Package ‘diveMove’

February 19, 2015

Type Package
Title Dive analysis and calibration
Version 1.3.9
Depends R (>= 2.13.0), methods, stats4
Suggests lattice, tcltk
Imports caTools, KernSmooth, quantreg, geosphere
Author Sebastian P. Luque <spluque@gmail.com>
Maintainer Sebastian P. Luque <spluque@gmail.com>
Description Utilities to represent, visualize, filter, analyse, and summarize
time-depth recorder (TDR) data. Miscellaneous functions for
handling location data are also provided.
LazyLoad yes
LazyData no
ZipData no
BuildResaveData no
oneDiveStats.R plotTDR.R plotZOC.R readLocs.R readTDR.R
License GPL-3
URL http://r-forge.r-project.org/projects/divemove
NeedsCompilation no
Repository CRAN
Date/Publication 2014-02-20 07:02:18

R topics documented:

  diveMove-package .................................................. 2
  austFilter ......................................................... 3
**diveMove-package**

**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`. 

```r

bout-methods ...................................................... 6
bout-misc .......................................................... 8
bouts2MLE ........................................................... 10
bouts2NLS ............................................................. 13
bouts3NLS ............................................................. 15
calibrateDepth ......................................................... 17
calibrateSpeed ......................................................... 21
distSpeed ............................................................... 22
diveModel-class ......................................................... 23
dives ................................................................. 24
diveStats ............................................................... 25
extractDive-methods .................................................. 27
plotDiveModel-methods ............................................... 28
plotTDR-methods ....................................................... 30
plotZOC-methods ....................................................... 33
readLocs ............................................................... 35
readTDR ................................................................. 36
rqPlot ................................................................. 38
sealLocs ............................................................... 39
TDR-accessors ......................................................... 40
TDR-class ............................................................... 41
TDRcalibrate-accessors ............................................. 42
TDRcalibrate-class ................................................... 45
timeBudget-methods ................................................... 46

Index 48

---

diveMove-package  Dive Analysis and Calibration

---
Examples

```r
## read in data and create a TDR object
zz <- system.file(file.path("data", "dives.csv"),
      package="diveMove", mustWork=TRUE)
(sealX <- readTDR(zz, speed=TRUE, sep=";", na.strings="", as.is=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)
## Use the "offset" ZOC method to zero-offset correct depth at 3 m
(dcalib <- calibrateDepth(sealX, zoc.method="offset", offset=3))

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, surface=FALSE)
  ## plot the first 300 dives, showing dive phases and surface readings
  plotTDR(dcalib, diveNo=seq(300), 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
## Identify which phase each dive belongs to
stamps <- stampDive(vcalib)
sumtab <- data.frame(stamps, dives)
head(sumtab)
```

### austFilter

**Filter satellite locations**

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

#### Usage

```r
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**: numeric scalar: speed threshold (m/s) above which filter tests should fail any given point.
- **dist.thr**: numeric scalar: distance threshold (km) above which the last filter test should fail any given point.
- **window**: integer: 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 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 preceding/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


See Also

distSpeed

Examples

```r
## Using the example from '?readLocs':
utils::example("readLocs", package="diveMove",
ask=FALSE, echo=FALSE)
ringy <- subset(locs, id == "ringy" & !is.na(lon) & !is.na(lat))

## Examples below use default Meeus algorithm for computing distances.
## See ?distSpeed for specifying other methods.
## 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")
```
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
```r
## S4 method for signature 'nls'
plotBouts(fit, ...)  
## S4 method for signature 'mle'
plotBouts(fit, x, ...)
```

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")
```
## S4 method for signature 'nls'
bec3(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- or 3-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.
- **bec3** signature(fit="nls"): Extract the estimated bout ending criterion from a fitted 3-process model.

### Author(s)

Sebastian P. Luque <spluque@gmail.com>

### References


### See Also

`bouts.mle, bouts2.nls, bouts3.nls` for examples.
bouj-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.

**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`: numeric scalar: bin width for the histogram.
- `method, bec.method`: character: 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.
- `...`: For `boutfreqs`, arguments passed to `hist` (must exclude `breaks` and `include.lowest`); for `boutinit`, arguments passed to `plot` (must exclude `type`).
- `lnfreq`: `data.frame` with components `lnfreq` (log frequencies) and corresponding `x` (mid points of histogram bins).
- `x.break`: vector of length 1 or 2 with `x` value(s) defining the break(s) point(s) for broken stick model, such that `x < x.break[1]` is 1st process, and `x >= x.break[1] & x < x.break[2]` is 2nd one, and `x >= x.break[2]` is 3rd 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`: numeric vector of proportions (0-1) to transform to the logit scale.
- `logit`: numeric scalar: 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 chosen value(s) to separate the two or three processes.

labelBouts labels each element (or row, if a matrix) of x with a sequential number, identifying which bout the reading belongs to. The bec argument needs to have the same dimensions as x to allow for situations where bec within x.

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 (histogram of input data) is produced as a side effect if argument plot is TRUE. See the Details section.

boutinit returns a list with as many elements as the number of processes implied by x.Nbreak (i.e. length(x.break) + 1). Each element is a vector of length two, corresponding to a and lambda, 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


See Also

bouts2.nls, bouts.mle. These include an example for labelBouts.
Examples

```r
## Using the Example from 'diveStats':
utils::example("diveStats", package="diveMove",
               ask=FALSE, echo=FALSE)
postdives <- tdrX.tab$dpostdive.dur[tdrX.tab$phase.no == 2]
## Remove isolated dives
postdives <- postdives[postdives > 2000]
lnfreq <- boutfreqs(postdives, bw=0.1, method="seq.diff", plot=FALSE)
boutinit(lnfreq, 50)

## See ?bouts.mle for labelBouts() example
```

---

### 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**

```r
bouts2.mleFUN(x, p, lambda1, lambda2)
bouts2.ll(x)
ll2.LL(x)
bouts.mle(ll.fun, start, x, ...)
bouts.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`: numeric; 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` (must exclude `xaxs, yaxs`). For `plotBouts2.nls`, arguments passed to `plot` (must exclude `type`).
- `fit`: `mle` object.
- `xlab, ylab`: character; 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.

`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 \( \lambda_1 \) and \( \lambda_2 \). Function `bouts2.L1` 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.L1` 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.L1`, useful lower bounds are \( \text{rep}(1e-08, 3) \). 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.

The lambdas can be very small for some data, particularly \( \lambda_2 \), so the default `ndeps` in `optim` can be so large as to push the search outside the bounds given. To avoid this problem, provide a smaller `ndeps` value.
Author(s)
Sebastian P. Luque <spluque@gmail.com>

References

See Also
mle, optim, logit, unlogit for transforming and fitting a reparameterized model.

