Package ‘spNetwork’

January 14, 2025

Type Package

Title Spatial Analysis on Network

Version 0.4.4.4

Description Perform spatial analysis on network.
Implement several methods for spatial analysis on network: Network Kernel Density estimation, building of spatial matrices based on network distance ('listw' objects from 'spdep' package), K functions estimation
for point pattern analysis on network, k nearest neighbours on network, reachable area calculation, and graph generation
References: Okabe et al (2019) <doi:10.1080/13658810802475491>;
Okabe et al (2012, ISBN:978-0470770818);Baddeley et al (2015, ISBN:9781482210200).

License GPL-2

Encoding UTF-8

LazyData true

Imports spdep (>= 1.1.2), igraph (>= 1.2.6), cubature (>= 2.0.4.1), future.apply (>= 1.4.0), methods (>= 1.7.1), ggplot2 (>= 3.3.0), progressr (>= 0.4.0), data.table (>= 1.12.8), Rcpp (>= 1.0.4.6), Rdpack (>= 2.1.1), dbscan (>= 1.1-8), sf (>= 1.0-3), abind (>= 1.4-5), sfheaders (>= 0.4.4), cppRouting (>= 3.1)

Depends R (>= 3.6)

Suggests future (>= 1.16.0), testthat (>= 3.0.0), kableExtra (>= 1.1.0), RColorBrewer (>= 1.1-2), classInt (>= 0.4-3), reshape2 (>= 1.4.3), rlang (>= 0.4.6), rgl (>= 0.107.14), tmap (>= 3.3-1), smoothr (>= 0.2.2), concaveman (>= 1.1.0), covr (>= 3.5.1), knitr, rmarkdown

RoxygenNote 7.3.2

VignetteBuilder knitr

URL `https://jeremygelb.github.io/spNetwork/`

BugReports `https://github.com/JeremyGelb/spNetwork/issues`

LinkingTo Rcpp, RcppProgress, RcppArmadillo, BH

RdMacros Rdpack

Language en-CA

SystemRequirements C++17

NeedsCompilation yes

Author Jeremy Gelb [aut, cre] (<https://orcid.org/0000-0002-7114-2714>),
Philippe Apparicio [ctb] (<https://orcid.org/0000-0001-6466-9342>)

Maintainer Jeremy Gelb <jeremy.gelb@ucs.inrs.ca>

Repository CRAN

Date/Publication 2025-01-14 09:50:02 UTC

Contents

`spNetwork-package` *spNetwork: Spatial Analysis on Network*

Description

Perform spatial analysis on network. Implement several methods for spatial analysis on network: Network Kernel Density estimation, building of spatial matrices based on network distance ('listw' objects from 'spdep' package), K functions estimation for point pattern analysis on network, k nearest neighbours on network, reachable area calculation, and graph generation References: Okabe et al (2019) doi:10.1080/13658810802475491; Okabe et al (2012, ISBN:978-0470770818);Baddeley et al (2015, ISBN:9781482210200).

Perform spatial analysis on network. Implement several methods for spatial analysis on network: Network Kernel Density estimation, building of spatial matrices based on network distance ('listw' objects from 'spdep' package), K functions estimation for point pattern analysis on network, k nearest neighbours on network, reachable area calculation, and graph generation References: Okabe et al (2019) doi:10.1080/13658810802475491; Okabe et al (2012, ISBN:978-0470770818);Baddeley et al (2015, ISBN:9781482210200).

Author(s)

Maintainer: Jeremy Gelb `<jeremy.gelb@ucs.inrs.ca>` (ORCID)

Other contributors:

- Philippe Apparicio `<philippe.apparicio@ucs.inrs.ca>` (ORCID) [contributor]

See Also

Useful links:

- `https://jeremygelb.github.io/spNetwork/`
- Report bugs at `https://github.com/JeremyGelb/spNetwork/issues`

Useful links:

- `https://jeremygelb.github.io/spNetwork/`
- Report bugs at `https://github.com/JeremyGelb/spNetwork/issues`

adaptive_bw_tnkde_cpp *The exposed function to calculate adaptive bandwidth with space-time interaction for TNKDE (INTERNAL)*

Description

The exposed function to calculate adaptive bandwidth with space-time interaction for TNKDE (INTERNAL)

Usage

```
adaptive_bw_tnkde_cpp(
  method,
  neighbour_list,
  sel_events,
  sel_events_wid,
  sel_events_time,
  events,
  events_wid,
  events_time,
  weights,
  bws_net,
  bws_time,
  kernel_name,
  line_list,
  max_depth,
  min_tol
)
```

Arguments

`method`	a string, one of "simple", "continuous", "discontinuous"
`neighbour_list`	a List, giving for each node an IntegerVector with its neighbours
`sel_events`	a Numeric vector indicating the selected events (id of nodes)
`sel_events_wid`	a Numeric Vector indicating the unique if of the selected events
`sel_events_time`	a Numeric Vector indicating the time of the selected events
`events`	a NumericVector indicating the nodes in the graph being events
`events_wid`	a NumericVector indicating the unique id of all the events
`events_time`	a NumericVector indicating the timestamp of each event
`weights`	a cube with the weights associated with each event for each bws_net and bws_time.
`bws_net`	an arma::vec with the network bandwidths to consider
`bws_time`	an arma::vec with the time bandwidths to consider
`kernel_name`	a string with the name of the kernel to use
`line_list`	a DataFrame describing the lines
`max_depth`	the maximum recursion depth
`min_tol`	a double indicating by how much 0 in density values must be replaced

Value

a vector witht the estimated density at each event location

Examples

```
# no example provided, this is an internal function
```

adaptive_bw_tnkde_cpp2 *The exposed function to calculate adaptive bandwidth with space-time interaction for TNKDE (INTERNAL)*

Description

The exposed function to calculate adaptive bandwidth with space-time interaction for TNKDE (INTERNAL)

Usage

```
adaptive_bw_tnkde_cpp2(
  method,
  neighbour_list,
  sel_events,
  sel_events_wid,
  sel_events_time,
  events,
  events_wid,
  events_time,
  weights,
  bws_net,
  bws_time,
  kernel_name,
  line_list,
  max_depth,
  min_tol
)
```

Arguments

`method`	a string, one of "simple", "continuous", "discontinuous"
`neighbour_list`	a List, giving for each node an IntegerVector with its neighbours
`sel_events`	a Numeric vector indicating the selected events (id of nodes)
`sel_events_wid`	a Numeric Vector indicating the unique if of the selected events
`sel_events_time`	a Numeric Vector indicating the time of the selected events
`events`	a NumericVector indicating the nodes in the graph being events
`events_wid`	a NumericVector indicating the unique id of all the events
`events_time`	a NumericVector indicating the timestamp of each event
`weights`	a cube with the weights associated with each event for each bws_net and bws_time.
`bws_net`	an arma::vec with the network bandwidths to consider
`bws_time`	an arma::vec with the time bandwidths to consider

`kernel_name`	a string with the name of the kernel to use
`line_list`	a DataFrame describing the lines
`max_depth`	the maximum recursion depth
`min_tol`	a double indicating by how much 0 in density values must be replaced

Value

a vector with the estimated density at each event location

Examples

```
# no example provided, this is an internal function
```

`aggregate_points` *Events aggregation*

Description

Function to aggregate points within a radius.

Usage

```
aggregate_points(points, maxdist, weight = "weight", return_ids = FALSE)
```

Arguments

`points`	The feature collection of points to contract (must have a weight column)
`maxdist`	The distance to use
`weight`	The name of the column to use as weight (default is "weight"). The values of the aggregated points for this column will be summed. For all the other columns, only the max value is retained.
`return_ids`	A boolean (default is FALSE), if TRUE, then an index indicating for each point the group it belongs to is returned. If FALSE, then a spatial point features is returned with the points already aggregated.

Details

This function can be used to aggregate points within a radius. This is done by using the dbscan algorithm. This process is repeated until no more modification is applied.

Value

A new feature collection of points

Examples

```
data(bike_accidents)
bike_accidents$weight <- 1
agg_points <- aggregate_points(bike_accidents, 5)
```

`bike_accidents` *Road accidents including a bicyle in Montreal in 2016*

Description

A feature collection (sf object) representing road accidents including a cyclist in Montreal in 2016. The EPSG is 3797, and the data comes from the Montreal OpenData website. It is only a small subset in central districts used to demonstrate the main functions of spNetwork.

Usage

```
bike_accidents
```

Format

A sf object with 347 rows and 4 variables

NB_VICTIME the number of victims

AN the year of the accident

Date the date of the accident (yyyy/mm/dd)

geom the geometry (points)

Source

`https://donnees.montreal.ca/dataset/collisions-routieres`

`build_graph` *Network generation with igraph*

Description

Generate an igraph object from a feature collection of linestrings

Usage

```
build_graph(lines, digits, line_weight, attrs = FALSE)
```

Arguments

`lines`	A feature collection of lines
`digits`	The number of digits to keep from the coordinates
`line_weight`	The name of the column giving the weight of the lines
`attrs`	A boolean indicating if the original lines' attributes should be stored in the final object

Details

This function can be used to generate an undirected graph object (igraph object). It uses the coordinates of the linestrings extremities to create the nodes of the graph. This is why the number of digits in the coordinates is important. Too high precision (high number of digits) might break some connections.

Value

A list containing the following elements:

- graph: an igraph object;
- linelist: the dataframe used to build the graph;
- lines: the original feature collection of linestrings;
- spvertices: a feature collection of points representing the vertices of the graph;
- digits : the number of digits kept for the coordinates.

Examples

```
data(mtl_network)
mtl_network$length <- as.numeric(sf::st_length(mtl_network))
graph_result <- build_graph(mtl_network, 2, "length", attrs = TRUE)
```

`build_graph_directed` *Directed network generation*

Description

Generate a directed igraph object from a feature collection of linestrings

Usage

```
build_graph_directed(lines, digits, line_weight, direction, attrs = FALSE)
```

Arguments

`lines`	A feature collection of linestrings
`digits`	The number of digits to keep from the coordinates
`line_weight`	The name of the column giving the weight of the lines
`direction`	A column name indicating authorized travelling direction on lines. if NULL, then all lines can be used in both directions. Must be the name of a column otherwise. The values of the column must be "FT" (From - To), "TF" (To - From) or "Both"
`attrs`	A boolean indicating if the original lines' attributes should be stored in the final object

Details

This function can be used to generate a directed graph object (igraph object). It uses the coordinates of the linestrings extremities to create the nodes of the graph. This is why the number of digits in the coordinates is important. Too high precision (high number of digits) might break some connections. The column used to indicate directions can only have the following values: "FT" (From-To), "TF" (To-From) and "Both".

Value

A list containing the following elements:

- graph: an igraph object;
- linelist: the dataframe used to build the graph;
- lines: the original feature collection of lines;
- spvertices: a feature collection of points representing the vertices of the graph;
- digits : the number of digits kept for the coordinates.

Examples

```
data(mtl_network)
mtl_network$length <- as.numeric(sf::st_length(mtl_network))
mtl_network$direction <- "Both"
mtl_network[6, "direction"] <- "TF"
mtl_network_directed <- lines_direction(mtl_network, "direction")
graph_result <- build_graph_directed(lines = mtl_network_directed,
        digits = 2,
        line_weight = "length",
        direction = "direction",
        attrs = TRUE)
```

bw_cvl_calc *Bandwidth selection by Cronie and Van Lieshout's Criterion*

Description

Calculate for multiple bandwidth the Cronie and Van Lieshout's Criterion to select an appropriate bandwidth in a data-driven approach.

Usage

```
bw_cvl_calc(
  bws = NULL,
  lines,
  events,
  w,
  kernel_name,
  method,
  diggle_correction = FALSE,
  study_area = NULL,
  adaptive = FALSE,
  trim_bws = NULL,
  mat_bws = NULL,
  max_depth = 15,
  digits = 5,
  tol = 0.1,
  agg = NULL,
  sparse = TRUE,
  zero_strat = "min_double",
  grid_shape = c(1, 1),
  sub_sample = 1,
  verbose = TRUE,
  check = TRUE
)
```

Arguments

`bws`	An ordered numeric vector with the bandwidths
`lines`	A feature collection of linestrings representing the underlying network. The geometries must be simple Linestrings (may crash if some geometries are invalid) without MultiLineSring.
`events`	events A feature collection of points representing the events on the network. The points will be snapped on the network to their closest line.
`w`	A vector representing the weight of each event
`kernel_name`	The name of the kernel to use. Must be one of triangle, gaussian, tricube, cosine, triweight, quartic, epanechnikov or uniform.
`method`	The method to use when calculating the NKDE, must be one of simple / discontinuous / continuous (see nkde details for more information)
`diggle_correction`	A Boolean indicating if the correction factor for edge effect must be used.
`study_area`	A feature collection of polygons representing the limits of the study area.
`adaptive`	A boolean indicating if an adaptive bandwidth must be used. If adaptive = TRUE, the local bandwidth are derived from the global bandwidths (bws)
`trim_bws`	A vector indicating the maximum value an adaptive bandwidth can reach. Higher values will be trimmed. It must have the same length as bws.
`mat_bws`	A matrix giving the bandwidths for each observation and for each global bandwidth. This is usefull when the user want to use a different method from Abramson's smoothing regimen.
`max_depth`	when using the continuous and discontinuous methods, the calculation time and memory use can go wild if the network has many small edges (area with many of intersections and many events). To avoid it, it is possible to set here a maximum depth. Considering that the kernel is divided at intersections, a value of 10 should yield good estimates in most cases. A larger value can be used without

	a problem for the discontinuous method. For the continuous method, a larger value will strongly impact calculation speed.
`digits`	The number of digits to retain from the spatial coordinates. It ensures that topology is good when building the network. Default is 3. Too high a precision (high number of digits) might break some connections
`tol`	A float indicating the minimum distance between the events and the lines' extremities when adding the point to the network. When points are closer, they are added at the extremity of the lines.
`agg`	A double indicating if the events must be aggregated within a distance. If NULL, the events are aggregated only by rounding the coordinates.
`sparse`	A Boolean indicating if sparse or regular matrices should be used by the Rcpp functions. These matrices are used to store edge indices between two nodes in a graph. Regular matrices are faster, but require more memory, in particular with multiprocessing. Sparse matrices are slower (a bit), but require much less memory.
`zero_strat`	A string indicating what to do when density is 0 when calculating LOO density estimate for an isolated event. "min_double" (default) replace the 0 value by the minimum double possible on the machine. "remove" will remove them from the final score. The first approach penalizes more strongly the small bandwidths.
`grid_shape`	A vector of two values indicating how the study area must be split when performing the calculus. Default is c(1,1) (no split). A finer grid could reduce memory usage and increase speed when a large dataset is used. When using multiprocessing, the work in each grid is dispatched between the workers.
`sub_sample`	A float between 0 and 1 indicating the percentage of quadra to keep in the calculus. For large datasets, it may be useful to limit the bandwidth evaluation and thus reduce calculation time.
`verbose`	A Boolean, indicating if the function should print messages about the process.
`check`	A Boolean indicating if the geometry checks must be run before the operation. This might take some times, but it will ensure that the CRS of the provided objects are valid and identical, and that geometries are valid.

