Package: spatsoc (via r-universe)

December 9, 2024

Title Group Animal Relocation Data by Spatial and Temporal Relationship

Version 0.2.8

Description Detects spatial and temporal groups in GPS relocations (Robitaille et al. (2019) <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.

License GPL-3 | file LICENSE

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

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

Depends R (>= 3.4)

Imports adehabitatHR (>= 0.4.21), data.table (>= 1.15.0), igraph, sf, lwgeom, CircStats, stats, units

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

VignetteBuilder knitr

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

SystemRequirements GDAL (>= 2.0.1), GEOS (>= 3.4.0), PROJ (>= 4.8.0), sqlite3

Remotes r-quantities/units@889cf39

Config/pak/sysreqs libgdal-dev gdal-bin libgeos-dev libglpk-dev libxml2-dev libssl-dev libproj-dev libsqlite3-dev libudunits2-dev

Repository https://ropensci.r-universe.dev

RemoteUrl https://github.com/ropensci/spatsoc

RemoteRef main

RemoteSha f4dbe2c1c1a6c6035c8a44051afe6c1553c5ad51

Contents

`build_lines` *Build Lines*

Description

`build_lines` generates a simple feature collection with LINESTRINGs from a `data.table`. The function expects a `data.table` with relocation data, individual identifiers, a sorting column and a `projection`. The relocation data is transformed into LINESTRINGs for each individual and, optionally, combination of columns listed in `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`	numeric or character defining the coordinate reference system to be passed to sf::st_crs. For example, either `projection = "EPSG:32736"` or `projection = 32736`.
`id`	character string of ID column name
`coords`	character vector of X coordinate and Y coordinate column names. Note: the order is assumed X followed by Y 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

R-spatial evolution:

Please note, spatsoc has followed updates from R spatial, GDAL and PROJ for handling projections, see more at `https://r-spatial.org/r/2020/03/17/wkt.html`.

In addition, `build_lines` previously used sp::SpatialLines but has been updated to use sf::st_as_sf and sf::st_linestring according to the R-spatial evolution, see more at `https://r-spatial.org/r/2022/04/12/evolution.html`.

Notes on arguments:

The `projection` argument expects a numeric or character defining the coordinate reference system. For example, for UTM zone 36N (EPSG 32736), the projection argument is either `projection = 'EPSG:32736'` or `projection = 32736`. See details in `sf::st_crs()` and `https://spatialreference.org` for a list of EPSG codes.

The `sortBy` argument is used to order the input `DT` when creating sf LINESTRINGs. It must a column in the input `DT` of type POSIXct to ensure the rows are sorted by date time.

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

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

Value

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

Individuals (or combinations of individuals and `splitBy`) with less than two relocations are dropped since it requires at least two relocations 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 <- 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 generates a simple feature collection with POLYGONs from a data.table. The function expects a data.table with relocation data, individual identifiers, a projection, home range type and parameters. The relocation data is transformed into POLYGONs using either adehabitatHR::mcp or adehabitatHR::kernelUD for each individual and, optionally, combination of columns listed in 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`	numeric or character defining the coordinate reference system to be passed to sf::st_crs. For example, either `projection = "EPSG:32736"` or `projection = 32736`.
`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. Note: the order is assumed X followed by Y 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, as specified by adehabitatHR::mcp or adehabitatHR::kernelUD

Details

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

R-spatial evolution:

Please note, spatsoc has followed updates from R spatial, GDAL and PROJ for handling projections, see more below and details at `https://r-spatial.org/r/2020/03/17/wkt.html`.

In addition, `build_polys` previously used sp::SpatialPoints but has been updated to use sf::st_as_sf according to the R-spatial evolution, see more at `https://r-spatial.org/r/2022/04/12/evolution.html`.

Notes on arguments:

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 or numeric defining the coordinate reference system to be passed to sf::st_crs. For example, for UTM zone 36S (EPSG 32736), the projection argument is `projection = "EPSG:32736"` or `projection = 32736`. See `https://spatialreference.org` for a list of EPSG codes.

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 POLYGONs. 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 POLYGONs for each individual in each year.

Value

build_polys returns a simple feature collection with POLYGONs for each individual (and optionally splitBy combination).

An error is returned when hrParams do not match the arguments of the respective 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 <- 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')
```

---

centroid_dyad *Dyad centroid*

Description

`centroid_dyad` calculates the centroid (mean location) of a dyad in each observation identified by `edge_nn` or `edge_dist`. The function accepts an edge list generated by `edge_nn` or `edge_dist` and a `data.table` with relocation data appended with a `timegroup` column from `group_times`. It is recommended to use the argument `fillNA = FALSE` for `edge_dist` when using `centroid_dyad` to avoid unnecessarily merging additional rows. Relocation data should be in two columns representing the X and Y coordinates.

Usage

