

# Package ‘trip’

February 15, 2012

**Type** Package

**Title** Spatial analysis of animal track data

**Version** 1.1-10

**Depends** methods, sp (>= 0.9-64), spatstat (>= 1.18-4)

**Suggests** maptools, rgdal, adehabitat, lattice

**Author** Michael D. Sumner

**Maintainer** Michael D. Sumner <mdsumner@utas.edu.au>

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

**License** GPL (>= 2)

**Repository** CRAN

**Date/Publication** 2011-05-03 07:28:39

## R topics documented:

adjust.duplicateTimes . . . . .	2
argos.sigma . . . . .	3
as.ltraj.trip . . . . .	4
as.ppp.trip . . . . .	5
as.trip.SpatialLinesDataFrame . . . . .	6
filter.penSS . . . . .	6
forceCompliance . . . . .	8
makeGridTopology . . . . .	9
oc.theme . . . . .	10
ppp-class . . . . .	11
readArgos . . . . .	11
readDiag . . . . .	13
sepIdGaps . . . . .	13
speedfilter . . . . .	14

TimeOrderedRecords . . . . .	16
TimeOrderedRecords-class . . . . .	17
trackDistance . . . . .	18
trip . . . . .	19
trip-class . . . . .	20
trip.split.exact . . . . .	22
tripGrid . . . . .	23
tripGrid.interp . . . . .	24
tripTransform . . . . .	25

<b>Index</b>	<b>27</b>
--------------	-----------

---

`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

<code>time</code>	vector of DateTime values
<code>id</code>	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.

### Author(s)

Michael D. Sumner

## References

<http://staff.acecrc.org.au/~mdsummer/>

## 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 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

```
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.ltraj.trip

*Coercion between trip objects and ltraj objects*


---

**Description**

coercion between classes

**Usage**

```
as.ltraj.trip(xy, typeII = TRUE, slsp = "remove")
ltraj2trip(ltr)
```

**Arguments**

xy	trip object
typeII	see <a href="#">as.ltraj</a>
slsp	details for the <a href="#">ltraj</a> turning angles
ltr	ltraj object

**Methods**

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

**Author(s)**

Michael D. Sumner

**Examples**

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

coordinates(d) <- ~x+y

tr <- trip(d, c("tms", "id"))
require(adehabitat)
l <- as.ltraj.trip(tr)

ltraj2trip(l)
```

**Description**

coercion between classes

**Usage**

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

**Arguments**

X, x	trip object
...	Ignored
fatal	Logical value, see Details of as.ppp
from, to	See as.psp

**Methods**

```
coerce signature(from = "trip", to = "ppp")  
coerce signature(from = "trip", to = "psp")
```

**Author(s)**

Michael D. Sumner

**Examples**

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

as.trip.SpatialLinesDataFrame

*Coercion between trip objects and sp line objects*

---

### Description

coercion between classes

### Usage

```
as.trip.SpatialLinesDataFrame(from)
```

### Arguments

from            trip object

### Methods

**coerce** signature(from = "trip", to = "SpatialLinesDataFrame")

### Author(s)

Michael D. Sumner

### Examples

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

---

filter.penSS

*Non-destructive smoothing filter.*

---

### Description

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

**Usage**

```
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.
...	

**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 lamda = 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**

Green, P. J. and Silverman, B. W. (1994). Nonparametric regression and generalized linear models: a roughness penalty approach. CRC Press.

**See Also**

[speedfilter](#)

**Examples**

```
## 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 = "1bb")), TimeOrderedRecords(c

  plot(tr)

  ## the filtered version
  trf <- filter.penSS(tr, lambda = 1, iterlim = 400, print.level = 1)

  lines(trf)

  ## End(Not run)

```

---

 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**

Dataframe 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.

**Author(s)**

Michael D. Sumner

**See Also**

See Also [trip](#)

---

makeGridTopology	<i>Generate a GridTopology from a Spatial object</i>
------------------	--

---

**Description**

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

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

**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

---

`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.

**Author(s)**

Michael D. Sumner

**See Also**

Similar functions in `sp` [sp.theme](#), [bpy.colors](#)

**Examples**

```
oc.colors(10)
library(lattice)
trellis.par.set(oc.theme)
```

---

ppp-class                      *Virtual classes for coercion to and from trip objects*

---

### Description

Virtual S4 class definition for S3 classes in the spatstat and adehabitat packages to allow S4-style coercion to these classes

### Objects from the Class

Virtual Classes: No objects may be created from these

### Author(s)

Michael D. Sumner

---

readArgos                      *Read Argos "DAT" 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)
```

### Arguments

x	vector of file names of Argos data
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

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", "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.

