Package ‘contact’

October 30, 2019

Title Creating Contact and Social Networks

Version 1.0.1

Description Process spatially- and temporally-discrete data into contact and social networks, and facilitate network analysis by randomizing individuals' movement paths and/or related categorical variables. To use this package, users need only have a dataset containing spatial data (i.e., latitude/longitude, or planar x & y coordinates), individual IDs relating spatial data to specific individuals, and date/time information relating spatial locations to temporal locations. The functionality of this package ranges from data ``cleaning'' via multiple filtration functions, to spatial and temporal data interpolation, and network creation and analysis. Functions within this package are not limited to describing interpersonal contacts. Package functions can also identify and quantify ``contacts'' between individuals and fixed areas (e.g., home ranges, water bodies, buildings, etc.). As such, this package is an incredibly useful resource for facilitating epidemiological, ecological, ethological and sociological research.

Depends R (>= 3.6.0)

Imports ape (>= 5.3), data.table (>= 1.12.2), geosphere (>= 1.5-10), igraph (>= 1.2.4.1), lubridate (>= 1.7.4), methods (>= 3.6.0), parallel (>= 3.6.0), raster (>= 2.9-5), rgdal (>= 1.4-4), rgeos (>= 0.4-3), sf (>= 0.7-4), sp (>= 1.3-1), stats (>= 3.6.0)

License CC0

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

BugReports https://github.com/lanzaslab/contact/issues

Suggests knitr, rmarkdown

VignetteBuilder knitr

NeedsCompilation no

Author Trevor Farthing [aut, cre],
Daniel Dawson [aut],
Cristina Lanzas [ctb]

Maintainer Trevor Farthing <`tsfarthi@ncsu.edu`>

Repository CRAN

Date/Publication 2019-10-30 17:30:03 UTC

R topics documented:

`baboons` *Real-time location data for 19 baboons*

Description

A dataset containing geographic real-time point locations for 19 baboons observed between 03:00:00 and 04:00:00 UTC on August 13th 2012, and are included here primarily to be used for function-testing purposes.

Usage

```
data(baboons)
```

Format

A data frame with 65140 rows and 5 variables:

timestamp The date and time a sensor measurement was taken. Time units are in UTC (Coordinated Universal Time) or GPS time, which is a few leap seconds different from UTC.

location.long The geographic longitude of a location along an animal track as estimated by the processed sensor data. Positive values are east of the Greenwich Meridian, negative values are west of it. Presented as decimal degrees based on the WGS84 reference system.

location.lat The geographic lattitude of a location along an animal track as estimated by the processed sensor data. Positive values are north of the equator, negative values are west of it. Presented as decimal degrees based on the WGS84 reference system.

individual.local.identifier A unique individual identifier for the animal, provided by the data owner.

dateTime The date and time, rounded to the nearest second that a sensor measurement was taken. Derrived from timestamps. Note that this variable is not present in the source data set.

Details

This data file a subset of a larger one published by the Movebank Data Repository (www.datarepository.movebank.org). The larger data set on Movebank contains baboon locations between 08/01/2012 and 08/14/2012. As of the time of publication of this package, a version of the published animal tracking data set can be viewed on Movebank (www.movebank.org) in the study "Collective movement in wild baboons (data from Strandburg-Peshkin et al. 2015)". Individual attributes in the data files are defined here and in the Movebank Attribute Dictionary, available at www.movebank.org/node/2381.

The item descriptions described herein appear in the README text provided for the repository entry verbatim.

Note that according to data publishers, "this dataset does not include interpolated locations or locations that failed the speed filter (see Strandburg-Peshkin et al. 2015 for details)."

Source

https://doi.org/10.5441/001/1.kn0816jn

References

Strandburg-Peshkin A, Farine DR, Couzin ID, Crofoot MC (2015) Shared decision-making drives collective movement in wild baboons. Science. doi:10.1126/science.aaa5099.

Crofoot MC, Kays RW, Wikelski M (2015) Data from: Shared decision-making drives collective movement in wild baboons. Movebank Data Repository. doi:10.5441/001/1.kn0816jn.

Examples

```
data("baboons") #alternatively, you may use the command: contact::baboons
head(baboons)
```

calves *Real-time location data for 10 calves on May 2nd 2016*

Description

A dataset containing planar real-time point locations for 10 calves between 00:00:00 and 02:00:00 UTC on May 2nd, 2016. These data are a subset of the data set published in the supplemental materials of Dawson et al. 2019, and are included here primarily to be used for function-testing purposes.

Usage

```
data(calves)
```

Format

A data frame with 11118 rows and 5 variables:

calftag a unique identifier for each calf

x planar x coordinate

y planar y coordinate

time UTC time at which location fix was obtained

date date on which fix location occurred

Details

Calves were approximately 1.5-year-old beef cattle kept in a 30 X 35 m2 pen at the Kansas State University Beef Cattle Research Center in Manhattan, KS.

Data collection was supported by U.S. National Institute of Health (NIH) grant R01GM117618 as part of the joint National Science Foundation-NIH-United States Department of Agriculture Ecology and Evolution of Infectious Disease program.

Source

https://doi.org/10.1016/j.epidem.2018.08.003

References

Dawson, D.E., Farthing, T.S., Sanderson, M.W., and Lanzas, C. 2019. Transmission on empirical dynamic contact networks is influenced by data processing decisions. Epidemics 26:32-42.

Examples

```
data("calves") #alternatively, you may use the command: contact::calves
head(calves)
```

`calves2018` *Real-time location data for 20 calves in June 2018*

Description

A dataset containing planar real-time point locations for 20 calves between 00:00:00 on June 1st, 2018 and 23:59:59 UTC on June 3, 2018.

Usage

```
data(calves2018)
```

Format

A data frame with 193551 rows and 4 variables:

calftag a unique identifier for each calf

x planar x coordinate

y planar y coordinate

dateTime UTC date and time at which location fix was obtained

Details

Calves were approximately 1.5-year-old castrated male cattle (i.e., steer) kept in a 30 X 35 m2 pen at the Kansas State University Beef Cattle Research Center in Manhattan, KS.

Data collection was supported by U.S. National Institute of Health (NIH) grant R01GM117618 as part of the joint National Science Foundation-NIH-United States Department of Agriculture Ecology and Evolution of Infectious Disease program.

References

Farthing, T.S., Dawson, D.E., Sanderson, M.W., and Lanzas, C. in Review. Accounting for space and uncertainty in real-time-location- system-derived contact networks. Ecology and Evolution.

Examples

```
data("calves2018") #alternatively, you may use the command: contact::calves2018
head(calves2018)
```

`confine` *Identify and Remove Data Points Outside of a Specified Area*

Description

Identifies and removes timepoints when tracked individuals were observed outside of a defined polygon (note: the polygon should be described by the vectors confinementCoord.x (x coordinates) and confinementCoord.y (y coordinates). These vectors must be the same length and the coordinates should be listed in the clockwise or counter-clockwise order that they are observed on the confining polygon.

Usage

```
confine(x, point.x = NULL, point.y = NULL, confinementCoord.x,
  confinementCoord.y, filterOutput = TRUE)
```

Arguments

`x`	Data frame or non-data-frame list that will be filtered.
`point.x`	Vector of length nrow(data.frame(x)) or singular character data, detailing the relevant colname in x, that denotes what planar-x or longitude coordinate information will be used. If argument == NULL, the function assumes a column with the colname "x" exists in x. Defaults to NULL.
`point.y`	Vector of length nrow(data.frame(x)) or singular character data, detailing the relevant colname in x, that denotes what planar-y or lattitude coordinate information will be used. If argument == NULL, the function assumes a column with the colname "y" exists in x. Defaults to NULL.
`confinementCoord.x`	Vector describing x-coordinates of confining-polygon vertices. Each vertex should be described in clockwise or counter-clockwise order, and ordering should be consistent with confinementCoord.y.
`confinementCoord.y`	Vector describing y-coordinates of confining-polygon vertices. Each vertex should be described in clockwise or counter-clockwise order, and ordering should be consistent with confinementCoord.x.
`filterOutput`	Logical. If TRUE, output will be a data frame or list of data frames (depending on whether or not x is a data frame or not) containing only points within confinement polygons. If FALSE, no observations are removed and a "confinement_status" column is appended to x, detailing the relationship of each point to the confinement polygon. Defaults to TRUE.

Details

If users are not actually interested in filtering datasets, but rather, determining what observations should be filtered, they may set filterOutput == FALSE. By doing so, this function will append a "confinement_status" column to the output dataframe, which reports the results of sp::point.in.polygon function that is used to determine if individuals are confined within a given polygon. In this column, values are: 0: point is strictly exterior to pol; 1: point is strictly interior to pol; 2: point lies on the relative interior of an edge of pol; 3: point is a vertex of pol (see ?sp::point.in.polygon).

Value

If filterOutput == TRUE, returns `x` less observations where points were located outside of the polygon defined by points in `confinementCoord.x` and `confinementCoord.y`.

If filterOutput == FALSE, returns `x` appended with a "confinement_status" column which reports the results of sp::point.in.polygon function, which is used to determine if observed points are confined within the polygon defined by points in `confinementCoord.x` and `confinementCoord.y`.

Examples

```
data("calves")

water_trough.x<- c(61.43315, 61.89377, 62.37518, 61.82622) #water polygon x-coordinates
water_trough.y<- c(62.44815, 62.73341, 61.93864, 61.67411) #water polygon y-coordinates

headWater1<- confine(calves, point.x = calves$x, point.y = calves$y,
   confinementCoord.x = water_trough.x, confinementCoord.y = water_trough.y,
  filterOutput = TRUE) #creates a data set comprised ONLY of points within the water polygon.

headWater2<- confine(calves, point.x = calves$x, point.y = calves$y,
   confinementCoord.x = water_trough.x, confinementCoord.y = water_trough.y,
   filterOutput = FALSE) #appends the "confinement_status" column to x.
```

`contactDur.all` *Identify Inter-animal Contacts*

Description

This function uses the output from dist.all to determine when and for how long tracked individuals are in "contact" with one another. Individuals are said to be in a "contact" event if they are observed within a given distance (<= dist.threshold) at a given timestep. Contacts are broken when individuals are observed outside the specified distance threshold from one another for > sec.threshold seconds. Sec.threshold dictates the maximum amount of time between concurrent observations during which potential "contact" events remain unbroken. For example, if sec.threshold == 10, only "contacts" occurring within 10secs of one another will be regarded as a single "contact" event of duration sum(h). If in this case, a time difference between contacts was 11 seconds, the function will report two separate contact events.

The output of this function is a data frame containing a time-ordered contact edge set detailing inter-animal contacts.

Usage

```
contactDur.all(x, dist.threshold = 1, sec.threshold = 10,
  blocking = FALSE, blockUnit = "hours", blockLength = 1,
  equidistant.time = FALSE, parallel = FALSE,
  nCores = parallel::detectCores(), reportParameters = TRUE)
```

Arguments

`x`	Output from the dist.all function. Can be either a data frame or non-data-frame list.
`dist.threshold`	Numeric. Radial distance (in meters) within which "contact" can be said to occur. Defaults to 1. Note: If you are defining conttacts as occurring when polygons intersect, set dist.threshold to 0.
`sec.threshold`	Numeric. Dictates the maximum amount of time between concurrent observations during which potential "contact" events remain unbroken. Defaults to 10.
`blocking`	Logical. If TRUE, contacts will be evaluated for temporal blocks spanning blockLength blockUnit (e.g., 6 hours) within the data set. Defaults to FALSE.
`blockUnit`	Numerical. Describes the number blockUnits within each temporal block. Defaults to 1.
`blockLength`	Character string taking the values, "secs," "mins," "hours," "days," or "weeks." Describes the temporal unit associated with each block. Defaults to "hours."
`equidistant.time`	Logical. If TRUE, location fixes in individuals' movement paths are temporally equidistant (e.g., all fix intervals are 30 seconds). Defaults to FALSE. Note: This is a time-saving argument. A sub-function here calculates the time difference (dt) between each location fix. If all fix intervals in an individuals' path are identical, it saves a lot of time.
`parallel`	Logical. If TRUE, sub-functions within the contactDur.all wrapper will be parallelized. Note that this can significantly speed up processing of relatively small data sets, but may cause R to crash due to lack of available memory when attempting to process large datasets. Defaults to FALSE.
`nCores`	Integer. Describes the number of cores to be dedicated to parallel processes. Defaults to the maximum number of cores available (i.e., parallel::detectCores()).
`reportParameters`	Logical. If TRUE, function argument values will be appended to output data frame(s). Defaults to TRUE.

Value

Returns a data frame (or list of data frames if `x` is a list of data frames) with the following columns:

`dyadMember1`	The unique ID of an individual observed in contact with a specified second individual.
`dyadMember2`	The unique ID of an individual observed in contact with `dyadMember1`.
`dyadID`	The unique dyad ID used to identify the pair of individuals `dyadMember1` and `dyadMember2`.

