Package ‘spatsoc’

October 14, 2022

Title Group Animal Relocation Data by Spatial and Temporal Relationship

Version 0.1.16

Description Detects spatial and temporal groups in GPS relocations (Robitaille et al. (2020) <doi:10.1111/2041-210X.13215>). It can be used to convert GPS relocations to gambit-of-the-group format to build proximity-based social networks In addition, the randomizations function provides data-stream randomization methods suitable for GPS data.

Depends R (>= 3.4)

License GPL-3 | file LICENSE

Encoding UTF-8

LazyData true

Imports data.table (>= 1.10.5), sp, rgeos, adehabitatHR, igraph, methods

Suggests testthat (>= 2.1.0), knitr, rmarkdown, asnipe, markdown

SystemRequirements GEOS (>= 3.2.0)

RoxygenNote 7.1.1

VignetteBuilder knitr

BugReports `https://github.com/ropensci/spatsoc/issues`

URL `https://docs.ropensci.org/spatsoc/`, `https://github.com/ropensci/spatsoc`, `http://spatsoc.robitalec.ca`

NeedsCompilation no

Author Alec L. Robitaille [aut, cre] (<`https://orcid.org/0000-0002-4706-1762`>), Quinn Webber [aut] (<`https://orcid.org/0000-0002-0434-9360`>), Eric Vander Wal [aut] (<`https://orcid.org/0000-0002-8534-4317`>)

Maintainer Alec L. Robitaille <`robit.alec@gmail.com`>

Repository CRAN

Date/Publication 2021-02-24 06:50:02 UTC

R topics documented:

`build_lines` *Build Lines*

Description

`build_lines` creates a `SpatialLines` object from a `data.table`. The function accepts a `data.table` with relocation data, individual identifiers a sorting column and a `projection`. The relocation data is transformed into `SpatialLines` for each individual and optionally, each `splitBy`. Relocation data should be in two columns representing the X and Y coordinates.

Usage

```
build_lines(
  DT = NULL,
  projection = NULL,
  id = NULL,
  coords = NULL,
  sortBy = NULL,
  splitBy = NULL
)
```

Arguments

`DT`	input data.table
`projection`	character string defining the projection to be passed to `sp::CRS`. For example, for UTM zone 36S (EPSG 32736), the projection argument is ’EPSG:32736’. See details.
`id`	Character string of ID column name
`coords`	Character vector of X coordinate and Y coordinate column names

`sortBy`	Character string of date time column(s) to sort rows by. Must be a POSIXct.
`splitBy`	(optional) character string or vector of grouping column name(s) upon which the grouping will be calculated

Details

The `projection` argument expects a character string defining the EPSG code. For example, for UTM zone 36N (EPSG 32736), the projection argument is 'EPSG:32736'. See `https://spatialreference.org` for a list of EPSG codes. Please note, R spatial has followed updates to GDAL and PROJ for handling projections, see more at `https://www.r-spatial.org/r/2020/03/17/wkt.html`.

The `sortBy` is used to order the input `data.table` when creating `SpatialLines`. It must a `POSIXct` to ensure the rows are sorted by date time.

The `splitBy` argument offers further control building `SpatialLines`. If in your `DT`, you have multiple temporal groups (e.g.: years) for example, you can provide the name of the column which identifies them and build `SpatialLines` for each individual in each year.

`build_lines` is used by `group_lines` for grouping overlapping lines created from relocations.

Value

`build_lines` returns a `SpatialLines` object with a line for each individual (and optionally `splitBy` combination).

An error is returned when an individual has less than 2 relocations, making it impossible to build a line.

See Also

`group_lines`

Other Build functions: `build_polys()`

Examples

```
# Load data.table
library(data.table)

# Read example data
DT <- fread(system.file("extdata", "DT.csv", package = "spatsoc"))

# Cast the character column to POSIXct
DT[, datetime := as.POSIXct(datetime, tz = 'UTC')]

# EPSG code for example data
utm <- 'EPSG:32736'

# Build lines for each individual
lines <- build_lines(DT, projection = utm, id = 'ID', coords = c('X', 'Y'),
            sortBy = 'datetime')

# Build lines for each individual by year
DT[, yr := year(datetime)]
```

```
lines <- build_lines(DT, projection = utm, id = 'ID', coords = c('X', 'Y'),
           sortBy = 'datetime', splitBy = 'yr')
```

`build_polys` *Build Polygons*

Description

`build_polys` creates a `SpatialPolygons` object from a `data.table`. The function accepts a `data.table` with relocation data, individual identifiers, a `projection`, `hrType` and `hrParams`. The relocation data is transformed into `SpatialPolygons` for each individual and optionally, each `splitBy`. Relocation data should be in two columns representing the X and Y coordinates.