Examples

```r
## Using the Example from 'diveStats'
utils::example("diveStats", package="diveMove", ask=FALSE, echo=FALSE)

postdives <- tdrX.tab$postdive.dur[tdrX.tab$phase.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[[1]]"a" / (startval[[1]]"a" + startval[[2]]"a")

## Fit the reparameterized (transformed parameters) model
## Drop names by wrapping around as.vector()
init.parms <- list(p=as.vector(logit(p)),
  lambda1=as.vector(log(startval[[1]]"lambda"))),
  lambda2=as.vector(log(startval[[2]]"lambda")))
bout.fit1 <- bouts.mle(bouts2.MM, 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.MM, 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

bec <- bec2(bout.fit2)

### Label bouts

labelBouts(postdives, rep(bec, length(postdives)),
            bec.method="seq.diff")

---

**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**

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

**Arguments**

- **x**: numeric vector with values to model.
- **a1, lambda1, a2, lambda2**: numeric: 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

63-69.

See Also

bouts.mle for a better approach; boutfreqs; boutinit

Examples

```r
## Using the Example from 'diveStats':
utils::example("diveStats", package="diveMove",
               ask=FALSE, echo=FALSE)
## Postdive durations
postdives <- tdrX.tab$postdive.dur[tdrX.tab$phase.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)
## Drop names by wrapping around as.vector()
startval.1 <- list(a1=as.vector(startval[[1]]"a"),
                   lambda1=as.vector(startval[[1]]"lambda"),
                   a2=as.vector(startval[[2]]"a"),
                   lambda2=as.vector(startval[[2]]"lambda"))

## Fit the 2 process model
bout.fit <- bouts2.nls(lnfreq, start=startval.1, maxiter=500)
summary(bout.fit)
plotBouts(bout.fit)

## Estimated BEC
bec2(bout.fit)
```
bouts3NLS

Fit mixture of 3 Poisson Processes to Log Frequency data

Description

Functions to model a mixture of 3 random Poisson processes to histogram-like data of log frequency vs interval mid points. This follows Sibly et al. (1990) method, adapted for a three-process model by Berdoy (1993).

Usage

bouts3.nlsFUN(x, a1, lambda1, a2, lambda2, a3, lambda3)
bouts3.nls(lnfreq, start, maxiter)
bouts3.nlsBEC(fit)
plotBouts3.nls(fit, lnfreq, bec.lty, ...)

Arguments

x numeric vector with values to model.
a1, lambda1, a2, lambda2, a3, lambda3 numeric: 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

bouts3.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 bouts3.nls.
bouts3.nls fits the nonlinear least-squares model itself.
bouts3.nlsBEC calculates the BEC from a list object, as the one that is returned by nls, representing a fit of the model. plotBouts3.nls plots such an object.

Value

bouts3.nlsFUN returns a numeric vector evaluating the mixture of 3 Poisson process.
bouts3.nls returns an nls object resulting from fitting this model to data.
bouts3.nlsBEC returns a number corresponding to the bout ending criterion derived from the model.
plotBouts3.nls plots the fitted model with the corresponding data.
Author(s)

Sebastian P. Luque <spluque@gmail.com>

References


See Also

bouts.mle for a better approach; boutfreqs; boutinit

Examples

## Using the Example from 'diveStats'
utils::example("diveStats", package="diveMove", ask=FALSE, echo=FALSE)

## Postdive durations
postdives <- tdrX.tab$postdive.dur
postdives.diff <- abs(diff(postdives))

## Remove isolated dives
postdives.diff <- postdives.diff[postdives.diff < 4000]

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

startval <- boutinit(lnfreq, c(50, 400))

## Drop names by wrapping around as vector()
startval.1 <- list(a1=as.vector(startval[[1]]"a"),
lambda1=as.vector(startval[[1]]"lambda"),
a2=as.vector(startval[[2]]"a"),
lambda2=as.vector(startval[[2]]"lambda"),
a3=as.vector(startval[[3]]"a"),
lambda3=as.vector(startval[[3]]"lambda"))

## Fit the 3 process model
bout.fit <- bouts3.nls(lnfreq, start=startval.1, maxiter=500)
summary(bout.fit)
plotBouts(bout.fit)

## Estimated BEC
bec3(bout.fit)
**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**

