diveMove: dive analysis in R

Sebastián P. Luque[*]

DRAFT

Contents

1 Introduction

Dive analysis usually involves handling of large amounts of data, as new instruments allow for frequent sampling of variables over long periods of time. The aim of this package is to make this process more efficient for summarizing and extracting information gathered by time-depth recorders (TDRs, hereafter). The principal motivation

[*]Contact: `spluque@gmail.com`. Comments for improvement are very welcome!

Table 1. Sample TDR file structure.

date	time	depth	light	temperature	speed
16/02/2004	14:30:00	12	200	8.4	1.44
16/02/2004	14:30:05	15	180	8.0	1.75
16/02/2004	14:30:10	19	170	7.6	1.99
...	...	...	...	...	...

for developing `diveMove` was to provide more flexibility during the various stages of analysis than that offered by popular commercial software. This is achieved by making the results from intermediate calculations easily accessible, allowing the user to make his/her own summaries beyond the default choices the package provides. The following sections of this vignette illustrate a typical work flow during analysis of TDR data, using the `dives` data available in `diveMove` as an example.

DRAFT

2 Starting up

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

```
> library(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)
```

3 Reading Input Files

Input files must be simple, comma-delimited text files[1]. The order of columns is not significant, as the column numbers indicating the variables of interest can be supplied as arguments. Table 1 shows the file structure that readTDR assumes by default, which is a standard structure of files from common TDR models.

Depending on the TDR model, speed may be omitted.

To read the file into R , use the function `readTDR`:

```
> sealX <- readTDR(system.file(file.path("data",
+     "dives.csv"), package = "diveMove"), speed = TRUE)
```

Read the help page for `readTDR` using `?readTDR` following common R help facilities.

[1]The extension does not matter, but conventionally these files have a .csv extension

Thus, data could have been subsampled at a larger interval than that in the original file, so that the time interval between readings is 10 s:

```
> sealX <- readTDR(system.file(file.path("data",
+     "dives.csv"), package = "diveMove"), speed = TRUE,
+     subsamp = 10)
```

But since the original 5 s interval (which is the default value for *subsamp*) is what will be used for the subsequent sections, it is recreated it here:

```
> sealX <- readTDR(system.file(file.path("data",
+     "dives.csv"), package = "diveMove"), speed = TRUE)
```

The format in which date and time should be interpreted can be controlled with the argument *dtformat*. If the data are already available in the R session, e.g. as a `data frame`, then the function `createTDR` can be used to convert it to a form that facilitates further analyses.

Both of these functions store the data in an object of class *TDR* or *TDRspeed*, which hold information on the source file and sampling interval, in addition to the variables described above. Which of these objects is created is determined by the *speed*.

DRAFT

4 Extracting Information from TDR and TDRspeed Objects

For convenience, extractor methods are available to access the different slots from objects of these classes. The standard *show* method will display the usual overview information on the object:

```
> sealX

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 : [ -4 , 91 ]
  Other variables      : light temperature speed
```

Other extractor methods are named after the component they extract: *getTime*, *getDepth*, *getSpeed*, and *getDtime*, where the latter extracts the sampling interval. The *plotTDR* method brings up a plot of the data covering the entire record, although a `tcltk` widget provides controls for zooming and panning to any particular time window.

Alernatively, the underlying function `plotTD` provides the same functionality, but takes separate *time* and *depth* arguments, rather than a *TDR* object.

At any time, TDR objects can be coerced to a simple data frame, which can later be exported or manipulated any other way:

```
> sealX.df <- as.data.frame(sealX)
> head(sealX.df)
```

```
                 time depth light temperature speed
1 2002-01-05 11:32:00    NA    NA          NA    NA
2 2002-01-05 11:32:05    NA    NA          NA    NA
3 2002-01-05 11:32:10    NA    NA          NA    NA
4 2002-01-05 11:32:15    NA    NA          NA    NA
5 2002-01-05 11:32:20    NA    NA          NA    NA
6 2002-01-05 11:32:25    NA    NA          NA    NA
```

5 Zero-Offset Depth Correction and Summary of Wet/Dry Periods

One the first steps of dive analysis involves correcting depth for shifts in the pressure transducer, so that surface readings correspond to the value zero. Although some complex algorithms exist for detecting where these shifts occur in the record, the shifts remain difficult to detect and dives are often missed, which a visual examination of the data would have exposed. The trade off is that visually zero-adjusting depth is tedious, but the advantages of this approach far outweigh this cost, as much insight is gained by visually exploring the data. Not to mention the fact that obvious problems in the records are more effectively dealt with in this manner.

That personal rant aside, zero offset correction (ZOC) is done in `diveMove` using the function `zoc`. However, a more efficient method of doing this is by using the `calibrateDepth` function, which takes a *TDR* object (or inheriting from it) to perform three basic tasks. The first is to ZOC the data, using the `tcltk` package to be able to do it interactively:

```
> dcalib <- calibrateDepth(sealX)
```

This command brings up a plot with `tcltk` controls allowing to pan and zoom in or out of the data, as well as adjustment of the `depth` scale. Thus, an appropriate time window with a unique surface depth value can be displayed. It is important to make the display such that the `depth` scale is small enough to allow the resolution of the surface value with the mouse. Clicking on the ZOC button waits for two clicks:

1. the coordinates of the first click define the starting time for the window to be ZOC'ed, and the depth corresponding to the surface,

DRAFT

2. the second click defines the end time for the window (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, there is no need to go through all this procedure, and the value can be provided as the argument *offset* to `calibrateDepth`.

Once depth has been ZOC'ed, `calibrateDepth` will identify dry and wet periods in the record. Wet periods are those where a depth reading is available, dry periods are those without a depth reading. Records often have abherrant missing depth that should not be considered dry periods, as they are often of very short duration. Likewise, there may be periods of wet activity that are too short to be compared with other wet periods. This can be controlled by setting the arguments *dry.thr* and *wet.thr*.

Finally, `calibrateDepth` identifies all dives in the record, according to a minimum depth criteria given as its *divethres* argument. The result (value) of this function is an object of class *TDRcalibrate*, where all the information obtained during the tasks described above are stored. Again, an appropriate *show* method is available to display a short overview of such objects:

```
> dcalib

Depth calibration -- Class TDRcalibrate object
  Source file                    : dives.csv
  Containing TDR of class        : TDRspeed
  Number of dry phases           : 4
  Number of aquatic phases       : 3
  Number of dives detected       : 317
  Dry threshold used (s)         : 70
  Aquatic theshold used (s)      : 3610
  Dive threshold used (s)        : 4
  Speed calibration coefficients: a = 0 ; b = 1
```

6 Access to Elements from TDRcalibrate Objects

Extractor methods are also available to access the information stored in *TDRcalibrate* objects. These include: *getTDR*, *getGAct*, *getDAct*, *getDPhaseLab*, and *getSpeedCoefs*. These are all generic functions[2] that access the (depth) calibrated *TDR* object, details from wet/dry periods, dives, dive phases, and speed calibration coefficients (see Section 7), respectively. Below is a short explanation of these methods.

getTDR This method simply takes the *TDRcalibrate* object as its single argument and

[2]A few of them with more than one method

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 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) number, respectively (output omitted):

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

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

DRAFT

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

The latter method is useful for visually inspecting the assignment of points to particular dive phases. Before doing that though, this is a good time to introduce another generic function that allows the subsetting of the original *TDR* object to a single a dive or group of dives' data:

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

Time-Depth Recorder data -- Class TDRspeed object
  Source File          : dives.csv
  Sampling Interval (s): 5
  Number of Samples    : 2410
  Sampling Begins      : 2002-01-06 00:45:15
  Sampling Ends        : 2002-01-07 03:27:10
  Total Duration (d)   : 1.112
  Measured depth range : [ 0 , 88 ]
  Other variables      : light temperature speed
```

As can be seen, the function 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 *plotTDR* method[4]:

```
> plotTDR(subSealX, phaseCol = dphases)
```

7 Speed Calibration

Calibration of speed readings is done using the principles described in Blackwell et al. (1999) and Hindell et al. (1999). The function `calibrateSpeed` performs this operation, and allows the selection of the particular subset of the data that should be used for the calibration:

```
> vcalib <- calibrateSpeed(dcalib, z = 1)

> vcalib

Depth calibration -- Class TDRcalibrate object
  Source file                    : dives.csv
  Containing TDR of class        : TDRspeed
  Number of dry phases           : 4
```

[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`

DRAFT

```
Number of aquatic phases       : 3
Number of dives detected       : 317
Dry threshold used (s)         : 70
Aquatic theshold used (s)      : 3610
Dive threshold used (s)        : 4
Speed calibration coefficients: a = -0.44 ; b = 1.1
```

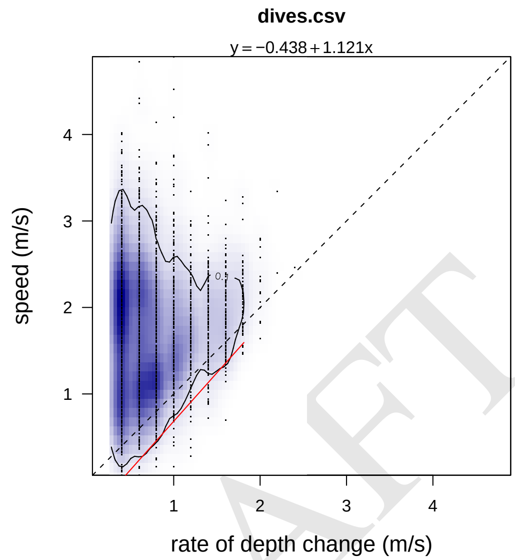

Figure 1. Example speed calibration line from a TDR record.

Using *plot=FALSE* it is possible to turn off the default side effect of producing a plot displaying the quantile regression fit (Figure 1).

Control is possible by the use of arguments *bad*, which controls minimum rates of depth change and speeds through which the calibration line should be drawn. To control for the resolution of the TDR, *z* can be used to include only changes in depth greater than a given value for the construction of the calibration line.

If the calibration coefficients from the implicit quantile regression are known a priori, then these can be supplied to the function via its *coefs* argument. In this case, no plots are created.

8 TDR dive and postdive statistics

Once data have been calibrated and the record broken up at "trip" and "dive" scales, obtaining dive statistics is a trivial call to function `diveStats`:

DRAFT

