Package 'trip'

June 29, 2023

Type Package

Title Tracking Data

Version 1.10.0

Depends R (>= 3.3.0)

Imports geodist, MASS, methods, raster, reproj, sp, spatstat.geom, spatstat.explore, glue, viridis, traipse (>= 0.2.0), crsmeta, dplyr, rlang

Suggests adehabitatLT, knitr, testthat, covr, rmarkdown, lubridate, maps, spelling, lattice

Description Access and manipulate spatial tracking data, with straightforward coercion from and to other formats. Filter for speed and create time spent maps from tracking data. There are coercion methods to convert between 'trip' and 'ltraj' from 'adehabitatLT', and between 'trip' and 'psp' and 'ppp' from 'spatstat'. Trip objects can be created from raw or grouped data frames, and from types in the 'sp', sf', 'amt', 'trackeR', 'mousetrap', and other packages, Sumner, MD (2011) <https://figshare.utas.edu.au/articles/thesis/The_tag_location_problem/23209538>.

URL https://github.com/Trackage/trip

BugReports https://github.com/Trackage/trip/issues

NeedsCompilation no

ByteCompile yes

License GPL-3

LazyData yes

VignetteBuilder knitr

Encoding UTF-8

Language en-US

RoxygenNote 7.2.3

Author Michael D. Sumner [aut, cre],
Sebastian Luque [ctb],
Anthony Fischbach [ctb],
Tomislav Hengl [ctb]

Maintainer Michael D. Sumner <mdsumner@gmail.com>

Repository CRAN

Date/Publication 2023-06-29 14:30:02 UTC

R topics documented:

`trip-package` *trip.*

Description

Functions for accessing and manipulating spatial data for animal tracking, with straightforward coercion from and to other formats. Filter for speed and create time spent maps from animal track data. There are coercion methods to convert between 'trip' and 'ltraj' from 'adehabitatLT', and between 'trip' and 'psp' and 'ppp' from 'spatstat'. Trip objects can be created from raw or grouped data frames, and from types in the 'sp', 'sf', 'amt', 'trackeR', and other packages.

`adjust.duplicateTimes` *Adjust duplicate DateTime values*

Description

Duplicated DateTime values within ID are adjusted forward (recursively) by one second until no duplicates are present. This is considered reasonable way of avoiding the nonsensical problem of duplicate times.

Usage

```
adjust.duplicateTimes(time, id)
```

Arguments

`time`	vector of DateTime values
`id`	vector of ID values, matching DateTimes that are assumed sorted within ID

Details

This function is used to remove duplicate time records in animal track data, rather than removing the record completely.

Value

The adjusted DateTime vector is returned.

Warning

I have no idea what goes on at CLS when they output data that are either not ordered by time or have duplicates. If this problem exists in your data it's probably worth finding out why.

See Also

`readArgos`

Examples

```
## DateTimes with a duplicate within ID
tms <- Sys.time() + c(1:6, 6, 7:10) *10
id <- rep("a", length(tms))
range(diff(tms))

## duplicate record is now moved one second forward
tms.adj <- adjust.duplicateTimes(tms, id)
range(diff(tms.adj))
```

`argos.sigma` *Assign numeric values for Argos "class"*

Description

Assign numeric values for Argos "class" by matching the levels available to given numbers. An adjustment is made to allow sigma to be specified in kilometres, and the values returned are the approximate values for longlat degrees. It is assumed that the levels are part of an "ordered" factor from least precise to most precise.

Usage

```
argos.sigma(x, sigma = c(100, 80, 50, 20, 10, 4, 2), adjust = 111.12)
```

Arguments

`x`	factor of Argos location quality "classes"
`sigma`	numeric values (by default in kilometres)
`adjust`	a numeric adjustment to convert from kms to degrees

Details

The available levels in Argos are `levels=c("Z", "B", "A", "0", "1","2", "3")`.

The actual sigma values given by default are (as far as can be determined) a reasonable stab at what Argos believes.

Value

Numeric values for given levels.

Examples

```
cls <- ordered(sample(c("Z", "B", "A", "0", "1", "2", "3"), 30,
                      replace=TRUE),
               levels=c("Z", "B", "A", "0", "1", "2", "3"))
argos.sigma(cls)
```

`as.Other` *As ("trip", other-classes)*

Description

Coercing `trip` objects to other classes.

Function to create a SpatialLinesDataFrame from a trip object, resulting in a line segment for each implicit segment along the tracks. The object stores the start and end times, duration and the ID of the segment.

Usage

```
## S3 method for class 'trip'
as.ppp(X, ..., fatal)

## S3 method for class 'trip'
as.psp(x, ..., from, to)

as.track_xyt.trip(x, ..., from, to)

explode(x, ...)
```

Arguments

`X`	`trip` object.
`...`	reserved for future methods
`fatal`	Logical value, see Details of `as.ppp`
`x`	`trip` object
`from`	see `as.psp` for that method.
`to`	See `as.psp`.