```
centroid_dyad(
  edges = NULL,
  DT = NULL,
  id = NULL,
  coords = NULL,
  timegroup = "timegroup",
  na.rm = FALSE
)
```

Arguments

`edges`	edge list generated generated by `edge_dist` or `edge_nn`, with dyad ID column generated by `dyad_id`
`DT`	input data.table with timegroup column generated with `group_times` matching the input data.table used to generate the edge list with `edge_nn` or `edge_dist`
`id`	character string of ID column name
`coords`	character vector of X coordinate and Y coordinate column names. Note: the order is assumed X followed by Y column names.
`timegroup`	timegroup field in the DT within which the grouping will be calculated
`na.rm`	if NAs should be removed in calculating mean location, see `rowMeans`

Details

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

The `edges` and `DT` are internally merged in this function using the columns `id`, `dyadID` and `timegroup`. This function expects a `dyadID` present, generated with the `dyad_id` function. The `dyadID` and `timegroup` arguments expect the names of a column in `edges` which correspond to the dyadID and timegroup columns. The `id` and `timegroup` arguments expect the names of a column in `DT` which correspond to the X and Y coordinates and group columns. The `na.rm` argument is passed to the `rowMeans` function to control if NA values are removed before calculation.

Value

`centroid_dyad` returns the input `edges` appended with centroid columns for the X and Y coordinate columns.

These columns represents the centroid coordinate columns for the dyad. The naming of these columns will correspond to the provided coordinate column names prefixed with "centroid_".

Note: due to the merge required within this function, the output needs to be reassigned unlike some other `spatsoc` functions like `dyad_id` and `group_pts`.

A message is returned when centroid columns are already exists in the input `edges`, because they will be overwritten.

See Also

dyad_id edge_dist edge_nn group_pts

Other Centroid functions: `centroid_fusion()`, `centroid_group()`

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 = FALSE
  )

# Generate dyad id
dyad_id(edges, id1 = 'ID1', id2 = 'ID2')

# Calculate dyad centroid
centroids <- centroid_dyad(
  edges,
  DT,
  id = 'ID',
  coords = c('X', 'Y'),
  timegroup = 'timegroup', na.rm = TRUE
)

print(centroids)
```

`centroid_fusion` *Fusion centroid*

Description

`centroid_fusion` calculates the centroid (mean location) of each timestep in fusion events. The function accepts an edge list of fusion events identified by `fusion_id` from edge lists generated with `edge_dist` and a `data.table` with relocation data appended with a `timegroup` column from `group_times`. It is recommended to use the argument `fillNA = FALSE` for `edge_dist` when using `centroid_fusion` to avoid unnecessarily merging additional rows. Relocation data should be in two columns representing the X and Y coordinates.

Usage

```
centroid_fusion(
  edges = NULL,
  DT = NULL,
  id = NULL,
  coords = NULL,
  timegroup = "timegroup",
  na.rm = FALSE
)
```

Arguments

`edges`	edge list generated generated by `edge_dist` or `edge_nn`, with fusionID column generated by `fusion_id`
`DT`	input data.table with timegroup column generated with `group_times` matching the input data.table used to generate the edge list with `edge_nn` or `edge_dist`
`id`	character string of ID column name
`coords`	character vector of X coordinate and Y coordinate column names. Note: the order is assumed X followed by Y column names.
`timegroup`	timegroup field in the DT within which the grouping will be calculated
`na.rm`	if NAs should be removed in calculating mean location, see `rowMeans`

Details

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

The `edges` and `DT` are internally merged in this function using the columns `timegroup` (from `group_times`) and `ID1` and `ID2` (in `edges`, from `dyad_id`) and `id` (in `DT`). This function expects a `fusionID` present, generated with the `fusion_id` function. The `timegroup` argument expects the names of a column in `edges` which correspond to the timegroup column. The `id`, `coords` and `timegroup` arguments expect the names of a column in `DT` which correspond to the id, X and Y coordinates and timegroup columns. The `na.rm` argument is passed to the `rowMeans` function to control if NA values are removed before calculation.

Value

`centroid_fusion` returns the input `edges` appended with centroid columns for the X and Y coordinate columns.

These columns represents the centroid coordinate columns for each timestep in a fusion event. The naming of these columns will correspond to the provided coordinate column names prefixed with "centroid_".

Note: due to the merge required within this function, the output needs to be reassigned unlike some other `spatsoc` functions like `fusion_id` and `group_pts`.

A message is returned when centroid columns are already exists in the input `edges`, because they will be overwritten.

See Also

fusion_id edge_dist group_pts

Other Centroid functions: `centroid_dyad()`, `centroid_group()`

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 = FALSE
  )

# Generate dyad id
dyad_id(edges, id1 = 'ID1', id2 = 'ID2')

# Generate fusion id
fusion_id(edges, threshold = 100)

# Calculate fusion centroid
centroids <- centroid_fusion(
  edges,
  DT,
  id = 'ID',
  coords = c('X', 'Y'),
  timegroup = 'timegroup', na.rm = TRUE
)

print(centroids)
```

`centroid_group` *Group centroid*

Description

`centroid_group` calculates the centroid (mean location) of all individuals in each spatiotemporal group identified by `group_pts`. The function accepts a `data.table` with relocation data appended with a `group` column from `group_pts`. Relocation data should be in two columns representing the X and Y coordinates.

Usage