```
> dives <- diveStats(vcalib)
> head(dives, 3)
```

```
              begdesc             enddesc
1 2002-01-05 12:20:10 2002-01-05 12:20:10
2 2002-01-05 21:19:40 2002-01-05 21:20:10
3 2002-01-05 21:22:10 2002-01-05 21:23:05
               begasc desctim botttim asctim descdist
1 2002-01-05 12:20:25     2.5      15    2.5        3
2 2002-01-05 21:20:50    32.5      40   37.5       24
3 2002-01-05 21:23:50    57.5      45   72.5       61
  bottdist ascdist desc.tdist desc.mean.speed desc.angle
1        6       3         NA              NA         NA
2        9      25      63.62           2.121      22.16
3       10      67      98.08           1.783      38.46
  bott.tdist bott.mean.speed asc.tdist asc.mean.speed
1      42.87           2.858        NA             NA
2      87.59           2.190     55.67          1.591
3      69.92           1.554    108.13          1.545
  asc.angle divetim maxdep postdive.dur postdive.tdist
1        NA      20      6        32345       52784.67
2     26.69     110     29           35          35.78
3     38.29     175     67           75          89.21
  postdive.mean.speed
1               1.638
2               1.022
3               1.189
```

The function takes a single argument: an object of class *TDRcalibrate*, and returns a data frame with one row per dive in the record, with a suite of basic dive statistics in each column. Please consult `?diveStats` for an explanation of each of the variables estimated, although the names of the output data frame should be self explanatory. These variables are thus available for calculating any other derived values, by extracting them using the standard R subscripting facilities.

9 Miscellaneous functions

Other functions are included for handling location data, and these are `readLocs`, `austFilter`, and `distSpeed`. These are useful for reading, filtering, and summarizing travel information. For extensive animal movement analyses, refer to package `trip`.

10 Acknowledgements

Invaluable input and help during development of this package has been offered by my mentors John P.Y. Arnould, Christophe Guinet, and Edward H. Miller. I also thank the regular contributors to R-help for their help during development.

References

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

Hindell, M. A., McConnell, B. J., Fedak, M. A., Slip, D. J., Burton, H. R., Reijnders, P. J. H., and McMahon, C. R. (1999). Environmental and physiological determinants of successful foraging by naive southern elephant seal pups during their first trip to sea. *Can. J. Zool.*, 77:1807–1821.

DRAFT

diveMove

January 11, 2008

R topics documented:

DRAFT

`austFilter` *Filter satellite locations*

Description

Apply a three stage algorithm to eliminate erroneous locations, based on the procedure outlined in Austin et al. (2003).

Usage

```
austFilter(time, lon, lat, id=gl(1, 1, length(time)),
           speed.thr, dist.thr, window=5)
grpSpeedFilter(x, speed.thr, window=5)
rmsDistFilter(x, speed.thr, window=5, dist.thr)
```

Arguments

`time`	`POSIXct` object with dates and times for each point.
`lon`	Numeric vectors of longitudes, in decimal degrees.
`lat`	Numeric vector of latitudes, in decimal degrees.
`id`	A factor grouping points in different categories (e.g. individuals).
`speed.thr`	Speed threshold (m/s) above which filter tests should fail any given point.
`dist.thr`	Distance threshold (km) above which the last filter test should fail any given point.
`window`	Integer indicating the size of the moving window over which tests should be carried out.
`x`	3-column matrix with column 1: `POSIXct` vector; column 2: numeric longitude vector; column 3: numeric latitude vector.

Details

These functions implement the location filtering procedure outlined in Austin et al. (2003). `grpSpeedFilter` and `rmsDistFilter` can be used to perform only the first stage or the second and third stages of the algorithm on their own, respectively. Alternatively, the three filters can be run sequentially using `austFilter`.

The first stage of the filter is an iterative process which tests every point, except the first and last (w/2) - 1 (where w is the window size) points, for travel velocity relative to the preceeding/following (w/2) - 1 points. If all w - 1 speeds are greater than the specified threshold, the point is marked as failing the first stage. In this case, the next point is tested, removing the failing point from the set of test points.

The second stage runs McConnell et al. (1992) algorithm, which tests all the points that passed the first stage, in the same manner as above. The root mean square of all w - 1 speeds is calculated, and if it is greater than the specified threshold, the point is marked as failing the second stage.

The third stage is run simultaneously with the second stage, but if the mean distance of all *w* - 1 pairs of points is greater than the specified threshold, then the point is marked as failing the third stage.

The speed and distance threshold should be obtained separately (see `distSpeed`).

Value

`grpSpeedFilter` returns a logical vector indicating those lines that passed the test.

`rmsDistFilter` and `austFilter` return a matrix with 2 or 3 columns, respectively, of logical vectors with values TRUE for points that passed each stage. For the latter, positions that fail the first stage fail the other stages too. The second and third columns returned by `austFilter`, as well as those returned by `rmsDistFilter` are independent of one another; i.e. positions that fail stage 2 do not necessarily fail stage 3.

Warning

This function applies McConnell et al.'s filter as described in Austin et al. (2003), but other authors may have used the filter differently. Austin et al. (2003) have apparently applied the filter in a vectorized manner. It is not clear from the original paper whether the filter is applied iteratively or in a vectorized fashion, so it seems to be used inconsistently.

Author(s)

Sebastian P. Luque ⟨spluque@gmail.com⟩ and Andy Liaw.

References

McConnell BJ, Chambers C, Fedak MA. 1992. Foraging ecology of southern elephant seals in relation to bathymetry and productivity of the Southern Ocean. *Antarctic Science* 4:393-398.

Austin D, McMillan JI, Bowen D. 2003. A three-stage algorithm for filtering erroneous Argos satellite locations. *Marine Mammal Science* 19: 371-383.

See Also

`distSpeed`

Examples

```
locs <- readLocs(system.file(file.path("data", "sealLocs.csv"),
                             package="diveMove"), idCol=1, dateCol=2,
                 dtformat="%Y-%m-%d %H:%M:%S", classCol=3, lonCol=4,
                 latCol=5)
ringy <- subset(locs, id == "ringy" & !is.na(lon) & !is.na(lat))

## Austin et al.'s group filter alone
grp <- grpSpeedFilter(ringy[, 3:5], speed.thr=1.1)

## McConnell et al.'s filter (root mean square test), and distance test alone
rms <- rmsDistFilter(ringy[, 3:5], speed.thr=1.1, dist.thr=300)
```

DRAFT

```
## Show resulting tracks
n <- nrow(ringy)
plot.nofilter <- function(main) {
    plot(lat ~ lon, ringy, type="n", main=main)
    with(ringy, segments(lon[-n], lat[-n], lon[-1], lat[-1]))
}
layout(matrix(1:4, ncol=2, byrow=TRUE))
plot.nofilter(main="Unfiltered Track")
plot.nofilter(main="Group Filter")
n1 <- length(which(grp))
with(ringy[grp, ], segments(lon[-n1], lat[-n1], lon[-1], lat[-1],
                            col="blue"))
plot.nofilter(main="Root Mean Square Filter")
n2 <- length(which(rms[, 1]))
with(ringy[rms[, 1], ], segments(lon[-n2], lat[-n2], lon[-1], lat[-1],
                                 col="red"))
plot.nofilter(main="Distance Filter")
n3 <- length(which(rms[, 2]))
with(ringy[rms[, 2], ], segments(lon[-n3], lat[-n3], lon[-1], lat[-1],
                                 col="green"))

## All three tests in sequence (Austin et al. procedure)
austin <- with(ringy, austFilter(time, lon, lat, speed.thr=1.1,
                                 dist.thr=300))
layout(matrix(1:4, ncol=2, byrow=TRUE))
plot.nofilter(main="Unfiltered Track")
plot.nofilter(main="Stage 1")
n1 <- length(which(austin[, 1]))
with(ringy[austin[, 1], ], segments(lon[-n1], lat[-n1], lon[-1], lat[-1],
                                    col="blue"))
plot.nofilter(main="Stage 2")
n2 <- length(which(austin[, 2]))
with(ringy[austin[, 2], ], segments(lon[-n2], lat[-n2], lon[-1], lat[-1],
                                    col="red"))
plot.nofilter(main="Stage 3")
n3 <- length(which(austin[, 3]))
with(ringy[austin[, 3], ], segments(lon[-n3], lat[-n3], lon[-1], lat[-1],
                                    col="green"))
```

bout-methods — *Methods for Plotting and Extracting the Bout Ending Criterion*

Description

Plot results from fitted mixture of 2-process Poisson models, and calculate the bout ending criterion.

Usage

