Diving Behaviour Analysis in R*

An Introduction to the diveMove Package

Sebastián P. Luque†

Contents

1 Introduction

Remarkable developments in technology for electronic data collection and archival have increased researchers' ability to study the behaviour of aquatic animals while reducing the effort involved and impact on study animals. For example, interest in the study of diving behaviour led to the development of minute time-depth recorders (TDRs) that can collect more than 15 MB of data on depth, velocity, light levels, and other parameters as animals move through their habitat. Consequently, extracting useful information from TDRs has become a time-consuming and tedious task. Therefore, there is an increasing need for efficient software to automate these tasks, without compromising the freedom to control critical aspects of the procedure.

There are currently several programs available for analyzing TDR data to study diving behaviour. The large volume of peer-reviewed literature based on results from these programs attests to their usefulness. However, none of them are in the free software domain, to the best of my knowledge, with all the disadvantages it entails. Therefore, the main motivation for writing diveMove was to provide an R package for diving behaviour analysis allowing for more flexibility and access to intermediate calculations. The advantage of this approach is that researchers have all the elements they need at their disposal to take the analyses beyond the standard information returned by the program.

The purpose of this article is to outline the functionality of diveMove, demonstrating its most useful features through an example of a typical diving behaviour analysis session. Further information can be obtained by reading the vignette that is included in the package (`vignette("diveMove")`) which is currently under development, but already shows basic usage of its main functions. diveMove is available from CRAN, so it can easily be installed using `install.packages()`.

2 Features

diveMove offers functions to perform the following tasks:

*An earlier version of this introduction to diveMove has been published in R News (Luque 2007)

†Contact: spluque@gmail.com. Comments for improvement are very welcome!

- Identification of wet vs. dry periods, defined by consecutive readings with or without depth measurements, respectively, lasting more than a user-defined threshold. Depending on the sampling protocol programmed in the instrument, these correspond to wet vs. dry periods, respectively. Each period is individually identified for later retrieval.
- Calibration of depth readings, which is needed to correct for shifts in the pressure transducer. This can be done using a `tcltk` graphical user interface (GUI) for chosen periods in the record, or by providing a value determined a priori for shifting all depth readings.
- Identification of individual dives, with their different phases (descent, bottom, and ascent), using various criteria provided by the user. Again, each individual dive and dive phase is uniquely identified for future retrieval.
- Calibration of speed readings using the method described by Blackwell et al. (1999), providing a unique calibration for each animal and deployment. Arguments are provided to control the calibration based on given criteria. Diagnostic plots can be produced to assess the quality of the calibration.
- Summary of time budgets for wet vs. dry periods.
- Dive statistics for each dive, including maximum depth, dive duration, bottom time, postdive duration, and summaries for each dive phases, among other standard dive statistics.
- `tcltk` plots to conveniently visualize the entire dive record, allowing for zooming and panning across the record. Methods are provided to include the information obtained in the points above, allowing the user to quickly identify what part of the record is being displayed (period, dive, dive phase).

Additional features are included to aid in analysis of movement and location data, which are often collected concurrently with *TDR* data. They include calculation of distance and speed between successive locations, and filtering of erroneous locations using various methods. However, `diveMove` is primarily a diving behaviour analysis package, and other packages are available which provide more extensive animal movement analysis features (e.g. `trip`).

The tasks described above are possible thanks to the implementation of three formal S4 classes to represent TDR data. Classes *TDR* and *TDRspeed* are used to represent data from TDRs with and without speed sensor readings, respectively. The latter class inherits from the former, and other concurrent data can be included with either of these objects. A third formal class (*TDRcalibrate*) is used to represent data obtained during the various intermediate steps described above. This structure greatly facilitates the retrieval of useful information during analyses.

3 Preliminary Procedures

As with other packages in R, to use the package we load it with the funtion `library`:

```
library(diveMove)


## Loading required package:  stats4

## This is diveMove 1.4.2.  For overview type
vignette("diveMove")
```

This makes the objects in the package available in the current R session. A short overview of the most important functions can be seen by running the examples in the package's help page:

```
example(diveMove)
```

Data Preparation

`TDR` data are essentially a time-series of depth readings, possibly with other concurrent parameters, typically taken regularly at a user-defined interval. Depending on the instrument and manufacturer, however, the files obtained may contain various errors, such as repeated lines, missing sampling intervals, and invalid data. These errors are better dealt with using tools other than R, such as `awk` and its variants, because such stream editors use much less memory than R for this type of problems, especially with the typically large files obtained from `TDR`s. Therefore, `diveMove` currently makes no attempt to fix these errors. Validity checks for the TDR classes, however, do test for time series being in increasing order.

Most TDR manufacturers provide tools for downloading the data from their TDRs, but often in a proprietary format. Fortunately, some of these manufacturers also offer software to convert the files from their proprietary format into a portable format, such as comma-separated-values (csv). At least one of these formats can easily be understood by R, using standard functions, such as `read.table()` or `read.csv()`. `diveMove` provides constructors for its two main formal classes to read data from files in one of these formats, or from simple data frames.