Details

The Cronie and Van Lieshout's Criterion (Cronie and Van Lieshout 2018) find the optimal bandwidth by minimizing the difference between the size of the observation window and the sum of the reciprocal of the estimated kernel density at the events locations. In the network case, the size of the study area is the sum of the length of each line in the network. Thus, it is important to only use the necessary parts of the network.

Value

A dataframe with two columns, one for the bandwidths and the second for the Cronie and Van Lieshout's Criterion.

References

Cronie O, Van Lieshout MNM (2018). "A non-model-based approach to bandwidth selection for kernel estimators of spatial intensity functions." *Biometrika*, **105**(2), 455–462.

Examples

```
data(mtl_network)
data(bike_accidents)
cv_scores <- bw_cvl_calc(seq(200,400,50),
                              mtl_network, bike_accidents,
                              rep(1,nrow(bike_accidents)),
                              "quartic", "discontinuous",
                              diggle_correction = FALSE, study_area = NULL,
                              max_depth = 8,
                              digits=2, tol=0.1, agg=5,
                              sparse=TRUE, grid_shape=c(1,1),
                              sub_sample = 1, verbose=TRUE, check=TRUE)
```

`bw_cvl_calc.mc` *Bandwidth selection by Cronie and Van Lieshout's Criterion (multi-core version)*

Description

Calculate for multiple bandwidths the Cronie and Van Lieshout's Criterion to select an appropriate bandwidth in a data-driven approach. A plan from the package future can be used to split the work across several cores. The different cells generated in accordance with the argument grid_shape are used for the parallelization. So if only one cell is generated (`grid_shape = c(1,1)`), the function will use only one core. The progress bar displays the progression for the cells.

Usage

```
bw_cvl_calc.mc(
  bws = NULL,
  lines,
  events,
  w,
  kernel_name,
  method,
  diggle_correction = FALSE,
  study_area = NULL,
  adaptive = FALSE,
  trim_bws = NULL,
  mat_bws = NULL,
  max_depth = 15,
  digits = 5,
  tol = 0.1,
  agg = NULL,
  sparse = TRUE,
  zero_strat = "min_double",
  grid_shape = c(1, 1),
  sub_sample = 1,
  verbose = TRUE,
  check = TRUE
)
```

Arguments

`bws`	An ordered numeric vector with the bandwidths
`lines`	A feature collection of linestrings representing the underlying network. The geometries must be simple Linestrings (may crash if some geometries are invalid) without MultiLineSring.
`events`	events A feature collection of points representing the events on the network. The points will be snapped on the network to their closest line.
`w`	A vector representing the weight of each event
`kernel_name`	The name of the kernel to use. Must be one of triangle, gaussian, tricube, cosine, triweight, quartic, epanechnikov or uniform.
`method`	The method to use when calculating the NKDE, must be one of simple / discontinuous / continuous (see nkde details for more information)
`diggle_correction`	A Boolean indicating if the correction factor for edge effect must be used.
`study_area`	A feature collection of polygons representing the limits of the study area.
`adaptive`	A boolean indicating if an adaptive bandwidth must be used. If adaptive = TRUE, the local bandwidth are derived from the global bandwidths calculated from bw_range and bw_step.
`trim_bws`	A vector indicating the maximum value an adaptive bandwidth can reach. Higher values will be trimmed. It must have the same length as `bws`.
`mat_bws`	A matrix giving the bandwidths for each observation and for each global bandwidth. This is usefull when the user want to use a different method from Abramson's smoothing regimen.
`max_depth`	when using the continuous and discontinuous methods, the calculation time and memory use can go wild if the network has many small edges (area with many of intersections and many events). To avoid it, it is possible to set here a maximum depth. Considering that the kernel is divided at intersections, a value of 10 should yield good estimates in most cases. A larger value can be used without a problem for the discontinuous method. For the continuous method, a larger value will strongly impact calculation speed.
`digits`	The number of digits to retain from the spatial coordinates. It ensures that topology is good when building the network. Default is 3. Too high a precision (high number of digits) might break some connections
`tol`	A float indicating the minimum distance between the events and the lines' extremities when adding the point to the network. When points are closer, they are added at the extremity of the lines.
`agg`	A double indicating if the events must be aggregated within a distance. If NULL, the events are aggregated only by rounding the coordinates.

`sparse`	A Boolean indicating if sparse or regular matrices should be used by the Rcpp functions. These matrices are used to store edge indices between two nodes in a graph. Regular matrices are faster, but require more memory, in particular with multiprocessing. Sparse matrices are slower (a bit), but require much less memory.
`zero_strat`	A string indicating what to do when density is 0 when calculating LOO density estimate for an isolated event. "min_double" (default) replace the 0 value by the minimum double possible on the machine. "remove" will remove them from the final score. The first approach penalizes more strongly the small bandwidths.
`grid_shape`	A vector of two values indicating how the study area must be split when performing the calculus. Default is c(1,1) (no split). A finer grid could reduce memory usage and increase speed when a large dataset is used. When using multiprocessing, the work in each grid is dispatched between the workers.
`sub_sample`	A float between 0 and 1 indicating the percentage of quadra to keep in the calculus. For large datasets, it may be useful to limit the bandwidth evaluation and thus reduce calculation time.
`verbose`	A Boolean, indicating if the function should print messages about the process.
`check`	A Boolean indicating if the geometry checks must be run before the operation. This might take some times, but it will ensure that the CRS of the provided objects are valid and identical, and that geometries are valid.

Details

For more details, see help(bw_cvl_calc)

Value

A dataframe with two columns, one for the bandwidths and the second for the Cronie and Van Lieshout's Criterion.

Examples

```
data(mtl_network)
data(bike_accidents)
future::plan(future::multisession(workers=1))
cv_scores <- bw_cvl_calc.mc(seq(200,400,50),
                             mtl_network, bike_accidents,
                             rep(1,nrow(bike_accidents)),
                             "quartic", "discontinuous",
                             diggle_correction = FALSE, study_area = NULL,
                             max_depth = 8,
                             digits=2, tol=0.1, agg=5,
                             sparse=TRUE, grid_shape=c(1,1),
                             sub_sample = 1, verbose=TRUE, check=TRUE)
## make sure any open connections are closed afterward
if (!inherits(future::plan(), "sequential")) future::plan(future::sequential)
```

bw_cv_likelihood_calc *Bandwidth selection by likelihood cross validation*

Description

Calculate for multiple bandwidth the cross validation likelihood to select an appropriate bandwidth in a data-driven approach

Usage

```
bw_cv_likelihood_calc(
  bws = NULL,
  lines,
  events,
  w,
  kernel_name,
  method,
  diggle_correction = FALSE,
  study_area = NULL,
  adaptive = FALSE,
  trim_bws = NULL,
  mat_bws = NULL,
  max_depth = 15,
  digits = 5,
  tol = 0.1,
  agg = NULL,
  sparse = TRUE,
  grid_shape = c(1, 1),
  sub_sample = 1,
  zero_strat = "min_double",
  verbose = TRUE,
  check = TRUE
)
```

Arguments

`bws`	An ordered numeric vector with the bandwidths
`lines`	A feature collection of linestrings representing the underlying network. The geometries must be simple Linestrings (may crash if some geometries are invalid) without MultiLineSring.
`events`	events A feature collection of points representing the events on the network. The points will be snapped on the network to their closest line.
`w`	A vector representing the weight of each event
`kernel_name`	The name of the kernel to use. Must be one of triangle, gaussian, tricube, cosine, triweight, quartic, epanechnikov or uniform.

`method`	The method to use when calculating the NKDE, must be one of simple / discontinuous / continuous (see nkde details for more information)
`diggle_correction`	A Boolean indicating if the correction factor for edge effect must be used.
`study_area`	A feature collection of polygons representing the limits of the study area.
`adaptive`	A boolean indicating if an adaptive bandwidth must be used. If adaptive = TRUE, the local bandwidth are derived from the global bandwidths (bws)
`trim_bws`	A vector indicating the maximum value an adaptive bandwidth can reach. Higher values will be trimmed. It must have the same length as bws.
`mat_bws`	A matrix giving the bandwidths for each observation and for each global bandwidth. This is usefull when the user want to use a different method from Abramson's smoothing regimen.
`max_depth`	when using the continuous and discontinuous methods, the calculation time and memory use can go wild if the network has many small edges (area with many of intersections and many events). To avoid it, it is possible to set here a maximum depth. Considering that the kernel is divided at intersections, a value of 10 should yield good estimates in most cases. A larger value can be used without a problem for the discontinuous method. For the continuous method, a larger value will strongly impact calculation speed.
`digits`	The number of digits to retain from the spatial coordinates. It ensures that topology is good when building the network. Default is 3. Too high a precision (high number of digits) might break some connections
`tol`	A float indicating the minimum distance between the events and the lines' extremities when adding the point to the network. When points are closer, they are added at the extremity of the lines.
`agg`	A double indicating if the events must be aggregated within a distance. If NULL, the events are aggregated only by rounding the coordinates.
`sparse`	A Boolean indicating if sparse or regular matrices should be used by the Rcpp functions. These matrices are used to store edge indices between two nodes in a graph. Regular matrices are faster, but require more memory, in particular with multiprocessing. Sparse matrices are slower (a bit), but require much less memory.
`grid_shape`	A vector of two values indicating how the study area must be split when performing the calculus. Default is c(1,1) (no split). A finer grid could reduce memory usage and increase speed when a large dataset is used. When using multiprocessing, the work in each grid is dispatched between the workers.
`sub_sample`	A float between 0 and 1 indicating the percentage of quadra to keep in the calculus. For large datasets, it may be useful to limit the bandwidth evaluation and thus reduce calculation time.
`zero_strat`	A string indicating what to do when density is 0 when calculating LOO density estimate for an isolated event. "min_double" (default) replace the 0 value by the minimum double possible on the machine. "remove" will remove them from the final score. The first approach penalizes more strongly the small bandwidths.
`verbose`	A Boolean, indicating if the function should print messages about the process.

`check`	A Boolean indicating if the geometry checks must be run before the operation. This might take some times, but it will ensure that the CRS of the provided objects are valid and identical, and that geometries are valid.

Details

The function calculates the likelihood cross validation score for several bandwidths in order to find the most appropriate one. The general idea is to find the bandwidth that would produce the most similar results if one event was removed from the dataset (leave one out cross validation). We use here the shortcut formula as described by the package spatstat (Baddeley et al. 2021).

$LCV(h) = \sum_i \log \hat{\lambda}_{-i}(x_i)$

Where the sum is taken for all events x_i and where $\hat{\lambda}_{-i}(x_i)$ is the leave-one-out kernel estimate at x_i for a bandwidth h. A higher value indicates a better bandwidth.

Value

A dataframe with two columns, one for the bandwidths and the second for the cross validation score (the lower the better).

References

Baddeley A, Turner R, Rubak E (2021). *spatstat: Spatial Point Pattern Analysis, Model-Fitting, Simulation, Tests*. R package version 2.1-0, https://CRAN.R-project.org/package=spatstat.

Examples

```
data(mtl_network)
data(bike_accidents)
cv_scores <- bw_cv_likelihood_calc(seq(200,800,50),
                               mtl_network, bike_accidents,
                               rep(1,nrow(bike_accidents)),
                               "quartic", "simple",
                               diggle_correction = FALSE, study_area = NULL,
                               max_depth = 8,
                               digits=2, tol=0.1, agg=5,
                               sparse=TRUE, grid_shape=c(1,1),
                               sub_sample = 1, verbose=TRUE, check=TRUE)
```

`bw_cv_likelihood_calc.mc` *Bandwidth selection by likelihood cross validation (multicore)*

Description

Calculate for multiple bandwidth the cross validation likelihood to select an appropriate bandwidth in a data-driven approach

Usage

```
bw_cv_likelihood_calc.mc(
  bws,
  lines,
  events,
  w,
  kernel_name,
  method,
  diggle_correction = FALSE,
  study_area = NULL,
  adaptive = FALSE,
  trim_bws = NULL,
  mat_bws = NULL,
  max_depth = 15,
  digits = 5,
  tol = 0.1,
  agg = NULL,
  sparse = TRUE,
  grid_shape = c(1, 1),
  sub_sample = 1,
  zero_strat = "min_double",
  verbose = TRUE,
  check = TRUE
)
```

Arguments

`bws`	An ordered numeric vector with the bandwidths
`lines`	A feature collection of linestrings representing the underlying network. The geometries must be simple Linestrings (may crash if some geometries are invalid) without MultiLineSring.
`events`	events A feature collection of points representing the events on the network. The points will be snapped on the network to their closest line.
`w`	A vector representing the weight of each event
`kernel_name`	The name of the kernel to use. Must be one of triangle, gaussian, tricube, cosine, triweight, quartic, epanechnikov or uniform.
`method`	The method to use when calculating the NKDE, must be one of simple / discontinuous / continuous (see nkde details for more information)
`diggle_correction`	A Boolean indicating if the correction factor for edge effect must be used.
`study_area`	A feature collection of polygons representing the limits of the study area.
`adaptive`	A boolean indicating if an adaptive bandwidth must be used. If adaptive = TRUE, the local bandwidth are derived from the global bandwidths (bws)
`trim_bws`	A vector indicating the maximum value an adaptive bandwidth can reach. Higher values will be trimmed. It must have the same length as bws.

