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: 0x55b748e91a38>
#> <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 0.01136364   Cohort    50
#>   2:       1    4      50 0.01136364   Cohort    50
#>   3:       1    8      50 0.01136364   Cohort    50
#>   4:       1   12      50 0.01136364   Cohort    50
#>   5:       1   16      50 0.02272727   Cohort    50
#>  ---
#> 497:       1 1984      50 0.02272727   Cohort    50
#> 498:       1 1988      50 0.02272727   Cohort    50
#> 499:       1 1992      50 0.02272727   Cohort    50
#> 500:       1 1996      50 0.02272727   Cohort    50
#> 501:       1 2000      50 0.02272727   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
#> 428.00000000   0.21590909 152.00000000 396.00000000   0.01136364   0.02272727
```

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,] 429.7595 0.1985978 159.8869 314.6389  0.009709772 0.03376106
#> [2,] 634.9292 0.2635044 303.8081 215.3845 -0.020636092 0.02892360
#> [3,] 647.0655 0.2543769 518.9632 255.9871 -0.213087597 0.01368195
#> [4,] 723.0551 0.2582110 392.9509 252.9381 -0.054827082 0.03197292
#> [5,] 501.4843 0.2247729 500.8605 158.4164 -0.331698893 0.02522686
#> [6,] 460.7152 0.3067659 382.7323 166.0833 -0.243308940 0.03992168

## "New" fit
head(coef(same_fit_different_day))
#>            mu        ht     sig1     sig2       base1      base2
#> [1,] 424.0311 0.1985000 154.2040 319.4740  0.01136408 0.03363656
#> [2,] 634.8093 0.2635148 303.6648 215.4307 -0.02058587 0.02893339
#> [3,] 646.9448 0.2544417 517.6521 255.9967 -0.21165093 0.01357057
#> [4,] 723.0861 0.2582117 393.0037 252.9037 -0.05485216 0.03197492
#> [5,] 501.6132 0.2247893 500.9494 158.3115 -0.33217172 0.02522935
#> [6,] 460.7371 0.3067971 382.5232 165.9932 -0.24285941 0.03993349
```