4 How to Represent TDR Data?

TDR is the simplest class of objects used to represent TDR data in diveMove. This class, and its *TDRspeed* subclass, stores information on the source file for the data, the sampling interval, the time and depth readings, and an optional data frame containing additional parameters measured concurrently. The only difference between *TDR* and *TDRspeed* objects is that the latter ensures the presence of a speed vector in the data frame with concurrent measurements. These classes have the following slots:

file: character,

dtime: numeric,

time: POSIXct,

depth: numeric,

concurrentData: data.frame

Once the TDR data files are free of errors and in a portable format, they can be read into a data frame, using e.g.:

```
fp <- file.path("data", "dives.csv")
sfp <- system.file(fp, package="diveMove")
```

```
srcfn <- basename(sfp)
tdrXcsv <- read.csv(sfp, sep=";")
```

and then put into one of the *TDR* classes using the function `createTDR()`. Note, however, that this approach requires knowledge of the sampling interval and making sure that the data for each slot are valid:

```
ddtt.str <- paste(tdrXcsv$date, tdrXcsv$time)
ddtt <- strptime(ddtt.str,
                 format="%d/%m/%Y %H:%M:%S")
time.posixct <- as.POSIXct(ddtt, tz="GMT")
tdrX <- createTDR(time=time.posixct,
                  depth=tdrXcsv$depth,
                  concurrentData=tdrXcsv[, -c(1:3)],
                  dtime=5, file=srcfn)
## Or a TDRspeed object, since we know we have
## speed measurements:
tdrX <- createTDR(time=time.posixct,
                  depth=tdrXcsv$depth,
                  concurrentData=tdrXcsv[, -c(1:3)],
                  dtime=5, file=srcfn,
                  speed=TRUE)
```

If the files are in *.csv format, these steps can be automated using the `readTDR()` function to create an object of one of the formal classes representing TDR data (*TDRspeed* in this case), and immediately begin using the methods provided:

```
fp <- file.path("data", "dives.csv")
sfp <- system.file(fp, package="diveMove")
tdrX <- readTDR(sfp, speed=TRUE, sep=";",
                na.strings="", as.is=TRUE)
plotTDR(tdrX)
```

Several arguments for `readTDR()` allow mapping of data from the source file to the different slots in `diveMove`'s classes, the time format in the input and the time zone attribute to use for the time readings.

Various methods are available for displaying TDR objects, including `show()`, which provides an informative summary of the data in the object, extractors and replacement methods for all the slots. There is a `plotTDR()` method (Figure 1) for both *TDR* and *TDRspeed* objects. The *interact* argument allows for suppression of the `tcltk` interface. Information on these methods is available from *methods?TDR*.

TDR objects can easily be coerced to data frame (`as.data.frame()` method), without losing information from any of the slots. *TDR* objects can additionally be coerced to *TDRspeed*, whenever it makes sense to do so, using an *as.TDRspeed()* method.

5 Identification of Activities at Various Scales

One of the first steps of dive analysis is to identify dry and wet periods in the record. This is done with function `calibrateDepth()`. Wet periods are those

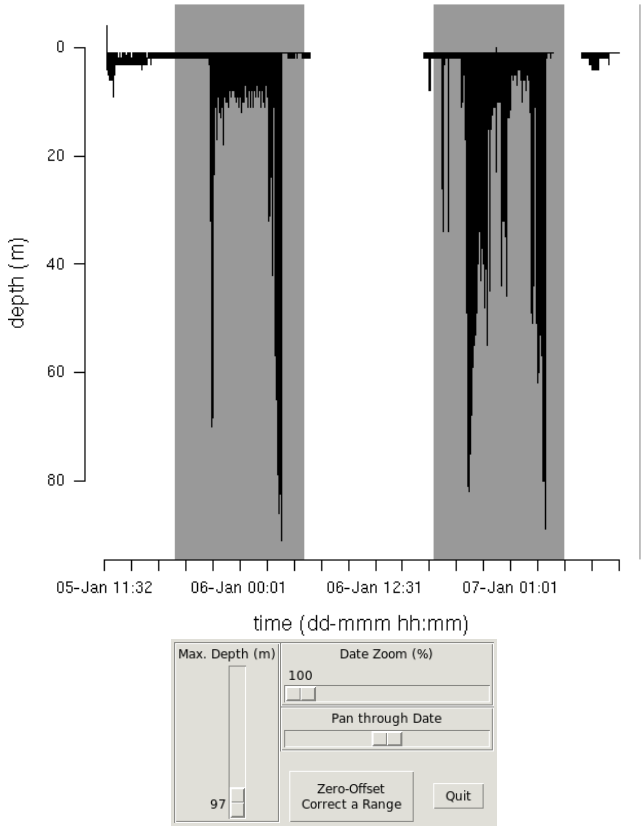

Figure 1. The *plotTDR()* method for *TDR* objects produces an interactive plot of the data, allowing for zooming and panning.

with depth readings, dry periods are those without them. However, records may have aberrant missing depth that should not define dry periods, as they are usually of very short duration[1]. Likewise, there may be periods of wet activity that are too short to be compared with other wet periods, and need to be excluded from further analyses. These aspects can be controlled by setting the arguments *dry.thr* and *wet.thr* to appropriate values.