Usage

```
build_polys(
  DT = NULL,
  projection = NULL,
  hrType = NULL,
  hrParams = NULL,
  id = NULL,
  coords = NULL,
  splitBy = NULL,
  spPts = NULL
)
```

Arguments

`DT`	input data.table
`projection`	character string defining the projection to be passed to `sp::CRS`. For example, for UTM zone 36S (EPSG 32736), the projection argument is ’EPSG:32736’. See details.
`hrType`	type of HR estimation, either ’mcp’ or ’kernel’
`hrParams`	a named list of parameters for `adehabitatHR` functions
`id`	Character string of ID column name
`coords`	Character vector of X coordinate and Y coordinate column names
`splitBy`	(optional) character string or vector of grouping column name(s) upon which the grouping will be calculated
`spPts`	alternatively, provide solely a SpatialPointsDataFrame with one column representing the ID of each point.

Details

The `DT` must be a `data.table`. If your data is a `data.frame`, you can convert it by reference using `data.table::setDT`.

The `id`, `coords` (and optional `splitBy`) arguments expect the names of respective columns in `DT` which correspond to the individual identifier, X and Y coordinates, and additional grouping columns.

The `projection` argument expects a character string defining the EPSG code. For example, for UTM zone 36N (EPSG 32736), the projection argument is "EPSG:32736". See `https://spatialreference.org` for a list of EPSG codes. Please note, R spatial has followed updates to GDAL and PROJ for handling projections, see more at `https://www.r-spatial.org/r/2020/03/17/wkt.html`. It is likely that `build_polys` will return "Warning in proj4string(xy) : CRS object has comment, which is lost in output" due to these changes.

The `hrType` must be either one of "kernel" or "mcp". The `hrParams` must be a named list of arguments matching those of `adehabitatHR::kernelUD` and `adehabitatHR::getverticeshr` or `adehabitatHR::mcp`.

The `splitBy` argument offers further control building `SpatialPolygons`. If in your `DT`, you have multiple temporal groups (e.g.: years) for example, you can provide the name of the column which identifies them and build `SpatialPolygons` for each individual in each year.

`group_polys` uses `build_polys` for grouping overlapping polygons created from relocations.

Value

`build_polys` returns a `SpatialPolygons` object with a polyon for each individual (and optionally `splitBy` combination).

An error is returned when `hrParams` do not match the arguments of the `hrType` `adehabitatHR` function.

See Also

`group_polys`

Other Build functions: `build_lines()`

Examples

```
# Load data.table
library(data.table)

# Read example data
DT <- fread(system.file("extdata", "DT.csv", package = "spatsoc"))

# Cast the character column to POSIXct
DT[, datetime := as.POSIXct(datetime, tz = 'UTC')]

# EPSG code for example data
utm <- 'EPSG:32736'

# Build polygons for each individual using kernelUD and getverticeshr
```

```
build_polys(DT, projection = utm, hrType = 'kernel',
            hrParams = list(grid = 60, percent = 95),
            id = 'ID', coords = c('X', 'Y'))

# Build polygons for each individual by year
DT[, yr := year(datetime)]
build_polys(DT, projection = utm, hrType = 'mcp', hrParams = list(percent = 95),
            id = 'ID', coords = c('X', 'Y'), splitBy = 'yr')

# Build polygons from SpatialPointsDataFrame
library(sp)
pts <- SpatialPointsDataFrame(coords = DT[, .(X, Y)],
                              proj4string = CRS(utm),
                              data = DT[, .(ID)]
)

build_polys(spPts = pts, hrType = 'mcp', hrParams = list(percent = 95))
```

`DT` *Movement of 10 "Newfoundland Bog Cows"*

Description

A dataset containing the GPS relocations of 10 individuals in winter 2016-2017.

Format

A data.table with 14297 rows and 5 variables:

ID individual identifier

X X coordinate of the relocation (UTM 36N)

Y Y coordinate of the relocation (UTM 36N)

datetime character string representing the date time

population sub population within the individuals

Examples

```
# Load data.table
library(data.table)

# Read example data
DT <- fread(system.file("extdata", "DT.csv", package = "spatsoc"))
```

`dyad_id` *Dyad ID*

Description

Generate a dyad ID for edge list generated by `edge_nn` or `edge_dist`.

Usage

```
dyad_id(DT = NULL, id1 = NULL, id2 = NULL)
```

Arguments

`DT`	input data.table with columns id1 and id2, as generated by `edge_dist` or `edge_nn`
`id1`	ID1 column name generated by `edge_dist` or `edge_nn`
`id2`	ID2 column name generated by `edge_dist` or `edge_nn`

Details

An undirected edge identifier between, for example individuals A and B will be A-B (and reverse B and A will be A-B). Internally sorts and pastes id columns.

More details in the edge and dyad vignette (in progress).

