

Package ‘diveMove’

October 21, 2009

Type Package

Title Dive analysis and calibration

Version 0.9.6

Depends R (>= 2.4.0), methods, stats4

Suggests KernSmooth, quantreg, tcltk

Imports stats4

Author Sebastian P. Luque <spluque@gmail.com>

Maintainer Sebastian P. Luque <spluque@gmail.com>

Description Functions to filter and summarize time-depth recorder (TDR) data, and miscellaneous functions for handling location data.

LazyLoad yes

ZipData no

Collate AllClass.R AllGenerics.R AllMethod.R austFilter.R bouts.R calibrate.R detDive.R detPhase.R distSpeed.R diveStats.R oneDiveStats.R plotTD.R readLocs.R readTDR.R speedStats.R stampDive.R zoc.R zzz.R

License GPL-3

Repository CRAN

Date/Publication 2009-10-21 16:54:21

R topics documented:

diveMove-package	2
austFilter	3
bout-methods	6
bout-misc	7
bouts2MLE	9

bouts2NLS	12
calibrateDepth	14
calibrateSpeed	16
distSpeed	17
dives	18
diveStats	20
extractDive-methods	22
plotTDR-methods	23
readLocs	24
readTDR	25
rqPlot	27
sealLocs	28
TDR-accessors	29
TDR-class	30
TDRcalibrate-accessors	31
TDRcalibrate-class	33
timeBudget-methods	34
zoc	35

Index	38
--------------	-----------

diveMove-package *Dive Analysis and Calibration*

Description

This package is a collection of functions for visualizing, and analyzing depth and speed data from time-depth recorders TDRs. 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)

```

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).

<code>speed.thr</code>	Speed threshold (m/s) above which filter tests should fail any given point.
<code>dist.thr</code>	Distance threshold (km) above which the last filter test should fail any given point.
<code>window</code>	Integer indicating the size of the moving window over which tests should be carried out.
<code>x</code>	3-column matrix with column 1: <code>POSIXct</code> 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 in a single call 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 (see [Warning](#) section below).

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 Freitas et al. (2008). According to the original description of the algorithm in McConnell et al. (1992), the filter makes a single pass through all locations. Austin et al. (2003) and other authors may have used the filter this way. However, as Freitas et al. (2008) noted, this causes locations adjacent to those flagged as failing to fail also, thereby rejecting too many locations. In `diveMove`, the algorithm was modified to reject only the "peaks" in each series of consecutive locations having root mean square speed higher than threshold.

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.

Freitas C, Lydersen, C, Fedak MA, Kovacs KM. 2008. A simple new algorithm to filter marine mammal ARGOS locations. *Marine Mammal Science* DOI: 10.1111/j.1748-7692.2007.00180.x

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)

## 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 (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 <code>plotBouts2.nls</code> and <code>plotBouts2.mle</code> .

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.

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

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

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

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

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

`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 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, so these suggested values may not be useful in 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$postdive.dur[divesSummary$strip.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, ...)

```

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 <i>lnfreq</i> (log frequencies) and corresponding x (mid points of histogram bins).
start, maxiter	Arguments passed to <code>nls</code> .
fit	<code>nls</code> object.
bec.lty	Line type specification for drawing the BEC reference line.
...	Arguments passed to <code>plot.default</code> .

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.