`contactDuration`	The number of sequential timepoints in `x` that `dyadMember1` and `dyadMember2` were observed to be in contact with one another.
`contactStartTime`	The timepoint in `x` at which contact between `dyadMember1` and `dyadMember2` begins.
`contactEndTime`	The timepoint in `x` at which contact between `dyadMember1` and `dyadMember2` ends.

If blocking == TRUE, the following columns are appended to the output data frame described above:

`block`	Integer ID describing unique blocks of time during which contacts occur.
`block.start`	The timepoint in `x` at which the `block` begins.
`block.end`	The timepoint in `x` at which the `block` ends.
`numBlocks`	Integer describing the total number of time blocks observed within `x` at which the `block`

Finally, if reportParameters == TRUE function arguments `distThreshold`, `secThreshold`, `equidistant.time`, and `blockLength` (if applicable) will be appended to the output data frame.

Examples

```
data(calves)

calves.dateTime<-datetime.append(calves, date = calves$date, time =
    calves$time) #create a dataframe with dateTime identifiers for location foxes

calves.agg<-tempAggregate(calves.dateTime, id = calves.dateTime$calftag,
    dateTime = calves.dateTime$dateTime, point.x = calves.dateTime$x,
    point.y = calves.dateTime$y, secondAgg = 300, extrapolate.left = FALSE,
    extrapolate.right = FALSE, resolutionLevel = "reduced", parallel = FALSE,
    na.rm = TRUE, smooth.type = 1) #smooth locations to 5-min fix intervals.

calves.dist<-dist2All_df(x = calves.agg, parallel = FALSE, dataType = "Point",
    lonlat = FALSE) #calculate distance between all individuals at each timepoint

calves.contact.block<-contactDur.all(x = calves.dist, dist.threshold=1,
    sec.threshold=10, blocking = TRUE, blockUnit = "hours", blockLength = 1,
    equidistant.time = FALSE, parallel = FALSE, reportParameters = TRUE)

calves.contact.NOblock<-contactDur.all(x = calves.dist, dist.threshold=1,
    sec.threshold=10, blocking = TRUE, blockUnit = "hours", blockLength = 1,
    equidistant.time = FALSE, parallel = FALSE, reportParameters = TRUE)
```

contactDur.area *Identify Environmental Contacts*

Description

This function uses the output from distToArea to determine when tracked individuals are in "contact" with fixed locations. Individuals are said to be in a "contact" event (h) if they are observed within a given distance (<= dist.threshold) at a given timestep(i). Sec.threshold dictates the maximum amount of time a single, potential "contact" event should exist. For example, if sec.threshold=10, only "contacts" occurring within 10secs of one another will be regarded as a single "contact" event of duration sum(h). If in this case, a time difference between contacts was 11 seconds, the function will report two separate contact events.

The output of this function is a data frame containing a time-ordered contact edge set detailing animal-environment contacts.

Usage

```
contactDur.area(x, dist.threshold = 1, sec.threshold = 10,
  blocking = FALSE, blockUnit = "mins", blockLength = 10,
  equidistant.time = FALSE, parallel = FALSE,
  nCores = parallel::detectCores(), reportParameters = TRUE)
```

Arguments

`x`	Output from the distToArea function (either df or sf variant). Can be either a data frame or non-data-frame list.
`dist.threshold`	Numeric. Radial distance (in meters) within which "contact" can be said to occur. Defaults to 1. Note: If you are defining conttacts as occurring when polygons intersect, set dist.threshold to 0.
`sec.threshold`	Numeric. Dictates the maximum amount of time between concurrent observations during which potential "contact" events remain unbroken. Defaults to 10.
`blocking`	Logical. If TRUE, contacts will be evaluated for temporal blocks spanning blockLength blockUnit (e.g., 6 hours) within the data set. Defaults to FALSE.
`blockUnit`	Numerical. Describes the number blockUnits within each temporal block. Defaults to 1.
`blockLength`	Character string taking the values, "secs," "mins," "hours," "days," or "weeks." Describes the temporal unit associated with each block. Defaults to "hours."
`equidistant.time`	Logical. If TRUE, location fixes in individuals' movement paths are temporally equidistant (e.g., all fix intervals are 30 seconds). Defaults to FALSE. Note: This is a time-saving argument. A sub-function here calculates the time difference (dt) between each location fix. If all fix intervals are identical, it saves a lot of time.

`parallel`	Logical. If TRUE, sub-functions within the contactDur.all wrapper will be parallelized. Note that this can significantly speed up processing of relatively small data sets, but may cause R to crash due to lack of available memory when attempting to process large datasets. Defaults to FALSE.
`nCores`	Integer. Describes the number of cores to be dedicated to parallel processes. Defaults to the maximum number of cores available (i.e., parallel::detectCores()).
`reportParameters`	Logical. If TRUE, function argument values will be appended to output data frame(s). Defaults to TRUE.

Value

Returns a data frame (or list of data frames if `x` is a list of data frames) with the following columns:

`indiv.id`	The unique ID of an individual observed in contact with a specified fixed point/polygon.
`area.id`	The unique ID of a fixed point/polygon observed in contact with `indiv.id`.
`contact.id`	The unique ID used to identify contacts between the `indiv.id` and `contact.id` pair.
`contactDuration`	The number of sequential timepoints in `x` that `indiv.id` and `area.id` were observed to be in contact.
`contactStartTime`	The timepoint in `x` at which contact between `indiv.id` and `area.id` begins.
`contactEndTime`	The timepoint in `x` at which contact between `indiv.id` and `area.id` ends.

If blocking == TRUE, the following columns are appended to the output data frame described above:

`block`	Integer ID describing unique blocks of time during which contacts occur.
`block.start`	The timepoint in `x` at which the `block` begins.
`block.end`	The timepoint in `x` at which the `block` ends.
`numBlocks`	Integer describing the total number of time blocks observed within `x` at which the `block`

Finally, if reportParameters == TRUE function arguments `distThreshold`, `secThreshold`, `equidistant.time`, and `blockLength` (if applicable) will be appended to the output data frame.

Examples

```
data(calves)

calves.dateTime<-datetime.append(calves, date = calves$date,
  time = calves$time) #create a dataframe with dateTime identifiers for location fixes.

calves.agg<-tempAggregate(calves.dateTime, id = calves.dateTime$calftag,
   dateTime = calves.dateTime$dateTime, point.x = calves.dateTime$x,
   point.y = calves.dateTime$y, secondAgg = 300, extrapolate.left = FALSE,
   extrapolate.right = FALSE, resolutionLevel = "reduced", parallel = FALSE,
   na.rm = TRUE, smooth.type = 1) #smooth to 5-min fix intervals.

water<- data.frame(x = c(61.43315, 61.89377, 62.37518, 61.82622),
                  y = c(62.44815, 62.73341, 61.93864, 61.67411))

water_poly<-data.frame(matrix(ncol = 8, nrow = 1)) #(ncol = number of vertices)*2 #arrange data
colnum = 0
for(h in 1:nrow(water)){
 water_poly[1,colnum + h] <- water$x[h] #pull the x location for each vertex
 water_poly[1, (colnum + 1 + h)] <- water$y[h] #pull the y location for each vertex
 colnum <- colnum + 1
}

water_dist<-dist2Area_df(x = calves.agg, y = water_poly,
  x.id = calves.agg$id, y.id = "water", dateTime = "dateTime", point.x = calves.agg$x,
  point.y = calves.agg$y, poly.xy = NULL, parallel = FALSE, dataType = "Point",
  lonlat = FALSE, numVertices = NULL) #find distances to the water trough

water_contacts <- contactDur.area(water_dist, dist.threshold=1,
  sec.threshold=10, blocking = FALSE, blockUnit = "mins", blockLength = 10,
  equidistant.time = FALSE, parallel = FALSE, reportParameters = TRUE)
```

contactTest — *Determine if Observed Contacts are More or Less Frequent than in a Random Distribution*

Description

This function is used to determine if tracked individuals in an empirical dataset had more or fewer contacts with other tracked individuals/specified locations than would be expected at random. The function works by comparing an empirically-based contactDur.all or contactDur.area function output (emp.input) to the contactDur.all or contactDur.area output generated from randomized data (rand.input).

Usage

```
contactTest(emp.input, rand.input, dist.input, test = "chisq",
  numPermutations = 5000, alternative.hyp = "two.sided",
  importBlocks = FALSE, shuffle.type = 0)
```

Arguments

`emp.input`	List or data frame containing contactDur.all or contactDur.area output refering to the empirical data. Note that if emp.input is a list of data frames, contacts used in analyses will be determined by averaging contacts reported in each list entry.

`rand.input`	List or data frame containing contactDur.all or contactDur.area output refering to the randomized-path data. Note that if rand.input is a list of data frames, contacts used in analyses will be determined by averaging contacts reported in each list entry.
`dist.input`	List or data frame containing dist.all/distToArea function output refering to the empirical data. Note that if test == "chisq," a dist.input argument is required. If test == "mantel," however, dist.input can be set to NULL. This input is used to determine the number of durations that each pair of individuals (or individuals and fixed locations/polygons if dist2Area output is used) were observed during the same timestep (i.e., the maximum number of durations dyad members could potentially be in contact with one another).
`test`	Character string. Describes the statistical test used to evaluate differences. Currently only takes the values "chisq," or "mantel." Defaults to "chisq." More tests will be added in later versions.
`numPermutations`	Integer. Number of times to permute the data given test == "mantel."
`alternative.hyp`	Character string. Describes the nature of the alternative hypothesis being tested when test == "mantel." Takes the values "two.sided," "less," or "greater." Defaults to "two.sided."
`importBlocks`	Logical. If true, each block in emp.input will be analyzed separately. Defaults to FALSE. Note that the "block" column must exist in emp.input.
`shuffle.type`	Integer. Describes which shuffle.type (from the randomizePaths function) was used to randomize the rand.input data set(s). Takes the values "0," "1," or "2." For tests other than "chisq" this value is irrelevant.

Details

Note: The current functionality is limited to comparisons using the X-squared "goodness of fit" test or Mantel test for evaluating correlations between two matrices. Please note that the output of this function changes based on what test is run. The assumptions and intricacies associated with running these tests here are described below in brief.

X-Squared (chisq.test): In this function, chisq.test is used to compare the distribution of observed inter-animal or animal-environment contacts in an empirical dataset, emp.input, to a distribution described in a NULL model, rand.input (i.e., expected contact counts). This test requires equidistant TSWs (temporal-sampling windows; see the tempAggregate function) in each movement path within dist.input. The dist.input (i.e., output from dist.all or dist2Area functions) is used here to determine how frequently each individual was observed in the empirical dataset/block of interest, allowing us to calculate the number of TSWs each individual was present but not involved in contacts. Note here that if X-squared expected values will be very small, approximations of p may not be correct (and in fact, all estimates will be poor). It may be best to weight these tests differently. To address this, We've added the "warning" column to the output which notifies users when the chisq.test function reported that results may be inaccurate.

Mantel test (abe::mantel.test): tests for similarity of the emp.input to rand.input. Please note that abe::mantel.test does not allow for missing values in matrices, so all NAs will be treated as 0. Output is a single data frame describing the test results.

This function was inspired by the methods described by Spiegel et al. 2016. Who determined individuals to be expressing social behavior when nodes had greater degree values than would be expected at random, with randomized contact networks derived from movement paths randomized according to their novel methodology (i.e., shuffle.type == 2). Here, however, by specifying a p-value threshold, users can also identify when more or fewer (demonstrated by the sign of values in the "difference" column) contacts with specific individuals than would be expected at random. Such relationships suggest social affinities or aversions, respectively, may exist between specific individuals.

Note:The default tested column (i.e., categorical data column from which data is drawn to be compared to randomized sets herein) is "id." This means that contacts involving each individual (defined by a unique "id") will be compared to randomized sets. Users may not use any data column for analysis other than "id." If users want to use another categorical data column in analyses rather than "id," we recommend re-processing data (starting from the dist.all/distToArea functions), while specifying this new data as an "id." For example, users may annotate an illness status column to the empirical input, wherein they describe if the tracked individual displayed gastrointestinal ("gastr"), respiratory ("respr"), both ("both"), illness symptoms, or were consistently healthy ("hel") over the course of the tracking period. Users could set this information as the "id," and carry it forward as such through the data-processing pipeline. Ultimately, they could determine if each of these disease states affected contact rates, relative to what would be expected at random.

Note: if importBlocks == TRUE, a "block" column MUST exist in emp.input. However, a "block" column need not exist in rand.contact. If no "block" column exists in rand.input, empirical values in all emp.input blocks will be compared to the overall average values in rand.input. Block columns will also be appended to function outputs.

Value

Output format is dependent on `test` value.