`mat_bws`	A matrix giving the bandwidths for each observation and for each global bandwidth. This is usefull when the user want to use a different method from Abramson's smoothing regimen.
`max_depth`	when using the continuous and discontinuous methods, the calculation time and memory use can go wild if the network has many small edges (area with many of intersections and many events). To avoid it, it is possible to set here a maximum depth. Considering that the kernel is divided at intersections, a value of 10 should yield good estimates in most cases. A larger value can be used without a problem for the discontinuous method. For the continuous method, a larger value will strongly impact calculation speed.
`digits`	The number of digits to retain from the spatial coordinates. It ensures that topology is good when building the network. Default is 3. Too high a precision (high number of digits) might break some connections
`tol`	A float indicating the minimum distance between the events and the lines' extremities when adding the point to the network. When points are closer, they are added at the extremity of the lines.
`agg`	A double indicating if the events must be aggregated within a distance. If NULL, the events are aggregated only by rounding the coordinates.
`sparse`	A Boolean indicating if sparse or regular matrices should be used by the Rcpp functions. These matrices are used to store edge indices between two nodes in a graph. Regular matrices are faster, but require more memory, in particular with multiprocessing. Sparse matrices are slower (a bit), but require much less memory.
`grid_shape`	A vector of two values indicating how the study area must be split when performing the calculus. Default is c(1,1) (no split). A finer grid could reduce memory usage and increase speed when a large dataset is used. When using multiprocessing, the work in each grid is dispatched between the workers.
`sub_sample`	A float between 0 and 1 indicating the percentage of quadra to keep in the calculus. For large datasets, it may be useful to limit the bandwidth evaluation and thus reduce calculation time.
`zero_strat`	A string indicating what to do when density is 0 when calculating LOO density estimate for an isolated event. "min_double" (default) replace the 0 value by the minimum double possible on the machine. "remove" will remove them from the final score. The first approach penalizes more strongly the small bandwidths.
`verbose`	A Boolean, indicating if the function should print messages about the process.
`check`	A Boolean indicating if the geometry checks must be run before the operation. This might take some times, but it will ensure that the CRS of the provided objects are valid and identical, and that geometries are valid.

Details

See the function bw_cv_likelihood_calc for more details. The calculation is split according to the parameter grid_shape. If `grid_shape = c(1,1)`, then parallel processing cannot be used.

Value

A dataframe with two columns, one for the bandwidths and the second for the cross validation score (the lower the better).

Examples

```
data(mtl_network)
data(bike_accidents)
future::plan(future::multisession(workers=1))
cv_scores <- bw_cv_likelihood_calc.mc(seq(200,800,50),
                               mtl_network, bike_accidents,
                               rep(1,nrow(bike_accidents)),
                               "quartic", "simple",
                               diggle_correction = FALSE, study_area = NULL,
                               max_depth = 8,
                               digits=2, tol=0.1, agg=5,
                               sparse=TRUE, grid_shape=c(1,1),
                               sub_sample = 1, verbose=TRUE, check=TRUE)
## make sure any open connections are closed afterward
if (!inherits(future::plan(), "sequential")) future::plan(future::sequential)
```

`bw_cv_likelihood_calc_tkde`

Bandwidth selection for Temporal Kernel density estimate by likelihood cross validation

---

Description

Calculate the likelihood cross validation score for several bandwidths for the Temporal Kernel density

Usage

```
bw_cv_likelihood_calc_tkde(events, w, bws, kernel_name)
```

Arguments

`events`	A numeric vector representing the moments of occurrence of events
`w`	The weight of the events
`bws`	A numeric vector, the bandwidths to use
`kernel_name`	The name of the kernel to use

Value

A vector with the cross validation scores (the higher the better).

Examples

```
data(bike_accidents)
bike_accidents$Date <- as.POSIXct(bike_accidents$Date, format = "%Y/%m/%d")
start <- min(bike_accidents$Date)
diff <- as.integer(difftime(bike_accidents$Date , start, units = "days"))
w <- rep(1,length(diff))
scores <- bw_cv_likelihood_calc_tkde(diff, w, seq(10,60,10), "quartic")
```

`bw_tnkde_cv_likelihood_calc`
Bandwidth selection by likelihood cross validation for temporal NKDE

Description

Calculate for multiple network and time bandwidths the cross validation likelihood to select an appropriate bandwidth in a data-driven approach

Usage

```
bw_tnkde_cv_likelihood_calc(
  bws_net = NULL,
  bws_time = NULL,
  lines,
  events,
  time_field,
  w,
  kernel_name,
  method,
  arr_bws_net = NULL,
  arr_bws_time = NULL,
  diggle_correction = FALSE,
  study_area = NULL,
  adaptive = FALSE,
  trim_net_bws = NULL,
  trim_time_bws = NULL,
  max_depth = 15,
  digits = 5,
  tol = 0.1,
  agg = NULL,
  sparse = TRUE,
  zero_strat = "min_double",
  grid_shape = c(1, 1),
  sub_sample = 1,
  verbose = TRUE,
  check = TRUE
)
```

Arguments

`bws_net`	An ordered numeric vector with all the network bandwidths
`bws_time`	An ordered numeric vector with all the time bandwidths
`lines`	A feature collection of linestrings representing the underlying network. The geometries must be simple Linestrings (may crash if some geometries are invalid) without MultiLineSring.
`events`	events A feature collection of points representing the events on the network. The points will be snapped on the network to their closest line.
`time_field`	The name of the field in events indicating when the events occurred. It must be a numeric field
`w`	A vector representing the weight of each event
`kernel_name`	The name of the kernel to use. Must be one of triangle, gaussian, tricube, cosine, triweight, quartic, epanechnikov or uniform.
`method`	The method to use when calculating the NKDE, must be one of simple / discontinuous / continuous (see nkde details for more information)
`arr_bws_net`	An array with all the local netowrk bandwidths precalculated (for each event, and at each possible combinaison of network and temporal bandwidths). The dimensions must be c(length(net_bws), length(time_bws), nrow(events)))
`arr_bws_time`	An array with all the local time bandwidths precalculated (for each event, and at each possible combinaison of network and temporal bandwidths). The dimensions must be c(length(net_bws), length(time_bws), nrow(events)))
`diggle_correction`	A Boolean indicating if the correction factor for edge effect must be used.
`study_area`	A feature collection of polygons representing the limits of the study area.
`adaptive`	A boolean indicating if local bandwidths must be calculated
`trim_net_bws`	A numeric vector with the maximum local network bandwidth. If local bandwidths have higher values, they will be replaced by the corresponding value in this vector.
`trim_time_bws`	A numeric vector with the maximum local time bandwidth. If local bandwidths have higher values, they will be replaced by the corresponding value in this vector.
`max_depth`	when using the continuous and discontinuous methods, the calculation time and memory use can go wild if the network has many small edges (area with many of intersections and many events). To avoid it, it is possible to set here a maximum depth. Considering that the kernel is divided at intersections, a value of 10 should yield good estimates in most cases. A larger value can be used without a problem for the discontinuous method. For the continuous method, a larger value will strongly impact calculation speed.
`digits`	The number of digits to retain from the spatial coordinates. It ensures that topology is good when building the network. Default is 3. Too high a precision (high number of digits) might break some connections
`tol`	A float indicating the minimum distance between the events and the lines' extremities when adding the point to the network. When points are closer, they are added at the extremity of the lines.

`agg`	A double indicating if the events must be aggregated within a distance. If NULL, the events are aggregated only by rounding the coordinates.
`sparse`	A Boolean indicating if sparse or regular matrices should be used by the Rcpp functions. These matrices are used to store edge indices between two nodes in a graph. Regular matrices are faster, but require more memory, in particular with multiprocessing. Sparse matrices are slower (a bit), but require much less memory.
`zero_strat`	A string indicating what to do when density is 0 when calculating LOO density estimate for an isolated event. "min_double" (default) replace the 0 value by the minimum double possible on the machine. "remove" will remove them from the final score. The first approach penalizes more strongly the small bandwidths.
`grid_shape`	A vector of two values indicating how the study area must be split when performing the calculus. Default is c(1,1) (no split). A finer grid could reduce memory usage and increase speed when a large dataset is used. When using multiprocessing, the work in each grid is dispatched between the workers.
`sub_sample`	A float between 0 and 1 indicating the percentage of quadra to keep in the calculus. For large datasets, it may be useful to limit the bandwidth evaluation and thus reduce calculation time.
`verbose`	A Boolean, indicating if the function should print messages about the process.
`check`	A Boolean indicating if the geometry checks must be run before the operation. This might take some times, but it will ensure that the CRS of the provided objects are valid and identical, and that geometries are valid.

Details

The function calculates the likelihood cross validation score for several time and network bandwidths in order to find the most appropriate one. The general idea is to find the pair of bandwidths that would produce the most similar results if one event is removed from the dataset (leave one out cross validation). We use here the shortcut formula as described by the package spatstat (Baddeley et al. 2021).

$LCV(h) = \sum_i \log \hat{\lambda}_{-i}(x_i)$

Where the sum is taken for all events x_i and where $\hat{\lambda}_{-i}(x_i)$ is the leave-one-out kernel estimate at x_i for a bandwidth h. A higher value indicates a better bandwidth.

Value

A matrix with the cross validation score. Each row corresponds to a network bandwidth and each column to a time bandwidth (the higher the better).

References

Baddeley A, Turner R, Rubak E (2021). *spatstat: Spatial Point Pattern Analysis, Model-Fitting, Simulation, Tests*. R package version 2.1-0, `https://CRAN.R-project.org/package=spatstat`.

Examples

```
# loading the data
data(mtl_network)
data(bike_accidents)

# converting the Date field to a numeric field (counting days)
bike_accidents$Time <- as.POSIXct(bike_accidents$Date, format = "%Y/%m/%d")
bike_accidents$Time <- difftime(bike_accidents$Time, min(bike_accidents$Time), units = "days")
bike_accidents$Time <- as.numeric(bike_accidents$Time)
bike_accidents <- subset(bike_accidents, bike_accidents$Time>=89)

# calculating the cross validation values
cv_scores <- bw_tnkde_cv_likelihood_calc(
  bws_net = seq(100,800,100),
  bws_time = seq(10,60,5),
  lines = mtl_network,
  events = bike_accidents,
  time_field = "Time",
  w = rep(1, nrow(bike_accidents)),
  kernel_name = "quartic",
  method = "discontinuous",
  diggle_correction = FALSE,
  study_area = NULL,
  max_depth = 10,
  digits = 2,
  tol = 0.1,
  agg = 15,
  sparse=TRUE,
  grid_shape=c(1,1),
  sub_sample=1,
  verbose = FALSE,
  check = TRUE)
```

---

bw_tnkde_cv_likelihood_calc.mc

Bandwidth selection by likelihood cross validation for temporal NKDE (multicore)

Description

Calculate for multiple network and time bandwidths the cross validation likelihood to select an appropriate bandwidth in a data-driven approach with multicore support

Usage

```
bw_tnkde_cv_likelihood_calc.mc(
  bws_net = NULL,
  bws_time = NULL,
  lines,
  events,
  time_field,
  w,
  kernel_name,
  method,
  arr_bws_net = NULL,
  arr_bws_time = NULL,
  diggle_correction = FALSE,
  study_area = NULL,
  adaptive = FALSE,
  trim_net_bws = NULL,
  trim_time_bws = NULL,
  max_depth = 15,
  digits = 5,
  tol = 0.1,
  agg = NULL,
  sparse = TRUE,
  zero_strat = "min_double",
  grid_shape = c(1, 1),
  sub_sample = 1,
  verbose = TRUE,
  check = TRUE
)
```

Arguments

`bws_net`	An ordered numeric vector with all the network bandwidths
`bws_time`	An ordered numeric vector with all the time bandwidths
`lines`	A feature collection of linestrings representing the underlying network. The geometries must be simple Linestrings (may crash if some geometries are invalid) without MultiLineSring.
`events`	events A feature collection of points representing the events on the network. The points will be snapped on the network to their closest line.
`time_field`	The name of the field in events indicating when the events occurred. It must be a numeric field
`w`	A vector representing the weight of each event
`kernel_name`	The name of the kernel to use. Must be one of triangle, gaussian, tricube, cosine, triweight, quartic, epanechnikov or uniform.
`method`	The method to use when calculating the NKDE, must be one of simple / discontinuous / continuous (see nkde details for more information)
`arr_bws_net`	An array with all the local netowrk bandwidths precalculated (for each event, and at each possible combinaison of network and temporal bandwidths). The dimensions must be c(length(net_bws), length(time_bws), nrow(events)))
`arr_bws_time`	An array with all the local time bandwidths precalculated (for each event, and at each possible combinaison of network and temporal bandwidths). The dimensions must be c(length(net_bws), length(time_bws), nrow(events)))

`diggle_correction`	A Boolean indicating if the correction factor for edge effect must be used.
`study_area`	A feature collection of polygons representing the limits of the study area.
`adaptive`	A boolean indicating if local bandwidths must be calculated
`trim_net_bws`	A numeric vector with the maximum local network bandwidth. If local bandwidths have higher values, they will be replaced by the corresponding value in this vector.
`trim_time_bws`	A numeric vector with the maximum local time bandwidth. If local bandwidths have higher values, they will be replaced by the corresponding value in this vector.
`max_depth`	when using the continuous and discontinuous methods, the calculation time and memory use can go wild if the network has many small edges (area with many of intersections and many events). To avoid it, it is possible to set here a maximum depth. Considering that the kernel is divided at intersections, a value of 10 should yield good estimates in most cases. A larger value can be used without a problem for the discontinuous method. For the continuous method, a larger value will strongly impact calculation speed.
`digits`	The number of digits to retain from the spatial coordinates. It ensures that topology is good when building the network. Default is 3. Too high a precision (high number of digits) might break some connections
`tol`	A float indicating the minimum distance between the events and the lines' extremities when adding the point to the network. When points are closer, they are added at the extremity of the lines.
`agg`	A double indicating if the events must be aggregated within a distance. If NULL, the events are aggregated only by rounding the coordinates.
`sparse`	A Boolean indicating if sparse or regular matrices should be used by the Rcpp functions. These matrices are used to store edge indices between two nodes in a graph. Regular matrices are faster, but require more memory, in particular with multiprocessing. Sparse matrices are slower (a bit), but require much less memory.
`zero_strat`	A string indicating what to do when density is 0 when calculating LOO density estimate for an isolated event. "min_double" (default) replace the 0 value by the minimum double possible on the machine. "remove" will remove them from the final score. The first approach penalizes more strongly the small bandwidths.
`grid_shape`	A vector of two values indicating how the study area must be split when performing the calculus. Default is c(1,1) (no split). A finer grid could reduce memory usage and increase speed when a large dataset is used. When using multiprocessing, the work in each grid is dispatched between the workers.
`sub_sample`	A float between 0 and 1 indicating the percentage of quadra to keep in the calculus. For large datasets, it may be useful to limit the bandwidth evaluation and thus reduce calculation time.
`verbose`	A Boolean, indicating if the function should print messages about the process.
`check`	A Boolean indicating if the geometry checks must be run before the operation. This might take some times, but it will ensure that the CRS of the provided objects are valid and identical, and that geometries are valid.