```
## S4 method for signature 'nls':
```

```
plotBouts(fit, ...)
## S4 method for signature 'mle':
plotBouts(fit, x, ...)
## S4 method for signature 'nls':
bec2(fit)
## S4 method for signature 'mle':
bec2(fit)
```

Arguments

`fit`	`nls` or `mle` object.
`x`	Numeric object with variable modelled.
`...`	Arguments passed to the underlying `plotBouts2.nls` and `plotBouts2.mle`.

General Methods

plotBouts `signature(fit="nls")`: Plot fitted 2-process model of log frequency vs the interval mid points, including observed data.

plotBouts `signature(x="mle")`: As the `nls` method, but models fitted through maximum likelihood method. This plots the fitted model and a density plot of observed data.

bec2 `signature(fit="nls")`: Extract the estimated bout ending criterion from a fitted 2-process model.

bec2 `signature(fit="mle")`: As the nls method, but extracts the value from a maximum likelihood model.

Author(s)

Sebastian P. Luque ⟨spluque@gmail.com⟩

References

Langton, S.; Collett, D. and Sibly, R. (1995) Splitting behaviour into bouts; a maximum likelihood approach. Behaviour **132**, 9-10.

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

Mori, Y.; Yoda, K. and Sato, K. (2001) Defining dive bouts using a sequential differences analysis. Behaviour **138**, 1451-1466.

Sibly, R.; Nott, H. and Fletcher, D. (1990) Splitting behaviour into bouts. Animal Behaviour **39**, 63-69.

See Also

`bouts.mle`, `bouts2.nls` for examples.

DRAFT

`bout-misc` *Fit a Broken Stick Model on Log Frequency Data for identification of bouts of behaviour*

Description

Application of methods described by Sibly et al. (1990) and Mori et al. (2001) for the identification of bouts of behaviour, based on sequential differences of a variable.

Usage

```
boutfreqs(x, bw, method=c("standard", "seq.diff"), plot=TRUE)
boutinit(lnfreq, x.break, plot=TRUE)
labelBouts(x, bec, bec.method=c("standard", "seq.diff"))
logit(p)
unLogit(logit)
```

Arguments

`x`	numeric vector on which bouts will be identified based on "method". For `labelBouts` it can also be a matrix with different variables for which bouts should be identified.
`bw`	bin width for the histogram.
`method, bec.method`	method used for calculating the frequencies: "standard" simply uses x, while "seq.diff" uses the sequential differences method.
`plot`	logical, whether to plot results or not.
`lnfreq`	data frame with components *lnfreq* (log frequencies) and corresponding *x* (mid points of histogram bins).
`x.break`	x value defining the break point for broken stick model, such that x < xlim is 1st process, and x >= xlim is 2nd one.
`bec`	numeric vector or matrix with values for the bout ending criterion which should be compared against the values in x for identifying the bouts.
`p`	vector of proportions (0-1) to transform to the logit scale.
`logit`	Logit value to transform back to original scale.

Details

This follows the procedure described in Mori et al. (2001), which is based on Sibly et al. 1990. Currently, only a two process model is supported.

`boutfreqs` creates a histogram with the log transformed frequencies of *x* with a chosen bin width and upper limit. Bins following empty ones have their frequencies averaged over the number of previous empty bins plus one.

DRAFT

`boutinit` fits a "broken stick" model to the log frequencies modelled as a function of *x* (well, the midpoints of the binned data), using a chosen value to separate the two processes.

`labelBouts` labels each element (or row, if a matrix) of *x* with a sequential number, identifying which bout the reading belongs to.

`logit` and `unLogit` are useful for reparameterizing the negative maximum likelihood function, if using Langton et al. (1995).

Value

`boutfreqs` returns a data frame with components *lnfreq* containing the log frequencies and *x*, containing the corresponding mid points of the histogram. Empty bins are excluded. A plot is produced as a side effect if argument plot is TRUE. See the Details section.

`boutinit` returns a list with components a1, lambda1, a2, and lambda2, which are starting values derived from broken stick model. A plot is produced as a side effect if argument plot is TRUE.

`labelBouts` returns a numeric vector sequentially labelling each row or element of *x*, which associates it with a particular bout.

`unLogit` and `logit` return a numeric vector with the (un)transformed arguments.

Author(s)

Sebastian P. Luque ⟨spluque@gmail.com⟩

References

Langton, S.; Collett, D. and Sibly, R. (1995) Splitting behaviour into bouts; a maximum likelihood approach. Behaviour **132**, 9-10.

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

Mori, Y.; Yoda, K. and Sato, K. (2001) Defining dive bouts using a sequential differences analysis. Behaviour, 2001 **138**, 1451-1466.

Sibly, R.; Nott, H. and Fletcher, D. (1990) Splitting behaviour into bouts. Animal Behaviour **39**, 63-69.

See Also

`bouts2.nls`, `bouts.mle`.

Examples

```
data(divesSummary)
postdives <- divesSummary$postdive.dur[divesSummary$trip.no == 2]
## Remove isolated dives
postdives <- postdives[postdives < 2000]
lnfreq <- boutfreqs(postdives, bw=0.1, method="seq.diff", plot=FALSE)
boutinit(lnfreq, 50)
```

DRAFT

`bouts2MLE` *Maximum Likelihood Model of mixture of 2 Poisson Processes*

Description

Functions to model a mixture of 2 random Poisson processes to identify bouts of behaviour. This follows Langton et al. (1995).

Usage

```
bouts2.mleFUN(x, p, lambda1, lambda2)
bouts2.ll(x)
bouts2.LL(x)
bouts.mle(ll.fun, start, x, ...)
bouts2.mleBEC(fit)
plotBouts2.mle(fit, x, xlab="x", ylab="Log Frequency", bec.lty=2, ...)
plotBouts2.cdf(fit, x, draw.bec=FALSE, bec.lty=2, ...)
```

Arguments

`x`	Numeric vector with values to model.
`p, lambda1, lambda2`	Parameters of the mixture of Poisson processes.
`ll.fun`	function returning the negative of the maximum likelihood function that should be maximized. This should be a valid `minuslogl` argument to `mle`.
`start, ...`	Arguments passed to `mle`. For `plotBouts2.cdf`, arguments passed to `plot.ecdf`. For `plotBouts2.mle`, arguments passed to `curve`.
`fit`	`mle` object.
`xlab, ylab`	Titles for the x and y axes.
`bec.lty`	Line type specification for drawing the BEC reference line.
`draw.bec`	Logical; do we draw the BEC?

Details

For now only a mixture of 2 Poisson processes is supported. Even in this relatively simple case, it is very important to provide good starting values for the parameters.

One useful strategy to get good starting parameter values is to proceed in 4 steps. First, fit a broken stick model to the log frequencies of binned data (see `boutinit`), to obtain estimates of 4 parameters corresponding to a 2-process model (Sibly et al. 1990). Second, calculate parameter *p* from the 2 alpha parameters obtained from the broken stick model, to get 3 tentative initial values for the 2-process model from Langton et al. (1995). Third, obtain MLE estimates for these 3 parameters, but using a reparameterized version of the -log L2 function. Lastly, obtain the final MLE estimates for the 3 parameters by using the estimates from step 3, un-transformed back to their original scales, maximizing the original parameterization of the -log L2 function.

DRAFT

boutinit can be used to perform step 1. Calculation of the mixing parameter *p* in step 2 is trivial from these estimates. Function bouts2.LL is a reparameterized version of the -log L2 function given by Langton et al. (1995), so can be used for step 3. This uses a logit (see logit) transformation of the mixing parameter *p*, and log transformations for both density parameters *lambda1* and *lambda2*. Function bouts2.ll is the -log L2 function corresponding to the un-transformed model, hence can be used for step 4.

bouts.mle is the function performing the main job of maximizing the -log L2 functions, and is essentially a wrapper around mle. It only takes the -log L2 function, a list of starting values, and the variable to be modelled, all of which are passed to mle for optimization. Additionally, any other arguments are also passed to mle, hence great control is provided for fitting any of the -log L2 functions.

In practice, step 3 does not pose major problems using the reparameterized -log L2 function, but it might be useful to use method "L-BFGS-B" with appropriate lower and upper bounds. Step 4 can be a bit more problematic, because the parameters are usually on very different scales. Therefore, it is almost always the rule to use method "L-BFGS-B", again with bounding the parameter search, as well as passing a `control` list with proper `parscale` for controlling the optimization. See `Note` below for useful constraints which can be tried.

Value

`bouts.mle` returns an object of class mle.

`bouts2.mleBEC` and `bouts2.mleFUN` return a numeric vector.

`bouts2.LL` and `bouts2.ll` return a function.

`plotBouts2.mle` and `plotBouts2.cdf` return nothing, but produce a plot as side effect.

Note

In the case of a mixture of 2 Poisson processes, useful values for lower bounds for the `bouts.LL` reparameterization are `c(-2, -5, -10)`. For `bouts2.ll`, useful lower bounds are `rep(1e-08, 3)`. A useful parscale argument for the latter is `c(1, 0.1, 0.01)`. However, I have only tested this for cases of diving behaviour in pinnipeds, and may with other cases.

Author(s)

Sebastian P. Luque ⟨spluque@gmail.com⟩

References

Langton, S.; Collett, D. and Sibly, R. (1995) Splitting behaviour into bouts; a maximum likelihood approach. Behaviour **132**, 9-10.

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

Sibly, R.; Nott, H. and Fletcher, D. (1990) Splitting behaviour into bouts. Animal Behaviour **39**, 63-69.

See Also

mle, optim, logit, unLogit for transforming and fitting a reparameterized model.

Examples