Value

ppp object

psp object

SpatialLinesDataFrame

SpatialLinesDataFrame object with each individual line segment identified by start/end time and trip ID

Examples

```
d <- data.frame(x=1:10, y=rnorm(10), tms=Sys.time() + 1:10, id=gl(2, 5))
sp::coordinates(d) <- ~x+y
tr <- trip(d, c("tms", "id"))

as(tr, "ppp")
d <- data.frame(x=1:10, y=rnorm(10), tms=Sys.time() + 1:10, id=gl(2, 5))
tr <- trip(d, c("tms", "id"))

 as(tr, "psp")
 as.psp(tr)
 d <- data.frame(x=1:10, y=rnorm(10), tms=Sys.time() + 1:10, id=gl(2, 5))
 tr <- trip(d)

spldf <- explode(tr)
summary(tr)
```

---

as.trip *Coercion from other classes to* trip *objects*

Description

Coercing objects to trip class

Usage

```
as.trip(x, ...)
```

Arguments

x,	ltr ltraj object
...	Arguments passed to other methods. Ignored for ltraj method.

Value

S4 trip object

Methods

coerce `signature(from="ltraj", to="trip")`

as.trip `signature(x="ltraj")`

Examples

```
 d <- data.frame(x=1:10, y=rnorm(10), tms=Sys.time() + 1:10, id=gl(2, 5))
tr <- trip(d)
if (require(adehabitatLT)) {
    l <- as(tr, "ltraj")
    ltraj2trip(l)
    as.trip(l)
}
```

---

cut.trip *Split trip events into exact time-based boundaries.*

Description

Split trip events within a single object into exact time boundaries, adding interpolated coordinates as required.

Usage

```
## S3 method for class 'trip'
cut(x, breaks, ...)
```

Arguments

x	A trip object.
breaks	A character string such as the breaks argument for cut.POSIXt, or alternatively a vector of date-time boundaries. (If the latter these must encompass all the time range of the entire trip object.)
...	Unused arguments.

Details

Motion between boundaries is assumed linear and extra coordinates are added at the cut points.

This function was completely rewritten in version 1.1-20.

Value

list of S4 trip objects, each with aligned boundaries in time based on cutting the input into intervals

A list of trip objects, named by the time boundary in which they lie.

Author(s)

Michael D. Sumner and Sebastian Luque

See Also

See also tripGrid.

Examples

```
set.seed(66)
d <- data.frame(x=1:100, y=rnorm(100, 1, 10),
           tms= as.POSIXct(as.character(Sys.time()), tz = "GMT") + c(seq(10, 1000, length=50),
                seq(100, 1500, length=50)), id=gl(2, 50))
sp::coordinates(d) <- ~x+y
tr <- trip(d, c("tms", "id"))

cut(tr, "200 sec")

bound.dates <- seq(min(tr$tms) - 1, max(tr$tms) + 1, length=5)
trip.list <- cut(tr, bound.dates)
bb <- sp::bbox(tr)
cn <- c(20, 8)
g <- sp::GridTopology(bb[, 1], apply(bb, 1, diff) / (cn - 1), cn)

tg <- tripGrid(tr, grid=g)
tg <- sp::as.image.SpatialGridDataFrame(tg)
tg$x <- tg$x - diff(tg$x[1:2]) / 2
tg$y <- tg$y - diff(tg$y[1:2]) / 2

op <- par(mfcol=c(4, 1))
for (i in 1:length(trip.list)) {
  plot(sp::coordinates(tr), pch=16, cex=0.7)
  title(names(trip.list)[i], cex.main=0.9)
  lines(trip.list[[i]])
  abline(h=tg$y, v=tg$x, col="grey")
  image(tripGrid(trip.list[[i]], grid=g), interpolate=FALSE,
  col=c("white", grey(seq(0.2, 0.7,  length=256))),add=TRUE)
  abline(h=tg$y, v=tg$x,  col="grey")
  lines(trip.list[[i]])
  points(trip.list[[i]], pch=16, cex=0.7)
}

par(op)
print("you may need to resize the window to see the grid data")

cn <- c(200, 80)
g <- sp::GridTopology(bb[, 1], apply(bb, 1, diff) / (cn - 1), cn)

tg <- tripGrid(tr, grid=g)
tg <- sp::as.image.SpatialGridDataFrame(tg)
```

```
tg$x <- tg$x - diff(tg$x[1:2]) / 2
tg$y <- tg$y - diff(tg$y[1:2]) / 2

op <- par(mfcol=c(4, 1))
for (i in 1:length(trip.list)) {
  plot(sp::coordinates(tr), pch=16, cex=0.7)
  title(names(trip.list)[i], cex.main=0.9)
  image(tripGrid(trip.list[[i]], grid=g, method="density", sigma=1),
        interpolate=FALSE,
        col=c("white", grey(seq(0.2, 0.7, length=256))),
        add=TRUE)
  lines(trip.list[[i]])
  points(trip.list[[i]], pch=16, cex=0.7)
}

par(op)
print("you may need to resize the window to see the grid data")



data("walrus818", package = "trip")
library(lubridate)
walrus_list <- cut(walrus818, seq(floor_date(min(walrus818$DataDT), "month"),
ceiling_date(max(walrus818$DataDT), "month"), by = "1 month"))
g <- rasterize(walrus818) * NA_real_
stk <- raster::stack(lapply(walrus_list, rasterize, grid = g))
st <- raster::aggregate(stk, fact = 4, fun = sum, na.rm = TRUE)
st[!st > 0] <- NA_real_

plot(st, col = oc.colors(52))
```