```
centroid_group(DT = NULL, coords = NULL, group = "group", na.rm = FALSE)
```

Arguments

`DT`	input data.table with group column generated with `group_pts`
`coords`	character vector of X coordinate and Y coordinate column names. Note: the order is assumed X followed by Y column names.
`group`	Character string of group column
`na.rm`	if NAs should be removed in calculating mean location, see `mean`

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` or by reassigning using `data.table::data.table`.

The `coords` and `group` arguments expect the names of a column in `DT` which correspond to the X and Y coordinates and group columns. The `na.rm` argument is passed to the `mean` function to control if NA values are removed before calculation.

Value

`centroid_group` returns the input `DT` appended with centroid columns for the X and Y coordinate columns.

These columns represents the centroid coordinate columns. The naming of these columns will correspond to the provided coordinate column names prefixed with "centroid_".

A message is returned when centroid columns are already exists in the input `DT`, because they will be overwritten.

See Also

group_pts

Other Centroid functions: centroid_dyad(), centroid_fusion()

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')

# Calculate group centroid
centroid_group(DT, coords = c('X', 'Y'), group = 'group', na.rm = TRUE)
```

---

diff_rad *Difference of two angles measured in radians*

Description

Internal function

Usage

```
diff_rad(x, y, signed = FALSE, return_units = FALSE)
```

Arguments

x	angle in radians
y	angle in radians
signed	boolean if signed difference should be returned, default FALSE
return_units	return difference with units = 'rad'

Value

Difference between x and y in radians. If signed is TRUE, the signed difference is returned. If signed is FALSE, the absolute difference is returned. Note: The difference is the smallest difference, eg.

References

adapted from https://stackoverflow.com/a/7869457

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')]

# Set order using data.table::setorder
setorder(DT, datetime)

# Calculate direction
direction_step(
  DT = DT,
  id = 'ID',
  coords = c('X', 'Y'),
  projection = 32736
)

# Differences
spatsoc:::diff_rad(DT[1, direction], DT[2, direction])
```

`direction_group` *Group mean direction*

Description

`direction_group` calculates the mean direction of all individuals in each spatiotemporal group identified by `group_pts`. The function accepts a `data.table` with relocation data appended with a `direction` column from `direction_step` and a group column from `group_pts`.

Usage

```
direction_group(DT, direction = "direction", group = "group")
```

Arguments

`DT`	input data.table with direction column generated by `direction_step` and group column generated with `group_pts`
`direction`	character string of direction column name, default "direction"
`group`	character string of group column name, default "group"

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 or by reassigning using data.table::data.table.

The direction and group arguments expect the names of columns in DT which correspond to the direction and group columns. The direction column is expected in units of radians and the mean calculated with CircStats::circ.mean().

Value

direction_group returns the input DT appended with a group_direction column representing the mean direction of all individuals in each spatiotemporal group.

The mean direction is calculated using CircStats::circ.mean() which expects units of radians.

A message is returned when the group_direction columns already exists in the input DT, because it will be overwritten.

References

See examples of using mean group direction:

- https://doi.org/10.1098/rsos.170148
- https://doi.org/10.1098/rsos.201128
- https://doi.org/10.1016/j.beproc.2018.01.013

See Also

direction_step, group_pts, CircStats::circ.mean()

Other Direction functions: direction_polarization(), direction_step(), direction_to_leader(), edge_delay()

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 = 50, id = 'ID',
          coords = c('X', 'Y'), timegroup = 'timegroup')

# Calculate direction at each step
```

```
direction_step(
  DT = DT,
  id = 'ID',
  coords = c('X', 'Y'),
  projection = 32736
)

# Calculate group direction
direction_group(DT)
```

`direction_polarization` *Polarization*

Description

`direction_polarization` calculates the polarization of individual directions in each spatiotemporal group identified by `group_pts`. The function expects a `data.table` with relocation data appended with a `direction` column from `direction_step` and a `group` column from `group_pts`.

Usage

```
direction_polarization(DT, direction = "direction", group = "group")
```

Arguments

`DT`	input data.table with direction column generated by `direction_step` and group column generated with `group_pts`
`direction`	character string of direction column name, default "direction"
`group`	character string of group column name, default "group"

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` or by reassigning using `data.table::data.table`.

The `direction` and `group` arguments expect the names of columns in `DT` which correspond to the direction and group columns. The direction column is expected in units of radians and the polarization is calculated with `CircStats::r.test()`.

Value

`direction_polarization` returns the input `DT` appended with a `polarization` column representing the direction polarization of all individuals in each spatiotemporal group.

The direction polarization is calculated using `CircStats::r.test()` which expects units of radians.

A message is returned when the `polarization` columns already exists in the input `DT`, because it will be overwritten.

References

See examples of using polarization:

- https://doi.org/10.1016/j.cub.2017.08.004
- https://doi.org/10.1371/journal.pcbi.1009437
- https://doi.org/10.7554/eLife.19505

See Also

direction_step, group_pts, CircStats::r.test()

Other Direction functions: direction_group(), direction_step(), direction_to_leader(), edge_delay()

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 = 50, id = 'ID',
          coords = c('X', 'Y'), timegroup = 'timegroup')

# Calculate direction at each step
direction_step(
  DT = DT,
  id = 'ID',
  coords = c('X', 'Y'),
  projection = 32736
)

# Calculate polarization
direction_polarization(DT)
```

---

direction_step *Calculate direction at each step*

Description

`direction_step` calculates the direction of movement steps in radians. The function accepts a `data.table` with relocation data and individual identifiers. Relocation data should be in two columns representing the X and Y coordinates. Note the order of rows is not modified by this function and therefore users must be cautious to set it explicitly. See example for one approach to setting order of rows using a datetime field.

Usage