The next step involves correcting depth for shifts in the pressure transducer, so that surface readings correspond to zero. Such shifts are usually constant for an entire deployment period, but there are cases where the shifts vary within a particular deployment, so shifts remain difficult to detect and dives are often missed. Therefore, a visual examination of the data is often the only way to detect the location and magnitude of the shifts. Visual adjustment for shifts in depth readings is tedious, but has many advantages which may save time during later stages of analysis. These advantages include increased understanding of the data, and early detection of obvious problems in the records, such as instrument malfunction during certain intervals, which should be excluded from analysis.

Function `calibrateDepth()` takes a *TDR* object to perform three basic tasks: 1. identify wet and dry periods, 2. zero-offset correct (ZOC) the data, and 3. identify all dives in the record and their phases. ZOC can be done using one of three methods: "visual", "offset", and "filter". The first one ("visual") is the default method, which let's the user perform the correction interactively, using the `tcltk` package:

```
dcalib <- calibrateDepth(tdrX)
```

This command brings up a plot with `tcltk` controls allowing to zoom in and out, as well as pan across the data, and adjust the `depth` scale. Thus, an appropriate time window with a unique surface depth value can be displayed. This allows the user to select a `depth` scale that is small enough to resolve the surface value using the mouse. Clicking on the ZOC button waits for two clicks: i) the coordinates of the first click define the starting time for the window to be ZOC'ed, and the depth corresponding to the surface, ii) the second click defines the

[1] They may result from animals resting at the surface of the water long enough to dry the sensors.

end time for the window (i.e. only the x coordinate has any meaning). This procedure can be repeated as many times as needed. If there is any overlap between time windows, then the last one prevails. However, if the offset is known a priori, method "offset" lets the user specify this value as the argument *offset* to `calibrateDepth()`. For example, preliminary inspection of object `tdrX` would have revealed a 3 m offset, and we could have simply called (without plotting):

```
dcalib <- calibrateDepth(tdrX,
                         zoc.method="offset",
                         offset=3)
```

A third method ("filter") implements a smoothing/filtering mechanism where running quantiles can be applied to depth measurements sequentially, using `.depth.filter`. It relies on the `caTools` package. This method is still under development, but reasonable results can already be achieved by applying two filters, the first one using a running median with a narrow window to denoise the time series, followed by a running low quantile using a wide time window. The integer vector given as argument k specifies the width of the moving window(s), where k_i is the width for the i^{th} filter in units of the sampling interval of the *TDR* object. Similarly, the integer vector given as argument *probs* specifies the quantile for each filter, where $probs_i$ is the quantile for the i^{th} filter. Smoothing/filtering can be performed within specified minimum and maximum depth bounds using argument *depth.bounds*[2], in cases where surface durations are relatively brief separated by long periods of deep diving. These cases usually require large windows, and using depth bounds helps to stabilize the surface signal. Further details on this method are provided by Luque and Fried (2011).

```
dcalib <- calibrateDepth(tdrX,
                         zoc.method="filter",
                         k=c(3, 5760),
                         probs=c(0.5, 0.02),
                         na.rm=TRUE)
```

Once the whole record has been zero-offset corrected, remaining depths below zero, are set to zero, as these are assumed to indicate values at the surface.

[2]Defaults to the depth range

Finally, `calibrateDepth()` identifies all dives in the record, according to a minimum depth criterion given as its *dive.thr* argument. The value for this criterion is typically determined by the resolution of the instrument and the level of noise close to the surface. Thus, dives are defined as departures from the surface to maximal depths below *dive.thr* and the subsequent return to the surface. Each dive may subsequently be referred to by an integer number indicating its position in the time series.

Dive phases are also identified at this last stage, and is done by fitting one of two cubic spline models to each dive: i) unimodal regression (default) ii) smoothing spline and then using evaluating the first derivative to determine where phases begin/end. Detection of dive phases is thus controlled by four arguments: a critical quantile for rates of vertical descent (*descent.crit.q*), a critical quantile for rates of ascent (*ascent.crit.q*), a smoothing parameter (*smooth.par*, relevant only for smoothing spline model (i)), and a factor (*knot.factor*) that multiplies the duration of the dive to obtain the number of knots at which to evaluate the derivative of the smoothing spline. The first two arguments are used to define the rate of descent below which the descent phase is deemed to have ended, and the rate of ascent above which the ascent phase is deemed to have started, respectively. The rates are obtained by evaluating the derivative of the spline at a number of knots placed regularly throughout the dive. Descent is deemed to have ended at the *first* minimum derivative, and the nearest input time observation is considered to indicate the end of descent. The sign of the comparisons is reversed for detecting the ascent.