```r
calibrateDepth(x, dry.thr=70, wet.cond, wet.thr=3610, dive.thr=4,  
zoc.method=c("visual", "offset", "filter"), ...,  
interp.wet=FALSE, smooth.par=0.1, knot.factor=3,  
descent.crit.q=0, ascent.crit.q=0)
```

**Arguments**

- `x`: An object of class TDR for calibrateDepth or an object of class TDRcalibrate for calibratespeed.
- `dry.thr`: numeric: dry error threshold in seconds. Dry phases shorter than this threshold will be considered as wet.
- `wet.cond`: logical: indicates which observations should be considered wet. If it is not provided, records with non-missing depth are assumed to correspond to wet conditions (see ‘Details’ and ‘Note’ below).
- `wet.thr`: numeric: wet threshold in seconds. At-sea phases shorter than this threshold will be considered as trivial wet.
- `dive.thr`: numeric: threshold depth below which an underwater phase should be considered a dive.
- `zoc.method`: character string to indicate the method to use for zero offset correction. One of "visual", "offset", or "filter" (see ‘Details’).
- `...`: Arguments required for ZOC methods filter (k, probs, depth.bounds (defaults to range), na.rm (defaults to TRUE)) and offset (offset).
- `interp.wet`: logical: if TRUE (default is FALSE), then an interpolating spline function is used to impute NA depths in wet periods (after ZOC). Use with caution: it may only be useful in cases where the missing data pattern in wet periods is restricted to shallow depths near the beginning and end of dives. This pattern is common in some satellite-linked TDRs.
- `smooth.par`: numeric scalar representing amount of smoothing (argument spar in smooth.spline. If it is NULL, then the smoothing parameter is determined by Generalized Cross-validation (GCV).
- `knot.factor`: numeric scalar that multiplies the number of samples in the dive. This is used to construct the time predictor for the derivative.
- `descent.crit.q`: numeric: critical quantile of rates of descent below which descent is deemed to have ended.
- `ascent.crit.q`: numeric: critical quantile of rates of ascent above which ascent is deemed to have started.
**Details**

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

The procedure starts by zero-offset correcting depth (see ‘ZOC’ below), and then a factor is created with value “L” (dry) for rows with NAs for depth and value “W” (wet) otherwise. This assumes that TDRs were programmed to turn off recording of depth when instrument is dry (typically by means of a salt-water switch). If this assumption cannot be made for any reason, then a logical vector as long as the time series should be supplied as argument `wetNcond` to indicate which observations should be considered wet. This argument is directly analogous to the `subset` argument in `subset.data.frame`, so it can refer to any variable in the TDR object (see ‘Note’ section below).

The duration of each of these phases of activity is subsequently calculated. If the duration of a dry phase (“L”) is less than `dryNthr`, 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 `wetNthr`, 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.

Some instruments produce a peculiar pattern of missing data near the surface, at the beginning and/or end of dives. The argument `interp.wet` may help to rectify this problem by using an interpolating spline function to impute the missing data, constraining the result to a minimum depth of zero. Please note that this optional step is performed after ZOC and before identifying dives, so that interpolation is performed through dry phases coded as wet because their duration was briefer than `dryNthr`. Therefore, `dryNthr` must be chosen carefully to avoid interpolation through legitimate dry periods.

The next step is to detect dives whenever the zero-offset corrected depth in an underwater phase is below the specified 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 identified using the descent, ascent and wiggle criteria (see ‘Detection of dive phases’ below). This procedure generates a factor with levels “D”, “DB”, “B”, “BA”, “DA”, “A”, and “X”, breaking the input into descent, descent/bottom, bottom, bottom/ascent, ascent, descent/ascent (occurring when no bottom phase can be detected) and non-dive (surface), respectively.

**Value**

An object of class `TDRcalibrate`.

**ZOC**

This procedure is required to correct drifts in the pressure transducer of TDR records and noise in depth measurements. Three methods are available to perform this correction.

Method “visual” calls `plotTDR`, which plots depth and, optionally, speed vs. time with the ability of zooming in and out on time, changing maximum depths displayed, and panning through time. The button to zero-offset correct sections of the record allows for the collection of ‘x’ and ‘y’ coordinates for two points, obtained by clicking on the plot region. The first point clicked represents the offset and beginning time of section to correct, and the second one represents the ending time of the section to correct. Multiple sections of the record can be corrected in this manner, by panning through the
calibrateDepth

Method “offset” can be used when the offset is known in advance, and this value is used to correct the entire time series. Therefore, offset=0 specifies no correction.

Method “filter” implements a smoothing/filtering mechanism where running quantiles can be applied to depth measurements in a recursive manner (Luque and Fried 2011), using \texttt{depth.filter}. It relies on function \texttt{runquantile} from the \texttt{caTools} package. The method calculates the first running quantile defined by \texttt{probs[1]} on a moving window of size \texttt{k[1]}. The next running quantile, defined by \texttt{probs[2]} and \texttt{k[2]}, is applied to the smoothed/filtered depth measurements from the previous step, and so on. The corrected depth measurements (d) are calculated as:

\[ d = d_0 - d_n \]

where \( d_0 \) is original depth and \( d_n \) is the last smoothed/filtered depth. This method is under development, but reasonable results can be achieved by applying two filters (see ‘Examples’). The default \texttt{na.rm=TRUE} works well when there are no level shifts between non-NA phases in the data, but \texttt{na.rm=FALSE} is better in the presence of such shifts. In other words, there is no reason to pollute the moving window with NAs when non-NA phases can be regarded as a continuum, so splicing non-NA phases makes sense. Conversely, if there are level shifts between non-NA phases, then it is better to retain NA phases to help the algorithm recognize the shifts while sliding the window(s).

The search for the surface can be limited to specified bounds during smoothing/filtering, so that observations outside these bounds are interpolated using the bounded smoothed/filtered series.

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

\section*{Detection of dive phases}

The process for each dive begins by taking all observations below the dive detection threshold, and setting the beginning and end depths to zero, at time steps prior to the first and after the last, respectively. The latter ensures that descent and ascent derivatives are non-negative and non-positive, respectively, so that the end and beginning of these phases are not truncated. A smoothing spline is used to model the dive and its derivative to investigate its changes in vertical rate. This method requires at least 4 observations (see \texttt{smooth.spline}), so the time series is linearly interpolated at equally spaced time steps if this limit is not achieved in the current dive. Wiggles at the beginning and end of the dive are assumed to be zero offset correction errors, so depth observations at these extremes are interpolated between zero and the next observations when this occurs.

A set of knots is established to fit the smoothing spline by using a regular time sequence with beginning and end equal to the extremes of the input sequence, and with length equal to \( N \times \texttt{knot.factor} \). The first derivative of the spline is evaluated at the same set of knots to calculate the vertical rate throughout the dive and determine the end of descent and beginning of ascent. Equivalent procedures are used for detecting descent and ascent phases.

The quantile corresponding to \( \texttt{descent.crit.q} \) of all the positive derivatives (rate of descent) at the beginning of the dive is used as threshold for determining the end of descent. Descent is deemed to have ended at the \textit{first} minimum derivative, and the nearest input time observation is considered to indicate the end of descent. The sign of the comparisons is reversed for detecting the ascent. If observed depth to the left and right of the derivative defining the ascent are the same, the right takes precedence.
The particular dive phase categories are subsequently defined using simple set operations.

**Note**

Note that the condition implied with argument `wet.cond` is evaluated after the ZOC procedure, so it can refer to corrected depth. In many cases, not all variables in the TDR object are sampled with the same frequency, so they may need to be interpolated before using them for this purpose. Note also that any of these variables may contain similar problems as those dealt with during ZOC, so programming instruments to record depth only when wet is likely the best way to ensure proper detection of wet/dry conditions.

**Author(s)**

Sebastian P. Luque <spluque@gmail.com>

**References**


**See Also**

`tdrCalibrate`, `.zoc`, `.depthFilter`, `.detPhase`, `.detDive`, `plotTDR`, and `plotZOC` to visually assess ZOC procedure.

**Examples**

data(divesTDR)
divesTDR

```
# Consider a 3 m offset, a dive threshold of 3 m, the 1% quantile for
# critical vertical rates, and a set of knots 20 times as long as the
# observed time steps.
(dcalib <- calibrateDepth(divesTDR, dive.thr=3, zoc.method="offset",
  offset=3, descent.crit.q=0.01, ascent.crit.q=0,
  knot.factor=20))
```

# Or ZOC algorithmically with method="filter":
# Not run: # This can take a while due to large window needed for 2nd
# filter in this dataset
(dcalib <- calibrateDepth(divesTDR, dive.thr=3, zoc.method="filter",
  k=c(3, 5760), probs=c(0.5, 0.02), na.rm=TRUE,
  descent.crit.q=0.01, ascent.crit.q=0,
  knot.factor=20))

```
# End(Not run)
```
**calibrateSpeed**

*Calibrate and build a "TDRcalibrate" object*

**Description**

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

**Usage**

```r
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`: numeric scalar: quantile on which to regress speed on rate of depth change; passed to `rq`.
- `contour.level`: numeric scalar: 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`: numeric scalar: only changes in depth larger than this value will be used for calibration.
- `bad`: numeric vector of length 2 indicating that only rates of depth change and speed greater than the given value should be used for calibration, respectively.
- `coefs`: numeric: 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: whether to plot the results.
- `postscript`: logical: 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