## Value

A `trip` object, if all goes well, or simply a `SpatialPointsDataFrame`.

## 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.

## Author(s)

Michael D. Sumner

## References

The Argos data documentation is at <http://www.argos-system.org/manual/>. Specific details on the PRV ("provide data") format were found here [http://www.cls.fr/manuel/html/chap4/chap4\\_4\\_8.htm](http://www.cls.fr/manuel/html/chap4/chap4_4_8.htm).

## 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 R manipulation of this type.

---

readDiag	<i>Read Argos DIAG format</i>
----------	-------------------------------

---

**Description**

Create a dataframe from Argos diag files.

**Usage**

```
readDiag(x)
```

**Arguments**

x	one or more names of DIAG files
---	---------------------------------

**Value**

A data frame with 8 columns.

lon1, lat1	first pair of coordinates
lon2, lat2	second pair of coordinates
gmt	DateTimes as POSIXct
id	Platform Transmitting Terminal (PTT) ID
lq	Argos location quality class
iq	some other thing

---

sepIdGaps	<i>Separate a set of IDs based on gaps</i>
-----------	--

---

**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

**Author(s)**

Michael D. Sumner

**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 per hour
test	cut the algorithm short and just return first pass

**Details**

Using an algorithm (McConnell 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 assumed 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)**

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**

[trip](#)

---

TimeOrderedRecords     *Functions to specify and obtain DateTime and ID data from within (Spatial) data frames.*

---

### Description

Specifies the DateTime and ID data within a data frame for objects of class `trip`. Functions also for obtaining the names of the columns used, and the data itself.

### Usage

```
TimeOrderedRecords(x)
getTORnames(obj)
getTimeID(obj)
```

### Arguments

<code>x</code>	Character vector of 2-elements, specifying the data columns of DateTimes and IDs.
<code>obj</code>	<code>trip</code> object.

### Details

These simple functions are for creating classes with TimeOrdered data. The main use is for `SpatialPointsDataFrames` which are used with `TimeOrderedRecords` for the class `trip`.

### Value

`TimeOrderedRecords` returns an object with a 2-element character vector, specifying the columns names. `getTORnames` obtains the column names from an object extending the class, and `getTimeID` returns the data as a data frame from an object extending the class.

### See Also

[trip](#), for the use of this class with [SpatialPointsDataFrame](#)

### Examples

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

---

TimeOrderedRecords-class

*Class "TimeOrderedRecords"*

---

## Description

A simple class to act as a place-holder for DateTime and ID records in spatial data

## Objects from the Class

Objects can be created by calls of the form `new("TimeOrderedRecords", TOR.columns)`. `TOR.columns` are a 2-element character vector specifying the DateTime and ID columns in an object of class `trip`.

## Slots

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

## Methods

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

**trip** signature(obj = "trip", TORnames = "TimeOrderedRecords"): create a trip object from an existing trip object

## 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.

## Author(s)

Michael D. Sumner

## References

~put references to the literature/web site here ~

## See Also

See Also [trip](#) for creating trip objects, and [trip-class](#) for the class.

## Examples

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

---

trackDistance	<i>Determine distance along a track</i>
---------------	---

---

### Description

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

### Usage

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

### Arguments

x1	matrix of 2-columns, with x/y coordinates OR a vector of x start coordinates
x2	vector of x end coordinates, if x1 is not a matrix
y1	vector of y start 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

### Details

trackDistance calls source code identical to sp's spDistN when longlat is TRUE.

This originally used [spDistsN1](#) but now implements the sp gcdist source directly in R.

### Value

Vector of distances between coordinates.

### References

Original source taken from sp package.

---

trip	<i>Function to handle animal track data, organized as "trip"s</i>
------	---

---

**Description**

Extend the basic functionality of a `Spatial` data frame by specifying the data columns that define the "TimeOrdered" quality of the records.