A more refined call to `calibrateDepth()` for object `tdrX` may be:

```
dcalib <- calibrateDepth(tdrX, dive.thr=3,
                         zoc.method="offset",
                         offset=3,
                         descent.crit.q=0.01,
                         ascent.crit.q=0,
                         knot.factor=20)


## Record is complete
## 7 phases detected

## 426 dives detected
```

The result (value) of this function is an object of

class *TDRcalibrate*, where all the information obtained during the tasks described above are stored.

6 How to Represent Calibrated TDR Data?

Objects of class TDRcalibrate contain the following slots, which store information during the major procedures performed by `calibrateDepth()`:

call: `TDR`. The call used to generate the object.

tdr: `TDR`. The object which was calibrated.

gross.activity: *list*. This list contains four components with details on wet/dry activities detected, such as start and end times, durations, and identifiers and labels for each activity period. Five activity categories are used for labelling each reading, indicating dry (L), wet (W), underwater (U), diving (D), and brief wet (Z) periods. However, underwater and diving periods are collapsed into wet activity at this stage (see below).

dive.activity: *data.frame*. This data frame contains three components with details on the diving activities detected, such as numeric vectors identifiying to which dive and post-dive interval each reading belongs to, and a factor labelling the activity each reading represents. Compared to the `gross.activity` slot, the underwater and diving periods are discerned here.

dive.phases: *factor*. This identifies each reading with a particular dive phase. Thus, each reading belongs to one of descent, descent/bottom, bottom, bottom/ascent, and ascent phases. The descent/bottom and bottom/ascent levels are useful for readings which could not unambiguously be assigned to one of the other levels.

dive.models: *list*. This list contains all the details of the modelling process used to identifies dive phases. Each member of this list consists of objects of class *diveModel*, for which important methods are available.

dry.thr: *numeric*.

wet.thr: *numeric*.

dive.thr: *numeric*. These last three slots store information given as arguments to `calibrateDepth()`, documenting criteria used during calibration.

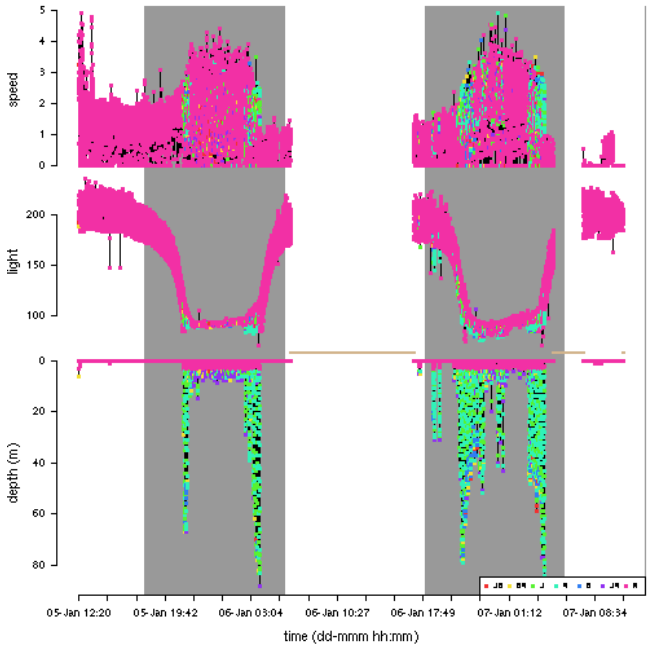

Figure 2. The `plotTDR()` method for *TDRcalibrate* objects displays information on the major activities identified throughout the record (wet/dry periods here).

speed.calib.coefs: *numeric*. If the object calibrated was of class *TDRspeed*, then this is a vector of length 2, with the intercept and the slope of the speed calibration line (see below).

All the information contained in each of these slots is easily accessible through extractor methods for objects of this class (see class?TDRcalibrate). An appropriate *show()* method is available to display a short summary of such objects, including the number of dry and wet periods identified, and the number of dives detected.