```
data(divesSummary)
postdives <- divesSummary$postdive.dur[divesSummary$trip.no == 2]
postdives.diff <- abs(diff(postdives))

## Remove isolated dives
postdives.diff <- postdives.diff[postdives.diff < 2000]
lnfreq <- boutfreqs(postdives.diff, bw=0.1, plot=FALSE)
startval <- boutinit(lnfreq, 50)
p <- startval$a1 / (startval$a1 + startval$a2)

## Fit the reparameterized (transformed parameters) model
init.parms <- list(p=logit(p), lambda1=log(startval$lambda1),
                   lambda2=log(startval$lambda2))
bout.fit1 <- bouts.mle(bouts2.LL, start=init.parms, x=postdives.diff,
                       method="L-BFGS-B", lower=c(-2, -5, -10))
coefs <- as.vector(coef(bout.fit1))

## Un-transform and fit the original parameterization
init.parms <- list(p=unLogit(coefs[1]), lambda1=exp(coefs[2]),
                   lambda2=exp(coefs[3]))
bout.fit2 <- bouts.mle(bouts2.ll, x=postdives.diff, start=init.parms,
                       method="L-BFGS-B", lower=rep(1e-08, 3),
                       control=list(parscale=c(1, 0.1, 0.01)))
plotBouts(bout.fit2, postdives.diff)

## Plot cumulative frequency distribution
plotBouts2.cdf(bout.fit2, postdives.diff)

## Estimated BEC
bec2(bout.fit2)
```

---

bouts2NLS *Fit mixture of 2 Poisson Processes to Log Frequency data*

Description

Functions to model a mixture of 2 random Poisson processes to histogram-like data of log frequency vs interval mid points. This follows Sibly et al. (1990) method.

Usage

```
bouts2.nlsFUN(x, a1, lambda1, a2, lambda2)
bouts2.nls(lnfreq, start, maxiter)
bouts2.nlsBEC(fit)
plotBouts2.nls(fit, lnfreq, bec.lty, ...)
```

DRAFT

Arguments

`x`	Numeric vector with values to model.
`a1, lambda1, a2, lambda2`	Parameters from the mixture of Poisson processes.
`lnfreq`	data frame with named components *lnfreq* (log frequencies) and corresponding *x* (mid points of histogram bins).
`start, maxiter`	Arguments passed to `nls`.
`fit`	nls object.
`bec.lty`	Line type specification for drawing the BEC reference line.
`...`	Arguments passed to `plot.default`.

Details

`bouts2.nlsFUN` is the function object defining the nonlinear least-squares relationship in the model. It is not meant to be used directly, but is used internally by `bouts2.nls`.

`bouts2.nls` fits the nonlinear least-squares model itself.

`bouts2.nlsBEC` calculates the BEC from a list object, as the one that is returned by `nls`, representing a fit of the model. `plotBouts2.nls` plots such an object.

Value

`bouts2.nlsFUN` returns a numeric vector evaluating the mixture of 2 Poisson process.

`bouts2.nls` returns an nls object resulting from fitting this model to data.

`bouts2.nlsBEC` returns a number corresponding to the bout ending criterion derived from the model.

`plotBouts2.nls` plots the fitted model with the corresponding data.

Author(s)

Sebastian P. Luque ⟨spluque@gmail.com⟩

References

Sibly, R.; Nott, H. and Fletcher, D. (1990) Splitting behaviour into bouts Animal Behaviour **39**, 63-69.

See Also

`bouts.mle` for a better approach.

DRAFT

Examples

```
data(divesSummary)
## Postdive durations
postdives <- divesSummary$postdive.dur[divesSummary$trip.no == 2]
postdives.diff <- abs(diff(postdives))
## Remove isolated dives
postdives.diff <- postdives.diff[postdives.diff < 2000]

## Construct histogram
lnfreq <- boutfreqs(postdives.diff, bw=0.1, plot=FALSE)
startval <- boutinit(lnfreq, 50)

## Fit the 2 process model
bout.fit1 <- bouts2.nls(lnfreq, start=startval, maxiter=500)
summary(bout.fit1)
plotBouts(bout.fit1)

## Estimated BEC
bec2(bout.fit1)
```

`calibrateDepth` *Calibrate Depth and Generate a "TDRcalibrate" object*

Description

Detect periods of major activities in a TDR record, calibrate depth readings, and generate a "TDRcalibrate" object essential for subsequent summaries of diving behaviour.

Usage

```
calibrateDepth(x, dry.thr=70, wet.thr=3610, dive.thr=4, offset,
               descent.crit.q=0.1, ascent.crit.q=0.1, wiggle.tol=0.8)
```

Arguments

`x`	An object of class `TDR` for `calibrateDepth` or an object of class `TDRcalibrate` for `calibrateSpeed`.
`dry.thr`	Dry error threshold in seconds. Dry phases shorter than this threshold will be considered as wet.
`wet.thr`	Wet threshold in seconds. At-sea phases shorter than this threshold will be considered as trivial wet.
`dive.thr`	Threshold depth below which an underwater phase should be considered a dive.
`offset`	Argument to `zoc`. If not provided, the offset is obtained using an interactive plot of the data.

DRAFT

`descent.crit.q`
: Critical quantile of rates of descent below which descent is deemed to have ended.

`ascent.crit.q`
: Critical quantile of rates of ascent above which ascent is deemed to have started.

`wiggle.tol`
: Proportion of maximum depth above which wiggles should not be allowed to define the end of descent. It's also the proportion of maximum depth below which wiggles should be considered part of bottom phase.

Details

This function is really a wrapper around `.detPhase` and `.detDive`, which perform the work on simplified objects. It performs zero-offset correction of depth, wet/dry phase detection, and detection of dives, as well as proper labelling of the latter.

The procedure starts by first creating a factor with value 'L' (dry) for rows with NAs for `depth` and value 'W' (wet) otherwise. It subsequently calculates the duration of each of these phases of activity. If the duration of an dry phase ('L') is less than `dry.thr`, then the values in the factor for that phase are changed to 'W' (wet). The duration of phases is then recalculated, and if the duration of a phase of wet activity is less than `wet.thr`, then the corresponding value for the factor is changed to 'Z' (trivial wet). The durations of all phases are recalculated a third time to provide final phase durations.

The next step is to detect dives whenever the zero-offset corrected depth in an underwater phase is below the supplied dive threshold. A new factor with finer levels of activity is thus generated, including 'U' (underwater), and 'D' (diving) in addition to the ones described above.

Once dives have been detected and assigned to a period of wet activity, phases within dives are detected using the descent, ascent and wiggle criteria. This procedure generates a factor with levels "D", "DB", "B", "BA", "A", "DA", and "X", breaking the input into descent, descent/bottom, bottom, bottom/ascent, ascent, and non-dive, respectively.

Value

An object of class `TDRcalibrate`.

Author(s)

Sebastian P. Luque ⟨spluque@gmail.com⟩

See Also

`TDRcalibrate`, `zoc`

Examples

```
data(divesTDR)
divesTDR

## Consider a 3 m offset, and a dive threshold of 3 m
dcalib <- calibrateDepth(divesTDR, dive.thr=3, offset=3)
```

DRAFT

```
if (dev.interactive(orNone=TRUE)) {
    plotTDR(dcalib, labels="dive.phase", surface=TRUE)
}
```

`calibrateSpeed` *Calibrate and build a "TDRcalibrate" object*

Description

These functions create a "TDRcalibrate" object which is necessary to obtain dive summary statistics.

Usage

```
calibrateSpeed(x, tau=0.1, contour.level=0.1, z=0, bad=c(0, 0),
               main=slot(getTDR(x), "file"), coefs, plot=TRUE,
               postscript=FALSE, ...)
```

Arguments

`x`	An object of class `TDR` for `calibrateDepth` or an object of class `TDRcalibrate` for `calibrateSpeed`.
`tau`	Quantile on which to regress speed on rate of depth change; passed to `rq`.
`contour.level`	The mesh obtained from the bivariate kernel density estimation corresponding to this contour will be used for the quantile regression to define the calibration line.
`z`	Only changes in depth larger than this value will be used for calibration.
`bad`	Length 2 numeric vector indicating that only rates of depth change and speed greater than the given value should be used for calibration, respectively.
`coefs`	Known speed calibration coefficients from quantile regression as a vector of length 2 (intercept, slope). If provided, these coefficients are used for calibrating speed, ignoring all other arguments, except `x`.
`main, ...`	Arguments passed to `rqPlot`.
`plot`	Logical indicating whether to plot the results.
`postscript`	Logical indicating whether to produce postscript file output.

Details

This calibrates speed readings following the procedure outlined in Blackwell et al. (1999).

Value

An object of class `TDRcalibrate`.

DRAFT

Author(s)

Sebastian P. Luque ⟨spluque@gmail.com⟩

References

Blackwell S, Haverl C, Le Boeuf B, Costa D (1999). A method for calibrating swim-speed recorders. Marine Mammal Science 15(3):894-905.

See Also

`TDRcalibrate`

Examples

```
data(divesTDRcalibrate)
divesTDRcalibrate

## Calibrate speed using only changes in depth > 2 m
vcalib <- calibrateSpeed(divesTDRcalibrate, z=2)
vcalib
```

`detDive-internal` *Detect dives from depth readings*

Description

Identify dives in TDR records based on a dive threshold.

Usage