See Also
TDRcalibrate

Examples

```r
## Continuing the Example from '?calibrateDepth':
utils::example("calibrateDepth", package="diveMove",
  ask=FALSE, echo=FALSE)
dcalib # the 'TDRcalibrate' that was created

## Calibrate speed using only changes in depth > 2 m
vcalib <- calibrateSpeed(dcalib, 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, method=c("Meeus", "VincentyEllipsoid"))

Arguments

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>pt1</td>
<td>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.</td>
</tr>
<tr>
<td>pt2</td>
<td>A matrix with the same size and structure as pt1.</td>
</tr>
<tr>
<td>method</td>
<td>character indicating which of the distance algorithms from geosphere-package to use (only default parameters used). Only Meeus and VincentyEllipsoid are supported for now.</td>
</tr>
</tbody>
</table>

Value
A matrix with three columns: distance (km), time difference (s), and speed (m/s).
```
diveModel-class

Author(s)
Sebastian P. Luque <spluque@gmail.com>

Examples

```r
## Using the Example from '?readLocs':
utils::example("readLocs", package="diveMove",
               ask=FALSE, echo=FALSE)

## 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))

## Default Meeus method
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
pts <- seq(10)
(meeus <- distSpeed(locs[pts, 3:5], locs[pts + 1, 3:5]))
(vincenty <- distSpeed(locs[pts, 3:5],
                       locs[pts + 1, 3:5],
                       method="VincentyEllipsoid"))
meeus - vincenty
```

diveModel-class  

Class "diveModel" for representing a model for identifying dive phases

Description
Details of model used to identify the different phases of a dive.

Objects from the Class
Objects can be created by calls of the form `new("diveModel", ...)`. ‘diveModel’ objects contain all relevant details of the process to identify phases of a dive. Objects of this class are typically generated during depth calibration, using `calibrateDepth`, more specifically `.cutDive`. 
**Slots**

- **label.matrix**: Object of class "matrix". A 2-column character matrix with row numbers matching each observation to the full TDR object, and a vector labelling the phases of each dive.
- **dive.spline**: Object of class "smooth.spline". Details of cubic smoothing spline fit (see `smooth.spline`).
- **spline.deriv**: Object of class "list". A list with the first derivative of the smoothing spline (see `predict.smooth.spline`).
- **descent.crit**: Object of class "numeric". The index of the observation at which the descent was deemed to have ended (from initial surface observation).
- **ascent.crit**: Object of class "numeric". The index of the observation at which the ascent was deemed to have ended (from initial surface observation).
- **descent.crit.rate**: Object of class "numeric". The rate of descent corresponding to the critical quantile used.
- **ascent.crit.rate**: Object of class "numeric". The rate of ascent corresponding to the critical quantile used.

**Author(s)**

Sebastian P. Luque <spluque@gmail.com>

**See Also**

`getDiveDeriv`, `plotDiveModel`

**Examples**

```r
showClass("diveModel")
```

---

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` is an example TDR object.

**Format**

Bzip2-compressed file. A comma separated value (csv) file with 69560 TDR readings, measured at 5 s intervals, with the following columns:

- **date** Date
- **time** Time
- **depth** Depth in m
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.

Source

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

See Also

readTDR, diveStats.

Examples

zz <- system.file(file.path("data", "dives.csv"),
                  package="diveMove", mustWork=TRUE)
str(read.csv(zz, sep=";", na.strings=""))

diveStats

Per-dive statistics

Description

Calculate dive statistics in TDR records.

Usage

diveStats(x, depth.deriv=TRUE)
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.

depth.deriv logical: should depth derivative statistics be calculated?
interval numeric scalar: sampling interval for interpreting x.
speed logical: should speed statistics be calculated?
ignoreZ logical: whether phases 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 phase 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.
divetim Dive duration.
descdist Numeric vector with last 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 first ascent depth.
bottdep.mean Mean bottom depth.
bottdep.median Median bottom depth.
bottdep.sd Standard deviation of bottom depths.
maxdep Numeric vector with maximum 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.
extractDive-methods

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.
postdive.dur
   Postdive duration.
postdive.tdist
   Numeric vector with postdive total distance, estimated from speed.
postdive.mean.speed
   Numeric vector with postdive mean speed.

If depth.deriv=TRUE, 21 additional columns with the minimum, first quartile, median, mean, third quartile, maximum, and standard deviation of the depth derivative for each phase of the dive. The number of columns also depends on argument speed.

stampDive returns a data.frame with phase number, activity, and start and end times for each dive.

Author(s)

Sebastian P. Luque <spluque@gmail.com>

See Also

calibrateDepth, .detPhase, TDRcalibrate-class

Examples

## Continuing the Example from '?calibrateDepth':
util::example("calibrateDepth", package="diveMove", ask=FALSE, echo=FALSE)
dcalib # the 'TDRcalibrate' that was created
tdrX <- diveStats(dcalib)
stamps <- stampDive(dcalib, ignoreZ=TRUE)
tdrX.tab <- data.frame(stamps, tdr)
summary(tdrX.tab)
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. Duplicates are ignored.
- **id**: numeric vector or scalar 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

```r
## Continuing the Example from '?calibrateDepth':
utils::example("calibrateDepth", package="diveMove",
             ask=FALSE, echo=FALSE)
dcalib # the 'TDRcalibrate' that was created
diveX <- extractDive(divesTDR, 9, getDAct(dcalib, "dive.id"))
plotTDR(diveX, interact=FALSE)
diveX <- extractDive(dcalib, 5:10)
plotTDR(diveX, interact=FALSE)
```

plotDiveModel-methods

Methods for plotting models of dive phases

Description

Methods for function plotDiveModel.
Usage

```r
## S4 method for signature 'diveModel,missing'
plotDiveModel(x, diveNo)

## S4 method for signature 'numeric,numeric'
plotDiveModel(x, y, times.s, depths.s, d.crit, a.crit,
               diveNo=1, times.deriv, depths.deriv,
               d.crit.rate, a.crit.rate)

## S4 method for signature 'TDRcalibrate,missing'
plotDiveModel(x, diveNo)
```

Arguments

- `x`: A `diveModel` (diveMode,missing method), `numeric` vector of time step observations (numeric,numeric method), or `TDRcalibrate` object (TDRcalibrate,numeric method).
- `diveNo`: integer representing the dive number selected for plotting.
- `y`: numeric vector with depth observations at each time step.
- `times.s`: numeric vector with time steps used to generate the smoothing spline (i.e. the knots, see `diveModel`).
- `depths.s`: numeric vector with smoothed depth (see `diveModel`).
- `d.crit`: integer denoting the index where descent ends in the observed time series (see `diveModel`).
- `a.crit`: integer denoting the index where ascent begins in the observed time series (see `diveModel`).
- `times.deriv`: numeric vector representing the time steps where the derivative of the smoothing spline was evaluated (see `diveModel`).
- `depths.deriv`: numeric vector representing the derivative of the smoothing spline evaluated at `times.deriv` (see `diveModel`).
- `d.crit.rate`: numeric scalar: vertical rate of descent corresponding to the quantile used (see `diveModel`).
- `a.crit.rate`: numeric scalar: vertical rate of ascent corresponding to the quantile used (see `diveModel`).