**Usage**

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

**Arguments**

obj	A <code>SpatialPointsDataFrame</code> , containing at least two columns with the Date-Time and ID data as per <code>TORnames</code>
TORnames	Either an object of <code>TimeOrderedRecords</code> , or a 2-element character vector specifying the <code>DateTime</code> and <code>ID</code> column of <code>obj</code>

**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.

**Author(s)**

Michael D. Sumner

**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
tr <- trip(d, c("tms", "id"))

## 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),
```

```

"

## 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
dat <- dat[!duplicated(dat), ]

## order the rows by seal, then by time
dat <- dat[order(dat$seal, dat$gmt), ]

## fudge duplicated times
dat$gmt <- adjust.duplicateTimes(dat$gmt, dat$seal)

## finally, convert to Spatial and create trip object
coordinates(dat) <- c("long", "lat")
tr <- trip(dat, c("gmt", "seal"))

## End(Not run)

```

---

trip-class

Class "trip" ~~~

---

### 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`

### Slots

**TOR.columns:** Object of class "character" specifying the DateTime and ID columns (in that order) in data

**data:** Object of class "data.frame" the native data object for a Spatial data frame

Also, other slots usual to a `SpatialPointsDataFrame`

**Extends**

Class "TimeOrderedRecords", directly. Class "SpatialPointsDataFrame", directly. Class "SpatialPoints", by class "SpatialPointsDataFrame". Class "Spatial", by class "SpatialPointsDataFrame".

**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("ANY"): try to return a trip object

**trip** signature(obj = "SpatialPointsDataFrame", TORnames = "ANY"): The usual construction  
 [ signature(x = "trip"): subset rows or columns as per SpatialPointsDataFrame. In the case that TimeOrderedRecords columns are dropped, the object reverts to the straight Spatial version.

**lines** signature(x = "trip"): add lines to a plot with separate colours for each trip

**plot** signature(x = "trip", y = "missing"): plot as SpatialPoints

**points** signature(x = "trip"): add points to a plot using the Spatial coordinates

**recenter** signature(obj = "trip"): perform coordinate recentering, from the [-180,180] convention to [0, 360]

**show** signature(object = "trip"): print a short summary of the trip data

**summary** signature(object = "trip"): print a summary as per SpatialPointsDataFrame including a summary of the trip data

**text** signature(x = "trip"): add text to a plot using Spatial coordinates

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

**trip** signature(obj = "trip", TORnames = "ANY"): (re)-create a trip object by some other means

**subset** signature(x = "trip", ...): subset a trip in the expected manner.

**Warning**

There are some kludges to allow `trip` to do things, such as replace POSIXt column data using `"$<-.trip"` and `"[[<-.trip"` which should not be necessary once sp implements the new `data.frame` class of R  $\geq 2.4.0$ .

**Author(s)**

Michael D. Sumner

**See Also**

[trip](#) for examples of directly using the class.

---

trip.split.exact      *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

```
trip.split.exact(x, dates)
```

### Arguments

x	A trip object.
dates	A vector of date-time boundaries. These must encompass all the time range of the entire trip object.

### 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

### See Also

See also [tripGrid](#).

### Examples

```
set.seed(66)
d <- data.frame(x = 1:100, y = rnorm(100, 1, 10), tms = Sys.time() + 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"))

bound.dates <- seq(min(tr$tms)-1, max(tr$tms)+1, length = 5)
trip.list <- trip.split.exact(tr, bound.dates)
bb <- bbox(tr)
cn <- c(20, 8)
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)
  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, 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")

```

**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**

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

**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.

---

<code>tripGrid.interp</code>	<i>Generate a grid of time spent using approximate methods</i>
------------------------------	--

---

**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, ...)
interpequal(x, dur = NULL, quiet = FALSE)
countPoints(x, dur = 1, grid = NULL)
kdePoints(x, h = NULL, grid = NULL, resetTime = TRUE, ...)
```

**Arguments**

x	object of class trip
grid	GridTopology - will be generated automatically if NULL
method	name of method for quantifying time spent, see Details
...	other arguments passed to interpequal or kdePoints
dur	The \"dur\"ation of time used to interpolate between available locations (see Details)
quiet	logical - report on difference between time summed and time in trip?
h	numeric vector of two elements specifying bandwidth for kernel density
resetTime	logical - reset the values of the kde grid to match the sum of the total time?

**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"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.

---

tripTransform	<i>Reproject trip objects.</i>
---------------	--------------------------------

---

**Description**

Projection transformation based on CRS strings in rgdal

**Usage**

```
tripTransform(x, crs, ...)
```

**Arguments**

x	trip object with projection metadata
crs	CRS object, or PROJ.4 string accepted by <a href="#">CRS</a>
...	Further arguments to <a href="#">spTransform</a>

**Value**

trip object, with spatial coordinates reprojected

**Note**

this is basically a cheat as I don't want to Depends on rgdal for trip, but cannot figure out the proper way to define things such that Suggests is enough for spTransform methods on trips

**Author(s)**

Michael D. Sumner

**See Also**

[spTransform](#)

