Package ‘tagtools’

July 18, 2023

Title Work with Data from High-Resolution Biologging Tags

Version 0.1.0

Description High-resolution movement-sensor tags typically include accelerometers
to measure body posture and sudden movements or changes in speed,
magnetometers to measure direction of travel, and pressure sensors
to measure dive depth in aquatic or marine animals. The sensors in these tags usually sample many times per second. Some tags include sensors for speed, turning rate (gyroscopes), and sound. This package provides software tools to facilitate calibration, processing, and analysis of such data. Tools are provided for: data import/export;
calibration (from raw data to calibrated data in scientific units);
visualization (for example, multi-panel time-series plots);
data processing (such as event detection, calculation of derived metrics like jerk and dynamic acceleration, dive detection, and dive parameter calculation); and statistical analysis (for example, track reconstruction, a rotation test, and Mahalanobis distance analysis).

Depends R (>= 3.4)

Imports CircStats, dplyr, graphics, latex2exp, lubridate, matlab,
ncdf4, plotly, pracma, readr, signal, stats, stringr, utils,
zoo, zoom

License GPL (>= 3)

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

URL <`https://animaltags.org`>,
<`https://animaltags.github.io/tagtools_r/index.html`>

BugReports `https://github.com/animaltags/tagtools_r/issues`

Config/Needs/website rmarkdown

Config/testthat/edition 3

NeedsCompilation no

Author Stacy DeRuiter [aut, cre, cph],
Mark Johnson [aut, cph],
David Sweeney [aut],
Ye Joo McNamara-Oh [aut],
Samuel Fynewever [aut],
(Oghenkevwe) Racheal Tejevbo [aut],
Tiago Marques [aut],
Yuqian Wang [aut],
(Oghenesuvwe) Su Ogedegbe [aut]

Maintainer Stacy DeRuiter <stacy.deruiter@calvin.edu>

Repository CRAN

Date/Publication 2023-07-18 09:10:02 UTC

R topics documented:

`a2pr` *Pitch and roll from acceleration*

Description

Pitch and roll estimation from triaxial accelerometer data. This is a non-iterative estimator with |pitch| constrained to <= 90 degrees. The pitch and roll estimates give the least-square-norm error between A and the A-vector that would be measured at the estimated pitch and roll. If A is in the animal frame, the resulting pitch and roll define the orientation of the animal with respect to its navigation frame. If A is in the tag frame, the pitch and roll will define the tag orientation with respect to its navigation frame.

Usage

```
a2pr(A, sampling_rate = NULL, fc = NULL)
```

Arguments

`A`	An nx3 acceleration matrix with columns [ax ay az] or acceleration sensor list (e.g., from readtag.R). Acceleration can be in any consistent unit, e.g., g or m/s^2.
`sampling_rate`	(optional) The sampling rate of the sensor data in Hz (samples per second). This is only needed if filtering is required. If A is a sensor data list, sampling_rate is obtained from its metadata (A$sampling_rate).
`fc`	(optional) The cut-off frequency of a low-pass filter to apply to A before computing pitch and roll. The filter cut-off frequency is in Hertz. The filter length is 4*sampling_rate/fc. Filtering adds no group delay. If fc is not specified, no filtering is performed.

Value

A list with 2 elements:

- **p:** The pitch estimate in radians
- **r:** The roll estimate in radians

Note

Output sampling rate is the same as the input sampling rate.

Frame: This function assumes a [north,east,up] navigation frame and a [forward,right,up] local frame. In these frames, a positive pitch angle is an anti-clockwise rotation around the y-axis. A positive roll angle is a clockwise rotation around the x-axis. A descending animal will have a negative pitch angle while an animal rolled with its right side up will have a positive roll angle.

See Also

`m2h`

Examples

```
samplematrix <- matrix(c(0.77, -0.6, -0.22, 0.45, -0.32, 0.99, 0.2, -0.56, 0.5),
  byrow = TRUE, nrow = 3
)
list <- a2pr(samplematrix)
```

---

absorption — *Calculates the absorption coefficient for sound in seawater*

Description

Calculates the absorption coefficient for sound in seawater

Usage

```
absorption(freq, temperature, d)
```

Arguments

`freq`	frequency in Hz
`temperature`	temperature in degrees C
`d`	depth in meters

Value

The sound absorption in dB per metre.

Note

Input arguments can be scalars, or a mixture of vectors and scalars as long as each argument is either a vector of length nx1 (with n being the same for all vector arguments) or a scalar.

After Kinsler and Frey pp. 159-160

Examples

```
absorption(140e3, 13, 10)
```

`acc_wgs84` *Calculate total acceleration*

Description

This function calculates the total acceleration due to gravitation and centripetal force at the earth's surface according to the WGS84 international gravity formula.

Usage

```
acc_wgs84(latitude)
```

Arguments

`latitude`	The latitude in degrees.

Value

g given in units of m/s^2

Note

Source: http://solid_earth.ou.edu/notes/potential/igf.htm

Examples

```
acc_wgs84(50)
```

`add_nc` *Save an item to a NetCDF or add one tag sensor or metadata variable to a NetCDF archive file.*

Description

Add one tag sensor or metadata variable to a NetCDF archive file. If the archive file does not exist, it is created. The file is assumed to be in the current working directory unless a pathname is added to the beginning of fname.

Usage

```
add_nc(file, D, vname)
```

Arguments

`file`	The name of the netCDF file to which to save. If the name does not include a .nc suffix, this will be added automatically.
`D`	The sensor data or metadata list to be saved.
`vname`	The name of the sensor data stream to be saved. Defaults to the entry "name" from the sensor or metadata list provided by the user (but an option to specify a name is provided to facilitate calling this function from `save_nc`).

Value

no return; adds a structure to an `animaltag` object

See Also

`save_nc`, `load_nc`

Examples

```
BW <- beaked_whale
add_nc("beaked_whale", njerk(BW$A), "Jerk")
```

`apply_cal` *Implement a calibration on tag sensor data*

Description

Given an appropriate set of calibration constants and information, this function will apply the calibration procedure to a tag sensor data set. Cal fields currently supported are: poly, cross, map, tcomp, tref

Usage

```
apply_cal(X, cal, Tempr = NULL)
```

Arguments

`X`	A tag sensor data list, or a matrix or vector containing tag sensor data
`cal`	A calibration list for the data in X from, for example, spherical_cal.
`Tempr`	a tag sensor data list or a vector of temperature measurements for use in temperature compensation. If Tempr is not a sensor data list, it must be the same size and sampling rate as the data in `X`. Tempr is only required if there is a tcomp item in the `cal` list.

Value

A tag sensor data structure (or a matrix or vector, if X was a matrix or vector) with the calibration implemented. Data size and sampling rate are the same as for the input data `X`, but units may have changed.

Examples

```
A_cal <- apply_cal(harbor_seal$A,spherical_cal(harbor_seal$A$data))
```

`beaked_whale` *Set of sensor lists for a beaked_whale*

Description

Data is from a _Mesoplodon densirostris_ with tag ID md13_134a. The device used was a DTAG3 and it was deployed at 2013-05-14 12:42:00 in El Hierro, Canary Islands, Spain.

Usage

```
beaked_whale
```

Format

A set of sensor lists:

A sensor list contining a triaxial acceleration matrix sampled at 25 Hz

M sensor list containing a triaxial magnetometer matrix sampled at 25 Hz

P sensor list containing a pressure (depth) vector sampled at 25 Hz

---

`block_acf` *Compute autocorrelation function*

Description

This function allows calculation of an autocorrelation function (ACF) for a dataset with multiple independent units (for example, data from several individuals, data from multiple dives by an individual animal, etc.). The groups (individual, dive, etc.) should be coded in a categorical variable. The function calculates correlation coefficients over all levels of the categorical variable, but respecting divisions between levels (for example, individual animals are kept separate).

Usage

```
block_acf(resids, blocks, max_lag, make_plot = TRUE, ...)
```

Arguments

`resids`	The variable for which the ACF is to be computed (often a vector of residuals from a fitted model)
`blocks`	A categorical variable indicating the groupings (must be the same length as resids. ACF will be computed only for data points within the same block.)
`max_lag`	ACF will be computed at 0-max_lag lags, ignoring all observations that span blocks. Defaults to the minimum number of observations in any block. The function will allow you to specify a max_lag longer than the shortest block if you so choose.
`make_plot`	Logical. Should a plot be produced? Defaults to TRUE.
`...`	Additional arguments to be passed to plot.acf

Value

A data frame with 1 variable containing the values of ACF.

Examples

```
block_acf(
  resids = ChickWeight$weight,
  blocks = ChickWeight$Chick
)
```

`block_mean` *Compute mean of sample blocks*

Description

This function is used to compute the means of successive blocks of samples.

Usage

```
block_mean(X, n, nov)
```

Arguments

`X`	A vector or a matrix containing samples of a signal in each column.
`n`	The number of samples from X to use in each analysis block.
`nov`	(optional) The number of samples that the next block overlaps the previous block. The default value is 0.

Value

A list with 2 elements:

- **Y:** A vector or matrix containing the mean value of each block. If X is a mxn matrix, Y is pxn where p is the number of complete n-length blocks with nov that can be made out of m samples, i.e., n+(p-1)*(n-nov) < m
- **samples:** The time at which each output in Y is reported, in units of samples of X. So if samples[1] = 12, then the value Y[1] corresponds to the "time" 12 samples in X.

Examples

```
samplematrix <- matrix(c(1, 3, 5, 7, 9, 11, 13, 15, 17), byrow = TRUE, ncol = 3)
list <- block_mean(samplematrix, n = 3, nov = 1)
```

`block_rms` *Compute RMS of sample blocks*

Description

This function is used to compute the RMS (root-mean-square) of successive blocks of samples.

Usage

```
block_rms(X, n, nov = NULL)
```

Arguments

`X`	A vector or a matrix containing samples of a signal in each column.
`n`	The number of samples from X to use in each analysis block.
`nov`	The number of samples that the next block overlaps the previous block.

Value

A list with 2 elements:

- **Y:** A vector or matrix containing the RMS value of each block. If X is a mxn matrix, Y is pxn where p is the number of complete n-length blocks with nov that can be made out of m samples, i.e., n+(p-1)*(n-nov) < m
- **samples:** The time at which each output in Y is reported, in units of samples of X. So if samples[1] = 12, then the value Y[1] corresponds to the "time" 12 samples in X. The times at which Y values are reported are the centers of the averaging windows.

Note

Output sampling rate is the same as the input sampling rate so s and v have the same size as p.

Frame: This function assumes a [north,east,up] navigation frame and a [forward,right,up] local frame. In these frames, a positive pitch angle is an anti-clockwise rotation around the y-axis. A descending animal will have a negative pitch angle.

Examples

```
X <- matrix(c(1:20), byrow = TRUE, nrow = 4)
block_rms(X, n = 2, nov = NULL)
```

`body_axes` *Generate the cardinal axes of an animal*

Description

This function is used to generate the cardinal axes of an animal (i.e., the longitudinal, transverse, and ventro-dorsal) from accelerometer and magnetic field measurements. This functions generates an approximate orthonormal basis from each measurement of A and M by: (i) normalizing A and M to unit length, (ii) rotating the magnetometer measurement to the horizontal plane (Mh), (iii) computing the cross-product, N, of A and Mh to generate the third axis, (iv) transposing [Mh,N,A] to form the body axis basis.

Usage

```
body_axes(A, M, sampling_rate = NULL, fc = NULL)
```

Arguments

`A`	The acceleration matrix with columns [ax ay az], or a sensor data list. Acceleration can be in any consistent unit, e.g., g or m/s^2.
`M`	The magnetometer signal matrix, M=[mx,my,mz], or a sensor data list, in any consistent unit (e.g., in uT or Gauss).
`sampling_rate`	sampling rate of A and M in Hz (optional if A and M are sensor data lists)
`fc`	(optional) The cut-off frequency of a low-pass filter to apply to A and M before computing the axes. The filter cut-off frequency is in Hz. The filter length is 4*fs/fc. Filtering adds no group delay. If fc is not specified, no filtering is performed.

Value

W, a list with entries `x`, `y`, and `z`; each is an nx3 matrix of body axes where n is the number of rows in M and A. W$x is a nx3 matrix (or a length-3 vector if A and M have one row) containing the X or longitudinal (caudo-rostral) axes. W$y is a nx3 matrix (or a length-3 vector if A and M have one row) containing the Y or transverse (left-right) axes. W$z is a nx3 matrix (or a length-3 vector if A and M have one row) containing the Z or ventro-dorsal axes. W$sampling_rate has the sampling rate of the A and M.

Note

Output sampling rate is the same as the input sampling rate. Irregularly sampled data can be used, but then filtering must not be applied (`fc = NULL`).

Frame: This function assumes a [north,east,up] navigation frame and a [forward,right,up] local frame. This function will only return the animal's cardinal axes if the tag was attached so that the sensor axes aligned with the animal's axes OR if the tag A and M measurements are rotated to account for the orientation of the tag on the animal. Otherwise, the axes returned by this function will be the cardinal axes of the tag, not the animal.

Examples

```
samplematrix1 <- matrix(c(7, 2, 3, 6, 4, 9), byrow = TRUE, ncol = 3)
samplematrix2 <- matrix(c(6, 5, 3, 4, 8, 9), byrow = TRUE, ncol = 3)
W <- body_axes(A = samplematrix1, M = samplematrix2, fc = NULL)
```

`buffer` *Buffers a signal vector into matrix*

Description

This function is used to buffer a signal vector into a matrix of data frames. If the input for nodelay is TRUE, the the signal is buffered with no delay. If nodelay is FALSE, and specifies a vector of samples to precede x[1] in an overlapping buffer.

Usage

```
buffer(x, n, overlap, opt, nodelay = FALSE)
```

Arguments

`x`	The signal vector to be buffered
`n`	The desired length of data segments (rows).
`overlap`	The desired amount of overlap between consecutive frames (columns) in the output matrix
`opt`	The vector of samples specified to precede x[1] in an overlapping buffer
`nodelay`	A logical statement to determine if the vector should be buffered with or without delay. Default is FALSE (with delay)

Value

A list with 3 elements is returned if nodelay = FALSE:

- **X:** A matrix of the buffered signal vector "vec" with "n" data segments and an overlap between consecutive frames specified by "p". The matrix starts with "opt" values if nodelay is FALSE.
- **z:** The remainder of the vector which was not included in the matrix if the last column did not have a full number of rows.

- **opt:** The last values, length of "p", of the matrix "X".

If nodelay = TRUE, then a matrix of the buffered signal vector "vec" with "n" data segments and an overlap between consecutive frames specified by "overlap". The matrix starts with "opt" values if nodelay is FALSE.

Examples

```
x <- c(1:10)
n <- 3
overlap <- 2
opt <- c(2, 1)
list1 <- buffer(x, n, overlap, opt)
list2 <- buffer(x, n, overlap, nodelay = TRUE)
```

`check_AM` *Compute field intensity of tag acceleration and magnetometer data.*

Description

Compute field intensity of acceleration and magnetometer data, and the inclination angle of the magnetic field. This is useful for checking the quality of a calibration, for detecting drift, and for validating the mapping of the sensor axes to the tag axes.

Usage

```
check_AM(A, M = NULL, fs = NULL, find_incl = TRUE)
```

Arguments

`A`	An accelerometer sensor structure or matrix with columns [ax ay az]. Acceleration can be in any consistent unit, e.g., g or m/s^2.
`M`	A magnetometer sensor structure or matrix, M=[mx,my,mz] in any consistent unit (e.g., in uT or Gauss).
`fs`	(optional) The sampling rate of the sensor data in Hz (samples per second). This is only needed if A and M are not sensor structures and filtering is required.
`find_incl`	(optional; logical) Should inclination be computed and returned? Default is TRUE.

Details

The sampling rate of fstr and incl is the same as the input sampling rate. This function automatically low-pass filters the data with a cut-off frequency of 5 Hz if the sampling rate is greater than 10 Hz. Frame: This function assumes a [north,east,up] navigation frame and a [forward,right,up] local frame.

