`, this can be done indirectly through the differences 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 a vehicle differentially impacts performance based on the vehicle type such as cars and trucks. We might then look at the difference between red cars and red trucks and then the difference between blue cars and blue trucks. If color does not mediate this difference, the difference between red cars and trucks should be the same as the difference between blue cars and trucks. If color does differentially impact performance between cars and trucks, this will be evident when considering the difference between the differences.

The original bdots package was predicated on comparing differences between dense, highly correlated time series by first specifying functional forms and then performing statistical tests on each of the observed time points. With verison 2.0, capabilities have drastically improved, and bdots is able to fit parametric functions to any type of data observed in time. Along with methodological improvements, we have included more options in determining statistical significance in the differences of curves, utilizing a robust permutation testing framework when the assumptions of autocorrelation do not hold. In addition to methodology, a number of quality-of-life improvements have also been made, greatly simplifying syntax, creating more robust functions, and including a collection of useful methods for handling returned objects.

In summary, `bdots` has transitioned from a package focused exclusively on densly sampled timeseries assuming a limited number of functional forms to a robust framework for identifying time windows of significant difference across a wide breadth of timeseries-adjacent data.

3 Methodology and Overview

There are a few minor changes and additions in the underlying methodology used in the bdots package which we will briefly review here. For those interested in a more detailed description, see chapter 3 of my dissertation.

Actually need to nail down how I want to organize this. There is basically:

1. Creating group distributions
   (a) Getting fits
   (b) Bootstrapping for dist
2. Finding areas of significance
   (a) Permutation testing
   (b) FWER/oleson adjustment

3.1 Creating Group Distributions

Broadly, there are two steps to performing an analysis with the `bdots` package: fitting the curves to observed data and bootstrapping differences between groups. The first step involves specifying an underlying curve f, which is assumed to be parametric[1]. Along with the observed data y for each ith subject, `bdots`, via fitting with `gnls`, returns a set of parameters along with an estimate of their covariance.

[1]the option to include non-parametric functions is anticipated in the future work of this package. The process will be similar, however, with θ then representing the number and location of the knots for splines

$$F : f \times y_i \to N(\hat{\theta}_i, V_i), \tag{1}$$

where θ is a length-p vector representing the parameters of the function.

Once fits have been made, we are ready for testing the bootstrapped difference between curves. After specifying the groups of interest for analysis, two algorithms are implemented: a bootstrapping algorithm is used to determine the distribution of each group of curves, and either permutation testing or bootstrapping is used to specify regions of statistically significant differences, depending on the underlying assumptoins made. The algorithm for bootstrapping for each group is as follows:

1. For a group of size n, select n subjects from the group, *with replacement*. This controls for the between subject variability
2. For each selected subject, draw a set of parameters from the distribution $\theta_i^* \sim N(\hat{\theta}_i, V_i)$. This permits us to account for within subject variability
3. For each of the resampled θ_i^*, find the bth bootstrap estimate for the group $\theta_b = \frac{1}{n}\sum_{i=1}^n \theta_i^*$
4. Perform this sequence B times

The end results is a $B \times p$ matrix containing a bootstrapped sample of the group distribution for θ. Each row of this matrix is used to create a $1 \times T$ vector representing f_θ evaluted at T time points. This results in a $B \times T$ matrix representing a collection of bootstrapped curves evaluated at each time point, in total representing a bootstrapped distribution of the curves.

Next we attend to idenfiying regions in which a statistically significant difference between curves is present, choosing from one of the two methods currently available.

3.2 Evaluating significance differences

There are a number of methods included in the `bdots` package for identifying windows where time series differ significantly. First, there are a collection of FWER alpha adjustments, including Oleson's method (2017). Included here in the alpha adjustment category is also adjustments for FDR. New to version 2.0 are methods related to permutation testing. A review of the theoretical considerations, as well as underlying assumptions for each are briefly presented here.

3.2.1 Permutation Testing

maybe switch order of this and fwer

The simplest method implemented for idenfiying time-specific differences is permutation testing, ideal when minimal assumptions can be made on the observed data.