The *TDRcalibrate plotTDR()* method for these objects allows visualizing the major wet/dry activities throughout the record (Figure 2):

```
plotTDR(dcalib, concurVars=c("speed", "light"),
        surface=TRUE)
```

The `dcalib` object contains a *TDRspeed* object in its `tdr` slot, and speed is plotted by default in this case. Additional measurements obtained concurrently can also be plotted using the *concurVars* argument. Titles for the depth axis and the concurrent parameters use separate arguments; the former

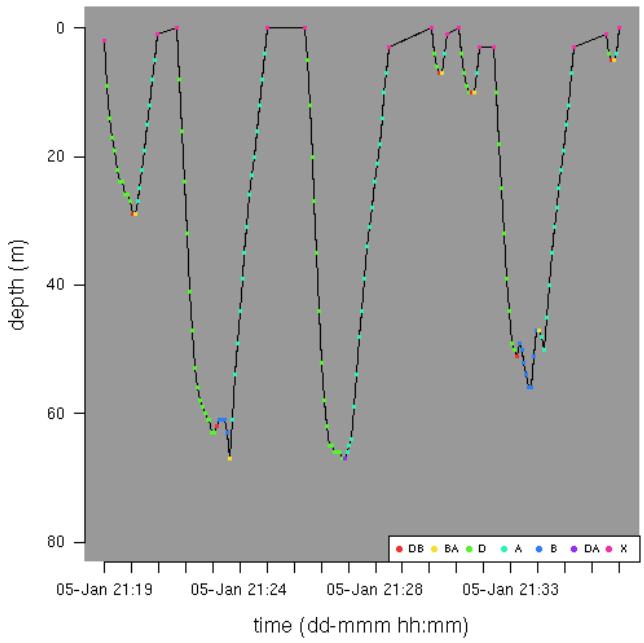

Figure 3. The *plotTDR()* method for *TDRcalibrate* objects can also display information on the different activities during each dive record (descent=D, descent/bottom=DB, bottom=B, bottom/ascent=BA, ascent=A, X=surface).

uses *ylab.depth*, while the latter uses *concurVarTitles*. Convenient default values for these are provided. The *surface* argument controls whether post-dive readings should be plotted; it is `FALSE` by default, causing only dive readings to be plotted which saves time plotting and re-plotting the data. All plot methods use the underlying `plotTD()` function, which has other useful arguments that can be passed from these methods.

A more detailed view of the record can be obtained by using a combination of the *diveNo* and the *labels* arguments to this *plotTDR()* method. This is useful if, for instance, closer inspection of certain dives is needed. The following call displays a plot of dives 2 through 8 (Figure 3):

```
plotTDR(dcalib, diveNo=2:8, what="phases")
```

The *labels* argument allows the visualization of the identified dive phases for all dives selected. The same information can also be obtained with the *extractDive()* method for *TDRcalibrate* objects:

```
extractDive(dcalib, diveNo=2:8)
```

Other useful extractors include: *getGAct()* and *getDAct()*. These methods extract the whole `gross.activity` and `dive.activity`, respectively, if given only the `TDRcalibrate` object, or a particular component of these slots, if supplied a string with the name of the component. For example: `getGAct(dcalib, "activity")` would retrieve the factor identifying each reading with a wet/dry activity and `getDAct(dcalib, "dive.activity")` would retrieve a more detailed factor with information on whether the reading belongs to a dive or a brief aquatic period. Below is a demonstration of these methods.

getTDR(): This method simply takes the *TDRcalibrate* object as its single argument and extracts the *TDR* object[3]:

```
getTDR(dcalib)


## Time-Depth Recorder data -- Class TDRspeed object
##   Source File           : dives.csv
##   Sampling Interval (s): 5
##   Number of Samples     : 34199
##   Sampling Begins       : 2002-01-05 11:32:00
##   Sampling Ends         : 2002-01-07 11:01:50
##   Total Duration (d)    : 1.979
##   Measured depth range  : [0, 88]
##   Other variables       :  light temperature speed
```

getGAct(): There are two methods for this generic, allowing access to a list with details about all wet/dry periods found. One of these extracts the entire *list* (output omitted for brevity):

```
getGAct(dcalib)
```

The other provides access to particular elements of the *list*, by their name. For example, if we are interested in extracting only the vector that tells us to which period number every row in the record belongs to, we would issue the command:

```
getGAct(dcalib, "phase.id")
```

Other elements that can be extracted are named "activity", "begin", and "end", and can be extracted in a similar fashion. These elements correspond to the activity performed for each reading (see `?detPhase` for a description of the labels for each activity), the

[3] In fact, a *TDRspeed* object in this example

beginning and ending time for each period, respectively.

getDAct(): This generic also has two methods; one to extract an entire data frame with details about all dive and postdive periods found (output omitted):

```
getDAct(dcalib)
```

The other method provides access to the columns of this data frame, which are named "dive.id", "dive.activity", and "postdive.id". Thus, providing any one of these strings to getDAct, as a second argument will extract the corresponding column.

getDPhaseLab(): This generic function extracts a factor identifying each row of the record to a particular dive phase (see `?detDive` for a description of the labels of the factor identifying each dive phase). Two methods are available; one to extract the entire factor, and the other to select particular dive(s), by its (their) index number, respectively (output omitted):

```
getDPhaseLab(dcalib)
getDPhaseLab(dcalib, 20)
```

```
dphases <- getDPhaseLab(dcalib, c(100:300))
```