`forceCompliance` *Function to ensure dates and times are in order with trip ID*

Description

A convenience function, that removes duplicate rows, sorts by the date-times within ID, and removes duplicates from a data frame or SpatialPointsDataFrame.

Usage

```
forceCompliance(x, tor)
```

Arguments

x	data.frame or SpatialPointsDataFrame-class
tor	character vector of names of date-times and trip ID columns

Value

data.frame or SpatialPointsDataFrame-class.

Note

It's really important that data used are of a given quality, but this function makes the most common trip problems easy to apply.

See Also

trip

---

homedist *Calculate maximum distance from 'home' for each trip*

Description

This function returns a distance from a given 'home' coordinate for each individual trip. Use the home argument to provide a single, common 2-element (x,y or lon,lat) coordinate. If home is NULL (the default), then each individual trip's first location is used.

Usage

```
homedist(x, home = NULL)
```

Arguments

x	trip object
home	see details

Value

numeric vector of distances in km (for longlat), or in the units of the trip's projection

See Also

spDistsN1

`interp_equal` *Track intermediate points*

Description

Calculate great circle intermediate points on longitude, latitude input vectors. A spherical model is used, from the geosphere package.

Usage

```
interp_equal(x, distance = NULL, duration = NULL)
```

Arguments

`x`	trip object
`distance`	optional minimum distance (metres) between interpolated points
`duration`	optional minimum duration (seconds) between interpolated points

Details

For the result to be sensible, the input must either be in longitude/latitude, or be in a projection and have a valid CRS. Great circle movement is assumed, there's no way to use this to interpolate equal-distance in the native projection.

If no input `distance` or `duration` is provided a default is used of 15 points between each input point.

if both `distance` AND `duration` is provided, `distance` is ignored.

Note, the original implementation of this function was called 'interpequal()', and was used for time spent calculations. The functionality is now provided by the traipse package.

Value

S4 trip object with interpolated new locations based on distance or duration parameters

`makeGridTopology` *Generate a GridTopology from a Spatial object*

Description

Sensible defaults are assumed, to match the extents of data to a manageable grid.

Usage

```
makeGridTopology(
  obj,
  cells.dim = c(100, 100),
  xlim = NULL,
  ylim = NULL,
  buffer = 0,
  cellsize = NULL,
  adjust2longlat = FALSE
)
```

Arguments

obj	any Spatial object, or other object for which bbox will work
cells.dim	the number of cells of the grid, x then y
xlim	x limits of the grid
ylim	y limits of the grid
buffer	proportional size of the buffer to add to the grid limits
cellsize	pixel cell size
adjust2longlat	assume cell size is in kilometres and provide simple adjustment for earth-radius cells at the north-south centre of the grid

Details

Approximations for kilometres in longlat can be made using cellsize and adjust2longlat.

Value

S4 class GridTopology with properties set variously from input parameters

---

oc.theme *SeaWiFS ocean colour colours*

Description

Generate ocean colour colours, using the SeaWiFS scheme

Usage

```
oc.theme(x = 50)

oc.colors(n)
```

Arguments

x	Number of colours to generate as part of a theme
n	Number of colours to generate

Details

This is a high-contrast palette, log-scaled originally for ocean chlorophyll.

Value

A set of colours or a theme object.

See Also

Similar functions in sp spplot, bpy.colors

Examples

```
 oc.colors(10)
 library(lattice)
 trellis.par.set(oc.theme())
 d <- data.frame(x=1:10, y=rnorm(10), tms=Sys.time() + 1:10, id=gl(2, 5))
tr <- trip(d)

 tg <- tripGrid(tr)
 plot(tg)
```

---

rasterize *Rasterize trip objects based on line-segment attributes.*

Description

Trip rasterize.

Arguments

x	trip object
y	Raster* object
field	attribute from which differences will be calculated, defaults to the time-stamp between trip locations

Value

RasterLayer

Examples

```
 d <- data.frame(x=1:10, y=rnorm(10), tms=Sys.time() + 1:10, id=gl(2, 5))
tr <- trip(d, c("tms", "id"))

tr$temp <- sort(runif(nrow(tr)))
r <- rasterize(tr)

rasterize(tr, grid = r)

rasterize(tr, r, field = "temp")
rasterize(tr, method = "density")
rasterize(tr, method = "density", grid = r)

rasterize(tr, r, field = "tms")
rasterize(tr, r)
```

`readArgos` *Read Argos "DAT" or "DIAG" files*

Description

Return a (Spatial) data frame of location records from raw Argos files. Multiple files may be read, and each set of records is appended to the data frame in turn. Basic validation of the data is enforced by default.