We begin by computing a t-statistic of the difference at each time point,

$$T(t) = \frac{|\overline{f}_1(t) - \overline{f}_2(t)|}{\sqrt{\frac{1}{n_2}\text{Var}(f_1(t)) + \frac{1}{n_2}\text{Var}(f_2(t))}}, \tag{2}$$

or, in the case of paired groups,

$$T(t) = \frac{\overline{f}_D(t)}{\sqrt{\frac{1}{n}\text{Var}(f_D(t))}}. \tag{3}$$

Next, we go about creating a null distribution against which to test our hypothesis that there is no difference between each group at each time point. We do this with permutations, the algorithm being as follows: for two groups, with n_1 and n_2 subjects in each:

1. Assign to each subject a label indicating group membership
2. Randomly shuffle the lables indicating membership, creating two new groups with n_1 and n_2 subjects in each
3. Recalculate the t-statistic, $T(t)$ and record the maximum of each permutation

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.

3.2.2 FWER Adjustment

In addition to permutation testing, there are also adjustments that can be made to control for the family-wise error rate. As was done with permutation testing, we begin by computing a t-statistic at each time point for the observed data,

$$T(t) = \frac{|\overline{f}_1(t) - \overline{f}_2(t)|}{\sqrt{\frac{1}{n_2}\text{Var}(f_1(t)) + \frac{1}{n_2}\text{Var}(f_2(t))}}, \tag{4}$$

or, again in the case of paired groups,

$$T(t) = \frac{\overline{f}_D(t)}{\sqrt{\frac{1}{n}\text{Var}(f_D(t))}}. \tag{5}$$

Unlike the case with the permuted data, we have no need for a null distribution, instead determining significance by considering the observed statistics against a modified α. Adjustments that can be made include all of the adjustments found in `stats::p.adjust`, and adjustment made on the false discovery rate (FDR), and finally the adjustment "oleson" used in the original `bdots` package. In short, the "oleson" adjustment makes use of an autocorrelation parameter to adjust for the highly correlated series of t-statistics. A full treatment of this methodology, along with a comparison to other FWER adjustments, is included in Oleson 2017.

4 Example Data

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 1: 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,

$$f_\theta(t) = b + \frac{p-b}{1+\exp\left(\frac{4s}{p-b}(x-t)\right)}, \tag{6}$$

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

5 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, ...)
```

Curve functions 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`. New here is `curveType`, taking as an argument an R call to a particular curve, for example the four parameter logistic, `bdots::logistic()`. This is done to self-contain any additional arguments associated with the fitting curve, for example the concavity of the double-Gaussian (`curveType = doubleGauss(concave = TRUE)`) or the number of degrees in a polynomial (`curveType = polynomial(degree = 5)`). A number of curves are included with the `bdots` package, including those for the four-parameter logistic, the double-Gaussian, an exponential curve, and polynomials of arbitrary degree. A detailed vignette on writing custom curves can be found with `vignette("bdots")`

Notably, `bdots` can now fit curves to an arbitrary number of groups at once, so long as all have the same parametric specification. Fitting a collection of curves to our vehicle data on all of the groups with the logistic function would look like

```
# Need to change these once I get real data
fit <- bfit(data = Vehicle, subject = "vehicle",
  time = "Time", y = "out", group = c("origin", "class", "color"),
  curveType = logistic(), cores, ...).
```

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`. One consequence of inheriting from `data.table`, we are able to utilize data.table syntax. Note, for example, the differences between `coef(fit)`, `coef(fit[group == "A",])`, and `coef(fit[group == "B",])`. This is especially helpful when looking to plot only a subset of the fitted curves, i.e., `plot(fit[group == "A",])`

Fit Codes need to decide what i'm going to do about this. would be nice if the metric used here was module, and it can be, just not right now because that would be a lot of extra work for nothing. I could just ignore this issue all togtetrher, indicate that this is for correlation and R2, and made absolutely no mention of how this possibly conflicts with other types of data

5.1 Worked example (fitting)

Need to decide on organization – later in paper, i have worked example using bootstrapping, along with summary and plots. Not sure if I should have this portion adjacent to it (that is, an entire worked example all in one place) or if i should move the bootstrapping section, along with this, to its own section all togther. the content, however, should more or less remain the same. Also, I technically have this written above as well

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