Value

If find_incl is false, then the matrix fstr is returned. Otherwise, check_AM returns a list with elements:

- `fstr`, The estimated field intensity of A and or M in the same units as A and M. fstr is a vector or a two column matrix. If only one type of data is input, fstr will be a column vector. If both A and M are input, fstr will have two columns with the field strength of A in the 1st column and the field strength of M in the 2nd column.
- `incl`, The estimated field inclination angle (i.e., the angle with respect to the horizontal plane) in radians. incl is a column vector. By convention, a field vector pointing below the horizon has a positive inclination angle. This is only returned if the function is called with both A and M data.

Examples

```
AMcheck <- check_AM(
  A = matrix(c(-0.3, 0.52, 0.8), nrow = 1),
  M = matrix(c(22, -22, 14), nrow = 1),
  fs = 1
)
```

`cline` *Add colored line segments to a plot*

Description

This function adds colored line segments to an existing plot. The line is plotted at points specified by inputs x and y, and colored according to factor input z (with one color for each level of z).

Usage

```
cline(x, y, z, color_vector)
```

Arguments

`x`	x positions of points to be plotted
`y`	y positions of points to be plotted
`z`	a factor, the same length as x and y. Line segments in the resulting plot will be colored according to the levels of z.
`color_vector`	a list of colors to use (length should match the number of levels in z).

Value

adds colored lines to a graph

Examples

```
cline(x=ChickWeight$Time, y=ChickWeight$weight,
      z=as.factor(ChickWeight$Diet),
      color_vector=c('black', 'grey20',
                     'grey50', 'grey70'))
```

`col_line` *Plot coloured line(s) in 2 dimensions*

Description

This function is used to plot two dimensional lines with each individual line possessing a different color.

Usage

```
col_line(x, y, color, ...)
```

Arguments

`x`	A vector or matrix of values to display on the horizontal axis.
`y`	A vector or matrix of values to display on the vertical axis.
`color`	A vector or matrix of values representing the colour to draw at each point.
`...`	Additional inputs for plot()

Value

a graph with a colored line

Note

x, y and c must all be the same size. If x, y, and c are matrices, one line is drawn for each column. The color axis will by default span the range of values in c, i.e., caxis will be c(min(min(c)), max(max(c))). This can be changed by calling caxis after colline.

col_line3 *Plot coloured line(s) in 3 dimensions with plot_ly*

Description

This function is used to plot three dimensional lines with segments colored. It may be just as simple to use plotly::plot_ly() directly.

Usage

```
col_line3(x, y, z = 0, c, ...)
```

Arguments

`x`	name of object or variable containing data for x axis
`y`	name of object or variable containing data for y axis
`z`	name of object or variable containing data for z axis
`c`	name of object or variable by which to color
`...`	Additional inputs for plot_ly()

Value

a plot_ly() graphics object

Note

x, y, z and c must all be the same size vectors. The color axis will by default span the range of values in c, i.e., caxis will be c(min(min(c)), max(max(c))).

See Also

`col_line`, `cline`

Examples

```
col_line3(1:20, 1:20, 1:20, 1:20)
```

`comp_filt` *Complementary filtering of a signal.*

Description

This function breaks signal X into two or more frequency bands such that the sum of the signals in the separate bands is equal to the original signal.

Usage

```
comp_filt(X, sampling_rate = NULL, fc)
```

Arguments

`X`	A sensor vector or matrix (i.e., with a signal in each column) or sensor list (e.g., from readtag.R).
`sampling_rate`	The sampling rate of the sensor data in Hz (samples per second).
`fc`	Specifies the cut-off frequency or frequencies of the complementary filters. Frequencies are in Hz. If one frequency is given, X will be split into a low- and a high-frequency component. If fc contains more than one value, X will be split into multiple complementary bands. Each filter length is 4*sampling_rate/fc. Filtering adds no group delay.

Details

Possible input combinations: comp_filt(X,sampling_rate,fc) if X is a vector or matrix, comp_filt(X,fc = fc) if X is a list

Value

A list of filtered signals. There are n+1 sections of the list where n is the length of fc. List sections are ordered in Xf from lowest to highest frequency. Each list section contains a vector or matrix of the same size as X, and at the same sampling rate as X.

Examples

```
Xf <- comp_filt(X = beaked_whale$A$data, sampling_rate = beaked_whale$A$sampling_rate, fc = .15)
xf <- list(Xf1 = Xf[[1]], Xf2 = Xf[[2]])
plott(xf, beaked_whale$A$sampling_rate)
```

`crop` *Interactive data cropping tool.*

Description

This function plots the input data # and allows the user to select start and end times for cropping.

Usage

```
crop(X, sampling_rate = NULL, times = NULL, quiet = FALSE)
```

Arguments

`X`	A sensor list, vector or matrix. X can be regularly or irregularly sampled data in any frame and unit.
`sampling_rate`	The sampling rate of X in Hz. This is only needed if X is not a sensor list. If X is regularly sampled, sampling_rate is one number.
`times`	A vector of sampling times for X. This is only needed if X is not a sensor list and X is not regularly sampled.
`quiet`	If quiet is false, print to the screen

Details

Possible input combinations include: crop(X) if X is a sensor list, crop(X, sampling_rate) if X is a vector or matrix.

Value

A list with 3 elements:

- **Y:** A sensor list, vector or matrix containing the cropped data segment. If the input is a sensor list, the output will also be. The output has the same units, frame and sampling characteristics as the input.
- **times:** A vector of sampling times for Y. This is only returned if X is irregularly sampled and X is not a sensor list. If X is a sensor list, the sampling times are stored in the list.
- **tcues:** tcues is a two-element vector containing the start and end time cue in seconds of the data segment kept, i.e., tcues = c(start_time, end_time).

Examples

```
data <- beaked_whale
Pc <- crop(data$P, quiet=TRUE)
Ydata <- Pc$data
plot(-Ydata)
```

`crop_all` *Reduce the time span of a dataset*

Description

This function is used to reduce the time span of a dataset by cropping out any data that falls before and after two time cues.

Usage

```
crop_all(tcues, X)
```

Arguments

`tcues`	A two-element vector containing the start and end time cue in seconds of the data segment to keep, i.e., tcues = c(start_time, end_time).
`X`	A sensor list or a set of sensor lists (e.g., from load_nc).

Details

Possible input combinations: crop_all(X) if X is a sensor list or set of sensor lists, crop_all(tcues, X, Y, ...) if X, Y, ... are sensor lists.

Value

A sensor list or set of sensor lists containing the cropped data segment. The output data have the same units, frame and sampling characteristics as the input. The list may have many sublists which are additional sensor structures as required to match the input.

Examples

```
d <- find_dives(beaked_whale$P,300)
        X <- crop_all(c(d$start[1], d$end[1]), beaked_whale) #crop all data to 1st dive
        plott(X = list(X$P, X$A), r = c(1, 0), panel_labels = c('Depth', 'Acc'))
```

`crop_to` *Reduce the time span of data*

Description

This function is used to reduce the time span of data by cropping out any data that falls before and after two time cues.

Usage

```
crop_to(X, sampling_rate = NULL, tcues, times = NULL)
```

Arguments

`X`	A sensor list, vector, or matrix. X can be regularly or irregularly samples data in any frame and unit.
`sampling_rate`	The sampling rate of X in Hz. This is only needed if X is not a sensor structure.
`tcues`	A two-element vector containing the start and end time cues in seconds of the data segment to keep (i.e., tcues <- c(start_time, end_time)).
`times`	A vector of sampling times for X. This is only needed if X is not a sensor list and X is not regularly sampled.

Value

Cropped data in the same format as X, unless X is irregularly sampled and NOT a sensor list. In that case, the function returns a list with 2 elements:

- **X:** A sensor list, vector or matrix containing the cropped data segment. If the input is a sensor list, the output will also be. The output has the same units, frame and sampling characteristics as the input.
- **times:** A vector of sampling times for Y. This is only returned if X is irregularly sampled and X is not a sensor list. (If X is a sensor list, the sampling times are stored in the list.)

Examples

```
 d <- find_dives(beaked_whale$P,300)
P2 <- crop_to(beaked_whale$P, tcues = c(d$start[1], d$end[1])) #crop to 1st dive
 plott(X = list(P2), r=c(1), panel_labels=c('Depth'))
```

`csv2struct` *Read tag metadata from csv*

Description

Read a CSV metadata file and convert it into a metadata list. A metadata file is a comma-separated text file (.csv) containing a line for each metadata entry. The first comma-separated field in each line is the name of the entry. The last field in each line contains the value to be assigned to this metadata entry. The value can be a string or number but is always saved as a string in the structure - it is up to downstream users of the metadata to parse/decode the entries.

Usage

```
csv2struct(fname)
```

Arguments

`fname`	Name of the text file to be read. If no file extension is provided, '.csv' will be added automatically. If the file is not located in the current working directory, then `fname` must include the correct relative or absolute path.

Value

a metadata list populated from `fname` (one list element per row in the file). All list elements are stored as `"character"` class objects (even if the field contains a number, a date, etc) - no attempt is made to determine the most appropriate class for each item.

Examples

```
hold <- system.file("extdata","metadata_example.csv", package = "tagtools", mustWork = TRUE)
S <- csv2struct(hold)
```

`decdc` *Reduce the sampling rate*

Description

This function is used to reduce the sampling rate of a time series by an integer factor.

Usage

```
decdc(x, df)
```

Arguments

`x`	A data structure, vector or matrix containing the signal(s) to be decimated. If x is a matrix, each column is decimated separately.
`df`	The decimation factor. The output sampling rate is the input sampling rate divided by df. df must be an integer greater than 1.

Value

y The decimated signal vector or matrix. It has the same number of columns as x but has 1/df of the rows.

Note

Decimation is performed by first low-pass filtering x and then keeping 1 sample out of every df. A symmetric FIR filter with length 12*df and cutoff frequency 0.4*fs/df is used. The group delay of the filter is removed. For large decimation factors (e.g., df»50), it is better to perform several decimations with lower factors. For example to decimate by 120, use: decdc(decdc(x,10),12).

Examples

```
s <- matrix(sin(2 * pi / 100 * c(0:1000) - 1), ncol = 1)
plot(c(1:length(s)), s)
y <- decdc(x = s, df = 4)
plot(c(1:length(y)), y)
```

decz — *Decimate sampling rate recursively.*

Description

Recursive sampling rate decimator. This function can be run iteratively over a long data set, e.g., to decimate an entire recording that is too large to be read into memory.

Usage

```
decz(x, df = NULL, Z = NULL, nf = 12, frbw = 0.8)
```

Arguments

`x`	A vector, matrix, or tag data list containing the signal(s) to be decimated. If x is a matrix, each column is decimated separately. If inputs `df` and `Z` are both provided, then the value of `df` stored in `Z` will override the user-provided `df`.
`df`	The decimation factor. The output sampling rate is the input sampling rate divided by df. df must be an integer greater than 1. df can also be a three element vector in which case: df(1) is the decimation factor; df(2) is the number of output samples spanned by the filter (default value is 12). A larger value makes the filter steeper; df(3) is the fractional bandwidth of the filter (default value is 0.8) relative to the output Nyquist frequency. If df(2) is greater than 12, df(3) can be closer to 1.
`Z`	The 'state' list that is generated by a previous call to decz. This is how the function keeps track of filter internal values (i.e., memory) from call-to-call.
`nf`	The number of output samples spanned by the filter (default value is 12). A larger value makes the filter steeper.
`frbw`	The fractional bandwidth of the filter (default value is 0.8) relative to the output Nyquist frequency. If `nf` is greater than 12, `frbw` can be closer to 1.

Details

The first time decz is called, use the following format: y = decz(x,df). The subsequent calls to decz for contiguous input data are: decz(x,Z). The final call when there is no more input data is: decz(x = NULL, Z = Z). Each output y in the above contains a segment of the decimated signal and so these need to be concatenated. Decimation is performed in the same way as for `decdc`. The group delay of the filter is removed. For large decimation factors (e.g., df much greater than 50), it is better to perform several nested decimations with lower factors.

Value

A list with elements:

- **y:** The decimated signal vector or matrix. It has the same number of columns as x but has, on average, 1/df of the rows.

- **Z:** The state list (for internal tracking of filter internal values). Contains elements df (the decimation factor), nf (used to compute the filter length), frbw (the bandwidth of the filter relative to the new Nyquist frequency), h (the FIR filter coefficients), n (the filter length), z (padded signal used for filtering), and ov ("overflow" samples to be passed to future iterations).

See Also

decdc

Examples

```
plott(list(Accel = beaked_whale$A)) # acceleration data before decimation
a_rows <- nrow(beaked_whale$A$data)
a_ind <- data.frame(start = c(1, floor(a_rows / 3), floor(2 * a_rows / 3)))
a_ind$end <- c(a_ind$start[2:3] - 1, a_rows)
df <- 10
Z <- NULL
y <- NULL
for (k in 1:nrow(a_ind)) {
  decz_out <- decz(
    x = beaked_whale$A$data[c(a_ind[k, 1]:a_ind[k, 2]), ],
    df = df, Z = Z
  )
  df <- NULL
  Z <- decz_out$Z
  y <- rbind(y, decz_out$y)
}
```

depth2pressure *Convert depth to pressure*

Description

This function is used to convert the depth (in meters) to the pressure in Pascals.

Usage

```
depth2pressure(d, latitude)
```

Arguments

d	The depth in meters
latitude	The latitude in degrees

Value

The pressure in Pa

Note

Based on the Leroy and Parthiot (1998) formula. See: http://resource.npl.co.uk/acoustics/techguides/soundseawater/content.h

Examples

```
depth2pressure(1000, 27)
```

`depth_rate` *Estimate the vertical velocity*

Description

This function is used to estimate the vertical velocity by differentiating a depth or altitude time series. A low-pass filter reduces the sensor noise that is amplified by the differentiation.

Usage

```
depth_rate(p, fs, fc, depth)
```

Arguments

`p`	A vector of depth or altitude data, or an animaltags list object containing depth or altitude data.
`fs`	(required only if p is a vector) is the sampling rate of p in Hz.
`fc`	(optional) A smoothing filter cut-off frequency in Hz. If fc is not given, a default value is used of 0.2 Hz (5 second time constant).
`depth`	(optional) The behavior of animals. Required only if dealing with animals not behave descent but ascent.

Value

v, The vertical velocity with the same sampling rate as p. v is a vector with the same dimensions as p. The unit of v depends on the unit of p. For example, if p is in meters, v is in meters/second

Note

The low-pass filter is a symmetric FIR with length 4fs/fc. The group delay of the filters is removed. Usually, the function handles data pertaining to diving animals, where data is measured as the depth beneath the water surface. For ascending data coming from birds and alike data, setting depth = FALSE will help calculating the right vertical velocity.

Examples

```
v <- depth_rate(p = beaked_whale$P)
plott(list(beaked_whale$P$data, v),
  fs = beaked_whale$P$sampling_rate,
  r = c(1, 0), panel_labels = c("Depth\n(m)", "Vertical Velocity\n(m/s)")
)
```

detect_peaks *Detect peaks in signal vector data*

Description

This function detects peaks in time series data that exceed a specified threshold and returns each peak's start time, end time, maximum peak value, time of the maximum peak value, threshold level, and blanking time.