If `test` == "chisq," output will be a list of two data frames. The first data frame contains pairwise analyses of node degree and total edge weight (i.e., the sum of all observed contacts involving each individual). The second data frame contains results of pairwise analyses specific dyadic relationships (e.g., contacts between individuals 1 and 2). Each data frame contains the following columns:

`id1`	the id of the first individual involved in the contact.
`id2`	designation of what is being compared (e.g., totalDegree, totalContactDurations, individual 2, etc.). Content will change depending on which data frame is being observed.
`method`	Statistical test used to determine significance.
`statistic`	Test statistic associated with the specific method.
`p.value`	p.values associated with each comparison.
`df`	Degrees of freedom associated with the statistical test.
`block`	Denotes the relevant time block for each analysis. (if applicable)
`warning`	Denotes if any specific warning occurred during analysis.
`empiricalContactDurations`	Describes the number of observed events in emp.input.

`randContactDurations.mean`	Describes the average number of observed events in rand.input.
`empiricalNoContactDurations`	Describes the number of events that were not observed given the total number of potential events in emp.input.
`randNoContactDurations.mean`	Describes the average number of events that were not observed given the total number of potential events in rand.input.
`difference`	The value given by subtracting randContactDurations.mean from empiricalContactDurations.

If `test` == "mantel," output will be a single data frame with the following columns:

`method`	Statistical test used to determine significance.
`z.val`	z statistic associated with the specific method.
`p.value`	p.values associated with each comparison.
`emp.mean`	mean contacts in the emp.input overall or by block (if applicable).
`rand.mean`	mean contacts in the rand.input overall or by block (if applicable).
`alternative.hyp`	The nature of the alternative hypothesis being tested.
`nperm`	Number of permutations used to generate p value.
`warning`	Denotes if any specific warning occurred during analysis.

References

Farine, D.R., 2017. A guide to null models for animal social network analysis. Methods in Ecology and Evolution 8:1309-1320. https://doi.org/10.1111/2041-210X.12772.

Spiegel, O., Leu, S.T., Sih, A., and C.M. Bull. 2016. Socially interacting or indifferent neighbors? Randomization of movement paths to tease apart social preference and spatial constraints. Methods in Ecology and Evolution 7:971-979. https://doi.org/10.1111/2041-210X.12553.

Mantel, N. 1967. The detection of disease clustering and a generalized regression approach. Cancer Research, 27:209–220.

Examples

```
data(calves)

calves.dateTime<-datetime.append(calves, date = calves$date,
   time = calves$time)

calves.agg<-tempAggregate(calves.dateTime, id = calves.dateTime$calftag,
   dateTime = calves.dateTime$dateTime, point.x = calves.dateTime$x,
   point.y = calves.dateTime$y, secondAgg = 300, extrapolate.left = FALSE,
   extrapolate.right = FALSE, resolutionLevel = "reduced", parallel = FALSE,
   na.rm = TRUE, smooth.type = 1)
```

```
calves.dist<-dist2All_df(x = calves.agg, parallel = FALSE,
   dataType = "Point", lonlat = FALSE)

calves.contact.block<-contactDur.all(x = calves.dist, dist.threshold=1,
   sec.threshold=10, blocking = TRUE, blockUnit = "hours", blockLength = 1,
   equidistant.time = FALSE, parallel = FALSE, reportParameters = TRUE)

calves.agg.rand<-randomizePaths(x = calves.agg, id = "id",
   dateTime = "dateTime", point.x = "x", point.y = "y", poly.xy = NULL,
   parallel = FALSE, dataType = "Point", numVertices = 1, blocking = TRUE,
   blockUnit = "mins", blockLength = 10, shuffle.type = 0, shuffleUnit = NA,
   indivPaths = TRUE, numRandomizations = 1)

calves.dist.rand<-dist2All_df(x = calves.agg.rand, point.x = "x.rand",
   point.y = "y.rand", parallel = FALSE, dataType = "Point", lonlat = FALSE)

calves.contact.rand<-contactDur.all(x = calves.dist.rand,
   dist.threshold=1, sec.threshold=10, blocking = TRUE, blockUnit = "hours",
   blockLength = 1, equidistant.time = FALSE, parallel = FALSE,
   reportParameters = TRUE)

nullTest<- contactTest(emp.input = calves.contact.block,
   rand.input = calves.contact.rand, dist.input = calves.dist,
   importBlocks = FALSE, shuffle.type = 0)
```

`dateFake` *Create Fake Date Information*

Description

This function assigns fake date information, beginning 01/01/startYear, to each empirical timestamp. Users can control what format the output vector is in by changing the dateFormat argument (format: "mdy" = month-day-year, "ymd" = year-month-day, "dmy" = day-month-year, or "ydm" = year-day-month).

This is a sub-function that can be found within datetime.append.

Usage

```
dateFake(timestamp, dateFormat = "mdy", startYear = 2000)
```

Arguments

`timestamp`	Vector of time information with format "hour:minute:second."
`dateFormat`	Character string. Defines how date information will be presented in output. Takes values "mdy" (i.e., month/day/year), "ymd" (i.e., year/month/day), "dmy" (i.e., day/month/year), or "ydm" (i.e., year/day/month). Defaults to "mdy."
`startYear`	Numerical. Denotes what year fake date information will begin if dateFake == TRUE. Defaults to 2000.

Details

Note that the timestamp argument should be a vector of all relevant timepoints. Additionally, timepoints should be in hms ("hour, minute, second") format.

Value

Output is a vector of date values (e.g., "01-1-2000") with length length(`timestamp`).

Examples

```
data("calves")
dateFake(calves$time, dateFormat = "mdy", startYear = 2000)
```

`datetime.append` *Append Date-Time Information to a Dataset*

Description

This function appends date-time information to a dataset in POSIXct date_time format. It also uses functions from the lubridate package and minor calculations to parse out month, day, hour, minute, second, daySecond (the sequentially ordered second of a day), and totalSecond (sequentially ordered second over the course of the study period) of observations in a given dataset with date (format: "mdy" = month/day/year, "ymd" = year/month/day, "dmy" = day/month/year, or "ydm" = year/day/month (note: no preceding zeroes should be included before numbers <10)) and time (format: hour:minute:second (note:preceding zeroes must be included before numbers < 10, ex. 00:00:01)) information, appends this metadata to the dataset, and can assign each day a unique ID.

Usage

```
datetime.append(x, date = NULL, time = NULL, dateTime = NULL,
  dateFormat = "mdy", dateFake = FALSE, startYear = 2000,
  tz.in = "UTC", tz.out = NULL, month = FALSE, day = FALSE,
  year = FALSE, hour = FALSE, minute = FALSE, second = FALSE,
  daySecond = FALSE, totalSecond = FALSE)
```

Arguments

`x`	Data frame or list of data frames to which new information will be appended.
`date`	Vector of length nrow(data.frame(x)) or singular character data, detailing the relevant colname in x, that denotes what date information will be used. If argument == NULL, datetime.append assumes a column with the colname "date" exists in x, or that the dateTime argument != NULL. Defaults to NULL.
`time`	Vector of length nrow(data.frame(x)) or singular character data, detailing the relevant colname in x, that denotes what time information will be used. If argument == NULL, datetime.append assumes a column with the colname "time" exists in x, or that the dateTime argument != NULL. Defaults to NULL.

`dateTime`	Vector of length nrow(data.frame(x)) or singular character data, detailing the relevant colname in x, that denotes what dateTime information will be used. If argument == NULL, date and time arguments must be appropriately defined, OR "date and "time" columns must exist in x. Defaults to NULL.
`dateFormat`	Character string. Defines how date information is presented. Takes values "mdy" (i.e., month/day/year), "ymd" (i.e., year/month/day), "dmy" (i.e., day/month/year), or "ydm" (i.e., year/day/month). Defaults to "mdy."
`dateFake`	Logical. If TRUE, the function will assign fake date information, beginning 01/01/startYear, to each of the timestamps. Defaults to FALSE.
`startYear`	Numerical. Denotes what year fake date information will begin if dateFake == TRUE. Defaults to 2000.
`tz.in`	Character. Identifies the timezone associated with the time/dateTime argument input. Defaults to "UTC." Timezone names often take the form "Country/City." See the listing of timezones at: http://en.wikipedia.org/wiki/List_of_tz_database_time_zones.
`tz.out`	Character. Identifies the timezone that the output dateTime information will be converted to. If NULL, tz.out will be identical to tz.in. Defaults to NULL. Timezone names often take the form "Country/City." See the listing of timezones at: http://en.wikipedia.org/wiki/List_of_tz_database_time_zones.
`month`	Logical. If TRUE, output will contain a "month" column with relevant information derived from dateTime information. Defaults to FALSE.
`day`	Logical. If TRUE, output will contain a "day" column with relevant information derived from dateTime information. Defaults to FALSE.
`year`	Logical. If TRUE, output will contain a "year" column with relevant information derived from dateTime information. Defaults to FALSE.
`hour`	Logical. If TRUE, output will contain a "hour" column with relevant information derived from dateTime information. Defaults to FALSE.
`minute`	Logical. If TRUE, output will contain a "minute" column with relevant information derived from dateTime information. Defaults to FALSE.
`second`	Logical. If TRUE, output will contain a "second" column with relevant information derived from dateTime information. Defaults to FALSE.
`daySecond`	Logical. If TRUE, output will contain a "daySecond" column with information detailing what the second of a given day the associated dateTime value corresponds to. Defaults to FALSE.
`totalSecond`	Logical. If TRUE, output will contain a "totalSecond" column with information detailing what the second of the entire data set the associated dateTime value corresponds to. Defaults to FALSE.

Value

Output is `x` with new columns appended according to corresponding argmuents.

Examples

```
data("calves")
calves.dateTime<-datetime.append(calves, date = calves$date, time = calves$time)
head(calves.dateTime) #see now that a dateTime column exists.
```

`dist2All_df` *Calculate Distances Between All Individuals*

Description

Calculates the distance between all tracked individuals at a given timestep. Users can choose whether to calculate distances based on a single point, or polygons representative of individuals' locations. If individuals set dataType == "Polygon", the distance matrix reported describes the shortest distances between polygons' edges (Note that the rgeos::gDistance function is used to obtain these distances).

Usage

```
dist2All_df(x = NULL, id = NULL, dateTime = NULL, point.x = NULL,
  point.y = NULL, poly.xy = NULL, elev = NULL, parallel = FALSE,
  nCores = parallel::detectCores(), dataType = "Point",
  lonlat = FALSE, numVertices = 4)
```

Arguments

`x`	Data frame or list of data frames containing real-time-location data.
`id`	Vector of length nrow(data.frame(x)) or singular character data, detailing the relevant colname in x, that denotes what unique ids for tracked individuals will be used. If argument == NULL, the function assumes a column with the colname "id" exists in x. Defaults to NULL.
`dateTime`	Vector of length nrow(data.frame(x)) or singular character data, detailing the relevant colname in x, that denotes what dateTime information will be used. If argument == NULL, the function assumes a column with the colname "dateTime" exists in x. Defaults to NULL.
`point.x`	Vector of length nrow(data.frame(x)) or singular character data, detailing the relevant colname in x, that denotes what planar-x or longitude coordinate information will be used. If argument == NULL, the function assumes a column with the colname "x" exists in x. Defaults to NULL.
`point.y`	Vector of length nrow(data.frame(x)) or singular character data, detailing the relevant colname in x, that denotes what planar-y or lattitude coordinate information will be used. If argument == NULL, the function assumes a column with the colname "y" exists in x. Defaults to NULL.
`poly.xy`	Columns within x denoting polygon xy-coordinates. Polygon coordinates must be arranged in the format of those in referencePointToPolygon output. Defaults to NULL.
`elev`	Vector of length nrow(data.frame(x)) or singular character data, detailing the relevant colname in x, that denotes vertical positioning of each individual in 3D space (e.g., elevation). If argument != NULL, relative vertical positioning will be incorporated into distance calculations. Defaults to NULL.

`parallel`	Logical. If TRUE, sub-functions within the dist2All wrapper will be parallelized. Note that this can significantly speed up processing of relatively small data sets, but may cause R to crash due to lack of available memory when attempting to process large datasets. Defaults to FALSE.
`nCores`	Integer. Describes the number of cores to be dedicated to parallel processes. Defaults to the maximum number of cores available (i.e., parallel::detectCores()).
`dataType`	Character string refering to the type of real-time-location data presented in x, taking values of "Point" or "Polygon." If argument == "Point," individuals' locations are drawn from point.x and point.y. If argument == "Polygon," individuals' locations are drawn from poly.xy. Defaults to "Point."
`lonlat`	Logical. If TRUE, point.x and point.y contain geographic coordinates (i.e., longitude and lattitude). If FALSE, point.x and point.y contain planar coordinates. Defaults to FALSE.
`numVertices`	Numerical. If dataType == "Polygon," users must specify the number of vertices contained in each polygon. Defaults to 4. Note: all polygons must contain the same number of vertices.

Details

If dataType == "Point," users have the option of setting lonlat == TRUE (by default lonlat == FALSE). lonlat is a logical argument that tells the function to calculate the distance between points on the WGS ellipsoid (if lonlat == TRUE), or on a plane (lonlat == FALSE) (see raster::pointDistance). If lonlat == TRUE, coordinates should be in degrees. Otherwise, coordinates should represent planar ('Euclidean') space (e.g. units of meters).This function is not currently able to calculate distances between polygons on the WGS ellipsoid (i.e., if dataType == "Polygon," lonlat must = FALSE). We aim to address this issue in future versions.