Methods

All methods produce a double panel plot. The top panel shows the depth against time, the cubic spline smoother, the identified descent and ascent phases (which form the basis for identifying the rest of the dive phases), while the bottom panel shows the first derivative of the smooth trace.

signature(x = "diveModel", y = "missing") Given a `diveModel` object and (possibly) the dive number that it corresponds to, the plot shows the model data.

signature(x = "numeric", y = "numeric") This is the main method, which requires all aspects of the model to be provided.
signature(x = "TDRcalibrate", y = "missing") Given a TDRcalibrate object and a dive number to extract from it, this method plots the observed data and the model. The intended use of this method is through plotTDR when what="dive.model".

Author(s)
Sebastian P. Luque <spluque@gmail.com>

See Also
diveModel

Examples

```r
## Continuing the Example from '?calibrateDepth':
utils::example("calibrateDepth", package="diveMove", 
               ask=FALSE, echo=FALSE)

## 'diveModel' method
dm <- getDiveModel(dcalib, 100)
plotDiveModel(dm, diveNo=100)

## 'TDRcalibrate' method
plotDiveModel(dcalib, diveNo=100)
```

Description

Main plotting method for objects of these classes. Plot and optionally set zero-offset correction windows 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

```r
## S4 method for signature 'POSIXt,numeric'
plotTDR(x, y, 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",
        dry.time=NULL, phase.factor=NULL, plot.points=FALSE,
        interact=TRUE, key=TRUE, cex.pts=0.4, ...)  

## S4 method for signature 'TDR,missing'
plotTDR(x, y, concurVars, concurVarTitles, ...)
```

Methods for plotting objects of class "TDR" and "TDRcalibrate"
## S4 method for signature 'TDRcalibrate,missing'

```r
plotTDR(x, y, what="phases", "dive.model"),
diveNo=seq(max(getDAct(x, "dive.id")), ...)
```

### Arguments

- **x**  
  POSIXct object with date and time, `TDR`, or `TDRcalibrate` object.

- **y**  
  numeric vector with depth in m.

- **concurVars**  
  matrix with additional variables in each column to plot concurrently with depth. For the `(TDR,missing)` and `(TDRcalibrate,missing)` methods, a character vector naming additional variables from the `concurrentData` slot to plot, if any.

- **xlim**  
  POSIXct or numeric vector of length 2, with lower and upper limits of time to be plotted.

- **depthLim**  
  numeric vector of length 2, with the lower and upper limits of depth to be plotted.

- **xlab, ylab, depth**  
  character strings to label the corresponding y-axes.

- **concurVarTitles**  
  character vector of titles to label each new variable given in `concurVars`.

- **xlab.format**  
  character: 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.

- **dry.time**  
  subset of time corresponding to observations considered to be dry.

- **phase.factor**  
  factor dividing rows into sections.

- **plot.points**  
  logical: whether to plot points.

- **interact**  
  logical: whether to provide interactive tcltk controls and access to the associated ZOC functionality.

- **key**  
  logical: whether to draw a key.

- **cex.pts**  
  Passed to `points` to set the relative size of points to plot (if any).

- **...**  
  For the `(POSIXt,numeric)` method, arguments passed to `par` for all methods; 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. For other methods, except `(TDRcalibrate,missing)`, these can be any of the arguments for the `(POSIXt,numeric)` method. For `(TDRcalibrate,missing)`, these are arguments for the appropriate methods.

- **diveNo**  
  numeric vector or scalar with dive numbers to plot.

- **what**  
  character: what aspect of the `TDRcalibrate` to plot, which selects the method to use for plotting.
Details

This function is used primarily to correct drifts in the pressure transducer of TDR records and noise in depth measurements via method="visual" in `calibrateDepth`.

Value

If called with the `interact` argument set to TRUE, returns a list (invisibly) 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`.

Methods

plotTDR signature(x="TDR", y="numeric"): interactive graphical display of time-depth data, with zooming and panning capabilities.

plotTDR signature(x="TDR", y="missing"): As method above.

plotTDR signature(x="TDRcalibrate", y="missing"): plot selected aspects of TDRcalibrate object. Currently, two aspects have plotting methods:

- phases (Optional arguments: concurVars, surface) Plots all dives, labelled by the activity phase they belong to. It produces a plot consisting of one or more panels; the first panel shows depth against time, and additional panels show other concurrent data in the object. Optional argument concurVars is a character vector indicating which additional components from the concurrentData slot to plot, if any. Optional argument surface is a logical: whether to plot surface readings.
- dive.model Plots the dive model for the selected dive number (diveNo argument).

Author(s)

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

See Also

calibrateDepth, .zoc

Examples

```r
## Continuing the Example from '?calibrateDepth' :
utils::example("calibrateDepth", package="diveMove",
               ask=FALSE, echo=FALSE)
## Use interact=TRUE (default) to set an offset interactively
## Plot the 'TDR' object
plotTDR(getTime(divesTDR), getDepth(divesTDR), interact=FALSE)
plotTDR(divesTDR, interact=FALSE)

## Plot different aspects of the 'TDRcalibrate' object
plotTDR(dcalib, interact=FALSE)
plotTDR(dcalib, diveNo=19:25, interact=FALSE)
plotTDR(dcalib, what="dive.model", diveNo=25)
if (dev.interactive(orNone=TRUE)) {
```
Methods for visually assessing results of ZOC procedure

Description

Plots for comparing the zero-offset corrected depth from a `tdrCalibrate` object with the uncorrected data in a `tdr` object, or the progress in each of the filters during recursive filtering for ZOC (`calibrateDepth`).

Usage

```r
## S4 method for signature 'TDR,matrix'
plotZOC(x, y, xlim, ylim, ylab="Depth (m)", ...)  
## S4 method for signature 'TDR,TDRcalibrate'
plotZOC(x, y, xlim, ylim, ylab="Depth (m)", ...)
```

Arguments

- `x`: TDR object.
- `y`: matrix with the same number of rows as there are observations in `x`, or a `tdrCalibrate` object.
- `xlim`: POSIXct or numeric vector of length 2, with lower and upper limits of time to be plotted. Defaults to time range of input.
- `ylim`: numeric vector of length 2 (upper, lower) with axis limits. Defaults to range of input.
- `ylab`: character strings to label the corresponding y-axis.
- `...`: Arguments passed to `legend`.