Value

`dyad_id` returns the input `data.table` with appended "dyadID" column

Examples

```
# Load data.table
library(data.table)

# Read example data
DT <- fread(system.file("extdata", "DT.csv", package = "spatsoc"))

# Cast the character column to POSIXct
DT[, datetime := as.POSIXct(datetime, tz = 'UTC')]

# Temporal grouping
group_times(DT, datetime = 'datetime', threshold = '20 minutes')

# Edge list generation
edges <- edge_dist(
    DT,
    threshold = 100,
    id = 'ID',
    coords = c('X', 'Y'),
    timegroup = 'timegroup',
```

```
    returnDist = TRUE,
    fillNA = TRUE
  )

# Generate dyad IDs
dyad_id(edges, 'ID1', 'ID2')
```

`edge_dist` *Distance based edge lists*

Description

`edge_dist` returns edge lists defined by a spatial distance within the user defined threshold. The function accepts a `data.table` with relocation data, individual identifiers and a threshold argument. The threshold argument is used to specify the criteria for distance between points which defines a group. Relocation data should be in two columns representing the X and Y coordinates.

Usage

```
edge_dist(
  DT = NULL,
  threshold,
  id = NULL,
  coords = NULL,
  timegroup,
  splitBy = NULL,
  returnDist = FALSE,
  fillNA = TRUE
)
```

Arguments

`DT`	input data.table
`threshold`	distance for grouping points, in the units of the coordinates
`id`	Character string of ID column name
`coords`	Character vector of X coordinate and Y coordinate column names
`timegroup`	timegroup field in the DT upon which the grouping will be calculated
`splitBy`	(optional) character string or vector of grouping column name(s) upon which the grouping will be calculated
`returnDist`	boolean indicating if the distance between individuals should be returned. If FALSE (default), only ID1, ID2 columns (and timegroup, splitBy columns if provided) are returned. If TRUE, another column "distance" is returned indicating the distance between ID1 and ID2.
`fillNA`	boolean indicating if NAs should be returned for individuals that were not within the threshold distance of any other. If TRUE, NAs are returned. If FALSE, only edges between individuals within the threshold distance are returned.

Details

The `DT` must be a `data.table`. If your data is a `data.frame`, you can convert it by reference using `data.table::setDT`.

The `id`, `coords` (and optional `timegroup` and `splitBy`) arguments expect the names of a column in `DT` which correspond to the individual identifier, X and Y coordinates, timegroup (generated by `group_times`) and additional grouping columns.

If provided, the `threshold` must be provided in the units of the coordinates and must be larger than 0. If the `threshold` is NULL, the distance to all other individuals will be returned. The coordinates must be planar coordinates (e.g.: UTM). In the case of UTM, a `threshold` = 50 would indicate a 50m distance threshold.

The `timegroup` argument is optional, but recommended to pair with `group_times`. The intended framework is to group rows temporally with `group_times` then spatially with `edge_dist` (or grouping functions).

The `splitBy` argument offers further control over grouping. If within your `DT`, you have multiple populations, subgroups or other distinct parts, you can provide the name of the column which identifies them to `splitBy`. `edge_dist` will only consider rows within each `splitBy` subgroup.

Value

`edge_dist` returns a `data.table` with columns ID1, ID2, timegroup (if supplied) and any columns provided in splitBy. If 'returnDist' is TRUE, column 'distance' is returned indicating the distance between ID1 and ID2.

The ID1 and ID2 columns represent the edges defined by the spatial (and temporal with `group_times`) thresholds.

See Also

Other Edge-list generation: `edge_nn()`

Examples

```
# Load data.table
library(data.table)

# Read example data
DT <- fread(system.file("extdata", "DT.csv", package = "spatsoc"))

# Cast the character column to POSIXct
DT[, datetime := as.POSIXct(datetime, tz = 'UTC')]

# Temporal grouping
group_times(DT, datetime = 'datetime', threshold = '20 minutes')

# Edge list generation
edges <- edge_dist(
    DT,
    threshold = 100,
    id = 'ID',
```

```
    coords = c('X', 'Y'),
    timegroup = 'timegroup',
    returnDist = TRUE,
    fillNA = TRUE
  )
```

---

edge_nn *Nearest neighbour based edge lists*

Description

`edge_nn` returns edge lists defined by the nearest neighbour. The function accepts a `data.table` with relocation data, individual identifiers and a threshold argument. The threshold argument is used to specify the criteria for distance between points which defines a group. Relocation data should be in two columns representing the X and Y coordinates.

Usage

```
edge_nn(
  DT = NULL,
  id = NULL,
  coords = NULL,
  timegroup,
  splitBy = NULL,
  threshold = NULL,
  returnDist = FALSE
)
```

Arguments

`DT`	input data.table
`id`	Character string of ID column name
`coords`	Character vector of X coordinate and Y coordinate column names
`timegroup`	timegroup field in the DT upon which the grouping will be calculated
`splitBy`	(optional) character string or vector of grouping column name(s) upon which the grouping will be calculated
`threshold`	(optional) spatial distance threshold to set maximum distance between an individual and their neighbour.
`returnDist`	boolean indicating if the distance between individuals should be returned. If FALSE (default), only ID, NN columns (and timegroup, splitBy columns if provided) are returned. If TRUE, another column "distance" is returned indicating the distance between ID and NN.