Examples

```
data(divesSummary)
## Postdive durations
postdives <- divesSummary$postdive.dur[divesSummary$strip.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

<code>x</code>	An object of class <code>TDR</code> for <code>calibrateDepth</code> or an object of class <code>TDRcalibrate</code> for <code>calibrateSpeed</code> .
<code>dry.thr</code>	Dry error threshold in seconds. Dry phases shorter than this threshold will be considered as wet.
<code>wet.thr</code>	Wet threshold in seconds. At-sea phases shorter than this threshold will be considered as trivial wet.
<code>dive.thr</code>	Threshold depth below which an underwater phase should be considered a dive.
<code>offset</code>	Argument to <code>zoc</code> . If not provided, the offset is obtained using an interactive plot of the data.
<code>descent.crit.q</code>	Critical quantile of rates of descent below which descent is deemed to have ended.
<code>ascent.crit.q</code>	Critical quantile of rates of ascent above which ascent is deemed to have started.
<code>wiggle.tol</code>	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`.

Detection of dive phases

A bottom depth is defined as the maximum depth multiplied by a factor (`wiggle.tol`, [0, 1])

Descent Using all depths from the first one in the dive down to the maximum depth, the rate of descent for each segment is calculated, and a critical rate is defined as the quantile (`descent.crit.q`) of all the positive rates of descent. This subsetting avoids defining a critical rate that may be negative, due to wiggling during the descent.

To allow detection of wiggles during descent, a vector of the indices of the rates of descent that were lower than the critical value is defined, and a logical vector with TRUE for rates of descent ≥ 0 above the bottom depth defined previously is also created.

The following tests are performed (in order):

If there were any rates below the critical value, as well as any descent wiggles, the indices where the wiggles occurred are removed. If this resulted in removal of all indices, then the index defining the end of descent is the number of rates of descent, otherwise it is the last after the removal.

If there were not any rates below the critical value, the index defining the end of descent is the number of rates of descent, otherwise it is the first of them.

Ascent The order of depths is reversed in order to detect ascent starting from the bottom, taking all depths after the maximum depth. The rate of ascent for each segment is calculated, and a critical rate is defined as the quantile (`ascent.crit.q`) of all the positive rates of ascent, analogously to the descent detection procedure.

To allow detection of bottom wiggling, a vector of the indices of the rates of ascent that were higher than the critical value is defined, and a logical vector with TRUE for rates of ascent ≤ 0 below the bottom depth defined previously is also created.

The following tests are performed (in order):

If there were any bottom wiggles, then the index defining the beginning of ascent is that corresponding to the maximum depth plus that corresponding to the last bottom wiggle, otherwise:

If there were no rates above the critical value, then the index defining the beginning of ascent is the last reading below the surface. Otherwise, it is the one corresponding to the maximum depth plus the first of the indices of rates above the critical value.

The particular dive phase categories are subsequently defined using simple set operations.

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)
if (dev.interactive(orElse=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

<code>x</code>	An object of class TDR for calibrateDepth or an object of class TDRcalibrate for calibrateSpeed .
<code>tau</code>	Quantile on which to regress speed on rate of depth change; passed to rq .
<code>contour.level</code>	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.
<code>z</code>	Only changes in depth larger than this value will be used for calibration.
<code>bad</code>	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 <code>rqPlot</code> .
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`.

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
```

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)

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

diveStats

Per-dive statistics

Description

Calculate dive statistics in TDR records.

Usage

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

Arguments

<code>x</code>	A <code>TDRcalibrate-class</code> object for <code>diveStats</code> and <code>stampDive</code> , and a data frame containing a single dive's data (a factor identifying the dive phases, a <code>POSIXct</code> object with the time for each reading, a numeric depth vector, and a numeric speed vector) for <code>oneDiveStats</code> .
<code>interval</code>	Sampling interval for interpreting <code>x</code> .
<code>speed</code>	Logical; should speed statistics be calculated?
<code>ignoreZ</code>	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):

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

```

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, from the surface plane.
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, from the bottom plane.
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)

```

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)
```

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

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

<code>x</code>	<code>TDR</code> , <code>TDRspeed</code> , or <code>TDRcalibrate</code> object.
<code>concurVars</code> , <code>concurVarTitles</code> , ...	Arguments passed to <code>plotTD</code> . For the <code>TDRspeed</code> method, <code>concurVars</code> is a matrix with variables to plot, in addition to speed, if any. <code>concurVarTitles</code> in this case is a character vector with axis labels for speed and the additional variables supplied in <code>concurVars</code> . For the <code>TDRcalibrate</code> method, <code>concurVars</code> is a character vector indicating which additional components from the concurrent data frame should also be plotted, if any.
<code>diveNo</code>	Numeric vector with dive numbers to plot.
<code>labels</code>	One of “phase.id” or “dive.phase”, specifying whether to label observations based on the gross phase ID of the <code>TDR</code> object, or based on each dive phase, respectively.
<code>surface</code>	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).

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.

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

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

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

Value

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

Note

Although `TDR` and `TDRspeed` classes check that time stamps are in increasing order, the integrity of the input must be thoroughly verified for common errors present in text output from TDR devices such as duplicate records, missing time stamps and non-numeric characters in numeric fields. These errors are much more efficiently dealt with outside of GNU R using tools like GNU `awk` or GNU `sed`, so `diveMove` does not currently attempt to fix these errors.

Author(s)

Sebastian P. Luque (spluque@gmail.com)

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.

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`.

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

```

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 classes 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](#).

