Package 'RchivalTag'

October 1, 2025

Type Package

Title Analyzing and Interactive Visualization of Archival Tagging Data

Version 0.2.2

Date 2025-09-19

Maintainer Robert K. Bauer <marine.biologging@gmail.com>

Description A set of functions to generate, access and analyze standard data products from archival tagging data.

Depends R (>= 3.5.0), maps, mapdata, leaflet

Imports plyr, oceanmap, cleangeo, suntools, graphics, stats, raster, readr, ncdf4, pracma, dygraphs, xts, lubridate, shiny, htmlwidgets, stringr, grDevices, sp, methods, ggplot2, plotly, leaflet.extras2, sf

License GPL (>= 3)

LazyLoad yes

Repository CRAN

Date/Publication 2025-10-01 10:40:02 UTC

NeedsCompilation no

Author Robert K. Bauer [aut, cre] (ORCID: 0000-0003-4224-8023)

Contents

bin_TempTS 2
classify_DayTime 4
combine_histos 5
dy_DepthTS 6
get_DayTimeLimits 8
get_thermalstrat 9
ggboxplot_DepthTS_by_hour 12
ggplotly_geopos 14
ggplot_geopos 16
hist_tad 18

hist_tat 22
image_TempDepthProfiles 26
interpolate_TempDepthProfiles 28
leaflet_geopos 30
merge_histos 32
plot_data_coverage 34
plot_DepthTempTS 37
plot_TS 39
RchivalTag 42
read_histos 44
read_PDT 47
read_TS 49
resample_PDT 51
resample_TS 53
simulate_DepthTS 54
ts2histos 55
unmerge_histos 58
update_leaflet_elementId 59

Index **61**

`bin_TempTS` *bin depth-temperature time series data*

Description

bins depth-temperature time series data to a user-defined resolution, returning the minimum, maxmium and average temperature recorded at each depth interval (bin) per sampling day. The output is comparable to that of read_PDT.

Why binning temperature data?
In case of archival tagging data, depth-temperature time series data at a given day may consist of multiple tempertaure profiles of different signatures, depending on the animal's behaviour. Slight differences in temperature profiles could impede further analyses (e.g. the estimation of the thermocline depth), if just the average profile is applied. To avoid such problmes, it is useful to calculate the average temperature at given depth intervals (bins) and thus smooth temperature profiles of a given period.

In addition, temperature at depth profiles can be interpolated and then visualized using functions interpolate_TempDepthProfiles and image_TempDepthProfiles, respectively. This faciliates the analysis of temporal changes of temperature profiles, for instance, in relation to animal behaviour (e.g. diving behaviour).

Usage

```
bin_TempTS(ts, res=8, verbose=FALSE)
```

Arguments

ts	a data.frame with columns date, Depth and Temperature
res	the depth interval at which temperatures should be binned.
verbose	whether the sampling dates should be indicated during the binning process (by default FALSE).

Value

A data.frame with the columns date, MeanTemp, MinTemp, MaxTemp, bin and MeanPDT (the latter being the average of the min and maximum water temperatures). Additional columns, used to distinguish tags, may include Serial, DeployID and Ptt, depending on their availability in the original ts-data.fralme.

Author(s)

Robert K. Bauer

References

Bauer, R., F. Forget and JM. Fromentin (2015) Optimizing PAT data transmission: assessing the accuracy of temperature summary data to estimate environmental conditions. Fisheries Oceanography, 24(6): 533-539, doi:10.1111/fog.12127

See Also

read_PDT, interpolate_TempDepthProfiles, image_TempDepthProfiles

Examples

```
# #### example 1) run on time series data:
## step I) read sample time series data file:
DepthTempTS <- read.table(system.file("example_files/104659-Series.csv",
                                      package="RchivalTag"),header = TRUE,sep=',')
DepthTempTS$date <- as.Date(DepthTempTS$Day,"%d-%b-%Y")
head(DepthTempTS)
#
#
# ## step Ib) bin temperature data on 10m depth bins
# ##          to increase later estimate accuracy (see Bauer et al. 2015):
# # DepthTempTS_binned <- bin_TempTS(DepthTempTS,res=10)
#
# ## step II) interpolate average temperature fields (MeanTemp) from binned data:
# m <- interpolate_TempDepthProfiles(DepthTempTS)
# # m <- interpolate_PDTs(DepthTempTS_binned)
# str(m)
# m$sm
#
# ## step III) calculate thermal stratifcation indicators per day (and tag):
# get_thermalstrat(m, all_info = TRUE)
# get_thermalstrat(m, all_info = FALSE)
```

```
#
# ## step IV) plot interpolated profiles:
# image_TempDepthProfiles(m$station.1)
```

`classify_DayTime` *Classifying the time period of the day*

Description

Classifying the time period of the day based on the timing of sunrise, sunset (and twilight events) or alternatively, geolocation estimates, as specified in get_DayTimeLimits, that allow their internal estimation during the function call.

Usage

```
classify_DayTime(pos, twilight.set="ast")
```

Arguments

`pos`	A data.frame `pos` with the columns `sunrise`, `sunset`, `dawn.ast`/`dawn.naut` and `dusk.ast`/`dusk.naut` in `POSIXct`-format. Note that the expected twilight vector (suffix `"ast"` for astronomical dawn and dusks; vs suffix `"naut"` for nautical twilight events) is defined by the function's second argument `twilight.set` (see description below).
`twilight.set`	Character string, indicating the type of twilight used for the long daytime classification: `"ast"` (default) for astronomical and `"naut"` for nautical twilight events with sun angles of 18 vs 12 below the horizon, respectively. Corresponding (expected) dawn and dusk vector names are `dawn.ast` & `dusk.ast` vs `dawn.naut` & `dusk.naut`.

Value

The input data.frame `pos` extended by the time vectors `daytime` and `daytime.long`. In the former case, "Day" and "Night" periods are distinguished. In the latter case, "Day", "Night", "Dawn" and "Dusk".

Author(s)

Robert K. Bauer

References

Meeus, J. (1991) Astronomical Algorithms. Willmann-Bell, Inc.

See Also

sunriset, crepuscule, get_DayTimeLimits

Examples

```
#### example 1) estimate current times of dawn, sunrise, dusk and sunset in Mainz, Germany:
pos <- data.frame(Lat=8.2667, Lon=50)
pos$datetime <- strptime(Sys.Date(), "%Y-%m-%d")
get_DayTimeLimits(pos)

#### example 1b) classify current time of the day in Mainz, Germany:
classify_DayTime(get_DayTimeLimits(pos))

## convert 1c) back-to-back histogram showing day vs night TAD frequencies:
### load sample depth and temperature time series data from miniPAT:
ts_file <- system.file('example_files/104659-Series.csv',package='RchivalTag')
ts_df <- read.table(ts_file, header = TRUE, sep = ",")
tad_breaks <- c(0, 2, 5, 10, 20, 50, 100, 200, 300, 400, 600, 2000)

ts_df$Lat <- 4; ts_df$Lon <- 42.5
ts_df$datetime <- strptime(paste(ts_df$Day, ts_df$Time), "%d-%B-%Y %H:%M:%S")
ts_df2 <- classify_DayTime(get_DayTimeLimits(ts_df))
head(ts_df2)

ts2histos(ts_df2, tad_breaks = tad_breaks, split_by = "daytime")
hist_tad(ts_df2, bin_breaks = tad_breaks, split_by = "daytime", do_mid.ticks = FALSE)
```

---

combine_histos *combine lists of TAD/TAT frequency data*

Description

This function allows to combine seperate lists of TAD/TAT frequency data from archival tags (i.e. by Wildlife Computers). The function requires ungrouped/unmerged TAD/TAT lists to avoid merging duplicate records (e.g. multiple TAD/TAT lists from the same individual). However, grouped/merged lists with TAD/TAT from multiple individuals and even duplicate records can be provided, as the function includes an internal call of unmerge_histos to meet this requirement.

Usage

```
combine_histos(hist_list1, hist_list2)
```

Arguments

`hist_list1`, `hist_list2`
: Two list-of-lists to be combined, each containing TAD and TAT frequency data and the corresponding `bin_breaks` from one or several archival tags.

Value

A list-of-lists of ungrouped/unmerged TAD and TAT frequency data.

$ TAD:List
..$ ID1 : List of 2

.. ..$ bin_breaks: num
.. ..$ df : data.frame
$ TAT:List
..$ ID1 : List of 2
.. ..$ bin_breaks: num
.. ..$ df : data.frame
..$ ID2 : List of 2
...

Author(s)

Robert K. Bauer

See Also

unmerge_histos, merge_histos, hist_tad, hist_tat

Examples

```
## example 1) read, merge and plot TAD frequency data from several files:
## part I - read histogram data from two files:
hist_dat_1 <- read_histos(system.file("example_files/104659-Histos.csv",package="RchivalTag"))
hist_dat_2 <- read_histos(system.file("example_files/104659b-Histos.csv",package="RchivalTag"))
## note the second list is based on the same data (tag), but on different bin_breaks

## part II - combine TAD/TAT frecuency data from seperate files in one list:
hist_dat_combined <- combine_histos(hist_dat_1, hist_dat_2)
par(mfrow=c(2,1))
hist_tad(hist_dat_combined)
hist_tat(hist_dat_combined)

## part III - force merge TAD/TAT frecuency data from seperate files
# in one list, by applying common bin_breaks:
hist_dat_merged <- merge_histos(hist_dat_combined,force_merge = TRUE)
hist_tad(hist_dat_merged)
hist_tat(hist_dat_merged)

## part IV - plot merged data:
hist_tad(hist_dat_merged) # of all tags
unique(hist_dat_merged$TAD$merged$df$DeployID) ## list unique tags in merged list
hist_tad(hist_dat_merged, select_id = "15P1019b", select_from = 'DeployID') # of one tag

## part V - unmerge data:
unmerge_histos(hist_dat_merged)
```

---

dy_DepthTS *Plot time series data via the dygraphs interactive interface*

Description

Plot time series data (e.g. depth or temperature time series from archival tags) via the dygraphs interactive interface.

Usage

```
dy_TS(ts_df, y="Depth", xlim, ylim,
      ylab=y, xlab, main,
      ID, ID_label="Serial",
      plot_DayTimePeriods=TRUE, twilight.set="ast",
      color="darkblue",
      doRangeSelector=TRUE, drawPoints=FALSE, pointSize=2, tz="UTC", ...)

dy_DepthTS(ts_df, y="Depth", xlim, ylim,
           ylab=y, xlab, main,
           ID, ID_label="Serial",
           plot_DayTimePeriods=TRUE, twilight.set="ast",
           color="darkblue",
           doRangeSelector=TRUE, drawPoints=FALSE, pointSize=2, tz="UTC", ...)
```

Arguments

`ts_df`	data.frame holding the time series data to be plotted, including the x-vector 'datetime' (`POSIXct`, UTC) and the numeric y-vector whose label is defined by `y`.
`y`	Character label of the time series vector to be plotted (default "Depth").
`xlim`	X-axis limits (x1, x2) of the plot (default range(ts_df$datetime), but can be specified in `empty.plot_TS`).
`ylim`	Y-axis limits (default range(ts_df[[y]]), but can be specified in `empty.plot_TS`).
`ylab, xlab`	Axis labels.
`main`	Main title (default "Tag ID").
`ID, ID_label`	Tag ID and column label (default "Serial").
`plot_DayTimePeriods, twilight.set`	Whether day-time periods ("Night", "Dawn", "Day", "Dusk") should be shaded. If `plot_DayTimePeriods=TRUE`, twilight columns are required (or will be estimated via get_DayTimeLimits). `twilight.set` selects "ast" (astronomical) vs "naut" (nautical) twilight.
`color`	Line color (default "darkblue").
`doRangeSelector`	Whether to add dygraph interactive range selector (default TRUE).
`drawPoints, pointSize`	Whether to draw points at sampling times and their size.
`tz`	Time zone for the plot (default "UTC").
`...`	Additional arguments passed to dygraph. Further arguments can be passed via dyOptions.

Value

An interactive dygraph plot object that can be altered further using dyOptions.

Author(s)

Robert K. Bauer

See Also

plot_TS, plot_DepthTempTS, dygraph, dyOptions

Examples

```
### load sample depth and temperature time series data from miniPAT
# ts_file <- system.file("example_files/104659-Series.csv", package="RchivalTag")
# ts_df <- read_TS(ts_file)
# ts_df$Serial <- ts_df$DeployID
# head(ts_df)

## plot depth-time series data
# dy_DepthTS(ts_df)

## add missing Lon, Lat info for day/night shading
# ts_df$Lon <- 5; ts_df$Lat <- 43
# dy_DepthTS(ts_df)

## same figure with plot_DepthTS
# plot_DepthTS(ts_df, plot_DayTimePeriods=TRUE)

## further arguments
# dy_DepthTS(ts_df, xlim=unique(ts_df$date)[2:3], plot_DayTimePeriods=FALSE)

## add further options via dyOptions
# dg <- dy_DepthTS(ts_df, xlim=unique(ts_df$date)[2:3], plot_DayTimePeriods=FALSE, drawPoints=TRUE)
# dg <- dyOptions(dg, drawGrid=FALSE)
# dg
```

---

get_DayTimeLimits *Estimating the timing of sunrise, sunset, astronomical and nautical twilight events*

Description

Estimating the timing of sunrise, sunset, astronomical and nautical twilight events in POSIXct-format based on geolocations and a similar time vector. The function is a simplified call of the sunriset and crepuscule functions of the suntools package, which use algorithms provided by NOAA.

Usage

```
get_DayTimeLimits(pos)
```

Arguments

pos	A data.frame with columns datetime (POSIXct), Lon, and Lat.

Value

The input data.frame pos extended by the time vectors sunrise, sunset, dawn.naut, dawn.ast, dusk.naut, and dusk.ast.

Author(s)

Robert K. Bauer

References

Meeus, J. (1991) Astronomical Algorithms. Willmann-Bell, Inc.

See Also

sunriset, crepuscule, classify_DayTime

Examples

```
#### Example 1: estimate current times of dawn, sunrise, dusk and sunset in Mainz, Germany
pos <- data.frame(Lat = 50, Lon = 8.2667)
pos$datetime <- strptime(Sys.Date(),"%Y-%m-%d")
get_DayTimeLimits(pos)

#### Example 1b: classify current time of the day in Mainz, Germany
classify_DayTime(get_DayTimeLimits(pos))
```

---

get_thermalstrat *estimate thermal stratification indices*

Description

estimates thermal stratification indices, including thermocline depth, gradient, mixed later depth and stratifcation index from daily temperature at depth profiles, as illustrated by Bauer et al. (2015) for archival tagging data.

Usage

```
get_thermalstrat(x, dz=20, strat_lim=100, na.rm=FALSE,
                 verbose=TRUE, Depth_res, all_info=FALSE)
```

Arguments

x	A list generated by interpolate_TempDepthProfiles, containing interpolated temperature at depth profiles and their corresponding date and depth vectors: $ Data_Source.ID_key:List of 3 ..$ Temperature_matrix: num ..$ Depth : num ..$ Date :Date ..$ sm :data.frame .. ..$ Date :chr .. ..$ nrecs :int .. ..$ Depths :chr
dz	size of the moving window in meters between which temperature values should be compared for the estimation of the thermocline gradient and depth (by default 20).
na.rm	whether interpolated temperature at depth profiles with missing values should be treated (default is FALSE).
strat_lim	up to which depth (in meters) temperature values should be considered for the estimation of the stratication index (by default 100).
Depth_res	numeric value, defining the depth resolution at which the temperature data should be interpolated.
verbose	whether the process of the function run should be indicated (by default TRUE).
all_info	whether the summary information of the input file should be generated in the output (by default FALSE).