Usage

```
detect_peaks(
  data,
  sr,
  FUN = NULL,
  thresh = NULL,
  bktime = NULL,
  plot_peaks = NULL,
  quiet = FALSE,
  ...
)
```

Arguments

`data`	A vector (of all positive values) or matrix of data to be used in peak detection. If data is a matrix, you must specify a FUN to be applied to data.
`sr`	The sampling rate in Hz of the date. This is the same as fs in other tagtools functions. This is used to calculate the bktime in the case that the input for bktime is missing.
`FUN`	A function to be applied to data before the data is run through the peak detector. Only specify the function name (i.e. njerk). If left blank, the data input will be immediately passed through the peak detector.
`thresh`	The threshold level above which peaks in signal are detected. Inputs must be in the same units as the signal. If the input for thresh is missing/empty, the default level is the 0.99 quantile
`bktime`	The specified length of time (seconds) between signal values detected above the threshold value (from the moment the first peak recedes below the threshold level to the moment the second peak surpasses the threshold level) that is required for each value to be considered a separate and unique peak. If the input for bktime is missing/empty the default value for the blanking time is set as the .80 quantile of the vector of time differences for signal values above the specified threshold.
`plot_peaks`	A conditional input. If the input is TRUE or missing, an interactive plot is generated, allowing the user to manipulate the thresh and bktime values and observe the changes in peak detection. If the input is FALSE, the interactive

	plot is not generated. Look to the console for help on how to use the plot upon running of this function.
`quiet`	If quiet is true, do not print to the screen
`...`	Additional inputs to be passed to FUN

Value

A data frame containing the start times, end times, peak times, peak maxima, thresh, and bktime. All times are presented as the sampling value.

Note

As specified above under the description for the input of plot_peaks, an interactive plot can be generated, allowing the user to manipulate the thresh and bktime values and observe the changes in peak detection. The plot output is only given if the input for plot_peaks is specified as true or if the input is left missing/empty.

Examples

```
BW <- beaked_whale
detect_peaks(data = BW$A$data, sr = BW$A$sampling_rate,
FUN = njerk, thresh = NULL, bktime = NULL,
plot_peaks = NULL, sampling_rate = BW$A$sampling_rate, quiet=TRUE)
```

`dive_stats` *Compute summary statistics for dives or flights*

Description

Given a depth/altitude profile and a series of dive/flight start and end times, compute summary dive statistics.

Usage

```
dive_stats(
  P,
  X = NULL,
  dive_cues,
  sampling_rate = NULL,
  prop = 0.85,
  angular = FALSE,
  X_name = NULL,
  na.rm = TRUE
)
```

Arguments

`P`	Depth data. A vector (or one-column matrix), or a tag sensor data list.
`X`	(optional) Another data stream (as a vector (or a one-column matrix) or a tag sensor data list) for which to compute mean and variability. If `angular` is TRUE, interpreted as angular data (for example pitch, roll, or heading) and means and variances are computed accordingly. The unit of measure must be radians (NOT degrees). Currently, `X` must be regularly sampled.
`dive_cues`	A two-column data frame or matrix with dive/flight start times in the first column and dive/flight end times in the second. May be obtained from `find_dives`. Units should be seconds since start of tag recording.
`sampling_rate`	(optional and ignored if `P` or `X` are tag sensor data lists) Sampling rate of `P` (and `X`, if `X` is given). If omitted, then input data must be sensor data lists. If one value is given and both `P` and `X` are input, they are assumed to have the same sampling rate. If `P` and `X` have different sampling rates, then this input can have two elements (first for `P`, second for `X`).
`prop`	The proportion of the maximal excursion to use for defining the "destination" phase of a dive or flight. For example, if `prop` is 0.85 (the default), then the destination phase lasts from the first to the last time depth/altitude exceeds 0.85 times the within-dive maximum.
`angular`	Is X angular data? Defaults to FALSE.
`X_name`	A short name to use for X variable in the output data frame. For example, if X is pitch data, use X_name='pitch' to get outputs column names like mean_pitch, etc. Defaults to 'angle' for angular data and 'aux' for non-angular data.
`na.rm`	Logical, default is TRUE. If TRUE, then returned mean values ignore missing values, computing an average over all non-missing observations.

Details

In addition to the maximum excursion and duration, `dive_stats` divides each excursion into three phases: "to" (descent for dives, ascent for flights), "from" (ascent for dives, descent for flights), and "destination". The "destination" (bottom for dives and top for flights) phase of the excursion is identified using a "proportion of maximum depth/altitude" method, whereby for example the bottom phase of a dive lasts from the first to the last time the depth exceeds a stated proportion of the maximum depth. Average vertical velocity is computed for the to and from phases using a simple method: total depth/altitude change divided by total time. If an angular data variable is also supplied (for example, pitch, roll or heading), then the circular mean (computed via `circ.mean`) and variance (computed via `circ.disp` and reporting the `var` output) are also computed for each dive phase and the dive as a whole.

Value

A data frame with one row for each dive/flight and columns as detailed below. All times are in seconds, and rates in units of x/sec where x is the units of `P`.

`max` The maximum depth or altitude `st` start time of dive (seconds) - from input dive_cues `et` end time of dive (seconds) - from input dive_cues `dur` The duration of the excursion `dest_st` The start time of the destination phase in seconds since start of tag recording (which is also the end time of to phase) `dest_et` The end time of the destination phase in seconds since start of tag recording (which is also the start of the from phase). `dest_dur` The duration in seconds of destination phase `to_dur` The duration in seconds of to phase `from_dur` The duration in seconds of from phase `mean_angle` If angular=TRUE and X is input, the mean angle for the entire excursion. Values for each phase are also provided in columns `mean_to_angle`, `mean_dest_angle`, and `mean_from_angle`. `angle_var` If angular=TRUE and X is input, the angular variance for the entire excursion. Values for each phase are also provided individually in columns `to_angle_var`, `dest_angle_var`, and `from_angle_var`. `mean_aux` If angular=FALSE and X is input, the mean value of X for the entire excursion. Values for each phase are also provided in columns `mean_to_aux`, `mean_dest_aux`, and `mean_from_aux`. `aux_sd` If angular=FALSE and X is input, the standard deviation of X for the entire excursion. Values for each phase are also provided individually in columns `to_aux_sd`, `dest_aux_sd`, and `from_aux_sd`. #'

See Also

`find_dives`

`draw_axis` *Draw time axis on plott plot.*

Description

This function is called by `plott` to add a time axis to a plot created by `plott`. Users are unlikely to need to call the function directly.

Usage

```
draw_axis(side = 1, x = NULL, date_time, last_panel)
```

Arguments

`side`	See `axis`.
`x`	A date-time or date object, or other types of objects that can be converted appropriately.
`date_time`	Logical. Is the data being plotted date-time (POSIX) or time in seconds?
`last_panel`	Logical. Is this the last panel (in other words, should x axis tick labels be drawn)?

Value

a time axis on a graph

dsf *Estimate the dominant stroke frequency*

Description

This function can be used to estimate the dominant stroke frequency from triaxial accelerometer data [ax,ay,az].

Usage

```
dsf(A, sampling_rate = NULL, fc = NULL, Nfft = NULL)
```

Arguments

`A`	A sensor data list or an nx3 acceleration matrix with columns [ax ay az]. Acceleration can be in any consistent unit, e.g., g or m/s^2.
`sampling_rate`	The sampling rate of the sensor data in Hz (samples per second).
`fc`	(optional) The cut-off frequency in Hz of a low-pass filter to apply to A before computing the spectra. This prevents high frequency transients e.g., in foraging, from dominating the spectra. The filter length is 6*sampling_rate/fc. If fc is not specified, it defaults to 2.5 Hz. If fc>sampling_rate/2, the filtering operation is skipped.
`Nfft`	(optional) The FFT length and therefore the frequency resolution. The default value is the power of two closest to 20*sampling_rate, i.e., an analysis block length of about 20 s and a frequency resolution of about 0.05 Hz. A shorter FFT may be required if movement behaviour is very variable. A longer FFT may work well if propulsion is continuous and stereotyped.

Details

Animals tend to produce propulsive movements with a narrow frequency range. These movements cause cyclical changes in posture and/or specific acceleration, both of which are measured by an animal-attached accelerometer. Thus sections of accelerometer data that largely contain propulsion should show a spectral peak in one or more axes at the dominant stroke frequency.

Value

A list with 2 elements:

- **fpk:** The dominant stroke frequency (i.e., the peak frequency in the sum of the acceleration power spectra) in Hz. Quadratic interpolation is used over the spectral peak to improve resolution.
- **q:** The quality of the peak measured by the peak power divided by the mean power of the spectra. This is a dimensionless number which is large if there is a clear spectral peak.

Note

Frame: This function makes no assumption about accelerometer frame. Data in any frame can be used.

Data selection: This function works best if the sensor matrix, A, covers an interval in which propulsion is the main activity. This could be a complete dive or an interval of running or flapping flight. The interval length should be at least Nfft/sampling_rate seconds, i.e., 20 s for the default FFT length.

Examples

```
dsf(harbor_seal$A)
```

`euler2rotmat` *Make a rotation (or direction cosine) matrix*

Description

This function is used to make a rotation (or direction cosine) matrix out of sets of Euler angles, pitch, roll, and heading.

Usage

```
euler2rotmat(p, r, h)
```

Arguments

`p`	The pitch angle in radians.
`r`	The roll angle in radians.
`h`	The heading or yaw angle in radians.

Value

One or more 3x3 rotation matrices. If p, r, and h are all scalars, Q is a 3x3 matrix, Q = H

Examples

```
vec1 <- matrix(c(1:10), nrow = 10)
vec2 <- matrix(c(11:20), nrow = 10)
vec3 <- matrix(c(21:30), nrow = 10)
Q <- euler2rotmat(p = vec1, r = vec2, h = vec3)
```

extract *Extract a sub-sample of data*

Description

This function is used to extract a sub-sample of data from a vector or matrix.

Usage

```
extract(x, sampling_rate, tst, ted)
```

Arguments

`x`	A vector or matrix of measurements. If x is a matrix, each column is treated as a separate measurement vector.
`sampling_rate`	the sampling rate in Hz of the data in x.
`tst`	Defines the start time in seconds of the interval to be extracted from x.
`ted`	Defines the end time in seconds of the interval to be extracted from x.

Value

X: A matrix containing a sub-sample of x. X has the same number of columns as x. The length of the sub-sample will be round(sampling_rate*(tend-tstart)) samples.

Note

Output sampling rate is the same as the input sampling rate.

If either tst or ted are beyond the length of x, non-existing samples will be replaced with NaN in X.

Examples

```
BW <- beaked_whale
BW_subset <- extract(x = BW$A$data, sampling_rate = BW$A$sampling_rate, tst = 3, ted = 100)
```

extract_cues *Extract multiple sub-samples of data*

Description

This function is used to extract multiple sub-samples of data from a vector or matrix.

Usage

```
extract_cues(x, sampling_rate, cues, len)
```

Arguments

`x`	is a vector or matrix of measurements. If x is a matrix, each column is treated as a separate measurement vector.
`sampling_rate`	is the sampling rate in Hz of the data in x.
`cues`	defines the start time in seconds of the intervals to be extracted from x.
`len`	is the length of the interval to extract in seconds. This should be a scalar.

Value

A list with 2 elements:

- **X:** A matrix containing sub-samples of x. If x is a vector, X has as many columns as there are cues, i.e., each cue generates a column of X. If x is a pxm matrix, X will be a qxmxn matrix where n is the size of cues and q is the length of the interval requested, i.e., round(sampling_rate*len) samples.
- **cues:** The list of cues actually used. cues that require data outside of x are rejected.

Note

Output sampling rate is the same as the input sampling rate.

Examples

```
BW <- beaked_whale # beaked_whale must be in your working directory
list <- extract_cues(x = BW$A$data, sampling_rate = BW$A$sampling_rate, cues = c(6, 40), len = 11)
```

`find_dives` *Find time cues for dives*

Description

This function is used to find the time cues for the start and end of either dives in a depth record or flights in an altitude record.

Usage

```
find_dives(p, mindepth, sampling_rate = NULL, surface = 1, findall = 0)
```

Arguments

`p`	A depth or altitude time series (a sensor data list or a vector) in meters.
`mindepth`	The threshold in meters at which to recognize a dive or flight. Dives shallow or flights lower than mindepth will be ignored.
`sampling_rate`	The sampling rate of the sensor data in Hz (samples per second).

`surface`	(optional) The threshold in meters at which the animal is presumed to have reached the surface. Default value is 1. A smaller value can be used if the dive/altitude data are very accurate and you need to detect shallow dives/flights.
`findall`	(optional) When TRUE, forces the algorithm to include incomplete dives at the start and end of the record. Default is FALSE which only recognizes complete dives.

Value

dives is a data frame with one row for each dive/flight found. The columns of dives are: start (time in seconds of the start of each dive/flight), end (time in seconds of the start of each dive/flight), max (maximum depth/altitude reached in each dive/flight), tmax (time in seconds at which the animal reaches the max depth/altitude).

Examples

```
BW <- beaked_whale
dives <- find_dives(p = BW$P$data,
sampling_rate = BW$P$sampling_rate,
mindepth = 25, surface = 5,
findall = FALSE)
```

`fir_nodelay` *Delay-free filtering*

Description

This function is used to gather a delay-free filtering using a linear-phase (symmetric) FIR filter followed by group delay correction. Delay-free filtering is needed when the relative timing between signals is important e.g., when integrating signals that have been sampled at different rates.

Usage

```
fir_nodelay(x, n, fc, qual = "low", return_coefs = FALSE)
```

Arguments

`x`	The signal to be filtered. It can be multi-channel with a signal in each column, e.g., an acceleration matrix. The number of samples (i.e., the number of rows in x) must be larger than the filter length, n.
`n`	The length of symmetric FIR filter to use in units of input samples (i.e., samples of x). The length should be at least 4/fc. A longer filter gives a steeper cut-off.
`fc`	The filter cut-off frequency relative to sampling_rate/2=1. If a single number is given, the filter is a low-pass or high-pass. If fc is a vector with two numbers, the filter is a bandpass filter with lower and upper cut-off frequencies given by fc(1) and fc(2). For a bandpass filter, n should be at least 4/fc(1) or 4/diff(fc) whichever is larger.

`qual`	An optional qualifier determining if the filter is: "low" for low-pass (the default value if fc has a single number), or "high" for high-pass. Default is "low".
`return_coefs`	Logical. Return filter coefficients instead of filtered signal? If TRUE, the function will return the FIR filter coefficients instead of the filtered signal. Default is FALSE.

Value

If return_coefs is FALSE (the default), `fir_nodelay()` returns the filtered signal (same size as x). If return_coefs is TRUE, returns the vector of filter coefficients only.

Note

The filter is generated by a call to `fir1`: `h <- fir1(n, fc, qual)`.

h is always an odd length filter even if n is even. This is needed to ensure that the filter is both symmetric and has a group delay which is an integer number of samples.

The filter has a support of n samples, i.e., it uses n samples from x to compute each sample in y.

The input samples used are those from n/2 samples before to n/2 samples after the sample number being computed. This means that samples at the start and end of the output vector y need input samples before the start of x and after the end of x. These are faked by reversing the first n/2 samples of x and concatenating them to the start of x. The same trick is used at the end of x. As a result, the first and last n/2 samples in y are untrustworthy. This initial condition problem is true for any filter but the FIR filter used here makes it easy to identify precisely which samples are unreliable.

Examples

```
x <- sin(t(2 * pi * 0.05 * (1:100)) +
  t(cos(2 * pi * 0.25 * (1:100))))
Y <- fir_nodelay(x = x, n = 30, fc = 0.2, qual = "low")
plot(c(1:length(x)), x,
  type = "l", col = "grey42",
  xlab = "index", ylab = "input x and output y"
)
lines(c(1:length(Y)), Y, lwd = 2)
```

`fit_tracks` *Integrate track with reference positions*

Description

Simple track integration method to merge infrequent but accurate positions with a regularly sampled track that is not absolutely accurate.

Usage

```
fit_tracks(P, times = NULL, D, sampling_rate)
```

Arguments

`P`	a two column matrix or data frame containing the anchor positions. The first column should be the "northing" and the second the "easting" coordinates. (If data frame is input, then columns with those two names, in any position, will be used if present).
`times`	a vector of times corresponding to the positions P. If P is a data frame with a column called "times" then that column will be used. Times are in seconds since the start of the regularly sampled track. times must have the same number of rows as P. Times must be greater than or equal to 0 and less than the time length of the regularly sampled track.
`D`	a two column matrix containing the regularly sampled track points. If D is a data frame with columns named 'northing' and 'easting' those will be used regardless of position; otherwise the first column will be northing and the second easting. The two columns contain the 'x' and 'y' coordinates of the track points in a local level frame. Units, axes and frame must match those of P.
`sampling_rate`	is the sampling rate in Hz of D.