```
fit <- bfit(data = Vehicle, subject = "vehicle",
  time = "Time", y = "out", group = c("origin", "class", "color"),
  curveType = logistic())
```

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. A subset of this summary output is included here:

```
> summary(fit)

bdotsFit Summary

Curve Type: logistic
Formula: fixations ~ mini + (peak - mini)/(1 + exp(4 * slope * (cross - (time))/(peak - mini)))
Time Range: (0, 2000) [501 points]


Origin: foreign Vehicle: car Color: red
Num Obs:  25
Parameter Values:
         mini          peak         slope
  0.028636012   0.853162096   0.001868519
        cross
692.500921721
########################################
############### FITS ###################
########################################
```

```
AR1,        0.95 <= R2          -- 25
AR1,        0.80 < R2 <= 0.95 -- 0
AR1,        R2 < 0.8            -- 0
Non-AR1,    0.95 <= R2          -- 0
Non-AR1,    0.8 < R2 <= 0.95  -- 0
Non-AR1,    R2 < 0.8            -- 0
No Fit                          -- 0

All Fits
Num Obs:  200
Parameter Values:
         mini          peak         slope
  0.029165824   0.896574456   0.001688192
        cross
720.433606844
########################################
############### FITS ###################
########################################
AR1,        0.95 <= R2          -- 190
AR1,        0.80 < R2 <= 0.95 -- 9
AR1,        R2 < 0.8            -- 1
Non-AR1,    0.95 <= R2          -- 0
Non-AR1,    0.8 < R2 <= 0.95  -- 0
Non-AR1,    R2 < 0.8            -- 0
No Fit                          -- 0
```

The default plotting method plots each of the fitted subjects, including observed and fit data, see Figure 1.

6 Bootstrapping

Once fits have been made, we are ready to begin estimating the group distributions.

Like the fitting function, the bootstrapping process has been consolidated to a single function, `bboot` (previously `bdotsBoot`). The number of options included in the `bboot` function have expanded to include a new formula syntax for specifying the analysis of itnerest as well as to include options for permutation testing. A call to `bboot` takes the following form