Note that if inputting a separate matrix/dataframe with polygon xy coordinates (poly.xy), coordinates must be arranged in the format of those in referencePointToPolygon outputs (i.e., col1 = point1.x, col2 = point1.y, col3 =point2.x, col4 = point2.y, etc., with points listed in a clockwise (or counter-clockwise) order).

Value

Returns a data frame (or list of data frames if `x` is a list of data frames) with the following columns:

`dateTime`	The unique date-time information corresponding to when tracked individuals were observed in `x`.
`totalIndividuals`	The total number of individuals observed at least one time within `x`.
`individualsAtTimestep`	The number of individuals in `x` observed at the timepoint described in the `dateTime` column.
`id`	The unique ID of a tracked individual for which we will evaluate distances to all other individuals observed in `x`.
`dist.to.indiv...`	The observed distance between the individual described in the `id` column to every other individual observed at specific timepoints.

Examples

```
data(calves)

calves.dateTime<-datetime.append(calves, date = calves$date,
   time = calves$time)

calves.agg<-tempAggregate(calves.dateTime, id = calves.dateTime$calftag,
   dateTime = calves.dateTime$dateTime, point.x = calves.dateTime$x,
   point.y = calves.dateTime$y, secondAgg = 300, extrapolate.left = FALSE,
   extrapolate.right = FALSE, resolutionLevel = "reduced", parallel = FALSE,
   na.rm = TRUE, smooth.type = 1) #smooth locations to 5-min fix intervals.

calves.dist<-dist2All_df(x = calves.agg, parallel = FALSE, dataType = "Point",
   lonlat = FALSE) #calculate distance between all individuals at each timepoint.
```

`dist2Area_df` *Calculate Distances Between Individuals and Fixed Points/Polygons*

Description

Calculate distances (either planar or great circle - see dist.all) between each individual, reported in x, and a fixed point(s)/polygon(s), reported in y, at each timestep.

Usage

```
dist2Area_df(x = NULL, y = NULL, x.id = NULL, y.id = NULL,
  dateTime = NULL, point.x = NULL, point.y = NULL, poly.xy = NULL,
  parallel = FALSE, nCores = parallel::detectCores(),
  dataType = "Point", lonlat = FALSE, numVertices = 4)
```

Arguments

`x`	Data frame or list of data frames containing real-time-location data for individuals.
`y`	Data frame or list of data frames describing fixed-area polygons/points for which we will calculate distances relative to tracked individuals at all time steps. Polygons contained within the same data frame must have the same number of vertices.
`x.id`	Vector of length nrow(data.frame(x)) or singular character data, detailing the relevant colname in x, that denotes what unique ids for tracked individuals will be used. If argument == NULL, the function assumes a column with the colname "id" exists in x. Defaults to NULL.
`y.id`	Vector of length sum(nrow(data.frame(y[1:length(y)]))) or singular character data, detailing the relevant colname in y, that denotes what unique ids for fixed-area polygons/points will be used. If argument == NULL, the function assumes a column with the colname "id" may exist in y. If such a column does exist,

	fixed-area polygons will be assigned unique ids based on values in this column. If no such column exists, fixed-area polygons/points will be assigned sequential numbers as unique identifiers. Defaults to NULL.
`dateTime`	Vector of length nrow(data.frame(x)) or singular character data, detailing the relevant colname in x, that denotes what dateTime information will be used. If argument == NULL, the function assumes a column with the colname "dateTime" exists in x. Defaults to NULL.
`point.x`	Vector of length nrow(data.frame(x)) or singular character data, detailing the relevant colname in x, that denotes what planar-x or longitude coordinate information will be used. If argument == NULL, the function assumes a column with the colname "x" exists in x. Defaults to NULL.
`point.y`	Vector of length nrow(data.frame(x)) or singular character data, detailing the relevant colname in x, that denotes what planar-y or lattitude coordinate information will be used. If argument == NULL, the function assumes a column with the colname "y" exists in x. Defaults to NULL.
`poly.xy`	Columns within x denoting polygon xy-coordinates. Polygon coordinates must be arranged in the format of those in referencePointToPolygon output. Defaults to NULL.
`parallel`	Logical. If TRUE, sub-functions within the dist2Area_df wrapper will be parallelized. Note that this can significantly speed up processing of relatively small data sets, but may cause R to crash due to lack of available memory when attempting to process large datasets. Defaults to FALSE.
`nCores`	Integer. Describes the number of cores to be dedicated to parallel processes. Defaults to the maximum number of cores available (i.e., parallel::detectCores()).
`dataType`	Character string refering to the type of real-time-location data presented in x, taking values of "Point" or "Polygon." If argument == "Point," individuals' locations are drawn from point.x and point.y. If argument == "Polygon," individuals' locations are drawn from poly.xy. Defaults to "Point."
`lonlat`	Logical. If TRUE, point.x and point.y contain geographic coordinates (i.e., longitude and lattitude). If FALSE, point.x and point.y contain planar coordinates. Defaults to FALSE.
`numVertices`	Numerical. If dataType == "Polygon," users must specify the number of vertices contained in each polygon described in x. Defaults to 4. Note: all polygons must contain the same number of vertices.

Details

Polygon coordinates (in both x and y inputs) must be arranged in the format of those in referencePointToPolygon outputs (i.e., col1 = point1.x, col2 = point1.y, col3 =point2.x, col4 = point2.y, etc., with points listed in a clockwise (or counter-clockwise) order).

This variant of dist2Area requires x and y inputs to be non-shapefile data.

Value

Returns a data frame (or list of data frames if `x` is a list of data frames) with the following columns:

`dateTime`	The unique date-time information corresponding to when tracked individuals were observed in `x`.
`totalIndividuals`	The total number of individuals observed at least one time within `x`.
`individualsAtTimestep`	The number of individuals in `x` observed at the timepoint described in the `dateTime` column.
`id`	The unique ID of a tracked individual for which we will evaluate distances to all other individuals observed in `x`.
`dist.to...`	The observed distance between the individual described in the `id` column to every each polygon/fixed location

Examples

```
data(calves)

calves.dateTime<-datetime.append(calves, date = calves$date,
  time = calves$time) #create a dataframe with dateTime identifiers for location fixes.

calves.agg<-tempAggregate(calves.dateTime, id = calves.dateTime$calftag,
   dateTime = calves.dateTime$dateTime, point.x = calves.dateTime$x,
   point.y = calves.dateTime$y, secondAgg = 300, extrapolate.left = FALSE,
   extrapolate.right = FALSE, resolutionLevel = "reduced", parallel = FALSE,
   na.rm = TRUE, smooth.type = 1) #smooth to 5-min fix intervals.

water<- data.frame(x = c(61.43315, 61.89377, 62.37518, 61.82622),
                 y = c(62.44815, 62.73341, 61.93864, 61.67411)) #delineate water polygon

water_poly<-data.frame(matrix(ncol = 8, nrow = 1)) #make coordinates to dist2Area specifications
colnum = 0
for(h in 1:nrow(water)){
 water_poly[1,colnum + h] <- water$x[h] #pull the x location for each vertex
 water_poly[1, (colnum + 1 + h)] <- water$y[h] #pull the y location for each vertex
 colnum <- colnum + 1
}

water_dist<-dist2Area_df(x = calves.agg, y = water_poly,
  x.id = calves.agg$id, y.id = "water", dateTime = "dateTime", point.x = calves.agg$x,
  point.y = calves.agg$y, poly.xy = NULL, parallel = FALSE, dataType = "Point",
  lonlat = FALSE, numVertices = NULL)
```

`dt.calc` *Calculate Time Difference Between Relocations*

Description

This function calculates the time difference between relocation events, accounting for individuals' ids. This function has the capability to calculate the differences between sequential timepoints related to two different features (e.g., contactStartTime and contactEndTime) if both dateTime1 and dateTime2 are defined, or just sequential timepoints from a single vector (e.g., contactStartTime) if only dateTime1 is defined.

This is a sub-function contained within contactDur variants and contactTest functions.

Usage

```
dt.calc(x = NULL, id = NULL, dateTime1 = NULL, dateTime2 = NULL,
  timeUnits = "secs", parallel = FALSE,
  nCores = parallel::detectCores(), timeStepRelation = 1)
```

Arguments

`x`	data frame containing time data. If NULL at least dateTime must be defined. Defaults to NULL.
`id`	Vector of length nrow(data.frame(x)) or singular character data, detailing the relevant colname in x, that denotes what unique ids for tracked individuals will be used. If argument == NULL, the function assumes a column with the colname "id" exists in x. Defaults to NULL.
`dateTime1`	Vector of length nrow(data.frame(x)) or singular character data, detailing the relevant colname in x, that denotes what dateTime information will be used. If argument == NULL, the function assumes a column with the colname "dateTime" exists in x. Defaults to NULL.
`dateTime2`	Vector of length nrow(data.frame(x)) or singular character data, detailing the relevant colname in x, that denotes what dateTime information will be used. If argument == NULL, the function will calculate differences between sequential timepoints in dateTime1. If != NULL, the function will calculate differences between dateTime1 and dateTime2 values. Defaults to NULL.
`timeUnits`	Chracter string describing the time unit of calculated differences. It takes the values "secs," "mins," "hours," "days," or "weeks." Defaults to "secs."
`parallel`	Logical. If TRUE, sub-functions within the dt.calc wrapper will be parallelized. Note that this can significantly speed up processing of relatively small data sets, but may cause R to crash due to lack of available memory when attempting to process large datasets. Defaults to FALSE.
`nCores`	Integer. Describes the number of cores to be dedicated to parallel processes. Defaults to the maximum number of cores available (i.e., parallel::detectCores()).
`timeStepRelation`	Numerical. Takes the value "1" or "2." If argument == "1," dt values in output represent the difference between time t and time t-1. If argument == "2," dt values in output represent the difference between time t and time t+1. Defaults to 1.

Value

Output is a data frame with the following columns

`id`	The unique ID of a tracked individual.

`dt`	Time difference between relocation events.
`units`	Temporal unit defined by `timeUnits` argument.

Examples

```
data(calves) #load calves data set
calves.datetime<-datetime.append(calves)
dt<-dt.calc(x = calves.datetime, id = calves.datetime$calftag,
   dateTime1 = calves.datetime$dateTime, dateTime2 = NULL,
   timeUnits = "secs", parallel = FALSE, timeStepRelation = 1)

head(dt)
```

---

dup *Identify and Remove Duplicated Data Points*

Description

dup (a.k.a. Multiple instance filter) identifies and removes timepoints when tracked individuals were observed in >1 place concurrently. If avg == TRUE, duplicates are replaced by a single row describing an individuals' average location (e.g., planar xy coordinates) during the duplicated time point. If avg == FALSE, all duplicated timepoints will be removed, as there is no way for the function to determine which instance among the duplicates should stay. If users are not actually interested in filtering datasets, but rather, determining what observations should be filtered, they may set filterOutput == FALSE. By doing so, this function will append a "duplicated" column to the dataset, which reports values that describe if any timepoints in a given individual's path are duplicated. Values are: 0: timepoint is not duplicated, 1: timepoint is duplicated.

Usage

```
dup(x, id = NULL, point.x = NULL, point.y = NULL, dateTime = NULL,
  avg = TRUE, parallel = FALSE, nCores = parallel::detectCores(),
  filterOutput = TRUE)
```

Arguments

`x`	Data frame containing real-time-location data that will be filtered.
`id`	Vector of length nrow(data.frame(x)) or singular character data, detailing the relevant colname in x, that denotes what unique ids for tracked individuals will be used. If argument == NULL, the function assumes a column with the colname "id" exists in x. Defaults to NULL.
`point.x`	Vector of length nrow(data.frame(x)) or singular character data, detailing the relevant colname in x, that denotes what planar-x or longitude coordinate information will be used. If argument == NULL, the function assumes a column with the colname "x" exists in x. Defaults to NULL.

`point.y`	Vector of length nrow(data.frame(x)) or singular character data, detailing the relevant colname in x, that denotes what planar-y or lattitude coordinate information will be used. If argument == NULL, the function assumes a column with the colname "y" exists in x. Defaults to NULL.
`dateTime`	Vector of length nrow(data.frame(x)) or singular character data, detailing the relevant colname in x, that denotes what dateTime information will be used. If argument == NULL, the function assumes a column with the colname "dateTime" exists in x. Defaults to NULL.
`avg`	Logical. If TRUE, point.x and point.y values for duplicated time steps will be averaged, producing a singular point for all time steps in individuals' movement paths. If FALSE, all duplicated time steps are removed from the data set.
`parallel`	Logical. If TRUE, sub-functions within the dup wrapper will be parallelized. Note that this can significantly speed up processing of relatively small data sets, but may cause R to crash due to lack of available memory when attempting to process large datasets. Defaults to FALSE.
`nCores`	Integer. Describes the number of cores to be dedicated to parallel processes. Defaults to the maximum number of cores available (i.e., parallel::detectCores()).
`filterOutput`	Logical. If TRUE, output will be a data frame containing only movement paths with non-duplicated timesteps. If FALSE, no observations are removed and a "duplicated" column is appended to x, detailing if time steps are duplicated (column value == 1), or not (column value == 0). Defaults to TRUE.