Value

D, a data frame with 4 columns: "northing" and "easting" along the new track, and "current_n" and "current_e", the track increments needed to match the tracks. If the difference between the two tracks is due to the medium moving, these increments can be considered an estimate of the current in m/s. The axes and frame are the same as for the input data.

`fix_offset_3d` *Estimate the offset in each axis*

Description

This function is used to estimate the offset in each axis of a triaxial field measurement, e.g., from an accelerometer or magnetometer. This is useful for correcting drift or calibration errors in a sensor.

Usage

```
fix_offset_3d(X)
```

Arguments

`X`	A sensor list or matrix containing measurements from a triaxial field sensor such as an accelerometer of magnetometer. X can be in any units and frame.

Value

A list with 2 elements:

- **X:** A sensor list or matrix containing the adjusted triaxial sensor measurements. It is the same size and has the same sampling rate and units as the input data. If the input is a sensor list, the output will also.
- **G:** A calibration list containing one field: G$poly. The first column of G$poly contains 1 as this function does not adjust the scale factor of X. The second column of G$poly is the offset added to each column of X.

Note

This function is only usable for field sensors. It will not work for gyroscope data.

Examples

```
s <- fix_offset_3d(harbor_seal$A)
```

`fix_pressure` *Correct a depth or altitude profile*

Description

This function is used to correct a depth or altitude profile for offsets caused by miscalibration and temperature. This function finds minima in the dive/altitude profile that are consistent with surfacing/landing. It uses the depth/height at these points to fit a temperature regression.

Usage

```
fix_pressure(p, t, sampling_rate, maxp = NULL)
```

Arguments

`p`	A sensor list or vector of depth/altitude in meters
`t`	A sensor list or vector of temperature in degrees Celsius
`sampling_rate`	The sampling_rate of p and t in Hz. This is only needed if p and t are not sensor lists. The depth and temperature must both have the same sampling rate (use ‘decdc‘ if needed to achieve this).
`maxp`	The maximum depth or altitude reading in the pressure data for which the animal could actually be at the surface. This is a rough measurement of the potential error in the pressure data. The unit is meters. Start with a small value, e.g., 2m and rerun fix_pressure with a larger value if there are still obvious temperature-related errors in the resulting depth/altitude profile.

Value

A list with 2 elements:

- **p:** A sensor list or vector of corrected depth/altitude measurements at the same sampling rate as the input data. If the input is a sensor list, the output will also be.
- **pc:** A list containing the pressure offset and temperature correction coefficients. It has fields: pc$tref which is the temperature compensation polynomial. This is used within the function to correct pressure as follows: p + stats::polyval(pc$tcomp, t - pc$tref).

Note

This function makes a number of assumptions about the depth/altitude data and about the behaviour of animals: First, the depth data should have few incorrect outlier (negative) values that fall well beyond the surface. These can be reduced using median_filter.m before calling fix_depth. Second, the animal is assumed to be near the surface at least 2

`get_researcher` *Find matching researcher in a list of known tag researchers*

Description

Find matching researcher in a list of known tag researchers

Usage

```
get_researcher(initial)
```

Arguments

`initial`	a two-letter code for the researcher of interest (first letter of first name and first letter of last name)

`get_species` *Find matching species in a list of marine mammals*

Description

Find matching species in a list of marine mammals

Usage

```
get_species(initial)
```

Arguments

`initial`	a two-letter code for the species of interest (first letter of Genus and first letter of species)

harbor_seal *Set of sensor lists for a harbor seal*

Description

Data is from a _Phoca vitulina_ with tag ID 'hs16_265c'. The device used was a DTAG4 and it was deployed at 2016-09-21 07:55:22 in Husum, Germany.

Usage

```
harbor_seal
```

Format

A set of sensor lists:

A sensor list contining a triaxial acceleration matrix sampled at 5 Hz

M sensor list containing a triaxial magnetometer matrix sampled at 5 Hz

P sensor list containing a pressure (depth) vector sampled at 5 Hz

POS sensor list containing a position matrix with columns [sampling time, latitude, longitude]

hilbert_env *Compute the envelope of X using Hilbert transform. Compute the envelope of the signal matrix X using the Hilbert transform. To avoid long transforms, this function uses the overlap and add method.*

Description

Compute the envelope of X using Hilbert transform.

Compute the envelope of the signal matrix X using the Hilbert transform. To avoid long transforms, this function uses the overlap and add method.

Usage

```
hilbert_env(X, N = 1024)
```

Arguments

`X`	a vector or matrix of signals. If X is a matrix, each column is treated as a separate signal. The signals must be regularly sampled for the result to be correctly interpretable as the envelope.
`N`	(optional) specifies the transform length used. The default value is 1024 and this may be fine for most situations.

Value

E, the envelope of X. E is the same size as X: it has the same number of columns and the same number of samples per signal. It has the same units as X but being an envelope, all values are >=0.

Examples

```
s <- matrix(sin(0.1 * c(1:10000)), ncol = 1) *
 matrix(sin(0.001 * c(1:10000)), ncol = 1)
E <- hilbert_env(s)
plot(c(1:length(s)), s, col = 'grey34')
lines(c(1:length(E)), E, col = 'black')
```

`hilbert_transform` *Return the Hilbert transform of a signal*

Description

This function is used to compute the Hilbert transform of a signal. It is based on function Hilbert-Transform() from (defunct) package hht, which was modified from the EMD package by Donghoh Kim and Hee-Seok Oh (http://dasan.sejong.ac.kr/~dhkim/software.emd.html)

Usage

```
hilbert_transform(x)
```

Arguments

`x`	The signal vector to be buffered

Value

The "analytic signal," in other words the Hilbert transform of the input signal x

Examples

```
timez <- seq(from = 0, by = 1/1024, to = 1)
x <- sin(2*pi*60*timez)
y <- hilbert_transform(x)
```

htrack *Simple horizontal dead-reckoned track*

Description

This function is used to estimate the simple horizontal dead-reckoned track (pseudo-track) based on speed and heading. This differs from ptrack in that the animals body angle is not considered. This makes it appropriate for animals that do not always move in the direction of their longitudinal axis.

Usage

```
htrack(A, M, s, sampling_rate = NULL, fc = 0.2)
```

Arguments

`A`	The nx3 acceleration matrix with columns [ax ay az] or acceleration sensor list. Acceleration can be in any consistent unit, e.g., g or m/s^2.
`M`	The magnetometer signal matrix, M = [mx,my,mz] in any consistent unit (e.g., in uT or Gauss) or magnetometer sensor list. A and M must have the same size (and so are both measured at the same sampling rate).
`s`	The forward speed of the animal in m/s. s can be a single number meaning that the animal is assumed to travel at a constant speed. s can also be a vector with the same number of rows as M, e.g., generated by `ocdr`.
`sampling_rate`	The sampling rate of the sensor data in Hz (samples per second).
`fc`	(optional) Specifies the cut-off frequency of a low-pass filter to apply to A and M before computing heading. The filter cut-off frequency is in Hz. The filter length is 4*sampling_rate/fc. Filtering adds no group delay. If fc is empty or not given, the default value of 0.2 Hz (i.e., a 5 second time constant) is used.

Value

Data frame track containing the estimated track in a local level frame. The track is defined as meters of northward and eastward movement (termed 'northing' and 'easting', i.e, columns of track are `northing` and `easting` relative to the animal's position at the start of the measurements (which is defined as [0,0]). The track sampling rate is the same as for the input data and so each row of track defines the track coordinates at times 0,1/sampling_rate,2/sampling_rate,... relative to the start time of the measurements.

Note

Frame: This function assumes a [north,east,up] navigation frame and a [forward,right,up] local frame. Both A and M must be rotated if needed to match the animal's cardinal axes otherwise the track will not be meaningful. Unless the local declination angle is also corrected with rotframe, the dead-reckoned track will use magnetic north rather than true north.

CAUTION: dead-reckoned tracks are usually very inaccurate. They are useful to get an idea of HOW animals move rather than WHERE they go. Few animals probably travel in exactly the direction of their longitudinal axis. Additionally, measuring the precise orientation of the longitudinal axis of a non-rigid animal is fraught with error. Moreover, if there is net flow in the medium, the animal will be advected by the flow in addition to its autonomous movement. For swimming animals this can lead to substantial errors. The forward speed is assumed to be with respect to the medium so the track derived here is NOT the 'track-made-good', i.e., the geographic movement of the animal. It estimates the movement of the animal with respect to the medium. There are numerous other sources of error so use at your own risk!

See Also

`ptrack`, `fit_tracks`, `track3D`

Examples

```
bwhtrack <- htrack(A = beaked_whale$A, M = beaked_whale$M, s = 4)
plot(bwhtrack$easting, bwhtrack$northing, xlab = "Easting, m", ylab = "Northing, m")
```

`image_irreg` *Plot an image with an irregular grid.*

Description

This function is used to plot an image with an irregular grid. This is useful for plotting matrix data (i.e., sampled data that is a function of two parameters) in which one or both of the sampling schemes is not regularly spaced. image_irreg plots R(i,j) as a coloured patch centered on x(i), y(j) and with dimension determined by x[i] - x[i-1] and y[i] - y[i-1].

Usage

```
image_irreg(x, y, R)
```

Arguments

`x`	is a vector with the horizontal axis coordinates of each value in R.
`y`	is a vector with the vertical axis coordinates of each value in R.
`R`	is a matrix of measurements to display. The values in R are converted to colours in the current colormap and caxis. R must be length(x) by length(y). Use NaN to have a patch not display.

Value

an image plot on an irregular grid

inclination *Estimate the inclination angle*

Description

This function is used to estimate the local magnetic field vector inclination angle directly from acceleration and magnetic field measurements.

Usage

```
inclination(A, M, fc = NULL)
```

Arguments

A	The accelerometer data structure or signal matrix, A = [ax,ay,az] in any consistent unit (e.g., in g or m/s2). A can be in any frame.
M	The magnetometer data structure or signal matrix, M = [mx,my,mz] in any consistent unit (e.g., in uT or Gauss). M must be in the same frame as A.
fc	(optional) The cut-off frequency of a low-pass filter to apply to A and M before computing the inclination angle. The filter cut-off frequency is with respect to 1=Nyquist frequency. Filtering adds no group delay. If fc is not specified, no filtering is performed.

Value

The magnetic field inclination angle in radians.

Note

Output sampling rate is the same as the input sampling rate.

Frame: This function assumes a [north,east,up] navigation frame and a [forward,right,up] local frame. In these frames, the magnetic field vector has a positive inclination angle when it points below the horizon. Other frames can be used as long as A and M are in the same frame however the interpretation of incl will differ accordingly.

Examples

```
A <- matrix(c(1, -0.5, 0.1, 0.8, -0.2, 0.6, 0.5, -0.9, -0.7),
  byrow = TRUE, nrow = 3, ncol = 3
)
M <- matrix(c(1.3, -0.25, 0.16, 0.78, -0.3, 0.5, 0.5, -0.49, -0.6),
  byrow = TRUE, nrow = 3, ncol = 3
)
incl <- inclination(A, M)
```

`interp2length`	*Interpolate regularly sampled data to increase its sampling rate and match its length to another variable.*

Description

This function is used to reduce the time span of data by cropping out any data that falls before and after two time cues.

Usage

```
interp2length(X, Z, fs_in = NULL, fs_out = NULL, n_out = NULL)
```

Arguments

`X`	A sensor list, vector, or matrix. If x is or contains matrix, each column is treated as an independent signal.
`Z`	is a sensor structure, vector or matrix whose sampling rate and length is to be matched.
`fs_in`	is the sampling rate in Hz of the data in X. This is only needed if X is not a sensor structure.
`fs_out`	is the required new sampling rate in Hz. This is only needed if Z is not given.
`n_out`	is an optional length for the output data. If n_out is not given, the output data length will be the input data length * fs_out/fs_in.

Value

Y is a sensor structure, vector or matrix of interpolated data with the same number of columns as X.

Examples

```
plott(X = list(harbor_seal$P), fsx = 5)
P_dec <- decdc(harbor_seal$P, 5)
P_interp <- interp2length(X = P_dec, Z = harbor_seal$A)
plott(X = list(P_interp$data), fsx = 1)
```

interp_nan — *Remove NAs from sensor data and return indices of (rows of) filled values*

Description

This is an internal function used by htrack

Usage

```
interp_nan(data)
```

Arguments

data	a data vector or matrix

Value

A list with entries data (the input data with NAs filled in) and k a logical vector indicating the position (if data was a vector) or rows (if data was a matrix) where NAs were filled in. Internal NAs are filled via linear interoplation, while leading and trailing ones are filled using the first following or last preceding good value.

Examples

```
A <- matrix(c(NA, NA, 3, 4, 5, 6, 7, 8, 9, 10, NA, NA, 13, 14, 15, 16, NA, NA), ncol = 2)
result <- interp_nan(A)
```

inv_axis — *Identify invariant axis in triaxial movement measurements.*

Description

This function processes tri-axial movement data (for example, from an accelerometer, magentometer or gyroscope) to identify the one axis that varies the least, i.e., the invariant axis.

Usage

```
inv_axis(data)
```

Arguments

data	The triaxial sensor measurement axis e.g., from on accelerometer or magnetometer. The frame and unit of A do not matter.

Details

Rotational and linear movement in some types of propulsion largely occur in 2 dimensions e.g., body rotation in cetacean caudal propulsion occurs around the animal's transverse axis. Likewise sychronized wing flaps in flight or pectoral swimming may generate acceleration in the longitudinal and dorso-ventral axes but much less in the transverse axis. This function identifies the direction of the axis that moves the least in a movement matrix.

Value

A list with two entries:

- `V` A 3x1 numeric vector defining the least varying axis in `data`. This vector is a direction vector so has a magnitude of 1 and is unit-less. The vector is defined in the same frame as A, so the first, second, and third entries correspond to the first, second and third columns of the data matrix, and axis orientation conventions are preserved.
- `q` The proportion of movement in the invariant axis. A small number (e.g., less than 0.05) implies that very little movement occurs in this axis and so the movement is largely planar (i.e., two-dimensional). If the fraction is much larger than 0.05, the motion in A is better described as three-dimensional. `q` is a proportion and so it is unitless.

Note

This function returns one invariant axis that applies to the entire input signal so it is important to choose a relevant sub-sample of movement data, A, to analyse.

Examples

```
s <- matrix(sin( 2 * pi * 0.1 * c(1:100)), ncol=1)
A <- s %*% c(0.9, -0.4, 0.3) + s^2 %*% c(0, 0.2, 0.1)
inv_axis_out <- inv_axis(A)
```

`julian_day` *Convert between dates and Julian day numbers.*

Description

This function is used to convert between dates and Julian day numbers. There are three different input arrangements, each of which returns a different output. For a description of the different input arrangements, see below.

Usage

```
julian_day(y = NULL, m = NULL, d = NULL)
```

Arguments

`y`	A single year or vector of years
`m`	A single month or vector of months
`d`	A single day or vector of days

Details

Possible input combinations: (n <- julian_day()) returns the Julian day number for today. (n = julian_day(y,d)) where y is a single year or a vector of years and d is a single day number or a vector of daynumbers, returns the date vector [year,month,day] for each year, day pair. (n = julian_day(y,m,d)) where y is a single year or a vector of years, m is a single month or vector of months, and d is a single month day or a vector of month days, returns the Julian day number for each year, month, day.

Value

See the description section for details on the return.

Examples

```
julian_day(y = 2016, d = 12, m = 10)
julian_day(y = 2016, 286)
```

`lalo2llf` *Convert latitude-longitude track points into a local level frame*

Description

Convert latitude-longitude track points into a local level frame

Usage

```
lalo2llf(trk, pt = NULL)
```

Arguments

`trk`	A data frame, two-column matrix, two-element vector of track points c(latitude, longitude) or sensor data structure.
`pt`	c(latitude, longitude) of the centre point of the local level frame. If pt is not given, the first point in the track will be used.