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 <code>getDAct</code> , to extract the numeric dive ID, the factor identifying dive phases in each dive, or the numeric postdive ID, respectively. In the case of <code>getGAct</code> 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
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.

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.

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	<code>POSIXct</code> 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, ...)
```

Arguments

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

Index

- *Topic **arith**
 - diveStats, 19
 - rqPlot, 26
- *Topic **classes**
 - TDR-class, 29
 - TDRcalibrate-class, 32
- *Topic **datasets**
 - dives, 18
 - sealLocs, 27
- *Topic **hplot**
 - rqPlot, 26
- *Topic **iplot**
 - zoc, 35
- *Topic **iteration**
 - austFilter, 3
- *Topic **manip**
 - austFilter, 3
 - bout-misc, 7
 - bouts2MLE, 9
 - bouts2NLS, 12
 - calibrateDepth, 13
 - calibrateSpeed, 16
 - distSpeed, 17
 - readLocs, 23
 - readTDR, 25
 - rqPlot, 26
- *Topic **math**
 - calibrateDepth, 13
 - calibrateSpeed, 16
 - distSpeed, 17
 - diveStats, 19
- *Topic **methods**
 - bout-methods, 6
 - extractDive-methods, 21
 - plotTDR-methods, 22
 - TDR-accessors, 28
 - TDRcalibrate-accessors, 30
 - timeBudget-methods, 34
- *Topic **misc**
 - bout-misc, 7
- *Topic **models**
 - bouts2MLE, 9
 - bouts2NLS, 12
- *Topic **package**
 - diveMove-package, 2
 - .detDive, 33
 - .detPhase, 21, 33
 - .labDivePhase, 33
 - [, TDR-method (*TDR-accessors*), 28
 - as.data.frame, TDR-method (*TDR-accessors*), 28
 - as.TDRspeed (*TDR-accessors*), 28
 - as.TDRspeed, TDR-method (*TDR-accessors*), 28
 - austFilter, 3
 - bec2 (*bout-methods*), 6
 - bec2, mle-method (*bout-methods*), 6
 - bec2, nls-method (*bout-methods*), 6
 - bout-methods, 6
 - bout-misc, 7
 - boutfreqs (*bout-misc*), 7
 - boutinit, 10
 - boutinit (*bout-misc*), 7
 - bouts.mle, 7, 8, 13
 - bouts.mle (*bouts2MLE*), 9
 - bouts2.LL, 10
 - bouts2.LL (*bouts2MLE*), 9
 - bouts2.ll, 10
 - bouts2.ll (*bouts2MLE*), 9
 - bouts2.mleBEC (*bouts2MLE*), 9
 - bouts2.mleFUN (*bouts2MLE*), 9
 - bouts2.nls, 7, 8
 - bouts2.nls (*bouts2NLS*), 12
 - bouts2.nlsBEC (*bouts2NLS*), 12
 - bouts2.nlsFUN (*bouts2NLS*), 12
 - bouts2MLE, 9
 - bouts2NLS, 12

- calibrateDepth, 2, 13, 14, 16, 19, 34, 36
- calibrateSpeed, 2, 14, 16, 16
- ccData<- (TDR-accessors), 28
- ccData<- , TDR, data.frame-method
(TDR-accessors), 28
- coerce, TDR, data.frame-method
(TDR-accessors), 28
- coerce, TDR, TDRspeed-method
(TDR-accessors), 28
- createTDR (readTDR), 25
- curve, 9
- data.frame, 20, 30
- depth<- (TDR-accessors), 28
- depth<- , TDR, numeric-method
(TDR-accessors), 28
- distSpeed, 4, 17, 28
- diveMove, 26
- diveMove (diveMove-package), 2
- diveMove-package, 2
- dives, 18
- divesSummary (dives), 18
- diveStats, 19, 19, 27
- divesTDR (dives), 18
- divesTDRcalibrate (dives), 18
- extractDive, 29
- extractDive
(extractDive-methods), 21
- extractDive, TDR, numeric, numeric-method
(extractDive-methods), 21
- extractDive, TDRcalibrate, numeric, missing-method
(extractDive-methods), 21
- extractDive-methods, 21
- getCCData (TDR-accessors), 28