Details

The `tdr,matrix` method produces a plot like those shown in Luque and Fried (2011). The `tdr,TDRcalibrate` method overlays the corrected depth from the second argument over that from the first.

Value

Nothing; a plot as side effect.
Methods

**plotTDR** signature(x="TDR", y="matrix"): This plot helps in finding appropriate parameters for `diveMove:::depthfilter`, and consists of three panels. The upper panel shows the original data, the middle panel shows the filters, and the last panel shows the corrected data. method="visual" in `calibrateDepth`.

**plotTDR** signature(x="TDR", y="TDRcalibrate"): This plots depth from the TDRcalibrate object over the one from the TDR object.

Author(s)

Sebastian P. Luque <spluque@gmail.com>

References


See Also

`calibrateDepth`, .zoc

Examples

```r
## Using the Example from '?diveStats'::

utils::example("diveStats", package="diveMove",
    ask=FALSE, echo=FALSE)

## Plot filters for ZOC
## Work on first phase (trip) subset, to save processing time, since
## there's no drift nor shifts between trips
tdr <- divesTDR[1:15000]
## Try window widths (K), quantiles (P) and bound the search (db)
K <- c(3, 360); P <- c(0.5, 0.02); db <- c(0, 5)
d.filter <- diveMove:::depthFilter(depth=getDepth(tdr),
    k=K, probs=P, depth.bounds=db,
    na.rm=TRUE)
old.par <- par(no.readonly=TRUE)
plotZOC(tdr, d.filter, ylim=c(0, 6))
par(old.par)

## Plot corrected and uncorrected depth, regardless of method
## Look at three different scales
xlim1 <- c(getTime(divesTDR)[7100], getTime(divesTDR)[11700])
xlim2 <- c(getTime(divesTDR)[7100], getTime(divesTDR)[7400])
xlim3 <- c(getTime(divesTDR)[7100], getTime(divesTDR)[7200])
par(mar=c(3, 4, 0, 1) + 0.1, cex=1.1, las=1)
layout(seq(3))
plotZOC(divesTDR, dcalib, xlim=xlim1, ylim=c(0, 6))
plotZOC(divesTDR, dcalib, xlim=xlim2, ylim=c(0, 70))
```
readLocs

Description

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

Usage

readLocs(locations, 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

locations character: a string indicating the path to the file to read, or a data.frame available in the search list. Provide the entire path if the file is not on the current directory. This can also be a text-mode connection, as allowed in read.csv.

loc.idCol integer: column number containing location ID. If missing, a loc.id column is generated with sequential integers as long as the input.

idCol integer: 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 integer: column number containing dates, and, optionally, times.

timeCol integer: column number containing times.

dtformat character: a string specifying the format in which the date and time columns, when pasted together, should be interpreted (see strptime) in file.

tz character: a string indicating the time zone for the date and time readings.

lonCol integer: column number containing longitude readings.

latCol integer: column number containing latitude readings.

classCol integer: column number containing the ARGOS rating for each location.

alt.lonCol integer: column number containing alternative longitude readings.

alt.latCol integer: 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

```r
## Do example to define object zz with location of dataset
utils::example("sealLocs", package="diveMove",
  ask=FALSE, echo=FALSE)
locs <- readLocs(zz, idCol=1, dateCol=2,
  dtformat="%Y-%m-%d %H:%M:%S", classCol=3,
  lonCol=4, latCol=5, sep=";")

summary(locs)
```

---

**readTDR**

*Read comma-delimited file with "TDR" data*

Description

Read a 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

```r
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(matrix(ncol=0, nrow=length(time)),
    speed=FALSE, dtime, file)
```

Arguments

- **file**: character: a string indicating the path to the file to read. This can also be a text-mode connection, as allowed in `read.csv`.
- **dateCol**: integer: column number containing dates, and optionally, times.
- **timeCol**: integer: column number with times.
- **depthCol**: integer: column number containing depth readings.
- **speed**: logical: whether speed is included in one of the columns of concurrentCols.
subsamp numeric scalar: subsample rows in file with subsamp interval, in s.
concurrentCols integer vector of column numbers to include as concurrent data collected.
dtformat character: a string specifying the format in which the date and time columns, when pasted together, should be interpreted (see \texttt{strptime}).
tz character: a string indicating the time zone assumed for the date and time readings.

data frame with additional, concurrent data collected.
dtime numeric scalar: 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 \texttt{speed} is \texttt{TRUE} and \texttt{concurrentCols} contains a column named speed or velocity, then an object of class \texttt{TDRspeed} is created, where speed is considered to be the column matching this name.

Value

An object of class \texttt{TDR} or \texttt{TDRspeed}.

Note

Although \texttt{TDR} and \texttt{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 \texttt{R} using tools like GNU awk or GNU sed, so \texttt{diveMove} does not currently attempt to fix these errors.

Author(s)

Sebastian P. Luque <spluque@gmail.com>

Examples

```r
## Do example to define object zz with location of dataset
utils::example("dives", package="diveMove", ask=FALSE, echo=FALSE)
srcfn <- basename(zz)
readTDR(zz, speed=TRUE, sep=";", na.strings="", as.is=TRUE)

## Or more pedestrian
```
`rqPlot`  

Plot of quantile regression for speed calibrations

**Description**

Plot of quantile regression for assessing quality of speed calibrations

**Usage**

```r
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` numeric vector with speed in m/s.
- `rddepth` numeric vector with rate of depth change.
- `z` 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` character: string with title prefix to include in output plot.
- `xlab`, `ylab` character vectors with 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` numeric: value specifying the amount by which to enlarge the size of points.

**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 represents 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**

Bzip2-compressed file. 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](http://whale.wheelock.edu/Welcome.html).

See Also

readLocs, distSpeed.

**Examples**

```r
zz <- system.file(file.path("data", "sealLocs.csv"),
                   package="diveMove", mustWork=TRUE)
str(read.csv(zz, sep=";"))
```
Description

Basic methods for manipulating objects of class TDR.

Show Method

`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 subbed 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`.

`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`, `plotTDR`. 
Examples

data(divesTDR)

## Retrieve the name of the source file
getFileName(divesTDR)

## Retrieve concurrent temperature measurements
temp <- getCCData(divesTDR, "temperature"); head(temp)
temp <- getCCData(divesTDR); head(temp)

## 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

Show and extract information from TDRcalibrate objects.

Usage