Details

The `DT` must be a `data.table`. If your data is a `data.frame`, you can convert it by reference using `data.table::setDT`.

The `id`, `coords` (and optional `timegroup` and `splitBy`) arguments expect the names of a column in `DT` which correspond to the individual identifier, X and Y coordinates, timegroup (generated by `group_times`) and additional grouping columns.

The `threshold` must be provided in the units of the coordinates. The `threshold` must be larger than 0. The coordinates must be planar coordinates (e.g.: UTM). In the case of UTM, a `threshold` = 50 would indicate a 50m distance threshold.

The `timegroup` argument is optional, but recommended to pair with `group_times`. The intended framework is to group rows temporally with `group_times` then spatially with `edge_nn` (or grouping functions).

The `splitBy` argument offers further control over grouping. If within your `DT`, you have multiple populations, subgroups or other distinct parts, you can provide the name of the column which identifies them to `splitBy`. `edge_nn` will only consider rows within each `splitBy` subgroup.

Value

`edge_nn` returns a `data.table` with three columns: timegroup, ID and NN. If 'returnDist' is TRUE, column 'distance' is returned indicating the distance between ID and NN.

The ID and NN columns represent the edges defined by the nearest neighbours (and temporal thresholds with `group_times`).

If an individual was alone in a timegroup or splitBy, or did not have any neighbours within the threshold distance, they are assigned NA for nearest neighbour.

See Also

Other Edge-list generation: `edge_dist()`

Examples

```
# Load data.table
library(data.table)

# Read example data
DT <- fread(system.file("extdata", "DT.csv", package = "spatsoc"))

# Cast the character column to POSIXct
DT[, datetime := as.POSIXct(datetime, tz = 'UTC')]

# Temporal grouping
group_times(DT, datetime = 'datetime', threshold = '20 minutes')

# Edge list generation
edges <- edge_nn(DT, id = 'ID', coords = c('X', 'Y'),
        timegroup = 'timegroup')

# Edge list generation using maximum distance threshold
```

```
edges <- edge_nn(DT, id = 'ID', coords = c('X', 'Y'),
        timegroup = 'timegroup', threshold = 100)

# Edge list generation, returning distance between nearest neighbours
edge_nn(DT, id = 'ID', coords = c('X', 'Y'),
        timegroup = 'timegroup', threshold = 100,
        returnDist = TRUE)
```

`get_gbi` *Generate group by individual matrix*

Description

`get_gbi` generates a group by individual matrix. The function accepts a `data.table` with individual identifiers and a group column. The group by individual matrix can then be used to build a network using `asnipe::get_network`.

Usage

```
get_gbi(DT = NULL, group = "group", id = NULL)
```

Arguments

`DT`	input data.table
`group`	Character string of group column (generated from one of spatsoc's spatial grouping functions)
`id`	Character string of ID column name

Details

The `DT` must be a `data.table`. If your data is a `data.frame`, you can convert it by reference using `data.table::setDT`.

The `group` argument expects the name of a column which corresponds to an integer group identifier (generated by `spatsoc`'s grouping functions).

The `id` argument expects the name of a column which corresponds to the individual identifier.

Value

`get_gbi` returns a group by individual matrix (columns represent individuals and rows represent groups).

Note that `get_gbi` is identical in function for turning the outputs of `spatsoc` into social networks as `asnipe::get_group_by_individual` but is more efficient thanks to `data.table::dcast`.

See Also

`group_pts` `group_lines` `group_polys`

Other Social network tools: `randomizations()`

Examples

```
# Load data.table
library(data.table)

# Read example data
DT <- fread(system.file("extdata", "DT.csv", package = "spatsoc"))

# Cast the character column to POSIXct
DT[, datetime := as.POSIXct(datetime, tz = 'UTC')]
DT[, yr := year(datetime)]

# EPSG code for example data
utm <- 'EPSG:32736'

group_polys(DT, area = FALSE, hrType = 'mcp',
            hrParams = list(percent = 95),
            projection = utm, id = 'ID', coords = c('X', 'Y'),
            splitBy = 'yr')

gbiMtrx <- get_gbi(DT = DT, group = 'group', id = 'ID')
```

`group_lines` *Group Lines*

Description

`group_lines` groups rows into spatial groups by creating trajectories and grouping based on spatial overlap. The function accepts a `data.table` with relocation data, individual identifiers and a `threshold`. The relocation data is transformed into `SpatialLines` and overlapping `SpatialLines` are grouped. The `threshold` argument is used to specify the criteria for distance between lines. Relocation data should be in two columns representing the X and Y coordinates.