```
direction_step(
  DT = NULL,
  id = NULL,
  coords = NULL,
  projection = NULL,
  splitBy = NULL
)
```

Arguments

`DT`	input data.table
`id`	character string of ID column name
`coords`	character vector of X coordinate and Y coordinate column names. Note: the order is assumed X followed by Y column names.
`projection`	numeric or character defining the coordinate reference system to be passed to sf::st_crs. For example, either `projection = "EPSG:32736"` or `projection = 32736`.
`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` or by reassigning using `data.table::data.table`.

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

The `projection` argument expects a character string or numeric defining the coordinate reference system to be passed to sf::st_crs. For example, for UTM zone 36S (EPSG 32736), the projection argument is `projection = "EPSG:32736"` or `projection = 32736`. See `https://spatialreference.org` for #' a list of EPSG codes.

The `splitBy` argument offers further control over grouping. If within your `DT`, you have distinct sampling periods for each individual, you can provide the column name(s) which identify them to `splitBy`. The direction calculation by `direction_step` will only consider rows within each `id` and `splitBy` subgroup.

Value

`direction_step` returns the input `DT` appended with a direction column with units set to radians using the `units` package.

This column represents the azimuth between the sequence of points for each individual computed using `lwgeom::st_geod_azimuth`. Note, the order of points is not modified by this function and therefore it is crucial the user sets the order of rows to their specific question before using `direction_step`. In addition, the direction column will include an `NA` value for the last point in each sequence of points since there is no future point to calculate a direction to.

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

See Also

`amt::direction_abs()`, `geosphere::bearing()`

Other Direction functions: `direction_group()`, `direction_polarization()`, `direction_to_leader()`, `edge_delay()`

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')]

# Set order using data.table::setorder
setorder(DT, datetime)

# Calculate direction
direction_step(
  DT = DT,
  id = 'ID',
  coords = c('X', 'Y'),
  projection = 32736
)

# Example result for East, North, West, South steps
example <- data.table(
  X = c(0, 5, 5, 0, 0),
  Y = c(0, 0, 5, 5, 0),
  step = c('E', 'N', 'W', 'S', NA),
  ID = 'A'
)

direction_step(example, 'ID', c('X', 'Y'), projection = 4326)
example[, .(direction, units::set_units(direction, 'degree'))]
```

`direction_to_centroid` *Direction to group centroid*

Description

`direction_to_centroid` calculates the direction of each relocation to the centroid of the spatiotemporal group identified by `group_pts`. The function accepts a `data.table` with relocation data appended with a `group` column from `group_pts` and centroid columns from `centroid_group`. Relocation data should be in planar coordinates provided in two columns representing the X and Y coordinates.

Usage

```
direction_to_centroid(DT = NULL, coords = NULL)
```

Arguments

`DT`	input data.table with centroid columns generated by eg. `centroid_group`
`coords`	character vector of X coordinate and Y coordinate column names. Note: the order is assumed X followed by Y column names.

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` or by reassigning using `data.table::data.table`.

This function expects a `group` column present generated with the `group_pts` function and centroid coordinate columns generated with the `centroid_group` function. The `coords` and `group` arguments expect the names of columns in `DT` which correspond to the X and Y coordinates and group columns.

Value

`direction_to_centroid` returns the input `DT` appended with a `direction_centroid` column indicating the direction to group centroid in radians. The direction is measured in radians in the range of 0 to 2 * pi from the positive x-axis.

A message is returned when `direction_centroid` column already exist in the input `DT`, because they will be overwritten.

References

See example of using direction to group centroid:

- `https://doi.org/10.1016/j.cub.2017.08.004`

See Also

centroid_group, group_pts

Other Distance functions: `distance_to_centroid()`, `distance_to_leader()`

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')

# Calculate group centroid
centroid_group(DT, coords = c('X', 'Y'), group = 'group', na.rm = TRUE)

# Calculate direction to group centroid
direction_to_centroid(DT, coords = c('X', 'Y'))
```

`direction_to_leader` *Direction to group leader*

Description

`direction_to_leader` calculates the direction to the leader of each spatiotemporal group. The function accepts a `data.table` with relocation data appended with a `rank_position_group_direction` column indicating the ranked position along the group direction generated with `leader_direction_group(return_rank = TRUE)`. Relocation data should be in planar coordinates provided in two columns representing the X and Y coordinates.

Usage

```
direction_to_leader(DT = NULL, coords = NULL, group = "group")
```

Arguments

`DT`	input data.table with 'rank_position_group_direction' column generated by `leader_direction_group` and group column generated by `group_pts`
`coords`	character vector of X coordinate and Y coordinate column names. Note: the order is assumed X followed by Y column names.
`group`	group column name, generated by `group_pts`, default 'group'

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 or by reassigning using data.table::data.table.

This function expects a rank_position_group_direction column generated with leader_direction_group(return_rar = TRUE), a group column generated with the group_pts function. The coords and group arguments expect the names of columns in DT which correspond to the X and Y coordinates and group columns.

Value

direction_to_leader returns the input DT appended with a direction_leader column indicating the direction to the group leader.

A message is returned when the direction_leader column is already exist in the input DT because it will be overwritten.

References

See examples of using direction to leader and position within group:

- https://doi.org/10.1016/j.anbehav.2023.09.009
- https://doi.org/10.1016/j.beproc.2013.10.007
- https://doi.org/10.1371/journal.pone.0036567

See Also

distance_to_leader, leader_direction_group, group_pts

Other Direction functions: direction_group(), direction_polarization(), direction_step(), edge_delay()

Examples

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


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

# (Subset example data to reduce example run time)
DT <- DT[year(datetime) == 2016]

# 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 = 50, id = 'ID',
          coords = c('X', 'Y'), timegroup = 'timegroup')

# Calculate direction at each step
direction_step(
  DT = DT,
  id = 'ID',
  coords = c('X', 'Y'),
  projection = 32736
)

# Calculate group centroid
centroid_group(DT, coords = c('X', 'Y'))

# Calculate group direction
direction_group(DT)

# Calculate leader in terms of position along group direction
leader_direction_group(
  DT,
  coords = c('X', 'Y'),
  return_rank = TRUE
)

# Calculate direction to leader
direction_to_leader(DT, coords = c('X', 'Y'))
```