Value

a data.frame composed of

Date a date vector (see input argument x)

maxDepth_interp the maxmium depth (in meters) of a daily temperature at depth profile to which its interpolation is limitied)

tgrad the maximum temperature gradient of all possible moving windows of size dz)

tcline the thermocline depth, defined as the average depth (of the depth range) of the moving window(s) with the maximum temperature gradient tgrad)

dz size of the moving window in meters between which temperature values were compared

mld mixed layer depth, defined as the average depth of the first moving window that meets maximum temperature gradient criterium)

mld_0.5 mixed layer depth, defined as as the depth at which T = SST-0.5 degrees, the temperature criterion of Monterey and Levitus (1997).

mld_0.8 mixed layer depth, defined as the depth at which T = SST-0.8 degrees, the temperature criterion of Kara et al. (2000, 2003).

strat_index stratification index, defined as the standard deviation of all interpolated temperature values up to the depth defined by the argument strat_lim

... optional columns to be taken from the sm data.frame of the input list (in case that all_info=TRUE)

nrecs number of records of the non-interpolated daily temperature at depth profiles

Depths unique depth records of the non-interpolated daily temperature at depth profiles, seperated by '; '

Author(s)

Robert K. Bauer

References

Bauer, R., F. Forget and JM. Fromentin (2015) Optimizing PAT data transmission: assessing the accuracy of temperature summary data to estimate environmental conditions. Fisheries Oceanography, 24(6): 533-539, doi:10.1111/fog.12127

Kara, A. B., P. A. Rochford, and H. E. Hurlburt (2000). An optimal definition for ocean mixed layer depth. Journal of Geophysical Research, 105:16803-16821, doi:10.1029/2000JC900072

Kara, A. B., P. A. Rochford, and H. E. Hurlburt (2003) Mixed layer depth variability over the global ocean. Journal of Geophysical Research, 108:3079, doi:10.1029/2000JC000736

Monterey, G., and S. Levitus (1997) Seasonal variability of mixed layer depth for the world ocean. NOAA Atlas NESDIS 14, U. S. Govt. Printing Office.

See Also

interpolate_TempDepthProfiles

Examples

```
#### example 1) run on PDT file:
## step I) read sample PDT data file:
path <- system.file("example_files",package="RchivalTag")
PDT <- read_PDT("104659-PDTs.csv",folder=path)
head(PDT)
#
# ## step II) interpolate average temperature fields (MeanPDT) from PDT file:
# m <- interpolate_PDTs(PDT)
# str(m)
# m$sm
#
# ## step III) calculate thermal stratifcation indicators per day (and tag):
# get_thermalstrat(m, all_info = TRUE)
# get_thermalstrat(m, all_info = FALSE)
#
#
# #### example 2) run on time series data:
# ## step I) read sample time series data file:
# DepthTempTS <- read.table(system.file("example_files/104659-Series.csv",
#                                       package="RchivalTag"),header = TRUE,sep=',')
# DepthTempTS$date <- as.Date(DepthTempTS$Day,"%d-%b-%Y")
# head(DepthTempTS)
#
```

```
#
# ## step Ib) bin temperature data on 10m depth bins
# ##          to increase later estimate accuracy (see Bauer et al. 2015):
# # DepthTempTS_binned <- bin_TempTS(DepthTempTS,res=10)
#
# ## step II) interpolate average temperature fields (MeanTemp) from binned data:
# m <- interpolate_TempDepthProfiles(DepthTempTS)
# # m <- interpolate_PDTs(DepthTempTS_binned)
# str(m)
# m$sm
#
# ## step III) calculate thermal stratifcation indicators per day (and tag):
# get_thermalstrat(m, all_info = TRUE)
# get_thermalstrat(m, all_info = FALSE)
```

---

ggboxplot_DepthTS_by_hour

Boxplot on hourly aggregated DepthTS records

Description

This function creates a boxplot via the ggplot library from hourly aggregated DepthTS records

Usage

```
ggboxplot_DepthTS_by_hour(ts_df, ylim, min_perc=75,
                          main="", submain, add_ids_to_submain=FALSE, xlab,
                          ID, ID_label="Serial",
                          plot_DayTimePeriods=TRUE, twilight.set="ast",
                          box=TRUE, jitter=FALSE,
                          color_by=ID_label, cb.title=color_by,
                          pal, opacity=0.1,tz="UTC")
```

Arguments

ts_df	data.frame with DepthTS data,
ylim	limits of the y-axis.
min_perc	the minimum data coverage (in percent) of daily DepthTS recoords (by default 75%).
main, submain, add_ids_to_submain, xlab	The title and subtitle of the figure as well as the label of the x-axis.
ID, ID_label	Tag ID and its label (column name; by default "Serial") to be selected (e.g. if input data frame holds tagging data from several tags).

plot_DayTimePeriods, twilight.set
: whether day-time periods ('Night', 'Dawn', 'Day', 'Dusk') should be plotted as shaded areas. In case that plot_DayTimePeriods is set TRUE), the limits of each time period are required (columns sunrise, sunset, dawn.ast,/dawn.naut and dawn.ast/dawn.naut in POSIXct-format. In case of the twilight events, the additional argument twilight.set defines the suffix of the twlight-set to be selected ("ast" for astronomical dawn and dusks vs "naut" for nautical twilight events).

box
: whether to draw a box around the figure (by default TRUE)

jitter, color_by, cb.title, pal, opacity
: whether to draw all Depth records on top of the boxplot (by default FALSE). If selected, records will be colored via the different groups in the column specified in color_by with colors specified by the color palette pal and the opacity value between 0 (transparent) and 1 (solid).

tz
: The time zone in which the data should be illustrated (By default "UTC"). ATTENTION: The required date format of the input data is "UTC" (across all RchivalTag-functions). Run OlsonNames(tzdir = NULL) for valid time zone definitions. Be aware that changes of the time zone will alter the internal date definition and thus may result in different datasets to be displayed (depending on the min_perc requirements). This concerns mainly partially transmitted data sets.

Author(s)

Robert K. Bauer

See Also

hist_tad, plot_DepthTS, dy_DepthTS

Examples

```
# ts_file <- system.file("example_files/104659-Series.csv",package="RchivalTag")
# ts_df <- read_TS(ts_file)
# ggboxplot_DepthTS_by_hour(ts_df)
#
# ## Let's add position data to obtain twilight and nighttime shadings:
#
# ts_df$Lon <- 5; ts_df$Lat <- 43
# ts_df2 <- get_DayTimeLimits(ts_df)
# ggboxplot_DepthTS_by_hour(ts_df2,ylim=c(0,100))
#
# ### Let's add the actual depth records on top of the boxplot
# ###(only meaningful in case of few amounts of data):
#
# ggboxplot_DepthTS_by_hour(ts_df2,jitter = T,opacity = 0.1)
```

ggplotly_geopos *Converts a ggplot2 object from ggplot_geopos() to plotly*

Description

This function converts a ggplot2 object created by RchivalTag::ggplot_geopos() to a plotly object.

Usage

```
ggplotly_geopos(ggobj, fixedrange=F, grid=F,expand=10)
```

Arguments

ggobj	Character string identifying regions predefined by the region_definitions-dataset, Raster* or Extent object (corresponds to v_area of the v-function). If missing, region is derived from geographical coordinates, denoted by lat and lon. See add.region to define new region definitions and delete.region to delete unproper region definitions.
fixedrange	Vector returning longitude coordinates of the area to be plotted.
grid	whether a grid should be plotted (default is TRUE)
expand	By default, the underlying ggplotly-function does not stick to the plotting region of the ggobj, but extends it. This can result in missing countries or islands. The expand-argument extends the plotly-plotting window in each direction in order to cover the corresponding landmasks.)

Details

ggoplotmaply uses the ggplotly functions to convert the ggplot object into the plotly format.

Author(s)

Robert K. Bauer

See Also

leaflet_geopos, ggplot_geopos, ggplotmap, ggplotly

Examples

```
# ## example 1a) line plot from several csv-files:
# library(oceanmap)
# csv_file <- system.file("example_files/15P1019-104659-1-GPE3.csv",package="RchivalTag")
# pos <- get_geopos(csv_file)  ## show tracks as line plot
# ggobj <- ggplot_geopos(pos)
# ggobj
# ggplotly_geopos(ggobj)
```

```
#
# ## load second file and add to plot:
# csv_file2 <- system.file("example_files/14P0911-46177-1-GPE3.csv",package="RchivalTag")
# pos2 <- get_geopos(csv_file2)  ## show tracks as line plot
# ggobj2 <- ggplot_geopos(pos2)
# ggplotly_geopos(ggobj2)
#
# pos3 <- rbind(pos,pos2)
# ggobj3 <- ggplot_geopos(pos3,type = "l")
# # ggobj3 <- ggplot_geopos(pos3,type = "b")
# # ggobj3 <- ggplot_geopos(pos3,type = "p")
# ggplotly_geopos(ggobj3)
#
#
# ## example 1b) scatter plot from csv-file on existing landmask:
# ggobj <- oceanmap::ggplotmap('lion',grid.res = 5) # use keyword to derive area limits
# ggobj4 <- ggplot_geopos(csv_file,ggobj)
# ggplotly_geopos(ggobj4)
#
# ## alternatives:
# pos <- get_geopos(csv_file)
# r <- oceanmap::regions("lion")
# ggobj5 <- ggplot_geopos(pos, xlim = r$xlim, ylim = r$ylim)
# ggplotly_geopos(ggobj5)
#
#
# ## example 2) probability surfaces of horizontal tracks from nc-file:
# ## this can take some time as it inlcudes time consuming data processing
# nc_file <- system.file("example_files/15P1019-104659-1-GPE3.nc",package="RchivalTag")
# ggobj6 <- ggplot_geopos(nc_file)
# ggobj6
# ggplotly_geopos(ggobj6)
#
#
# ## alternative:
# pols_df <- get_geopos(nc_file)
# ggplot_geopos(pols_df)
#
#
# ## example 3) probability surfaces of horizontal tracks from kmz-file:
# kmz_file <- system.file("example_files/15P1019-104659-1-GPE3.kmz",package="RchivalTag")
# ggobj7 <- ggplot_geopos(kmz_file)
# ggobj7
# ggplotly_geopos(ggobj7)
#
#
# kmz_file2 <- system.file("example_files/15P0986-15P0986-2-GPE3.kmz",package="RchivalTag")
# ggobj8 <- ggplot_geopos(kmz_file)
# ggobj8
# ggplotly_geopos(ggobj8)
#
# ## example 4) combine polygon tracks:
# k1 = get_geopos(kmz_file)
```

```
# k2 = get_geopos(kmz_file2)
#
# ggobj <- ggplotmap("mednw4")
# ## p1 <- ggplot_geopos(k1,ggobj = ggobj) ## not working, need to change date format:
# p1 <- ggplot_geopos(k1,date_format = "%d-%b-%Y %H:%M:%S")
# p1
# p2 <- ggplot_geopos(k2,p1,zlim = as.Date(range(c(k1$datetime,k2$datetime))),
#       date_format = "%d-%b-%Y %H:%M:%S")
# ggplotly_geopos(p2)
#
# ## change plot window:
# p1b <- ggplot_geopos(k1,ggobj = ggobj, date_format = "%d-%b-%Y %H:%M:%S")
# p2b <- ggplot_geopos(k2,p1b,zlim = as.Date(range(c(k1$datetime,k2$datetime))),
#                      date_format = "%d-%b-%Y %H:%M:%S")
# p2b
# ggplotly_geopos(p2b)
```

`ggplot_geopos` *Reads and plots geolocation estimates derived from archival tagging data*

Description

Geolocations provided via CSV or data frames are plotted as line or scatter plots. If netCDF (.nc) or KMZ files from the Wildlife Computers portal are selected, a SpatialPolygonsDataFrame is created and surface probability maps are illustrated. The netCDF transformation follows the R-code in the Location Processing User Guide. KMZ files already include contour lines of the 50, 95, and 99% likelihood areas, which are extracted and transformed to a SpatialPolygonsDataFrame.

Usage

```
ggplot_geopos(x, ggobj, xlim, ylim, zlim, standard_year=FALSE,
              full_year=standard_year, date_format, lang_format="en", tz="UTC",
           Breaks, cb.title, cb.date_format, cbpos, cb.height = 10, cb.xlab = "",
         cb.reverse=FALSE, pal.reverse=cb.reverse, prob_lim=.75, color_by="date",
              pal, alpha=70, type="p",
              main ,lwd=1, size=2, shape=19, verbose= FALSE, ...)

get_geopos(x, xlim, ylim, date_format, lang_format="en", tz="UTC",
           proj4string, prob_lim=.5, verbose=TRUE)
```

Arguments

`x`	data.frame containing horizontal position records, or a path/filename of `.csv`, `.kmz`, `.kml`, or `.nc`-files.
`ggobj`	A ggplot object (optional base plot).
`xlim, ylim`	Numeric vectors defining the limits of the x- and y-axes.

`zlim`, `Breaks`, `standard_year`, `full_year`
: Date range and breaks of the colorbar. If `standard_year = TRUE`, positions are standardized yearly and tick breaks are ignored. If `full_year = TRUE`, the color scale covers all months from January to December.

`date_format`, `lang_format`, `tz`
: character strings indicating the date format, language format and the corresponding time zone, defined by the vectors Date and Time (by default: date_format="%d-%b-%Y %H:%M:%S", lang_format="en", tz='UTC') If formatting fails, please check as well the input language format, defined by `lang_format` (and use abrviations such as "en" for English,"es" for Spanish, "fr" for French, etc.) as well.

`proj4string`
: Coordinate reference system (CRS; projection).

`cb.title`
: Title of the colorbar (default "Date").

`cb.date_format`
: Date format of the colorbar ticks (default "%Y-%m-%d").

`cbpos`, `cb.xlab`, `cb.height`
: Position, x-axis label, and height of the colorbar.

`prob_lim`
: For kmz, kml, or netCDF (.nc) files, defines the probability surface limit in %. Default 50%. Valid KMZ/KML values: 50, 95, 99.

`color_by`
: Column or vector used for coloring geolocations (default "date").

`pal`
: Color map used for polygon (.nc) or scatter plots. See cmap for pre-installed color maps.

`cb.reverse`, `pal.reverse`
: Inverse order of ticks and colormap of colorscale.

`alpha`
: Transparency of polygons or dots in percent (default 70).

`type`
: Plot type: "p" for points (default), "l" for lines, "b" for both.

`size`, `shape`
: Point size and type (default 2 and 19 for solid dots).

`lwd`
: Line width.

`...`
: Additional arguments passed to ggplotmap.

`main`
: Overall plot title.

`verbose`
: Whether file names should be printed when loading geolocation files (default TRUE for `get_geopos()`, FALSE for `ggplot_geopos()`).