**Examples**

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

tr <- trip(d, c("tms", "id"))
proj4string(tr) <- CRS("+proj=laea +lon_0=146")

tripTransform(tr, "+proj=longlat")
```

# Index

- \*Topic **IO**
  - readArgos, 11
  - readDiag, 13
- \*Topic **\textasciitildekw1**
  - filter.penSS, 6
- \*Topic **\textasciitildekw2**
  - filter.penSS, 6
- \*Topic **chron**
  - trip.split.exact, 22
- \*Topic **classes**
  - ppp-class, 11
  - TimeOrderedRecords-class, 17
  - trip-class, 20
- \*Topic **color**
  - oc.theme, 10
- \*Topic **manip**
  - adjust.duplicateTimes, 2
  - argos.sigma, 3
  - as.ltraj.trip, 4
  - as.ppp.trip, 5
  - as.trip.SpatialLinesDataFrame, 6
  - forceCompliance, 8
  - makeGridTopology, 9
  - readArgos, 11
  - readDiag, 13
  - sepIdGaps, 13
  - speedfilter, 14
  - TimeOrderedRecords, 16
  - trackDistance, 18
  - trip, 19
  - trip.split.exact, 22
  - tripGrid, 23
  - tripGrid.interp, 24
  - tripTransform, 25
- \*Topic **spatial**
  - as.ltraj.trip, 4
  - as.ppp.trip, 5
  - as.trip.SpatialLinesDataFrame, 6
- [, trip, ANY, ANY, ANY-method (trip-class), 20
- [[<-, trip, ANY, missing-method (trip-class), 20
- \$<-, trip, character (trip-class), 20
- \$<-, trip, character-method (trip-class), 20
- adjust.duplicateTimes, 2, 12
- argos.sigma, 3, 12
- as.data.frame.trip (trip-class), 20
- as.ltraj, 4
- as.ltraj.trip, 4
- as.ppp (as.ppp.trip), 5
- as.ppp.trip, 5
- as.psp (as.ppp.trip), 5
- as.trip.SpatialLinesDataFrame, 6
- bpy.colors, 10
- coerce, ltraj, trip-method (as.ltraj.trip), 4
- coerce, trip, ltraj-method (as.ltraj.trip), 4
- coerce, trip, ppp-method (as.ppp.trip), 5
- coerce, trip, psp-method (as.ppp.trip), 5
- coerce, trip, SpatialLinesDataFrame-method (as.trip.SpatialLinesDataFrame), 6
- countPoints (tripGrid.interp), 24
- CRS, 26
- duplicated, 12
- filter.penSS, 6
- forceCompliance, 8
- getTimeID (TimeOrderedRecords), 16
- getTORnames (TimeOrderedRecords), 16
- interpequal (tripGrid.interp), 24

kdePoints (tripGrid.interp), 24  
 lines, trip-method (trip-class), 20  
 ltraj, 4  
 ltraj-class (ppp-class), 11  
 ltraj2trip (as.ltraj.trip), 4  
 makeGridTopology, 9  
 names.trip (trip-class), 20  
 names<- .trip (trip-class), 20  
 oc.colors (oc.theme), 10  
 oc.theme, 10  
 order, 12  
 ordered, 12  
 owin-class (ppp-class), 11  
 plot, trip, missing-method (trip-class),  
     20  
 points, trip-method (trip-class), 20  
 ppp-class, 11  
 psp-class (ppp-class), 11  
 readArgos, 3, 11  
 readDiag, 13  
 recenter, trip-method (trip-class), 20  
 sepIdGaps, 12, 13  
 show, trip-method (trip-class), 20  
 sp.theme, 10  
 SpatialPointsDataFrame, 12, 16  
 spDistsN1, 18  
 speedfilter, 7, 14, 19  
 spTransform, 26  
 subset, trip-method (trip-class), 20  
 summary, trip-method (trip-class), 20  
 text, trip-method (trip-class), 20  
 TimeOrderedRecords, 16  
 TimeOrderedRecords-class, 17  
 trackDistance, 18  
 trip, 9, 12, 14–17, 19, 21  
 trip, ANY, ANY (trip-class), 20  
 trip, ANY, TimeOrderedRecords-method  
     (TimeOrderedRecords-class), 17  
 trip, SpatialPointsDataFrame, ANY-method  
     (trip-class), 20  
 trip, trip, ANY-method (trip-class), 20  
 trip, trip, TimeOrderedRecords-method  
     (TimeOrderedRecords-class), 17  
 trip, trip-method (trip-class), 20  
 trip-class, 17  
 trip-class, 20  
 trip.split.exact, 22  
 tripGrid, 19, 22, 23  
 tripGrid.interp, 24  
 tripTransform, 25