## Package ‘trip’

February 20, 2015

<table>
<thead>
<tr>
<th>Type</th>
<th>Package</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>Spatial Analysis of Animal Track Data</td>
</tr>
<tr>
<td>Version</td>
<td>1.1-19</td>
</tr>
<tr>
<td>Depends</td>
<td>methods, sp</td>
</tr>
<tr>
<td>Imports</td>
<td>maptools, MASS, spatstat</td>
</tr>
<tr>
<td>Suggests</td>
<td>adehabitatLT, rgdal</td>
</tr>
</tbody>
</table>

**Description**  Functions for accessing and manipulating spatial data for animal tracking. Filter for speed and create time spent plots from animal track data.

| NeedsCompilation | no |
| License          | GPL-3 |
| Author           | Michael D. Sumner [aut, cre], Sebastian Luque [ctb] |
| Maintainer       | Michael D. Sumner <mdsumner@gmail.com> |
| Repository       | CRAN |
| Date/Publication | 2014-11-05 13:33:05 |

### R topics documented:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>adjust.duplicateTimes</td>
<td>2</td>
</tr>
<tr>
<td>argos.sigma</td>
<td>3</td>
</tr>
<tr>
<td>as.Other</td>
<td>4</td>
</tr>
<tr>
<td>as.trip</td>
<td>5</td>
</tr>
<tr>
<td>cut.trip</td>
<td>6</td>
</tr>
<tr>
<td>filter.penSS</td>
<td>8</td>
</tr>
<tr>
<td>forceCompliance</td>
<td>10</td>
</tr>
<tr>
<td>makeGridTopology</td>
<td>11</td>
</tr>
<tr>
<td>oc.theme</td>
<td>11</td>
</tr>
<tr>
<td>readArgos</td>
<td>12</td>
</tr>
<tr>
<td>sda</td>
<td>14</td>
</tr>
</tbody>
</table>
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 DateTime 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.
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 kilometers, 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

```r
class(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)

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
as.trip

Examples

## Not run:
## Continuing the example from '?trip-methods:
utils::example("trip-methods", package="trip",
    ask=FALSE, echo=FALSE)
as(tr, "ppp")
## End(Not run)
## Not run:
## Continuing the example from '?trip-methods:
utils::example("trip-methods", package="trip",
    ask=FALSE, echo=FALSE)
as.psp.trip(tr)
## End(Not run)
## Continuing the example from '?trip-methods:
utils::example("trip-methods", package="trip",
    ask=FALSE, echo=FALSE)
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.

Methods

coerce signature(from="ltraj", to="trip")
as.trip signature(x="ltraj")

Examples

## Continuing the example from '?trip-methods:
utils::example("trip-methods", package="trip",
    ask=FALSE, echo=FALSE)
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, dates, ...)

Arguments

x
A trip object.

dates
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.

Value

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

```r
## Not run:
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))
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 <- bbox(tr)
ct <- c(20, 8)
g <- GridTopology(bb[, 1], apply(bb, 1, diff) / (ct - 1), ct)

tg <- tripGrid(tr, grid=g)
tg <- 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(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 <- GridTopology(bb[, 1], apply(bb, 1, diff) / (cn - 1), cn)

tg <- tripGrid(tr, grid=g)
tg <- 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(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,
`)
Non-destructive smoothing filter

Description

Non-destructive filter for track data using penalty smoothing on velocity.

Usage

```r
filter.penSS(tr, lambda, first = TRUE, last = TRUE, ...)
```

Arguments

- `tr` A trip object.
- `lambda` Smoothing parameter, see Details.
- `first` Fix the first location and prevent it from being updated by the filter.
- `last` Fix the last location and prevent it from being updated by the filter.
- `...` Arguments passed on to `nlm`

Details

Destructive filters such as `speedfilter` can be recast using a penalty smoothing approach in the style of Green and Silverman (1994).