Details

See the function bws_tnkde_cv_likelihood_calc for more details. Note that the calculation is split according to the grid_shape argument. If the grid_shape is `c(1,1)` then only one process can be used.

Value

A matrix with the cross validation score. Each row corresponds to a network bandwidth and each column to a time bandwidth (the higher the better).

Examples

```
# loading the data
data(mtl_network)
data(bike_accidents)

# converting the Date field to a numeric field (counting days)
bike_accidents$Time <- as.POSIXct(bike_accidents$Date, format = "%Y/%m/%d")
bike_accidents$Time <- difftime(bike_accidents$Time, min(bike_accidents$Time), units = "days")
bike_accidents$Time <- as.numeric(bike_accidents$Time)
bike_accidents <- subset(bike_accidents, bike_accidents$Time>=89)

future::plan(future::multisession(workers=1))

# calculating the cross validation values
cv_scores <- bw_tnkde_cv_likelihood_calc.mc(
  bws_net = seq(100,800,100),
  bws_time = seq(10,60,5),
  lines = mtl_network,
  events = bike_accidents,
  time_field = "Time",
  w = rep(1, nrow(bike_accidents)),
  kernel_name = "quartic",
  method = "discontinuous",
  diggle_correction = FALSE,
  study_area = NULL,
  max_depth = 10,
  digits = 2,
  tol = 0.1,
  agg = 15,
  sparse=TRUE,
  grid_shape=c(1,1),
  sub_sample=1,
  verbose = FALSE,
  check = TRUE)

## make sure any open connections are closed afterward
if (!inherits(future::plan(), "sequential")) future::plan(future::sequential)
```

calc_isochrones *Isochrones calculation*

Description

Calculate isochrones on a network

Usage

```
calc_isochrones(
  lines,
  dists,
  start_points,
  donught = FALSE,
  mindist = 1,
  weight = NULL,
  direction = NULL
)
```

Arguments

`lines`	A feature collection of lines representing the edges of the network
`dists`	A vector of the size of the desired isochrones. Can also be a list of vector when each start point must have its own distances. If so, the length of the list must be equal to the number of rows in start_points.
`start_points`	A feature collection of points representing the starting points if the isochrones
`donught`	A boolean indicating if the returned lines must overlap for each distance (FALSE, default) or if the lines must be cut between each distance step (TRUE).
`mindist`	The minimum distance between two points. When two points are too close, they might end up snapped at the same location on a line. Default is 1.
`weight`	The name of the column in lines to use an edge weight. If NULL, the geographical length is used. Note that if lines are split during the network creation, the weight column is recalculated proportionally to the new lines length.
`direction`	The name of the column indicating authorized travelling direction on lines. if NULL, then all lines can be used in both directions (undirected). The values of the column must be "FT" (From - To), "TF" (To - From) or "Both".

Details

An isochrone is the set of reachable lines around a node in a network within a specified distance (or time). This function perform dynamic segmentation to return the part of the edges reached and not only the fully covered edges. Several start points and several distances can be given. The network can also be directed. The lines returned by the function are the most accurate representation of the isochrones. However, if polygons are required for mapping, the vignette "Calculating isochrones" shows how to create smooth polygons from the returned sets of lines.

Value

A feature collection of lines representing the isochrones with the following columns

- point_id: the index of the point at the centre of the isochrone;
- distance: the size of the isochrone

Examples

```
library(sf)
# creating a simple network
wkt_lines <- c(
  "LINESTRING (0.0 0.0, 5.0 0.0)",
  "LINESTRING (0.0 -5.0, 5.0 -5.0)",
  "LINESTRING (5.0 0.0, 5.0 5.0)",
  "LINESTRING (5.0 -5.0, 5.0 -10.0)",
  "LINESTRING (5.0 0.0, 5.0 -5.0)",
  "LINESTRING (5.0 0.0, 10.0 0.0)",
  "LINESTRING (5.0 -5.0, 10.0 -5.0)",
  "LINESTRING (10.0 0, 10.0 -5.0)",
  "LINESTRING (10.0 -10.0, 10.0 -5.0)",
  "LINESTRING (15.0 -5.0, 10.0 -5.0)",
  "LINESTRING (10.0 0.0, 15.0 0.0)",
  "LINESTRING (10.0 0.0, 10.0 5.0)")

linesdf <- data.frame(wkt = wkt_lines,
                      id = paste("l",1:length(wkt_lines),sep=""))

lines <- st_as_sf(linesdf, wkt = "wkt", crs = 32188)

# and the definition of the starting point
start_points <- data.frame(x=c(5),
                           y=c(-2.5))
start_points <- st_as_sf(start_points, coords = c("x","y"), crs = 32188)

# setting the directions

lines$direction <- "Both"
lines[6,"direction"] <- "TF"

isochrones <- calc_isochrones(lines,dists = c(10,12),
                              donught = TRUE,
                              start_points = start_points,
                              direction = "direction")
```

---

closest_points *Find closest points*

Description

Solve the nearest neighbour problem for two feature collections of points This is a simple wrap-up of the dbscan::kNN function

Usage

```
closest_points(origins, targets)
```

Arguments

`origins`	a feature collection of points
`targets`	a feature collection of points

Value

for each origin point, the index of the nearest target point

Examples

```
data(mtl_libraries)
data(mtl_theatres)
close_libs <- closest_points(mtl_theatres, mtl_libraries)
```

`cosine_kernel` *Cosine kernel*

Description

Function implementing the cosine kernel.

Usage

```
cosine_kernel(d, bw)
```

Arguments

`d`	The distance from the event
`bw`	The bandwidth used for the kernel

Value

The estimated density

Examples

```
#This is an internal function, no example provided
```

cross_gfunc_cpp *c++ cross g function*

Description

c++ cross g function (INTERNAL)

Usage

```
cross_gfunc_cpp(dist_mat, start, end, step, width, Lt, na, nb, wa, wb)
```

Arguments

dist_mat	A matrix with the distances between points
start	A float, the start value for evaluating the g-function
end	A float, the last value for evaluating the g-function
step	A float, the jump between two evaluations of the k-function
width	The width of each donut
Lt	The total length of the network
na	The number of points in set A
nb	The number of points in set B
wa	The weight of the points in set A (coincident points)
wb	The weight of the points in set B (coincident points)

cross_kfunctions *Network cross k and g functions (maturing)*

Description

Calculate the cross k and g functions for a set of points on a network. (maturing)

Usage

```
cross_kfunctions(
  lines,
  pointsA,
  pointsB,
  start,
  end,
  step,
  width,
  nsim,
  conf_int = 0.05,
  digits = 2,
  tol = 0.1,
  resolution = NULL,
  agg = NULL,
  verbose = TRUE,
  return_sims = FALSE,
  calc_g_func = TRUE
)
```

Arguments

`lines`	A feature collection of linestrings representing the underlying network. The geometries must be simple Linestrings (may crash if some geometries are invalid) without MultiLineSring
`pointsA`	A feature collection of points representing the points to which the distances are calculated.
`pointsB`	A feature collection of points representing the points from which the distances are calculated.
`start`	A double, the lowest distance used to evaluate the k and g functions
`end`	A double, the highest distance used to evaluate the k and g functions
`step`	A double, the step between two evaluations of the k and g function. start, end and step are used to create a vector of distances with the function seq
`width`	The width of each donut for the g-function. Half of the width is applied on both sides of the considered distance
`nsim`	An integer indicating the number of Monte Carlo simulations to perform for inference
`conf_int`	A double indicating the width confidence interval (default = 0.05) calculated on the Monte Carlo simulations
`digits`	An integer indicating the number of digits to retain from the spatial coordinates
`tol`	When adding the points to the network, specify the minimum distance between these points and the lines' extremities. When points are closer, they are added at the extremity of the lines
`resolution`	When simulating random points on the network, selecting a resolution will reduce greatly the calculation time. When resolution is null the random points can occur everywhere on the graph. If a value is specified, the edges are split according to this value and the random points can only be vertices on the new network
`agg`	A double indicating if the events must be aggregated within a distance. If NULL, the events are aggregated only by rounding the coordinates
`verbose`	A Boolean indicating if progress messages should be displayed
`return_sims`	a boolean indicating if the simulated k and g values must also be returned.
`calc_g_func`	A Boolean indicating if the G function must also be calculated (TRUE by default). If FALSE, then only the K function is calculated

Details

The cross k-function is a method to characterize the dispersion of a set of points (A) around a second set of points (B). For each point in B, the numbers of other points in A in subsequent radii are calculated. This empirical cross k-function can be more or less clustered than a cross k-function obtained if the points in A were randomly located around points in B. In a network, the network distance is used instead of the Euclidean distance. This function uses Monte Carlo simulations to assess if the points are clustered or dispersed and gives the results as a line plot. If the line of the observed cross k-function is higher than the shaded area representing the values of the simulations, then the points in A are more clustered around points in B than what we can expect from randomness and vice-versa. The function also calculates the cross g-function, a modified version of the cross k-function using rings instead of disks. The width of the ring must be chosen. The main interest is to avoid the cumulative effect of the classical k-function. Note that the cross k-function of points A around B is not necessarily the same as the cross k-function of points B around A. This function is maturing, it works as expected (unit tests) but will probably be modified in the future releases (gain speed, advanced features, etc.).

Value

A list with the following values :

`plotk`	A ggplot2 object representing the values of the cross k-function
`plotg`	A ggplot2 object representing the values of the cross g-function
`values`	A DataFrame with the values used to build the plots

Examples

```
data(main_network_mtl)
data(mtl_libraries)
data(mtl_theatres)
result <- cross_kfunctions(main_network_mtl, mtl_theatres, mtl_libraries,
                           start = 0, end = 2500, step = 10, width = 250,
                           nsim = 50, conf_int = 0.05, digits = 2,
                           tol = 0.1, agg = NULL, verbose = FALSE)
```

`cross_kfunctions.mc` *Network cross k and g functions (maturing, multicore)*

Description

Calculate the cross k and g functions for a set of points on a network. For more details, see the document of the function cross_kfunctions.

Usage

```
cross_kfunctions.mc(
  lines,
  pointsA,
  pointsB,
  start,
  end,
  step,
  width,
  nsim,
  conf_int = 0.05,
  digits = 2,
  tol = 0.1,
  resolution = NULL,
  agg = NULL,
  verbose = TRUE,
  return_sims = FALSE,
  calc_g_func = TRUE,
  grid_shape = c(1, 1)
)
```

Arguments

`lines`	A feature collection of linestrings representing the underlying network. The geometries must be simple Linestrings (may crash if some geometries are invalid) without MultiLineSring
`pointsA`	A feature collection of points representing the points to which the distances are calculated.
`pointsB`	A feature collection of points representing the points from which the distances are calculated.
`start`	A double, the lowest distance used to evaluate the k and g functions
`end`	A double, the highest distance used to evaluate the k and g functions
`step`	A double, the step between two evaluations of the k and g function. start, end and step are used to create a vector of distances with the function seq
`width`	The width of each donut for the g-function. Half of the width is applied on both sides of the considered distance
`nsim`	An integer indicating the number of Monte Carlo simulations to perform for inference
`conf_int`	A double indicating the width confidence interval (default = 0.05) calculated on the Monte Carlo simulations
`digits`	An integer indicating the number of digits to retain from the spatial coordinates
`tol`	When adding the points to the network, specify the minimum distance between these points and the lines' extremities. When points are closer, they are added at the extremity of the lines

`resolution`	When simulating random points on the network, selecting a resolution will reduce greatly the calculation time. When resolution is null the random points can occur everywhere on the graph. If a value is specified, the edges are split according to this value and the random points can only be vertices on the new network
`agg`	A double indicating if the events must be aggregated within a distance. If NULL, the events are aggregated only by rounding the coordinates
`verbose`	A Boolean indicating if progress messages should be displayed
`return_sims`	a boolean indicating if the simulated k and g values must also be returned.
`calc_g_func`	A Boolean indicating if the G function must also be calculated (TRUE by default). If FALSE, then only the K function is calculated
`grid_shape`	A vector of two values indicating how the study area must be split when performing the calculus. Default is c(1,1) (no split). A finer grid could reduce memory usage and increase speed when a large dataset is used. When using multiprocessing, the work in each grid is dispatched between the workers.

Value

A list with the following values :

`plotk`	A ggplot2 object representing the values of the cross k-function
`plotg`	A ggplot2 object representing the values of the cross g-function
`values`	A DataFrame with the values used to build the plots

Examples

```
data(main_network_mtl)
data(mtl_libraries)
data(mtl_theatres)
future::plan(future::multisession(workers=1))
result <- cross_kfunctions.mc(main_network_mtl, mtl_theatres, mtl_libraries,
                            start = 0, end = 2500, step = 10, width = 250,
                            nsim = 50, conf_int = 0.05, digits = 2,
                            tol = 0.1, agg = NULL, verbose = FALSE)
```

`cross_kfunc_cpp` *c++ cross k function*

Description

c++ cross k function

Usage

```
cross_kfunc_cpp(dist_mat, start, end, step, Lt, na, nb, wa, wb)
```

Arguments

`dist_mat`	A square matrix with the distances between points
`start`	A float, the start value for evaluating the k-function
`end`	A float, the last value for evaluating the k-function
`step`	A float, the jump between two evaluations of the k-function
`Lt`	The total length of the network
`na`	The number of points in set A
`nb`	The number of points in set B
`wa`	The weight of the points in set A (coincident points)
`wb`	The weight of the points in set B (coincident points)

`epanechnikov_kernel` *Epanechnikov kernel*

Description

Function implementing the epanechnikov kernel.