Author(s)

Robert K. Bauer

See Also

leaflet_geopos, ggplotly_geopos, ggplotmap, SpatialPolygonsDataFrame

Examples

```
# Example 1a: Line plot from CSV
# library(oceanmap)
# csv_file <- system.file("example_files/15P1019-104659-1-GPE3.csv", package="RchivalTag")
# pos <- get_geopos(csv_file)
# ggobj <- ggplot_geopos(pos)
# ggobj
# ggplotly_geopos(ggobj)

# Example 1b: Scatter plot on existing landmask
# ggobj <- oceanmap::ggplotmap("lion", grid.res=5)
# ggobj4 <- ggplot_geopos(csv_file, ggobj)
# ggplotly_geopos(ggobj4)

# Example 2: Probability surfaces from netCDF
# nc_file <- system.file("example_files/15P1019-104659-1-GPE3.nc", package="RchivalTag")
# ggobj6 <- ggplot_geopos(nc_file)
# ggplotly_geopos(ggobj6)

# Example 3: Probability surfaces from KMZ
# kmz_file <- system.file("example_files/15P1019-104659-1-GPE3.kmz", package="RchivalTag")
# ggobj7 <- ggplot_geopos(kmz_file)
# ggplotly_geopos(ggobj7)
```

`hist_tad` *Time-at-Depth histogram*

Description

generates daily or back-to-back (e.g. Day-vs-Night-) Time-at-Depth histograms from binned depth or depth time series data

Usage

```
hist_tad(df,
            bin_breaks=NULL, bin_prefix="Bin",
            select_id, select_from='Ptt', aggregate_by='Ptt',
            date, min_perc,
            main, xlab='Time at Depth (%)', ylab="Depth (m)", labeling=TRUE,
            xlim=c(0, 100), adaptive.xlim=FALSE,
            split_by=NULL, split_levels, xlab2=split_levels,
            ylab.side=2, ylab.line, ylab.font=1,
            xlab.side=3, xlab.line=2.5, xlab.font=1,
            xlab2.side=1, xlab2.line=1, xlab2.font=2,
            main.side=3, main.line=3.8, main.font=1,
            col=c("darkgrey", "white"),
            xticks, ylabels,
            do_mid.ticks=TRUE, yaxis.pos=0,
```

```
mars, space=0,
plot_sd=TRUE, plot_se, plot_nrec=TRUE, plot_ntags=TRUE,
cex=1.2, cex.main=cex, cex.lab=cex, cex.inf=cex.axis,cex.axis=1,
return.sm=FALSE,
subplot=FALSE, inside=FALSE,Type="TAD")
```

Arguments

df	dataframe that either contains depth time series data (as a vector "Depth") or several vectors of Time-at-Depth frequencies. In the latter case, column names composed of a common bin_prefix (default is "Bin.") hold the pre-binned Time-at-Depth frequencies whose depth limits are defined in bin_breaks.
bin_breaks, bin_prefix	bin_breaks is a numeric vector of depth bin breaks for the histogram data. In case of binned data (e.g. from standard wildlife computer histogram files), column names with a bin_prefix are expected to contain the preprossesed data (by default: Bin1, Bin2, Bin3, etc.). Alternatively, depth time series data will be directly converted using function ts2histos.
select_id, select_from	these arguments allow to take a direct subset of the input dataframe. select_from defines the vector whereas select_id defines the identification key(s) that should be selected.
aggregate_by	character vector defining the columns by which the tagging data should be aggregated. Should contain columns that identify tags (e.g. Serial, Ptt, DeployID) the date and/or day time period (to seperate records from night, day, dawn and dusk see classify_DayTime). Default values are: date, Day and Ptt.
date	An optional vector to select depth data of a specified date/-range.
min_perc	optional number, defining the minimum data coverage (in percent) of histogram entries obtained from depth time series data.
main, xlab, ylab, labeling	The titles for the plot, x- and y-axes to be plotted if labeling is set TRUE (default).
xlim, adaptive.xlim	a vector defining the limits (x1,x2) of the x-axis, by default c(0,100). However, if adaptive.xlim is set TRUE, these limits will be overwritten, and the maximum value (xlim[2]) will be chosen from the histogram data.
split_by	Name of the logical vector by which TaD data should be splitted (e.g. daytime; see classify_DayTime).
split_levels, xlab2	Character vector defining the name and order of the levels of the split_by vector (e.g. c("Night", "Day") for split_by vector 'day.time'. The same groups are plotted as a second x-axis label if not defined otherwise (xlab2=split_levels).
ylab.side, ylab.line, ylab.font	side, line and font of second y-axis label.
xlab.side, xlab.line, xlab.font	side, line and font of first x-axis label.

`xlab2.side`, `xlab2.line`, `xlab2.font`
: side, line and font of second x-axis labels.

`main.side`, `main.line`, `main.font`
: side, line and font of plot title.

`col`
: colours to be used for the TaD-histogram, by default 'grey' and 'white' (corresponding to the values of split_by/split_levels).

`xticks`, `ylabels`
: tick labels of the x-axis and ylabels of the y-axis to show in the plot.

`do_mid.ticks`
: whether centered tick-labels, indicating the depth range of histogram cells, shall be plotted (by default `FALSE`). Alternatively, tick labels will be indicated at the breakpoints of the histrogram cells.

`yaxis.pos`
: x-axis coordinate at which the y-axis should be plotted (by default xlim[1], and thus 0).

`mars`
: a numerical vector of the form `c(bottom, left, top, right)`, decribing the number of margin lines to be specified on the each side of the plot. The default is `c(2.1, 4.1, 6.1, 2.1)`. In case that `do_mid.ticks` is `TRUE` margins are: `c(2.1, 8, 6.1, 2.1)`.

`space`
: the space between the histogram bars.

`plot_sd`, `plot_se`, `plot_nrec`, `plot_ntags`
: whether standard deviation or standard error bars, the number of records and tags shall be plotted (default is `TRUE`) inside the TaD/TaT histogram. (if plot_se is `TRUE`, plot_sd will be overwritten!).

`cex`, `cex.main`, `cex.lab`, `cex.inf`, `cex.axis`
: font size of the title (`cex.main`), x- and y-axes labels (`cex.lab`), other labels, like the number of records (`cex.inf`) as well as of the tick marks.

`return.sm`
: whether summary information of the TaD histograms, including the number of records per summary period, the relative frequencies per bin and corresponding standard deviation, should be plotted (default is `TRUE`).

`subplot`, `inside`
: whether the TaD histogram is a subplot or an inner plot of a figure (default is `FALSE`). If `subplot` or `inside` are set `TRUE`, graphic margins will not be set by `hist_tad`. In case that `inside` is `TRUE`, no axis-labels and titels wil be plotted.

`Type`
: The Type of data to be plotted (`TAD`: Time-at-Depth histograms; `TAT`: Time-at-Temperature histograms)

Details

Time-at-Temperature (Tat) and Time-at-Depth (TaD) fequencies are a standard data product of archival tags (incl. tag models TDR-Mk9, PAT-Mk10 and miniPAT by Wildlife Computers) that allow to assess habitat preferences of tagged animals (see function read_histos). It can be likewise generated from transmitted or recovered time series data sets using function ts2histos.

However, different depth and temperature bin breaks are often used during different deployment programs, which makes a later comparitive analysis of TaT and TaT data difficult. For such cases, the function combine_histos and merge_histos can be applied to merge TaT and TaD frequencies based on common bin breaks of different tags.

The purpose of this function is the visualization of Time-at-Depth (TaD) histograms, whereas hist_tad is the related function for Time-at-Temperature (TaT) data.

Author(s)

Robert K. Bauer

See Also

ts2histos, combine_histos, merge_histos, hist_tat

Examples

```
ts_file <- system.file("example_files/104659-Series.csv",package="RchivalTag")
ts_df <- read_TS(ts_file)
head(ts_df)

tad_breaks <- c(0, 2, 5, 10, 20, 50, 100, 200, 300, 400, 600, 2000)
tat_breaks <- c(10,12,15,17,18,19,20,21,22,23,24,27)

## example 1a) convert only DepthTS data to daily TAD frequencies:
ts2histos(ts_df, tad_breaks = tad_breaks)
hist_tad(ts_df, bin_breaks = tad_breaks)
hist_tad(ts_df, bin_breaks = tad_breaks, do_mid.ticks = FALSE)

## convert 1b) only TemperatureTS data to daily TAT frequencies:
tat <- ts2histos(ts_df, tat_breaks = tat_breaks)
hist_tat(ts_df, bin_breaks = tat_breaks, do_mid.ticks = FALSE)
hist_tat(tat$TAT$merged, do_mid.ticks = FALSE)

## convert 1c) DepthTS & TemperatureTS data to daily TAD & TAT frequencies:
ts2histos(ts_df, tad_breaks = tad_breaks, tat_breaks = tat_breaks)

## convert 1d) back-to-back histogram showing day vs night TAD frequencies:
ts_df$Lat <- 4; ts_df$Lon=42.5 ## required geolocations to estimate daytime
head(ts_df)
ts_df2 <- classify_DayTime(get_DayTimeLimits(ts_df)) # estimate daytime
head(ts_df2)

ts2histos(ts_df2, tad_breaks = tad_breaks,split_by = "daytime")
hist_tad(ts_df2, bin_breaks = tad_breaks,split_by = "daytime", do_mid.ticks = FALSE)


## example 2) rebin daily TAD frequencies:
tad <- ts2histos(ts_df, tad_breaks = tad_breaks)
tad2 <- rebin_histos(hist_list = tad, tad_breaks = tad_breaks[c(1:3,6:12)])
par(mfrow=c(2,2))
hist_tad(tad, do_mid.ticks = FALSE) ## example for multiple individuals
hist_tad(tad$TAD$merged, do_mid.ticks = FALSE)
hist_tad(tad$TAD$merged, bin_breaks = tad_breaks[c(1:3,6:12)]) ## from inside hist_tad


## example 3) read, merge and plot TAD frequency data from several files:
## part I - read histogram data from two files:
hist_dat_1 <- read_histos(system.file("example_files/104659-Histos.csv",package="RchivalTag"))
hist_dat_2 <- read_histos(system.file("example_files/104659b-Histos.csv",package="RchivalTag"))
```

```
## note the second list is based on the same data (tag), but on different bin_breaks

## part II - combine TAD/TAT frecuency data from seperate files in one list:
hist_dat_combined <- combine_histos(hist_dat_1, hist_dat_2)
par(mfrow=c(2,1))
hist_tad(hist_dat_combined)
hist_tat(hist_dat_combined)

## part III - force merge TAD/TAT frecuency data from seperate files
# in one list, by applying common bin_breaks:
hist_dat_merged <- merge_histos(hist_dat_combined,force_merge = TRUE)
hist_tad(hist_dat_merged)
hist_tat(hist_dat_merged)

## part IV - plot merged data:
hist_tad(hist_dat_merged) # of all tags
unique(hist_dat_merged$TAD$merged$df$DeployID) ## list unique tags in merged list
hist_tad(hist_dat_merged, select_id = "15P1019b", select_from = 'DeployID') # of one tag

## part V - unmerge data:
unmerge_histos(hist_dat_merged)
```

`hist_tat` *Time-at-Temperature histogram*

Description

generates daily or back-to-back (e.g. Day-vs-Night-) Time-at-Temperature histograms from binned Temperature or Temperature time series data

Usage

```
hist_tat(df,
            bin_breaks=NULL, bin_prefix="Bin",
            main, xlab="Time at Temperature (%)",
            ylab=expression(paste("Temperature (",degree,"C)")), labeling=TRUE,
            Type="TAT", ...)
```

Arguments

`df` dataframe that either contains Temperature time series data (as a vector "Temperature") or several vectors of Time-at-Temperature frequencies. In the latter case, vector names are composed of a common `bin_prefix` (default is "tad."), followed by the upper Temperature limit (bin break).

`bin_breaks`, `bin_prefix`
`bin_breaks` is a numeric vector of depth bin breaks for the histogram data. In case of binned data (e.g. from standard wildlife computer histogram files), column names with a `bin_prefix` are expected to contain the preprossesed data (by default: Bin1, Bin2, Bin3, etc.). Alternatively, depth time series data will be directly converted using function ts2histos.

`main`, `xlab`, `ylab`, `labeling`
: The titles for the plot, x- and y-axes to be plotted if `labeling` is set `TRUE` (default).

`Type`
: The Type of data to be plotted (`TAD`: Time-at-Depth histograms; `TAT`: Time-at-Temperature histograms)

`...`
: additional arguments to be passed:

select_id, select_from these arguments allow to take a direct subset of the input dataframe. `select_from` defines the vector whereas `select_id` defines the identification key(s) that should be selected.

aggregate_by character vector defining the columns by which the tagging data should be aggregated. Should contain columns that identify tags (e.g. Serial, Ptt, DeployID) the date and/or day time period (to seperate records from night, day, dawn and dusk see classify_DayTime). Default values are: date, Day and Ptt.

date An optional vector to select depth data of a specified date/-range.

xlim, adaptive.xlim a vector defining the limits (x1,x2) of the x-axis, by default c(0,100). However, if `adaptive.xlim` is set `TRUE`, these limits will be overwritten, and the maximum value (xlim[2]) will be chosen from the histogram data.

split_by Name of the logical vector by which TaD data should be splitted (e.g. daytime; see classify_DayTime).

split_levels, xlab2 Character vector defining the name and order of the levels of the split_by vector (e.g. c("Night", "Day") for split_by vector 'day.time'. The same groups are plotted as a second x-axis label if not defined otherwise (`xlab2=split_levels`).

ylab.side, ylab.line, ylab.font side, line and font of second y-axis label.

xlab.side, xlab.line, xlab.font side, line and font of first x-axis label.

xlab2.side, xlab2.line, xlab2.font side, line and font of second x-axis labels.

main.side, main.line, main.font side, line and font of plot title.

col colours to be used for the TaD-histogram, by default 'grey' and 'white' (corresponding to the values of split_by/split_levels).

xticks, ylabels tick labels of the x-axis and ylabels of the y-axis to show in the plot.

do_mid.ticks whether centered tick-labels, indicating the depth range of histogram cells, shall be plotted (by default `FALSE`). Alternatively, tick labels will be indicated at the breakpoints of the histrogram cells.

yaxis.pos x-axis coordinate at which the y-axis should be plotted (by default xlim[1], and thus 0).

mars a numerical vector of the form `c(bottom, left, top, right)`, decribing the number of margin lines to be specified on the each side of the plot. The default is `c(2.1, 4.1, 6.1, 2.1)`. In case that `do_mid.ticks` is `TRUE` margins are: `c(2.1, 8, 6.1, 2.1)`.

space the space between the histogram bars.

plot_sd, plot_se, plot_nrec , plot_ntags whether standard deviation or standard error bars, the number of records and tags shall be plotted (default is `TRUE`)

inside the TaD/TaT histogram. (if plot_se is TRUE, plot_sd will be overwritten!).

cex, cex.main, cex.lab, cex.inf font size of the title (cex.main), x- and y-axes labels (cex.lab), and other labels, like the number of records (cex.inf).

return.sm whether summary information of the TaD histograms, including the number of records per summary period, the relative frequencies per bin and corresponding standard deviation, should be plotted (default is TRUE).

subplot, inside whether the TaD histogram is a subplot or an inner plot of a figure (default is FALSE). If subplot or inside are set TRUE, graphic margins will not be set by hist_tat. In case that inside is TRUE, no axis-labels and titels wil be plotted.