```
.detDive(zdepth, act, dive.thr=4, ...)
```

Arguments

`zdepth`	Vector of zero-offset corrected depths.
`act`	Factor as long as `depth` coding activity, with levels specified as in `.detPhase`.
`dive.thr`	Threshold depth below which an underwater phase should be considered a dive.
`...`	The sampling interval in seconds.

Value

A data frame with the following elements for `.detDive`

`dive.id`	Numeric vector numbering each dive in the record.
`dive.activity`	Factor with levels 'L', 'W', 'U', 'D', and 'Z', see `.detPhase`. All levels may be represented.
`postdive.id`	Numeric vector numbering each postdive interval with the same value as the preceding dive.

Author(s)

Sebastian P. Luque ⟨spluque@gmail.com⟩

See Also

`.detPhase`, `zoc`

`detPhase-internal` *Detect phases of activity from depth readings*

Description

Functions to identify sections of a TDR record displaying one of three possible activities: dry, wet, and trivial wet.

Usage

```
.detPhase(time, depth, dry.thr, wet.thr, ...)
```

Arguments

`time`	`POSIXct` object with date and time for all depths.
`depth`	Numeric vector with depth readings.
`dry.thr, wet.thr`	Passed from `calibrateDepth`.
`...`	Passed from `calibrateDepth`; sampling interval in seconds.

Details

`.detPhase` first creates a factor with value 'L' (dry) for rows with NAs for `depth` and value 'W' (wet) otherwise. It subsequently calculates the duration of each of these phases of activity. If the duration of an dry phase ('L') is less than `dry.thr`, then the values in the factor for that phase are changed to 'W' (wet). The duration of phases is then recalculated, and if the duration of a phase of wet activity is less than `wet.thr`, then the corresponding value for the factor is changed to 'Z' (trivial wet). The durations of all phases are recalculated a third time to provide final phase durations.

Value

A list with components:

`phase.id`	Numeric vector identifying each activity phase, starting from 1 for every input record.
`activity`	Factor with levels 'L' indicating dry, 'W' indicating wet, 'U' for underwater (above dive criterion), 'D' for diving, 'Z' for trivial wet animal activities. Only 'L', 'W', and 'Z' are actually represented.

DRAFT

`begin`	A `POSIXct` object as long as the number of unique activity phases identified, indicating the start times for each activity phase.
`end`	A `POSIXct` object as long as the number of unique activity phases identified, indicating the end times for each activity phase.

Author(s)

Sebastian P. Luque ⟨spluque@gmail.com⟩ and Andy Liaw.

See Also

`.detDive`

`distSpeed` *Calculate distance and speed between locations*

Description

Calculate distance, time difference, and speed between pairs of points defined by latitude and longitude, given the time at which all points were measured.

Usage

```
distSpeed(pt1, pt2)
```

Arguments

`pt1`	A matrix or data frame with three columns; the first a `POSIXct` object with dates and times for all points, the second and third numeric vectors of longitude and latitude for all points, respectively, in decimal degrees.
`pt2`	A matrix with the same size and structure as `pt1`.

Value

A matrix with three columns: distance (km), time difference (s), and speed (m/s).

Author(s)

Sebastian P. Luque ⟨spluque@gmail.com⟩

DRAFT

Examples

```
locs <- readLocs(system.file(file.path("data", "sealLocs.csv"),
                             package="diveMove"), idCol=1, dateCol=2,
                  dtformat="%Y-%m-%d %H:%M:%S", classCol=3, lonCol=4,
                  latCol=5)

## Travel summary between successive standard locations
locs.std <- subset(locs, subset=class == "0" | class == "1" |
                   class == "2" | class == "3" &
                   !is.na(lon) & !is.na(lat))
locs.std.tr <- by(locs.std, locs.std$id, function(x) {
    distSpeed(x[-nrow(x), 3:5], x[-1, 3:5])
})
lapply(locs.std.tr, head)

## Particular quantiles from travel summaries
lapply(locs.std.tr, function(x) {
    quantile(x[, 3], seq(0.90, 0.99, 0.01), na.rm=TRUE) # speed
})
lapply(locs.std.tr, function(x) {
    quantile(x[, 1], seq(0.90, 0.99, 0.01), na.rm=TRUE) # distance
})

## Travel summary between arbitrary two sets of points
distSpeed(locs[c(1, 5, 10), 3:5], locs[c(25, 30, 35), 3:5])
```

`diveMove-internal` *Internal diveMove Functions*

Description

Functions used for very particular tasks within larger functions in diveMove

Usage

```
.diveIndices(diveID, diveNo)
.getInterval(time)
.speedStats(x, vdist)
.night(time, sunrise.time, sunset.time)
.rleActivity(time, act, interval)
.speedCol(x)
```

Arguments

`diveID`	Numeric vector of all dive and non dive IDs.
`diveNo`	Numeric vector of unique dive indices to extract from `diveID`.

`time`	`POSIXct` object representing time.
`x`	For `.speedStats`, a matrix with a dive's section data (time, speed). A single dive's data, a 3-col matrix with time, depth, and speed. For .speedCol, a data frame where names are searched for strings matching `.speedNames` (see Details).
`vdist`	Vertical distance travelled during this time. If `vdist` is missing, then it's all horizontal movements (no angles).
`sunrise.time, sunset.time`	Passed from `plotTD`.
`act`	A numeric vector indicating the activity for every element of `time`.
`interval`	Sampling interval in seconds.

Details

These functions are not meant to be called directly by the user, as he/she could not care less (right?). This may change in the future.

`.speedNames` is a character vector with possible names for a speed vector.

`.rleActivity` takes a factor indicating different activity phases, their associated time, and the sampling interval to return a factor uniquely identifying each phase of activity, i.e. labelling them. In addition, it returns the duration of each phase, and their beginning and end times.

Value

`.diveIndices` returns a numeric vector with the indices of dives (and their beginning/end indices) in `diveID`.

`.getInterval` returns a scalar, the mode of intervals between time readings.

`.speedStats` returns a 3-column matrix with total distance, mean speed, and angle for a section of a dive.

`.night` returns a list with sunrise and sunset times for dates in `time`.

`.speedCol` returns column number where speed is located in x.

`.rleActivity` returns a list with components:

`time.br`	A factor dividing `act` into different periods of activity.
`time.peract`	The duration of each period of activity.
`beg.time, end.time`	POSIXct objects indicating the beginning and ending times of each period of activity.

Author(s)

Sebastian P. Luque ⟨spluque@gmail.com⟩

DRAFT

`diveMove-package` *Time depth recorder analysis*

Description

This package is a collection of functions for visualizing, and analyzing depth and speed data from time-depth recorders *TDR*s. These can be used to zero-offset correct depth, calibrate speed, and divide the record into different phases, or time budget. Functions are provided for calculating summary dive statistics for the whole record, or at smaller scales within dives.

Author(s)

Sebastian P. Luque ⟨spluque@gmail.com⟩

See Also

A vignette with a guide to this package is available by doing 'vignette("diveMove")'. `TDR-class`, `calibrateDepth`, `calibrateSpeed`, `timeBudget`, `stampDive`.

Examples

```
## read in data and create a TDR object
(sealX <- readTDR(system.file(file.path("data", "dives.csv"),
                              package="diveMove"), speed=TRUE))

if (dev.interactive(orNone=TRUE)) plotTDR(sealX)  # interactively pan and zoom

## detect periods of activity, and calibrate depth, creating
## a 'TDRcalibrate' object
if (dev.interactive(orNone=TRUE)) dcalib <- calibrateDepth(sealX)
(dcalib <- calibrateDepth(sealX, offset=3)) # zero-offset correct at 3 m

if (dev.interactive(orNone=TRUE)) {
    ## plot all readings and label them with the phase of the record
    ## they belong to, excluding surface readings
    plotTDR(dcalib, labels="phase.id", surface=FALSE)
    ## plot the first 300 dives, showing dive phases and surface readings
    plotTDR(dcalib, diveNo=seq(300), labels="dive.phase", surface=TRUE)
}

## calibrate speed (using changes in depth > 1 m and default remaining arguments)
(vcalib <- calibrateSpeed(dcalib, z=1))

## Obtain dive statistics for all dives detected
dives <- diveStats(vcalib)
head(dives)

## Attendance table
att <- timeBudget(vcalib, FALSE) # taking trivial aquatic activities into account
att <- timeBudget(vcalib, TRUE)  # ignoring them
```

```
## Add trip stamps to each dive
stamps <- stampDive(vcalib)
sumtab <- data.frame(stamps, dives)
head(sumtab)
```

---

dives *Sample TDR data from a fur seal*

Description

This data set is meant to show a typical organization of a TDR *.csv file, suitable as input for `readTDR`, or to construct a `TDR` object. `divesTDR` and `divesTDRcalibrate` are example `TDR` and `TDRcalibrate` objects.

Format

A comma separated value (csv) file with 69560 TDR readings with the following columns:

date Date

time Time

depth Depth in m

light Light level

temperature Temperature in C

speed Speed in m/s

The data are also provided as a TDR object (*.RData format) for convenience.

Details

The data are a subset of an entire TDR record, so they are not meant to make valid inferences from this particular individual/deployment.

`divesTDR` is a `TDR` object representation of the data in `dives`.

`divesTDRcalibrate` is a `TDRcalibrate` object representing the data in dives, calibrated at default criteria (see `calibrateDepth`), and 3 m offset.

`divesSummary` is a data frame containing a summary of all dives in this dataset (see `diveStats` and `stampDive` for the information contained in this object.

Source

Sebastian P. Luque, Christophe Guinet, John P.Y. Arnould

See Also

`readTDR`, `diveStats`.

DRAFT

diveStats *Per-dive statistics*

Description

Calculate dive statistics in TDR records.

Usage