Usage

```
epanechnikov_kernel(d, bw)
```

Arguments

`d`	The distance from the event
`bw`	The bandwidth used for the kernel

Value

The estimated density

Examples

```
#This is an internal function, no example provided
```

esc_kernel_loo_nkde *The worker function to calculate continuous TNKDE likelihood cv*

Description

The worker function to calculate continuous TNKDE likelihood cv (INTERNAL)

Arguments

`kernel_func`	a cpp pointer function (selected with the kernel name)
`edge_mat`	matrix, to find the id of each edge given two neighbours.
`events`	a NumericVector indicating the nodes in the graph being events
`neighbour_list`	a List, giving for each node an IntegerVector with its neighbours
`v`	the actual node to consider (int)
`bws_net`	an arma::vec with the network bandwidths to consider
`line_weights`	a vector with the length of the edges
`depth`	the actual recursion depth
`max_depth`	the maximum recursion depth

Value

a cube with the impact of the event v on each other events for each pair of bandwidths (cube(bws_net, bws_time, events))

esc_kernel_loo_tnkde *The worker function to calculate continuous TNKDE likelihood cv*

Description

The worker function to calculate continuous TNKDE likelihood cv (INTERNAL)

Arguments

`kernel_func`	a cpp pointer function (selected with the kernel name)
`edge_mat`	matrix, to find the id of each edge given two neighbours.
`events`	a NumericVector indicating the nodes in the graph being events
`time_events`	a NumericVector indicating the timestamp of each event
`neighbour_list`	a List, giving for each node an IntegerVector with its neighbours
`v`	the actual node to consider (int)
`v_time`	the time of v (double)
`bws_net`	an arma::vec with the network bandwidths to consider

`bws_time`	an arma::vec with the time bandwidths to consider
`line_weights`	a vector with the length of the edges
`depth`	the actual recursion depth
`max_depth`	the maximum recursion depth

Value

a cube with the impact of the event v on each other event for each pair of bandwidths (cube(bws_net, bws_time, events))

`esd_kernel_loo_nkde` *The worker function to calculate discontinuous TNKDE likelihood cv*

Description

The worker function to calculate discontinuous TNKDE likelihood cv (INTERNAL)

Arguments

`kernel_func`	a cpp pointer function (selected with the kernel name)
`edge_mat`	matrix, to find the id of each edge given two neighbours.
`events`	a NumericVector indicating the nodes in the graph being events
`neighbour_list`	a List, giving for each node an IntegerVector with its neighbours
`v`	the actual node to consider (int)
`bws_net`	an arma::vec with the network bandwidths to consider
`line_weights`	a vector with the length of the edges
`depth`	the actual recursion depth
`max_depth`	the maximum recursion depth

Value

a cube with the impact of the event v on each other events for each pair of bandwidths (cube(bws_net, bws_time, events))

`esd_kernel_loo_tnkde` *The worker function to calculate discontinuous TNKDE likelihood cv*

Description

The worker function to calculate discontinuous TNKDE likelihood cv (INTERNAL)

Arguments

`kernel_func`	a cpp pointer function (selected with the kernel name)
`edge_mat`	matrix, to find the id of each edge given two neighbours.
`events`	a NumericVector indicating the nodes in the graph being events
`time_events`	a NumericVector indicating the timestamp of each event
`neighbour_list`	a List, giving for each node an IntegerVector with its neighbours
`v`	the actual node to consider (int)
`v_time`	the time of v (double)
`bws_net`	an arma::vec with the network bandwidths to consider
`bws_time`	an arma::vec with the time bandwidths to consider
`line_weights`	a vector with the length of the edges
`depth`	the actual recursion depth
`max_depth`	the maximum recursion depth

Value

a cube with the impact of the event v on each other event for each pair of bandwidths (cube(bws_net, bws_time, events))

`gaussian_kernel` *Gaussian kernel*

Description

Function implementing the gaussian kernel.

Usage

```
gaussian_kernel(d, bw)
```

Arguments

`d`	The distance from the event
`bw`	The bandwidth used for the kernel

Value

The estimated density

Examples

```
#This is an internal function, no example provided
```

---

gaussian_kernel_scaled   *Scaled gaussian kernel*

Description

Function implementing the scaled gaussian kernel.

Usage

```
gaussian_kernel_scaled(d, bw)
```

Arguments

d	The distance from the event
bw	The bandwidth used for the kernel

Value

The estimated density

Examples

```
#This is an internal function, no example provided
```

---

gfunc_cpp   *c++ g function*

Description

c++ g function (INTERNAL)

Usage

```
gfunc_cpp(dist_mat, start, end, step, width, Lt, n, w)
```

Arguments

`dist_mat`	A square matrix with the distances between points
`start`	A float, the start value for evaluating the g-function
`end`	A float, the last value for evaluating the g-function
`step`	A float, the jump between two evaluations of the k-function
`width`	The width of each donut
`Lt`	The total length of the network
`n`	The number of points
`w`	The weight of the points (coincident points)

Value

A numeric vector with the values of the g function evaluated at the required distances

graph_checking *Topological error*

Description

A utility function to find topological errors in a network.

Usage

```
graph_checking(lines, digits, max_search = 5, tol = 0.1)
```

Arguments

`lines`	A feature collection of linestrings representing the network
`digits`	An integer indicating the number of digits to retain for coordinates
`max_search`	The maximum number of nearest neighbour to search to find close_nodes
`tol`	The minimum distance expected between two nodes. If two nodes are closer, they are returned in the result of the function.

Details

This function can be used to check for three common problems in networks: disconnected components, dangle nodes and close nodes. When a network has disconnected components, this means that several unconnected graphs are composing the overall network. This can be caused by topological errors in the dataset. Dangle nodes are nodes connected to only one other node. This type of node can be normal at the border of a network, but can also be caused by topological errors. Close nodes are nodes that are not coincident, but so close that they probably should be coincident.

Value

A list with three elements. The first is a feature collection of points indicating for each node of the network to which component it belongs. The second is a feature collection of points with nodes that are too close one of each other. The third is a feature collection of points with the dangle nodes of the network.

Examples

```
data(mtl_netowrk)
topo_errors <- graph_checking(mtl_network, 2)
```

`kfunctions` *Network k and g functions (maturing)*

Description

Calculate the k and g functions for a set of points on a network (maturing).

Usage

```
kfunctions(
  lines,
  points,
  start,
  end,
  step,
  width,
  nsim,
  conf_int = 0.05,
  digits = 2,
  tol = 0.1,
  agg = NULL,
  verbose = TRUE,
  return_sims = FALSE,
  calc_g_func = TRUE,
  resolution = NULL
)
```

Arguments

`lines`	A feature collection of linestrings representing the underlying network. The geometries must be simple Linestrings (may crash if some geometries are invalid) without MultiLineSring
`points`	A feature collection of points representing the points on the network. These points will be snapped on their nearest line
`start`	A double, the lowest distance used to evaluate the k and g functions

`end`	A double, the highest distance used to evaluate the k and g functions
`step`	A double, the step between two evaluations of the k and g function. start, end and step are used to create a vector of distances with the function seq
`width`	The width of each donut for the g-function. Half of the width is applied on both sides of the considered distance
`nsim`	An integer indicating the number of Monte Carlo simulations to perform for inference
`conf_int`	A double indicating the width confidence interval (default = 0.05) calculated on the Monte Carlo simulations
`digits`	An integer indicating the number of digits to retain from the spatial coordinates
`tol`	When adding the points to the network, specify the minimum distance between these points and the lines' extremities. When points are closer, they are added at the extremity of the lines
`agg`	A double indicating if the events must be aggregated within a distance. If NULL, the events are aggregated only by rounding the coordinates
`verbose`	A Boolean indicating if progress messages should be displayed
`return_sims`	a boolean indicating if the simulated k and g values must also be returned.
`calc_g_func`	A Boolean indicating if the G function must also be calculated (TRUE by default). If FALSE, then only the K function is calculated
`resolution`	When simulating random points on the network, selecting a resolution will reduce greatly the calculation time. When resolution is null the random points can occur everywhere on the graph. If a value is specified, the edges are split according to this value and the random points can only be vertices on the new network

Details

The k-function is a method to characterize the dispersion of a set of points. For each point, the numbers of other points in subsequent radii are calculated. This empirical k-function can be more or less clustered than a k-function obtained if the points were randomly located in space. In a network, the network distance is used instead of the Euclidean distance. This function uses Monte Carlo simulations to assess if the points are clustered or dispersed, and gives the results as a line plot. If the line of the observed k-function is higher than the shaded area representing the values of the simulations, then the points are more clustered than what we can expect from randomness and vice-versa. The function also calculates the g-function, a modified version of the k-function using rings instead of disks. The width of the ring must be chosen. The main interest is to avoid the cumulative effect of the classical k-function. This function is maturing, it works as expected (unit tests) but will probably be modified in the future releases (gain speed, advanced features, etc.).

Value

A list with the following values :

- plotk: A ggplot2 object representing the values of the k-function
- plotg: A ggplot2 object representing the values of the g-function
- values: A DataFrame with the values used to build the plots

Examples

```
data(main_network_mtl)
data(mtl_libraries)
result <- kfunctions(main_network_mtl, mtl_libraries,
     start = 0, end = 2500, step = 100,
     width = 200, nsim = 50,
     conf_int = 0.05, tol = 0.1, agg = NULL,
     calc_g_func = TRUE,
     verbose = FALSE)
```

`kfunctions.mc` *Network k and g functions (multicore)*

Description

Calculate the k and g functions for a set of points on a network with multicore support. For details, please see the function kfunctions. (maturing)

Usage

```
kfunctions.mc(
  lines,
  points,
  start,
  end,
  step,
  width,
  nsim,
  conf_int = 0.05,
  digits = 2,
  tol = 0.1,
  agg = NULL,
  verbose = TRUE,
  return_sims = FALSE,
  calc_g_func = TRUE,
  resolution = NULL,
  grid_shape = c(1, 1)
)
```

Arguments

`lines`	A feature collection of linestrings representing the underlying network. The geometries must be simple Linestrings (may crash if some geometries are invalid) without MultiLineSring
`points`	A feature collection of points representing the points on the network. These points will be snapped on their nearest line

`start`	A double, the lowest distance used to evaluate the k and g functions
`end`	A double, the highest distance used to evaluate the k and g functions
`step`	A double, the step between two evaluations of the k and g function. start, end and step are used to create a vector of distances with the function seq
`width`	The width of each donut for the g-function. Half of the width is applied on both sides of the considered distance
`nsim`	An integer indicating the number of Monte Carlo simulations to perform for inference
`conf_int`	A double indicating the width confidence interval (default = 0.05) calculated on the Monte Carlo simulations
`digits`	An integer indicating the number of digits to retain from the spatial coordinates
`tol`	When adding the points to the network, specify the minimum distance between these points and the lines' extremities. When points are closer, they are added at the extremity of the lines
`agg`	A double indicating if the events must be aggregated within a distance. If NULL, the events are aggregated only by rounding the coordinates
`verbose`	A Boolean indicating if progress messages should be displayed
`return_sims`	a boolean indicating if the simulated k and g values must also be returned.
`calc_g_func`	A Boolean indicating if the G function must also be calculated (TRUE by default). If FALSE, then only the K function is calculated
`resolution`	When simulating random points on the network, selecting a resolution will reduce greatly the calculation time. When resolution is null the random points can occur everywhere on the graph. If a value is specified, the edges are split according to this value and the random points can only be vertices on the new network
`grid_shape`	A vector of two values indicating how the study area must be split when performing the calculus. Default is c(1,1) (no split). A finer grid could reduce memory usage and increase speed when a large dataset is used. When using multiprocessing, the work in each grid is dispatched between the workers.

Details

For details, please look at the function kfunctions.

Value

A list with the following values :

- plotk: A ggplot2 object representing the values of the k-function
- plotg: A ggplot2 object representing the values of the g-function
- values: A DataFrame with the values used to build the plots

Examples

```
data(main_network_mtl)
data(mtl_libraries)
result <- kfunctions(main_network_mtl, mtl_libraries,
     start = 0, end = 2500, step = 10,
     width = 200, nsim = 50,
     conf_int = 0.05, tol = 0.1, agg = NULL,
     verbose = FALSE)
```

`kfunc_cpp` *c++ k function*

Description

c++ k function (INTERNAL)

Usage

```
kfunc_cpp(dist_mat, start, end, step, Lt, n, w)
```

Arguments

`dist_mat`	A square matrix with the distances between points
`start`	A float, the start value for evaluating the k-function
`end`	A float, the last value for evaluating the k-function
`step`	A float, the jump between two evaluations of the k-function
`Lt`	The total length of the network
`n`	The number of points
`w`	The weight of the points (coincident points)

Value

A numeric vector with the values of the k function evaluated at the required distances `k_nt_functions` *Network k and g functions for spatio-temporal data (experimental, NOT READY FOR USE)*

Description

Calculate the k and g functions for a set of points on a network and in time (experimental, NOT READY FOR USE).

Usage

```
k_nt_functions(
  lines,
  points,
  points_time,
  start_net,
  end_net,
  step_net,
  width_net,
  start_time,
  end_time,
  step_time,
  width_time,
  nsim,
  conf_int = 0.05,
  digits = 2,
  tol = 0.1,
  resolution = NULL,
  agg = NULL,
  verbose = TRUE,
  calc_g_func = TRUE
)
```

Arguments

`lines`	A feature collection of linestrings representing the underlying network. The geometries must be simple Linestrings (may crash if some geometries are invalid) without MultiLineSring
`points`	A feature collection of points representing the points on the network. These points will be snapped on their nearest line
`points_time`	A numeric vector indicating when the point occured
`start_net`	A double, the lowest network distance used to evaluate the k and g functions
`end_net`	A double, the highest network distance used to evaluate the k and g functions
`step_net`	A double, the step between two evaluations of the k and g for the network distance function. start_net, end_net and step_net are used to create a vector of distances with the function seq

`width_net`	The width (network distance) of each donut for the g-function. Half of the width is applied on both sides of the considered distance
`start_time`	A double, the lowest time distance used to evaluate the k and g functions
`end_time`	A double, the highest time distance used to evaluate the k and g functions
`step_time`	A double, the step between two evaluations of the k and g for the time distance function. start_time, end_time and step_time are used to create a vector of distances with the function seq
`width_time`	The width (time distance) of each donut for the g-function. Half of the width is applied on both sides of the considered distance
`nsim`	An integer indicating the number of Monte Carlo simulations to perform for inference
`conf_int`	A double indicating the width confidence interval (default = 0.05) calculated on the Monte Carlo simulations
`digits`	An integer indicating the number of digits to retain from the spatial coordinates
`tol`	When adding the points to the network, specify the minimum distance between these points and the lines' extremities. When points are closer, they are added at the extremity of the lines
`resolution`	When simulating random points on the network, selecting a resolution will reduce greatly the calculation time. When resolution is null the random points can occur everywhere on the graph. If a value is specified, the edges are split according to this value and the random points can only be vertices on the new network
`agg`	A double indicating if the events must be aggregated within a distance. If NULL, the events are aggregated only by rounding the coordinates
`verbose`	A Boolean indicating if progress messages should be displayed
`calc_g_func`	A boolean indicating if the G function must also be calculated

Details

The k-function is a method to characterize the dispersion of a set of points. For each point, the numbers of other points in subsequent radii are calculated in both space and time. This empirical k-function can be more or less clustered than a k-function obtained if the points were randomly located . In a network, the network distance is used instead of the Euclidean distance. This function uses Monte Carlo simulations to assess if the points are clustered or dispersed. The function also calculates the g-function, a modified version of the k-function using rings instead of disks. The width of the ring must be chosen. The main interest is to avoid the cumulative effect of the classical k-function. This function is maturing, it works as expected (unit tests) but will probably be modified in the future releases (gain speed, advanced features, etc.).