Details

Time-at-Temperature (Tat) and Time-at-Depth (TaD) fequencies are a standard data product of archival tags (incl. tag models TDR-Mk9, PAT-Mk10 and miniPAT by Wildlife Computers) that allow to assess habitat preferences of tagged animals (see function read_histos). It can be likewise generated from transmitted or recovered time series data sets using function ts2histos.

However, different depth and temperature bin breaks are often used during different deployment programs, which makes a later comparitive analysis of TaT and TaT data difficult. For such cases, the function combine_histos and merge_histos can be applied to merge TaT and TaD frequencies based on common bin breaks of different tags.

The purpose of this function is the visualization of Time-at-Temperature (TaT) histograms, whereas hist_tad is the related function for Time-at-Depth (TaD) data.

Author(s)

Robert K. Bauer

See Also

ts2histos, combine_histos, merge_histos, hist_tad

Examples

```
ts_file <- system.file("example_files/104659-Series.csv",package="RchivalTag")
ts_df <- read_TS(ts_file)
head(ts_df)

tad_breaks <- c(0, 2, 5, 10, 20, 50, 100, 200, 300, 400, 600, 2000)
tat_breaks <- c(10,12,15,17,18,19,20,21,22,23,24,27)

## example 1a) convert only DepthTS data to daily TAD frequencies:
ts2histos(ts_df, tad_breaks = tad_breaks)
# hist_tad(ts_df, bin_breaks = tad_breaks)
hist_tad(ts_df, bin_breaks = tad_breaks, do_mid.ticks = FALSE)

## convert 1b) only TemperatureTS data to daily TAT frequencies:
tat <- ts2histos(ts_df, tat_breaks = tat_breaks)
```

```
hist_tat(ts_df, bin_breaks = tat_breaks, do_mid.ticks = FALSE)
hist_tat(tat$TAT$merged, do_mid.ticks = FALSE)

## convert 1c) DepthTS & TemperatureTS data to daily TAD & TAT frequencies:
ts2histos(ts_df, tad_breaks = tad_breaks, tat_breaks = tat_breaks)

## convert 1d) back-to-back histogram showing day vs night TAD frequencies:
ts_df$Lat <- 4; ts_df$Lon=42.5 ## required geolocations to estimate daytime
head(ts_df)
ts_df2 <- classify_DayTime(get_DayTimeLimits(ts_df)) # estimate daytime
head(ts_df2)

ts2histos(ts_df2, tad_breaks = tad_breaks,split_by = "daytime")
hist_tad(ts_df2, bin_breaks = tad_breaks,split_by = "daytime", do_mid.ticks = FALSE)


## example 2) rebin daily TAD frequencies:
tad <- ts2histos(ts_df, tad_breaks = tad_breaks)
tad2 <- rebin_histos(hist_list = tad, tad_breaks = tad_breaks[c(1:3,6:12)])
par(mfrow=c(2,2))
hist_tad(tad, do_mid.ticks = FALSE) ## example for multiple individuals
hist_tad(tad$TAD$merged, do_mid.ticks = FALSE)
hist_tad(tad$TAD$merged, bin_breaks = tad_breaks[c(1:3,6:12)]) ## from inside hist_tad


## example 3) read, merge and plot TAD frequency data from several files:
## part I - read histogram data from two files:
hist_dat_1 <- read_histos(system.file("example_files/104659-Histos.csv",package="RchivalTag"))
hist_dat_2 <- read_histos(system.file("example_files/104659b-Histos.csv",package="RchivalTag"))
## note the second list is based on the same data (tag), but on different bin_breaks

## part II - combine TAD/TAT frecuency data from seperate files in one list:
hist_dat_combined <- combine_histos(hist_dat_1, hist_dat_2)
par(mfrow=c(2,1))
hist_tad(hist_dat_combined)
hist_tat(hist_dat_combined)

## part III - force merge TAD/TAT frecuency data from seperate files
# in one list, by applying common bin_breaks:
hist_dat_merged <- merge_histos(hist_dat_combined,force_merge = TRUE)
hist_tad(hist_dat_merged)
hist_tat(hist_dat_merged)

## part IV - plot merged data:
hist_tad(hist_dat_merged) # of all tags
unique(hist_dat_merged$TAD$merged$df$DeployID) ## list unique tags in merged list
hist_tad(hist_dat_merged, select_id = "15P1019b", select_from = 'DeployID') # of one tag

## part V - unmerge data:
unmerge_histos(hist_dat_merged)
```

`image_TempDepthProfiles` *plots interpolated daily temperature at depth profiles*

Description

plots interpolated daily temperature at depth profiles, thus faciliating the analysis of temporal changes of temperature profiles, for instance, in relation to animal behaviour (e.g. diving behaviour). See Bauer et al. (2015) for further examples.

Usage

```
image_TempDepthProfiles(x, main=NULL, xlab='Date', ylab="Depth (m)",
                        cb.xlab=expression(paste("Temperature (",degree,"C)")),
                        cex.cb.xlab=1, cex.cb.ticks=1,
                        xlim, ylim, zlim, pal="jet", only.months, month.line=0,
                        mars, axes=TRUE, do.colorbar=TRUE, ...)
```

Arguments

`x`	A list , generated by interpolate_TempDepthProfiles or interpolate_PDTs, containing interpolated temperature at depth profiles and their corresponding date and interpolated depth values as well as a summary table with the original depth values and their number per day: $ Temperature_matrix: num $ Depth : num $ Date :Date $ sm :data.frame
`main`, `xlab`, `ylab`	the title, x- and y-axis labels to be plotted.
`cb.xlab`	character string indicating the x-axis label of the colorbar.
`cex.cb.xlab`, `cex.cb.ticks`	*cex.cb.xlab*: font size of the x-axis label of the colorbar (by default 1). *cex.cb.ticks*: font size of the x-axis tick labels of the colorbar (by default 1).
`xlim`, `ylim`, `zlim`	the x, y and z limits of the plot.
`pal`	color map to be plotted (default is `'jet'`). See cmap for available color maps.
`only.months`, `month.line`	whether only mid-months shall be plotted as tick labels of the x-axis (by default FALSE for time ranges of less than 3 months (93 days)). In case, that only.months is set `TRUE`, `month.line` defines the line where the month labels shall be plotted.
`mars`	a numerical vector defining the plot margins `c(bottom, left, top, right)` (by default `c(5,4,4,9)`).
`axes`, `do.colorbar`	whether the axes and colorbar should be plotted.
`...`	additional arguments to be passed to set.colorbarp

Author(s)

Robert K. Bauer

References

Bauer, R., F. Forget and JM. Fromentin (2015) Optimizing PAT data transmission: assessing the accuracy of temperature summary data to estimate environmental conditions. Fisheries Oceanography, 24(6): 533-539, doi:10.1111/fog.12127

See Also

read_PDT, bin_TempTS, get_thermalstrat, image_TempDepthProfiles

Examples

```
#### example 1) run on PDT file:
## step I) read sample PDT data file:
path <- system.file("example_files",package="RchivalTag")
PDT <- read_PDT("104659-PDTs.csv",folder=path)
head(PDT)
#
# ## step II) interpolate average temperature fields (MeanPDT) from PDT file:
# m <- interpolate_PDTs(PDT)
# str(m)
# m$sm
#
# ## step III) calculate thermal stratifcation indicators per day (and tag):
# get_thermalstrat(m, all_info = TRUE)
# get_thermalstrat(m, all_info = FALSE)
#
# ## step IV) plot interpolated profiles:
# image_TempDepthProfiles(m$station.1)
#
#
# #### example 2) run on time series data:
# ## step I) read sample time series data file:
# DepthTempTS <- read.table(system.file("example_files/104659-Series.csv",
#                                       package="RchivalTag"),header = TRUE,sep=',')
# DepthTempTS$date <- as.Date(DepthTempTS$Day,"%d-%b-%Y")
# head(DepthTempTS)
#
#
# ## step Ib) bin temperature data on 10m depth bins
# ##          to increase later estimate accuracy (see Bauer et al. 2015):
# # DepthTempTS_binned <- bin_TempTS(DepthTempTS,res=10)
#
# ## step II) interpolate average temperature fields (MeanTemp) from binned data:
# m <- interpolate_TempDepthProfiles(DepthTempTS)
# # m <- interpolate_PDTs(DepthTempTS_binned)
# str(m)
# m$sm
#
```

```
# ## step III) calculate thermal stratifcation indicators per day (and tag):
# get_thermalstrat(m, all_info = TRUE)
# get_thermalstrat(m, all_info = FALSE)
#
# ## step IV) plot interpolated profiles:
# image_TempDepthProfiles(m$station.1)
```

`interpolate_TempDepthProfiles` *interpolate daily temperature at depth profiles*

Description

interpolates depth-temperature data and returns daily average temperature at depth profiles on a user-specified resolution (`Depth_res`).
Results are returned as a list containing the interpolated Temperature-matrix, and the corresponding date and depth values. Thus interpolated temperature at depth profiles can be visualized using function image_TempDepthProfiles and faciliates the analysis of temporal changes of temperature profiles, for instance, in relation to animal behaviour (e.g. diving behaviour).

Usage

```
interpolate_TempDepthProfiles(ts, Temp_field="Temperature", ID_key="Serial",
                              Depth_res=.5, verbose=TRUE, Data_Source='station')

interpolate_PDTs(ts, Temp_field="MeanPDT", ID_key="Serial", #return_as_matrix=FALSE,
                              Depth_res=.5, verbose=TRUE, Data_Source='station')
```

Arguments

`ts`, `Temp_field`, `ID_key`
: `ts` is a data.frame with temperature at depth data. Required columns are `Depth` for the depth data and a column containing temperature data, whose name is defined by `Temp_field`. `ID_key` specifies the name of an optional column on which sampling stations or tags can be distinguished (by default `Serial`).

`Depth_res`
: numeric value, defining the depth resolution at which the temperature data should be interpolated.

`verbose`
: whether the sampling dates and ids of stations or tags, as defined by the columns `date` and `ID_key`, should be indicated during the interpolation process.

`Data_Source`
: a character string, defining the data source (by default `station`).

Value

A list containing the interpolated temperature at depth profiles and their corresponding date and interpolated depth values as well as a summary table with the original depth values and their number per day:

$ Data_Source.ID_key:List of 4
..$ Temperature_matrix: num
..$ Depth : num
..$ Date :Date
..$ sm :data.frame

Please see the examples for further understaning.

Author(s)

Robert K. Bauer

References

Bauer, R., F. Forget and JM. Fromentin (2015) Optimizing PAT data transmission: assessing the accuracy of temperature summary data to estimate environmental conditions. Fisheries Oceanography, 24(6): 533-539, doi:10.1111/fog.12127

See Also

read_PDT, bin_TempTS, get_thermalstrat, image_TempDepthProfiles

Examples

```
#### example 1) run on PDT file:
## step I) read sample PDT data file:
path <- system.file("example_files",package="RchivalTag")
PDT <- read_PDT("104659-PDTs.csv",folder=path)
head(PDT)
#
# ## step II) interpolate average temperature fields (MeanPDT) from PDT file:
# m <- interpolate_PDTs(PDT)
# str(m)
# m$sm
#
# ## step III) calculate thermal stratifcation indicators per day (and tag):
# get_thermalstrat(m, all_info = TRUE)
# get_thermalstrat(m, all_info = FALSE)
#
# ## step IV) plot interpolated profiles:
# image_TempDepthProfiles(m$station.1)
#
#
# #### example 2) run on time series data:
# ## step I) read sample time series data file:
# ts_file <- system.file("example_files/104659-Series.csv",package="RchivalTag")
```

```
# DepthTempTS <- read_TS(ts_file)
#
#
# ## step Ib) bin temperature data on 10m depth bins
# ##          to increase later estimate accuracy (see Bauer et al. 2015):
# # DepthTempTS_binned <- bin_TempTS(DepthTempTS,res=10)
#
# ## step II) interpolate average temperature fields (MeanTemp) from binned data:
# m <- interpolate_TempDepthProfiles(DepthTempTS)
# # m <- interpolate_PDTs(DepthTempTS_binned)
# str(m)
# m$sm
#
# ## step III) calculate thermal stratifcation indicators per day (and tag):
# get_thermalstrat(m, all_info = TRUE)
# get_thermalstrat(m, all_info = FALSE)
#
# ## step IV) plot interpolated profiles:
# image_TempDepthProfiles(m$station.1)
```

`leaflet_geopos` *reads and plots geolocation estimates derived from archival tagging data*

Description

This function creates a Leaflet map widget using htmlwidgets from spatial tagging data. The widget can be rendered on HTML pages generated from R Markdown, Shiny, or other applications.

Usage

```
leaflet_geopos(data, ID_label, add_label=NULL, except_label=NULL,
               collapsedLayers=TRUE, radius=1000, pal,
           layer_title=ID_label, colorby="date", cb.title, cbpos="bottomright",
               showScaleBar=TRUE, showSlideBar=FALSE)
```

Arguments

`data`	spatial data such as data.frame or SpatialPolygonsDataFrame.
`ID_label`	Vector in spatial data that defines the ID of the tracks to be plotted.
`add_label, except_label`	additional labels or labels to be removed from hover.
`collapsedLayers`	whether to collapse leaflet layer legend.
`radius`	circle size, in case of simple Lon/Lat tracking dataframes.

`pal` color map to be plotted in case of polygon (.nc-files) or scatter plots (default is the 'jet'-colormap, and 'year.jet' in case `standard_year` & `full_year` are set `TRUE`). See cmap for pre-installed color maps. Note that tracking data with constant time steps is being assumed in the color assignment. To verify this, a data.frame containing the colors at each time steps will be returned for polygon and scatter plots.

`layer_title` character string indicating the title of the layer legend (by default defined by `'ID_label'`.)

`colorby` character string indicating the vector for which to apply the colorbar (by default `'date'`.)

`cb.title` character string indicating the title of the colorbar (by default inferred from `colorby`),

`cbpos` position of the colorbar (by default `'bottomright'`.)

`showScaleBar, showSlideBar`
whether to show the scale bar or the slide bar.