```
diveStats(x)
oneDiveStats(x, interval, speed=FALSE)
stampDive(x, ignoreZ=TRUE)
```

Arguments

`x`	A `TDRcalibrate-class` object for `diveStats` and `stampDive`, and a data frame containing a single dive's data (a factor identifying the dive phases, a POSIXct object with the time for each reading, a numeric depth vector, and a numeric speed vector) for `oneDiveStats`.
`interval`	Sampling interval for interpreting `x`.
`speed`	Logical; should speed statistics be calculated?
`ignoreZ`	Logical indicating whether trips should be numbered considering all aquatic activities ("W" and "Z") or ignoring "Z" activities.

Details

`diveStats` calculates various dive statistics based on time and depth for an entire TDR record. `oneDiveStats` obtains these statistics from a single dive, and `stampDive` stamps each dive with associated trip information.

Value

A `data.frame` with one row per dive detected (durations are in s, and linear variables in m):

`begdesc`	A `POSIXct` object, specifying the start time of each dive.
`enddesc`	A `POSIXct` object, as `begdesc` indicating descent's end time.
`begasc`	A `POSIXct` object, as `begdesc` indicating the time ascent began.
`desctim`	Descent duration of each dive.
`botttim`	Bottom duration of each dive.
`asctim`	Ascent duration of each dive.
`descdist`	Numeric vector with descent depth.
`bottdist`	Numeric vector with the sum of absolute depth differences while at the bottom of each dive; measure of amount of "wiggling" while at bottom.
`ascdist`	Numeric vector with ascent depth.

DRAFT

`desc.tdist` Numeric vector with descent total distance, estimated from speed.

`desc.mean.speed` Numeric vector with descent mean speed.

`desc.angle` Numeric vector with descent angle.

`bott.tdist` Numeric vector with bottom total distance, estimated from speed.

`bott.mean.speed` Numeric vector with bottom mean speed.

`asc.tdist` Numeric vector with ascent total distance, estimated from speed.

`asc.mean.speed` Numeric vector with ascent mean speed.

`asc.angle` Numeric vector with ascent angle.

`divetim` Dive duration.

`maxdep` Numeric vector with maximum depth.

`postdive.dur` Postdive duration.

`postdive.tdist` Numeric vector with postdive total distance, estimated from speed.

`postdive.mean.speed` Numeric vector with postdive mean speed.

The number of columns depends on the value of speed.

`stampDive` returns a data.frame with trip number, trip type, and start and end times for each dive.

Author(s)

Sebastian P. Luque ⟨spluque@gmail.com⟩

See Also

`.detPhase`, `zoc`, `TDRcalibrate-class`

Examples

```
data(divesTDRcalibrate)
divesTDRcalibrate

tdrX <- diveStats(divesTDRcalibrate)
stamps <- stampDive(divesTDRcalibrate, ignoreZ=TRUE)
tdrX.tab <- data.frame(stamps, tdrX)
summary(tdrX.tab)
```

DRAFT

extractDive-methods

Extract Dives from "TDR" or "TDRcalibrate" Objects

Description

Extract data corresponding to a particular dive(s), referred to by number.

Usage

```
## S4 method for signature 'TDR, numeric, numeric':
extractDive(obj, diveNo, id)
## S4 method for signature 'TDRcalibrate, numeric,
##   missing':
extractDive(obj, diveNo)
```

Arguments

`obj`	"TDR" object.
`diveNo`	Numeric vector or scalar with dive numbers to extract.
`id`	Numeric vector of dive numbers from where `diveNo` should be chosen.

Value

An object of class `TDR` or `TDRspeed`.

Author(s)

Sebastian P. Luque ⟨spluque@gmail.com⟩

Examples

```
data(divesTDR)
divesTDR
data(divesTDRcalibrate)
divesTDRcalibrate

diveX <- extractDive(divesTDR, 9, getDAct(divesTDRcalibrate, "dive.id"))
plotTDR(diveX, interact=FALSE)

diveX <- extractDive(divesTDRcalibrate, 5:10)
plotTDR(diveX, interact=FALSE)
```

DRAFT

labDive-internal *Internal Functions used for Detection of Dives*

Description

These functions provide information for particular dives,

Usage

```
.cutDive(x, descent.crit.q, ascent.crit.q, wiggle.tol)
.labDive(act, string, interval)
.labDivePhase(x, diveID, descent.crit.q, ascent.crit.q, wiggle.tol)
```

Arguments

`x`	For `.labDivePhase`, a class 'TDR' object. For `.cutDive`, a 3-col matrix with subscript in original TDR object, non NA depths, and numeric vector representing POSIXct times.
`descent.crit.q, ascent.crit.q, wiggle.tol`	Passed from `calibrateDepth`.
`act`	Factor with values to label.
`string`	A character belonging to a level of `act` to search for and label sequentially.
`interval`	The sampling interval in seconds.
`diveID`	Numeric vector indexing each dive (non-dives should be 0)

Details

These functions are for internal use and are not meant to be called by the user.

Value

`.labDive` returns a matrix with as many rows as its first two arguments with two columns: dive.id, and postdive.id, each one sequentially numbering each dive and postdive period.

`.labDivePhase` returns a factor with levels "D", "DB", "B", "BA", "A", "DA", and "X", breaking the input into descent, descent/bottom, bottom, bottom/ascent, ascent, and non-dive, respectively. If `x` contains no dives, only level "X" is present for all readings.

.cutDive generates a character vector that breaks a dive into descent, descent/bottom, bottom, bottom/ascent, ascent, and/or descent/ascent given a proportion of maximum depth for bottom time. It return a character matrix with orig ID and corresponding label.

Author(s)

Sebastian P. Luque ⟨spluque@gmail.com⟩

DRAFT

`plotTDR-methods` *Methods for plotting objects of class "TDR", "TDRspeed", and "TDRcalibrate"*

Description

Main plotting method for objects of these classes.

Usage

```
## S4 method for signature 'TDR':
plotTDR(x, ...)
## S4 method for signature 'TDRspeed':
plotTDR(x, concurVars, concurVarTitles, ...)
## S4 method for signature 'TDRcalibrate':
plotTDR(x, diveNo=seq(max(getDAct(x, "dive.id"))),
        labels="phase.id", concurVars, surface=FALSE, ...)
```

Arguments

`x` "`TDR`", "`TDRspeed`", or "`TDRcalibrate`"object.

`concurVars, concurVarTitles, ...`
Arguments passed to `plotTD`. For the `TDRspeed` method, *concurVars* is a matrix with variables to plot, in addition to speed, if any. *concurVarTitles* in this case is a character vector with axis labels for speed and the additional variables supplied in *concurVars*. For the TDRcalibrate method, *concurVars* is a **character** vector indicating which additional components from the concurrent data frame should also be plotted, if any.

`diveNo` Numeric vector with dive numbers to plot.

`labels` One of “phase.id” or “dive.phase”, specifying whether to label observations based on the gross phase ID of the "`TDR`" object, or based on each dive phase, respectively.

`surface` Logical indicating whether to plot surface readings.

Value

If called with the `interact` argument set to `TRUE`, returns coordinates from the ZOC procedure (see `zoc`).

Methods

plotTDR `signature(x="TDR")`: interactive graphical display of the data, with zooming and panning capabilities.

plotTDR `signature(x="TDRspeed")`: As the TDR method, but also plots the concurrent speed readings.

plotTDR `signature(x="TDRcalibrate")`: plot the TDR object, labelling identified sections of it (see `Usage`).

DRAFT

Author(s)

Sebastian P. Luque ⟨spluque@gmail.com⟩

See Also

zoc

Examples

```
data(divesTDR)
divesTDR

plotTDR(divesTDR, interact=FALSE)

data(divesTDRcalibrate)
divesTDRcalibrate

plotTDR(divesTDRcalibrate, interact=FALSE)
plotTDR(divesTDRcalibrate, diveNo=19:25, interact=FALSE)
plotTDR(divesTDRcalibrate, labels="dive.phase", interact=FALSE)
```

---

readLocs — *Read comma-delimited file with location data*

Description

Read a comma delimited (*.csv) file with (at least) time, latitude, longitude readings.

Usage

```
readLocs(file, loc.idCol, idCol, dateCol, timeCol=NULL,
         dtformat="%m/%d/%Y %H:%M:%S", tz="GMT",
         classCol, lonCol, latCol, alt.lonCol=NULL, alt.latCol=NULL, ...)
```

Arguments

`file`	A string indicating the name of the file to read. Provide the entire path if the file is not on the current directory.
`loc.idCol`	Column number containing location ID. If missing, a `loc.id` column is generated with sequential integers as long as the input.
`idCol`	Column number containing an identifier for locations belonging to different groups. If missing, an id column is generated with number one repeated as many times as the input.
`dateCol`	Column number containing dates, and, optionally, times.
`timeCol`	Column number containing times.

`dtformat`	A string, specifying the format in which the date and time columns, when pasted together, should be interpreted (see `strptime`) in `file`.
`tz`	A string indicating the time zone for the date and time readings.
`lonCol`	Column number containing longitude readings.
`latCol`	Column number containing latitude readings.
`classCol`	Column number containing the ARGOS rating for each location.
`alt.lonCol`	Column number containing alternative longitude readings.
`alt.latCol`	Column number containing alternative latitude readings.
`...`	Passed to `read.csv`

Details

The file must have a header row identifying each field, and all rows must be complete (i.e. have the same number of fields). Field names need not follow any convention.

Value

A data frame.

Author(s)

Sebastian P. Luque ⟨spluque@gmail.com⟩

Examples