`distance_to_centroid` *Distance to group centroid*

Description

`distance_to_centroid` calculates the distance of each relocation to the centroid of the spatiotemporal group identified by `group_pts`. The function accepts a `data.table` with relocation data appended with a `group` column from `group_pts` and centroid columns from `centroid_group`. Relocation data should be in planar coordinates provided in two columns representing the X and Y coordinates.

Usage

```
distance_to_centroid(
  DT = NULL,
  coords = NULL,
  group = "group",
  return_rank = FALSE,
  ties.method = NULL
)
```

Arguments

`DT`	input data.table with centroid columns generated by eg. `centroid_group`
`coords`	character vector of X coordinate and Y coordinate column names. Note: the order is assumed X followed by Y column names.
`group`	group column name, generated by `group_pts`, default 'group'
`return_rank`	boolean if rank distance should also be returned, default FALSE
`ties.method`	see `?data.table::frank`

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` or by reassigning using `data.table::data.table`.

This function expects a `group` column present generated with the `group_pts` function and centroid coordinate columns generated with the `centroid_group` function. The `coords` and `group` arguments expect the names of columns in `DT` which correspond to the X and Y coordinates and group columns. The `return_rank` argument controls if the rank of each individual's distance to the group centroid is also returned. The `ties.method` argument is passed to `data.table::frank`, see details at `?data.table::frank`.

Value

`distance_to_centroid` returns the input `DT` appended with a `distance_centroid` column indicating the distance to group centroid and, optionally, a `rank_distance_centroid` column indicating the within group rank distance to group centroid (if `return_rank` = `TRUE`).

A message is returned when `distance_centroid` and optional `rank_distance_centroid` columns already exist in the input `DT`, because they will be overwritten.

References

See examples of using distance to group centroid:

- `https://doi.org/10.1016/j.anbehav.2021.08.004`
- `https://doi.org/10.1111/eth.12336`
- `https://doi.org/10.1007/s13364-018-0400-2`

See Also

centroid_group, group_pts

Other Distance functions: `direction_to_centroid()`, `distance_to_leader()`

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')

# Calculate group centroid
centroid_group(DT, coords = c('X', 'Y'), group = 'group', na.rm = TRUE)

# Calculate distance to group centroid
distance_to_centroid(
  DT,
  coords = c('X', 'Y'),
  group = 'group',
  return_rank = TRUE
)
```

distance_to_leader *Distance to group leader*

Description

`distance_to_leader` calculates the distance to the leader of each spatiotemporal group. The function accepts a `data.table` with relocation data appended with a `rank_position_group_direction` column indicating the ranked position along the group direction generated with `leader_direction_group(return_rank = TRUE)`. Relocation data should be in planar coordinates provided in two columns representing the X and Y coordinates.

Usage

```
distance_to_leader(DT = NULL, coords = NULL, group = "group")
```

Arguments

`DT`	input data.table with 'rank_position_group_direction' column generated by `leader_direction_group` and group column generated by `group_pts`
`coords`	character vector of X coordinate and Y coordinate column names. Note: the order is assumed X followed by Y column names.
`group`	group column name, generated by `group_pts`, default 'group'

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` or by reassigning using `data.table::data.table`.

This function expects a `rank_position_group_direction` column generated with `leader_direction_group(return_rar`
`= TRUE)`, a `group` column generated with the `group_pts` function. The `coords` and `group` arguments expect the names of columns in `DT` which correspond to the X and Y coordinates and group columns.

Value

`distance_to_leader` returns the input `DT` appended with a `distance_leader` column indicating the distance to the group leader.

A message is returned when the `distance_leader` column is already exist in the input `DT` because it will be overwritten.

References

See examples of using distance to leader and position within group:

- `https://doi.org/10.1111/jfb.15315`
- `https://doi.org/10.1098/rspb.2017.2629`
- `https://doi.org/10.1016/j.anbehav.2023.09.009`

See Also

direction_to_leader, leader_direction_group, group_pts

Other Distance functions: `direction_to_centroid`(), `distance_to_centroid`()

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')]

# (Subset example data to reduce example run time)
DT <- DT[year(datetime) == 2016]

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

# Spatial grouping with timegroup
group_pts(DT, threshold = 50, id = 'ID',
          coords = c('X', 'Y'), timegroup = 'timegroup')
```

```
# Calculate direction at each step
direction_step(
  DT = DT,
  id = 'ID',
  coords = c('X', 'Y'),
  projection = 32736
)

# Calculate group centroid
centroid_group(DT, coords = c('X', 'Y'))

# Calculate group direction
direction_group(DT)

# Calculate leader in terms of position along group direction
leader_direction_group(
  DT,
  coords = c('X', 'Y'),
  return_rank = TRUE
)

# Calculate distance to leader
distance_to_leader(DT, coords = c('X', 'Y'))
```

`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_delay *Directional correlation delay based edge lists*

Description

edge_delay returns edge lists defined by the directional correlation delay between individuals. The function expects a distance based edge list generated by edge_dist or edge_nn, a data.table with relocation data, individual identifiers and a window argument. The window argument is used to specify the temporal window within which to measure the directional correlation delay. Relocation data should be in two columns representing the X and Y coordinates.