Details

If users want to remove specific duplicated observations, we suggest setting filterOutput == FALSE, reviewing what duplicated timepoints exist in individuals' paths, and manually removing observations of interest.

Value

If filterOutput == TRUE, returns `x` less observations at duplicated timepoints.

If filterOutput == FALSE, returns `x` appended with a "duplicated" column which reports timepoints are duplicated (column value == 1), or not (column value == 0).

Examples

```
data(calves2018) #load the data set

calves_dup<- dup(calves2018, id = calves2018$calftag,
   point.x = calves2018$x, point.y = calves2018$y,
   dateTime = calves2018$dateTime, avg = FALSE, parallel = FALSE,
   filterOutput = TRUE) #there were no duplicates to remove in the first place.
```

`findDistThresh` *Identify Distance Threshold for Contact*

Description

Sample from a multivariate normal distribution to create "in-contact" point pairs (n = n1) based on real-time-location systems accuracy, and generate distribution of length n2 describing average distances between point pairs. This function outputs upper confidence intervals associated with average distances between "in-contact" points.

Usage

```
findDistThresh(n1 = 1000, n2 = 1000, acc.Dist1 = 0.5,
  acc.Dist2 = NULL, pWithin1 = 90, pWithin2 = NULL, spTh = 0.666)
```

Arguments

`n1`	Numerical. Number of points used in the expected-distance distribution(s). Defaults to 1000.
`n2`	Numerical. Number of expected-distance distribution iterations to be averaaged. Defaults to 1000.
`acc.Dist1`	Numerical. Accuracy distance for point 1.
`acc.Dist2`	Numerical. Accuracy distance for point 2. If == NULL, defaults to acc.Dist1 value.
`pWithin1`	Numerical. Percentage of data points within acc.Dist of true locations for point 1.
`pWithin2`	Numerical. Percentage of data points within acc.Dist of true locations for point 2. If == NULL, defaults to pWithin1 value.
`spTh`	Numerical. Pre-determined distance representing biological threshold for contact.

Details

This function is for adjusting contact-distance thresholds (spTh) to account for positional accuracty of real-time-location systems, assuming random (non-biased) error in location-fix positions relative to true locations. Essentially this function can be used to determine an adjusted spTh value that likely includes >= 99-percent of true contacts defined using the initial spTh.

Value

Output is a named vector with 22 observations describing the mean, max, and upper 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95, and 99-percent CI values calculated from the contact-distance distribution.

References

Farthing, T.S., Dawson, D.E., Sanderson, M.W., and Lanzas, C. in Review. Accounting for space and uncertainty in real-time-location- system-derived contact networks. Ecology and Evolution.

Examples

```
findDistThresh(n1 = 10, n2 = 10, acc.Dist1 = 0.5, acc.Dist2 = NULL,
   pWithin1 = 90, pWithin2 = NULL, spTh = 0.5)

findDistThresh(n1 = 10, n2 = 10, acc.Dist1 = 0.5, acc.Dist2 = NULL,
   pWithin1 = 90, pWithin2 = NULL, spTh = 0)
```

---

makePlanar *Project Geographic Coordinates onto a Plane*

Description

This function converts lon/lat data (decimal degrees) from a geographic coordinate system to planar coordinates using a custom azimuthal equidistant projection, and appends these new coordinates to an input data frame (x). By default, the function assumes longitude and latitude coordinates were produced using the WGS84 datum, but users may change this datum if they wish.

Usage

```
makePlanar(x = NULL, x.lon = NULL, x.lat = NULL, origin.lon = NULL,
  origin.lat = NULL, datum = "WGS84")
```

Arguments

`x`	Data frame or matrix containing geographic data. Defaults to NULL.
`x.lon`	Vector of length(nrow(x)) or singular character data, detailng the relevant colname in x, that denotes what longitude information will be used. If argument == NULL, makePlanar assumes a column with the colname "lon" exists in x. Defaults to NULL.
`x.lat`	Vector of length(nrow(x)) or singular character data, detailing the relevant colname in x, that denotes what latitude information will be used. If argument == NULL, makePlanar assumes a column with the colname "lat" exists in x. Defaults to NULL.
`origin.lon`	Numerical. Describes the longitude will be used as the origin-point longitude for the azimuthal-equidistant projection. If NULL, defaults to the longitude of the data set's centroid. Defaults to NULL.
`origin.lat`	Numerical. Describes the latitude will be used as the origin-point latitude for the azimuthal-equidistant projection. If NULL, defaults to the latitude of the data set's centroid. Defaults to NULL.
`datum`	Character string describing the datum used to generate x.lon and x.lat. Defaults to "WGS84."

Details

Users may specify longitude and latitude coordinates to become the origin of the projection (i.e., the (0,0) coordinate). If they do not specify these values, however, the function calculates the centroid of the data and will use this as the origin point.

Note: this function does not allow any NA coordinate values in longitude/latitude vectors. If NAs exist you will get the following error: "Error in .local(obj, ...) : NA values in coordinates." If NAs exist in your data, we suggest 1.) removing them, or 2.) smoothing data using contact::tempAggregate prior to running this function.

Value

Output is `x` appended with the following columns:

`planar.x`	Planar x-coordinate values derived from longitude observations.
`planar.y`	Planar y-coordinate values derived from latitude observations.
`origin.lon`	Longitude of the origin point, either user specified or the longitude of the data's centroid.
`origin.lat`	Latitude of the origin point, either user specified or the latitude of the data's centroid.
`origin.distance`	Linear distance (m) between every point and the origin point.

Examples

```
data(baboons)

head(baboons) #see that locations are in geographic coordinates

lon.na <- which(is.na(baboons$location.long) == TRUE) #pull row ids of lon NAs
lat.na <- which(is.na(baboons$location.lat) == TRUE) #pull row ids of lat NAs

baboons.naRM <- droplevels(baboons[-unique(c(lon.na, lat.na)),]) #remove NAs

baboons.naRM_planar <- makePlanar(x = baboons.naRM,
   x.lon = baboons.naRM$location.long, x.lat = baboons.naRM$location.lat,
   origin.lon = NULL, origin.lat = NULL, datum = "WGS84") #note no specified origin coords

head(baboons.naRM_planar) #see that planar coordinates are reported
```

`mps` *Identify and Remove Data Points Based on Observed Movement Speed*

Description

mps (a.k.a. Meters-per-Second Filter) identifies and removes timepoints when tracked individuals were observed moving faster than a set distance threshold (representing either the great-circle distance between two points a planar distance metric, depending on whether or not lonlat == TRUE or FALSE, respectively) per second. (i.e., if it is impossible/highly unlikely that individuals moved faster than a given speed (mps), we can assume that any instances when they were observed doing so were the result of erroneous reporting, and should be removed). When running the mps filter, users have the option of setting lonlat == TRUE (by default lonlat == FALSE). lonlat is a logical argument that tells the function to calculate the distance between points on the WGS ellipsoid (if lonlat == TRUE), or on a plane (lonlat == FALSE) (see raster::pointDistance). If lonlat == TRUE, coordinates should be in degrees. Otherwise, coordinates should represent planar ('Euclidean') space (e.g. units of meters).

Usage

```
mps(x, id = NULL, point.x = NULL, point.y = NULL, dateTime = NULL,
  mpsThreshold = 10, lonlat = FALSE, parallel = FALSE,
  nCores = parallel::detectCores(), filterOutput = TRUE)
```

Arguments

`x`	List or data frame containing real-time location data that will be filtered.
`id`	Vector of length nrow(data.frame(x)) or singular character data, detailing the relevant colname in x, that denotes what unique ids for tracked individuals will be used. If argument == NULL, the function assumes a column with the colname "id" exists in x. Defaults to NULL.
`point.x`	Vector of length nrow(data.frame(x)) or singular character data, detailing the relevant colname in x, that denotes what planar-x or longitude coordinate information will be used. If argument == NULL, the function assumes a column with the colname "x" exists in x. Defaults to NULL.
`point.y`	Vector of length nrow(data.frame(x)) or singular character data, detailing the relevant colname in x, that denotes what planar-y or lattitude coordinate information will be used. If argument == NULL, the function assumes a column with the colname "y" exists in x. Defaults to NULL.
`dateTime`	Vector of length nrow(data.frame(x)) or singular character data, detailing the relevant colname in x, that denotes what dateTime information will be used. If argument == NULL, the function assumes a column with the colname "dateTime" exists in x. Defaults to NULL.
`mpsThreshold`	Numerical. Distance (in meters) representing the maximum distance individuals can realistically travel over a single second.
`lonlat`	Logical. If TRUE, point.x and point.y contain geographic coordinates (i.e., longitude and lattitude). If FALSE, point.x and point.y contain planar coordinates. Defaults to FALSE.
`parallel`	Logical. If TRUE, sub-functions within the mps wrapper will be parallelized. Note that this can significantly speed up processing of relatively small data sets, but may cause R to crash due to lack of available memory when attempting to process large datasets. Defaults to FALSE.

`nCores`	Integer. Describes the number of cores to be dedicated to parallel processes. Defaults to the maximum number of cores available (i.e., parallel::detectCores()).
`filterOutput`	Logical. If TRUE, output will be a data frame or list of data frames (depending on whether or not x is a data frame or not) containing only points that adhere to the mpsThreshold rule. If FALSE, no observartions are removed and an "mps" column is appended to x,which reports the avg distance per second individuals moved to get from observation i-1 to observation i. Defaults to TRUE.

Details

If users are not actually interested in filtering datasets, but rather determining what observations should be filtered, they may set filterOutput == FALSE. By doing so, this function will append up an "mps" column to the dataset, which reports the avg distance per second individuals moved to get from observation i-1 to observation i.

Value

If filterOutput == TRUE, returns `x` less observations representing impossible/unlikely movements.

If filterOutput == FALSE, returns `x` appended with an "mps" column which reports the avg distance per second individuals moved to get from observation i-1 to observation i.

Examples

```
data(calves) #load calves data

calves.dateTime<-datetime.append(calves, date = calves$date,
   time = calves$time) #create a dataframe with dateTime identifiers for location fixes.

calves_filter1 <- mps(calves.dateTime, id = calves.dateTime$calftag,
   point.x = calves.dateTime$x, point.y = calves.dateTime$y,
  dateTime = calves.dateTime$dateTime, mpsThreshold = 10, lonlat = FALSE, parallel = FALSE,
   filterOutput = TRUE)
```

`ntwrkEdges` *Compile List of Network Edges from a Contact Table*

Description

This function takes the output from contactDur.all or contactDur.area and generates a data frame showing the list of edges in the contact network.

Usage

```
ntwrkEdges(x, importBlocks = FALSE, removeDuplicates = TRUE)
```

Arguments

`x`	Output from the contactDur.all or contactDur.area functions. Can be either a data frame or list.
`importBlocks`	Logical. If true blocks will be carried over from x input, allowing for time-ordered and time-aggregated network creation. Defaults to FALSE.
`removeDuplicates`	Logical. If x is from contactDur.all, to-from node pairs in output will be reported twice (i.e., nodes will be listed as both a to- and a from-node). If removeDuplicates == true, duplicated edges are removed. Defaults to TRUE.

Value

Output is a data frame with the following columns, and can easily be used as input for igraph functions.

`from`	The "from" nodes in a contact network. Can also be considered "tail" nodes.
`to`	The "to" nodes in a contact network. Can also be considered "head" nodes.
`durations`	The duration of each contact reported in `x`.
`block`	(if applicable) time block during which reported contacts occurred.

Examples

```
data("calves")

calves.dateTime<-datetime.append(calves, date = calves$date,
   time = calves$time) #create a dataframe with dateTime identifiers for location fixes.

calves.agg<-tempAggregate(calves.dateTime, id = calves.dateTime$calftag,
   dateTime = calves.dateTime$dateTime, point.x = calves.dateTime$x,
   point.y = calves.dateTime$y, secondAgg = 300, extrapolate.left = FALSE,
   extrapolate.right = FALSE, resolutionLevel = "reduced", parallel = FALSE,
   na.rm = TRUE, smooth.type = 1) #smooth to 5-min fix intervals.

calves.dist<-dist2All_df(x = calves.agg, parallel = FALSE,
  dataType = "Point", lonlat = FALSE) #calculate inter-animal distances at each timepoint.

calves.contact.NOblock<-contactDur.all(x = calves.dist, dist.threshold=1,
   sec.threshold=10, blocking = FALSE, equidistant.time = FALSE,
   parallel = FALSE, reportParameters = TRUE)

calves.edges1<-ntwrkEdges(x =calves.contact.NOblock, importBlocks = FALSE,
   removeDuplicates = TRUE)

head(calves.edges1)

calves.network1 <- igraph::graph_from_data_frame(d=calves.edges1,
   directed=FALSE)

igraph::V(calves.network1)$color<- "orange1"
```

```
igraph::V(calves.network1)$size <-13
igraph::E(calves.network1)$width <- calves.edges1$duration
igraph::E(calves.network1)$color <- "black"
igraph::plot.igraph(calves.network1, vertex.label.cex=0.4,
   layout = igraph::layout.circle, main = "Inter-Calf Contacts") #plot the network
```

randomizeFeature *Randomize or Pseudorandomize Categorical Variables*

Description

This function randomizes the values in a given column (or set of columns (i.e., c(colname(x)[1], colname(x)[2]))), identified by the "feature" argument in a dataset (x).