Author(s)

Robert K. Bauer

See Also

ggplotly_geopos, ggplot_geopos, update_leaflet_elementId

Examples

```
# csv_file <- system.file("example_files/15P1019-104659-1-GPE3.csv",package="RchivalTag")
# s0 <- get_geopos(csv_file)
# ggplot_geopos(s0)
# leaflet_geopos(s0,ID_label="DeployID")
# leaflet_geopos(s0,ID_label="DeployID",showSlideBar = T)
#
# kmz_file <- system.file("example_files/15P1019-104659-1-GPE3.kmz",package="RchivalTag")
# k1 <- get_geopos(kmz_file)
# kmz_file2 <- system.file("example_files/15P0986-15P0986-2-GPE3.kmz",package="RchivalTag")
# k2 <- get_geopos(kmz_file2)
# k0 <- k3 <- rbind(k1,k2)
# ggplot_geopos(k0,ggobj = ggplotmap("lion"))
#
# # ggobj <- ggplot_geopos(k1)
# # ggplot_geopos(ggobj = ggobj,k2)
# leaflet_geopos(k0,ID_label="DeployID",collapsedLayers = F) %>% addMiniMap()
# leaflet_geopos(k0,ID_label="DeployID",showSlideBar = T)
#
## Code to illustrate how to avoid rendering issues in RMarkdown:
## only valid in RMarkdown chunks:
# kmz_file2 <- system.file("example_files/15P0986-15P0986-2-GPE3.kmz",package="RchivalTag")
# k2 <- get_geopos(kmz_file2)
# k0 <- k3 <- rbind(k1,k2)
#
```

```
# library(leaflet)
# map <- leaflet_geopos(k0, ID_label="DeployID", collapsedLayers = F)
# map
# map # plot again to show rendering issues (in the layer menu title)
## this is required to avoid rendering issues when plotting the same map twice via RMarkdown
# map <- update_leaflet_elementId(map)
#
# plot again with updated elementID:
# map %>% addMiniMap()
```

`merge_histos` *merge and/or rebin TAD/TAT-frequency data*

Description

The joint analysis of archival tagging data from different tagging programs is often hampered by differences in the tags' setups, e.g. by the user-specified temporal resolution of time series data or the definition of summary data products. The latter particularly concerns different selected bin breaks of Time-at-Depth (TAD) and Time-at-Temperature (TAT) frequency data from archival tags by Wildlife Computers.

The purpose of this function is to allow:

1) a grouping of TAD and TAT data from multiple tags based on similiar bin breaks (For this, run the function with default statements, i.e. `force_merge` is `FALSE`),

2) merging (rebinning) of TAD and TAT data from multiple tags based on the bin breaks that all tags have in common (To do so, run the function with `force_merge` set `TRUE`).

3) merging (rebinning) of TAD and TAT data from multiple tags based on new user-specified `tad_breaks` and/or `tat_breaks`. In this case, the `force_merge`-statements `TRUE` and `FALSE` will omit or seperately group tags that do not share all user-specified bin breaks, respectively.

To combine of TAD/TAT data of several `hist_lists`, see combine_histos.

To visualize Time-at-Temperature (TaT) and Time-at-Depth (TaD) data, please see hist_tat and hist_tad, respectively.

Usage

```
merge_histos(hist_list, tad_breaks=NULL, tat_breaks=NULL, force_merge=FALSE)
rebin_histos(hist_list, tad_breaks=NULL, tat_breaks=NULL, force_merge=FALSE)
```

Arguments

`hist_list`	A list-of-lists containing the TAD and TAT frequency data and the corresponding `bin_breaks` from one or several tags.

`tad_breaks` a numeric vector defining the `bin_breaks` for the merging (rebinning) of the TAD frequency data.
In case that the additional argument `force_merge` is set `TRUE`, only tags whose original TAD bin breaks included all of the user-specified `tad_breaks` will be merged in a single group ('merged') based on the new bin breaks, while other tags will be omitted in the output. By contrast, if `force_merge` is set `FALSE`, tags that do not contain all specified `tad_breaks` will be merged in seperate groups (group2, group3, etc.), based on similar `bin_breaks`.

`tat_breaks` a numeric vector defining the `bin_breaks` for the merging (rebinning) of the TAT frequency data.
In case that the additional argument `force_merge` is set `TRUE`, only tags whose original TAT bin breaks included all of the user-specified `tat_breaks` will be merged in a single group ('merged') based on the new bin breaks, while other tags will be omitted in the output. By contrast, if `force_merge` is set `FALSE`, tags that do not contain all specified `tat_breaks` will be merged in seperate groups (group2, group3, etc.), based on similar `bin_breaks`.

`force_merge` If `FALSE` (default), groups of tags with similar TAD and TAT-`bin_breaks` will be combined (no merging on new bin breaks) and identifier labels renamed as group1, group2, etc.
If set `TRUE`, TAD and TAT frequency data will be merged on user-specified `tad/tat_breaks` or, if those arguments are missing, on the `bin_breaks` that all tags have in common. In both latter cases, identifier labels will be renamed "merged".

Value

A list-of-lists of grouped or merged TAD and TAT frequency data.

$ TAD:List
..$ group1 : List of 2
.. ..$ bin_breaks: num
.. ..$ df : data.frame
$ TAT:List
..$ group1 : List of 2
.. ..$ bin_breaks: num
.. ..$ df : data.frame
..$ group2 : List of 2
...

Author(s)

Robert K. Bauer

See Also

unmerge_histos, combine_histos, hist_tad

Examples

```
## example 1) read, merge and plot TAD frequency data from several files:
```

```
## part I - read histogram data from two files:
hist_dat_1 <- read_histos(system.file("example_files/104659-Histos.csv",package="RchivalTag"))
hist_dat_2 <- read_histos(system.file("example_files/104659b-Histos.csv",package="RchivalTag"))
## note the second list is based on the same data (tag), but on different bin_breaks

## part II - combine TAD/TAT frecuency data from seperate files in one list:
hist_dat_combined <- combine_histos(hist_dat_1, hist_dat_2)
par(mfrow=c(2,1))
hist_tad(hist_dat_combined)
hist_tat(hist_dat_combined)

## part III - force merge TAD/TAT frecuency data from seperate files
# in one list, by applying common bin_breaks:
hist_dat_merged <- merge_histos(hist_dat_combined,force_merge = TRUE)
hist_tad(hist_dat_merged)
hist_tat(hist_dat_merged)

## part IV - plot merged data:
hist_tad(hist_dat_merged) # of all tags
unique(hist_dat_merged$TAD$merged$df$DeployID) ## list unique tags in merged list
hist_tad(hist_dat_merged, select_id = "15P1019b", select_from = 'DeployID') # of one tag

## part V - unmerge data:
unmerge_histos(hist_dat_merged)
```

`plot_data_coverage` *Abacus plot to illustrate the data coverage of different data products per tag throughout their deployment period.*

Description

Abacus plot to illustrate the data coverage of different data products per tag throughout their deployment period.

Usage

```
plot_data_coverage(x, type, type2, meta,
                   Identifier="Serial", fields=c("Serial","Ptt"),
                   date_range_std, show_fullmonths=TRUE,
                   zlim, mars, na.omit=TRUE,
                   do.arrows=TRUE, xpos.arrows=-.25, xpos.years=-.27,
                   xpos.fields=c(-.01,-.12), ypos.fields,
                   main, cex.main=1.2,
                   cb.xlab, cex.cb.xlab=1,
                   cb.ticks, cex.cb.ticks=.9,
                   pal="jet", bg="grey")

abacus_plot(x, type, type2, meta,
            Identifier="Serial", fields=c("Serial","Ptt"),
```

```
            date_range_std, show_fullmonths=TRUE,
            zlim, mars, na.omit=TRUE,
            do.arrows=TRUE, xpos.arrows=-.25, xpos.years=-.27,
            xpos.fields=c(-.01,-.12), ypos.fields,
            main, cex.main=1.2,
            cb.xlab, cex.cb.xlab=1,
            cb.ticks, cex.cb.ticks=.9,
            pal="jet", bg="grey")
```

Arguments

x	a list with the tagging data whose data covereage will be illustrated as abacus plot
type, type2	the data type of x ("ts", "lightlocs","tad","tat") and the name of the variable to be plotted: "perc" in case of lightlocation and time series data, "perc_dat" in case of histogram data ("tad", "tat").
meta	a data.frame containing the meta data of the tagging program. Required column names include: 'dep.date', 'pop.date' as well as the definitions of the Identifier, fields vectors (see below).
Identifier	unit vector to identify the tags within the tagging data list (must appear in names of sublist) and meta table (column name). By default 'Serial' of the tag.
date_range_std, show_fullmonths	standardized date range to be plotted (deplyoment years are reset to 0; e.g 0-10-03 and 01-02-17 for a fish that was in the water from 2017-10-03 until 2018-02-17. If missing, the date range will be estimated from the tagging data. If show_fullmonths=TRUE, the estimated date range will be rounded to full months (e.g. 0-10-01 and 01-03-01). However, show_fullmonths is ignored if date_range_std is specified.
mars	a numerical vector of the form c(bottom, left, top, right), decribing the number of margin lines to be specified on the each side of the plot. The default is c(5,12,4,8). In case of length(fields) > 2 you may need to increase the space to the left as well as related arguments (i.e. xpos.arrows, xpos.years, xpos.fields), in order to provide the neccessary space for the columns of the meta data to be shown.
na.omit	whether missing data points within the time series shall be converted to 0. If FALSE, such missing points will be shaded in grey.
zlim	the minimum and maximum z values for which colors should be plotted. By default c(0,100).
do.arrows, xpos.arrows, xpos.years	whether arrows shall be shown next to the deployment years as well as the horizontal position of arrows and deployment years.
fields, xpos.fields, ypos.fields	vectors to define the column(s) in meta data to be illustrated to the left of the data coverage abacus plot as well as related horizontal and vertical positions.
main, cex.main	the title and it's font size. if missing, title will be set to "type".

`cb.xlab`, `cex.cb.xlab`
: the title of the colorbar and it's font size.

`cb.ticks`, `cex.cb.ticks`
: the tick labels of the colorbar and it's font size.

`pal`, `bg`
: the color palette of the colorbar (either a vector or a single keyword referring to a colorbar from the colorbars of the oceanmap package) as well as the background color of the data coverage abacus plot. Please note also, that if the argument 'na.omit' is set to FALSE, missing data points will be shaded in grey.

Details

Abacus plot to illustrate the data coverage of different data products per tag throughout their deployment period as shown in Figure 3 of Bauer et al. (2020).

Author(s)

Robert K. Bauer

References

Bauer, R., F. Forget, JM. Fromentin and M. Capello (2020) Surfacing and vertical behaviour of Atlantic bluefin tuna (Thunnus thynnus) in the Mediterranean Sea: implications for aerial surveys. ICES Journal of Marine Science. doi:10.1093/icesjms/fsaa083

See Also

read_histos, ts2histos, read_TS

Examples

```
# sample_file <- system.file("example_files/abacus_sample_data.rd",package="RchivalTag")
# load(sample_file, verbose=T)
#
# ## Please note: the sample data is contains only the columns required to produce the figures.
# ## Other fields (e.g. Bins and bin breaks in the histos data are missing).
# ## The basic structure has been mantained and needs to be adopted
# ## to apply the plot_data_coverage-function.
#
# str(meta) # meta data example with all required columns
# str(lightlocs) ## not yet implemented, but can structure be adapted for other data sets.
# str(histos) ## combined but not merged histogram data.
              ## Please compare with read_histos or ts2histos output and examples
# str(ts_list) ## list of depth time series data. compare with read_TS output
#
# plot_data_coverage(lightlocs, type="lightlocs", meta=meta)
# plot_data_coverage(histos, type="tad", meta=meta)
# plot_data_coverage(ts_list, type="ts", meta=meta)
```

plot_DepthTempTS *plot Depth Temperature time series data*

Description

line plot for xyz-time series data with colorized z-variable (e.g. depth-temperature time series data from archival tags).

Usage

```
plot_DepthTempTS(ts_df, y="Depth", z="Temperature",
                 xlim, ylim, zlim, show.colorbar=TRUE,
                 pal="jet", cb.xlab, cb.xlab.line=0,
                 pt.lwd, do_interp=TRUE, Return=FALSE, mars, tz="UTC", ...)

plot_DepthTempTS_resampled(ts_df, y="Depth",z="Temperature", bin_res=10,
                           xlim, ylim, zlim, show.colorbar=TRUE,
                           pal="jet", cb.xlab, cb.xlab.line=0,
                       pt.lwd, do_interp=TRUE, Return=FALSE, mars, tz="UTC", ...)

plot_DepthTempTS_resampled_PDT(ts_df, PDT, y="Depth", z="Temperature",
                               xlim, ylim, zlim, show.colorbar=TRUE,
                               pal="jet", cb.xlab, cb.xlab.line=0,
                     pt.lwd, do_interp=TRUE, Return=FALSE, mars, tz="UTC", ...)
```

Arguments

ts_df, PDT	data.frames holding the time series data to be plotted, including the x-vector 'datetime' (in POSIXct-format and UTC), and the numeric y-vector whose label is defined by y. In case of plot_DepthTempTS_resampled the depth temperature time series data will be interpolated on a daily basis and then resampled for temperature data by the depth records of the original time series data. plot_DepthTempTS_resampled_PDT does the same but uses PDT data for resampling.
y	character label of time series vector to be plotted (by default 'Depth').
z	character label of time series vector to be plotted (by default 'Temperature').
bin_res	specific argument for plot_DepthTempTS_resampled: the depth interval at which temperature records should be binned. (by default 10).
xlim	the x limits (x1, x2) of the plot (by default range(ts_df$datetime)).
ylim	the y limits of the plot (by default range(ts_df[[y]])).
zlim	the y limits of the plot (by default range(ts_df[[z]])).
show.colorbar	weather a colorbar should be plotted for image plots (default is TRUE).
pal	color map to be plotted (default is the 'jet'-colormap of the oceanmap-package. See cmap for available color maps.

`cb.xlab`, `cb.xlab.line`
: character string indicating the label of the colorbar (default is Temperature in degrees) and `cb.xlab.line` its placement line (default is 0).

`pt.lwd`
: size of points and lines.

`do_interp`
: whether z-values shall be interpolated over the covered range of the time series data. The default `TRUE` value will produce a line plot. If set to FALSE only the available data points of the z-variable will be plotted.

`Return`
: whether edited time series data set should be returned (by default `FALSE`).

`mars`
: A numerical vector of the form c(bottom, left, top, right) which gives the number of lines of margin to be specified on the four sides of the plot. The default is c(5,4,4,10).

`tz`
: The time zone in which the data should be illustrated (By default "UTC"). ATTENTION: The required date format of the input data is "UTC" (across all RchivalTag-functions).

`...`
: additional arguments to be passed to plot_TS.

Author(s)

Robert K. Bauer

See Also

plot_DepthTS, plot_TS

Examples

```
### load sample depth and temperature time series data from miniPAT:
ts_file <- system.file("example_files/104659-Series.csv",package="RchivalTag")
ts_df <- read_TS(ts_file)
head(ts_df)
ts_df$Serial <- ts_df$DeployID
# plot_DepthTempTS(ts_df, do_interp = FALSE)
# plot_DepthTempTS(ts_df, do_interp = TRUE)
# plot_DepthTempTS_resampled(ts_df, do_interp = TRUE) # more accurate

# ts_df$Lon <- 5; ts_df$Lat <- 43
# plot_DepthTempTS(ts_df, plot_DayTimePeriods = TRUE, xlim = unique(ts_df$date)[2:3])
# plot_DepthTempTS(ts_df, plot_DayTimePeriods = TRUE, xlim = unique(ts_df$date)[2:3])
# plot_DepthTempTS_resampled(ts_df, plot_DayTimePeriods = TRUE, xlim = unique(ts_df$date)[2:3])
# plot_DepthTempTS_resampled_PDT(ts_df, PDT, plot_DayTimePeriods = TRUE)
```

plot_TS *plot time series data*

Description

plotting functions for time series data (e.g. depth or temperature time series data from archival tags) with user specified xtick intervals.