The latter method is useful for visually inspecting the assignment of points to particular dive phases. More information about the dive phase identification procedure can be gleaned by using the *plotDiveModel* (Figure 4):

```
plotDiveModel(dcalib, diveNo=260)
```

Another generic function that allows the subsetting of the original *TDR* object to a single a dive or group of dives' data:

```
sealX <- extractDive(dcalib, diveNo=c(100:300))
sealX
```

```
## Time-Depth Recorder data -- Class TDRspeed object
##   Source File           : dives.csv
##   Sampling Interval (s): 5
##   Number of Samples     : 1757
##   Sampling Begins       : 2002-01-05 23:40:20
##   Sampling Ends         : 2002-01-06 23:04:45
##   Total Duration (d)    : 0.9753
##   Measured depth range  : [0, 88]
##   Other variables       :  light temperature speed
```

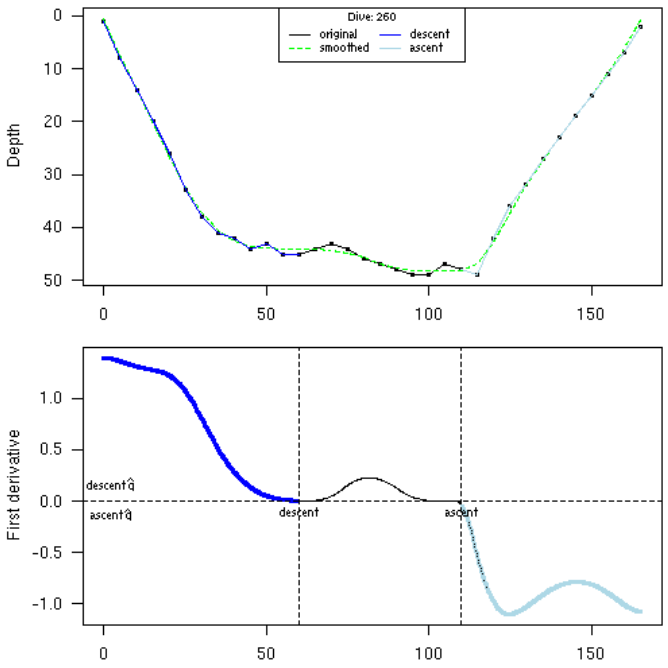

Figure 4. Details of the process of identification of dive phases shown by *plotDiveModel*, which has methods for objects of class *TDRcalibrate* and *diveModel*.

As can be seen, the function `extractDive` takes a *TDRcalibrate* object and a vector indicating the dive numbers to extract, and returns a *TDR* object containing the subsetted data. Once a subset of data has been selected, it is possible to plot them and pass the factor labelling dive phases as the argument *phaseCol* to the *plot* method[4]:

```
plotTDR(sealX, phaseCol=dphases)
```

With the information obtained during this calibration procedure, it is possible to calculate dive statistics for each dive in the record.

7 Dive Summaries

A table providing summary statistics for each dive can be obtained with the function `diveStats()` (Figure 5).

`diveStats()` returns a data frame with the final summaries for each dive (Figure 5), providing the

[4]The function that the method uses is actually `plotTD`, so all the possible arguments can be studied by reading the help page for `plotTD`

```
tdrXSumm1 <- diveStats(dcalib)

## Warning in (function (..., deparse.level = 1) :  number of columns of result is not a multiple of vector length
(arg 1)

names(tdrXSumm1)

##  [1] "begdesc"             "enddesc"             "begasc"              "desctim"
##  [5] "botttim"             "asctim"              "divetim"             "descdist"
##  [9] "bottdist"            "ascdist"             "bottdep.mean"        "bottdep.median"
## [13] "bottdep.sd"          "maxdep"              "desc.tdist"          "desc.mean.speed"
## [17] "desc.angle"          "bott.tdist"          "bott.mean.speed"     "asc.tdist"
## [21] "asc.mean.speed"      "asc.angle"           "postdive.dur"        "postdive.tdist"
## [25] "postdive.mean.speed" "descD.min"           "descD.1stqu"         "descD.median"
## [29] "descD.mean"          "descD.3rdqu"         "descD.max"           "descD.sd"
## [33] "bottD.min"           "bottD.1stqu"         "bottD.median"        "bottD.mean"
## [37] "bottD.3rdqu"         "bottD.max"           "bottD.sd"            "ascD.min"
## [41] "ascD.1stqu"          "ascD.median"         "ascD.mean"           "ascD.3rdqu"
## [45] "ascD.max"            "ascD.na.s"           "ascD.sd"

tbudget <- timeBudget(dcalib, ignoreZ=TRUE)
head(tbudget, 4)

##   phase.no activity                 beg                 end
## 1        1        L 2002-01-05 11:32:00 2002-01-05 11:39:40
## 2        2        W 2002-01-05 11:39:45 2002-01-06 06:30:00
## 3        3        L 2002-01-06 06:30:05 2002-01-06 17:01:10
## 4        4        W 2002-01-06 17:01:15 2002-01-07 05:00:30

trip.labs <- stampDive(dcalib, ignoreZ=TRUE)
tdrXSumm2 <- data.frame(trip.labs, tdrXSumm1)
names(tdrXSumm2)

##  [1] "phase.no"            "activity"            "beg"                 "end"
##  [5] "begdesc"             "enddesc"             "begasc"              "desctim"
##  [9] "botttim"             "asctim"              "divetim"             "descdist"
## [13] "bottdist"            "ascdist"             "bottdep.mean"        "bottdep.median"
## [17] "bottdep.sd"          "maxdep"              "desc.tdist"          "desc.mean.speed"
## [21] "desc.angle"          "bott.tdist"          "bott.mean.speed"     "asc.tdist"
## [25] "asc.mean.speed"      "asc.angle"           "postdive.dur"        "postdive.tdist"
## [29] "postdive.mean.speed" "descD.min"           "descD.1stqu"         "descD.median"
## [33] "descD.mean"          "descD.3rdqu"         "descD.max"           "descD.sd"
## [37] "bottD.min"           "bottD.1stqu"         "bottD.median"        "bottD.mean"
## [41] "bottD.3rdqu"         "bottD.max"           "bottD.sd"            "ascD.min"
## [45] "ascD.1stqu"          "ascD.median"         "ascD.mean"           "ascD.3rdqu"
## [49] "ascD.max"            "ascD.na.s"           "ascD.sd"
```