Value

A list with the following values :

- obs_k: A matrix with the observed k-values
- lower_k: A matrix with the lower bounds of the simulated k-values

- upper_k: A matrix with the upper bounds of the simulated k-values
- obs_g: A matrix with the observed g-values
- lower_g: A matrix with the lower bounds of the simulated g-values
- upper_g: A matrix with the upper bounds of the simulated g-values
- distances_net: A vector with the used network distances
- distances_time: A vector with the used time distances

Examples

```
data(mtl_network)
data(bike_accidents)

# converting the Date field to a numeric field (counting days)
bike_accidents$Time <- as.POSIXct(bike_accidents$Date, format = "%Y/%m/%d")
start <- as.POSIXct("2016/01/01", format = "%Y/%m/%d")
bike_accidents$Time <- difftime(bike_accidents$Time, start, units = "days")
bike_accidents$Time <- as.numeric(bike_accidents$Time)

values <- k_nt_functions(
      lines =  mtl_network,
      points = bike_accidents,
      points_time = bike_accidents$Time,
      start_net = 0 ,
      end_net = 2000,
      step_net = 10,
      width_net = 200,
      start_time = 0,
      end_time = 360,
      step_time = 7,
      width_time = 14,
      nsim = 50,
      conf_int = 0.05,
      digits = 2,
      tol = 0.1,
      resolution = NULL,
      agg = 15,
      verbose = TRUE)
```

`k_nt_functions.mc` *Network k and g functions for spatio-temporal data (multicore, experimental, NOT READY FOR USE)*

Description

Calculate the k and g functions for a set of points on a network and in time (multicore, experimental, NOT READY FOR USE).

Usage

```
k_nt_functions.mc(
  lines,
  points,
  points_time,
  start_net,
  end_net,
  step_net,
  width_net,
  start_time,
  end_time,
  step_time,
  width_time,
  nsim,
  conf_int = 0.05,
  digits = 2,
  tol = 0.1,
  resolution = NULL,
  agg = NULL,
  verbose = TRUE,
  calc_g_func = TRUE,
  grid_shape = c(1, 1)
)
```

Arguments

`lines`	A feature collection of linestrings representing the underlying network. The geometries must be simple Linestrings (may crash if some geometries are invalid) without MultiLineSring
`points`	A feature collection of points representing the points on the network. These points will be snapped on their nearest line
`points_time`	A numeric vector indicating when the point occured
`start_net`	A double, the lowest network distance used to evaluate the k and g functions
`end_net`	A double, the highest network distance used to evaluate the k and g functions
`step_net`	A double, the step between two evaluations of the k and g for the network distance function. start_net, end_net and step_net are used to create a vector of distances with the function seq
`width_net`	The width (network distance) of each donut for the g-function. Half of the width is applied on both sides of the considered distance
`start_time`	A double, the lowest time distance used to evaluate the k and g functions
`end_time`	A double, the highest time distance used to evaluate the k and g functions
`step_time`	A double, the step between two evaluations of the k and g for the time distance function. start_time, end_time and step_time are used to create a vector of distances with the function seq
`width_time`	The width (time distance) of each donut for the g-function. Half of the width is applied on both sides of the considered distance

`nsim`	An integer indicating the number of Monte Carlo simulations to perform for inference
`conf_int`	A double indicating the width confidence interval (default = 0.05) calculated on the Monte Carlo simulations
`digits`	An integer indicating the number of digits to retain from the spatial coordinates
`tol`	When adding the points to the network, specify the minimum distance between these points and the lines' extremities. When points are closer, they are added at the extremity of the lines
`resolution`	When simulating random points on the network, selecting a resolution will reduce greatly the calculation time. When resolution is null the random points can occur everywhere on the graph. If a value is specified, the edges are split according to this value and the random points can only be vertices on the new network
`agg`	A double indicating if the events must be aggregated within a distance. If NULL, the events are aggregated only by rounding the coordinates
`verbose`	A Boolean indicating if progress messages should be displayed
`calc_g_func`	A boolean indicating if the G function must also be calculated
`grid_shape`	A vector of two values indicating how the study area must be split when performing the calculus. Default is c(1,1) (no split). A finer grid could reduce memory usage and increase speed when a large dataset is used. When using multiprocessing, the work in each grid is dispatched between the workers.

Details

The k-function is a method to characterize the dispersion of a set of points. For each point, the numbers of other points in subsequent radii are calculated. This empirical k-function can be more or less clustered than a k-function obtained if the points were randomly located in space. In a network, the network distance is used instead of the Euclidean distance. This function uses Monte Carlo simulations to assess if the points are clustered or dispersed, and gives the results as a line plot. If the line of the observed k-function is higher than the shaded area representing the values of the simulations, then the points are more clustered than what we can expect from randomness and vice-versa. The function also calculates the g-function, a modified version of the k-function using rings instead of disks. The width of the ring must be chosen. The main interest is to avoid the cumulative effect of the classical k-function. This function is maturing, it works as expected (unit tests) but will probably be modified in the future releases (gain speed, advanced features, etc.).

Value

A list with the following values :

- obs_k: A matrix with the observed k-values
- lower_k: A matrix with the lower bounds of the simulated k-values
- upper_k: A matrix with the upper bounds of the simulated k-values
- obs_g: A matrix with the observed g-values
- lower_g: A matrix with the lower bounds of the simulated g-values

- upper_g: A matrix with the upper bounds of the simulated g-values
- distances_net: A vector with the used network distances
- distances_time: A vector with the used time distances

`lines_center` *Centre points of lines*

Description

Generate a feature collection of points at the centre of the lines of a feature collection of linestrings. The length of the lines is used to determine their centres.

Usage

```
lines_center(lines)
```

Arguments

`lines`	A feature collection of linestrings to use

Value

A feature collection of points

Examples

```
data(mtl_network)
centers <- lines_center(mtl_network)
```

`lines_direction` *Unify lines direction*

Description

A function to deal with the directions of lines. It ensures that only From-To situation are present by reverting To-From lines. For the lines labelled as To-From, the order of their vertices is reverted.

Usage

```
lines_direction(lines, field)
```

Arguments

`lines`	A sf object with linestring type geometries
`field`	Indicate a field giving information about authorized travelling direction on lines. if NULL, then all lines can be used in both directions. Must be the name of a column otherwise. The values of the column must be "FT" (From - To), "TF" (To - From) or "Both".

Value

A sf object with linestring type geometries

Examples

```
data(mtl_network)
mtl_network$length <- as.numeric(sf::st_length(mtl_network))
mtl_network$direction <- "Both"
mtl_network[6, "direction"] <- "TF"
mtl_network_directed <- lines_direction(mtl_network, "direction")
```

`lines_extremities` *Get lines extremities*

Description

Generate a feature collection of points with the first and last vertex of each line in a feature collection of linestrings.

Usage

```
lines_extremities(lines)
```

Arguments

`lines`	A feature collection of linestrings (simple Linestrings)

Value

A feature collection of points

Examples

```
wkt_lines <- c(
"LINESTRING (0 0, 1 0)",
"LINESTRING (1 0, 2 0)",
"LINESTRING (2 0, 3 0)",
"LINESTRING (0 1, 1 1)")

linesdf <- data.frame(wkt = wkt_lines,
                      id = paste("l",1:length(wkt_lines),sep=""))

all_lines <- sf::st_as_sf(linesdf, wkt = "wkt")
all_lines <- cbind(linesdf$wkt,all_lines)
points <- lines_extremities(all_lines)
```

`lines_points_along` *Points along lines*

Description

Generate a feature collection of points along the lines of feature collection of Linestrings.

Usage

```
lines_points_along(lines, dist)
```

Arguments

`lines`	A feature collection of linestrings to use
`dist`	The distance between the points along the lines

Value

A feature collection of points

Examples

```
data(mtl_network)
new_pts <- lines_points_along(mtl_network,50)
```

`lixelize_lines` *Cut lines into lixels*

Description

Cut the lines of a feature collection of linestrings into lixels with a specified minimal distance may fail if the line geometries are self intersecting.

Usage

```
lixelize_lines(lines, lx_length, mindist = NULL)
```

Arguments

`lines`	The sf object with linestring geometry type to modify
`lx_length`	The length of a lixel
`mindist`	The minimum length of a lixel. After cut, if the length of the final lixel is shorter than the minimum distance, then it is added to the previous lixel. if NULL, then mindist = maxdist/10. Note that the segments that are already shorter than the minimum distance are not modified.

Value

An sf object with linestring geometry type

Examples

```
data(mtl_network)
lixels <- lixelize_lines(mtl_network,150,50)
```

`lixelize_lines.mc` *Cut lines into lixels (multicore)*

Description

Cut the lines of a feature collection of linestrings into lixels with a specified minimal distance may fail if the line geometries are self intersecting with multicore support.

Usage

```
lixelize_lines.mc(
  lines,
  lx_length,
  mindist = NULL,
  verbose = TRUE,
  chunk_size = 100
)
```

Arguments

`lines`	A feature collection of linestrings to convert to lixels
`lx_length`	The length of a lixel
`mindist`	The minimum length of a lixel. After cut, if the length of the final lixel is shorter than the minimum distance, then it is added to the previous lixel. If NULL, then mindist = maxdist/10
`verbose`	A Boolean indicating if a progress bar must be displayed
`chunk_size`	The size of a chunk used for multiprocessing. Default is 100.

Value

A feature collection of linestrings

Examples

```
data(mtl_network)
future::plan(future::multisession(workers=1))
lixels <- lixelize_lines.mc(mtl_network,150,50)
## make sure any open connections are closed afterward
if (!inherits(future::plan(), "sequential")){
future::plan(future::sequential)
}
```

`main_network_mtl` *Primary road network of Montreal*

Description

A feature collection (sf object) representing the primary road network of Montreal. The EPSG is 3797, and the data comes from the Montreal OpenData website.

Usage

```
main_network_mtl
```

Format

A sf object with 2945 rows and 2 variables

TYPE the type of road

geom the geometry (linestrings)

Source

`https://donnees.montreal.ca/dataset/geobase`

`mtl_libraries` *Libraries of Montreal*

Description

A feature collection (sf object) representing the libraries of Montreal. The EPSG is 3797 and the data comes from the Montreal OpenData website.

Usage

```
mtl_libraries
```

Format

A sf object with 55 rows and 3 variables.

CP the postal code

NAME the name of the library

geom the geometry (points)

Source

https://donnees.montreal.ca/dataset/lieux-culturels

`mtl_network` *Road network of Montreal*

Description

A feature collection (sf object) representing the road network of Montreal. The EPSG is 3797, and the data comes from the Montreal OpenData website. It is only a small subset in central districts used to demonstrate the main functions of spNetwork.

Usage

```
mtl_network
```

Format

A sf object with 2945 rows and 2 variables

ClsRte the category of the road

geom the geometry (linestrings)

Source

https://donnees.montreal.ca/dataset/geobase

`mtl_theatres` *Theatres of Montreal*

Description

A feature collection (sf object) representing the theatres of Montreal. The EPSG is 3797 and the data comes from the Montreal OpenData website.

Usage

```
mtl_theatres
```

Format

A sf object with 54 rows and 3 variables.

CP the postal code

NAME the name of the theatre

geom the geometry (points)

Source

https://donnees.montreal.ca/dataset/lieux-culturels

`network_knn` *K-nearest points on network*

Description

Calculate the K-nearest points for a set of points on a network.

Usage

```
network_knn(
  origins,
  lines,
  k,
  destinations = NULL,
  maxdistance = 0,
  snap_dist = Inf,
  line_weight = "length",
  direction = NULL,
  grid_shape = c(1, 1),
  verbose = FALSE,
  digits = 3,
  tol = 0.1
)
```

Arguments

`origins`	A feature collection of points, for each point, its k nearest neighbours will be found on the network.
`lines`	A feature collection of linestrings representing the underlying network
`k`	An integer indicating the number of neighbours to find.
`destinations`	A feature collection of points, might be used if the neighbours must be found in a separate set of points NULL if the neighbours must be found in origins.
`maxdistance`	The maximum distance between two observations to consider them as neighbours. It is useful only if a grid is used, a lower value will reduce calculating time, but one must be sure that the k nearest neighbours are within this radius. Otherwise NAs will be present in the results.
`snap_dist`	The maximum distance to snap the start and end points on the network.
`line_weight`	The weighting to use for lines. Default is "length" (the geographical length), but can be the name of a column. The value is considered proportional to the geographical length of the lines.
`direction`	The name of a column indicating authorized travelling direction on lines. if NULL, then all lines can be used in both directions. Must be the name of a column otherwise. The values of the column must be "FT" (From - To), "TF" (To - From) or "Both".
`grid_shape`	A vector of length 2 indicating the shape of the grid to use for splitting the dataset. Default is c(1,1), so all the calculation is done in one go. It might be necessary to split it if the dataset is large.
`verbose`	A Boolean indicating if the function should print its progress
`digits`	The number of digits to retain from the spatial coordinates (simplification used to reduce risk of topological error)
`tol`	A float indicating the minimum distance between the points and the lines' extremities when adding the point to the network. When points are closer, they are added at the extremity of the lines.

Details

The k nearest neighbours of each point are found by using the network distance. The results could not be exact if some points share the exact same location. As an example, consider the following case. If A and B are two points at the exact same location, and C is a third point close to A and B. If the 1 nearest neighbour is requested for C, the function could return either A or B but not both. When such situation happens, a warning is raised by the function.

Value

A list with two matrices, one with the index of the neighbours and one with the distances.