Value

A data frame with columns `northing` and `easting` of track points in the local level frame. Northing and easting are in metres. The axes of the frame are true (geographic) north and true east.

Note

This function assumes the track is on the surface of the geoid, and also uses a simple spherical model for the geoid. For more accurate conversion to a Cartesian frame, use spatial and mapping packages in Matlab/Octave.

Examples

```
coordinates <- matrix(c(
-122.4194, 37.7749,
-73.9352,  40.7306), nrow = 2, ncol = 2, byrow = TRUE)
lalo2llf(coordinates, c(15,19))
```

`load_nc` *Load a tag dataset from a netCDF file.*

Description

This function loads a tag dataset from a netCDF file (this is an archival file format supported by the tagtools package and suitable for submission to online data archives).

Usage

```
load_nc(file, which_vars = NULL)
```

Arguments

`file`	File name (and path, if necessary) of netCDF file to be read, as a quoted character string.
`which_vars`	(Optional) A list of quoted character strings giving the exact names of variables to be read in. Default is to read all variables present in the file. parameters should be read in.

Value

An `animaltag` object (a list) containing sensor and metadata structures. The item names in X will be the same as the names of the variables in the NetCDF file (plus an "info" one), e.g., if the file contains A and P, output object X will have fields XA, XP and X$info (the file metadata).

Examples

```
hold <- system.file("extdata","beaked_whale.nc", package = "tagtools", mustWork = TRUE)
load_nc(hold)
```

m2h *Heading from accelerometer and magnetometer data*

Description

This function is used to compute the heading, field intensity, and the inclination angle by gimballing the magnetic field measurement matrix with the pitch and roll estimated from the accelerometer matrix.

Usage

```
m2h(M, A, sampling_rate = NULL, fc = NULL)
```

Arguments

`M`	A sensor data structure or matrix, M = [mx,my,mz] in any consistent unit (e.g., in uT or Gauss) or magnetometer sensor list (e.g., from readtag.R).
`A`	A sensor data structure or matrix with columns [ax ay az] or acceleration sensor list (e.g., from readtag.R). Acceleration can be in any consistent unit, e.g., g or m/s^2.
`sampling_rate`	(optional) The sampling rate of the sensor data in Hz (samples per second). This is only needed if filtering is required. If `A` and `M` are sensor data lists, then sampling_rate is obtained from them.
`fc`	(optional) The cut-off frequency of a low-pass filter to apply to A and M before computing heading. The filter cut-off frequency is with in Hertz. The filter length is 4*sampling_rate/fc. Filtering adds no group delay. If fc is not specified, no filtering is performed.

Value

A list with 3 elements:

- **h:** The heading in radians in the same frame as M. The heading is with respect to magnetic north (i.e., the north vector of the navigation frame) and so must be corrected for declination.
- **v:** The estimated magnetic field intensity in the same units as M. This is computed by taking the 2-norm of M, after filtering (if any filtering was specified).
- **incl:** The estimated magnetic field inclination angle (i.e., the angle with respect to the horizontal plane) in radians. By convention, a field vector pointing below the horizon has a positive inclination angle. See note in the function if using incl.

Note

Output sampling rate is the same as the input sampling rate (i.e. h, v, and incl are estimated with the same sampling rate as M and A and so are each nx1 vectors).

Frame: This function assumes a [north,east,up] navigation frame and a [forward,right,up] local frame. North and east are magnetic, not true. In these frames a positive heading is a clockwise rotation around the z-axis.

The heading is computed with respect to the frame of M and is the magnetic heading NOT the true heading. M and A must have the same sampling rate, frame, and number of rows.

See Also

a2pr

Examples

```
m2h_out <- m2h(M = matrix(c(22, -24, 14), nrow = 1),
                        A = matrix(c(-0.3, 0.52, 0.8), nrow = 1))
```

make_info *Make an info structure with tag metadata*

Description

This function allows the user to generate a "skeleton" info structure for a tag deployment, with some common pieces of metadata filled in. Additional information can then be added manually or using a custom script before saving this `info` as part of a netCDF file.

Usage

```
make_info(depid, tagtype, species, owner)
```

Arguments

`depid`	Deployment id string for this tag record
`tagtype`	String identifying the tag type, for example 'dtag', 'cats', 'mk10', ...
`species`	(optional) 2-letter string with the first letters of the species binomial
`owner`	(optional) String with initials of the tag data owner

Value

A list containing metadata for a tag deployment. It's recommended to name this output "info" and save it as part of a netCDF tag data archive file (along with the tag sensor data).

Examples

```
info <- make_info("d4_template", "dtag4", "zc", "sdr")
```

make_specgram *Plot a spectrogram with default settings*

Description

This is a wrapper function for specgram to draw a spectrogram with the same input argument names and defaults as the tag tools Matlab/Octave function make_specgram.

Usage

```
make_specgram(
  x,
  nfft = 256,
  fs = 2,
  window = signal::hanning(nfft),
  noverlap = length(window)/2,
  draw_plot = TRUE
)
```

Arguments

`x`	The input signal
`nfft`	specifies the number of frequency points used to calculate the discrete Fourier transforms.
`fs`	The sampling frequency in Hz
`window`	If you specify a scalar for `window`, make_specgram uses a Hanning window of that length. `window` must have length smaller than or equal to `nfft` and greater than `noverlap`.
`noverlap`	The number of samples the sections of `x` overlap.
`draw_plot`	(logical) Should a plot be drawn? Defaults to TRUE.

Value

if `draw_plot` is TRUE, a plot is produced. If it is FALSE, a list is returned, with as follows. Each element is a matrix and all three matrices are the same size.

- `s,` A matrix of spectrogram values of signal x in dB.
- `f,` Frequencies (Hz) corresponding to the rows of s
- `t,` Time indices corresponding to the columns of s

Examples

```
x <- signal::chirp(seq(from = 0, by = 0.001, to = 2),
  f0 = 0,
  t1 = 2,
  f1 = 500
)
fs <- 2
nfft <- 256
numoverlap <- 128
window <- signal::hanning(nfft)
S <- make_specgram(x, nfft, fs, window, numoverlap, draw_plot = FALSE)
```

`mean_absorption` *Calculate the mean absorption in salt water*

Description

This function is used to calculate the mean absorption in salt water over a frequency range.

Usage

```
mean_absorption(freq, r, depth, Ttab = NULL)
```

Arguments

`freq`	Specifies the frequency range, freq = c(fmin, fmax) in Hz. For a single frequency, use a scalar value for freq.
`r`	The path (slant) length in metres.
`depth`	The depths covered by the path. This can be a single value for a horizontal path or a two component vector i.e., depth = c(dmax, dmin) for a path that extends between two depths.
`Ttab`	(optional) The temperature (a scalar) in degrees C or specifies a temperature profile Ttab = c(depth, tempr) where depth and tempr are equal-sized column vectors. Default value is an isothermal profile of 13 degrees.

Value

The mean sound absorption over the path in dB.

Note

After Kinsler and Frey pp. 159-160.

Examples

```
mean_absorption(c(25e3, 60e3), 1000, c(0, 700))
```

`median_filter` *Computes the nth-order median filter*

Description

This function computes the nth-order median filter each column of X. The filter output is the median of each consecutive group of n samples. This is useful for removing occasional outliers in data that is otherwise fairly smooth. This makes it appropriate for pressure, temperature and magnetometer data (amongst other sensors) but not so suitable for acceleration which can be highly dynamic. The filter does not introduce delay. The start and end values, i.e., within n samples of the start or end of the input data, are computed with decreasing order median filters unless noend = TRUE. If noend = TRUE, start and end values are taken directly from X without short median filters.

Usage

```
median_filter(X, n, noend = TRUE)
```

Arguments

`X`	A sensor list or a vector or matrix. If there are multiple columns in the data, each column is treated as a separate signal to be filtered.
`n`	The filter length. If an even n is given, it is automatically incremented to make it odd. This ensures that the median is well-defined (the median of an even length vector is usually defined as the mean of the middle two points but may differ in different programmes). Note that a short n (e.g., 3 or 5) is usually sufficient and that processing will be very slow if n is large.
`noend`	If TRUE (the default), then start and end values are taken directly from X without short median filters.

Value

The output of the filter. It has the same size as S and has the same sampling rate and units as X. If X is a sensor list, the return will also be.

Examples

```
v <- matrix(c(1, 3, 4, 4, 20, -10, 5, 6, 6, 7), ncol = 1)
w <- median_filter(v, n = 3)
```

merge_fields *Merge the fields of two lists*

Description

This function is used to merge the fields of two lists. If there are duplicate fields, the fields in s1 are taken.

Usage

```
merge_fields(s1, s2)
```

Arguments

s1	Arbitrary list e.g., containing metadata or settings.
s2	Arbitrary list e.g., containing metadata or settings.

Value

A list containing all of the fields in s1 and s2

Examples

```
s1 <- list(a = 1, b = c(2, 3, 4))
s2 <- list(b = 3, c = "cat")
s <- merge_fields(s1, s2)
```

metadata_editor *Edits a html file from given csv.*

Description

Takes data from csv, and edits a default or given html to fill in data from the csv. HTML must be tagmetadata.html or variations, csv should only contain metadata of tag.

Usage

```
metadata_editor(
  masterHTML = system.file("extdata", "tagmetadata.html", package = "tagtools"),
  csvfilename = system.file("extdata", "blank_template.csv", package = "tagtools")
)
```

Arguments

masterHTML	default masterHTML is located in the package, or can be changed according to user input.
csvfilename	file name of csv to be used for editing the HTML

Value

A "dynamic tagmetadata.html" which is the masterHTML with changes from csv. This file is written to the current working directory, and also opened for editing by the user.

`msa` *Compute MSA*

Description

This function is used to compute the Minimum Specific Acceleration (MSA). This is the absolute value of the norm of the acceleration minus 1 g, i.e., the amount that the acceleration differs from the gravity value. This is always equal to or less than the actual specific acceleration if A is correctly calibrated.

Usage

```
msa(A, ref)
```

Arguments

`A`	An nx3 acceleration matrix with columns [ax ay az], or a tag sensor data list containing acceleration data. Acceleration can be in any consistent unit, e.g., g or m/s^2. A can be in any frame as the MSA is rotation independent.
`ref`	The gravitational field strength in the same units as A. This is not needed if A is a sensor structure. If A is a matrix, the default value is 9.81 which assumes that A is in m/s^2. Use ref = 1 if the unit of A is g.

Details

Possible input combinations: msa(A) if A is a list, msa(A,ref) if A is a matrix.

Value

A column vector of MSA with the same number of rows as A, or a tag sensor data list (output matches input). m has the same units as A.

Note

See Simon et al. (2012) Journal of Experimental Biology, 215:3786-3798.

See Also

`odba`, `njerk`

Examples

```
sampleMatrix <- matrix(c(1, -0.5, 0.1, 0.8, -0.2, 0.6, 0.5, -0.9, -0.7),
  byrow = TRUE, nrow = 3, ncol = 3
)
msa(A = sampleMatrix, ref = 1)
```

m_dist *Calculate Mahalanobis distance*

Description

This function is used to calculate the Mahalanobis distance for a multivariate time series.

Usage

```
m_dist(
  data,
  sampling_rate,
  smoothDur,
  overlap,
  consec,
  cumSum,
  expStart,
  expEnd,
  baselineStart,
  baselineEnd,
  BL_COV
)
```

Arguments

`data`	A data frame or matrix with one row for each time point. Note that the Mahalanobis distance calculation should be carried out on continuous data only, so if your data contain logical, factor or character data, proceed at your own risk...errors (or at least meaningless results) will probably ensue.
`sampling_rate`	The sampling rate in Hz (data should be regularly sampled). If not specified it will be assumed to be 1 Hz.
`smoothDur`	The length, in minutes, of the window to use for calculation of "comparison" values. If not specified or zero, there will be no smoothing (a distance will be calculated for each data observation).
`overlap`	The amount of overlap, in minutes, between consecutive "comparison" windows. smooth_dur - overlap will give the time resolution of the resulting distance time series. If not specified or zero, there will be no overlap. Overlap will also be set to zero if smoothDur is unspecified or zero.

`consec`	Logical. If consec = TRUE, then the calculated distances are between consecutive windows of duration smoothDur, sliding forward over the data set by a time step of (smoothDur-overlap) minutes. If TRUE, baselineStart and baselineEnd inputs will be used to define the period used to calculate the data covariance matrix. Default is consec = FALSE.
`cumSum`	Logical. If cum_sum = TRUE, then output will be the cumulative sum of the calculated distances, rather than the distances themselves. Default is cum_sum = FALSE.
`expStart`	Start times (in seconds since start of the data set) of the experimental exposure period(s).
`expEnd`	End times (in seconds since start of the data set) of the experimental exposure period(s). If either or both of exp_start and exp_end are missing, the distance will be calculated over whole dataset and full dataset will be assumed to be baseline.
`baselineStart`	Start time (in seconds since start of the data set) of the baseline period (the mean data values for this period will be used as the 'control' to which all "comparison" data points (or windows) will be compared. if not specified, it will be assumed to be 0 (start of record).
`baselineEnd`	End time (in seconds since start of the data set) of the baseline period. If not specified, the entire data set will be used (baseline_end will be the last sampled time-point in the data set).
`BL_COV`	Logical. If BL_COV= TRUE, then a covariance matrix using all data in baseline period will be used for calculating the Mahalanobis distance. Default is BL_COV = FALSE.

Value

Data frame containing results: variable seconds is times in seconds since start of dataset, at which Mahalanobis distances are reported. If a smoothDur was applied, then the reported times will be the start times of each "comparison" window. Variable dist is the Mahalanobis distances between the specified baseline period and the specified "comparison" periods.

Examples

```
BW <- beaked_whale
m_dist_result <- m_dist(BW$A$data, BW$A$sampling_rate)
```

`njerk` *Compute the norm-jerk*

Description

This function is used to compute the norm-jerk from triaxial acceleration data.

Usage

```
njerk(A, sampling_rate)
```

Arguments

`A`	A tag sensor data list or a nx3 acceleration matrix with columns [ax ay az]. Acceleration can be in any consistent unit, e.g., g or m/s^2. A can be in any frame as the norm-jerk is rotation independent. A must have at least 2 rows (i.e., n>=2).
`sampling_rate`	The sampling rate in Hz of the acceleration signals. This is used to estimate the differential by a first-order difference.

Value

The norm-jerk from triaxial acceleration data in the form of a column vector with the same number of rows as in A, or a tag sensor data structure (if the input A was one). The norm-jerk is ||dA/dt||, where ||x|| is the 2-norm of x, i.e., the square-root of the sum of the squares of each axis. If the unit of A is m/s^2, the norm-jerk has unit m/s^3. If the unit of A is g, the norm-jerk has unit g/s. As j is the norm of the jerk, it is always positive or zero (if the acceleration is constant). The final value in j is always 0 because the last finite difference cannot be calculated.

See Also

`msa`, `odba`

Examples

```
sampleMatrix <- matrix(c(1, 2, 3, 2, 2, 4, 1, -2, 4, 4, 4, 4), byrow = TRUE, nrow = 4, ncol = 3)
norm_jerk <- njerk(A = sampleMatrix, sampling_rate = 5)
```

`norm2` *Compute the row-wise vector norm*

Description

This function is used to compute the row-wise vector norm of X if X is a matrix. If X is a vector (row or column), v is the vector norm.

Usage

```
norm2(X)
```

Arguments

`X`	A data structure, vector or matrix.

Value

The row-wise vector-norm of matrix X, i.e., the square-root of the sum of the squares for each row. If X is a vector (row or column), v is the vector norm and norm2() is equivalent to the built-in function norm(). But if X is a matrix e.g., a triaxial accelerometer or magnetometer matrix, norm() gives the overall norm of the matrix whereas norm2() gives the vector norm of each row (i.e., the field strength in the case of a magnetometer matrix).