Figure 5. Per-dive summaries can be obtained with functions `diveStats()`, and a summary of time budgets with `timeBudget()`. `diveStats()` takes a *TDRcalibrate* object as a single argument (object `dcalib` above, see text for how it was created).

following information:

- The time of start of the dive, the end of descent, and the time when ascent began.
- The total duration of the dive, and that of the descent, bottom, and ascent phases.
- The vertical distance covered during the descent, the bottom (a measure of the level of "wiggling", i.e. up and down movement performed during the bottom phase), and the vertical distance covered during the ascent.
- The maximum depth attained.
- The duration of the post-dive interval.

A summary of time budgets of wet vs. dry periods can be obtained with `timeBudget()`, which returns a data frame with the beginning and ending times for each consecutive period (Figure 5). It takes a *TDRcalibrate* object and another argument (*ignoreZ*) controlling whether aquatic periods that were briefer than the user-specified threshold[5] should be collapsed within the enclosing period of dry activity.

These summaries are the primary goal of `diveMove`, but they form the basis from which more elaborate and customized analyses are possible, depending on the particular research problem. These include investigation of descent/ascent rates based on the depth profiles, and bout structure analysis. Some of these will be implemented in the future.

In the particular case of *TDRspeed* objects, however, it may be necessary to calibrate the speed readings before calculating these statistics.

8 Calibrating Speed Sensor Readings

Calibration of speed sensor readings is performed using the procedure described by Blackwell et al. (1999). Briefly the method rests on the principle that for any given rate of depth change, the lowest measured speeds correspond to the steepest descent angles, i.e. vertical descent/ascent. In this case, measured speed and rate of depth change are expected to be equal. Therefore, a line drawn through the bottom edge of the distribution of observations in a plot of measured speed vs. rate of depth change would provide a calibration line. The calibrated speeds, therefore, can be calculated by reverse estimation of rate of depth change from the regression line.

`diveMove` implements this procedure with function `calibrateSpeed()`. This function performs the following tasks:

1. Subset the necessary data from the record. By default only data corresponding to depth changes > 0 are included in the analysis, but higher constraints can be imposed using the z argument. A further argument limiting the data to be used for calibration is *bad*, which is a vector with the minimum *rate* of depth change and minimum speed readings to include in the calibration. By default, values > 0 for both parameters are used.
2. Calculate the binned bivariate kernel density and extract the desired contour. Once the proper data were obtained, a bivariate normal kernel density grid is calculated from the relationship between measured speed and rate of depth change (using the `KernSmooth` package). The choice of bandwidths for the binned kernel density is made using *bw.nrd*. The *contour.level* argument to `calibrateSpeed()` controls which particular contour should be extracted from the density grid. Since the interest is in defining a regression line passing through the lower densities of the grid, this value should be relatively low (it is set to 0.1 by default).
3. Define the regression line passing through the lower edge of the chosen contour. A quantile regression through a chosen quantile is used for this purpose. The quantile can be specified using the *tau* argument, which is passed to the rq() function in package `quantreg`. *tau* is set to 0.1 by default.
4. Finally, the speed readings in the *TDR* object are calibrated.

As recognized by Blackwell et al. (1999), the advantage of this method is that it calibrates the instrument based on the particular deployment conditions (i.e. controls for effects of position of the instrument on the animal, and size and shape of the instrument, relative to the animal's morphometry, among others). However, it is possible to supply

[5] This corresponds to the value given as the *wet.thr* argument to `calibrateDepth()`.