Usage

```
readArgos(
  x,
  correct.all = TRUE,
  dtFormat = "%Y-%m-%d %H:%M:%S",
  tz = "GMT",
  duplicateTimes.eps = 0.01,
  p4 = "+proj=longlat +ellps=WGS84",
  verbose = FALSE,
  read_alt = NULL,
  ...
)

readDiag(x, return_trip = FALSE, read_alt = 1L, ...)
```

Arguments

`x`	vector of file names of Argos "DAT" or "DIAG" files.
`correct.all`	logical - enforce validity of data as much as possible? (see Details)

dtFormat	the DateTime format used by the Argos data "date" and "time" pasted together
tz	timezone - GMT/UTC is assumed
duplicateTimes.eps	what is the tolerance for times being duplicate?
p4	PROJ.4 projection string, "+proj=longlat +ellps=WGS84" is assumed
verbose	if TRUE, details on date-time adjustment is reported
read_alt	is NULL by default, with longitude and latitude read from the PRV message, if 1 or 2 then attempt is made to read the alternative locations (but these are not always present)
...	reserved for future use
return_trip	for readDiag() if TRUE will return a trip object, use read_alt to control the location

Details

readArgos performs basic validation checks for class trip are made, and enforced based on correct.all:

No duplicate records in the data, these are simply removed. Records are ordered by DateTime ("date", "time", "gmt") within ID ("ptt"). No duplicate DateTime values within ID are allowed: to enforce this the time values are moved forward by one second - this is done recursively and is not robust.

If validation fails the function will return a SpatialPointsDataFrame-class. Files that are not obviously of the required format are skipped.

Argos location quality data "class" are ordered, assuming that the available levels is levels=c("Z", "B", "A", "0", "1", "2", "3").

A projection string is added to the data, assuming the PROJ.4 longlat - if any longitudes are greater than 360 the PROJ.4 argument "+over" is added.

readDiag simply builds a data.frame.

With read_alt the default value NULL returns the PRV location as-is. Some files may have a standardized location, and a dummy. If read_alt is set to 1 or 2 the corresponding "alternative" location is returned. 1 is a standardized location corresponding to the original PRV message, and 2 is a "dummy" location.

Value

readArgos returns a trip object, if all goes well, or simply a SpatialPointsDataFrame-class.

readDiag returns a data.frame with 8 columns:

- lon1,lat1 first pair of coordinates
- lon1,lat1 second pair of coordinates
- gmt DateTimes as POSIXct
- id Platform Transmitting Terminal (PTT) ID
- lq Argos location quality class
- iq some other thing

Warning

This works on some Argos files I have seen.

References

The Argos data documentation was (ca. 2003) at http://www.argos-system.org/manual. Specific details on the PRV ("provide data") format were found in Chapter 4_4_8, originally at 'http://www.cls.fr/manuel/html/chap4/cha

See Also

`trip`, `SpatialPointsDataFrame-class`, `adjust.duplicateTimes`, for manipulating these data, and `argos.sigma` for relating a numeric value to Argos quality "classes".

`sepIdGaps` for splitting the IDs in these data on some minimum gap.

`order`, `duplicated`, , `ordered` for general manipulation of this type.

Examples

```
argosfile <-
  system.file("extdata/argos/98feb.dat", package = "trip", mustWork = TRUE)
argos <- readArgos(argosfile)
```

---

reproj *Reprojection*

Description

A reproj method for trip objects.

Usage

```
## S3 method for class 'trip'
reproj(x, target, ..., source = NULL)
```

Arguments

`x`	trip object
`target`	target projection
`...`	ignored
`source`	projection of source data, usually ignore this for trips

Value

a trip reprojected to 'target'

sda *Filter track for speed, distance and angle.*

Description

Create a filter index of a track for "bad" points with a combination of speed, distance and angle tests.

Usage

```
sda(x, smax, ang = c(15, 25), distlim = c(2.5, 5), pre = NULL)
```

Arguments

x	trip object
smax	maximum speed, in km/h
ang	minimum turning angle/s in degrees
distlim	maximum step lengths in km
pre	include this filter in the removal

Details

This is an independent implementation from that in the package argosfilter by Freitas 2008.

Value

logical vector, with FALSE values where the tests failed

References

Freitas, C., Lydersen, C., Fedak, M. A. and Kovacs, K. M. (2008), A simple new algorithm to filter marine mammal Argos locations. Marine Mammal Science, 24: 315?V325. doi: 10.1111/j.1748-7692.2007.00180.x

sepIdGaps *Separate a set of IDs based on gaps*

Description

A new set of ID levels can be created by separating those given based on a minimum gap in another set of data. This is useful for separating instruments identified only by their ID into separate events in time.