Examples

```
data(main_network_mtl)
data(mtl_libraries)
```

```
    results <- network_knn(mtl_libraries, main_network_mtl,
        k = 3, maxdistance = 1000, line_weight = "length",
        grid_shape=c(1,1), verbose = FALSE)
```

---

network_knn.mc *K-nearest points on network (multicore version)*

Description

Calculate the K-nearest points for a set of points on a network with multicore support.

Usage

```
network_knn.mc(
  origins,
  lines,
  k,
  destinations = NULL,
  maxdistance = 0,
  snap_dist = Inf,
  line_weight = "length",
  direction = NULL,
  grid_shape = c(1, 1),
  verbose = FALSE,
  digits = 3,
  tol = 0.1
)
```

Arguments

origins	A feature collection of points, for each point, its k nearest neighbours will be found on the network.
lines	A feature collection of linestrings representing the underlying network
k	An integer indicating the number of neighbours to find.
destinations	A feature collection of points, might be used if the neighbours must be found in a separate set of points NULL if the neighbours must be found in origins.
maxdistance	The maximum distance between two observations to consider them as neighbours. It is useful only if a grid is used, a lower value will reduce calculating time, but one must be sure that the k nearest neighbours are within this radius. Otherwise NAs will be present in the results.
snap_dist	The maximum distance to snap the start and end points on the network.
line_weight	The weighting to use for lines. Default is "length" (the geographical length), but can be the name of a column. The value is considered proportional to the geographical length of the lines.

`direction`	The name of a column indicating authorized travelling direction on lines. if NULL, then all lines can be used in both directions. Must be the name of a column otherwise. The values of the column must be "FT" (From - To), "TF" (To - From) or "Both".
`grid_shape`	A vector of length 2 indicating the shape of the grid to use for splitting the dataset. Default is c(1,1), so all the calculation is done in one go. It might be necessary to split it if the dataset is large.
`verbose`	A Boolean indicating if the function should print its progress
`digits`	The number of digits to retain from the spatial coordinates (simplification used to reduce risk of topological error)
`tol`	A float indicating the minimum distance between the points and the lines' extremities when adding the point to the network. When points are closer, they are added at the extremity of the lines.

Value

A list with two matrices, one with the index of the neighbours and one with the distances.

Examples

```
data(main_network_mtl)
data(mtl_libraries)
future::plan(future::multisession(workers=1))
results <- network_knn.mc(mtl_libraries, main_network_mtl,
    k = 3, maxdistance = 1000, line_weight = "length",
    grid_shape=c(1,1), verbose = FALSE)
## make sure any open connections are closed afterward
if (!inherits(future::plan(), "sequential")) future::plan(future::sequential)
```

network_listw *Network distance listw*

Description

Generate listw object (spdep like) based on network distances.

Usage

```
network_listw(
  origins,
  lines,
  maxdistance,
  method = "centroid",
  point_dist = NULL,
  snap_dist = Inf,
  line_weight = "length",
  mindist = 10,
  direction = NULL,
  dist_func = "inverse",
  matrice_type = "B",
  grid_shape = c(1, 1),
  verbose = FALSE,
  digits = 3,
  tol = 0.1
)
```

Arguments

origins	A feature collection of lines, points, or polygons for which the spatial neighbouring list will be built
lines	A feature collection of lines representing the network
maxdistance	The maximum distance between two observations to consider them as neighbours.
method	A string indicating how the starting points will be built. If 'centroid' is used, then the centre of lines or polygons is used. If 'pointsalong' is used, then points will be placed along polygons' borders or along lines as starting and end points. If 'ends' is used (only for lines) the first and last vertices of lines are used as starting and ending points.
point_dist	A float, defining the distance between points when the method 'pointsalong' is selected.
snap_dist	The maximum distance to snap the start and end points on the network.
line_weight	The weighting to use for lines. Default is "length" (the geographical length), but can be the name of a column. The value is considered proportional to the geographical length of the lines.
mindist	The minimum distance between two different observations. It is important for it to be different from 0 when a W style is used.
direction	Indicates a field providing information about authorized travelling direction on lines. if NULL, then all lines can be used in both directions. Must be the name of a column otherwise. The values of the column must be "FT" (From - To), "TF" (To - From) or "Both".
dist_func	Indicates the function to use to convert the distance between observation in spatial weights. Can be 'identity', 'inverse', 'squared inverse' or a function with one parameter x that will be vectorized internally
matrice_type	The type of the weighting scheme. Can be 'B' for Binary, 'W' for row weighted, or 'I' (identity), see the documentation of spdep::nb2listw for details
grid_shape	A vector of length 2 indicating the shape of the grid to use for splitting the dataset. Default is c(1,1), so all the calculation is done in one go. It might be necessary to split it if the dataset is large.
verbose	A Boolean indicating if the function should print its progress
digits	The number of digits to retain in the spatial coordinates (simplification used to reduce risk of topological error)
tol	A float indicating the spatial tolerance when points are added as vertices to lines.

Value

A listw object (spdep like) if matrice_type is "B" or "W". If matrice_type is I, then a list with a nblist object and a list of weights is returned.

Examples

```
data(mtl_network)
listw <- network_listw(mtl_network,
    mtl_network,
    maxdistance = 500,
    method = "centroid",
    line_weight = "length",
    dist_func = 'squared inverse',
    matrice_type='B',
    grid_shape = c(2,2))
```

`network_listw.mc` *Network distance listw (multicore)*

Description

Generate listw object (spdep like) based on network distances with multicore support.

Usage

```
network_listw.mc(
  origins,
  lines,
  maxdistance,
  method = "centroid",
  point_dist = NULL,
  snap_dist = Inf,
  line_weight = "length",
  mindist = 10,
  direction = NULL,
  dist_func = "inverse",
  matrice_type = "B",
  grid_shape = c(1, 1),
  verbose = FALSE,
  digits = 3,
  tol = 0.1
)
```

Arguments

`origins`	A feature collection of linestrings, points or polygons for which the spatial neighbouring list will be built.
`lines`	A feature collection of linestrings representing the network
`maxdistance`	The maximum distance between two observations to consider them as neighbours.
`method`	A string indicating how the starting points will be built. If 'centroid' is used, then the centre of lines or polygons is used. If 'pointsalong' is used, then points will be placed along polygons' borders or along lines as starting and end points. If 'ends' is used (only for lines) the first and last vertices of lines are used as starting and ending points.
`point_dist`	A float, defining the distance between points when the method pointsalong is selected.
`snap_dist`	the maximum distance to snap the start and end points on the network.
`line_weight`	The weights to use for lines. Default is "length" (the geographical length), but can be the name of a column. The value is considered proportional with the geographical length of the lines.
`mindist`	The minimum distance between two different observations. It is important for it to be different from 0 when a W style is used.
`direction`	Indicates a field giving information about authorized travelling direction on lines. if NULL, then all lines can be used in both directions. Must be the name of a column otherwise. The values of the column must be "FT" (From - To), "TF" (To - From) or "Both".
`dist_func`	Indicates the function to use to convert the distance between observation in spatial weights. Can be 'identity', 'inverse', 'squared inverse' or a function with one parameter x that will be vectorized internally
`matrice_type`	The type of the weighting scheme. Can be 'B' for Binary, 'W' for row weighted, or 'I' (identity) see the documentation of spdep::nb2listw for details
`grid_shape`	A vector of length 2 indicating the shape of the grid to use for splitting the dataset. Default is c(1,1), so all the calculation is done in one go. It might be necessary to split it if the dataset is large.
`verbose`	A Boolean indicating if the function should print its progress
`digits`	The number of digits to retain in the spatial coordinates (simplification used to reduce risk of topological error)
`tol`	A float indicating the spatial tolerance when points are added as vertices to lines.

Value

A listw object (spdep like) if matrice_type is "B" or "W". If matrice_type is I, then a list with a nblist object and a list of weights is returned.

Examples

```
data(mtl_network)
future::plan(future::multisession(workers=1))
listw <- network_listw.mc(mtl_network,mtl_network,maxdistance=500,
        method = "centroid", line_weight = "length",
        dist_func = 'squared inverse', matrice_type='B', grid_shape = c(2,2))
## make sure any open connections are closed afterward
if (!inherits(future::plan(), "sequential")) future::plan(future::sequential)
```

`nkde` *Network Kernel density estimate*

Description

Calculate the Network Kernel Density Estimate based on a network of lines, sampling points, and events

Usage

```
nkde(
  lines,
  events,
  w,
  samples,
  kernel_name,
  bw,
  adaptive = FALSE,
  trim_bw = NULL,
  method,
  div = "bw",
  diggle_correction = FALSE,
  study_area = NULL,
  max_depth = 15,
  digits = 5,
  tol = 0.1,
  agg = NULL,
  sparse = TRUE,
  grid_shape = c(1, 1),
  verbose = TRUE,
  check = TRUE
)
```

Arguments

`lines`	A feature collection of linestrings representing the underlying network. The geometries must be simple Linestrings (may crash if some geometries are invalid) without MultiLineSring.

`events`	events A feature collection of points representing the events on the network. The points will be snapped on the network to their closest line.
`w`	A vector representing the weight of each event
`samples`	A feature collection of points representing the locations for which the densities will be estimated.
`kernel_name`	The name of the kernel to use. Must be one of triangle, gaussian, tricube, cosine, triweight, quartic, epanechnikov or uniform.
`bw`	The kernel bandwidth (using the scale of the lines), can be a single float or a numeric vector if a different bandwidth must be used for each event.
`adaptive`	A Boolean, indicating if an adaptive bandwidth must be used
`trim_bw`	A float, indicating the maximum value for the adaptive bandwidth
`method`	The method to use when calculating the NKDE, must be one of simple / discontinuous / continuous (see nkde details for more information)
`div`	The divisor to use for the kernel. Must be "n" (the number of events within the radius around each sampling point), "bw" (the bandwidth) "none" (the simple sum).
`diggle_correction`	A Boolean indicating if the correction factor for edge effect must be used.
`study_area`	A feature collection of polygons representing the limits of the study area.
`max_depth`	when using the continuous and discontinuous methods, the calculation time and memory use can go wild if the network has many small edges (area with many of intersections and many events). To avoid it, it is possible to set here a maximum depth. Considering that the kernel is divided at intersections, a value of 10 should yield good estimates in most cases. A larger value can be used without a problem for the discontinuous method. For the continuous method, a larger value will strongly impact calculation speed.
`digits`	The number of digits to retain from the spatial coordinates. It ensures that topology is good when building the network. Default is 3. Too high a precision (high number of digits) might break some connections
`tol`	A float indicating the minimum distance between the events and the lines' extremities when adding the point to the network. When points are closer, they are added at the extremity of the lines.
`agg`	A double indicating if the events must be aggregated within a distance. If NULL, the events are aggregated only by rounding the coordinates.
`sparse`	A Boolean indicating if sparse or regular matrices should be used by the Rcpp functions. These matrices are used to store edge indices between two nodes in a graph. Regular matrices are faster, but require more memory, in particular with multiprocessing. Sparse matrices are slower (a bit), but require much less memory.
`grid_shape`	A vector of two values indicating how the study area must be split when performing the calculus. Default is c(1,1) (no split). A finer grid could reduce memory usage and increase speed when a large dataset is used. When using multiprocessing, the work in each grid is dispatched between the workers.
`verbose`	A Boolean, indicating if the function should print messages about the process.

`check`	A Boolean indicating if the geometry checks must be run before the operation. This might take some times, but it will ensure that the CRS of the provided objects are valid and identical, and that geometries are valid.

Details

The three NKDE methods

Estimating the density of a point process is commonly done by using an ordinary two-dimensional kernel density function. However, there are numerous cases for which the events do not occur in a two-dimensional space but on a network (like car crashes, outdoor crimes, leaks in pipelines, etc.). New methods were developed to adapt the methodology to networks, three of them are available in this package.

- The simple method: This first method was presented by (Xie and Yan 2008) and proposes an intuitive solution. The distances between events and sampling points are replaced by network distances, and the formula of the kernel is adapted to calculate the density over a linear unit instead of an areal unit.
- The discontinuous method: The previous method has been criticized by (Okabe et al. 2009), arguing that the estimator proposed is biased, leading to an overestimation of density in events hot-spots. More specifically, the simple method does not conserve mass and the induced kernel is not a probability density along the network. They thus proposed a discontinuous version of the kernel function on network, which equally "divides" the mass density of an event at intersections.
- The continuous method: If the discontinuous method is unbiased, it leads to a discontinuous kernel function which is a bit counter-intuitive. Okabe et al. (2009) proposed another version of the kernel, which divides the mass of the density at intersections but adjusts the density before the intersection to make the function continuous.

The three methods are available because, even though that the simple method is less precise statistically speaking, it might be more intuitive. From a purely geographical view, it might be seen as a sort of distance decay function as used in Geographically Weighted Regression.

adaptive bandwidth

It is possible to use adaptive bandwidth instead of fixed bandwidth. Adaptive bandwidths are calculated using the Abramson's smoothing regimen (Abramson 1982). To do so, an original fixed bandwidth must be specified (bw parameter), and is used to estimate the priory densitiy at event locations. These densities are then used to calculate local bandwidth. The maximum size of the local bandwidth can be limited with the parameter trim_bw. For more details, see the vignettes.

Optimization parameters

The grid_shape parameter allows to split the calculus of the NKDE according to a grid dividing the study area. It might be necessary for big dataset to reduce the memory used. If the grid_shape is c(1,1), then a full network is built for the area. If the grid_shape is c(2,2), then the area is split in 4 rectangles. For each rectangle, the sample points falling in the rectangle are used, the events and the lines in a radius of the bandwidth length are used. The results are combined at the end and ordered to match the original order of the samples.

The geographical coordinates of the start and end of lines are used to build the network. To avoid troubles with digits, we truncate the coordinates according to the digit parameter. A minimal loss of precision is expected but results in a fast construction of the network.

To calculate the distances on the network, all the events are added as vertices. To reduce the size of the network, it is possible to reduce the number of vertices by adding the events at the extremity of the lines if they are close to them. This is controlled by the parameter tol.

In the same way, it is possible to limit the number of vertices by aggregating the events that are close to each other. In that case, the weights of the aggregated events are summed. According to an aggregation distance, a buffer is drawn around the fist event, all events falling in that buffer are aggregated to the first event, forming a new event. The coordinates of this new event are the means of the original events coordinates. This procedure is repeated until no events are aggregated. The aggregation distance can be fixed with the parameter agg.