Usage

```
group_lines(
  DT = NULL,
  threshold = NULL,
  projection = NULL,
  id = NULL,
  coords = NULL,
  timegroup = NULL,
  sortBy = NULL,
  splitBy = NULL,
  spLines = NULL
)
```

Arguments

`DT`	input data.table
`threshold`	The width of the buffer around the lines in the units of the projection. Supply 0 to compare intersection without buffering.
`projection`	character string defining the projection to be passed to `sp::CRS`. For example, for UTM zone 36S (EPSG 32736), the projection argument is 'EPSG:32736'. See details.
`id`	Character string of ID column name
`coords`	Character vector of X coordinate and Y coordinate column names
`timegroup`	timegroup field in the DT upon which the grouping will be calculated
`sortBy`	Character string of date time column(s) to sort rows by. Must be a POSIXct.
`splitBy`	(optional) character string or vector of grouping column name(s) upon which the grouping will be calculated
`spLines`	Alternatively to providing a DT, provide a SpatialLines object created with the sp package. If a spLines object is provided, groups cannot be calculated by a timegroup or splitBy.

Details

The `DT` must be a `data.table`. If your data is a `data.frame`, you can convert it by reference using `data.table::setDT`.

The `id`, `coords`, `sortBy` (and optional `timegroup` and `splitBy`) arguments expect the names of respective columns in `DT` which correspond to the individual identifier, X and Y coordinates, sorting, timegroup (generated by `group_times`) and additional grouping columns.

The `projection` argument expects a character string defining the EPSG code. For example, for UTM zone 36N (EPSG 32736), the projection argument is "EPSG:32736". See `https://spatialreference.org` for a list of EPSG codes. Please note, R spatial has followed updates to GDAL and PROJ for handling projections, see more at `https://www.r-spatial.org/r/2020/03/17/wkt.html`. It is likely that `build_polys` will return "Warning in proj4string(xy) : CRS object has comment, which is lost in output" due to these changes.

The `sortBy` is used to order the input `data.table` when creating `SpatialLines`. It must a `POSIXct` to ensure the rows are sorted by date time.

The `threshold` must be provided in the units of the coordinates. The `threshold` can be equal to 0 if strict overlap is required, else it needs to be greater than 0. The coordinates must be planar coordinates (e.g.: UTM). In the case of UTM, a `threshold` = 50 would indicate a 50m distance threshold.

The `timegroup` argument is optional, but recommended to pair with `group_times`. The intended framework is to group rows temporally with `group_times` then spatially with `group_lines` (or `group_pts`, `group_polys`). With `group_lines`, pick a relevant `group_times threshold` such as `'1 day'` or `'7 days'` which is informed by your study species and system.

The `splitBy` argument offers further control over grouping. If within your `DT`, you have multiple populations, subgroups or other distinct parts, you can provide the name of the column which identifies them to `splitBy`. The grouping performed by `group_lines` will only consider rows within each `splitBy` subgroup.

Value

`group_lines` returns the input `DT` appended with a `group` column.

This column represents the spatial (and if `timegroup` was provided - spatiotemporal) group calculated by overlapping lines. As with the other grouping functions, the actual value of group is arbitrary and represents the identity of a given group where 1 or more individuals are assigned to a group. If the data was reordered, the group may change, but the contents of each group would not.

A message is returned when a column named `group` already exists in the input `DT`, because it will be overwritten.

See Also

`build_lines` `group_times`

Other Spatial grouping: `group_polys()`, `group_pts()`

Examples

```
# Load data.table
library(data.table)

# Read example data
DT <- fread(system.file("extdata", "DT.csv", package = "spatsoc"))

# Subset only individuals A, B, and C
DT <- DT[ID %in% c('A', 'B', 'C')]

# Cast the character column to POSIXct
DT[, datetime := as.POSIXct(datetime, tz = 'UTC')]

# EPSG code for example data
utm <- 'EPSG:32736'

group_lines(DT, threshold = 50, projection = utm, sortBy = 'datetime',
            id = 'ID', coords = c('X', 'Y'))

## Daily movement tracks
# Temporal grouping
group_times(DT, datetime = 'datetime', threshold = '1 day')

# Subset only first 50 days
DT <- DT[timegroup < 25]

# Spatial grouping
group_lines(DT, threshold = 50, projection = utm,
            id = 'ID', coords = c('X', 'Y'),
            timegroup = 'timegroup', sortBy = 'datetime')

## Daily movement tracks by population
group_lines(DT, threshold = 50, projection = utm,
            id = 'ID', coords = c('X', 'Y'),
            timegroup = 'timegroup', sortBy = 'datetime',
```

```
            splitBy = 'population')
```

`group_polys` *Group Polygons*

Description

`group_polys` groups rows into spatial groups by overlapping polygons (home ranges). The function accepts a `data.table` with relocation data, individual identifiers and an `area` argument. The relocation data is transformed into home range `SpatialPolygons`. If the `area` argument is `FALSE`, `group_polys` returns grouping calculated by overlap. If the `area` argument is `TRUE`, the area and proportion of overlap is calculated. Relocation data should be in two columns representing the X and Y coordinates.