Usage

```
plot_DepthTS(ts_df, y="Depth", xlim, ylim, xticks_interval,
             ylab=y, xlab, main, main.line=1, plot_info=TRUE,
             ID, ID_label="Serial",
             plot_DayTimePeriods=FALSE, twilight.set="ast",
             cex=1, cex.main=1.2*cex, cex.lab=1*cex,
             cex.axis=.9*cex, cex.axis2=1*cex,
             type="l", las=1, xaxs="i", yaxs="i",
             plot_box=TRUE, bty="l", Return=FALSE, tz="UTC",...)


plot_TS(ts_df, y="Depth", xlim, ylim, xticks_interval,
        ylab=y, xlab, main, main.line=1, plot_info=TRUE,
        ID, ID_label="Serial",
        plot_DayTimePeriods=FALSE, twilight.set="ast",
        cex=1, cex.main=1.2*cex, cex.lab=1*cex,
        cex.axis=.9*cex, cex.axis2=1*cex,
        type="l", las=1, xaxs="i", yaxs="i",
        plot_box=TRUE, bty="l", Return=FALSE, tz="UTC", ...)

empty.plot_TS(xlim, ylim, xticks_interval, ylab="", xlab, main="",
              cex=1, cex.main=1.2*cex, cex.lab=1*cex,
              cex.axis=.9*cex, cex.axis2=1*cex,
              las=1, xaxs="i", yaxs="i", do_xaxis=TRUE, do_yaxis = TRUE,
              plot_box=TRUE, bty="l", tz="UTC", ...)
```

Arguments

ts_df	data.frame holding the time series data to be plotted, including the x-vector 'datetime' (in POSIXct-format and UTC), and the numeric y-vector whose label is defined by y.
y	character label of time series vector to be plotted (by default 'Depth').
xlim	the x limits (x1, x2) of the plot (by default range(ts_df$datetime), but needs to be specified in empty.plot_TS).
ylim	the y limits of the plot (by default range(ts_df[[y]]), but needs to be specified in empty.plot_TS).

`xticks_interval`
: time step of the x-axis ticklabels in (full) hours. By default 3 hours for xlim differences <= 1 day, and 6 hours for differences > 1 day.

`ylab`, `xlab`
: the y- and x-axis labels. By default xlab="Time (UTC)" and ylab = "Depth (m)"

`main`, `main.line`
: main title (by default "Tag ID") for the plot and its line (see mtext for reference).

`plot_info`
: whether the plot title and axes labels should be shown (by default `TRUE`).

`ID`, `ID_label`
: Tag ID and its label (column name; by default "Serial") to be selected (e.g. if input data frame holds tagging data from several tags).

`type`
: what type of plot should be drawn. Possible types are:

  - `"p"` for points,
  - `"l"` for lines (default),
  - `"b"` for both,
  - `"c"` for the lines part alone of "b",
  - `"o"` for both 'overlapped',
  - `"n"` for nothing (similar to `empty.plot_TS`-function call)

`las`
: numeric in {0,1,2,3}; the style of axis labels

  0: always parallel to the axis [*default*],

  1: always horizontal,

  2: always perpendicular to the axis,

  3: always vertical.

`xaxs`, `yaxs`
: The style of axis interval calculation to be used for the x-and y-axes. Possible values are "r" and "i" (default). The styles are generally controlled by the range of data or xlim, if given.

  Style "r" (regular) first extends the data range by 4 percent at each end and then finds an axis with pretty labels that fits within the extended range.

  Style "i" (internal) just finds an axis with pretty labels that fits within the original data range.

`cex`, `cex.main`, `cex.lab`, `cex.axis`, `cex.axis2`
: The standard font size of title, axis labels and tick labels.

`plot_DayTimePeriods`, `twilight.set`
: whether day-time periods ('Night', 'Dawn', 'Day', 'Dusk') should be plotted as shaded areas. In case that plot_DayTimePeriods is set `TRUE`, the limits of each time period are required (columns `sunrise`, `sunset`, `dawn.ast`,/`dawn.naut` and `dawn.ast`/`dawn.naut` in `POSIXct`-format. In case of the twilight events, the additional argument `twilight.set` defines the suffix of the twlight-set to be selected (`"ast"` for astronomical dawn and dusks vs `"naut"` for nautical twilight events). If any of the day-time columns, described above, is missing, it/they will be calculated based on geolocation estimates (required columns `Lon` and `Lat`) through an internal call of function `get_DayTimeLimits`.

`do_xaxis`, `do_yaxis`
: Optional arguments in empty.plot_TS to define whether a x and and y-axis shall be plotted (by default `TRUE`).

`plot_box`, `bty`
: whether a box of box-type `bty` should be plotted (by default `TRUE`. `bty` is one of "o" (the default), "l", "7", "c", "u", or "]" the resulting box resembles the corresponding upper case letter.

`Return`
: whether edited time series data set should be returned (by default `FALSE`).

`tz`
: The time zone in which the data should be illustrated (By default "UTC"). ATTENTION: The required date format of the input data is "UTC" (across all RchivalTag-functions). Run `OlsonNames(tzdir = NULL)` for valid time zone definitions.

`...`
: additional arguments to be passed to plot.

Author(s)

Robert K. Bauer

See Also

ggboxplot_DepthTS_by_hour, dy_DepthTS, plot_DepthTempTS

Examples

```
### load sample depth and temperature time series data from miniPAT:
ts_file <- system.file("example_files/104659-Series.csv",package="RchivalTag")
ts_df <- read_TS(ts_file)
ts_df$Serial <- ts_df$DeployID
head(ts_df)

## load same data in LOTEK format
ts_file <- system.file("example_files/104659_PSAT_Dive_Log.csv",package="RchivalTag")
ts_df <- read_TS(ts_file,date_format="%m/%d/%Y %H:%M:%S")
head(ts_df) ## attention no identifier (Ptt, Serial, DeployID) included!
ts_df$DeployID <- ts_df$Ptt <- "104659"
ts_df$Serial <- "Tag1"

### select subsets (dates to plot)
# plot_DepthTS(ts_df, plot_DayTimePeriods = FALSE, xlim = unique(ts_df$date)[2:3])
# xlim <- c("2016-08-10 6:10:00", "2016-08-11 17:40:00")
# plot_DepthTS(ts_df, plot_DayTimePeriods = FALSE, xlim = xlim)

### check xtick time step:
# plot_DepthTS(ts_df, plot_DayTimePeriods = FALSE, xlim = "2016-08-10")
# plot_DepthTS(ts_df, plot_DayTimePeriods = FALSE, xlim = "2016-08-10", xticks_interval = 2)


### add daytime periods during plot-function call and return extended data set
# ts_df$Lon <- 5; ts_df$Lat <- 43
# plot_DepthTS(ts_df, plot_DayTimePeriods = TRUE, xlim = unique(ts_df$date)[2:3])
# ts_df2 <- plot_DepthTS(ts_df, plot_DayTimePeriods = TRUE, Return = TRUE)
```

```
# names(ts_df)
# names(ts_df2)

### add daytime periods before function call
# ts_df_extended <- get_DayTimeLimits(ts_df)
# plot_DepthTS(ts_df_extended, plot_DayTimePeriods = TRUE)
# plot_DepthTS(ts_df_extended, plot_DayTimePeriods = TRUE, twilight.set = "naut")

### introduce data transmission gaps that are then filled internally
### as well as daytime periods based on interpolated Lon & Lat positions
# ts_df_cutted <- ts_df[-c(200:400, 1800:2200), ]
# plot_DepthTS(ts_df_cutted, plot_DayTimePeriods = FALSE)
# plot_DepthTS(ts_df_cutted, plot_DayTimePeriods = TRUE)

### example for empty.plotTS and adding time series data as line:
# empty.plot_TS(xlim="2016-08-10",ylim=c(100,0))
# lines(ts_df$datetime, ts_df$Depth)

### alternative:
# plot_DepthTS(ts_df, xlim=c("2016-08-10","2016-08-12"), plot_DayTimePeriods = TRUE, type='n')
# lines(ts_df$datetime, ts_df$Depth)
```

RchivalTag *RchivalTag - Analyzing and Interactive Visualization of Archival Tagging Data*

Description

RchivalTag provides a set of functions to analyze and visualize different data products from Archival Tags (Supported Models include amongst others: MiniPAT, sPAT, mk10, mk9 from Wildlife Computers as well as LOTEK PSAT Models LOTEK. Models from other Manufactorers might be supported as well.

- "(Depth) time series data" (See plot_TS, plot_DepthTS, dy_DepthTS, empty.plot_TS)
- "Time-at-Depth (TaD) and Time-at-Temperature (TaT) fequencies" (See ts2histos, merge_histos, hist_tad & hist_tat)
- "Depth Temperature profiles (time series data)" (See plot_DepthTempTS, plot_DepthTempTS_resampled, plot_DepthTempTS_resampled_PDT, interpolate_TempDepthProfiles, get_thermalstrat & image_TempDepthProfiles)
- "PDT (PAT-style Depth Temperature profiles) data" (See read_PDT, interpolate_TempDepthProfiles, get_thermalstrat & image_TempDepthProfiles)
- "visualization of geolocation estimates" (See: ggplot_geopos, ggplotly_geopos)

Details

TaD-/TaT-histogram data
- The package allows to read and calculate standard summary data products (TaD-/TaT-profiles, see above) from recovered or transmitted time series data sets as well as to merge and visualize such summary data products from different tag setups/tagging programs. For more information on these data products, please see: Wildlife Computers (2016).

Depth time series data
- data visualization, optionally highlighting daytime differences (dawn, day, dusk, night).

Depth-temperature time series data
- data visualization and examination of the thermal stratification of the water column (i.e. thermocline depth, gradient and stratification index), based on previously interpolated. The paper by Bauer et al. (2015) is highly recommended in this context.

Depth-temperature time series data
- data visualization and examination of the thermal stratification of the water column (i.e. thermocline depth, gradient and stratification index), based on previously interpolated. The paper by Bauer et al. (2015) is highly recommended in this context.

interactive geolocation vizualiation
- data visualization via ggplot2 and plot_ly based on oceanmap standard maps.

Compatibility
So far, the package is mainly adapted for archival tagging data from Wildlife Computers, but can also be applied to data from other tag manufacturers (e.g. see ts2histos in order to calculate TaD & TaT-frequencies from time series data). Function examples are based on the transmitted data sets of a miniPAT-tag from the BLUEMED-project, funded by the French National Research Agency (ANR; `https://anr.fr`).

Author(s)

Robert K. Bauer

References

Bauer, R., F. Forget and JM. Fromentin (2015) Optimizing PAT data transmission: assessing the accuracy of temperature summary data to estimate environmental conditions. Fisheries Oceanography, 24(6): 533-539, doi:10.1111/fog.12127

Bauer, R., JM. Fromentin, H. Demarcq and S. Bonhommeau (2017) Habitat use, vertical and horizontal behaviour of Atlantic bluefin tuna (Thunnus thynnus) in the Northwestern Mediterranean Sea in relation to oceanographic conditions. Deep-Sea Research Part II: Topical Studies in Oceanography, 141: 248-261, doi:10.1016/j.dsr2.2017.04.006

Bauer, R., F. Forget, JM. Fromentin and M. Capello (2020) Surfacing and vertical behaviour of Atlantic bluefin tuna (*Thunnus thynnus*) in the Mediterranean Sea: implications for aerial surveys. ICES Journal of Marine Science, 77(5): 1979-1991, doi:10.1093/icesjms/fsaa083

Wildlife Computers (2016) MiniPAT-User-Guide, 4 April 2016, 26 pp. `https://static.wildlifecomputers.com/MiniPAT-User-Guide1.pdf`

read_histos *reads a TAD/TAT-histogram file from archival tags*

Description

reads or posttreats a manually loaded standard histogram data file, containing Time-at-Depth (TAD) and Time-at-Temperature (TAT) frequency data, from archival tags by Wildlife Computers.

Usage

```
read_histos(hist_file, date_format, lang_format="en", tz="UTC", dep.end, Serial,
            force_24h=TRUE, min_perc, omit_negatives=TRUE, right_truncate=TRUE)
```

Arguments

hist_file	character string indicating the name of a standard Wildlife Computers file to read or the data.frame of a manually loaded histogram data file. The combination of the columns DeployID, Ptt and Serial is assumed to provide an unique key to distinguish data from indiviudal tags.
force_24h	whether histogram data with a time step of less than 24h should be merged to 24h (default is TRUE). Note that the current version of hist_tad and hist_tat was written for 24h data!
date_format, lang_format, tz	character strings indicating the date format, language format and the corresponding time zone, defined by the vectors Date and Time (by default: date_format="%H:%M:%S %d-%b-%Y", lang_format="en", tz='UTC') If formatting fails, please check as well the input language format, defined by lang_format (and use abbrviations such as "en" for English,"es" for Spanish, "fr" for French, etc.) as well.
dep.end	Date specifying the deployment end of the tag.
Serial	character-string indicating the Serial number of the tag to be selected. (in case of multi-tag histogram files.)
min_perc	optional number, defining the minimum data coverage (in percent) of histogram entries obtained from depth time series data.
omit_negatives	merge negative depth and temperature bins with next positive bin (>= 0; default is TRUE).
right_truncate	truncate the values of the last tat- and tad-bin to 45 degrees and 2000 m, respectively (default is TRUE).

Details

This function reads or posttreats a manually loaded standard Wildlife Computers histogram file including Time-at-Depth (TAD) and Time-at-Temperature (TAT) frequency data. In the post-treatment, the histogram data is split in lists of TAD and TAT per individual (see below). Thus processed data from several histogram files (or similarly processed time series data) can be combined using the function combine_histos. Merging of histogram data from several tags, based on similar or user-specified TAD and TAT-bin_breaks, can be done by applying function merge_histos. To generate TAD/TAT histogram data from depth and temperature time series data, see ts2histos.

Value

A list-of-lists containing the loaded histogram data. Lists of TAD and TAT data are distinguished at the first nesting level. Further sublists include the `bin_breaks` and data.frames of the histogram data per tag (ID). Tag IDs are constructed based on the columns `DeployID`, `Ptt` and `Serial` keys (e.g. DeployID.101_Ptt.102525). **The data.frames of the histogram data also contain average (avg) and standard deviation (SD) of depth and temperature values that are estimated internally from the TAD and TAT data sets (not measured!). The accuracy of these estimates thus depends on the number and selection of bin breaks**, unlike ts2histos-generated values that are directly estimated from time series data. See statistics-example below.

$ TAD:List
..$ ID1 : List of 2
.. ..$ bin_breaks: num
.. ..$ df : data.frame
.. .. ..$ DeployID
.. .. ..$ Ptt
.. .. ..$ datetime
.. .. ..$ date
.. .. ..$ Bin1
.. ..$ Bin? (up to number of bin breaks)
.. .. ..$ avg (average depth estimated!! from histogram data)
.. .. ..$ SD (average depth estimated!! from histogram data)

$ TAT:List
..$ ID1 : List of 2
.. ..$ bin_breaks: num
.. ..$ df : data.frame (with columns as above)
..$ ID2 : List of 2
...