```
bboot(formula, bdObj, Niter, alpha, padj = "oleson",
        cores = 0, permutation = FALSE, ...)
```

By default, significance is determined with an adjustment to the family-wise error rate, as was done in the original implementation of `bdots`, using method `padj = "oleson"`. When setting the argument `permutation = TRUE`, `padj` is ignored and permutation testing for regions of difference is used instead. Bootstrapping is still used for determining the group distribution, however.

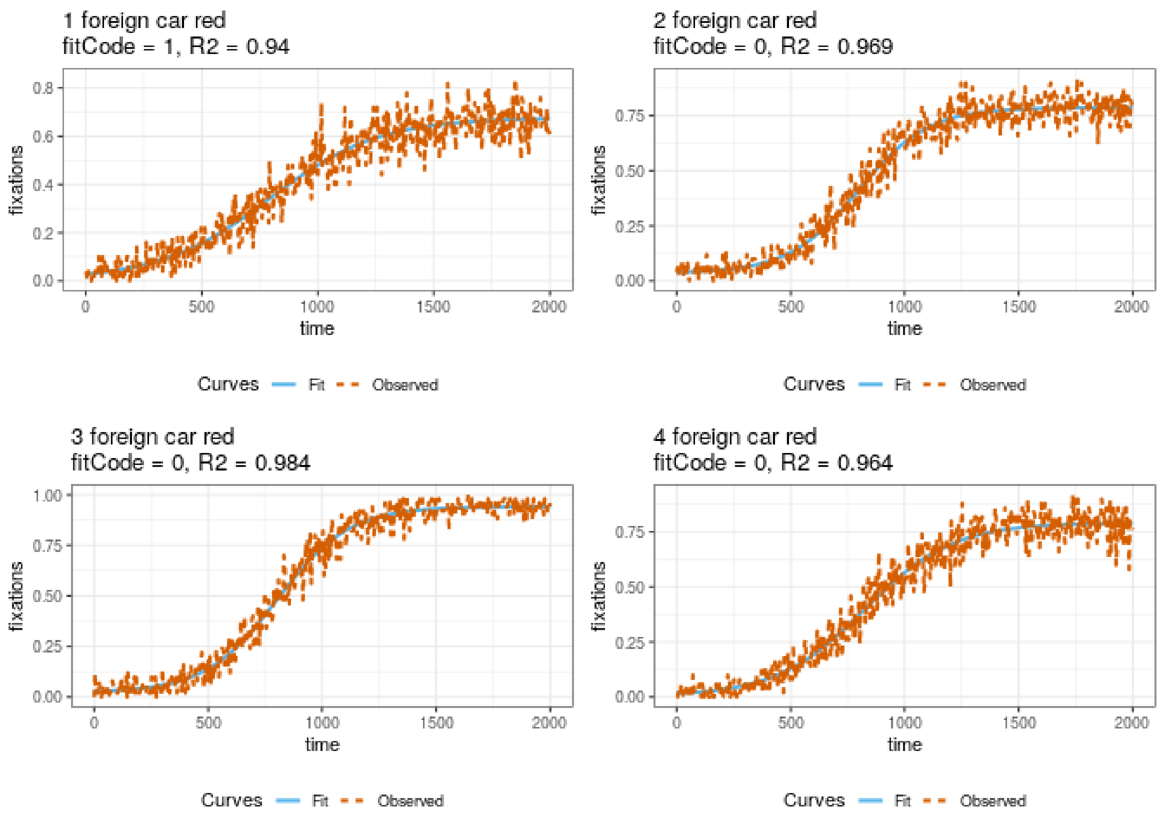

Figure 1: plot of fits, obviously this doesn't look great, placeholder for now

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.

6.1 Bootstrapping Formula

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.

Let's begin with a simple case: supposing we want to investigate the difference in outcome between foreign and domestic cars. Notionally, we would write

$$\texttt{y} \sim \texttt{Origin(foreign, domestic)}.$$

Specifically, note that this is of the form

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

Table 2: table of stuff

```
outcome ~ GroupName(value1, value2)[<- include this?]
```

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 cars, we would do this by including an extra term specifying the group and the desired value. In this case,

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

To compare only foreign and domestic *red* cars, we would add an additional term for color:

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

[this section needs rewritten still its not correct] In each of these cases, we have specifed particular groups or nesting of groups who's outcomes we wish to compare. Alternatively, we may be interested in comparing the *difference of differences.* For example, suppose we suspect that there may be some difference between red and blue vehicles, and that this difference may be different for cars compared to trucks. Whereas previously we were interested in comparing the differences in `y` between origin, we are now interested in comparing the differences of `y` between colors. Consequently, we will include this difference on the left hand side of the formula as our new outcome. This is done using `diffs` syntax as such:

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

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, Color(red, blue)) ~ Class(car, truck) + 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 the appendix.

6.2 Bootstrapping and Permutation

Once we have determined the groups we are interested in comparing, we are ready to call the `bboot` function. As detailed earlier, `bboot` is now able to perform permutation testing on curve differences in addition to bootstrapping. We will discuss each of these briefly in more detail.

Bootstrapping As in the original iteration of the package, `bdots` seeks to identify differences in curves without a priori specification of a time window. This is done as in (other paper), where curves are bootstrapped to create a sampling distribution, and a collection of t-tests is performed at each observed timepoint. The FWER is controlled against this series of tests by making an adjustment to the nominal α – the novel contribution of the original `bdots` package was implementing a correction based on an estimate of the autocorrelation of tests (oleson 2017). In addition to this adjustment, `bdots` also allows corrections to be made using a number of other methods, including bonferonni. Although not a correction towards FWER, corrections based on the false discovery rate (FDR) are also included. The type of correction made is specified with the `padj` argument in `bboot`.

Permutation In addition to the bootstrapping algorithm just described, `bdots` now also includes the option for running a permutation test to establish a null distribution on the differences between curves. This is done by setting the argument in `bboot(..., permutation = TRUE)`. `Niter` now refers to the number of permutations performed rather than the number of bootstraps. Additionally, when `permutation = TRUE` is set, the argument to `padj` is ignored. Permutation testing may have less power in some cases (explored in other paper), but is maximally robust and ideal for situations in which the number of assumptions made is limited.

6.3 Summary and Analysis

Let's begin first by running `bboot` using bootstrapping to compare the outcome between domestic cars and trucks using the FWER adjustment `padj = "oleson"`