Examples

```
sampleMatrix <- matrix(c(0.2, 0.4, -0.7, -0.3, 1.1, 0.1), byrow = TRUE, nrow = 2, ncol = 3)
norm2(X = sampleMatrix)
```

`ocdr` *Estimate the forward speed*

Description

This function is used to estimate the forward speed of a flying or diving animal by first computing the altitude or depth-rate (i.e., the first differential of the pressure in meters) and then correcting for the pitch angle. This is called the Orientation Corrected Depth Rate. There are two major assumptions in this method: (i) the animal moves in the direction of its longitudinal axis, and (ii) the frame of A coincides with the animal's axes.

Usage

```
ocdr(p, A, sampling_rate, fc, plim)
```

Arguments

`p`	The depth or altitude vector (a regularly sampled time series) or depth or altitude sensors list in meters, sampled at sampling_rate Hz.
`A`	The nx3 acceleration matrix with columns [ax ay az] or acceleration sensor list (e.g., from readtag.R). Acceleration can be in any consistent unit, e.g., g or m/s^2. A must have the same number of rows as p.
`sampling_rate`	The sampling rate of p and A in Hz (samples per second).
`fc`	(optional) Specifies the cut-off frequency of a low-pass filter to apply to p after computing depth-rate and to A before computing pitch. The filter cut-off frequency is in Hz. The filter length is 4*sampling_rate/fc. Filtering adds no group delay. If fc is empty or not given, the default value of 0.2 Hz (i.e., a 5 second time constant) is used.
`plim`	(optional) Specifies the minimum pitch angle in radians at which speed can be computed. Errors in speed estimation using this method increase strongly at low pitch angles. To avoid estimates with poor accuracy being used in later analyses, speed estimates at low pitch angles are replaced by NaN (not-a-number). The default threshold for this is 20 degrees.

Details

Possible input combinations: ocdr(p,A) if p and A are lists, ocdr(p,A,fc = fc) if p and A are lists, ocdr(p,A,fc = fc,plim = plim) if p and A are lists, ocdr(p,A,sampling_rate) if p and A are vectors/matrices, ocdr(p,A,sampling_rate,fc) if p and A are vectors/matrices, ocdr(p,A,sampling_rate,fc,plim) if p and A are vectors/matrices.

Value

The forward speed estimate in m/s

Note

Output sampling rate is the same as the input sampling rate so s has the same size as p.

Frame: This function assumes a [north,east,up] navigation frame and a [forward,right,up] local frame. In these frames, a positive pitch angle is an anti-clockwise rotation around the y-axis. A descending animal will have a negative pitch angle.

Examples

```
HS <- harbor_seal
s <- ocdr(p = HS$P$data, A = HS$A$data, sampling_rate = HS$P$sampling_rate, fc = NULL, plim = NULL)
speed <- list(s = s)
plott(speed, HS$P$sampling_rate)
```

`odba` *Compute ODBA*

Description

This function is used to compute the 'Overall Dynamic Body Acceleration' sensu Wilson et al. 2006. ODBA is the norm of the high-pass-filtered acceleration. Several methods for computing ODBA are in use which differ by which norm and which filter are used. In the Wilson paper, the 1-norm and a rectangular window (moving average) filter are used. The moving average is subtracted from the input accelerations to implement a high-pass filter. The 2-norm may be preferable if the tag orientation is unknown or may change and this is termed VeDBA. A tapered symmetric FIR filter gives more efficient high-pass filtering compared to the rectangular window method and avoids lobes in the response.

Usage

```
odba(A, sampling_rate = NULL, fh = NULL, method = "fir", n = NULL)
```

Arguments

`A`	A tag sensor data list containing tri-axial acceleration data or an nx3 acceleration matrix with columns [ax ay az]. Acceleration can be in any consistent unit, e.g., g or m/s^2. A can be in any frame but the result depends on the method used to compute ODBA. The default method and VeDBA method are rotation independent and so give the same result irrespective of the frame of A. The 1-norm method has a more complex dependency on frame.
`sampling_rate`	The sampling rate in Hz of the acceleration signals. Required for 'fir' method if A is not a tag sensor data list.
`fh`	The high-pass filter cut-off frequency in Hz. This should be chosen to be about half of the stroking rate for the animal (e.g., using dsf.R). Required for the default 'fir' method.
`method`	A character containing either 'wilson' or 'vedba' or 'fir'. This determines the ethod by which the ODBA is calculated. The default method is 'fir'.
`n`	The rectangular window (moving average) length in samples. This is only needed if using the classic ODBA and VeDBA forms (methods 'wilson' and 'vedba').

Value

A column vector of ODBA with the same number of rows as A. e has the same units as A.

Note

If applying the default (FIR filtering) method to calculate odba, use the inputs A, sampling_rate, and fh. When applying the 'vedba' or 'wilson' method, use the inputs A, n, and method.

Examples

```
BW <- beaked_whale
e <- odba(A = BW$A$data, sampling_rate = BW$A$sampling_rate, fh = 0.05)
ba <- list(e = e)
plott(ba, BW$A$sampling_rate)
```

`plott` *Plot tag data time series*

Description

Plot time series in a single or multi-paneled figure. This is useful, for example, for comparing measurements across different sensors in an animaltag data object. The time axis is automatically displayed in seconds, minutes, hours, or days according to the span of the data.

Usage

```
plott(
  X,
  fsx = NULL,
  r = FALSE,
  offset = 0,
  date_time_axis = FALSE,
  recording_start = NULL,
  panel_heights = rep.int(1, length(X)),
  panel_labels = names(X),
  line_colors,
  interactive = FALSE,
  par_opts,
  ...
)
```

Arguments

`X`	List whose elements are either lists (containing data and metadata) or vectors/matrices of time series data. See details.
`fsx`	(Optional) A numeric vector whose length matches the number of sensor data streams (list elements) in X. (If shorter, `fsx` will be recycled to the appropriate length). `fsx` gives the sampling rate in Hz for each data object. Sampling rates are not needed when the data object(s) `X` are list(s) that contain sampling rate information – and beware, because `fsx` (if given) will override sensor metadata.
`r`	(Optional) Logical. Should the direction of the y-axis be flipped? Default is FALSE. If `r` is of length one (or shorter than the number of sensor data streams in X) it will be recycled to match the number of sensor data streams. Reversed y-axes are useful, for example, for plotting dive profiles which match the physical situation (with greater depths lower in the display). If the name of a sensor list is "P" or contains the word "depth", it will automatically be reversed.
`offset`	(Optional) A vector of offsets, in seconds, between the start of each sensor data stream and the start of the first one. For example, if acceleration data collection started and then depth data collection commenced 436 seconds later, then the `offset` for the depth data would be 436.
`date_time_axis`	(Optional) Logical. Should the x-axis units be date-times rather than time-since-start-of-recording? Ignored if `recording_start` is not provided and `X` does not contain metadata on recording start time. Default is FALSE.
`recording_start`	(Optional) The start time of the tag recording as a `POSIXct` object. If provided, the time axis will show calendar date/times; if not, it will show days/hours/minutes/seconds (as appropriate) since time 0 = the start of recording. If a character string is provided it will be coerced to POSIXct with `as.POSIXct`.
`panel_heights`	(Optional) A vector of relative or absolute heights for the different panels (one entry for each sensor data stream in `X`). Default is equal-height panels. If `panel_heights` is a numeric vector, it is interpreted as relative panel heights. To specify absolute panel heights in centimeters using `lcm` (see help for `layout`).

`panel_labels`	(Optional) A list of y-axis labels for the panels. Defaults to names(X).
`line_colors`	(Optional) A list of colors for lines for multivariate data streams (for example, if a panel plots tri-axial acceleration, it will have three lines – their line colors will be the first three in this list). May be specified in any specification R understands for colors. Defaults to c("#000000", "#009E73", "#9ad0f3", "#0072B2", "#e79f00", "#D55E00")
`interactive`	(Optional) Should an interactive figure (allowing zoom/pan/etc.) be produced? Default is FALSE. Interactive plotting requires the zoom package for its `zm` function.
`par_opts`	(Optional) A list of options to be passed to `par` before plotting. Default is mar=c(1,5,0,0), oma=c(2,0,2,1), las=1, lwd=1, cex=0.8.
`...`	Additional arguments to be passed to `plot`.

Details

If the input data X is an `animaltag` object, then all sensor variables in the object will be plotted. To plot only selected sensors from the `animaltag` object `my_tag`, for example, the input X=list(my_tag$A, my_tag$M) would plot just the accelerometer and magnetometer data. If possible, the plot will have

Value

A plot of time-series data

Note

This is a flexible plotting tool which can be used to display and explore sensor data with different sampling rates on a uniform time grid.

Examples

```
HS <- harbor_seal
list <- list(depth = HS$P$data, A = HS$A$data)
plott(list, HS$P$sampling_rate, r = c(TRUE, FALSE))
```

`prh_predictor1` *Predict the tag position on a diving animal from depth and acceleration data*

Description

Predict the tag position on a diving animal parameterized by p0, r0, and h0, the canonical angles between the principal axes of the tag and the animal. The tag orientation on the animal can change with time and this function provides a way to estimate the orientation at the start and end of each suitable dive. The function critically assumes that the animal rests horizontally at the surface (at least on average) and dives steeply away from the surface without an initial roll. If ascents are processed, there must also be no roll in the last seconds of the ascents. See prh_predictor2 for a method more suitable to animals that make short dives between respirations. The function provides a graphical interface showing the estimated tag-to-animal orientation throughout the deployment. Follow the directions above the top panel of the figure to edit or delete an orientation estimate.

Usage

```
prh_predictor1(P, A, sampling_rate = NULL, TH = 100, DIR = "descent")
```

Arguments

`P`	is a dive depth vector or sensor structure with units of m H2O.
`A`	is an acceleration matrix or sensor structure with columns ax, ay, and az. Acceleration can be in any consistent unit, e.g., g or m/s^2, and must have the same sampling rate as P.
`sampling_rate`	is the sampling rate of the sensor data in Hz (samples per second). This is only needed if neither A nor M are sensor structures.
`TH`	is an optional minimum dive depth threshold (default is 100m). Only the descents at the start of dives deeper than TH will be analysed (and the ascents at the end of dives deeper than TH if ALL is true).
`DIR`	is an optional dive direction constraint. The default (DIR = 'descent') is to only analyse descents as these tend to give better results. But if DIR = 'both', both descents and ascents are analysed.

Value

PRH, a data frame with columns `cue` `p0`, `r0`, `h0`, and `q` with a row for each dive edge analysed. `cue` is the time in second-since-tag-start of the dive edge analysed. `p0`, `r0`, and `h0` are the deduced tag orientation angles in radians. `q` is the quality indicator with a low value (near 0, e.g., <0.05) indicating that the data fit more consistently with the assumptions of the method.

See Also

prh_predictor2, tag2animal

`prh_predictor2` *Predict the tag position on a diving animal from depth and acceleration data*

Description

Predict the tag position on a diving animal parametrized by p0, r0, and h0, the canonical angles between the principal axes of the tag and the animal. The tag orientation on the animal can change with time and this function provides a way to estimate the orientation at the start and end of each suitable dive. The function critically assumes that the animal makes a sequence of short dives between respirations and that the animal remains upright (i.e., does not roll) during these shallow dives. See prh_predictor1 for a method more suitable to animals that rest horizontally at the surface. The function provides a graphical interface showing the estimated tag-to-animal orientation throughout the deployment. Follow the directions above the top panel of the figure to edit or delete an orientation estimate. The function provides a graphical interface showing the estimated tag-to-animal orientation throughout the deployment. Follow the directions above the top panel of the figure to edit or delete an orientation estimate.

Usage

```
prh_predictor2(P, A, sampling_rate = NULL, MAXD = 10)
```

Arguments

`P`	is a dive depth vector or sensor structure with units of m H2O.
`A`	is an acceleration matrix or sensor structure with columns ax, ay, and az. Acceleration can be in any consistent unit, e.g., g or m/s^2, and must have the same sampling rate as P.
`sampling_rate`	is the sampling rate of the sensor data in Hz (samples per second). This is only needed if neither A nor M are sensor structures.
`MAXD`	is the optional maximum depth of near-surface dives. The default value is 10 m. This is used to find contiguous surface intervals suitable for analysis.

Value

PRH, a data frame with columns `cue` `p0`, `r0`, `h0`, and `q` with a row for each dive edge analysed. `cue` is the time in second-since-tag-start of the dive edge analysed. `p0`, `r0`, and `h0` are the deduced tag orientation angles in radians. `q` is the quality indicator with a low value (near 0, e.g., <0.05) indicating that the data fit more consistently with the assumptions of the method.

See Also

prh_predictor1, tag2animal

`ptrack` *Estimate simple dead-reckoned track*

Description

This function is used to estimate the simple dead-reckoned track (pseudo-track) based on speed and bodypointing angle.

Usage

```
ptrack(A, M, s, sampling_rate = NULL, fc = 0.2, return_pe = FALSE)
```

Arguments

`A`	An nx3 acceleration matrix with columns [ax ay az] or acceleration sensor list. Acceleration can be in any consistent unit, e.g., g or m/s^2.
`M`	The magnetometer signal matrix, M = [mx,my,mz] in any consistent unit (e.g., in uT or Gauss) or magnetometer sensor list. A and M must have the same size (and so are both measured at the same sampling rate).
`s`	The forward speed of the animal in m/s. s can be a single number meaning that the animal is assumed to travel at a constant speed. s can also be a vector with the same number of rows as A and M, e.g., generated by speed_from_depth().
`sampling_rate`	The sampling rate of the sensor data in Hz (samples per second). This input will be ignored if A and/or M are sensor lists, in which case the sampling rate will be extracted from them.
`fc`	(optional) The cut-off frequency of a low-pass filter to apply to A and M before computing bodypointing angle. The filter cut-off frequency is in Hz. The filter length is 4*sampling_rate/fc. Filtering adds no group delay. If fc is empty or not given, the default value of 0.2 Hz (i.e., a 5 second time constant) is used.
`return_pe`	Logical. If return_pe is TRUE, the estimated depth or altitude predicted will be returned with the estimated track. Default is FALSE.

Value

The estimated track in a local level frame. The track is defined as meters of northward and eastward movement (variables 'northing' and 'easting' in the output data frame) relative to the animal's position at the start of the measurements (which is defined as [0,0]). The track sampling rate is the same as for the input data and so each row of track object defines the track coordinates at times 0,1/sampling_rate,2/sampling_rate,... relative to the start time of the measurements. OR, if return_pe = TRUE, this function returns the above value and the estimated depth or altitude predicted from the speed and pitch angle. This can be compared against the measured depth/altitude to assess errors in the dead-reckoned track. Note that even if pe matches the observed depth, this does not guarantee that the track is accurate.

Note

Frame: This function assumes a [north,east,up] navigation frame and a [forward,right,up] local frame. Both A and M must be rotated if needed to match the animal's cardinal axes otherwise the track will not be meaningful.

CAUTION: dead-reckoned tracks are usually very inaccurate. They are useful to get an idea of HOW animals move rather than WHERE they go. Few animals probably travel in exactly the direction of their longitudinal axis and anyway measuring the precise orientation of the longitudinal axis of a non-rigid animal is fraught with error. Moreover, if there is net flow in the medium, the animal will be affected by the flow in addition to its autonomous movement. For swimming animals this can lead to substantial errors. The forward speed is assumed to be with respect to the medium so the track derived here is NOT the 'track-made-good', i.e., the geographic movement of the animal. It estimates the movement of the animal with respect to the medium. There are numerous other sources of error so use at your own risk!

See Also

htrack, fit_tracks, track3D

Examples

```
BW <- beaked_whale
list <- ptrack(A = BW$A$data, M = BW$M$data, s = 3,
sampling_rate = BW$A$sampling_rate, fc = NULL,
return_pe = TRUE)
plot(list$track$easting, list$track$northing, xlab = "Easting, m", ylab = "Northing, m")
```

---

read_cats *Read a CATS data file and convert to .nc*

Description

Read a .csv file with data from a CATS tag deployment, including associated metadata, and store the resulting data in a .nc file.