Usage

```
randomizeFeature(x, feature = NULL, shuffle = FALSE,
  maintainDistr = TRUE, numRandomizations = 10)
```

Arguments

x	Data frame containing real-time-location data.
feature	Vector of 1 or more column names describing variables in x to be randomized.
shuffle	Logical. If TRUE, unique values will be replaced with another, random unique value from the same distribution with 100 certainty. For example if the values in a "dose" column c(0mg,100mg,300mg) were shuffled, one possible outcome would be: x$dose.shuff[which(x$dose == "0mg")] <- 300mg, x$dose.shuff[which(x$dose == "100mg")] <- 0mg, and x$dose.shuff[which(x$dose == "300mg")] <- 100mg. Defaults to FALSE.
maintainDistr	Logical. If TRUE, the number of each unique value in the column will be maintained in the function output. Otherwise, the function will draw on the initial distribution to assign randomized values, but the specific number of each unique value may not be maintained. Defaults to TRUE.
numRandomizations	Description imminent

Details

Note: the shuffle argument supercedes the maintainDistr argument. Therefore, if shuffle == TRUE, the maintainDistr argument is irrelevant.

Value

Output is x appended with columns described below.

...shuff	Randomized value of specified variables.
randomRep	Randomization replicate.

References

Farine, D.R., 2017. A guide to null models for animal social network analysis. Methods in Ecology and Evolution 8:1309-1320. https://doi.org/10.1111/2041-210X.12772.

Examples

```
data(calves)

system.time(randomizedValues<-contact::randomizeFeature(x = calves,
   feature = c("calftag", "date"), shuffle = TRUE, maintainDistr = TRUE,
   numRandomizations = 3))

randomizedFrame<-data.frame(randomizedValues[[1]])

head(randomizedFrame) #see that randomized-value columns have been appended.
```

`randomizePaths` *Randomize or Pseudorandomize Individuals' Relocation Events*

Description

Randomizes or pseudorandomizes individuals' spatial locations. Randomized datasets can later be compared to empirical ones to determine if individuals' space use differ from what would be expected at random (using the contactTest function).

Usage

```
randomizePaths(x = NULL, id = NULL, dateTime = NULL,
  point.x = NULL, point.y = NULL, poly.xy = NULL, parallel = FALSE,
  nCores = parallel::detectCores(), dataType = "Point",
  numVertices = 4, blocking = TRUE, blockUnit = "hours",
  blockLength = 1, shuffle.type = 0, shuffleUnit = "days",
  indivPaths = TRUE, numRandomizations = 1)
```

Arguments

`x`	Data frame containing real-time-location data.
`id`	Vector of length nrow(x) or singular character data, detailing the relevant colname in x, that denotes what unique ids for tracked individuals will be used. If argument == NULL, the function assumes a column with the colname "id" exists in x. Defaults to NULL.
`dateTime`	Vector of length nrow(x) or singular character data, detailing the relevant colname in x, that denotes what dateTime information will be used. If argument == NULL, the function assumes a column with the colname "dateTime" exists in x. Defaults to NULL.

`point.x`	Vector of length nrow(x) or singular character data, detailing the relevant colname in x, that denotes what planar-x or longitude coordinate information will be used. If argument == NULL, the function assumes a column with the colname "x" exists in x. Defaults to NULL.
`point.y`	Vector of length nrow(x) or singular character data, detailing the relevant colname in x, that denotes what planar-y or lattitude coordinate information will be used. If argument == NULL, the function assumes a column with the colname "y" exists in x. Defaults to NULL.
`poly.xy`	Columns within x denoting polygon xy-coordinates. Polygon coordinates must be arranged in the format of those in referencePointToPolygon output. Defaults to NULL.
`parallel`	Logical. If TRUE, sub-functions within the randomizePaths wrapper will be parallelized. Note that this can significantly speed up processing of relatively small data sets, but may cause R to crash due to lack of available memory when attempting to process large datasets. Defaults to FALSE.
`nCores`	Integer. Describes the number of cores to be dedicated to parallel processes. Defaults to the maximum number of cores available (i.e., parallel::detectCores()).
`dataType`	Character string refering to the type of real-time-location data presented in x, taking values of "Point" or "Polygon." If dataType == "Point," individuals' locations are drawn from point.x and point.y. If argument == "Polygon," individuals' locations are drawn from poly.xy. Defaults to "Point."
`numVertices`	Numerical. If dataType == "Polygon," users must specify the number of vertices contained in each polygon. Defaults to 4. Note: all polygons must contain the same number of vertices.
`blocking`	Logical. If TRUE, prior to randomization, timepoints will be categorized into a series of temporal blocks of blockLength-blockUnit length (e.g., 10 mins). After generating blocks, the spatial-location randomization methodology will follow shuffle.type. If FALSE, paths will be randomized by sampling from observed timepoints. No timepoints will be represented more than once in the randomized set. Defaults to TRUE.
`blockUnit`	Numerical. Describes the number blockUnits within each temporal block. Defaults to 1.
`blockLength`	Character string taking the values, "secs," "mins," "hours," "days," or "weeks." Describes the temporal unit associated with each block. Defaults to "hours."
`shuffle.type`	Numerical. Describes which shuffle.type is used to randomize the rand.input data set(s), given that blocking == TRUE (Note: this value is irrelevant if blocking == FALSE). Takes the values "0," "1," or "2," and defaults to 0. Descriptions of each shuffle.type value can be found under Details.
`shuffleUnit`	Character string taking the values, "secs," "mins," "hours," "days," or "weeks." Defaults to "days." Describes what temporal unit blocks will be shuffled across given shuffle.type == 2. Blocklength-units may never exceed 1 shuffleUnit (e.g., 25-hour blocks cannot be shuffled using shuffleUnit == "Days," but 1:24-hour blocks work just fine).
`indivPaths`	Logical. If TRUE, paths will be randomized with no location switching between ids (e.g., randomized xy locations for individual 1 will be generated by

	sampling only from individual 1's location distribution). If FALSE, paths will be randomized with potential location switching between ids (e.g., randomized xy locations for individual 1 will be generated by sampling from the entire dataset's location distribution). Defaults to TRUE.
`numRandomizations`	Numerical. The number of replicate data frames produced in output. Defaults to 1.

Details

Paths can be randomized, or pseudorandomized differently according to what logical arguments are set to TRUE.

Detailed shuffle.type description: If shuffle.type == 0, within-block timepoints will be randomized by sampling from observed timepoints only within the relevant block (e.g., points in block 1 may be redistributed). Block order in the data set does not change, and no timepoints will be represented more than once in the randomized set. If shuffle.type == 1, blocks are shuffled around, but points within blocks are not redistributed (e.g., block 1 <- block 5, block 3 <- block 2, block 5 <- block 4). If shuffle.type == 2, blocks are shuffled, but their relative temporal location in the shuffleUnit is maintained. For example, there are 144 10-min blocks in a 24-hour day. Say our data set contains 3 days worth of data (i.e., 432 blocks). During the randomization, because blocks maintain their relative locations in shuffleUnits (e.g., Days), block 1 in the random set will be determined by sampling from a distribution of blocks 1,145,and 289, which each representing the first block of a given shuffleUnit (e.g., Day 1, Day 2, Day 3). All blocks in the randomized set are decided in this fashion (e.g., block 2 of the randomized set is identified by sampling from a distribution of blocks 2, 146, and 290). Therefore, blocklength-units may never exceed 1 shuffleUnit (e.g., 25-hour blocks cannot be shuffled using shuffleUnit == "Days," but 1:24-hour blocks work just fine). Points within blocks are not redistributed. This particular shuffle.type (i.e., 2) is based off of the methodology described by Spiegel et al. 2016.

Note that, if shuffle.type == 2, all dateTime values in individuals movement paths described in x must be equidistant (e.g., relocations for individual i occur every 10 seconds), or erroneous dateTimes will be reported. If raw dateTime values are not equidistant, we recommend using our tempAggregate function, with na.rm == FALSE, to make it so.

Value

Output is `x` appended with columns described below.

`x.rand`	Randomized values taken from the `point.x` argument.
`y.rand`	Randomized values taken from the `point.y` argument.
`shuffle.type`	The value specified by the `shuffle.type` argument.
`rand.rep`	Randomization replicate.

If blocking == TRUE, the following codes are appended to the output data frame described above:

`block`	Integer ID describing unique blocks of time during which contacts occur.
`block.start`	The timepoint in `x` at which the `block` begins.
`block.end`	The timepoint in `x` at which the `block` ends.

`numBlocks`	Integer describing the total number of time blocks observed within `x` at which the `block`
`blockLength`	Character string describing the length of blocks described by `blockLength` and `blockUnit` arguments.

References

Spiegel, O., Leu, S.T., Sih, A., and C.M. Bull. 2016. Socially interacting or indifferent neighbors? Randomization of movement paths to tease apart social preference and spatial constraints. Methods in Ecology and Evolution 7:971-979. https://doi.org/10.1111/2041-210X.12553.

Farine, D.R., 2017. A guide to null models for animal social network analysis. Methods in Ecology and Evolution 8:1309-1320. https://doi.org/10.1111/2041-210X.12772.

Examples

```
data(calves)

calves.dateTime<-datetime.append(calves, date = calves$date,
   time = calves$time) #create a dataframe with dateTime identifiers for location fixes.

calves.agg<-tempAggregate(calves.dateTime, id = calves.dateTime$calftag,
   dateTime = calves.dateTime$dateTime, point.x = calves.dateTime$x,
   point.y = calves.dateTime$y, secondAgg = 300, extrapolate.left = FALSE,
   extrapolate.right = FALSE, resolutionLevel = "reduced", parallel = FALSE,
   na.rm = TRUE, smooth.type = 1) #smooth to 5-min fix intervals.

calves.agg.rand<-randomizePaths(x = calves.agg, id = "id",
   dateTime = "dateTime", point.x = "x", point.y = "y", poly.xy = NULL,
   parallel = FALSE, dataType = "Point", numVertices = 1, blocking = TRUE,
   blockUnit = "mins", blockLength = 10, shuffle.type = 0, shuffleUnit = NA,
   indivPaths = TRUE, numRandomizations = 1)
```

`referencePoint2Polygon` *Create a Rectangular Polygon Using Planar XY Coordinates*

Description

This function creates a square/rectangular polygon from a single reference point by translating its location multiple times using the same method used in repositionReferencePoint. For example, even though calves in our study see (data(calves2018)) were only equiped with RFID tags on their left ear. With this function, we can create polygons that account for the total space used by each individual at each time step.This function is different from similar point-to-polygon functions for two reasons: 1.) It does not assume points lie within the center of the polygon. Rather, the reference point must be a corner of the polygon (Note: "UL" denotes that reference point lies on the upper-left corner of the polygon, "UR" denotes that reference point lies on the upper-right corner of the polygon,"DL" denotes that reference point lies on the down-left corner of the polygon, "DR" denotes that reference point lies on the down-left corner of the polygon). Note that if you want the reference point to be at the center of the polygon, you can first translate the reference point to a central location on tracked individuals using repositionReferencePoint. 2.) Polygon angles/directionality are based on observed movements of tracked individuals.