- getCCData, TDR, character-method
(TDR-accessors), 28
- getCCData, TDR, missing-method
(TDR-accessors), 28
- getDAct, 33
- getDAct (TDRcalibrate-accessors),
30
- getDAct, TDRcalibrate, character-method
(TDRcalibrate-accessors),
30
- getDAct, TDRcalibrate, missing-method
(TDRcalibrate-accessors),
30
- getDepth (TDR-accessors), 28
- getDepth, TDR-method
(TDR-accessors), 28
- getDPhaseLab
(TDRcalibrate-accessors),
30
- getDPhaseLab, TDRcalibrate, missing-method
(TDRcalibrate-accessors),
30
- getDPhaseLab, TDRcalibrate, numeric-method
(TDRcalibrate-accessors),
30
- getDtime (TDR-accessors), 28
- getDtime, TDR-method
(TDR-accessors), 28
- getFileName (TDR-accessors), 28
- getFileName, TDR-method
(TDR-accessors), 28
- getGAct (TDRcalibrate-accessors),
30
- getGAct, TDRcalibrate, character-method
(TDRcalibrate-accessors),
30
- getGAct, TDRcalibrate, missing-method
(TDRcalibrate-accessors),
30
- getSpeed (TDR-accessors), 28
- getSpeed, TDRspeed-method
(TDR-accessors), 28
- getSpeedCoef
(TDRcalibrate-accessors),
30
- getSpeedCoef, TDRcalibrate-method
(TDRcalibrate-accessors),
30
- getTDR, 33
- getTDR (TDRcalibrate-accessors),
30
- getTDR, TDRcalibrate-method
(TDRcalibrate-accessors),
30
- getTime (TDR-accessors), 28
- getTime, TDR-method
(TDR-accessors), 28
- grpSpeedFilter (austFilter), 3
- labelBouts (bout-misc), 7
- locator, 36
- logit, 10, 11

- logit (*bout-misc*), 7
- mle, 6, 9–11
- nls, 6, 12
- oneDiveStats (*diveStats*), 19
- optim, 11
- par, 36
- plot.default, 12
- plot.ecdf, 9
- plotBouts (*bout-methods*), 6
- plotBouts, mle-method
(*bout-methods*), 6
- plotBouts, nls-method
(*bout-methods*), 6
- plotBouts2.cdf (*bouts2MLE*), 9
- plotBouts2.mle, 6
- plotBouts2.mle (*bouts2MLE*), 9
- plotBouts2.nls, 6
- plotBouts2.nls (*bouts2NLS*), 12
- plotTD, 22, 29
- plotTD (*zoc*), 35
- plotTDR, 36
- plotTDR (*plotTDR-methods*), 22
- plotTDR, TDR-method
(*plotTDR-methods*), 22
- plotTDR, TDRcalibrate-method
(*plotTDR-methods*), 22
- plotTDR, TDRspeed-method
(*plotTDR-methods*), 22
- plotTDR-methods, 22
- points, 35
- POSIXct, 30, 33, 34
- read.csv, 24
- readLocs, 23, 28
- readTDR, 18, 19, 25, 29, 30, 33
- rmsDistFilter (*austFilter*), 3
- rq, 16
- rqPlot, 16, 26
- sealLocs, 27
- show, TDR-method (*TDR-accessors*),
28
- show, TDRcalibrate-method
(*TDRcalibrate-accessors*),
30
- speed<- (*TDR-accessors*), 28
- speed<-, TDRspeed, numeric-method
(*TDR-accessors*), 28
- stampDive, 2, 19
- stampDive (*diveStats*), 19
- strptime, 24, 25, 35
- TDR, 14, 16, 18, 19, 21, 22, 26, 28, 31, 33
- TDR (*TDR-class*), 29
- TDR-class, 2
- TDR-accessors, 28
- TDR-class, 29
- TDR-methods (*TDR-accessors*), 28
- TDRcalibrate, 13–19, 22, 30–32, 34
- TDRcalibrate
(*TDRcalibrate-class*), 32
- TDRcalibrate-class, 19, 21
- TDRcalibrate-methods, 33
- TDRcalibrate-accessors, 30
- TDRcalibrate-class, 32
- TDRcalibrate-methods
(*TDRcalibrate-accessors*),
30
- TDRspeed, 21, 22, 25, 26, 28
- TDRspeed (*TDR-class*), 29
- TDRspeed-class (*TDR-class*), 29
- timeBudget, 2
- timeBudget (*timeBudget-methods*),
34
- timeBudget, TDRcalibrate, logical-method
(*timeBudget-methods*), 34
- timeBudget-methods, 34
- unLogit, 11
- unLogit (*bout-misc*), 7
- zoc, 14, 15, 21, 23, 35