Usage

```
read_cats(fname, depid)
```

Arguments

fname	is the file name of the CATS CSV file including the complete path name if the file is not in the current working directory or in a directory on the path. The .csv suffix is optional.
depid	is a string containing the deployment identification code assigned to this deployment, for example, 'mn12_186a'.

Value

A string (constructed by: 'depid_raw.nc'; for example, 'mn12_186a_raw.nc') containing the file name of the netCDF (.nc) file in which the output has been saved. This function generates a netCDF file in the current working directory containing the tag data variables, including:

- A, Accelerometer data structure
- M, Magnetometer data structure
- temp, Temperature sensor data structure
- info Information structure for the deployment

Note

CATS loggers can produce very large csv files which are slow to process. This function is (somewhat) optimised for speed and memory use so will tolerate large files. But processing could ppube slow.

Examples

```
## Not run:
nc_filename <- read_cats("my_cats_file.csv", "my_cats_deplyment_name")
load_nc("my_cats_deployment_name_raw.nc")

## End(Not run)
```

`read_cats_csv` *Read a CSV file with sensor data from a CATS tag*

Description

Read in data from a CATS tag deployment (stored in a .csv file). This function is usable by itself but is more normally called by `read_cats` which handles metadata and creates a NetCDF file.

Usage

```
read_cats_csv(fname, max_samps = Inf, skip_samps = 0)
```

Arguments

`fname`	is the file name of the CATS CSV file including the complete path name if the file is not in the current working directory or in a directory on the path. The .csv suffix is optional.
`max_samps`	is optional and is used to limit reading to a maximum number of samples (rows) per sensor. This is useful to read in a part of a very large file for testing. If max_samps is not given, the entire file is read.
`skip_samps`	Number of lines of data to skip (excluding header) before starting to read in data. Defaults to 0 (start at the beginning of the file), but can be used to read in a part of a file, or to read in and process a large file in chunks.

Value

A tibble data frame containing the data read from the file. The column names are taken from the first line of the CSV file and include units and axis. Some columns may be empty (if for example, a tag did not record data from a certain sensor type).

Note

CATS csv files can be extremely large; perhaps too large to read the entire file into memory at once and work with it.

rotate_data *Rotate data.*

Description

Rotate a numeric vector (for rotation_test, this will be a set of event times). "Rotating" the vector entails advancing all values by a random increment, then subtracting the maximum expected value from all rotated entries that exceed that maximum. This is a utility function used by `rotation_test`, but advanced users may wish to use it directly to carry out non-standard rotation tests.

Usage

```
rotate_data(event_times, full_period)
```

Arguments

`event_times`	A vector of the times of events. Times can be given in any format. If `event_times` should not be sorted prior to analysis (for example, if times are given in hours of the day and the times in the dataset span several days), be sure to specify `skip_sort=TRUE`.
`full_period`	A length two vector giving the start and end times of the full period during which events in event_times might have occurred. If missing, default is range(`event_times`).

Details

The rotation test was applied in Miller et al. 2004 and detailed in DeRuiter and Solow 2008. This test is a variation on standard randomization or permutation tests that is appropriate for time-series of non-independent events (for example, time series of behavioral events that tend to occur in clusters). This implementation of the rotation test compares a test statistic (some summary of an "experimental" time-period) to its expected value during non-experimental periods. Instead of resampling random subsets of observations from the original dataset, the rotation test samples many contiguous blocks from the original data, each the same duration as the experimental period. The summary statistic, computed for these "rotated" samples, provides a distribution to which the test statistic from the data can be compared.

Value

A vector of numeric values the same length as `event_times` generated by rotating the event times by a random amount

Examples

```
my_events <- 1500 * stats::runif(10) # 10 events at "times" between 0 and 1500
my_events
rotated_events <- rotate_data(my_events, full_period = c(0, 1500))
rotated_events
```

rotate_vecs *Rotate triaxial vector measurements*

Description

This function is used to rotate triaxial vector measurements from one frame to another.

Usage

```
rotate_vecs(V, Q)
```

Arguments

V	is a tag data structure, a 3-element vector or a 3-column matrix of vector measurements for example V could be from an accelerometer or magnetometer.
Q	is the rotation matrix. If Q is a single 3x3 matrix, the same rotation is applied to all vectors in V. If Q is a 3x3xn matrix where n is the number of rows in V, a different transformation given by Q[,, k] is applied to each row of V.

Value

The rotated vector or matrix with the same size as the input V.

Note

Frame: This function makes no assumptions about frame.

Examples

```
x <- (pi / 180) * matrix(c(25, -60, 33), ncol = 3)
Q <- euler2rotmat(x[, 1], x[, 2], x[, 3])
V <- rotate_vecs(c(0.77, -0.6, -0.22), Q)
```

rotation_test *Carry out a rotation randomization test.*

Description

Carry out a rotation test (as applied in Miller et al. 2004 and detailed in DeRuiter and Solow 2008). This test is a variation on standard randomization or permutation tests that is appropriate for time-series of non-independent events (for example, time series of behavioral events that tend to occur in clusters).

Usage

```
rotation_test(
  event_times,
  exp_period,
  full_period = range(event_times, na.rm = TRUE),
  n_rot = 10000,
  ts_fun = length,
  skip_sort = FALSE,
  conf_level = 0.95,
  return_rot_stats = FALSE,
  ...
)
```

Arguments

`event_times`	A vector of the times of events. Times can be given in any format. If `event_times` should not be sorted prior to analysis (for example, if times are given in hours of the day and the times in the dataset span several days), be sure to specify `skip_sort=TRUE`.
`exp_period`	A two-column vector, matrix, or data frame specifying the start and end times of the "experimental" period for the test. If a matrix or data frame is provided, one column should be start time(s) and the other end time(s). Note that all data that falls into any experimental period will be concatenated and passed to `ts_fun`. If finer control is desired, consider writing your own test using the underlying function `rotate_data`.
`full_period`	A length two vector giving the start and end times of the full period during which events in event_times might have occurred. If missing, default is range(`event_times`).
`n_rot`	Number of rotations (randomizations) to carry out. Default is `n_rot=10000`.
`ts_fun`	A function to compute the test statistic. Input provided to this function will be the times of events that occur during the "experimental" period. The default function is `length` - in other words, the default test statistis is the number of events that happen during the experimental period.
`skip_sort`	Logical. Should times be sorted in ascending order? Default is `skip_sort=FALSE`.
`conf_level`	Confidence level to be used for the bootstrap CI calculation, specified as a proportion. (default is `conf_level=0.95`, or 95% confidence.)
`return_rot_stats`	Logical. Should output include the test statistics computed for each rotation of the data? Default is `return_rot_stats=FALSE`.
`...`	Additional inputs to be passed to `ts_fun`

Details

This implementation of the rotation test compares a test statistic (some summary of an "experimental" time-period) to its expected value during non-experimental periods. Instead of resampling random subsets of observations from the original dataset, the rotation test samples many contiguous blocks from the original data, each the same duration as the experimental period. The summary statistic, computed for these "rotated" samples, provides a distribution to which the test statistic from the data can be compared.

Value

A list containing the following components:

- **result**, A one-row data frame with rows:
  - **statistic:** Test statistic (from original data)
  - **p_value:** P-value of the test (2-sided)
  - **n_rot:** Number of rotations
  - **CI_low:** Lower bound on rotation-resampling percentile-based confidence interval
  - **CI_up:** Upper bound on rotation-resampling percentile-based confidence interval
  - **conf_level:** Confidence level, as a proportion
- **rot_stats** (If `return_rot_stats` is TRUE), a vector of `n_rot` statistics from the rotated datasets

References

Miller, P. J. O., Shapiro, A. D., Tyack, P. L. and Solow, A. R. (2004). Call-type matching in vocal exchanges of free-ranging resident killer whales, Orcinus orca. Anim. Behav. 67, 1099–1107.

DeRuiter, S. L. and Solow, A. R. (2008). A rotation test for behavioural point-process data. Anim. Behav. 76, 1103–1452.

See Also

Advanced users seeking more flexibility may want to use the underlying function `rotate_data` to carry out customized rotation resampling. `rotate_data` generates one rotated dataset from `event_times` and `exp_period`.

Examples

```
r <- rotation_test(
  event_times =
    2000 * runif(500),
  exp_period = c(100, 200),
  return_rot_stats = TRUE, ts_fun = mean
)
```

`rotmat2euler` *Decompose a rotation (or direction cosine) matrix*

Description

This function is used to decompose a rotation (or direction cosine) matrix into Euler angles, pitch, roll, and heading.

Usage

```
rotmat2euler(Q)
```

Arguments

`Q`	is a 3x3 rotation matrix.

Value

A 1x3 vector containing: prh=[p,r,h] where p is the pitch angle in radians, r is the roll angle in radians, and h is the heading or yaw angle in radians.

Examples

```
set <- matrix(c(0.6765458, 0.7227523, 0.1411200,
0.3675912, -0.4975063, 0.7857252,
0.6380928, -0.4797047, -0.6022632), nrow = 3, ncol = 3)
rotmat2euler(set)
```

`rough_cal_3d` *Estimate scale factors and offsets*

Description

This function is used to estimate scale factors and offsets for measurements from a triaxial field sensor. This function estimates the scale factor needed to make the magnitude of X close to the expected field strength. It then calls fix_offset_3d to correct any offset errors in X. This function does not try to optimize the results. See spherical_cal for a more powerful data-driven calibration method.

Usage

```
rough_cal_3d(X, fstr)
```

Arguments

`X`	A sensor structure or matrix containing measurements from a triaxial field sensor such as an accelerometer or magnetometer. X can be in any units and frame.
`fstr`	The expected field strength at the measurement location in the same units as X

Value

A list with 2 elements:

- **X:** A sensor structure or matrix containing the adjusted triaxial sensor measurements. It is the same size and has the same sampling rate and units as the input data. If the input is a sensor structure, the output will be also.

- **G:** A list of calibration information containing one field: G$poly, a 3x2 matrix. Rows correspond to X,Y,Z axes. with one column for each of the X, Y, Z axes. The first column of G$poly contains scale factors and second column of G$poly is the offset added to each column of X after scaling.

Note

This function requires a lot of data as it is looking for extreme values in each axis. A minimum data size of 1000 samples should be used. This function is only usable for field sensors. It will not work for gyroscope data.

Examples

```
BW <- beaked_whale
plot(x = c(1:length(BW$M$data)), y = BW$M$data)
rcal <- rough_cal_3d(BW$M$data, fstr = 38.2)
cal <- list(x = c(1:length(rcal$X)), y = rcal$X)
plot(cal)
```

`save_nc` *Save a tag dataset to a netCDF file.*

Description

This function saves a tag dataset to a netCDF file (this is an archival file format supported by the tagtools package and suitable for submission to online data archives).

Usage

```
save_nc(file, X, ...)
```

Arguments

`file`	The name of the data and metadata file to be written. If `file` does not include a .nc suffix, this will be added automatically.
`X`	An `animaltag` object, or a list of tag sensor and/or metadata lists. Alternatively, sensor and metadata lists may be input as multiple separate unnamed inputs. Only these kind of variables can be saved in a NetCDF file because the supporting information in these structures is needed to describe the contents of the file. For non-archive and non-portable storage of variables, consider using `save` or various functions to write data to text files.
`...`	Additional sensor or metadata lists, if user has not bundled them all into a list already but is providing individual structures.

Details

Warning: this will overwrite any previous NetCDF file with the same name. The file is assumed to be in the current working directory unless `file` includes file path information.

Value

no return; saves a dataset to an nc file

Examples

```
BW <- beaked_whale
save_nc("beaked_whale_test", BW)
```

`sens_struct` *Generate a sensor structure from a sensor data vector or matrix.*

Description

Generate a sensor structure from a sensor data vector or matrix.

Usage

```
sens_struct(
  data,
  sampling_rate = NULL,
  times = NULL,
  depid,
  type,
  unit = NULL,
  frame = NULL,
  name = NULL,
  start_offset = 0,
  start_offset_units = "second",
  quiet = FALSE
)
```

Arguments

`data`	sensor data vector or matrix
`sampling_rate`	(optional) sensor data sampling rate in Hz
`times`	(optional) is the time in seconds of each measurement in data for irregularly sampled data. The time reference (i.e., the 0 time) should be with respect to the start time of the deployment.
`depid`	string that provides a unique identifier for this tag deployment
`type`	is a string containing the first few letters of the sensor type, e.g., acc for acceleration. These will be matched to the list of sensor names in the sensor_names.csv file. If more than one sensor matches type, a warning will be given. type can be in upper or lower case.

`unit`	(optional) units in which data are sampled. Default determined by matching `type` with defaults in sensor_names.csv
`frame`	(optional) frame of reference for data axes, for example 'animal' or 'tag'. Default determined by matching `type` with defaults in sensor_names.csv.
`name`	(optional) "full name" to assign to the variable. Default determined by matching `type` to defaults in sensor_names.csv/
`start_offset`	(optional) offset in start time for this sensor relative to start of tag recording. Defaults to 0.
`start_offset_units`	(optional) units of start_offset. default is 'second'.
`quiet`	prints to screen if quiet is set to false

Value

A sensor list with field `data` containing the data and with metadata fields pre-populated from the sensor_names.csv file. Change these manually as needed (or specify the relevant inputs to `sens_struct`) to the correct values.

Examples

```
HB <- harbor_seal
A <- sens_struct(data=HB$A$data,sampling_rate=3,depid='md13_134a', type='acc', quiet=TRUE)
```

`smooth` *Low pass filter a time series*

Description

This function is used to low pass filter (smooth) a regularly-sampled time series.

Usage

```
smooth(x, n)
```

Arguments

`x`	The signal to be filtered. It can be multi-channel with a signal in each column, e.g., an acceleration matrix. The number of samples (i.e., the number of rows in x) must be larger than the filter length, n.
`n`	The smoothing parameter - use a larger number to smooth more. n must be greater than 1. Signal components above 1/n of the Nyquist frequency are filtered out.

Value

The input signal has the first and fifth harmonic. Applying the low-pass filter removes most of the fifth harmonic so the output appears as a sinewave except for the first few samples which are affected by the filter startup transient. Smooth uses fir_nodelay to perform the filtering and so introduces no delay.

Examples

```
y1 <- sin((2 * pi * 0.05) %*% t(c(1:100))) + cos((2 * pi * 0.25) %*% t(c(1:100)))
x1 = c(1:length(y1))
plot(x = x1, y = y1)
y2 <- smooth(x1, n = 4)
x2 = c(1:length(y2))
plot(x = x2, y = y2)
```

`sound_speed` *Sound speed estimation*

Description

This function is used to estimate the sound speed using Coppens equation

Usage

```
sound_speed(temperature, D = NULL, S = NULL)
```

Arguments

`temperature`	The temperature in degrees C
`D`	(optional) The depth in meters (defaults to 1 m)
`S`	The salinity in part-per-thousand (defaults to 35 ppt)

Value

The sound speed in m/s

Note

Range of validity: temperature 0 to 35 °C, salinity 0 to 45 parts per thousand, depth 0 to 4000 m

Source: http://resource.npl.co.uk/acoustics/techguides/soundseawater/content.html#UNESCO

Examples

```
sound_speed(8, 1000, 34)
```

spectrum_level *Compute the spectrum level of a signal x.*

Description

This function is used to compute the spectrum level of a signal x.

Usage

```
spectrum_level(x, nfft, sampling_rate, w, nov)
```

Arguments

`x`	A vector containing the signal to be processed. For signals with multiple channels, each channel should be in a column of x.
`nfft`	The length of the fft to use. Choose a power of two for fastest operation. Default value is 512.
`sampling_rate`	The sampling rate of x in Hz. Default value is 1. sampling_rate is the vector of frequencies at which SL is calculated.
`w`	The window length. The default value is nfft. If w<nfft, each segment of w samples is zero-padded to nfft.
`nov`	The number of samples to overlap each segment. The default value is half of the window length.