```r
## 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'
getDiveModel(x)
## S4 method for signature 'TDRcalibrate,numeric'
getDiveModel(x, diveNo)
## S4 method for signature 'diveModel'
getDiveDeriv(x, phase=c("all", "descent", "bottom", "ascent"))
## S4 method for signature 'TDRcalibrate'
getDiveDeriv(x, diveNo, phase=c("all", "descent", "bottom", "ascent"))
## S4 method for signature 'TDRcalibrate,missing'
getGAct(x)
## S4 method for signature 'TDRcalibrate,character'
getGAct(x, y)
## S4 method for signature 'TDRcalibrate'
getSpeedCoeff(x)
## S4 method for signature 'TDRcalibrate'
getTDR(x)
```
TDRcalibrate-accessors

Arguments

- **x**: TDRcalibrate object.
- **diveNo**: numeric vector with dive numbers to extract information from.
- **y**: string; “dive.id”, “dive.activity”, or “postdive.id” in the case of getDAct, to extract the numeric dive ID, the factor identifying activity phases (with underwater and diving levels possibly represented), 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 (where diving and underwater levels are not represented), or the beginning and end times of each phase in the record, respectively.
- **phase**: character vector indicating phase of the dive for which to extract the derivative.

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.
- **show** signature(object="diveModel"): prints an informative summary of a dive model.

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.
- **getDiveModel** signature(x="TDRcalibrate", diveNo="missing"): extracts a list with all dive phase models. This accesses the dive.models slot of TDRcalibrate objects.
- **getDiveModel** signature(x="TDRcalibrate", diveNo="numeric"): as the method for missing diveNo, but selects data from a particular dive number to extract.
- **getDiveDeriv** signature(x="TDRcalibrate"): extracts the derivative (list) of the dive model (smoothing spline) from the dive.models slot of TDRcalibrate objects for one or all phases of a dive.
- **getDiveDeriv** signature(x="diveModel"): as the method for TDRcalibrate, but selects data from one or all phases of a dive.
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>

See Also
diveModel, plotDiveModel, plotTDR.

Examples

```r
## Continuing the Example from '?calibratedepth'
utils::example("calibratedepth", package="diveMove",
   ask=FALSE, echo=FALSE)
dcalib # the 'TDRcalibrate' that was created

## Beginning times of each successive phase in record
getGAct(dcalib, "begin")

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

## Factor of dive phases for given dive
getDPhaseLab(dcalib, 19)
## Full dive model
(dm <- getDivemodel(dcalib, 19))
str(dm)

## Derivatives
getDiveDeriv(dcalib, diveNo=19)
(derivs.desc <- getDiveDeriv(dcalib, diveNo=19, phase="descent"))
(derivs.bott <- getDiveDeriv(dcalib, diveNo=19, phase="bottom"))
(derivs.asc <- getDiveDeriv(dcalib, diveNo=19, phase="ascent"))
if (require(lattice)) {
  fl <- c("descent", "bottom", "ascent")
  bwplot(~ derivs.desc+y + derivs.bott+y + derivs.asc+y,
         outer=TRUE, allow.multiple=TRUE, layout=c(1, 3),
         xlab=expression(paste("Vertical rate (", m %.% s^{-1}, ", "))),
         strip=strip.custom(factor.levels=fl))
}
```
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

call: Object of class call.
   The matched call to the function that created the object.

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’. 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 the “phase.labels” component of the list returned by \texttt{.labDivePhase}.

dive.models: Object of class ‘list’. This slot contains the details of the process of dive phase identification for each dive. It has the same form as the \texttt{dive.models} component of the list returned by \texttt{.labDivePhase}. It has as many components as there are dives in the TDR object, each of them of class \texttt{diveModel}.

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 \texttt{c(0, 1)} meaning uncalibrated speeds.

Author(s)
Sebastian P. Luque <spluque@gmail.com>

See Also

\texttt{tdr} for links to other classes in the package. \texttt{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

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

Arguments

- \texttt{obj} \texttt{TDRcalibrate} object.
- \texttt{ignoreZ} logical: 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` `POSIXct` objects indicating the beginning and end of each period.

Author(s)

Sebastian P. Luque <spluque@gmail.com>

See Also

calibrateDepth

Examples