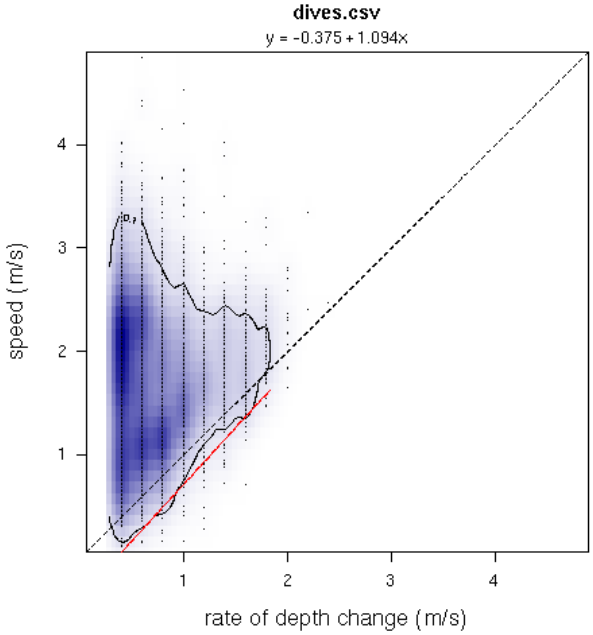

Figure 6. The relationship between measured speed and rate of depth change can be used to calibrate speed readings. The line defining the calibration for speed measurements passes through the bottom edge of a chosen contour, extracted from a bivariate kernel density grid.

the coefficients of this regression if they were estimated separately; for instance, from an experiment. The argument *coefs* can be used for this purpose, which is then assumed to contain the intercept and the slope of the line. `calibrateSpeed()` returns a *TDRcalibrate* object, with calibrated speed readings included in its `tdr` slot, and the coefficients used for calibration.

For instance, to calibrate speed readings using the 0.1 quantile regression of measured speed vs. rate of depth change, based on the 0.1 contour of the bivariate kernel densities, and including only changes in depth > 1, measured speeds and rates of depth change > 0:

```
vcalib <- calibrateSpeed(dcalib, tau=0.1,
                         contour.level=0.1,
                         z=1, bad=c(0, 0),
                         cex.pts=0.2)
```

This call produces the plot shown in Figure 6, which can be suppressed by the use of the logical argument *plot*. Calibrating speed readings allows for the meaningful interpretation of further parameters calculated by `diveStats()`, whenever a *TDRspeed* object was found in the *TDRcalibrate* object:

- The total distance travelled, mean speed, and diving angle during the descent and ascent phases of the dive.
- The total distance travelled and mean speed during the bottom phase of the dive, and the post-dive interval.

9 Bout Detection

Diving behaviour often occurs in bouts for several species, so `diveMove` implements procedures for defining bout ending criteria (Langton et al. 1995; Luque and Guinet 2007). Please see `?bouts2.mle` and `?bouts2.nls` for examples of 2-process models.

10 Summary

The `diveMove` package provides tools for analyzing diving behaviour, including convenient methods for the visualization of the typically large amounts of data collected by `TDR`s. The package's main strengths are its ability to:

1. identify wet vs. dry periods,
2. calibrate depth readings,
3. identify individual dives and their phases,
4. summarize time budgets,
5. calibrate speed sensor readings,
6. provide basic summaries for each dive identified in `TDR` records, and
7. provide tools for identification of dive bout end criteria.

Formal `S4` classes are supplied to efficiently store `TDR` data and results from intermediate analysis, making the retrieval of intermediate results readily available for customized analysis. Development of the package is ongoing, and feedback, bug reports, or other comments from users are very welcome.

Acknowledgements

Many of the ideas implemented in this package developed over fruitful discussions with my mentors John P.Y. Arnould, Christophe Guinet, and Edward H. Miller. I would like to thank Laurent Dubroca who wrote draft code for some of `diveMove`'s functions. I am also greatly endebted to the regular contributors to the R-help newsgroup who helped me solve many problems during development.

References

S. Blackwell, C. A. Haverl, B. J. Le Boeuf, and D. P. Costa. A method for calibrating swim-speed recorders. *Mar Mamm Sci*, 15(3):894–905, 1999.

S. D. Langton, D. Collett, and R. M. Sibly. Splitting behaviour into bouts; a maximum likelihood approach. *Behaviour*, 132:781–799, 1995.

S. P. Luque. Diving behaviour analysis in R. *R News*, 7:8–14, 2007.

S. P. Luque and R. Fried. Recursive filtering for zero offset correction of diving depth time series with gnu r package divemove. *PLoS ONE*, 6(1):e15850, 2011. doi: doi:10.1371/journal.pone.0015850.

S. P. Luque and C. Guinet. A maximum likelihood approach for identifying dive bouts improves accuracy, precision, and objectivity. *Behaviour*, 144: 1315–1332, 2007.