Usage

```
group_polys(
  DT = NULL,
  area = NULL,
  hrType = NULL,
  hrParams = NULL,
  projection = NULL,
  id = NULL,
  coords = NULL,
  splitBy = NULL,
  spPolys = NULL
)
```

Arguments

`DT`	input data.table
`area`	boolean indicating either overlap group (when `FALSE`) or area and proportion of overlap (when `TRUE`)
`hrType`	type of HR estimation, either 'mcp' or 'kernel'
`hrParams`	a named list of parameters for `adehabitatHR` functions
`projection`	character string defining the projection to be passed to `sp::CRS`. For example, for UTM zone 36S (EPSG 32736), the projection argument is 'EPSG:32736'. See details.
`id`	Character string of ID column name
`coords`	Character vector of X coordinate and Y coordinate column names
`splitBy`	(optional) character string or vector of grouping column name(s) upon which the grouping will be calculated
`spPolys`	Alternatively, provide solely a SpatialPolygons object

Details

The `DT` must be a `data.table`. If your data is a `data.frame`, you can convert it by reference using `data.table::setDT`.

The `id`, `coords` (and optional `splitBy`) arguments expect the names of respective columns in `DT` which correspond to the individual identifier, X and Y coordinates, and additional grouping columns.

The `projection` argument expects a character string defining the EPSG code. For example, for UTM zone 36N (EPSG 32736), the projection argument is 'EPSG:32736'. See `https://spatialreference.org` for a list of EPSG codes. Please note, R spatial has followed updates to GDAL and PROJ for handling projections, see more at `https://www.r-spatial.org/r/2020/03/17/wkt.html`. It is likely that `build_polys` will return "Warning in proj4string(xy) : CRS object has comment, which is lost in output" due to these changes.

The `hrType` must be either one of "kernel" or "mcp". The `hrParams` must be a named list of arguments matching those of `adehabitatHR::kernelUD` or `adehabitatHR::mcp`.

The `splitBy` argument offers further control over grouping. If within your `DT`, you have multiple populations, subgroups or other distinct parts, you can provide the name of the column which identifies them to `splitBy`. The grouping performed by `group_polys` will only consider rows within each `splitBy` subgroup.

Value

When `area` is `FALSE`, `group_polys` returns the input `DT` appended with a `group` column. As with the other grouping functions, the actual value of `group` is arbitrary and represents the identity of a given group where 1 or more individuals are assigned to a group. If the data was reordered, the `group` may change, but the contents of each group would not. When `area` is `TRUE`, `group_polys` returns a proportional area overlap `data.table`. In this case, ID refers to the focal individual of which the total area is compared against the overlapping area of ID2.

If `area` is `FALSE`, a message is returned when a column named `group` already exists in the input `DT`, because it will be overwritten.

See Also

`build_polys` `group_times`

Other Spatial grouping: `group_lines()`, `group_pts()`

Examples

```
# Load data.table
library(data.table)

# Read example data
DT <- fread(system.file("extdata", "DT.csv", package = "spatsoc"))

# Cast the character column to POSIXct
DT[, datetime := as.POSIXct(datetime, tz = 'UTC')]

# EPSG code for example data
utm <- 'EPSG:32736'
```

```
group_polys(DT, area = FALSE, hrType = 'mcp',
            hrParams = list(percent = 95), projection = utm,
            id = 'ID', coords = c('X', 'Y'))

areaDT <- group_polys(DT, area = TRUE, hrType = 'mcp',
                      hrParams = list(percent = 95), projection = utm,
                      id = 'ID', coords = c('X', 'Y'))
```

`group_pts` *Group Points*

Description

`group_pts` groups rows into spatial groups. The function accepts a `data.table` with relocation data, individual identifiers and a threshold argument. The threshold argument is used to specify the criteria for distance between points which defines a group. Relocation data should be in two columns representing the X and Y coordinates.

Usage

```
group_pts(
  DT = NULL,
  threshold = NULL,
  id = NULL,
  coords = NULL,
  timegroup,
  splitBy = NULL
)
```

Arguments

`DT`	input data.table
`threshold`	distance for grouping points, in the units of the coordinates
`id`	Character string of ID column name
`coords`	Character vector of X coordinate and Y coordinate column names
`timegroup`	timegroup field in the DT upon which the grouping will be calculated
`splitBy`	(optional) character string or vector of grouping column name(s) upon which the grouping will be calculated

Details

The `DT` must be a `data.table`. If your data is a `data.frame`, you can convert it by reference using `data.table::setDT`.

The `id`, `coords` (and optional `timegroup` and `splitBy`) arguments expect the names of a column in `DT` which correspond to the individual identifier, X and Y coordinates, timegroup (generated by `group_times`) and additional grouping columns.

The `threshold` must be provided in the units of the coordinates. The `threshold` must be larger than 0. The coordinates must be planar coordinates (e.g.: UTM). In the case of UTM, a `threshold` = 50 would indicate a 50m distance threshold.