Usage

```
sepIdGaps(id, gapdata, minGap = 3600 * 24 * 7)
```

Arguments

id	existing ID levels
gapdata	data matching id with gaps to use as separators
minGap	the minimum "gap" to use in gapdata to create a new ID level

Details

The assumption is that a week is a long time for a tag not to record anything.

Value

A new set of ID levels, named following the pattern that "ID" split into 3 would provided "ID", "ID_2" and "ID_3".

Warning

It is assumed that each vector provides is sorted by gapdata within id. No checking is done, and so it is suggested that this only be used on ID columns within existing, validated trip objects.

See Also

trip

Examples

```
id <- gl(2, 8)
gd <- Sys.time() + 1:16
gd[c(4:6, 12:16)] <- gd[c(4:6, 12:16)] + 10000
sepIdGaps(id, gd, 1000)
```

---

speedfilter *Filter track data for speed*

Description

Create a filter of a track for "bad" points implying a speed of motion that is unrealistic.

Usage

```
speedfilter(x, max.speed = NULL, test = FALSE)
```

Arguments

x	trip object
max.speed	speed in kilometres (or other unit) per hour, the unit is kilometres if the trip is in longitude latitude coordinates, or in the unit of the projection projection (usually metres per hour)
test	cut the algorithm short and just return first pass

Details

Using an algorithm (McConnnell et al., 1992), points are tested for speed between previous / next and 2nd previous / next points. Contiguous sections with an root mean square speed above a given maximum have their highest rms point removed, then rms is recalculated, until all points are below the maximum. By default an (internal) root mean square function is used, this can be specified by the user.

If the coordinates of the `trip` data are not projected, or NA the distance calculation assumes longlat and kilometres (great circle). For projected coordinates the speed must match the units of the coordinate system. (The PROJ.4 argument "units=km" is suggested).

Value

Logical vector matching positions in the coordinate records that pass the filter.

Warning

This algorithm is destructive, and provides little information about location uncertainty. It is provided because it's commonly used and provides an illustrative benchmark for further work.

It is possible for the filter to become stuck in an infinite loop, depending on the function passed to the filter. Several minutes is probably too long for hundreds of points, test on smaller sections if unsure.

Note

This algorithm was originally taken from IDL code by David Watts at the Australian Antarctic Division, and used in various other environments before the development of this version.

Author(s)

David Watts and Michael D. Sumner

References

The algorithm comes from McConnell, B. J. and Chambers, C. and Fedak, M. A. (1992) Foraging ecology of southern elephant seals in relation to the bathymetry and productivity of the southern ocean. Antarctic Science *4* 393-398

See Also

`sda` for a fast distance angle filter to combine with speed filtering

`TimeOrderedRecords` *TimeOrderedRecords*

Description

Object to identify DateTimes and IDs in a Spatial object.

Usage

```
TimeOrderedRecords(x)
```

Arguments

`x`	Character vector of 2 elements specifying the data columns of DateTimes and IDs

Value

`TimeOrderedRecords` holds a 2-element character vector, naming the data columns of DateTimes and IDs.

Examples

```
##' tor <- TimeOrderedRecords(c("datetime", "ID"))
```

`TimeOrderedRecords-class`
A class for the identifiers of DateTime and ID records in spatial data.

Description

The main use of this class and creator function is for `SpatialPointsDataFrame-class`s which are used with TimeOrderedRecords for the class `trip`.

Value

S4 object, TimeOrderedRecords (a class to hold the names of the date-time and id columns)

Slots

`TOR.columns`**:** 2-element vector of class `"character"`

Note

Future versions may change significantly, this class is very basic and could probably be implemented in a better way. Specifying TOR columns by formula would be a useful addition.

See Also

TimeOrderedRecords, trip for creating trip objects, and trip-class for that class

Examples

```
showClass("TimeOrderedRecords")
tor <- new("TimeOrderedRecords", TOR.columns=c("datetime", "ID"))
```

---

trackAngle *Determine internal angles along a track*

Description

Calculate the angles between subsequent 2-D coordinates using Great Circle distance (spherical) methods.

Usage

```
trackAngle(x)

## S3 method for class 'trip'
trackAngle(x)

## Default S3 method:
trackAngle(x)
```

Arguments

x	trip object, or matrix of 2-columns, with x/y coordinates

Details

If x is a trip object, the return result has an extra element for the start and end point of each individual trip, with value NA.

This is an optimized hybrid of "raster::bearing" and "maptools::gzAzimuth". New code is in the traipse package.

Value

Vector of angles (degrees) between coordinates.

`trackDistance` *Determine distances along a track*

Description

Calculate the distances between subsequent 2-D coordinates using Euclidean or Great Circle distance (WGS84 ellipsoid) methods.