Author(s)

Robert K. Bauer

See Also

ts2histos, combine_histos, merge_histos, hist_tad, hist_tat

Examples

```
## read and merge 12h histogram data:
# 12h_hist_file <- system.file("example_files/67851-12h-Histos.csv",package="RchivalTag")
# hist_dat_0 <- read_histos(12h_hist_file,min_perc=100) # omit incomplete days
# hist_tad(hist_dat_0)
#hist_tat(hist_dat_0)

## example 1) read, merge and plot TAD frequency data from several files:
## part I - read histogram data from two files:
hist_dat_1 <- read_histos(system.file("example_files/104659-Histos.csv",package="RchivalTag"))
hist_dat_2 <- read_histos(system.file("example_files/104659b-Histos.csv",package="RchivalTag"))
```

```
## note the second list is based on the same data (tag), but on different bin_breaks

## part II - combine TAD/TAT frecuency data from seperate files in one list:
hist_dat_combined <- combine_histos(hist_dat_1, hist_dat_2)
par(mfrow=c(2,1))
hist_tad(hist_dat_combined)
hist_tat(hist_dat_combined)

## part III - force merge TAD/TAT frecuency data from seperate files
# in one list, by applying common bin_breaks:
hist_dat_merged <- merge_histos(hist_dat_combined,force_merge = TRUE)
hist_tad(hist_dat_merged)
hist_tat(hist_dat_merged)

## part IV - plot merged data:
hist_tad(hist_dat_merged) # of all tags
unique(hist_dat_merged$TAD$merged$df$DeployID) ## list unique tags in merged list
hist_tad(hist_dat_merged, select_id = "15P1019b", select_from = 'DeployID') # of one tag

## part V - unmerge data:
unmerge_histos(hist_dat_merged)

## part VI - statistics:
# get histogram data with histogram-derived average depth and temperature values
hist_dat_1 <- read_histos(system.file("example_files/104659-Histos.csv",package="RchivalTag"))
avg1 <- hist_dat_1$TAD$DeployID.15P1019_Ptt.104659$df$avg # infered from the histogram data

# generate histogram data and average/sd-estimates from depth time series data of the same tag.
# attention! unlike for histogram files, the average/sd-estimates are calculated
# directly from depth time series data and not from the binned histogram data
ts_file <- system.file("example_files/104659-Series.csv",package="RchivalTag")
ts_df <- read_TS(ts_file)

tad_breaks <- c(0, 2, 5, 10, 20, 50, 100, 200, 300, 400, 600, 2000)
hist_dat_2 <- ts2histos(ts_df, tad_breaks = tad_breaks)
avg2 <- hist_dat_2$TAD$merged$df$avg # directly estimated from the depth time series data

# check accuracy of average depth values:
plot(avg1, avg2)
avg1-avg2
abline(0,b = 1,lty="dotted")

## crosscheck!
# library(plyr)
# ts_stats <- ddply(ts_df,c("date"),function(x) c(avg=mean(x$Depth,na.rm=T),SD=sd(x$Depth,na.rm=T)))
# avg2==ts_stats$avg

# path <- system.file("example_files",package="RchivalTag")
# PDT <- read_PDT("104659-PDTs.csv",folder=path)
# head(PDT)
# image_TempDepthProfiles(interpolate_PDTs(PDT)[[1]])

## add information
```

```
# lines(ts_stats$date+.5,ts_stats$avg)
# add <- hist_dat_2$TAD$merged$df
# lines(add$date+.5,add$avg)
# axis(2,at=50,las=1)
# abline(h=20,lty="dashed",col="violet",lwd=3)
```

`read_PDT` *read PDT data from archival one or multiple tags*

Description

reads PDT data (PAT-style Depth Temperature profiles) from archival tags by Wildlife Computers). The PDT file can contain data from one or multiple tags.

What are PDTs?
PDT data provides minimum and maximum water temperatures during a user-programmed interval (usually 24h) at 8 to 16 depths. The sampled depths are thereby rounded (binned) to multiples of 8 and include the minimum and maximum depth bins as well as the 6 to 14 most frequent depth bins at which the tagged animal was located. The total number of depth bins (8 or 16) also depends on the tagged animals' behaviour. If the animal was in waters deeper than 400 m during the summary data period, the range of temperature at 16 depth bins will be reported, otherwise 8.

Why using PDT data?
Despite its low resolution, PDT data can give accurate information on the in-situ thermal stratification of the water column (e.g. thermocline depth, stratification index, ocean heat content) experienced by the tagged animal, as illustrated by Bauer et al. (2015). Accordingly, PDT data can provide precious insights into the relations between animal behaviour and environmental conditions. See the example section below on how to obtain thermal stratification indicators of the water column from PDT data.

For instance, daily PDT data can be interpolated and then visualized using functions interpolate_PDTs and image_TempDepthProfiles, respectively. This faciliates the analysis of temporal changes of temperature profiles, for instance, in relation to animal behaviour (e.g. diving behaviour).

Usage

```
read_PDT(pdt_file, folder, sep=",",date_format,lang_format="en",tz="UTC")
```

Arguments

`pdt_file`	character string indicating the name of a standard PDT-file. The Date-vector of the file is expected to be or the format "%H:%M:%S %d-%b-%Y, tz='UTC'".
`folder`	path to pdt-file.
`sep`	the field separator character. Values on each line of the file are separated by this character (default is ',').

`date_format`, `lang_format`, `tz`
: character strings indicating the date format, language format and the corresponding time zone, defined by the vectors Date and Time (by default: date_format="%H:%M:%S %d-%b-%Y", lang_format="en", tz='UTC') If formatting fails, please check as well the input language format, defined by `lang_format` (and use abbrviations such as "en" for English,"es" for Spanish, "fr" for French, etc.) as well.

Value

A data.frame with the columns:
"pdt_file", "DeployID", "Ptt", "NumBins", "Depth", "MinTemp", "MaxTemp", "datetime", "date", "MeanPDT"
Attention: Column "MeanPDT" is not measured but calculated as the average of "MinTemp" and "MaxTemp" values.

Author(s)

Robert K. Bauer

References

Bauer, R., F. Forget and JM. Fromentin (2015) Optimizing PAT data transmission: assessing the accuracy of temperature summary data to estimate environmental conditions. Fisheries Oceanography, 24(6): 533-539, doi:10.1111/fog.12127

See Also

bin_TempTS, interpolate_PDTs, image_TempDepthProfiles

Examples

```
## step I) read sample PDT data file:
path <- system.file("example_files",package="RchivalTag")
PDT <- read_PDT("104659-PDTs.csv",folder=path)
head(PDT)

## step II) interpolate average temperature fields (MeanPDT) from PDT file:
# m <- interpolate_PDTs(PDT)
# str(m)
# m$sm

## step III) calculate thermal stratifcation indicators per day (and tag):
# strat <- get_thermalstrat(m, all_info = TRUE)
# strat <- get_thermalstrat(m, all_info = FALSE)

## step IV) plot interpolated profiles:
# image_TempDepthProfiles(m$station.1)
```

`read_TS` *reads Time Series Data from Archival Tags*

Description

reads Time Series Data (e.g. Depth and Temperature) from Archival Tags (Supported Models: MiniPAT, sPAT, recovered mk10, mk9 from Wildlife Computers as well as LOTEK PSAT Models LOTEK. Models from other Manufactorers might be supported as well.

Usage

```
read_TS(ts_file, header=TRUE, sep=",", skip = 0,
        date_format, lang_format = "en", tz = "UTC")
```

Arguments

`ts_file`	character string indicating the name of a standard Wildlife Computers file to read or the data.frame of a manually loaded histogram data file. The file is assumed to include the columns `Day`, `Time` (or a preformatted date-time vector termed `datetime` in "UTC" format.) as well as at one of the subsequent columns `DeployID`, `Ptt` and `Serial` to distinguish data from indiviudal tags.
`header`	a logical value indicating whether the file contains the names of the variables as its first line. If missing, the value is determined from the file format: header is set to TRUE if and only if the first row contains one fewer field than the number of columns.
`sep`	the field separator character. Values on each line of the file are separated by this character. If sep = "" the separator is 'white space', that is one or more spaces, tabs, newlines or carriage returns.
`skip`	integer: the number of lines of the data file to skip before beginning to read data.
`date_format, lang_format, tz`	character strings indicating the date format, language format and the corresponding time zone, defined by the vectors Date and Time (by default: date_format="%d-%b-%Y %H:%M:%S", lang_format="en", tz='UTC') If formatting fails, please check as well the input language format, defined by `lang_format` (and use abrviations such as "en" for English,"es" for Spanish, "fr" for French, etc.) as well.

Details

This function reads a time series data file from archival tags. Data sets are "completed" to facilitate an assessment of the data coverage (i.e. by ts2histos or hist_tad).

Value

A data frame (data.frame) containing a representation of the data in the file.

Author(s)

Robert K. Bauer

See Also

ts2histos, hist_tad, plot_TS

Examples

```
### load sample depth and temperature time series data from miniPAT:
ts_file <- system.file("example_files/104659-Series.csv",package="RchivalTag")
ts_df <- read_TS(ts_file)
head(ts_df)

## other date_format:
ts_file2 <- system.file("example_files/104659-Series_date_format2.csv",package="RchivalTag")
# ts_miniPAT2 <- read_TS(ts_file2) # run to see error message
ts_miniPAT2 <- read_TS(ts_file2,date_format = "%d-%m-%Y %H:%M:%S")
head(ts_miniPAT2)

## other date_format and lang_format:
ts_file_ES <- system.file("example_files/104659-Series_date_format_ES.csv",package="RchivalTag")
# ts_miniPAT_ES <- read_TS(ts_file_ES) # run to see error message
ts_miniPAT_ES <- read_TS(ts_file_ES,skip=1,sep=";",header = TRUE,
                         date_format = "%d/%b/%y %H:%M:%S",lang_format = "es")
head(ts_miniPAT_ES)


## load same data in LOTEK format
ts_file <- system.file("example_files/104659_PSAT_Dive_Log.csv",package="RchivalTag")
ts_df <- read_TS(ts_file,date_format="%m/%d/%Y %H:%M:%S")
head(ts_df) ## attention no identifier (Ptt, Serial, DeployID) included!
ts_df$DeployID <- ts_df$Ptt <- "104659"

## example 1) convert only DepthTS data to daily TaD frequencies:
tad_breaks <- c(0, 2, 5, 10, 20, 50, 100, 200, 300, 400, 600, 2000)
tat_breaks <- c(10,12,15,17,18,19,20,21,22,23,24,27)

histos <- ts2histos(ts_df, tad_breaks = tad_breaks, tat_breaks = tat_breaks)
histos$TAD$merged$df$nperc ## check completeness of TAD data sets
histos$TAT$merged$df$nperc ## check completeness of TAT data sets
# histos <- ts2histos(ts_df, tad_breaks = tad_breaks, tat_breaks = tat_breaks,min_perc = 90)


### example 2) add daytime (Day vs Night) information and plot results
# add daytime periods during plot-function call and return extended data set
# ts_df$Lon <- 5; ts_df$Lat <- 43
# plot_DepthTS(ts_df, plot_DayTimePeriods = TRUE, xlim = unique(ts_df$date)[2:3])
# ts_df2 <- plot_DepthTS(ts_df, plot_DayTimePeriods = TRUE, Return = TRUE)
# names(ts_df)
# names(ts_df2)
```

```
### add daytime periods before function call
# ts_df_extended <- get_DayTimeLimits(ts_df)
# plot_DepthTS(ts_df_extended, plot_DayTimePeriods = TRUE)
# plot_DepthTS(ts_df_extended, plot_DayTimePeriods = TRUE, twilight.set = "naut")
```

`resample_PDT` *resample temperature at depth data from interpolated daily temperature at depth profiles or time series data*

Description

interpolates depth-temperature data from a provided source (depth-temperature time series data or PDT data) and resamples the interpolated data by the depth time series data provided, to faciliate plot_DepthTempTS-plots even for tags with no temperature time series data or to improve interpolation results of the plot_DepthTempTS-plots from low-resolution depth-temperature time series data.

Usage

```
resample_PDT(ts_df, PDT, ...)
resample_DepthTempTS(ts_df, ...)
```

Arguments

`ts_df`	`ts_df` is a data.frame with depth-temperature time series data data or only depth time series data. Required columns are `Depth` for the depth data and a column containing temperature data, whose name is defined by `Temp_field`, by default Temperature.
`PDT`	an optional data.frame containing PDT-data from read_PDT.
`...`	additional arguments to be passed to interpolate_TempDepthProfiles, or interpolate_PDTs.

Value

a data.frame with depth-temperature time series data.

Author(s)

Robert K. Bauer

References

Bauer, R., F. Forget and JM. Fromentin (2015) Optimizing PAT data transmission: assessing the accuracy of temperature summary data to estimate environmental conditions. Fisheries Oceanography, 24(6): 533-539, doi:10.1111/fog.12127

See Also

read_PDT, interpolate_TempDepthProfiles, get_thermalstrat, image_TempDepthProfiles

Examples

```
## read in depth temperature time series data (sampling rate 5min)
ts_file <- system.file("example_files/104659-Series.csv",package="RchivalTag")
ts_df <- read_TS(ts_file)
head(ts_df)

## run daily interpolation of depth temperature time series data
m <- interpolate_TempDepthProfiles(ts_df)
image_TempDepthProfiles(m$station.1)
ts_df2 <- resample_DepthTempTS(ts_df) ## reassign temperature at depth values

## read PDT data from same tag
## (= low resolution depth temperature data (8 Depth and Temperature records per day))
path <- system.file("example_files",package="RchivalTag")
PDT <- read_PDT("104659-PDTs.csv",folder=path)
head(PDT)

m <- interpolate_PDTs(PDT) ## interpolate PDTs
image_TempDepthProfiles(m$station.1)
ts_df3 <- resample_PDT(ts_df, PDT) ## reassign temperature at depth values



#### plot results:
## 1) dot plots:

## dot plot of RECORDED depth temperature time series data
## plot_DepthTempTS(ts_df, do_interp = FALSE)

## dot plot of RESAMPLED depth temperature time series data
## from previously daily interpolated depth temperature time series data
# plot_DepthTempTS(ts_df2, do_interp = FALSE)

## dot plot of RESAMPLED depth temperature time series data
## from daily interpolated PDT data (external resampling)
# plot_DepthTempTS(ts_df3, do_interp = FALSE)

## dot plot of RESAMPLED depth temperature time series data
## from daily interpolated PDT data (internal resampling)
# plot_DepthTempTS_resampled_PDT(ts_df, PDT, do_interp = FALSE)


## 2) line plots:

## line plot of depth temperature time series data
## (internal interpolation between neighboring temperature records)
## not recommended for low resolution time series data
# plot_DepthTempTS(ts_df, do_interp = TRUE)
```

```
## line plot of depth temperature time series data
## (based on internal daily interpolated depth temperature time series data)
# plot_DepthTempTS_resampled(ts_df, do_interp = TRUE)

## line plot of depth temperature time series data
## from daily interpolated PDT data (external resampling)
# plot_DepthTempTS(ts_df3, do_interp = TRUE)

## line plot of depth temperature time series data
## from daily interpolated PDT data (internal resampling)
# plot_DepthTempTS_resampled_PDT(ts_df, PDT, do_interp = TRUE)
```

`resample_TS` *resample time series data at a lower resolution*

Description

resample time series data at a lower resolution

Usage

```
resample_TS(df, tstep, nsims)
```

Arguments

`df`	data.frame holding the time series data to be resampled, including a 'datetime'-vector (in `POSIXct`-format and UTC).
`tstep`	numeric vector indicating the resampling resolution (in seconds).
`nsims`	number of simulated datasets to be generated. If missing, the maximum number of datasets will be returned, based on a moving window of the temporal resolution of the input dataset.