This filter works by penalizing the fit of the smoothed track to the observed locations by the sum of squared velocities. That is, we trade off goodness of fit against increasing the total sum of squared velocities.

When lambda=0 the smoothed track reproduces the raw track exactly. Increasing lambda favours tracks requiring less extreme velocities, at the expense of reproducing the original locations.

Value

A trip object with updated coordinate values based on the filter - all the data, including original coordinates which are maintained in the trip data frame.
Author(s)
Simon Wotherspoon and Michael Sumner

References

See Also
speedfilter

Examples

```r
## Not run: ## Example takes a few minutes

## Fake some data

## Brownian motion tethered at each end
brownian.bridge <- function(n, r) {
  x <- cumsum(rnorm(n, 0, 1))
  x <- x - (x[1] + seq(0, 1, length=n) * (x[n] - x[1]))
  r * x
}

## Number of days and number of obs
days <- 50
n <- 200

## Make separation between obs gamma distributed
x <- rgamma(n, 3)
x <- cumsum(x)
x <- x/x[n]

## Track is lissajous + brownian bridge
b.scale <- 0.6
r.scale <- sample(c(0.1, 2, 10.2), n, replace=TRUE,
                   prob=c(0.8, 0.18, 0.02))
set.seed(44)
tms <- ISOdate(2001, 1, 1) + trunc(days * 24 * 60 * 60 *x)
lon <- 120 + 20 * sin(2 * pi * x) +
  brownian.bridge(n, b.scale) + rnorm(n, 0, r.scale)
lat <- -40 + 10 * (sin(3 * 2 * pi * x) + cos(2 * pi * x) - 1) +
  brownian.bridge(n, b.scale) + rnorm(n, 0, r.scale)
tr <- new("trip",
    SpatialPointsDataFrame(cbind(lon, lat),
      data.frame(gmt=tms, id="lbb"),
      TimeOrderedRecords(c("gmt", "id")))
plot(tr)
```
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
tor character vector of names of date-times and trip ID columns

Value
data.frame or SpatialPointsDataFrame.

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
**makeGridTopology**

*Generate a GridTopology from a Spatial object*

**Description**

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

**Usage**

```r
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`.

---

**oc.theme**

*SeaWiFS ocean colour colours*

**Description**

Generate ocean colour colours, using the SeaWiFS scheme

**Usage**

```r
oc.theme(x = 50)

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 sp.theme, bpy.colors

Examples

## Not run:

cocolors(10)
library(lattice)
trellis.par.set(oc.theme())
utils::example("trip-methods", package="trip",
               ask=FALSE, echo=FALSE)
tg <- tripGrid(tr)
spplot(tg)

## End(Not run)

---

**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

\[
\text{readArgos}(x, \text{correct.all = TRUE}, \text{dtFormat = "\%Y-\%m-\%d \%H:\%M:\%S"}, \\
\text{tz = "GMT"}, \text{duplicateTimes.eps = 0.01,} \\
\text{p4 = "+proj=longlat +ellps=WGS84", verbose = FALSE})
\]

\[
\text{readDiag}(x)
\]

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

**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. 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"). 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.

**Value**

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

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, it is not a guaranteed method and is in no way linked officially to Argos.

**References**

**See Also**

`trip`, `SpatialPointsDataFrame`, `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.

---

**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**

```r
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 Frietas 2008.

**Value**

Logical vector, with FALSE values where the tests failed

**References**

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
```r
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 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 assumeds 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 not considered to be particularly relevant to the problems involved with location uncertainty in animal tracking. It is provided merely as 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


See Also

trip

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

```r
# 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 SpatialPointsDataFrames which are used with TimeOrderedRecords for the class trip.

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)
**trackDistance**

**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 `gzAzimuth`.

**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` but now implements the `sp gdist` source directly in R.
Value

Vector of distances between coordinates.

Author(s)

Roger Bivand and Michael Sumner

References

Original source taken from sp package.

Examples