Usage

```
referencePoint2Polygon(x = NULL, id = NULL, dateTime = NULL,
  point.x = NULL, point.y = NULL, direction = NULL,
  StartLocation = "UL", UpDownRepositionLen = 1,
  LeftRightRepositionLen = 1, CenterPoint = FALSE, MidPoints = FALSE,
  immobThreshold = 0, parallel = FALSE,
  nCores = parallel::detectCores(), modelOrientation = 90)
```

Arguments

`x`	Data frame or list of data frames containing real-time-location point data.
`id`	Vector of length nrow(data.frame(x)) or singular character data, detailing the relevant colname in x, that denotes what unique ids for tracked individuals will be used. If argument == NULL, the function assumes a column with the colname "id" exists in x. Defaults to NULL.
`dateTime`	Vector of length nrow(data.frame(x)) or singular character data, detailing the relevant colname in x, that denotes what dateTime information will be used. If argument == NULL, the function assumes a column with the colname "dateTime" exists in x. Defaults to NULL.
`point.x`	Vector of length nrow(data.frame(x)) or singular character data, detailing the relevant colname in x, that denotes what planar-x or longitude coordinate information will be used. If argument == NULL, the function assumes a column with the colname "x" exists in x. Defaults to NULL.
`point.y`	Vector of length nrow(data.frame(x)) or singular character data, detailing the relevant colname in x, that denotes what planar-y or lattitude coordinate information will be used. If argument == NULL, the function assumes a column with the colname "y" exists in x. Defaults to NULL.
`direction`	Numerical vector of length nrow(data.frame(x)) or singular character data detailing the relevant colname in x, that denotes what movement-direction information will be used. Observations in this vector represent the direction (in degrees) that tracked individuals moved to reach their position at each time point, NOT the direction that they will move to reach their subsequent position (i.e., values represent known orientations at each time point). Note that for the purposes of this function, observations of 0, 90, 180, and 270 degrees indicates that an individual moved straight Eastward, Northward, Westward, and Southward, respectively. If NULL, direction will be calculated using observed point-locations. Defaults to NULL.
`StartLocation`	Character string taking the values "UL," "UR," "DL," or "DR" describing where the reference point (i.e., point corresponding to xy-coordinates in the data set) lies on the rectangle that this function will delineate. Defaults to "UL."

`UpDownRepositionLen`	Numerical. Describes the height, in planar units (e.g., meters) of the output polygon. Planar units are inherent to the real-time-location input. Defaults to 1.
`LeftRightRepositionLen`	Numerical. Describes the width, in planar units (e.g., meters) of the output polygon. Planar units are inherent to the real-time-location input. Defaults to 1.
`CenterPoint`	Logical. If TRUE, in addition to the xy-coordinates for each polygon vertex, xy-coordinates for centroid of each polygon will be reported in the output. Defaults to FALSE.
`MidPoints`	Logical. If TRUE, in addition to the xy-coordinates for each polygon vertex, xy-coordinates for mid-point of each polygon edge will be reported in the output. Defaults to FALSE.
`immobThreshold`	Numerical. Describes what we call, the immobility threshold, which is a movement distance (in planar units) within which we assume individuals' physical locations and orientations remain unchanged. This immobility threshold allows us to discount observed movements so miniscule that the majority of animals' physical-space usage is likely unaffected (e.g., head shaking). Defaults to 0.
`parallel`	Logical. If TRUE, sub-functions within the referencePoint2Polygon wrapper will be parallelized. Note that this can significantly speed up processing of relatively small data sets, but may cause R to crash due to lack of available memory when attempting to process large datasets. Defaults to FALSE.
`nCores`	Integer. Describes the number of cores to be dedicated to parallel processes. Defaults to the maximum number of cores available (i.e., parallel::detectCores()).
`modelOrientation`	Numerical. Describes the relative orientation (in degrees) of a planar model (see vignette or Farthing et al. in Review (note: when this manuscript is officially published, we will update this citation/reference information)) describing vertex locations relative to tracking-device point-locations. Defaults to 90.

Details

Currently, this function only supports input data with coordinates representing planar ('Euclidean') space (e.g. units of meters).

In the output, point1.x and point1.y represent the xy coordinates from the input file. Point2-n coordinates move in a clockwise direction from point1. For example: if point1 is located on the upper left ("UL") corner of the polygon, point2 would be on the upper right corner, point3 on the bottom right, and point 4 on the bottom left.

Because this function needs information (dist, dx, dy) from 2 points on an individual's path to work, at least the first point in each individual's path will be removed (the function will report NAs for adjusted locations). Also note that if the distance between an individual's first point in their path and the second one is 0, the function will also report NAs for the second point's adjusted coordinates. The first non-NA values will only be reported for the instance where dist > 0.

Note that populating the direction argument with gyroscopic accelerometer data (or data collected using similar devices) collected concurrently with point-locations allows us to overcome a couple of assumptions associtated with using point-locations alone.

First, unless the direction argument is specifically given (i.e., direction != NULL), vertex locations in output are subject to the assumption that dt values are sufficiently small to capture individuals' orientations (i.e., individuals do not face unknown directions inbetween observed relocations). If input was previously processed using tempAggregate with resolutionLevel == "reduced," dt > secondAgg indicates that tracked individuals were missing in the original dataset for a period of time. In this case, the assumption that individuals are facing a given direction because they moved from the previous timepoint may not be accurate. Consider removing these rows (rows following one with dt > secondAgg; remember that dt indicates the time between reported xy coordinates in row i to row i + 1) from your data set.

Second, unless the direction argument is specifically given (i.e., direction != NULL), this function assumes tracked individuals are always forward-facing. This is because by observing only a single point on each individual, we cannot ascertain the true positioning of individuals' bodies. For example, even if we know a point-location moved x distance in a 90-degree direction, from this information alone we cannot determine what direction said individual was facing at the time (e.g., this could be an example of forward, bawckward, or sideward movement). However, gyroscopic data (or data collected using similar devices) can tell us absolute movement directions, as opposed to relative ones.

Value

Output is a data frame with the following columns:

`id`	Unique ID of tracked individuals.
`cornerPoint...x`	Planar x coordinates of polygon-corner vertices.
`cornerPoint...y`	Planar y coordinates of polygon-corner vertices.
`startLocation`	Describes the location of input point-locations in the vertex outputs. see `StartLocation` argument.
`upDownRepositionLength`	Describes the vertical movement of point-locations on planar models. see `UpDownRepositionLen` argument.
`leftRightRepositionLength`	Describes the horizontal movement of point-locations on planar models. see `leftRightRepositionLen` argument.
`immob`	If "0", distance between observed movements is < `immobThreshold`.
`immobThreshold`	Returns the value from the `immobThreshold` argument.
`dateTime`	Timepoint at which polygons were observed.
`dt`	The the time between reported xy coordinates in row i to row i + 1 in each individuals' movement path.

If MidPoints or CenterPoints == TRUE, additional columns will be appended to output data frame.

References

Farthing, T.S., Dawson, D.E., Sanderson, M.W., and Lanzas, C. in Review. Accounting for space and uncertainty in real-time-location- system-derived contact networks. Ecology and Evolution.

Examples

```
data("calves")
calves.dateTime<-datetime.append(calves, date = calves$date,
   time = calves$time) #add dateTime identifiers for location fixes.

calves.agg<-tempAggregate(calves.dateTime, id = calves.dateTime$calftag,
   dateTime = calves.dateTime$dateTime, point.x = calves.dateTime$x,
   point.y = calves.dateTime$y, secondAgg = 300, extrapolate.left = FALSE,
   extrapolate.right = FALSE, resolutionLevel = "reduced", parallel = FALSE,
   na.rm = TRUE, smooth.type = 1) #smooth to 5-min fix intervals.

calf_heads <- referencePoint2Polygon(x = calves.agg,
   id = calves.agg$id, dateTime = calves.agg$dateTime,
   point.x = calves.agg$x, point.y = calves.agg$y, direction = NULL,
   StartLocation = "DL", UpDownRepositionLen = 0.333, LeftRightRepositionLen = 0.333,
   CenterPoint = FALSE, MidPoints = FALSE, immobThreshold = 0.1, parallel = FALSE,
   modelOrientation = 90)
```

`repositionReferencePoint` *Move Data Point a Specified Distance*

Description

Translates locations of a single rfid tag/gps transmitter to a different location a fixed distance away, given a known angular offset (in degrees), while maintaining orientations associated with observed movements (see vignette or Farthing et al. in Review (note: when this manuscript is officially published, we will update this citation/reference information)) For example, calves in our study (see calves2018) were equiped with RFID tags on their left ear. With this function, we can move this reference point somewhere else on the body of each individual. This might be done for a number of reasons, but is very useful for use in the referencePoint2Polygon function later on (for delineating polygons representing entire individuals). Currently, this function only supports input data with coordinates representing planar ('Euclidean') space (e.g. units of meters).

Usage

```
repositionReferencePoint(x = NULL, id = NULL, dateTime = NULL,
  point.x = NULL, point.y = NULL, direction = NULL,
  repositionAngle = 0, repositionDist = 1, immobThreshold = 0,
  parallel = FALSE, nCores = parallel::detectCores(),
  modelOrientation = 90)
```

Arguments

`x`	Data frame or list of data frames containing real-time-location point data.

`id`	Vector of length nrow(data.frame(x)) or singular character data, detailing the relevant colname in x, that denotes what unique ids for tracked individuals will be used. If argument == NULL, the function assumes a column with the colname "id" exists in x. Defaults to NULL.
`dateTime`	Vector of length nrow(data.frame(x)) or singular character data, detailing the relevant colname in x, that denotes what dateTime information will be used. If argument == NULL, the function assumes a column with the colname "dateTime" exists in x. Defaults to NULL.
`point.x`	Vector of length nrow(data.frame(x)) or singular character data, detailing the relevant colname in x, that denotes what planar-x or longitude coordinate information will be used. If argument == NULL, the function assumes a column with the colname "x" exists in x. Defaults to NULL.
`point.y`	Vector of length nrow(data.frame(x)) or singular character data, detailing the relevant colname in x, that denotes what planar-y or lattitude coordinate information will be used. If argument == NULL, the function assumes a column with the colname "y" exists in x. Defaults to NULL.
`direction`	Numerical vector of length nrow(data.frame(x)) or singular character data detailing the relevant colname in x, that denotes what movement-direction information will be used. Observations in this vector represent the direction (in degrees) that tracked individuals moved to reach their position at each time point, NOT the direction that they will move to reach their subsequent position (i.e., values represent known orientations at each time point). Note that for the purposes of this function, observations of 0, 90, 180, and 270 degrees indicates that an individual moved straight Eastward, Northward, Westward, and Southward, respectively. If NULL, direction will be calculated using observed point-locations. Defaults to NULL.
`repositionAngle`	Numerical. Describes the angle (in degrees) between empirical point-locations and the desired vertex location as represented in a planar model (see vignette or Farthing et al. in Review (note: when this manuscript is officially published, we will update this citation/reference information)). Essentially, this is the direction you want new points to be from orginal points. Note that for the purposes of this function, observations of 0, 90, 180, and 270 degrees indicates that an individual moved straight Eastward, Northward, Westward, and Southward, respectively. Defaults to 0.
`repositionDist`	Numerical. Describes the distance from the empirical point-locations to desired vertex locations in planar units (e.g., meters) inherent to the real-time-location input. Defaults to 1.
`immobThreshold`	Numerical. Describes what we call, the immobility threshold, which is a movement distance (in planar units) within which we assume individuals' physical locations and orientations remain unchanged. This immobility threshold allows us to discount observed movements so miniscule that the majority of animals' physical-space usage is likely unaffected (e.g., head shaking). Defaults to 0.
`parallel`	Logical. If TRUE, sub-functions within the repositionReferencePoint wrapper will be parallelized. Note that this can significantly speed up processing of relatively small data sets, but may cause R to crash due to lack of available memory when attempting to process large datasets. Defaults to FALSE.

`nCores`	Integer. Describes the number of cores to be dedicated to parallel processes. Defaults to the maximum number of cores available (i.e., parallel::detectCores()).
`modelOrientation`	Numerical. Describes the relative orientation (in degrees) of a planar model (see vignette or Farthing et al. in Review (note: when this manuscript is officially published, we will update this citation/reference information)) describing vertex locations relative to tracking-device point-locations. Defaults to 90.

Details

In this function, if the distance individuals moved was less than/equal to the noted immobThreshold, individuals are said to be immobile ("immob"), and their position will not change relative to their previous one. (i.e., you assume that any observed movement less than immobThreshold was due to errors or miniscule bodily movements (e.g., head shaking) that are not indicative of actual movement.)

Because this function needs information (dist, dx, dy) from 2 points on an individual's path to work, at least the first point in each individual's path will be removed (the function will report NAs for adjusted locations). Also note that if the distance between an individual's first point in their path and the second one is 0, the function will also report NAs for the second point's adjusted coordinates. The first non-NA values will only be reported for the instance where dist > 0.

Note that populating the direction argument with gyroscopic accelerometer data (or data collected using similar devices) collected concurrently with point-locations allows us to overcome a couple of assumptions associated with using point-locations alone.

First, unless the direction argument is specifically given (i.e., direction != NULL), new point-locations in output are subject to the assumption that dt values are sufficiently small to capture individuals' orientations (i.e., individuals do not face unknown directions inbetween observed re-locations). If input was previously processed using tempAggregate with resolutionLevel == "reduced," dt > secondAgg indicates that tracked individuals were missing in the original dataset for a period of time. In this case, the assumption that individuals are facing a given direction because they moved from the previous timepoint may not be accurate. Consider removing these rows (rows following one with dt > secondAgg; remember that dt indicates the time between recording xy coordinates in row i to row i + 1) from your data set.