Usage

```
edge_delay(edges, DT, window = NULL, id = NULL, direction = "direction")
```

Arguments

edges	edge list generated generated by edge_dist or edge_nn, with fusionID column generated by fusion_id
DT	input data.table with timegroup column generated with group_times matching the input data.table used to generate the edge list with edge_nn or edge_dist
window	temporal window in unit of timegroup column generated with group_times, eg. window = 4 corresponds to the 4 timegroups before and after the focal observation
id	character string of ID column name
direction	character string of direction column name, default "direction"

Details

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

The edges argument expects a distance based edge list generated with edge_nn or edge_dist. The DT argument expects relocation data with a timegroup column generated with group_times.

The rows in edges and DT are internally matched in edge_delay using the columns timegroup (from group_times) and ID1 and ID2 (in edges, from dyad_id) with id (in DT). This function expects a fusionID present, generated with the fusion_id function, and a dyadID present, generated with the dyad_id function. The id, and direction arguments expect the names of a column in DT which correspond to the id, and direction columns.

Value

edge_delay returns the input edges appended with a 'dir_corr_delay' column indicating the temporal delay (in units of timegroups) at which ID1's direction of movement is most similar to ID2's direction of movement, within the temporal window defined. For example, if focal individual 'A' moves in a 45 degree direction at time 2 and individual 'B' moves in a most similar direction within the window at time 5, the directional correlation delay between A and B is 3. Positive values of directional correlation delay indicate a directed leadership edge from ID1 to ID2.

References

The directional correlation delay is defined in Nagy et al. 2010 (https://doi.org/10.1038/nature08891).

See examples of measuring the directional correlation delay:

- https://doi.org/10.1016/j.anbehav.2013.07.005
- https://doi.org/10.1073/pnas.1305552110
- https://doi.org/10.1111/jfb.15315
- https://doi.org/10.1371/journal.pcbi.1003446

See Also

Other Edge-list generation: edge_dist(), edge_nn()

Other Direction functions: direction_group(), direction_polarization(), direction_step(), direction_to_leader()

Examples

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


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

# Select only individuals A, B, C for this example
DT <- DT[ID %in% c('A', 'B', 'C')]

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

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

# Calculate direction
direction_step(
  DT = DT,
  id = 'ID',
  coords = c('X', 'Y'),
  projection = 32736
)

# Distance based edge list generation
edges <- edge_dist(
  DT,
  threshold = 100,
  id = 'ID',
  coords = c('X', 'Y'),
  timegroup = 'timegroup',
  returnDist = TRUE,
  fillNA = FALSE
)

# Generate dyad id
dyad_id(edges, id1 = 'ID1', id2 = 'ID2')

# Generate fusion id
fusion_id(edges, threshold = 100)

# Directional correlation delay
delay <- edge_delay(
  edges = edges,
  DT = DT,
  window = 3,
  id = 'ID'
)

delay[, mean(dir_corr_delay, na.rm = TRUE), by = .(ID1, ID2)][V1 > 0]
```

`edge_dist` *Distance based edge lists*

Description

`edge_dist` returns edge lists defined by a spatial distance within the user defined threshold. The function expects 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. Note: the order is assumed X followed by Y column names.
`timegroup`	timegroup field in the DT within 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` `timegroup` (and optional `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 required to define the temporal groups within which edges are calculated. The intended framework is to group rows temporally with `group_times` then spatially with `edge_dist`. If you have already calculated temporal groups without `group_times`, you can pass this column to the `timegroup` argument. Note that the expectation is that each individual will be observed only once per timegroup. Caution that accidentally including huge numbers of rows within timegroups can overload your machine since all pairwise distances are calculated within each timegroup.

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_delay(), 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 expects 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. Note: the order is assumed X followed by Y column names.
`timegroup`	timegroup field in the DT within 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`, `timegroup` (and optional `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 required to define the temporal groups within which edge nearest neighbours are calculated. The intended framework is to group rows temporally with `group_times` then spatially with `edge_nn`. If you have already calculated temporal groups without `group_times`, you can pass this column to the `timegroup` argument. Note that the expectation is that each individual will be observed only once per timegroup. Caution that accidentally including huge numbers of rows within timegroups can overload your machine since all pairwise distances are calculated within each timegroup.

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_delay()`, `edge_dist()`

Examples

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


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

# Select only individuals A, B, C for this example
DT <- DT[ID %in% c('A', 'B', 'C')]

# 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)
```

fusion_id *Fission-fusion events*

Description

`fusion_id` identifies fusion events in distance based edge lists. The function accepts a distance based edge list generated by `edge_dist`, a threshold argument and arguments controlling how fusion events are defined.