```
locs <- readLocs(system.file(file.path("data", "sealLocs.csv"),
                             package="diveMove"), idCol=1, dateCol=2,
                 dtformat="%Y-%m-%d %H:%M:%S",
                 classCol=3, lonCol=4, latCol=5)

summary(locs)
```

`readTDR` *Read comma-delimited file with TDR data*

Description

Read a comma delimited (*.csv) file containing time-depth recorder (TDR) data from various TDR models. Return a `TDR` or `TDRspeed` object. `createTDR` creates an object of one of these classes from other objects.

DRAFT

Usage

```
readTDR(file, dateCol=1, timeCol=2, depthCol=3, speed=FALSE,
        subsamp=5, concurrentCols=4:6,
        dtformat="%d/%m/%Y %H:%M:%S", tz="GMT")
createTDR(time, depth, concurrentData=data.frame(), speed=FALSE, dtime, file)
```

Arguments

`file`	A string indicating the path to the file to read.
`dateCol`	Column number containing dates, and optionally, times.
`timeCol`	Column number with times.
`depthCol`	Column number containing depth readings.
`speed`	For `readTDR`: Logical indicating whether speed is included in one of the columns of concurrentCols.
`subsamp`	Subsample rows in `file` with `subsamp` interval, in s.
`concurrentCols`	Column numbers to include as concurrent data collected.
`dtformat`	A string, specifying the format in which the date and time columns, when pasted together, should be interpreted (see `strptime`).
`tz`	A string indicating the time zone assumed for the date and time readings.
`time`	A `POSIXct` object with date and time readings for each reading.
`depth`	Numeric vector with depth readings.
`concurrentData`	Data frame with additional, concurrent data collected.
`dtime`	Sampling interval used in seconds. If missing, it is calculated from the `time` argument.

Details

The input file is assumed to have a header row identifying each field, and all rows must be complete (i.e. have the same number of fields). Field names need not follow any convention. However, depth and speed are assumed to be in m, and $m \cdot s^{-1}$, respectively, for further analyses.

If *speed* is TRUE and concurrentCols contains a column named speed or velocity, then an object of class TDRspeed is created, where speed is considered the column matching this name.

Value

An object of class 'TDR' or 'TDRspeed'.

Author(s)

Sebastian P. Luque ⟨spluque@gmail.com⟩

DRAFT

Examples

```
readTDR(system.file(file.path("data", "dives.csv"),
                    package="diveMove"), speed=TRUE)

## Or more pedestrian
tdrX <- read.csv(system.file(file.path("data", "dives.csv"),
                             package="diveMove"), na.strings="", as.is=TRUE)
date.time <- paste(tdrX$date, tdrX$time)
tdr.time <- as.POSIXct(strptime(date.time, format="%d/%m/%Y %H:%M:%S"),
                       tz="GMT")
createTDR(tdr.time, tdrX$depth, concurrentData=data.frame(speed=tdrX$speed),
          file="dives.csv", speed=TRUE)
```

`rqPlot` *Plot of quantile regression for speed calibrations*

Description

Plot of quantile regression for assessing quality of speed calibrations

Usage

```
rqPlot(rddepth, speed, z, contours, rqFit, main="qtRegression",
       xlab="rate of depth change (m/s)", ylab="speed (m/s)",
       colramp=colorRampPalette(c("white", "darkblue")),
       col.line="red", cex.pts=1)
```

Arguments

`speed`	Speed in m/s.
`rddepth`	Numeric vector with rate of depth change.
`z`	A list with the bivariate kernel density estimates (1st component the x points of the mesh, 2nd the y points, and 3rd the matrix of densities).
`contours`	List with components: `pts` which should be a matrix with columns named `x` and `y`, `level` a number indicating the contour level the points in `pts` correspond to.
`rqFit`	Object of class "rq" representing a quantile regression fit of rate of depth change on mean speed.
`main`	String; title prefix to include in ouput plot.
`xlab, ylab`	axis labels.
`colramp`	Function taking an integer n as an argument and returning n colors.
`col.line`	Color to use for the regression line.
`cex.pts`	A numerical value specifying the amount by which to enlarge the size of points.

DRAFT

Details

The dashed line in the plot represents a reference indicating a one to one relationship between speed and rate of depth change. The other line represent the quantile regression fit.

Author(s)

Sebastian P. Luque ⟨spluque@gmail.com⟩

See Also

`diveStats`

`sealLocs` *Ringed and Gray Seal ARGOS Satellite Location Data*

Description

Satellite locations of a gray (Stephanie) and a ringed (Ringy) seal caught and released in New York.

Format

A data frame with the following information:

id String naming the seal the data come from.

time The date and time of the location.

class The ARGOS location quality classification.

lon, lat x and y geographic coordinates of each location.

Source

WhaleNet Satellite Tracking Program `http://whale.wheelock.edu/Welcome.html`.

See Also

`readLocs`, `distSpeed`.

DRAFT

`TDR-accessors` *Coerce, Extractor, and Replacement methods for class "TDR" objects*

Description

Basic methods for manipulating objects of class "TDR".

Show Methods

show `signature(object="TDR")`: print an informative summary of the data.

Coerce Methods

as.data.frame `signature(x="TDR")`: Coerce object to data.frame. This method returns a data frame, with attributes 'file' and 'dtime' indicating the source file and the interval between samples.

as.data.frame `signature(x="TDRspeed")`: Coerce object to data.frame. Returns an object as for `TDR` objects.

as.TDRspeed `signature(x="TDR")`: Coerce object to TDRspeed class.

Extractor Methods

[`signature(x="TDR")`: Subset a TDR object; these objects can be subsetted on a single index i. Selects given rows from object.

getDepth `signature(x = "TDR")`: depth slot accessor.

getCCData `signature(x="TDR", y="missing")`: concurrentData slot accessor.

getCCData `signature(x="TDR", y="character")`: access component named y in x.

getDtime `signature(x = "TDR")`: sampling interval accessor.

getFileName `signature(x="TDR")`: source file name accessor.

getTime `signature(x = "TDR")`: time slot accessor.

getSpeed `signature(x = "TDRspeed")`: speed accessor for `TDRspeed` objects.

Replacement Methods

depth<- `signature(x="TDR")`: depth replacement.

speed<- `signature(x="TDR")`: speed replacement.

ccData<- `signature(x="TDR")`: concurrent data frame replacement.

Author(s)

Sebastian P. Luque ⟨spluque@gmail.com⟩

See Also

`extractDive`, `plotTD`.

DRAFT

Examples

```
data(divesTDR)

## Retrieve the name of the source file
getFileName(divesTDR)
## Retrieve concurrent temperature measurements
temp <- getCCData(divesTDR, "temperature")

## Coerce to a data frame
dives.df <- as.data.frame(divesTDR)
head(dives.df)

## Replace speed measurements
newspeed <- getSpeed(divesTDR) + 2
speed(divesTDR) <- newspeed
```

`TDRcalibrate-accessors` *Methods to Show and Extract Basic Information from "TDRcalibrate" Objects*

Description

Plot, print summaries and extract information from "TDRcalibrate" objects.

Usage

```
## S4 method for signature 'TDRcalibrate, missing':
getDAct(x)
## S4 method for signature 'TDRcalibrate, character':
getDAct(x, y)
## S4 method for signature 'TDRcalibrate, missing':
getDPhaseLab(x)
## S4 method for signature 'TDRcalibrate, numeric':
getDPhaseLab(x, diveNo)
## S4 method for signature 'TDRcalibrate, missing':
getGAct(x)
## S4 method for signature 'TDRcalibrate, character':
getGAct(x, y)
```

Arguments

`x`	"`TDRcalibrate`" object.
`diveNo`	numeric vector with dive numbers to plot.
`y`	string; “dive.id”, “dive.activity”, or “postdive.id” in the case of `getDAct`, to extract the numeric dive ID, the factor identifying dive phases in each dive, or the numeric postdive ID, respectively. In the case of `getGAct` it should be

one of “phase.id”, “activity”, “begin”, or “end”, to extract the numeric phase ID for each observation, a factor indicating what major activity the observation corresponds to, or the beginning and end times of each phase in the record, respectively.

Value

The extractor methods return an object of the same class as elements of the slot they extracted.

Show Methods

show `signature(object="TDRcalibrate")`: prints an informative summary of the data.

Extractor Methods

getDAct `signature(x="TDRcalibrate", y="missing")`: this accesses the `dive.activity` slot of `TDRcalibrate` objects. Thus, it extracts a data frame with vectors identifying all readings to a particular dive and postdive number, and a factor identifying all readings to a particular activity.

getDAct `signature(x="TDRcalibrate", y = "character")`: as the method for missing `y`, but selects a particular vector to extract. See `TDRcalibrate` for possible strings.

getDPhaseLab `signature(x="TDRcalibrate", diveNo = "missing")`: extracts a factor identifying all readings to a particular dive phase. This accesses the dive.phases slot of `TDRcalibrate` objects, which is a factor.

getDPhaseLab `signature(x="TDRcalibrate", diveNo = "numeric")`: as the method for missing `y`, but selects data from a particular dive number to extract.

getGAct `signature(x="TDRcalibrate", y="missing")`: this accesses the `gross.activity` slot of `TDRcalibrate` objects, which is a named list. It extracts elements that divide the data into major wet and dry activities.

getGAct `signature(x="TDRcalibrate", y="character")`: as the method for missing `y`, but extracts particular elements.

getTDR `signature(x="TDRcalibrate")`: this accesses the tdr slot of `TDRcalibrate` objects, which is a `TDR` object.

getSpeedCoef `signature(x="TDRcalibrate")`: this accesses the `speed.calib.coefs` slot of `TDRcalibrate` objects; the speed calibration coefficients.

Author(s)

Sebastian P. Luque ⟨spluque@gmail.com⟩

Examples