Usage

```
trackDistance(x1, y1, x2, y2, longlat = TRUE, prev = FALSE)
```

Arguments

`x1`	trip object, matrix of 2-columns, with x/y coordinates OR a vector of x start coordinates
`y1`	vector of y start coordinates, if x1 is not a matrix
`x2`	vector of x end coordinates, if x1 is not a matrix
`y2`	vector of y end coordinates, if x1 is not a matrix
`longlat`	if FALSE, Euclidean distance, if TRUE Great Circle distance
`prev`	if TRUE and x1 is a trip, the return value has a padded end value (\"prev\"ious), rather than start (\"next\")

Details

If `x1` is a trip object, arguments `x2`, `x3`, `y2` are ignored and the return result has an extra element for the start point of each individual trip, with value 0.0.

The `prev` argument is ignore unless x1 is a trip.

Distance values are in the units of the input coordinate system when longlat is FALSE, and in kilometres when longlat is TRUE.

This originally used `spDistsN1`, then implemented the sp `gcdist` source directly in R, and now uses `geodist`.

Please see the traipse package for a more modern approach.

Value

Vector of distances between coordinates.

References

Original source taken from sp package, but now using Helmert from Karney (2013) see the geodist package.

Examples

```
 d <- data.frame(x=1:10, y=rnorm(10), tms=Sys.time() + 1:10, id=gl(2, 5))
 tr <- trip(d, c("tms", "id"))

 ## the method knows this is a trip, so there is a distance for every
 ## point, including 0s as the start and at transitions between
 ## individual trips
trackDistance(tr)

## the default method does not know about the trips, so this is
##(n-1) distances between all points
trackDistance(coordinates(tr), longlat = FALSE)

## we get NA at the start, end and at transitions between trips

 angles <- trackAngle(tr)
```

`trip-accessors` *Functions to retrieve DateTime and ID data from within (Spatial) data frames.*

Description

Functions for retrieving the names of the columns used for DateTime and ID, as well as the data.

Usage

```
getTORnames(obj)

getTimeID(obj)

## S3 method for class 'summary.TORdata'
print(x, ...)
```

Arguments

`obj`	`trip` object.
`x`	trip object
`...`	currently ignored

Value

`getTORnames` retrieves the column names from an object extending the class `TimeOrderedRecords`, and `getTimeID` returns the data as a data frame from an object extending the class `TimeOrderedRecords`.

See Also

`trip-class`, for the use of this class with `SpatialPointsDataFrame-class`.

`trip`

Examples

```
tor <- TimeOrderedRecords(c("time", "id"))
getTORnames(tor)
```

`trip-class` *A class for sets of animal trips (track data).*

Description

An extension of `SpatialPointsDataFrame-class` by including `"TimeOrderedRecords"`. The records within the data frame are explicitly ordered by DateTime data within IDs.

Objects from the Class

Objects can be created by calls of the form `trip(obj="SpatialPointsDataFrame", TORnames="TimeOrderedRecords")`. The object contains all the slots present within a `SpatialPointsDataFrame-class`, particularly `data` which contains columns of at least those specified by `TOR.columns`.

See Also

`trip` for examples of directly using the class.

`trip-accessors` describes methods for accessing information on `trip` objects.

Examples

```
showClass("trip")

d <- data.frame(x=1:10, y=rnorm(10), tms=Sys.time() + 1:10, id=gl(2, 5))
tr <- trip(d)

summary(tr)
plot(tr)
lines(tr)

dim(tr)
names(tr)
subset(tr, id == "2")
as.data.frame(tr)

tr[1:3, ]
tr[, 1]
tr[[1]]
```

trip-methods *Function to handle animal track data, organized as* trip *objects*

Description

Create an object of class trip, extending the basic functionality of SpatialPointsDataFrame-class by specifying the data columns that define the "TimeOrdered" quality of the records.

Usage

```
trip(obj, TORnames, correct_all = TRUE)

trip(obj) <- value

## S4 method for signature 'trip,ANY'
split(x, f, drop = FALSE, ...)

## S4 method for signature 'trip,ANY,ANY,ANY'
x[i, j, ..., drop = TRUE]
```

Arguments

obj	A data frame, a grouped data frame or a SpatialPointsDataFrame-class containing at least two columns with the DateTime and ID data as per TORnames. See Details.
TORnames	Either a TimeOrderedRecords object, or a 2-element character vector specifying the DateTime and ID column of obj
correct_all	logical value, if TRUE the input data is corrected for common problems
value	A 4-element character vector specifying the X, Y, DateTime coordinates and ID of obj.
x	trip object
f	grouping vector as per split()
drop	unused but necessary for method consistency
i, j, ...	indices specifying elements to extract

Details

The original form of trip() required very strict input as a 'SpatialPointsDataFrame' and specifying which were the time and ID columns, but the input can be more flexible. If the object is a grouped data frame ('dplyr-style') then the (first) grouping is assumed to define individual trips and that columns 1, 2, 3 are the x-, y-, time-coordinates in that order. It can also be a trip object for redefining TORnames.