Usage

```
fusion_id(
  edges = NULL,
  threshold = 50,
  n_min_length = 0,
  n_max_missing = 0,
  allow_split = FALSE
)
```

Arguments

`edges`	distance based edge list generated by `edge_dist` function, with dyad ID generated by `dyad_ID`
`threshold`	spatial distance threshold in the units of the projection
`n_min_length`	minimum length of fusion events
`n_max_missing`	maximum number of missing observations within a fusion event
`allow_split`	boolean defining if a single observation can be greater than the threshold distance without initiating fission event

Details

The `edges` must be a `data.table` returned by the `edge_dist` function. In addition, `fusion_id` requires a dyad ID set on the edge list generated by `dyad_id`. If your data is a `data.frame`, you can convert it by reference using `data.table::setDT`.

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 50 m distance threshold.

The `n_min_length` argument defines the minimum number of successive fixes that are required to establish a fusion event. The `n_max_missing` argument defines the the maximum number of allowable missing observations for the dyad within a fusion event. The `allow_split` argument defines if a single observation can be greater than the threshold distance without initiating fission event.

Value

`fusion_id` returns the input `edges` appended with a `fusionID` column.

This column represents the fusion event id. As with `spatsoc`'s grouping functions, the actual value of `fusionID` is arbitrary and represents the identity of a given fusion event. If the data was reordered, the `fusionID` may change, but the membership of each fusion event would not.

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

References

See examples of identifying fission-fusion events with spatiotemporal data:

- https://doi.org/10.1111/ele.12457
- https://doi.org/10.1016/j.anbehav.2018.03.014
- https://doi.org/10.1890/08-0345.1

See Also

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_dist(
    DT,
    threshold = 100,
    id = 'ID',
    coords = c('X', 'Y'),
    timegroup = 'timegroup',
    returnDist = TRUE,
    fillNA = TRUE
  )

dyad_id(edges, 'ID1', 'ID2')

fusion_id(
  edges = edges,
  threshold = 100,
  n_min_length = 1,
  n_max_missing = 0,
  allow_split = FALSE
)
```

get_gbi *Generate group by individual matrix*

Description

get_gbi generates a group by individual matrix. The function expects 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 generating LINESTRINGs and grouping based on spatial intersection. The function expects a `data.table` with relocation data, individual identifiers and a distance threshold. The relocation data is transformed into sf LINESTRINGs using build_lines and intersecting LINESTRINGs are grouped. The threshold argument is used to specify the distance criteria for grouping. 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,
  sfLines = NULL
)
```

Arguments

`DT`	input data.table
`threshold`	The width of the buffer around the lines in the units of the projection. Use `threshold = 0` to compare intersection without buffering.
`projection`	numeric or character defining the coordinate reference system to be passed to sf::st_crs. For example, either `projection = "EPSG:32736"` or `projection = 32736`.
`id`	character string of ID column name
`coords`	character vector of X coordinate and Y coordinate column names. Note: the order is assumed X followed by Y column names.
`timegroup`	timegroup field in the DT within 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
`sfLines`	Alternatively to providing a DT, provide a simple feature LINESTRING object generated with the sf package. The id argument is required to provide the identifier matching each LINESTRING. If an sfLines object is provided, groups cannot be calculated by timegroup or splitBy.

Details

R-spatial evolution:

Please note, spatsoc has followed updates from R spatial, GDAL and PROJ for handling projections, see more at `https://r-spatial.org/r/2020/03/17/wkt.html`.

In addition, `group_lines` (and build_lines) previously used sp::SpatialLines, rgeos::gIntersects, rgeos::gBuffer but have been updated to use sf::st_as_sf, sf::st_linestring, sf::st_intersects, and sf::st_buffer according to the R-spatial evolution, see more at `https://r-spatial.org/r/2022/04/12/evolution.html`.

Notes on arguments:

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 numeric or character defining the coordinate reference system. For example, for UTM zone 36N (EPSG 32736), the projection argument is either `projection = 'EPSG:32736'` or `projection = 32736`. See details in `sf::st_crs()` and `https://spatialreference.org` for a list of EPSG codes.

The `sortBy` argument is used to order the input `DT` when creating sf LINESTRINGs. It must a column in the input `DT` of type 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 intended, otherwise it should be some value 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, system or question.

The `splitBy` argument offers further control building LINESTRINGs. If in your input `DT`, you have multiple temporal groups (e.g.: years) for example, you can provide the name of the column which identifies them and build LINESTRINGs for each individual in each year. 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 intersecting 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 <- 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 expects a `data.table` with relocation data, individual identifiers and an `area` argument. The relocation data is transformed into home range POLYGONs using `build_polys()` with adehabitatHR::mcp or adehabitatHR::kernelUD. If the `area` argument is `FALSE`, `group_polys` returns grouping calculated by spatial overlap. If the `area` argument is `TRUE`, `group_polys` returns the area area and proportion of overlap. 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,
  sfPolys = 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`	numeric or character defining the coordinate reference system to be passed to sf::st_crs. For example, either `projection = "EPSG:32736"` or `projection = 32736`.
`id`	character string of ID column name
`coords`	character vector of X coordinate and Y coordinate column names. Note: the order is assumed X followed by Y column names.
`splitBy`	(optional) character string or vector of grouping column name(s) upon which the grouping will be calculated
`sfPolys`	Alternatively, provide solely a simple features object with POLYGONs or MULTIPOLYGONs. If sfPolys are provided, id is required and splitBy cannot be used.