```
data(divesTDRcalibrate)

divesTDRcalibrate                 # show

## Beginning times of each successive phase in record
```

DRAFT

```
getGAct(divesTDRcalibrate, "begin")

## Factor of dive IDs
dids <- getDAct(divesTDRcalibrate, "dive.id")
table(dids[dids > 0])           # samples per dive

## Factor of dive phases for given dive
getDPhaseLab(divesTDRcalibrate, 19)
```

TDRcalibrate-class *Class "TDRcalibrate" for dive analysis*

Description

This class holds information produced at various stages of dive analysis. Methods are provided for extracting data from each slot.

Details

This is perhaps the most important class in diveMove, as it holds all the information necessary for calculating requested summaries for a TDR.

Objects from the Class

Objects can be created by calls of the form `new("TDRcalibrate", ...)`. The objects of this class contain information necessary to divide the record into sections (e.g. dry/water), dive/surface, and different sections within dives. They also contain the parameters used to calibrate speed and criteria to divide the record into phases.

Slots

tdr: Object of class "TDR".

This slot contains the time, zero-offset corrected depth, and possibly a data frame. If the object is also of class "TDRspeed", then the data frame might contain calibrated or uncalibrated speed. See `readTDR` and the accessor function `getTDR` for this slot.

gross.activity: Object of class "list".

This slot holds a list of the form returned by `.detPhase`, composed of 4 elements. It contains a vector (named `phase.id`) numbering each major activity phase found in the record, a factor (named `activity`) labelling each row as being dry, wet, or trivial wet activity. These two elements are as long as there are rows in `tdr`. This list also contains two more vectors, named `begin` and `end`: one with the beginning time of each phase, and another with the ending time; both represented as `POSIXct` objects. See `.detPhase`.

dive.activity: Object of class "data.frame".

This slot contains a data.frame of the form returned by `.detDive`, with as many rows as those in `tdr`, consisting of three vectors named: `dive.id`, which is an integer vector, sequentially numbering each dive (rows that are not part of a dive are labelled 0), dive.activity is a factor which completes that in `activity` above, further identifying rows in the record belonging to a dive. The third vector in `dive.activity` is an integer vector sequentially numbering each postdive interval (all rows that belong to a dive are labelled 0). See `.detDive`, and `getDAct` to access all or any one of these vectors.

`dive.phases`: Object of class "factor". must be the same as value returned by `.labDivePhase`. This slot is a factor that labels each row in the record as belonging to a particular phase of a dive. It has the same form as objects returned by `.labDivePhase`.

`dry.thr`: Object of class "numeric" the temporal criteria used for detecting dry periods that should be considered as wet.

`wet.thr`: Object of class "numeric" the temporal criteria used for detecting periods wet that should not be considered as foraging time.

`dive.thr`: Object of class "numeric" the criteria used for defining a dive.

`speed.calib.coefs`: Object of class "numeric" the intercept and slope derived from the speed calibration procedure. Defaults to c(0, 1) meaning uncalibrated speeds.

Author(s)

Sebastian P. Luque ⟨spluque@gmail.com⟩

See Also

`TDR` for links to other classes in the package. `TDRcalibrate-methods` for the various methods available.

`TDR-class` *Classes "TDR" and "TDRspeed" for representing TDR information*

Description

These classes store information gathered by time-depth recorders.

Details

Since the data to store in objects of these clases usually come from a file, the easiest way to construct such objects is with the function `readTDR` to retrieve all the necessary information. The methods listed above can thus be used to access all slots.

Objects from the Class

Objects can be created by calls of the form `new("TDR", ...)` and `new("TDRspeed", ...)`.

`TDR` objects contain concurrent time and depth readings, as well as a string indicating the file the data originates from, and a number indicating the sampling interval for these data. `TDRspeed` extends `TDR` objects containing additional concurrent speed readings.

Slots

In class *TDR*:

`file`: Object of class “character”, string indicating the file where the data comes from.

`dtime`: Object of class “numeric”, sampling interval in seconds.

`time`: Object of class “POSIXct”, time stamp for every reading.

`depth`: Object of class “numeric”, depth (m) readings.

`concurrentData`: Object of class “data.frame”, optional data collected concurrently.

Class *TDRspeed* must also satisfy the condition that a component of the concurrentData slot is named speed or velocity, containing the measured speed, a vector of class “numeric” containing speed measurements in (m/s) readings.

Author(s)

Sebastian P. Luque ⟨spluque@gmail.com⟩

See Also

`readTDR`, `TDRcalibrate`.

`timeBudget-methods` *Describe the Time Budget of Major Activities from "TDRcalibrate" object.*

Description

Summarize the major activities recognized into a time budget.

Usage

```
## S4 method for signature 'TDRcalibrate, logical':
timeBudget(obj, ignoreZ)
```

Arguments

`obj`	"`TDRcalibrate`" object.
`ignoreZ`	Logical indicating whether to ignore trivial aquatic periods.

Details

Ignored trivial aquatic periods are collapsed into the enclosing dry period.

DRAFT

Value

A data frame with components:

- `phaseno` A numeric vector numbering each period of activity.
- `activity` A factor labelling the period with the corresponding activity.
- `beg, end` `POSIXct` objects indicating the beginning and end of each period.

Author(s)

Sebastian P. Luque ⟨spluque@gmail.com⟩

See Also

`calibrateDepth`

Examples

```
data(divesTDRcalibrate)

timeBudget(divesTDRcalibrate, TRUE)
```

`zoc` *Interactive zero-offset correction of TDR data*

Description

Correct zero-offset in TDR records, with the aid of a graphical user interface (GUI), allowing for dynamic selection of offset and multiple time windows to perform the adjustment.

Usage

```
zoc(time, depth, offset)
plotTD(time, depth, concurVars=NULL, xlim=NULL, depth.lim=NULL,
       xlab="time (dd-mmm hh:mm)", ylab.depth="depth (m)",
       concurVarTitles=deparse(substitute(concurVars)),
       xlab.format="%d-%b %H:%M", sunrise.time="06:00:00",
       sunset.time="18:00:00", night.col="gray60",
       phaseCol=NULL, interact=TRUE, key=TRUE, cex.pts=0.4, ...)
```

DRAFT

Arguments

- `time` — `POSIXct` object with date and time.
- `depth` — Numeric vector with depth in m.
- `offset` — Known amount of meters to subtract for zero-offset correcting depth throughout the entire TDR record.
- `concurVars` — Matrix with additional variables in each column to plot concurrently with depth.
- `xlim` — Vector of length 2, with lower and upper limits of time to be plotted.
- `depth.lim` — Numeric vector of length 2, with the lower and upper limits of depth to be plotted.
- `xlab, ylab.depth` — Strings to label the corresponding y-axes.
- `concurVarTitles` — Character vector of titles to label each new variable given in *concurVars*.
- `xlab.format` — Format string for formatting the x axis; see `strptime`.
- `sunrise.time, sunset.time` — Character string with time of sunrise and sunset, respectively, in 24 hr format. This is used for shading night time.
- `night.col` — Color for shading night time.
- `phaseCol` — Factor dividing rows into sections.
- `interact` — Logical; whether to provide interactive tcltk controls and access to the associated ZOC functionality.
- `key` — Logical indicating whether to draw a key.
- `cex.pts` — Passed to `points` to set the relative size of points to plot (if any).
- `...` — Arguments passed to `par`; useful defaults `las=1`, `bty="n"`, and `mar` (the latter depending on whether additional concurrent data will be plotted) are provided, but they can be overridden.

Details

These functions are used primarily to correct, visually, drifts in the pressure transducer of TDR records. `zoc` calls `plotDive`, which plots depth and, optionally, speed vs. time with the possibility zooming in and out on time, changing maximum depths displayed, and panning through time. The option to zero-offset correct sections of the record gathers x and y coordinates for two points, obtained by clicking on the plot region. The first point clicked indicates the offset and beginning time of section to correct, and the second one indicates the ending time of the section to correct. Multiple sections of the record can be corrected in this manner, by panning through the time and repeating the procedure. In case there's overlap between zero offset corrected windows, the last one prevails.

Once the whole record has been zero-offset corrected, remaining points with depth values lower than zero, are turned into zeroes, as these are assumed to be values at the surface.

Value

`zoc` returns a numeric vector, as long as `depth` of zero-offset corrected depths.

`plotTD` returns (invisibly) a list with as many components as sections of the record that were zero-offset corrected, each consisting of two further lists with the same components as those returned by `locator`.

Author(s)

Sebastian P. Luque ⟨spluque@gmail.com⟩, with many ideas from CRAN package sfsmisc.

See Also

`calibrateDepth`, and `plotTDR`.

Examples

```
data(divesTDR)

## Use interact=TRUE (default) to set the offset interactively
depth.zoc <- zoc(getTime(divesTDR), getDepth(divesTDR), offset=3)
plotTD(getTime(divesTDR), depth.zoc, interact=FALSE)
```

DRAFT

Index

DRAFT

DRAFT

DRAFT