```r
## Continuing the example from '?trip-methods'
utils::example("trip-methods", package="trip", 
   ask=FALSE, echo=FALSE)

## 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

## Not run:
require(rgdal)
trackAngle(tr)

## End(Not run)
```

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

```r
getTORnames(obj)

getTimeID(obj)

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

**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.
- `trip`

**Examples**

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

---

**trip-class**

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

**Description**

An extension of SpatialPointsDataFrame 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, 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

```r
showClass("trip")

## Examples of general methods
## Continuing the example from '?trip-methods':
utils::example("trip-methods", package="trip",
               ask=FALSE, echo=FALSE)

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,]

if (exists("porpoise")) {
  dim(porpoise)
names(porpoise)
  porpoise[porpoise["id"] == "GUS", ]
}
```

**trip-methods**

*Function to handle animal track data, organized as "trip"s*

Description

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

Usage

```r
trip(obj, TORnames)
```

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

Arguments

- **obj**
  - A `SpatialPointsDataFrame`, or an object that can be coerced to one, containing at least two columns with the DateTime and ID data as per TORnames. It can also be a `trip` object for redefining TORnames.

- **TORnames**
  - Either a `TimeOrderedRecords` object, or a 2-element character vector specifying the DateTime and ID column of `obj`
trip-methods

x       trip object
i,j,... indices specifying elements to extract
drop    unused but necessary for method consistency

Value

A trip object, with the usual slots of a SpatialPointsDataFrame 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="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 simple(istic) 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))
coordinates(d) <- ~x+y
## this avoids complaints later, but these are not real track data (!)
proj4string(d) <- CRS("+proj=laea")
(tr <- trip(d, c("tms", "id"))

## don't want adehabitatMA to be loaded as a requirement here
## Not run:
## a simple example with the common fixes required for basic track data
dat <- read.csv("trackfile.csv")
names(dat) ## e.g. [1] "long" "lat" "seal" "date" "local" "lq"
library(sp)
coordinates(dat) <- c("long", "lat")

## date/times may be in a particular time zone, please check
dat$gmt <- as.POSIXct(strptime(paste(dat$date, dat$local),
  "%d-%b-%y %H:%M:%S"), tz="GMT")

## if there are problems in the data, this will error
tr <- trip(dat, c("gmt", "seal"))
## The Following Code Tries to Fix Common Problems

### Remove Completely-Duplicated Rows

```r
dat <- dat[! duplicated(dat),]
```

### Order the Rows by Seal, Then by Time

```r
dat <- dat[order(dat$seal, dat$gmt),]
```

### Fudge Duplicated Times

```r
dat$gmt <- adjust.duplicateTimes(dat$gmt, dat$seal)
```

### Finally, Convert to Spatial and Create Trip Object

```r
coordinates(dat) <- c("long", "lat")
tr <- trip(dat, c("gmt", "seal"))
```

### End (Not Run)

### Not Run:

```r
if (require(adehabitatLT)) {
  data(porpoise)
  porpoise <- as.trip(porpoise)
  proj4string(porpoise) <- CRS("+proj=utm +zone=21 +ellps=WGS84 +units=m +no_defs")
  summary(porpoise)
}
```

### Extended Example to Check that Our Projection Metadata Is Correct

```r
library(maptools)
data(wrld_simpl)
library(rgeos)
library(raster)

### 3 Degrees Either Side (For Half a Zone . . .)