The trip() function can ingest track_xyt, telemetry, SpatialPointsDataFrame, sf, trackeRdata, grouped_df, data.frame, tbl_df, mousetrap, and in some cases lists of those objects. Please get in touch if you think something that should work does not.

Track data often contains problems, with missing values in location or time, times out of order or with duplicated times. The `correct_all` argument is set to `TRUE` by default and will report any inconsistencies. Data really should be checked first rather than relying on this auto-cleanup. The following problems are common:

- duplicated records (every column with the same value in another row)
- duplicated date-time values
- missing date-time values, or missing x or y coordinates
- records out of order within trip ID

For some data types there's no formal structure, but a simple convention such as a set of names in a data frame. For example, the VTrack package has `AATAMS1` which may be turned into a trip with `trip(AATAMS1 %>% dplyr::select(longitude, latitude, timestamp, tag.ID, everything())` In time we can add support for all kinds of variants, detected by the names and contents.

See Chapter 2 of the trip thesis for more details.

Value

A trip object, with the usual slots of a `SpatialPointsDataFrame-class` and the added `TimeOrderedRecords`. For the most part this can be treated as a `data.frame` with `Spatial` coordinates.

Methods

Most of the methods available are by virtue of the sp package. Some, such as `split.data.frame` have been added to SPDF so that trip has the same functionality.

trip `signature(obj="SpatialPointsDataFrame",TORnames="ANY")`The main construction.

trip `signature(obj="SpatialPointsDataFrame",TORnames="TimeOrderedRecords")`Object and TimeOrdered records class

trip `signature(obj="ANY", TORnames="TimeOrderedRecords")`: create a `trip` object from a data frame.

trip `signature(obj="trip", TORnames="ANY")`: (Re)-create a `trip` object using a character vector for `TORnames`.

trip `signature(obj="trip", TORnames="TimeOrderedRecords")`: (re)-create a trip object using a `TimeOrderedRecords` object.

See Also

`speedfilter`, and `tripGrid` for simplistic speed filtering and spatial time spent gridding.

Examples

```
d <- data.frame(x=1:10, y=rnorm(10), tms=Sys.time() + 1:10, id=gl(2, 5))

## the simplest way to create a trip is by order of columns
```

```
trip(d)

tr <- trip(d)
 ## real world data in CSV
mi_dat <- read.csv(system.file("extdata/MI_albatross_sub10.csv", package = "trip"),
            stringsAsFactors = FALSE)
mi_dat$gmt <- as.POSIXct(mi_dat$gmt, tz = "UTC")
mi_dat$sp_id <-  sprintf("%s%s_%s_%s", mi_dat$species,
         substr(mi_dat$breeding_status, 1, 1), mi_dat$band, mi_dat$tag_ID)
sp::coordinates(mi_dat) <- c("lon", "lat")
## there are many warnings, but the outcome is fine
## (sp_id == 'WAi_14030938_2123' has < 3 locations as does LMi_12143650_14257)
mi_dat <- trip(mi_dat, c("gmt", "sp_id") )
plot(mi_dat, pch = ".")
#lines(mi_dat)  ## ugly

mi_dat_polar <- reproj(mi_dat, "+proj=stere +lat_0=-90 +lon_0=154 +datum=WGS84")
plot(mi_dat_polar, pch = ".")
lines(mi_dat_polar)
```

`trip.split.exact` *Deprecated functions in trip*

Description

These functions will be declared defunct in a future release.

Usage

```
as.SpatialLinesDataFrame.trip(from)

trip.split.exact(x, dates)

as.ltraj.trip(xy)

as.trip.SpatialLinesDataFrame(from)
```

Arguments

`from`	trip object
`x`	see `cut.trip`
`dates`	see `cut.trip`
`xy`	`trip` object

See Also

`cut.trip`, `as.Other`

`tripGrid` *Generate a grid of time spent by line-to-cell gridding*

Description

Create a grid of time spent from an object of class `trip` by exact cell crossing methods, weighted by the time between locations for separate trip events.

Usage

```
tripGrid(x, grid = NULL, method = "pixellate", ...)
```

Arguments

`x`	object of class `trip`
`grid`	GridTopology - will be generated automatically if NULL
`method`	pixellate or density
`...`	pass arguments to density.psp if that method is chosen (and temporary mechanism to direct users of legacy methods to `tripGrid.interp`)

Details

Zero-length lines cannot be summed directly, their time value is summed by assuming the line is a point. A warning used to be given, but as it achieved nothing but create confusion it has been removed. The density method returns proportionate values, not summed time durations.

See `pixellate.psp` and `pixellate.ppp` for the details on the method used. See `density.psp` for method="density".

Trip events are assumed to start and end as per the object passed in. To work with inferred "cutoff" positions see `split.trip.exact`.