Author(s)

Robert K. Bauer

See Also

plot_TS

Examples

```
### load sample depth and temperature time series data from miniPAT:
ts_file <- system.file("example_files/104659-Series.csv",package="RchivalTag")
ts_df <- read.table(ts_file, header = TRUE, sep = ",")
head(ts_df)
ts_df$datetime <- as.POSIXct(strptime(paste(ts_df$Day, ts_df$Time),
                              "%d-%b-%Y %H:%M:%S",tz = "UTC"))

tsims <- resample_TS(ts_df,600)
length(tsims)
```

---

simulate_DepthTS *simulate depth time series data*

Description

function to simulate depth series data from a template data set.

Usage

```
simulate_DepthTS(ts_df, ndays=10, gaps=TRUE, trate=90,
             random_Depth=TRUE, ref_Depth_lim=300, random_Depth_lim=c(100, 700))
```

Arguments

ts_df	data.frame holding the template depth time series data.
ndays	number of days of depth time series data that should be simulated).
gaps, trate	Whether gaps should be introduced into the dataset according to the transmission rate (trate), in addition to any potential gaps in the template data set.
random_Depth, ref_Depth_lim, random_Depth_lim	Whether depth records from the template data set should be additonaly randomized. In this case, depth records >= ref_Depth_lim will be replaced by values between the limits of random_Depth_lim (i.e. in the default settings a record of >= 300 will be replaced by a value between 100 and 700 m).

Author(s)

Robert K. Bauer

See Also

dy_DepthTS, plot_DepthTS, plot_data_coverage

Examples

```
### load sample depth and temperature time series data from miniPAT:
# ts_file <- system.file("example_files/104659-Series.csv",package="RchivalTag")
# ts_df <- read_TS(ts_file)
# ts_df$Serial <- ts_df$DeployID
# head(ts_df)
# dy_DepthTS(ts_df) # plot original data
#
# ts_df_sim <- simulate_DepthTS(ts_df) # simulate data
#
# dy_DepthTS(ts_df_sim) # plot simulated data
#
# library(dplyr)
# meta <- rbind(ts_df[,names(ts_df_sim)],ts_df_sim) %>%
#   group_by(DeployID, Serial, Ptt) %>%
#   summarise(dep.date=min(date),pop.date=max(date)) %>%
#   as.data.frame()
#
# ts_list <- list(ts_df,ts_df_sim)
# names(ts_list) <- meta$Serial
#
# # plot data coverage
# plot_data_coverage(x = ts_list,type="ts", meta = meta)
```

`ts2histos` *convert depth and temperature time series data to discrete Time-at-Depth and Time-at-Temperature data (histogram data)*

Description

convert depth and temperature time series data to discrete Time-at-Depth (TaD) and Time-at-Temperature (TaT) data (histogram data) at user-defined breakpoints

Usage

```
ts2histos(ts_df, tad_breaks=NULL, tat_breaks=NULL, split_by=NULL,
          aggregate_by="Ptt",min_perc, omit_negatives=TRUE)
```

Arguments

`ts_df`	dataframe of depth time series data. Obligatory columns are the numeric vector "Depth", "date" (of class Date) and "Serial". `split.by` defines an optional vector to consider (e.g. `day.period`).
`tad_breaks, tat_breaks`	a numeric vector, defining the depth and/or temperature breakpoints of the histogram cells.
`split_by`	Name of the column with logical entries by which TaD/TaT data shall be splitted (e.g. daytime; see classify_DayTime.).

aggregate_by	character vector defining the columns by which the tagging data should be aggregated. Should contain columns that identify tags (e.g. Serial, Ptt, DeployID) the date and/or day time period (to seperate records from night, day, dawn and dusk see classify_DayTime). Default values are: date, Day and Ptt.
min_perc	optional number, defining the minimum data coverage (in percent) of histogram entries obtained from depth time series data.
omit_negatives	treat negative depth and temperature records as 0 (default is TRUE).

Details

Time-at-Depth and Time-at-Temperature fequencies (histograms) are a standard data product of archival tags (incl. tag models TDR-Mk9, PAT-Mk10 and miniPAT by Wildlife Computers) that allow to assess habitat preferences of tagged animals. It can be likewise generated from transmitted or recovered time series data sets, which is the purpose of this function.

However, different depth and temperature bin breaks are often used during different deployment programs, which makes a later comparitive analysis of TaT and TaD data difficult. For such cases, the functions combine_histos and merge_histos can be applied to merge TaT and TaD frequencies based on common bin breaks of different tags.

To visualize Time-at-Temperature (TaT) and Time-at-Depth (TaD) data, please see hist_tat and hist_tad, respectively.

Value

A list-of-lists containing the loaded histogram data. Lists of TaD and TaT data are distinguished at the first nesting level. Further sublists include the bin_breaks and data.frames of the generated histogram data. **The data.frames of the histogram data thereby also contain average (avg) and standard deviation (SD) of depth and temperature values that are likewise directly estimated from time series data**, unlike read_histos-generated values that are estimated from the histogram data. The accuracy of latter estimates thus depends on the number and selection of bin breaks (see statistics-example in read_histos).

$ TaD:List
..$ merged : List of 2
.. ..$ bin_breaks: num
.. ..$ df : data.frame
.. .. ..$ DeployID
.. .. ..$ Ptt
.. .. ..$ datetime
.. .. ..$ date
.. .. ..$ Bin1
.. ..$ Bin? (up to number of bin breaks)
.. .. ..$ avg (average depth estimated!! from histogram data)
.. .. ..$ SD (average depth estimated!! from histogram data)

$ TaT:List
..$ merged : List of 2
.. ..$ bin_breaks: num
.. ..$ df : data.frame (with columns as above)

Author(s)

Robert K. Bauer

See Also

read_histos, hist_tad, merge_histos

Examples

```
### load sample depth and temperature time series data from miniPAT:
ts_file <- system.file("example_files/104659-Series.csv",package="RchivalTag")
ts_df <- read_TS(ts_file)
head(ts_df)

tad_breaks <- c(0, 2, 5, 10, 20, 50, 100, 200, 300, 400, 600, 2000)
tat_breaks <- c(10,12,15,17,18,19,20,21,22,23,24,27)


## example 1a) convert only DepthTS data to daily TaD frequencies:
ts2histos(ts_df, tad_breaks = tad_breaks)
# hist_tad(ts_df, bin_breaks = tad_breaks)
hist_tad(ts_df, bin_breaks = tad_breaks, do_mid.ticks = FALSE)

## convert 1b) only TemperatureTS data to daily TaT frequencies:
tat <- ts2histos(ts_df, tat_breaks = tat_breaks)
hist_tat(ts_df, bin_breaks = tat_breaks, do_mid.ticks = FALSE)
hist_tat(tat$TAT$merged, do_mid.ticks = FALSE)

## convert 1c) DepthTS & TemperatureTS data to daily TaD & TaT frequencies:
histos <- ts2histos(ts_df, tad_breaks = tad_breaks, tat_breaks = tat_breaks)
histos$TAD$merged$df$nperc ## check completeness of TAD data sets
histos$TAT$merged$df$nperc ## check completeness of TAT data sets
# histos <- ts2histos(ts_df, tad_breaks = tad_breaks, tat_breaks = tat_breaks,min_perc = 90)

## convert 1d) back-to-back histogram showing day vs night TaD frequencies:
ts_df$Lat <- 4; ts_df$Lon=42.5 ## required geolocations to estimate daytime
head(ts_df)
ts_df2 <- classify_DayTime(get_DayTimeLimits(ts_df)) # estimate daytime
head(ts_df2)

ts2histos(ts_df2, tad_breaks = tad_breaks,split_by = "daytime")
hist_tad(ts_df2, bin_breaks = tad_breaks,split_by = "daytime", do_mid.ticks = FALSE)


## example 2) rebin daily TaD frequencies:
tad <- ts2histos(ts_df, tad_breaks = tad_breaks)
tad2 <- rebin_histos(hist_list = tad, tad_breaks = tad_breaks[c(1:3,6:12)])
par(mfrow=c(2,2))
hist_tad(tad, do_mid.ticks = FALSE) ## example for multiple individuals
hist_tad(tad$TAD$merged, do_mid.ticks = FALSE)
hist_tad(tad$TAD$merged, bin_breaks = tad_breaks[c(1:3,6:12)]) ## from inside hist_tad
```

`unmerge_histos` *unmerge previously grouped or merged lists of TAD/TAT frequency data*

Description

This function unmerges previously grouped or merged lists of TAD/TAT frequency data, and thus allows to add TAD/TAT lists from new tags (see combine_histos).

Usage

```
unmerge_histos(hist_list)
```

Arguments

`hist_list`	A previously grouped or merged list-of-lists to be unmerged (seperated by tags).

Value

A list-of-lists of ungrouped/unmerged TAD and TAT frequency data.

$ TAD:List
..$ ID1 : List of 2
.. ..$ bin_breaks: num
.. ..$ df : data.frame
$ TAT:List
..$ ID1 : List of 2
.. ..$ bin_breaks: num
.. ..$ df : data.frame
..$ ID2 : List of 2
...

Author(s)

Robert K. Bauer

See Also

combine_histos, merge_histos, hist_tad

Examples

```
## example 1) read, merge and plot TAD frequency data from several files:
## part I - read histogram data from two files:
hist_dat_1 <- read_histos(system.file("example_files/104659-Histos.csv",package="RchivalTag"))
hist_dat_2 <- read_histos(system.file("example_files/104659b-Histos.csv",package="RchivalTag"))
## note the second list is based on the same data (tag), but on different bin_breaks

## part II - combine TAD/TAT frecuency data from seperate files in one list:
hist_dat_combined <- combine_histos(hist_dat_1, hist_dat_2)
par(mfrow=c(2,1))
hist_tad(hist_dat_combined)
hist_tat(hist_dat_combined)

## part III - force merge TAD/TAT frecuency data from seperate files
# in one list, by applying common bin_breaks:
hist_dat_merged <- merge_histos(hist_dat_combined,force_merge = TRUE)
hist_tad(hist_dat_merged)
hist_tat(hist_dat_merged)

## part IV - plot merged data:
hist_tad(hist_dat_merged) # of all tags
unique(hist_dat_merged$TAD$merged$df$DeployID) ## list unique tags in merged list
hist_tad(hist_dat_merged, select_id = "15P1019b", select_from = 'DeployID') # of one tag

## part V - unmerge data:
unmerge_histos(hist_dat_merged)
```

---

update_leaflet_elementId

update elementID of the leaflet map to avoid rendering issues in RMarkdown

---

Description

This function updates the elementID of the leaflet map to avoid rendering issues when plotting the same map twice in RMarkdown.

Usage

```
update_leaflet_elementId(map)
```

Arguments

map	a leaflet map

Value

the same leaflet map with an updated elementIDs.

Author(s)

Robert K. Bauer

See Also

leaflet_geopos

Examples

```
## only valid in RMarkdown chunks:
# kmz_file2 <- system.file("example_files/15P0986-15P0986-2-GPE3.kmz",package="RchivalTag")
# k2 <- get_geopos(kmz_file2)
# k0 <- k3 <- rbind(k1,k2)
#
# library(leaflet)
# map <- leaflet_geopos(k0, ID_label="DeployID", collapsedLayers = F)
# map
# map # plot again to show rendering issues (in the layer menu title)
## this is required to avoid rendering issues when plotting the same map twice via RMarkdown
# map <- update_leaflet_elementId(map)
#
# plot again with updated elementID:
# map %>% addMiniMap()
```

Index

abacus_plot *(*plot_data_coverage*)*, 34
add.region, *14*

bin_TempTS, 2, *27*, *29*, *48*

classify_DayTime, 4, *9*, *19*, *23*, *55, 56*
cmap, *17*, *26*, *31*, *37*
combine_histos, 5, *20, 21*, *24*, *32, 33*, *44, 45*, *56*, *58*
crepuscule, *4*, *8, 9*

data.frame, *3, 4*, *7*, *9, 10*, *16*, *28*, *30, 31*, *37*, *39*, *44, 45*, *48, 49*, *51*, *53, 54*, *56*
Date, *44*, *55*
delete.region, *14*
dy_DepthTS, 6, *13*, *41, 42*, *54*
dy_TS *(*dy_DepthTS*)*, 6
dygraph, *7, 8*
dyOptions, *7, 8*

empty.plot_TS, *42*
empty.plot_TS *(*plot_TS*)*, 39
empty.resample_TS *(*resample_TS*)*, 53

get_DayTimeLimits, *4*, *7*, 8
get_geopos *(*ggplot_geopos*)*, 16
get_thermalstrat, 9, *27*, *29*, *42*, *52*
ggboxplot_DepthTS_by_hour, 12, *41*
ggplot, *16*
ggplot_geopos, *14*, 16, *31*, *42*
ggplotly, *14*
ggplotly_geopos, 14, *17*, *31*, *42*
ggplotmap, *14*, *17*

hist_tad, *6*, *13*, 18, *20*, *24*, *32, 33*, *42*, *44, 45*, *49, 50*, *56–58*
hist_tat, *6*, *21*, 22, *32*, *42*, *44, 45*, *56*

image_TempDepthProfiles, *2, 3*, 26, *27–29*, *42*, *47, 48*, *52*
interpolate_PDTs, *26*, *47, 48*, *51*
interpolate_PDTs *(*interpolate_TempDepthProfiles*)*, *28*
interpolate_TempDepthProfiles, *2, 3*, *10, 11*, *26*, 28, *42*, *51, 52*

leaflet_geopos, *14*, *17*, 30, *60*

merge_histos, *6*, *20, 21*, *24*, 32, *42*, *44, 45*, *56–58*
mtext, *40*

oceanmap, *36, 37*

plot, *41*
plot_data_coverage, 34, *54*
plot_DepthTempTS, *8*, 37, *41, 42*, *51*
plot_DepthTempTS_resampled, *42*
plot_DepthTempTS_resampled *(*plot_DepthTempTS*)*, 37
plot_DepthTempTS_resampled_PDT, *42*
plot_DepthTempTS_resampled_PDT *(*plot_DepthTempTS*)*, 37
plot_DepthTS, *13*, *38*, *42*, *54*
plot_DepthTS *(*plot_TS*)*, 39
plot_TS, *8*, *38*, 39, *42*, *50*, *53*
projection, *17*

RchivalTag, 42
read_histos, *20*, *24*, *36*, 44, *56, 57*
read_PDT, *2, 3*, *27*, *29*, *42*, 47, *51, 52*
read_TS, *36*, 49
rebin_histos *(*merge_histos*)*, 32
region_definitions, *14*
resample_DepthTempTS *(*resample_PDT*)*, 51
resample_PDT, 51
resample_TS, 53

set.colorbarp, *26*
simulate_DepthTS, 54
SpatialPolygonsDataFrame, *16, 17*, *30*

sunriset, *4*, *8*, *9*

ts2histos, *19–22*, *24*, *36*, *42–45*, *49*, *50*, 55

unmerge_histos, *5*, *6*, *33*, 58
update_leaflet_elementId, *31*, 59

v, *14*