```r
## Continuing the Example from '?calibrateDepth':
utils::example("calibrateDepth", package="diveMove", 
               ask=FALSE, echo=FALSE)
dcalib # the 'TDRcalibrate' that was created
timeBudget(dcalib, TRUE)
```
Index

*Topic **arith**
  diveStats, 25
  rqPlot, 38

*Topic **classes**
  diveModel-class, 23
  TDR-class, 41
  TDRcalibrate-class, 45

*Topic **datasets**
  dives, 24
  sealLocs, 39

*Topic **hplot**
  rqPlot, 38

*Topic **iteration**
  austFilter, 3

*Topic **manip**
  austFilter, 3
  bout-misc, 8
  bouts2MLE, 10
  bouts2NLS, 13
  bouts3NLS, 15
  calibrateDepth, 17
  calibrateSpeed, 21
  distSpeed, 22
  readLocs, 35
  readTDR, 36
  rqPlot, 38

*Topic **math**
  calibrateDepth, 17
  calibrateSpeed, 21
  distSpeed, 22
  diveStats, 25

*Topic **methods**
  bout-methods, 6
  extractDive-methods, 27
  plotDiveModel-methods, 28
  plotTDR-methods, 30
  plotZOC-methods, 33
  TDR-accessors, 40
  TDRcalibrate-accessors, 42
  timeBudget-methods, 46

*Topic **misc**
  bout-misc, 8

*Topic **models**
  bouts2MLE, 10
  bouts2NLS, 13
  bouts3NLS, 15

*Topic **package**
  diveMove-package, 2
  .cutDive, 23
  .depthFilter, 20
  .detDive, 20, 45
  .detPhase, 20, 27, 45
  .labDivePhase, 46
  .zoc, 20, 32, 34
  [,TDR-method (TDR-accessors), 40
  as.data.frame,TDR-method (TDR-accessors), 40
  as.TDRspeed (TDR-accessors), 40
  as.TDRspeed,TDR-method (TDR-accessors), 40
  austFilter, 3
  bec2 (bout-methods), 6
  bec2,mle-method (bout-methods), 6
  bec2,nls-method (bout-methods), 6
  bec3 (bout-methods), 6
  bec3,nls-method (bout-methods), 6
  bout-methods, 6
  bout-misc, 8
  boutfreqs, 14, 16
  boutfreqs (bout-misc), 8
  boutinit, 11, 14, 16
  boutinit (bout-misc), 8
  bouts.mle, 7, 9, 14, 16
  bouts.mle (bouts2MLE), 10
INDEX

bouts2.LL, 11
bouts2.LL (bouts2MLE), 10
bouts2.11, 11
bouts2.11 (bouts2MLE), 10
bouts2.mleBEC (bouts2MLE), 10
bouts2.mleFUN (bouts2MLE), 10
bouts2.nls, 7, 9
bouts2.nls (bouts2NLS), 13
bouts2.nlsBEC (bouts2NLS), 13
bouts2.nlsFUN (bouts2NLS), 13
bouts2MLE, 10
bouts2NLS, 13
bouts3.nls, 7
bouts3.nls (bouts3NLS), 15
bouts3.nlsBEC (bouts3NLS), 15
bouts3.nlsFUN (bouts3NLS), 15
bouts3NLS, 15

calibrateDepth, 2, 17, 21, 23, 27, 32–34, 47
calibrateSpeed, 2, 17, 21, 21
call, 45
ccData <- (TDR-accessors), 40
ccData <- .TDR, data.frame-method (TDR-accessors), 40
character, 31
coerce, TDR, data.frame-method (TDR-accessors), 40
coerce, TDR, TDSpeed-method (TDR-accessors), 40
createTDR (readTDR), 36
curve, 10
data.frame, 8, 13, 15, 22, 26, 27, 35, 37, 39, 41, 45, 47
depth <- (TDR-accessors), 40
depth <- .TDR, numeric-method (TDR-accessors), 40
distSpeed, 4, 5, 22, 39
diveModel, 29, 30, 44, 46
diveModel (diveModel-class), 23
diveModel-class, 23
diveMove, 37
diveMove (diveMove-package), 2
diveMove-package, 2
dives, 24
diveStats, 25, 25, 39
divesTDR (dives), 24

eXtractDive, 40
extractDive (extractDive-methods), 27
extractDive, TDR, numeric, numeric-method (extractDive-methods), 27
extractDive, TDRcalibrate, numeric, missing-method (extractDive-methods), 27
extractDive-methods, 27

getCCData (TDR-accessors), 40
getCCData, TDR, character-method (TDR-accessors), 40
getCCData, TDR, missing-method (TDR-accessors), 40
getDAct, 45
getDAct (TDRcalibrate-accessors), 42
getDAct (TDRcalibrate, character-method (TDRcalibrate-accessors), 42
getDAct, TDRcalibrate, missing-method (TDRcalibrate-accessors), 42
getDepth (TDR-accessors), 40
getDepth, TDR-method (TDR-accessors), 40
getDiveDeriv, 24
getDiveDeriv (TDRcalibrate-accessors), 42
getDiveDeriv, diveModel-method (TDRcalibrate-accessors), 42
getDiveDeriv, TDRcalibrate-method (TDRcalibrate-accessors), 42
getDiveModel (TDRcalibrate-accessors), 42
getDiveModel, TDRcalibrate, missing-method (TDRcalibrate-accessors), 42
getDiveModel, TDRcalibrate, numeric-method (TDRcalibrate-accessors), 42
getDPhaseLab (TDRcalibrate-accessors), 42
getDPhaseLab, TDRcalibrate, missing-method (TDRcalibrate-accessors), 42
getDPhaseLab, TDRcalibrate, numeric-method (TDRcalibrate-accessors), 42
getAddress (TDRcalibrate-accessors), 42
getAddress, TDRcalibrate, character-method (TDRcalibrate-accessors), 42
getGAct, TDRcalibrate, missing-method (TDRcalibrate-accessors), 42
getSpeed (TDR-accessors), 40
getSpeed, TDRspeed-method (TDR-accessors), 40
getSpeedCoef (TDRcalibrate-accessors), 42
getSpeedCoef, TDRcalibrate-method (TDRcalibrate-accessors), 42
getTDR, 45
getTDR (TDRcalibrate-accessors), 42
getTDR, TDRcalibrate-method (TDRcalibrate-accessors), 42
getTime (TDR-accessors), 40
getTime, TDR-method (TDR-accessors), 40
grpSpeedFilter (austfilter), 3
labelBouts (bout-misc), 8
legend, 33
locator, 32
logit, 11, 12
logit (bout-misc), 8
mle, 7, 10–12
nls, 7, 13, 15
numeric, 29
oneDiveStats (diveStats), 25
optim, 11, 12
par, 31
plot, 8, 10
plot.default, 13, 15
plot.ecdf, 10
plotBouts (bout-methods), 6
plotBouts, mle-method (bout-methods), 6
plotBouts, nls-method (bout-methods), 6
plotBouts2.cdf (bouts2MLE), 10
plotBouts2.mle (bouts2MLE), 10
plotBouts2.nls, 7
plotBouts2.nls (bouts2NLS), 13
plotBouts3.nls (bouts3NLS), 15
plotDiveModel, 24, 44
plotDiveModel (plotDiveModel-methods), 28
plotDiveModel, diveModel, missing-method (plotDiveModel-methods), 28
plotDiveModel, numeric, numeric-method (plotDiveModel-methods), 28
plotDiveModel, TDRcalibrate, missing-method (plotDiveModel-methods), 28
plotDiveModel-methods, 28
plotTDR, 18, 20, 30, 40, 44
plotTDR (plotTDR-methods), 30
plotTDR, POSIXt, numeric-method (plotTDR-methods), 30
plotTDR, TDR, missing-method (plotTDR-methods), 30
plotTDR, TDRcalibrate, missing-method (plotTDR-methods), 30
plotTDR-methods, 30
plotZOC, 20
plotZOC, plotZOC-methods, 33
plotZOC, TDR, matrix-method (plotZOC-methods), 33
plotZOC, TDR, TDRcalibrate-method (plotZOC-methods), 33
plotZOC-methods, 33
points, 31
POSIXct, 41, 45, 47
predict.smooth.spline, 24
read.csv, 35–37
readLocs, 35, 39
readTDR, 24, 25, 36, 41, 42, 45
rmsDistFilter (austfilter), 3
rq, 21
rqPlot, 21, 38
runquantile, 19
seallocs, 39
show, diveModel-method (TDRcalibrate-accessors), 42
show, TDR-method (TDR-accessors), 40
show, TDRcalibrate-method (TDRcalibrate-accessors), 42
smooth.spline, 17, 19, 24
speed<-(TDR-accessors), 40
speed<-, TDRspeed, numeric-method (TDR-accessors), 40
stampDive, 2
stampDive (diveStats), 25
strptime, 31, 35, 37
subset.data.frame, 18
TDR, 17, 18, 20, 21, 24, 25, 28, 31, 33, 37, 40, 44–46
TDR (TDR-class), 41
TDR-accessors, 40
TDR-class, 41
TDR-methods (TDR-accessors), 40
TDRcalibrate, 17, 18, 20–22, 29–33, 42–44, 46
TDRcalibrate (TDRcalibrate-class), 45
TDRcalibrate-accessors, 42
TDRcalibrate-class, 45
TDRcalibrate-methods
(TDRcalibrate-accessors), 42
TDRspeed, 28, 37, 40
TDRspeed (TDR-class), 41
TDRspeed-class (TDR-class), 41
timeBudget, 2
timeBudget (timeBudget-methods), 46
timeBudget, TDRcalibrate, logical-method
(timeBudget-methods), 46
timeBudget-methods, 46

unLogit, 12
unLogit (bout-misc), 8