```
boot <- bboot(y ∼ Vehicle(car, truck) + Origin(domestic), fit, padj = "oleson")
```

This returns an object of class `bdotsBootObj`. The default printing method for this is also its summary. Here, it looks like this:

```
> summary(boot)

bdotsBoot Summary

Curve Type: logistic
Formula: fixations ~ mini + (peak - mini)/(1 + exp(4 * slope * (cross - (time))/(peak - mini)))
Time Range: (0, 2000) [501 points]

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

Autocorrelation Estimate: 0.9872545
Alpha adjust method: oleson
Alpha: 0.05
Adjusted alpha: 0.001020216
Significant Intervals at adjusted alpha:
     [,1] [,2]
[1,]    0  212
[2,]  464  960
```

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 (for interaction terms), are the elements of the groups paired, and what grouping was used in determining the differences. The final section includes information on the bootstrapped differences. Also the adjustment `"oleson"` was used, included here is an estimate of the autocorrelation, as well as the value of the adjusted alpha term. 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

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

7 Extensions? Plots? I don't know!

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

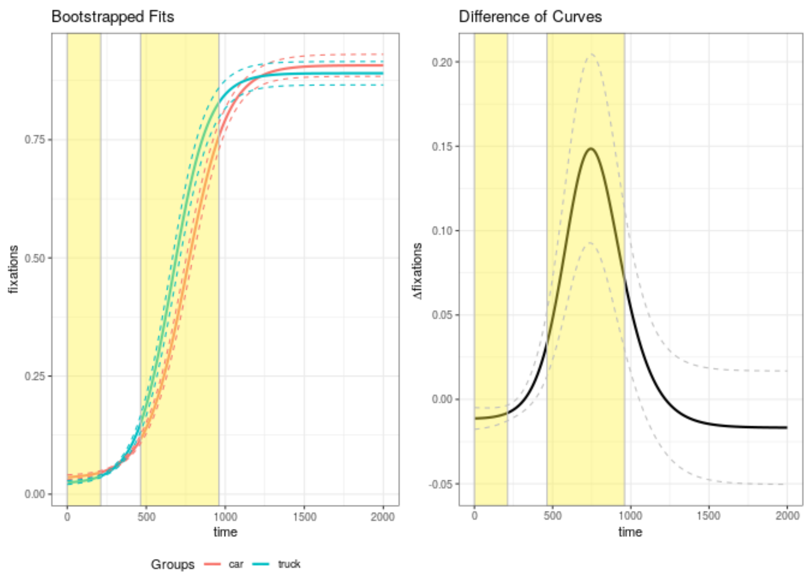

Figure 2: need to manually change title and legend size later

7.1 Non-homogenous sampling

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 observed time, or some custom range in between