Second, unless the direction argument is specifically given (i.e., direction != NULL), this function assumes tracked individuals are always forward-facing. This is because by observing only a single point on each individual, we cannot ascertain the true positioning of individuals' bodies. For example, even if we know a point-location moved x distance in a 90-degree direction, from this information alone we cannot determine what direction said individual was facing at the time (e.g., this could be an example of forward, bawckward, or sideward movement). However, gyroscopic data (or data collected using similar devices) can tell us absolute movement directions, as opposed to relative ones.

Value

Output is a data frame with the following columns:

`id`	Unique ID of tracked individuals.
`x.original`	Original x coordinates from input.

`y.original`	Original y coordinates from input.
`distance.original`	Original planar distance (m) between point-location i to point-location i + 1.
`dx.original`	Original difference between point-location x-coordinate i to x-coordinate i + 1.
`dy.original`	Original difference between point-location y-coordinate i to y-coordinate i + 1.
`x.adjusted`	Translated x coordinates.
`y.adjusted`	Translated y coordinates.
`dist.adjusted`	Planar distance (m) between translated point-location i to translated point-location i + 1.
`dx.adjusted`	Difference between translated point-location x-coordinate i to translated x-coordinate i + 1.
`dy.adjusted`	Difference between translated point-location y-coordinate i to translated y-coordinate i + 1.
`movementDirection`	Describes the angle of movement (in degrees) required to translate point-locations to be congruent with planar-model adjustments.
`repositionAngle`	Describes the value `repositionAngle` of the argument.
`repositionDist`	Describes the value `repositionDist` of the argument.
`immob`	If "0", distance between observed movements is < `immobThreshold`.
`immobThreshold`	Returns the value from the `immobThreshold` argument.
`dateTime`	Timepoint at which polygons were observed.
`dt`	The the time between reported xy coordinates in row i to row i + 1 in each individuals' movement path.

References

Farthing, T.S., Dawson, D.E., Sanderson, M.W., and Lanzas, C. in Review. Accounting for space and uncertainty in real-time-location- system-derived contact networks. Ecology and Evolution.

Examples

```
data("calves")
calves.dateTime<-datetime.append(calves, date = calves$date,
   time = calves$time) #create a dataframe with dateTime identifiers for location fixes.

calves.agg<-tempAggregate(calves.dateTime, id = calves.dateTime$calftag,
   dateTime = calves.dateTime$dateTime, point.x = calves.dateTime$x,
   point.y = calves.dateTime$y, secondAgg = 300, extrapolate.left = FALSE,
   extrapolate.right = FALSE, resolutionLevel = "reduced", parallel = FALSE,
   na.rm = TRUE, smooth.type = 1) #smooth to 5-min fix intervals.

leftShoulder.point<-contact::repositionReferencePoint(x = calves.agg,
   id = calves.agg$id, dateTime = calves.agg$dateTime,
   point.x = calves.agg$x, point.y = calves.agg$y, direction = NULL,
   repositionAngle = 180, repositionDist = 0.0835, immobThreshold = 0, parallel = FALSE,
   modelOrientation = 90)
```

`summarizeContacts` *Summarize Contact Events*

Description

This function takes the output from contactDur.all or contactDur.area and reports the number of durations when tracked individuals are in "contact" with one another (contactDur.all) or with specified fixed points/polygons (contactDur.area).

Usage

```
summarizeContacts(x, importBlocks = FALSE, avg = FALSE)
```

Arguments

`x`	Output from the contactDur.all or contactDur.area functions. Can be either a data frame or list of data frames.
`importBlocks`	Logical. If true, each block in x will be analyzed separately. Defaults to FALSE. Note that the "block" column must exist in x.
`avg`	Logical. If TRUE, summary output from all data frames contained in x will be averaged together. Output will produce an extra data frame containing the mean column values for each id (per block if importBlocks == TRUE). Defaults to FALSE.

Details

If x is a list, and avg == TRUE, this function will produce an extra data frame containing the mean column values for each id (per block if importBlocks == TRUE).

This is a sub-function found within the contactTest and ntrkEdges function.

Value

Returns a data frame (or list of data frames if `x` is a list of data frames) with the following columns:

`id`	The unique ID of a tracked individual for which we will summarize to all other individuals/fixed locations observed in `x`.
`id`	Sum number of individuals/fixed locations observed in contact specific individuals.
`id`	Sum number of contacts associated with specific individuals.
`contactDuration_...`	Number of contacts between specific dyads.

If importBlocks == TRUE, the following columns are appended to the output data frame described above:

`block`	Integer ID describing unique blocks of time during which contacts occur.
`block.start`	The timepoint in `x` at which the `block` begins.
`block.end`	The timepoint in `x` at which the `block` ends.
`numBlocks`	Integer describing the total number of time blocks observed within `x` at which the `block`

Examples

```
data(calves)

calves.dateTime<-datetime.append(calves, date = calves$date,
   time = calves$time) #create a dataframe with dateTime identifiers for location fixes

calves.agg<-tempAggregate(calves.dateTime, id = calves.dateTime$calftag,
   dateTime = calves.dateTime$dateTime, point.x = calves.dateTime$x,
   point.y = calves.dateTime$y, secondAgg = 300, extrapolate.left = FALSE,
   extrapolate.right = FALSE, resolutionLevel = "reduced", parallel = FALSE,
   na.rm = TRUE, smooth.type = 1) #smooth to 5-min fix intervals.

calves.dist<-dist2All_df(x = calves.agg, parallel = FALSE,
   dataType = "Point", lonlat = FALSE)
calves.contact.block<-contactDur.all(x = calves.dist, dist.threshold=1,
   sec.threshold=10, blocking = TRUE, blockUnit = "hours", blockLength = 1,
   equidistant.time = FALSE, parallel = FALSE, reportParameters = TRUE)

calves.contactSumm.NOblock <- summarizeContacts(calves.contact.block)
head(calves.contactSumm.NOblock)

calves.contactSumm.block <- summarizeContacts(calves.contact.block,
   importBlocks = TRUE)
head(calves.contactSumm.block)
```

`tempAggregate` *Smooth Point-Locations Over Time*

Description

Aggregate location data by secondAgg seconds over the course of each day represented in the dataset. The function smooths xy data forwards (smooth.type == 1) or backwards (smooth.type == 2) according to a data-point-averaging loess smoothing methodology. As part of the smoothing process, tempAggregate fills in any missing values (either due to a lack of data transmission or faulty prior interpolation). We recognize that this procedure is not sensitive to individual presence at given timesteps (e.g., some individuals may be missing on certain days, hours, etc., and therefore may produce inaccurate location aggregates if days/hours exist where individuals are not present in the dataset (e.g., they were purposefully removed, or moved outside of the monitoring area)). To increase accuracy, package users may specify a resolutionLevel ("full" or "reduced") to process individuals' locations at different resolutions. If resolution == "reduced", if no locations of individuals exist over any secondAgg time block, NAs will be produced for the time blocks of interest.

This function is based on real-time-location-data-smoothing methods presented by Dawson et al. 2019.

Usage

```
tempAggregate(x = NULL, id = NULL, point.x = NULL, point.y = NULL,
  dateTime = NULL, secondAgg = 10, extrapolate.left = FALSE,
  extrapolate.right = FALSE, resolutionLevel = "full",
  parallel = FALSE, nCores = parallel::detectCores(), na.rm = TRUE,
  smooth.type = 1)
```

Arguments

`x`	Data frame or list of data frames containing real-time-location data.
`id`	Vector of length nrow(data.frame(x)) or singular character data, detailing the relevant colname in x, that denotes what unique ids for tracked individuals will be used. If argument == NULL, the function assumes a column with the colname "id" exists in x. Defaults to NULL.
`point.x`	Vector of length nrow(data.frame(x)) or singular character data, detailing the relevant colname in x, that denotes what planar-x or longitude coordinate information will be used. If argument == NULL, the function assumes a column with the colname "x" exists in x. Defaults to NULL.
`point.y`	Vector of length nrow(data.frame(x)) or singular character data, detailing the relevant colname in x, that denotes what planar-y or lattitude coordinate information will be used. If argument == NULL, the function assumes a column with the colname "y" exists in x. Defaults to NULL.
`dateTime`	Vector of length nrow(data.frame(x)) or singular character data, detailing the relevant colname in x, that denotes what dateTime information will be used. If argument == NULL, the function assumes a column with the colname "dateTime" exists in x. Defaults to NULL.
`secondAgg`	Numerical. The number of seconds over which tracked-individuals' location will be averaged. Defaults to 10.
`extrapolate.left`	Logical. If TRUE, individuals position at time points prior to their first location fix will revert to their first recorded location. If FALSE, NAs will be placed at these time points in individuals' movement paths. Defaults to FALSE.
`extrapolate.right`	Logical. If TRUE, individuals position at time points following their last location fix will revert to their final recorded location. If FALSE, NAs will be placed at these time points in individuals' movement paths. Defaults to FALSE.
`resolutionLevel`	Character string taking the value of "full" or "reduced." If "full," if no known locations of individuals exist over any secondAgg time block, xy-coordinates

	revert to the last-known values for that individual. If "reduced," if no known locations of individuals exist over any secondAgg time block, NAs will be produced for the time blocks of interest. Defaults to "full."
`parallel`	Logical. If TRUE, sub-functions within the tempAggregate wrapper will be parallelized. Note that this can significantly speed up processing of relatively small data sets, but may cause R to crash due to lack of available memory when attempting to process large datasets. Defaults to FALSE.
`nCores`	Integer. Describes the number of cores to be dedicated to parallel processes. Defaults to the maximum number of cores available (i.e., parallel::detectCores()).
`na.rm`	Logical. If TRUE, all unknown locations (i.e., xy-coordinate pairs reported as NAs) will be removed from the output. Defaults to TRUE. Note that if na.rm == FALSE, all aggregated location fixes will be temporally equidistant.
`smooth.type`	Numerical, taking the values 1 or 2. Indicates the type of smooting used to average individuals' xy-coordinates. If smooth.type == 1, data are smoothed forwards. If smooth.type == 2, data are smoothed backwards. Defaults to 1.

Value

Returns a data frame (or list of data frames if `x` is a list of data frames) with the following columns:

`id`	The unique ID of tracked individuals.
`x`	Smoothed x coordinates.
`y`	Smoothed y coordinates.
`dateTime`	Timepoint at which smoothed points were observed.

References

Dawson, D.E., Farthing, T.S., Sanderson, M.W., and Lanzas, C. 2019. Transmission on empirical dynamic contact networks is influenced by data processing decisions. Epidemics 26:32-42. https://doi.org/10.1016/j.epidem.2018.08.003/

Examples

```
data("calves")
head(calves) #observe that fix intervals occur ever 4-5 seconds.

calves.dateTime<-datetime.append(calves, date = calves$date,
   time = calves$time) #add dateTime identifiers for location fixes.

calves.agg<-tempAggregate(calves.dateTime, id = calves.dateTime$calftag,
   dateTime = calves.dateTime$dateTime, point.x = calves.dateTime$x,
   point.y = calves.dateTime$y, secondAgg = 300, extrapolate.left = FALSE,
   extrapolate.right = FALSE, resolutionLevel = "reduced", parallel = FALSE,
   na.rm = TRUE, smooth.type = 1) #smooth to 5-min fix intervals.
```

`timeBlock.append` *Append TimeBlock Information to a Data Frame*

Description

Appends "block," "block.start," "block.end," and "numBlocks" columns to an input data frame (x) with a dateTime (see dateTime.append) column. This allows users to "block" data into blockLength-blockUnit-long (e.g., 10-min-long) temporal blocks. If x == NULL, the function output will be a data frame with "dateTime" and block-related columns.

Usage

```
timeBlock.append(x = NULL, dateTime = NULL, blockLength = 10,
  blockUnit = "mins")
```

Arguments

`x`	Data frame containing dateTime information, and to which block information will be appended. if NULL, dateTime input relies solely on the dateTime argument.
`dateTime`	Vector of length nrow(x) or singular character data, detailing the relevant colname in x, that denotes what dateTime information will be used. If argument == NULL, the function assumes a column with the colname "dateTime" exists in x. Defaults to NULL.
`blockLength`	Numerical. Describes the number blockUnits within each temporal block. Defaults to 10.
`blockUnit`	Character string taking the values, "secs," "mins," "hours," "days," or "weeks." Defaults to "hours."

Details

This is a sub-function that can be found in the contactDur functions.

Value

Appends the following columns to `x`.

`block`	Integer ID describing unique blocks of time of pre-specified length.
`block.start`	The timepoint in `x` at which the `block` begins.
`block.end`	The timepoint in `x` at which the `block` ends.
`numBlocks`	Integer describing the total number of time blocks observed within `x` at which the `block`

Examples

```
data("calves")
calves.dateTime<-contact::datetime.append(calves, date = calves$date,
   time = calves$time) #add dateTime identifiers for location fixes.
calves.block<-contact::timeBlock.append(x = calves.dateTime,
    dateTime = calves.dateTime$dateTime, blockLength = 10,
    blockUnit = "mins")
head(calves.block) #see that block information has been appended.
```

Index