Value

A list with 2 elements:

- **SL:** The spectrum level at each frequency in dB RMS re root-Hz. The spectrum is single-sided and extends to sampling_rate/2. The reference level is 1.0 (i.e., white noise with unit variance will have a spectrum level of 3-10*log10(sampling_rate). The 3dB is because both the negative and positive spectra are added together so that the total power in the signal is the same as the total power in the spectrum.
- **freq:** The vector of frequencies at which SL is calculated.

Note

The spectrum is single-sided and extends to sampling_rate/2. The reference level is 1.0 (i.e., white noise with unit variance will have a spectrum level of 3-10*log10(sampling_rate). The 3dB is because both the negative and positive spectra are added together so that the total power in the signal is the same as the total power in the spectrum.

Examples

```
BW <- beaked_whale
list <- spectrum_level(x = BW$P$data, nfft = 4, sampling_rate = BW$P$sampling_rate)
```

speed_from_depth *Estimate the forward speed of a diving animal*

Description

This function is used to estimate the forward speed of a diving animal by first computing the depth-rate (i.e., the first differential of the depth) and then correcting for the pitch angle.

Usage

```
speed_from_depth(
  p,
  A = NULL,
  fs_p = NULL,
  fs_A = NULL,
  fc = 0.2,
  plim = 20/180 * pi
)
```

Arguments

`p`	The depth vector (a regularly sampled time series) in meters. sampled at sampling_rate Hz. This can either be an animaltags sensor list, or a vector.
`A`	(optional) A matrix or animaltags sensor data list containing acceleration data. If A is not provided then only vertical velocity is returned (same output as depth_rate()). Acceleration can be in any consistent unit, e.g., g or m/s^2. Acceleration data must have the same number of rows as p.
`fs_p`	(optional) The sampling rate of p in Hz (samples per second). Required only if p is vector rather than sensor data list.
`fs_A`	(optional) The sampling rate of A in Hz (samples per second). Required only if A is vector rather than sensor data list.
`fc`	(optional) Specifies the cut-off frequency of a low-pass filter to apply to p after computing depth-rate and to A before computing pitch. The filter cut-off frequency is in Hz. The filter length is 4*sampling_rate/fc. Filtering adds no group delay. If fc is empty or not given, the default value of 0.2 Hz (i.e., a 5 second time constant) is used.
`plim`	(optional) Minimum pitch angle, in radians, at which speed can be computed. Default: 0.3490659 radians = 20 degrees. Errors in speed estimation using this method increase strongly at low pitch angles. To avoid estimates with poor accuracy being used in later analyses, speed estimates at low pitch angles are replaced by NaN (not-a-number). The default threshold for this is 20 degrees.

Value

Either forward speed or vertical speed:

- **s:** If both p and A are input, the forward speed estimate in m/s is returned
- **v:** If only p is input, the depth-rate (or vertical velocity) in m/s is returned

Note

Output sampling rate is the same as the input sampling rate. If A and p are input and A has a higher sampling rate, then p and the output are interpolated to match A using `interp2length` .

Frame: This function assumes a [north,east,up] navigation frame and a [forward,right,up] local frame. In these frames, a positive pitch angle is an anti-clockwise rotation around the y-axis. A descending animal will have a negative pitch angle.

Forward velocity for animals could be negative if its vertical velocity is negative and pitch angle is positive, or, its vertical velocity is positive and pitch angle is negative. One could avoid getting negative forward velocity by taking the absolute value of the output.

Examples

```
s <- speed_from_depth(harbor_seal$P, harbor_seal$A)
```

`spherical_cal` *Deduce the calibration constants*

Description

This function is used to deduce the calibration constants for a triaxial field sensor, such as an accelerometer or magnetometer, based on movement data. This can be used to do a 'bench' calibration of a sensor.

Usage

```
spherical_cal(X, n = NULL, method = NULL)
```

Arguments

X	The segment of triaxial sensor data to calibrate. It must be a 3-column matrix. X can come from any triaxial field sensor and can be in any unit and any frame.
n	The target field magnitude e.g., 9.81 for accelerometer data using m/s^2 as the unit.
method	An optional string selecting the type of calibration. The default is to calibrate for offset and scaling only. Other options are: 'gain' adjust gain of axes 2 and 3 relative to 1, or 'cross' adjust gain and remove cross-axis correlations

Details

The function reports the residual and the axial balance of the data. A low residual e.g., <5% indicates that the data can be calibrated well and there is not much noise. The axial balance indicates whether the movement in X is suitable for data-driven calibration. If the movement covers all directions fairly equally, the axial balance will be high. A balance <20 % may lead to unreliable calibration. For bench calibrations, a high axial balance is achieved by rotating the sensor through the full 3-dimensions. Sampling rate and frame of Y are the same as the input data so Y has the same size as X. The units of Y are the same as the units used for n. If n is not specified, the units of Y are the same as for the input data. It is a good idea to low-pass filter and/or remove outliers from the sensor data before using this function to reduce errors from specific acceleration and sensor noise.

Value

A list with 2 elements:

- **Y:** The matrix of converted sensor values. These will have the same units as for input argument n. The size of Y is the same as the size of X and it has the same frame and sampling rate.
- **G:** The calibration structure containing fields: G.poly is a matrix of polynomials. The first column of G.poly is the three scale factors applied to the columns of X. The second column is the offset added to each column of X after scaling. G.cross is a 3x3 matrix of cross-factors. If there are no cross-terms, this is the identity matrix. Off-axis terms correct for cross-axis sensitivity.

A message will also be printed to the screen presenting

Note

This function uses a Simplex search for optimal calibration parameters and so can be slow if the data size is large. For this reason it is most suitable for bench calibrations rather than field data. This function is only usable for field sensors. It will not work for gyroscope data.

Examples

```
p <- spherical_cal(harbor_seal$A$data)
```

`tag2animal` *Tag-frame to animal-frame conversion*

Description

Convert tag frame measurements to animal frame using pre-determined tag orientation(s) on the animal.

Usage

```
tag2animal(X, sampling_rate, OTAB, Ya = NULL)
```

Arguments

`X`	Data from a triaxial sensor such as an accelerometer, magnetometer or a gyroscope. X can be a three column matrix or a sensor structure (**not** a data frame or tbl). In either case, X is in the tag frame, i.e., expressed in the canonical axes of the tag, not the animal. X can have any unit and any regular sampling rate (i.e., measurements are regularly sampled; equally spaced in time).
`sampling_rate`	(optional) The sampling rate of the sensor data in Hz (samples per second). This is only needed if X is not a sensor structure. If X is a sensor data list, sampling_rate is obtained from its metadata (X$sampling_rate).
`OTAB`	is a matrix defining the orientation of the tag on the animal as a function of time. Each row of OTAB is: `cue1, cue2, pitch, roll, heading`. (See **Details**.)
`Ya`	is an optional sensor structure in which the sensor data has already been converted to the animal frame. The OTAB is extracted from this structure. This is useful, for example, to replicate tag-to-animal conversions at different sampling rates.

Details

This function uses the OTAB matrix to convert sensor data `X` from tag frame of reference to whale frame of reference. Each row of OTAB is: `cue1, cue2, pitch, roll, heading` where cue1 is the start time of a move in seconds with respect to the start of X. cue2 is the end time of the move. If cue1 and cue2 are the same, the move is instantaneous, otherwise a gradual move will be implemented in which the orientation of the tag is linearly interpolated between the previous and the new orientation. The pitch, roll and heading angles describe the tag orientation on the animal at the end of the move (angles are in radians). The first row of OTAB must have cue1 and cue2 equal to 0 as this is the initial orientation of the tag on the animal. Subsequent rows (if any) of OTAB describe

Value

Xa,the sensor data in the animal frame, i.e., rotated to correct for the tag orientation on the animal. If X is a sensor structure, Xa will also be one. In this case the structure elements 'frame' and 'name' will be changed. The OTAB will also be added to the structure.

See Also

[prh_predictor1], [prh_predictor2]

Examples

```
Aw <- tag2animal(beaked_whale$A, OTAB = matrix(c(0,0,0.1, 0.04, -0.2), nrow = 1))
```

tortuosity *Measure tortuosity index*

Description

This function is used to measure the tortuosity of a regularly sampled horizontal track. Tortuosity can be measured in a number of ways. This function compares the stretched-out track length (STL) over an interval of time with the distance made good (DMG, i.e., the distance actually covered in the interval). The index returned is (STL-DMG)/STL which is 0 for straightline movement and 1 for extreme circular movement.

Usage

```
tortuosity(track, sampling_rate, intvl)
```

Arguments

track	Contains the animal positions in a local horizontal plane. The track parameter has a row for each position and two columns: northing and easting. The positions can be in any consistent spatial unit, e.g., metres, km, nautical miles, and are referenced to an arbitrary 0,0 location. track object cannot be in degrees as the distance equivalent to a degree latitude is not the same as for a degree longitude.
sampling_rate	The sampling rate of the positions in Hertz (samples per second).
intvl	The time interval in seconds over which tortuosity is calculated. This should be chosen according to the scale of interest, e.g., the typical length of a foraging bout.

Value

The tortuosity index which is between 0 and 1 as described above. t contains a value for each period of intvl seconds.

Note

This tortuosity index is fairly insensitive to speed so if track is produced by dead-reckoning (e.g., using ptrack or htrack), the speed estimate is not important. Also the frame of track is not important as long as the two axes (nominally called northing and easting) used to describe the positions are perpendicular.

Examples

```
BW <- beaked_whale
track <- ptrack(
  A = BW$A$data, M = BW$M$data, s = 3,
  sampling_rate = BW$A$sampling_rate,
  fc = NULL, return_pe = TRUE
)$track
tortuosity <- tortuosity(track, sampling_rate = BW$A$sampling_rate, intvl = 25)
```

`track3D` *Reconstruct a track from pitch, heading and depth data, given a starting position*

Description

The track3D function will use data from a tag to reconstruct a track by fitting a state space model using a Kalman filter. If no x,y observations are provided then this corresponds to a pseudo-track obtained via dead reckoning and extreme care is required in interpreting the results.

Usage

```
track3D(
  z,
  phi,
  psi,
  sf,
  r = 0.001,
  q1p = 0.02,
  q2p = 0.08,
  q3p = 1.6e-05,
  tagonx,
  tagony,
  enforce = TRUE,
  x,
  y
)
```

Arguments

`z`	A vector with depth over time (in meters, an observation)
`phi`	A vector with pitch over time (in Radians, assumed as a known covariate)
`psi`	A vector with heading over time (in Radians, assumed as a known covariate)
`sf`	A scalar defining the sampling rate (in Hz)
`r`	Observation error
`q1p`	speed state error
`q2p`	depth state error
`q3p`	x and y state error
`tagonx`	Easting of starting position (in meters, so requires projected data)
`tagony`	Northing of starting position (in meters, so requires projected data)
`enforce`	If TRUE (the default), then speed and depth are kept strictly positive
`x`	Direct observations of Easting (in meters, so requires projected data)
`y`	Direct observations of Northing (in meters, so requires projected data)

Value

A list with 10 elements:

- **p:** the smoothed speeds
- **fit.ks:** the fitted speeds
- **fit.kd:** the fitted depths
- **fit.xs:** the fitted xs
- **fit.ys:** the fitted ys
- **fit.rd:** the smoothed depths
- **fit.rx:** the smoothed xs
- **fit.ry:** the smoothed ys
- **fit.kp:** the kalman a posteriori state covariance
- **fit.ksmo:** the kalman smoother variance

Note

Output sampling rate is the same as the input sampling rate.

Frame: This function assumes a [north,east,up] navigation frame and a [forward,right,up] local frame. In these frames, a positive pitch angle is an anti-clockwise rotation around the y-axis. A positive roll angle is a clockwise rotation around the x-axis. A descending animal will have a negative pitch angle while an animal rolled with its right side up will have a positive roll angle.

This function output can be quite sensitive to the inputs used, namely those that define the relative weight given to the existing data, in particular regarding (x,y)=(lat,long); increasing q3p, the (x,y) state variance, will increase the weight given to independent observations of (x,y), say from GPS readings

See Also

`m2h`,`a2pr`

Examples

```
p <- a2pr(A = beaked_whale$A$data)
h <- m2h(M = beaked_whale$M$data, A = beaked_whale$A$data)
track <- track3D(z = beaked_whale$P$data, phi = p$p,
psi = h$h, sf = beaked_whale$A$sampling_rate,
r = 0.001, q1p = 0.02, q2p = 0.08, q3p = 1.6e-05,
tagonx = 1000, tagony = 1000, enforce = TRUE, x = NA, y = NA)
oldpar <- graphics::par(no.readonly = TRUE)
graphics::par(mfrow = c(2, 1), mar = c(4, 4, 0.5, 0.5))
plot(-beaked_whale$P$data, pch = ".", ylab = "Depth (m)",
xlab = "Time")
plot(track$fit.rx, track$fit.ry, xlab = "X",
ylab = "Y", pch = ".")
points(track$fit.rx[c(1, length(track$fit.rx))],
track$fit.ry[c(1, length(track$fit.rx))], pch = 21, bg = 5:6)
legend("bottomright", cex = 0.7, legend = c("Start", "End"),
col = c(5, 6), pt.bg = c(5, 6), pch = c(21, 21))
graphics::par(oldpar)
```

undo_cal *Undo calibrations steps*

Description

This function is used to undo any calibration steps that have been applied to sensor data. This will reverse any re-mapping, scaling and offset adjustments that have been applied to the data, reverting the sensor data to the state it was when read in from the source (excluding any filtering or decimation steps).

Usage

```
undo_cal(X, temperature)
```

Arguments

X	A sensor list or set of sensor lists in the tag frame, i.e., with calibrations applied.
temperature	A vector of temperature measurements with the same number of samples and sampling rate as the data in the input sensor data structure X. The temperature parameter indicates the temperature experienced by the sensor during data collection (not necessarily the ambient temperature experienced by the animal), and may affect calibration because many sensors' output values change depending on the temperature.

Value

A sensor list or set of sensor lists reverted to the sensor frame, i.e., without calibrations.

Examples

```
BW <- beaked_whale
no_cal <- undo_cal(BW)
```

`zero_crossings` *Find zero-crossings in a vector*

Description

This function is used to find the zero-crossings in a vector using a hysteretic detector. This is useful, e.g., to locate cyclic postural changes due to propulsion.

Usage

```
zero_crossings(x, TH, Tmax = NULL)
```

Arguments

`x`	A vector of data. This can be from any sensor and with any sampling rate.
`TH`	The magnitude threshold for detecting a zero-crossing. A zero-crossing is only detected when values in x pass from -TH to +TH or vice versa.
`Tmax`	(optional) The maximum duration in samples between threshold crossings. To be accepted as a zero-crossing, the signal must pass from below -TH to above TH, or vice versa, in no more than Tmax samples. This is useful to eliminate slow transitions. If Tmax is not given, there is no limit on the number of samples between threshold crossings.

Value

A list with elements

- **K:** A vector of cues (in samples) to zero-crossings in x.
- **s:** A vector containing the sign of each zero-crossing (1 = positive-going, -1 = negative-going). s is the same size as K. If no zero-crossings are found, K and s will be empty
- **KK:** The zero crossings of the vertical velocity vector

Note

Frame: This function assumes a [north,east,up] navigation frame and a [forward,right,up] local frame. Both A and M must be rotated if needed to match the animal's cardinal axes otherwise the track will not be meaningful.

CAUTION: dead-reckoned tracks are usually very inaccurate. They are useful to get an idea of HOW animals move rather than WHERE they go. Few animals probably travel in exactly the direction of their longitudinal axis and anyway measuring the precise orientation of the longitudinal axis of a non-rigid animal is fraught with error. Moreover, if there is net flow in the medium, the animal will be advected by the flow in addition to its autonomous movement. For swimming animals this can lead to substantial errors. The forward speed is assumed to be with respect to the medium so the track derived here is NOT the 'track-made-good', i.e., the geographic movement of the animal. It estimates the movement of the animal with respect to the medium. There are numerous other sources of error so use at your own risk!

Examples

```
R <- zero_crossings(sin(2 * pi * 0.033 * c(1:100)), 0.3)
s <- c(-1, 1, -1, 1, -1, 1)
```

Index