```
<bfit, bboot, summary, plot> but for rats
```

`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.

7.2 Refitting

There are sometimes situations in which the fitted function returned by `bfit` is a poor fit. This 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:

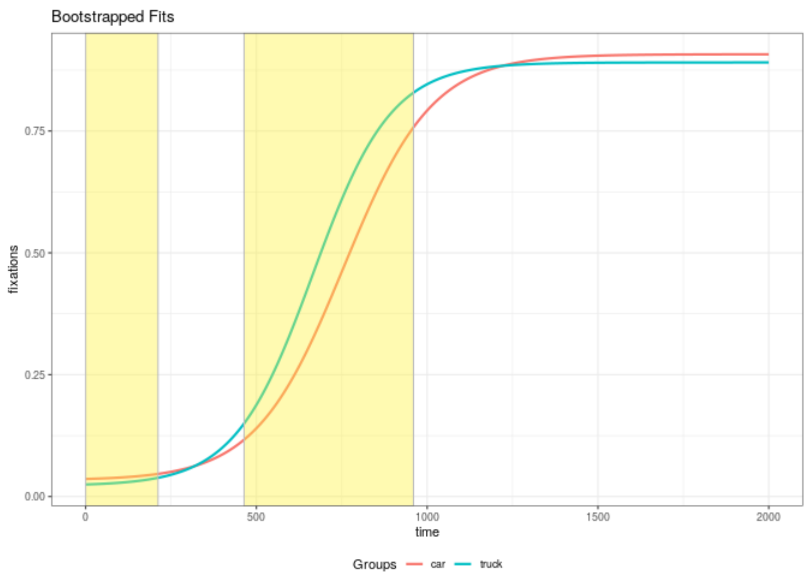

Figure 3: need to manually change title and legend size later and also maybe width of curves.

```
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. The `quickRefit` option allows the fitter to run automatically, jittering the previous set of parameters for each refitted subject and comparing the new fit to the previous, keeping the better of the two; `numRefits` indicates how many attempts the fitter should make in doing this. 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.

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. For example,

```
> brefit(fit, fitCode = 4)
Subject: 4
R2: 0.644
AR1: FALSE
rho: 0.9
fitCode: 5

 Model Parameters:
          mu           ht         sig1         sig2        base1 
848.00000000   0.28787879 456.00000000 256.00000000   0.00000000 
       base2 
  0.09090909 
```