Value

`tripGrid` returns an object of class `SpatialGridDataFrame`, with one column "z" containing the time spent in each cell in seconds.

`tripGrid.interp` *Generate a grid of time spent using approximate methods*

Description

Create a grid of time spent from an object of class `trip` by approximating the time between locations for separate trip events.

Usage

```
tripGrid.interp(x, grid = NULL, method = "count", dur = NULL, ...)

kdePoints(x, h = NULL, grid = NULL, resetTime = TRUE, ...)

countPoints(x, dur = 1, grid = NULL)
```

Arguments

`x`	object of class trip
`grid`	GridTopology - will be generated automatically if NULL
`method`	name of method for quantifying time spent, see Details
`dur`	The \"dur\"ation of time used to interpolate between available locations (see Details)
`...`	other arguments passed to `interpequal` or `kdePoints`
`h`	kernel bandwidth
`resetTime`	rescale result back to the total duration of the input

Details

This set of functions was the the original tripGrid from prior to version 1.1-6. `tripGrid` should be used for more exact and fast calculations assuming linear motion between fixes.

The intention is for `tripGrid.interp` to be used for exploring approximate methods of line-to-cell gridding.

Trip locations are first interpolated, based on an equal-time spacing between records. These interpolated points are then "binned" to a grid of cells. The time spacing is specified by the `dur` (duration) argument to `interpequal` in seconds (i.e. `dur=3600` is used for 1 hour). Shorter time periods will require longer computation with a closer approximation to the total time spent in the gridded result.

Currently there are methods "count" and "kde" for quantifying time spent, corresponding to the functions "countPoints" and "kdePoints". "kde" uses kernel density to smooth the locations, "count" simply counts the points falling in a grid cell.

Value

`tripGrid` returns an object of class `SpatialGridDataFrame`, with one column "z" containing the time spent in each cell in seconds. If kdePoints is used the units are not related to the time values and must be scaled for further use.

See Also

bandwidth.nrd for the calculation of bandwidth values used internally when not supplied by the user

---

walrus818 *Walrus tracking data set.*

Description

Behavior of Pacific Walruses Tracked from the Alaska Coast of the Chukchi Sea.

Details

Data set is provided as a 'trip' object. This is the abstract for the work:

"We tracked movements and haulout foraging behavior of walruses instrumented with satellite-linked data loggers from the Alaskan shores of the Chukchi Sea during the autumn of 2009 (n=13) and 2010 (n=2)." Jay, C. V. and Fischbach, A.S.

Examples

```
data(walrus818)
plot(walrus818)
lines(walrus818)
```

---

world_north *World north polygons*

Description

A spatial polygons object with coastlines of the northern hemisphere.

Usage

```
world_north
```

Format

An object of class SpatialPolygonsDataFrame with 185 rows and 11 columns.

Details

This data set exists purely to avoid requiring reprojection in the vignette, the data uses the same projection as walrus818.

`write_track_kml` *Create a time-continuous KML file*

Description

Export track data to a KML file, for use in Google Earth the continuous time slider.

Usage

```
write_track_kml(
  id,
  lon,
  lat,
  utc,
  z = NULL,
  kml_file = tempfile(fileext = ".kmz"),
  name = NULL,
 altitude_mode = c("absolute", "clampToGround", "clampToSeaFloor", "relativeToGround",
    "relativeToSeaFloor")
)
```

Arguments

`id`	vector of grouping IDs (or a trip object)
`lon`	vector of longitude (ignored if id is a trip)
`lat`	vector of latitude (ignored if id is a trip)
`utc`	vector of POSIXct date-times (ignored if id is a trip)
`z`	vector of elevations, this cannot be set if 'id' is a trip
`kml_file`	filename for KML (KML or KMZ) (must end in .kml or .kmz)
`name`	internal name of dat (derived from kml_file if not specified)
`altitude_mode`	the altitude mode, 'absolute', 'clampToGround', 'clampToSeaFloor', 'relativeToGround', or 'relativeToSeaFloor', see Details

Details

To include altitude set every argument explicitly, by input of separate 'id', 'lon', 'lat', 'utc' and 'z' arguments. If the first argument 'id' is a trip object there is no facility to include the 'z' altitude values.

If 'z' is included it is applied as a third coordinate, with 'altitude_mode' controlling the interpretation, see https://developers.google.com/kml/documentation/altitudemode. If the 'kml_file' ends with ".kmz" the file is compressed, otherwise it must end with ".kml" and the compression archive step is not applied.

Sadly the interactive time slider is only available with the desktop version of Google Earth, the data loads into the browser version but can't be interactive.

Value

character vector, file name location of file produced

Author(s)

Original implementation by Tomislav Hengl in the 'plotKML' package for 'SpatialLinesDataFrame', adapted by M. Sumner for use in continuous-time form.

Examples

```
kfile <- write_track_kml(walrus818[seq(1, 1000, by = 5), ])
print(kfile)
unlink(kfile)
```

Index