The `timegroup` argument is optional, but recommended to pair with `group_times`. The intended framework is to group rows temporally with `group_times` then spatially with `group_pts` (or `group_lines`, `group_polys`).

The `splitBy` argument offers further control over grouping. If within your `DT`, you have multiple populations, subgroups or other distinct parts, you can provide the name of the column which identifies them to `splitBy`. The grouping performed by `group_pts` will only consider rows within each `splitBy` subgroup.

Value

`group_pts` returns the input `DT` appended with a `group` column.

This column represents the spatial (and if `timegroup` was provided - spatiotemporal) group. As with the other grouping functions, the actual value of `group` is arbitrary and represents the identity of a given group where 1 or more individuals are assigned to a group. If the data was reordered, the `group` may change, but the contents of each group would not.

A message is returned when a column named `group` already exists in the input `DT`, because it will be overwritten.

See Also

`group_times`

Other Spatial grouping: `group_lines()`, `group_polys()`

Examples

```
# Load data.table
library(data.table)

# Read example data
DT <- fread(system.file("extdata", "DT.csv", package = "spatsoc"))

# Cast the character column to POSIXct
DT[, datetime := as.POSIXct(datetime, tz = 'UTC')]

# Temporal grouping
group_times(DT, datetime = 'datetime', threshold = '20 minutes')

# Spatial grouping with timegroup
group_pts(DT, threshold = 5, id = 'ID',
          coords = c('X', 'Y'), timegroup = 'timegroup')

# Spatial grouping with timegroup and splitBy on population
group_pts(DT, threshold = 5, id = 'ID', coords = c('X', 'Y'),
         timegroup = 'timegroup', splitBy = 'population')
```

`group_times` *Group Times*

Description

`group_times` groups rows into time groups. The function accepts date time formatted data and a threshold argument. The threshold argument is used to specify a time window within which rows are grouped.

Usage

```
group_times(DT = NULL, datetime = NULL, threshold = NULL)
```

Arguments

`DT`	input data.table
`datetime`	name of date time column(s). either 1 POSIXct or 2 IDate and ITime. e.g.: 'datetime' or c('idate', 'itime')
`threshold`	threshold for grouping times. e.g.: '2 hours', '10 minutes', etc. if not provided, times will be matched exactly. Note that provided threshold must be in the expected format: '## unit'

Details

The `DT` must be a `data.table`. If your data is a `data.frame`, you can convert it by reference using `data.table::setDT`.

The `datetime` argument expects the name of a column in `DT` which is of type `POSIXct` or the name of two columns in `DT` which are of type `IDate` and `ITime`.

`threshold` must be provided in units of minutes, hours or days. The character string should start with an integer followed by a unit, separated by a space. It is interpreted in terms of 24 hours which poses the following limitations:

- minutes, hours and days cannot be fractional
- minutes must divide evenly into 60
- minutes must not exceed 60
- minutes, hours which are nearer to the next day, are grouped as such
- hours must divide evenly into 24
- multi-day blocks should divide into the range of days, else the blocks may not be the same length

In addition, the `threshold` is considered a fixed window throughout the time series and the rows are grouped to the nearest interval.

If `threshold` is NULL, rows are grouped using the `datetime` column directly.

Value

`group_times` returns the input `DT` appended with a `timegroup` column and additional temporal grouping columns to help investigate, troubleshoot and interpret the timegroup.

The actual value of `timegroup` is arbitrary and represents the identity of a given `timegroup` which 1 or more individuals are assigned to. If the data was reordered, the group may change, but the contents of each group would not.

The temporal grouping columns added depend on the `threshold` provided:

- `threshold` with unit minutes: "minutes" column added identifying the nearest minute group for each row.
- `threshold` with unit hours: "hours" column added identifying the nearest hour group for each row.
- `threshold` with unit days: "block" columns added identifying the multiday block for each row.

A message is returned when any of these columns already exist in the input `DT`, because they will be overwritten.

See Also

`group_pts` `group_lines` `group_polys`

Examples

```
# Load data.table
library(data.table)

# Read example data
DT <- fread(system.file("extdata", "DT.csv", package = "spatsoc"))

# Cast the character column to POSIXct
DT[, datetime := as.POSIXct(datetime, tz = 'UTC')]

group_times(DT, datetime = 'datetime', threshold = '5 minutes')

group_times(DT, datetime = 'datetime', threshold = '2 hours')

group_times(DT, datetime = 'datetime', threshold = '10 days')
```

`randomizations` *Data-stream randomizations*

Description

`randomizations` performs data-stream social network randomization. The function accepts a `data.table` with relocation data, individual identifiers and a randomization `type`. The `data.table` is randomized either using `step` or `daily` between-individual methods, or within-individual daily `trajectory` method described by Spiegel et al. (2016).