```r
ext <- as(extent(spTransform(porpoise, CRS(proj4string(wrld_simpl))))) + 3, "SpatialPolygons"
proj4string(ext) <- CRS(proj4string(wrld_simpl))
```

### Crop to the Buffered Tracks, and Project to Its Native CRS

```r
w <- spTransform(gIntersection(wrld_simpl[grep("United States", wrld_simpl$NAME), ], ext), CRS(proj4string(porpoise)))
```

### End (Not Run)

---

### 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)

tripTransform(x, crs, ...)

Arguments

from  trip object
x     see cut.trip
dates see cut.trip
xy    trip object
crs   CRS object, or PROJ.4 string accepted by CRS
...

Further arguments to spTransform

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 is given. 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

```r
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)
- `h`: kernel bandwidth
- `resetTime`: rescale result back to the total duration of the input
- `...`: other arguments passed to `interpequal` or `kdePoints`
Details

This set of functions was 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"ation 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
Index

*Topic IO
  readArgos, 12
*Topic chron
  cut.trip, 6
*Topic classes
  trip-class, 21
*Topic color
  oc.theme, 11
*Topic manip
  argos.sigma, 3
  cut.trip, 6
  filter.penSS, 8
  makeGridTopology, 11
  readArgos, 12
  sepIdGaps, 15
  speedfilter, 16
  trip-accessors, 20
  tripGrid, 25
  tripGrid.interp, 26
*Topic misc
  filter.penSS, 8
  [,trip,ANY,ANY,ANY-method (trip-methods), 22
  [,trip-method(trip-methods), 22
  [[<-,trip,ANY,missing-method (trip-methods), 22

adjust.duplicateTimes, 2, 14
argos.sigma, 3, 14
as.ltraj.trip(trip.split.exact), 24
as.Other, 4, 25
as.ppp, 4
as.ppp.trip(as.Other), 4
as.psp, 4
as.psp.trip(as.Other), 4
as.SpatialLinesDataFrame.trip (trip.split.exact), 24
as.trip, 5
as.trip.ltraj-method(as.trip), 5
as.trip-methods(as.trip), 5
as.trip.SpatialLinesDataFrame (trip.split.exact), 24

bandwidth.nrd, 27
bpy.colors, 12
coerce,trip,ltraj-method(as.trip), 5
countPoints(tripGrid.interp), 26
CRS, 25
cut.POSIXt, 6
cut.trip, 6, 25
data.frame, 10
duplicated, 14
explode (as.Other), 4
filter.penSS, 8
forceCompliance, 10

getTimeID(trip-accessors), 20
getTORnames(trip-accessors), 20
gzAzimuth, 19
interpequal(tripGrid.interp), 26
kdePoints(tripGrid.interp), 26
lines,trip-method(trip-class), 21
ltraj2trip(as.trip), 5
makeGridTopology, 11

nlm, 8
oc.colors(oc.theme), 11
oc.theme, 11
order, 14
ordered, 14

plot,trip,missing-method(trip-class), 21
print.summary.TORdata(trip-accessors), 20
readArgos, 3, 12
readDiag(readArgos), 12
sda, 14
sepIdGaps, 14, 15
show, summary.TORdata-method
  (trip-class), 21
show,trip-method(trip-class), 21
sp.theme, 12
SpatialPointsDataFrame, 10, 13, 14, 18, 21–23
spDistsN1, 19
speedfilter, 8, 9, 16, 23
spTransform, 25
subset, trip-method(trip-class), 21
summary, trip-method(trip-class), 21
TimeOrderedRecords, 17, 18
TimeOrderedRecords-class, 18
trackAngle, 18
trackDistance, 19
trip, 10, 14, 15, 17, 18, 21
trip(trip-methods), 22
trip,ANY,TimeOrderedRecords-method
  (trip-methods), 22
trip, SpatialPointsDataFrame,ANY-method
  (trip-methods), 22
trip,trip,ANY-method(trip-methods), 22
trip,trip,TimeOrderedRecords-method
  (trip-methods), 22
trip-accessors, 20
trip-class, 21
trip-deprecated(trip.split.exact), 24
trip-methods, 22
trip.split.exact, 24
tripGrid, 6, 23, 25
tripGrid.interp, 25, 26
tripTransform(trip.split.exact), 24