Details

R-spatial evolution:

Please note, spatsoc has followed updates from R spatial, GDAL and PROJ for handling projections, see more below and details at `https://r-spatial.org/r/2020/03/17/wkt.html`.

In addition, `group_polys` previously used rgeos::gIntersection, rgeos::gIntersects and rgeos::gArea but has been updated to use sf::st_intersects, sf::st_intersection and sf::st_area according to the R-spatial evolution, see more at `https://r-spatial.org/r/2022/04/12/evolution.html`.

Notes on arguments:

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 or numeric defining the coordinate reference system to be passed to sf::st_crs. For example, for UTM zone 36S (EPSG 32736), the projection argument is `projection = "EPSG:32736"` or `projection = 32736`. See `https://spatialreference.org` for a list of EPSG codes.

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.

Along with changes to follow the R-spatial evolution, `group_polys` also now returns area and proportion of overlap with units explicitly specified through the `units` package.

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 <- 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'))
print(areaDT)
```

`group_pts` *Group Points*

Description

`group_pts` groups rows into spatial groups. The function expects 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. Note: the order is assumed X followed by Y column names.
`timegroup`	timegroup field in the DT within 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` or by reassigning using `data.table::data.table`.

The `id`, `coords`, `timegroup` (and optional `splitBy`) arguments expect the names of a column in `DT` which correspond to the individual identifier, X and Y coordinates, timegroup (typically 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 required to define the temporal groups within which spatial groups are calculated. The intended framework is to group rows temporally with `group_times` then spatially with `group_pts` (or `group_lines`, `group_polys`). If you have already calculated temporal groups without `group_times`, you can pass this column to the `timegroup` argument. Note that the expectation is that each individual will be observed only once per timegroup. Caution that accidentally including huge numbers of rows within timegroups can overload your machine since all pairwise distances are calculated within each timegroup.

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 spatialtemporal 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"))

# Select only individuals A, B, C for this example
DT <- DT[ID %in% c('A', 'B', 'C')]

# 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 expects 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')
```

---

leader_direction_group *Leadership along group direction*

Description

Given the mean direction of a group of animals, leader_direction_group shifts the coordinate system to a new origin at the group centroid and rotates the coordinate system by the mean direction to return each individual's position along the mean direction, representing leadership in terms of the front-back position in each group's mean direction.

Usage

```
leader_direction_group(
  DT = NULL,
  group_direction = "group_direction",
  coords = NULL,
  group = "group",
  return_rank = FALSE,
  ties.method = "average"
)
```

Arguments

`DT`	input data.table with group direction columns generated by `direction_group` and centroid columns generated by `centroid_group`
`group_direction`	group_direction column name generated using `direction_group`, default 'group_direction'
`coords`	character vector of X coordinate and Y coordinate column names. Note: the order is assumed X followed by Y column names.
`group`	group column name, generated by `group_pts`, default 'group'
`return_rank`	boolean if rank distance should also be returned, default FALSE
`ties.method`	see `?data.table::frank`

Details

The function accepts a `data.table` with relocation data appended with a `group_direction` column from `direction_group` and group centroid columns from `centroid_group`. Relocation data should be in two columns representing the X and Y coordinates.

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

The `group_direction` argument expects the names of columns in `DT` which correspond to the mean group direction generated by `direction_group`. The mean group direction column is expected in units of radians. The `coords` arguments expects the names of columns in `DT` which correspond to the X and Y coordinate columns. The `return_rank` argument controls if the rank of each individual's distance to the group centroid is also returned. If `return_rank` is TRUE, the `group` argument is required to specify the group column generated by `group_pts`. The `ties.method` argument is passed to `data.table::frank`, see details at `?data.table::frank`.

Value

`leader_direction_group` returns the input `DT` appended with a `position_group_direction` column indicating the position along the group direction in the units of the projection and, optionally when `return_rank = TRUE`, a `rank_position_group_direction` column indicating the the ranked position along the group direction.

A message is returned when `position_group_direction` or `rank_position_group_direction` columns already exist in the input `DT`, because they will be overwritten.

References

See examples of measuring leadership along group direction (also called forefront index):

- `https://doi.org/10.1371/journal.pone.0036567`
- `https://doi.org/10.1111/jfb.15315`
- `https://doi.org/10.1098/rspb.2021.0839`

See Also

direction_group, centroid_group

Examples

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


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

# (Subset example data to reduce example run time)
DT <- DT[year(datetime) == 2016]

# 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 = 50, id = 'ID',
          coords = c('X', 'Y'), timegroup = 'timegroup')

# Calculate direction at each step
direction_step(
  DT = DT,
  id = 'ID',
  coords = c('X', 'Y'),
  projection = 32736
)

# Calculate group centroid
centroid_group(DT, coords = c('X', 'Y'))

# Calculate group direction
direction_group(DT)

# Calculate leader in terms of position along group direction
leader_direction_group(DT, coords = c('X', 'Y'))
```

`randomizations` *Data-stream randomizations*

Description

`randomizations` performs data-stream social network randomization. The function expects 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. Note: the order is assumed X followed by Y 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 `type`s 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

doi: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"))

# Select only individuals A, B, C for this example
DT <- DT[ID %in% c('A', 'B', 'C')]

# 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
)
```

Index