Usage

```
randomizations(
  DT = NULL,
  type = NULL,
  id = NULL,
  group = NULL,
  coords = NULL,
  datetime = NULL,
  splitBy = NULL,
  iterations = NULL
)
```

Arguments

`DT`	input data.table
`type`	one of 'daily', 'step' or 'trajectory' - see details
`id`	Character string of ID column name
`group`	generated from spatial grouping functions - see details
`coords`	Character vector of X coordinate and Y coordinate column names
`datetime`	field used for providing date time or time group - see details
`splitBy`	List of fields in DT to split the randomization process by
`iterations`	The number of iterations to randomize

Details

The `DT` must be a `data.table`. If your data is a `data.frame`, you can convert it by reference using `data.table::setDT`.

Three randomization `types` are provided:

1. step - randomizes identities of relocations between individuals within each time step.
2. daily - randomizes identities of relocations between individuals within each day.
3. trajectory - randomizes daily trajectories within individuals (Spiegel et al. 2016).

Depending on the `type`, the `datetime` must be a certain format:

- step - datetime is integer group created by `group_times`
- daily - datetime is `POSIXct` format
- trajectory - datetime is `POSIXct` format

The `id`, `datetime`, (and optional `splitBy`) arguments expect the names of respective columns in `DT` which correspond to the individual identifier, date time, and additional grouping columns. The `coords` argument is only required when the `type` is "trajectory", since the coordinates are required for recalculating spatial groups with `group_pts`, `group_lines` or `group_polys`.

Please note that if the data extends over multiple years, a column indicating the year should be provided to the `splitBy` argument. This will ensure randomizations only occur within each year.

The `group` argument is expected only when `type` is 'step' or 'daily'.

For example, using `data.table::year`:

`DT[, yr := year(datetime)] randomizations(DT, type = 'step', id = 'ID', datetime = 'timegroup', splitBy = 'yr')`

`iterations` is set to 1 if not provided. Take caution with a large value for `iterations` with large input `DT`.

Value

`randomizations` returns the random date time or random id along with the original `DT`, depending on the randomization `type`. The length of the returned `data.table` is the original number of rows multiplied by the number of iterations + 1. For example, 3 iterations will return 4x - one observed and three randomized.

Two columns are always returned:

- observed - if the rows represent the observed (TRUE/FALSE)
- iteration - iteration of rows (where 0 is the observed)

In addition, depending on the randomization type, random ID or random date time columns are returned:

- step - `randomID` each time step
- daily - `randomID` for each day and `jul` indicating julian day
- trajectory - a random date time ("random" prefixed to `datetime` argument), observed `jul` and `randomJul` indicating the random day relocations are swapped to.

References

https://besjournals.onlinelibrary.wiley.com/doi/full/10.1111/2041-210X.12553

See Also

Other Social network tools: `get_gbi()`

Examples

```
# Load data.table
library(data.table)

# Read example data
DT <- fread(system.file("extdata", "DT.csv", package = "spatsoc"))

# Date time columns
DT[, datetime := as.POSIXct(datetime)]
DT[, yr := year(datetime)]

# Temporal grouping
group_times(DT, datetime = 'datetime', threshold = '5 minutes')

# Spatial grouping with timegroup
group_pts(DT, threshold = 5, id = 'ID', coords = c('X', 'Y'), timegroup = 'timegroup')
```

```
# Randomization: step
randStep <- randomizations(
    DT,
    type = 'step',
    id = 'ID',
    group = 'group',
    datetime = 'timegroup',
    splitBy = 'yr',
    iterations = 2
)

# Randomization: daily
randDaily <- randomizations(
    DT,
    type = 'daily',
    id = 'ID',
    group = 'group',
    datetime = 'datetime',
    splitBy = 'yr',
    iterations = 2
)

# Randomization: trajectory
randTraj <- randomizations(
    DT,
    type = 'trajectory',
    id = 'ID',
    group = NULL,
    coords = c('X', 'Y'),
    datetime = 'datetime',
    splitBy = 'yr',
    iterations = 2
)
```

`spatsoc` *spatsoc*

Description

spatsoc is an R package for detecting spatial and temporal groups in GPS relocations. It can be used to convert GPS relocations to gambit-of-the-group format to build proximity-based social networks. In addition, the randomization function provides data-stream randomization methods suitable for GPS data.

Details

The spatsoc package provides one temporal grouping function:

- `group_times`

three spatial grouping functions:

- `group_pts`
- `group_lines`
- `group_polys`

two edge list generating functions:

- `edge_dist`
- `edge_nn`

and two social network functions:

- `randomizations`
- `get_gbi`

Author(s)

Maintainer: Alec L. Robitaille `<robit.alec@gmail.com>` (ORCID)

Authors:

- Quinn Webber (ORCID)
- Eric Vander Wal (ORCID)

See Also

Useful links:

- `https://docs.ropensci.org/spatsoc/`
- `https://github.com/ropensci/spatsoc`
- `http://spatsoc.robitalec.ca`
- Report bugs at `https://github.com/ropensci/spatsoc/issues`

Index