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

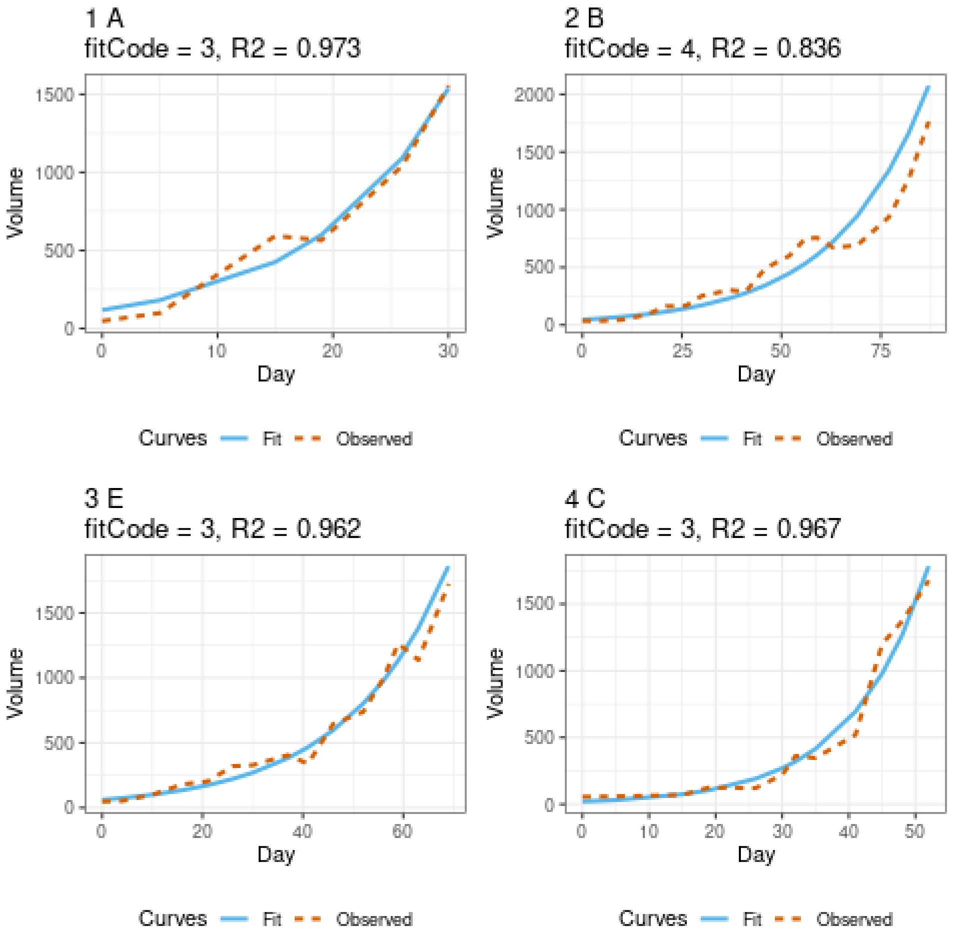

Figure 4: mousey data

```
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
```

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

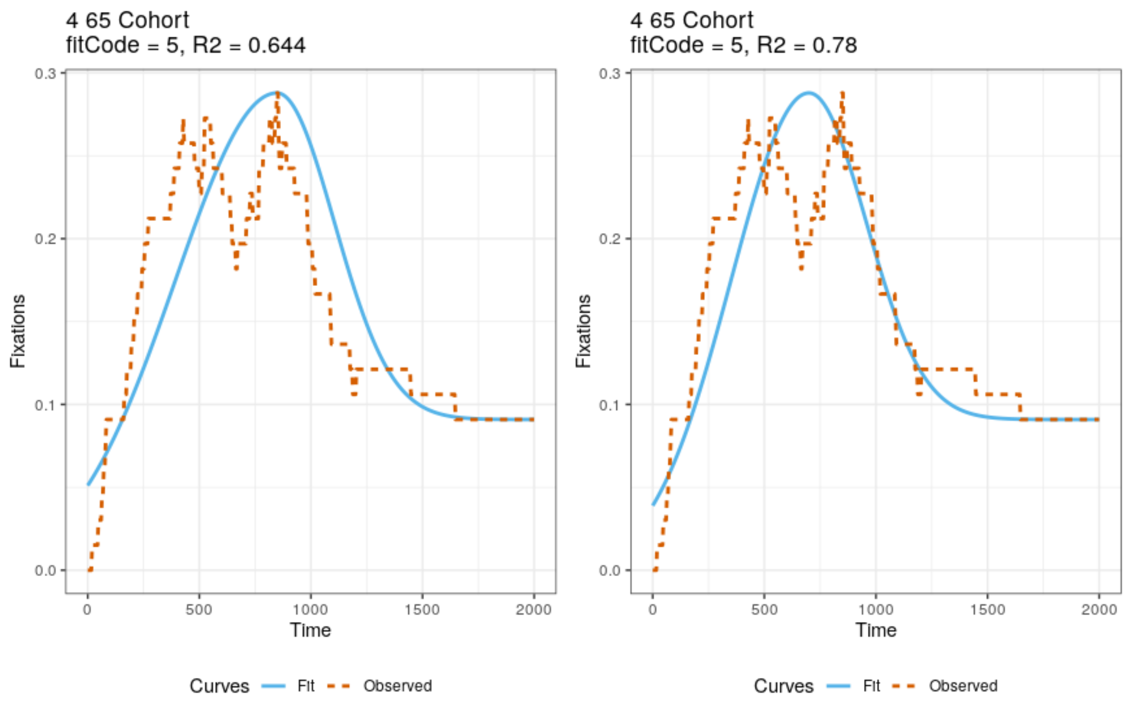

Figure 5: size is wrong on this, had to do interactively so can't save png also refitter sucks (do now have svg but will muck with it later)

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 where the user left off, pass the first refitted object back into the refitter as such:

```
refit <- brefit(fit, ...)
refit <- brefit(refit, ...)  # pass that shit back in
```

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 on 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.

Additionally, this can be done independent of the refitter with the ancillary function `bdRemove`.

7.3 User created curves

I know I mentioned this elsewhere, but I might erase it there and move a fuller discussion of it here

7.4 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

7.5 α Adjustment

This probably last section that needs anything special, and that is an update 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)
```

8 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 accomodate 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 programmer 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, accomodating 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

Appendix B - Fitting non-nested groups

(currently just copy pasted from the body of document)

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.