When using the continuous and discontinuous kernel, the density is reduced at each intersection crossed. In the discontinuous case, after 5 intersections with four directions each, the density value is divided by 243 leading to very small values. In the same situation but with the continuous NKDE, the density value is divided by approximately 7.6. The max_depth parameters allows the user to control the maximum depth of these two NKDE. The base value is 15, but a value of 10 would yield very close estimates. A lower value might have a critical impact on speed when the bandwidth is large.

When using the continuous and discontinuous kernel, the connections between graph nodes are stored in a matrix. This matrix is typically sparse, and so a sparse matrix object is used to limit memory use. If the network is small (typically when the grid used to split the data has small rectangles) then a classical matrix could be used instead of a sparse one. It significantly increases speed, but could lead to memory issues.

Value

A vector of values, they are the density estimates at sampling points

References

Abramson IS (1982). “On bandwidth variation in kernel estimates-a square root law.” *The annals of Statistics*, 1217–1223.

Okabe A, Satoh T, Sugihara K (2009). “A kernel density estimation method for networks, its computational method and a GIS-based tool.” *International Journal of Geographical Information Science*, **23**(1), 7–32.

Xie Z, Yan J (2008). “Kernel density estimation of traffic accidents in a network space.” *Computers, environment and urban systems*, **32**(5), 396–406.

Examples

```
data(mtl_network)
data(bike_accidents)
lixels <- lixelize_lines(mtl_network,200,mindist = 50)
```

```
samples <- lines_center(lixels)
densities <- nkde(mtl_network,
                  events = bike_accidents,
                  w = rep(1,nrow(bike_accidents)),
                  samples = samples,
                  kernel_name = "quartic",
                  bw = 300, div= "bw",
                  adaptive = FALSE,
                  method = "discontinuous", digits = 1, tol = 1,
                  agg = 15,
                  grid_shape = c(1,1),
                  verbose=FALSE)
```

nkde.mc *Network Kernel density estimate (multicore)*

Description

Calculate the Network Kernel Density Estimate based on a network of lines, sampling points, and events with multicore support.

Usage

```
nkde.mc(
  lines,
  events,
  w,
  samples,
  kernel_name,
  bw,
  adaptive = FALSE,
  trim_bw = NULL,
  method,
  div = "bw",
  diggle_correction = FALSE,
  study_area = NULL,
  max_depth = 15,
  digits = 5,
  tol = 0.1,
  agg = NULL,
  sparse = TRUE,
  grid_shape = c(1, 1),
  verbose = TRUE,
  check = TRUE
)
```

Arguments

`lines`	A feature collection of linestrings representing the underlying network. The geometries must be simple Linestrings (may crash if some geometries are invalid) without MultiLineSring.
`events`	events A feature collection of points representing the events on the network. The points will be snapped on the network to their closest line.
`w`	A vector representing the weight of each event
`samples`	A feature collection of points representing the locations for which the densities will be estimated.
`kernel_name`	The name of the kernel to use. Must be one of triangle, gaussian, tricube, cosine, triweight, quartic, epanechnikov or uniform.
`bw`	The kernel bandwidth (using the scale of the lines), can be a single float or a numeric vector if a different bandwidth must be used for each event.
`adaptive`	A Boolean, indicating if an adaptive bandwidth must be used
`trim_bw`	A float, indicating the maximum value for the adaptive bandwidth
`method`	The method to use when calculating the NKDE, must be one of simple / discontinuous / continuous (see nkde details for more information)
`div`	The divisor to use for the kernel. Must be "n" (the number of events within the radius around each sampling point), "bw" (the bandwidth) "none" (the simple sum).
`diggle_correction`	A Boolean indicating if the correction factor for edge effect must be used.
`study_area`	A feature collection of polygons representing the limits of the study area.
`max_depth`	when using the continuous and discontinuous methods, the calculation time and memory use can go wild if the network has many small edges (area with many of intersections and many events). To avoid it, it is possible to set here a maximum depth. Considering that the kernel is divided at intersections, a value of 10 should yield good estimates in most cases. A larger value can be used without a problem for the discontinuous method. For the continuous method, a larger value will strongly impact calculation speed.
`digits`	The number of digits to retain from the spatial coordinates. It ensures that topology is good when building the network. Default is 3. Too high a precision (high number of digits) might break some connections
`tol`	A float indicating the minimum distance between the events and the lines' extremities when adding the point to the network. When points are closer, they are added at the extremity of the lines.
`agg`	A double indicating if the events must be aggregated within a distance. If NULL, the events are aggregated only by rounding the coordinates.
`sparse`	A Boolean indicating if sparse or regular matrices should be used by the Rcpp functions. These matrices are used to store edge indices between two nodes in a graph. Regular matrices are faster, but require more memory, in particular with multiprocessing. Sparse matrices are slower (a bit), but require much less memory.

`grid_shape`	A vector of two values indicating how the study area must be split when performing the calculus. Default is c(1,1) (no split). A finer grid could reduce memory usage and increase speed when a large dataset is used. When using multiprocessing, the work in each grid is dispatched between the workers.
`verbose`	A Boolean, indicating if the function should print messages about the process.
`check`	A Boolean indicating if the geometry checks must be run before the operation. This might take some times, but it will ensure that the CRS of the provided objects are valid and identical, and that geometries are valid.

Details

For more details, see help(nkde)

Value

A vector of values, they are the density estimates at sampling points

Examples

```
data(mtl_network)
data(bike_accidents)
future::plan(future::multisession(workers=1))
lixels <- lixelize_lines(mtl_network,200,mindist = 50)
samples <- lines_center(lixels)
densities <- nkde.mc(mtl_network,
                  events = bike_accidents,
                  w = rep(1,nrow(bike_accidents)),
                  samples = samples,
                  kernel_name = "quartic",
                  bw = 300, div= "bw",
                  adaptive = FALSE, agg = 15,
                  method = "discontinuous", digits = 1, tol = 1,
                  grid_shape = c(3,3),
                  verbose=TRUE)
## make sure any open connections are closed afterward
if (!inherits(future::plan(), "sequential")) future::plan(future::sequential)
```

`nkde_get_loo_values` *The exposed function to calculate NKDE likelihood cv*

Description

The exposed function to calculate NKDE likelihood cv (INTERNAL)

Usage

```
nkde_get_loo_values(
  method,
  neighbour_list,
  sel_events,
  sel_events_wid,
  events,
  events_wid,
  weights,
  bws_net,
  kernel_name,
  line_list,
  max_depth,
  cvl
)
```

Arguments

`method`	a string, one of "simple", "continuous", "discontinuous"
`neighbour_list`	a List, giving for each node an IntegerVector with its neighbours
`sel_events`	a Numeric vector indicating the selected events (id of nodes)
`sel_events_wid`	a Numeric Vector indicating the unique if of the selected events
`events`	a NumericVector indicating the nodes in the graph being events
`events_wid`	a NumericVector indicating the unique id of all the events
`weights`	a matrix with the weights associated with each event (row) for each bws_net (cols).
`bws_net`	an arma::mat with the network bandwidths to consider for each event
`kernel_name`	a string with the name of the kernel to use
`line_list`	a DataFrame describing the lines
`max_depth`	the maximum recursion depth
`cvl`	a boolean indicating if the Cronie (TRUE) or CV likelihood (FALSE) must be used

Value

a vector with the CV score for each bandwidth and the densities if required

Examples

```
# no example provided, this is an internal function
```

`quartic_kernel` *Quartic kernel*

Description

Function implementing the quartic kernel.

Usage

```
quartic_kernel(d, bw)
```

Arguments

`d`	The distance from the event
`bw`	The bandwidth used for the kernel

Value

The estimated density

Examples

```
#This is an internal function, no example provided
```

`simple_lines` *LineString to simple Line*

Description

Split the polylines of a feature collection of linestrings in simple segments at each vertex. The values of the columns are duplicated for each segment.

Usage

```
simple_lines(lines)
```

Arguments

`lines`	The featue collection of linestrings to modify

Value

An featue collection of linestrings

Examples

```
data(mtl_network)
new_lines <- simple_lines(mtl_network)
```

simplify_network *Simplify a network*

Description

Simplify a network by applying two corrections: Healing edges and Removing mirror edges (experimental).

Usage

```
simplify_network(
  lines,
  digits = 3,
  heal = TRUE,
  mirror = TRUE,
  keep_shortest = TRUE,
  verbose = TRUE
)
```

Arguments

`lines`	A feature collection of linestrings
`digits`	An integer indicating the number of digits to keep in coordinates
`heal`	A boolean indicating if the healing operation must be performed
`mirror`	A boolean indicating if the mirror edges must be removed
`keep_shortest`	A boolean, if TRUE, then the shortest line is kept from mirror edges. if FALSE, then the longest line is kept.
`verbose`	A boolean indicating if messages and a progress bar should be displayed

Details

Healing is the operation to merge two connected linestring if the are intersecting at one extremity and do not intersect any other linestring. It helps to reduce the complexity of the network and thus can reduce calculation time. Removing mirror edges is the operation to remove edges that have the same extremities. If two edges start at the same point and end at the same point, they do not add information in the network and one can be removed to simplify the network. One can decide to keep the longest of the two edges or the shortest. NOTE: the edge healing does not consider lines directions currently!

Value

A feature collection of linestrings

Examples

```
data(mtl_network)
edited_lines <- simplify_network(mtl_network, digits = 3, verbose = FALSE)
```

`small_mtl_network` *Smaller subset road network of Montreal*

Description

A feature collection (sf object) representing the road network of Montreal. The EPSG is 3797, and the data comes from the Montreal OpenData website. It is only a small extract in central districts used to demonstrate the main functions of spNetwork. It is mainly used internally for tests.

Usage

```
small_mtl_network
```

Format

A sf object with 1244 rows and 2 variables

TYPE the type of road

geom the geometry (linestrings)

Source

https://donnees.montreal.ca/dataset/geobase

`split_graph_components` *Split graph components*

Description

Function to split the results of build_graph and build_graph_directed into their sub components

Usage

```
split_graph_components(graph_result)
```

Arguments

`graph_result`	A list typically obtained from the function build_graph or build_graph_directed

Value

A list of lists, the graph_result split for each graph component

Examples

```
data(mtl_network)
mtl_network$length <- as.numeric(sf::st_length(mtl_network))
graph_result <- build_graph(mtl_network, 2, "length", attrs = TRUE)
sub_elements <- split_graph_components(graph_result)
```

`split_lines_at_vertex` *Split lines at vertices in a feature collection of linestrings*

Description

Split lines (feature collection of linestrings) at their nearest vertices (feature collection of points), may fail if the line geometries are self intersecting.

Usage

```
split_lines_at_vertex(lines, points, nearest_lines_idx, mindist)
```

Arguments

`lines`	The feature collection of linestrings to split
`points`	The feature collection of points to add to as vertex to the lines
`nearest_lines_idx`	For each point, the index of the nearest line
`mindist`	The minimum distance between one point and the extremity of the line to add the point as a vertex.

Value

A feature collection of linestrings

Examples

```
# reading the data
data(mtl_network)
data(bike_accidents)
# aggregating points within a 5 metres radius
bike_accidents$weight <- 1
agg_points <- aggregate_points(bike_accidents, 5)
mtl_network$LineID <- 1:nrow(mtl_network)
# snapping point to lines
snapped_points <- snapPointsToLines2(agg_points,
    mtl_network,
    "LineID"
)
# splitting lines
new_lines <- split_lines_at_vertex(mtl_network, snapped_points,
    snapped_points$nearest_line_id, 1)
```

st_bbox_by_feature *Obtain all the bounding boxes of a feature collection*

Description

Obtain all the bounding boxes of a feature collection (INTERNAL).

Usage

```
st_bbox_by_feature(x)
```

Arguments

x	a feature collection

Value

a matrix (xmin, ymin, xmax, ymax)

Examples

```
#This is an internal function, no example provided
```

tkde *Temporal Kernel density estimate*

Description

Calculate the Temporal kernel density estimate based on sampling points in time and events

Usage

```
tkde(events, w, samples, bw, kernel_name, adaptive = FALSE)
```

Arguments

events	A numeric vector representing the moments of occurrence of events
w	The weight of the events
samples	A numeric vector representing the moments to sample
bw	A float, the bandwidth to use
kernel_name	The name of the kernel to use
adaptive	Boolean

Value

A numeric vector with the density values at the requested timestamps

Examples

```
data(bike_accidents)
bike_accidents$Date <- as.POSIXct(bike_accidents$Date, format = "%Y/%m/%d")
start <- min(bike_accidents$Date)
diff <- as.integer(difftime(bike_accidents$Date , start, units = "days"))
density <- tkde(diff, rep(1,length(diff)), seq(0,max(diff),1), 2, "quartic")
```

---

tnkde *Temporal Network Kernel density estimate*

Description

Calculate the Temporal Network Kernel Density Estimate based on a network of lines, sampling points in space and times, and events in space and time.

Usage

```
tnkde(
  lines,
  events,
  time_field,
  w,
  samples_loc,
  samples_time,
  kernel_name,
  bw_net,
  bw_time,
  adaptive = FALSE,
  adaptive_separate = TRUE,
  trim_bw_net = NULL,
  trim_bw_time = NULL,
  method,
  div = "bw",
  diggle_correction = FALSE,
  study_area = NULL,
  max_depth = 15,
  digits = 5,
  tol = 0.1,
  agg = NULL,
  sparse = TRUE,
  grid_shape = c(1, 1),
  verbose = TRUE,
  check = TRUE
)
```

Arguments

`lines`	A feature collection of linestrings representing the underlying network. The geometries must be simple Linestrings (may crash if some geometries are invalid) without MultiLineSring.
`events`	events A feature collection of points representing the events on the network. The points will be snapped on the network to their closest line.
`time_field`	The name of the field in events indicating when the events occurred. It must be a numeric field
`w`	A vector representing the weight of each event
`samples_loc`	A feature collection of points representing the locations for which the densities will be estimated.
`samples_time`	A numeric vector indicating when the densities will be sampled
`kernel_name`	The name of the kernel to use. Must be one of triangle, gaussian, tricube, cosine, triweight, quartic, epanechnikov or uniform.
`bw_net`	The network kernel bandwidth (using the scale of the lines), can be a single float or a numeric vector if a different bandwidth must be used for each event.
`bw_time`	The time kernel bandwidth, can be a single float or a numeric vector if a different bandwidth must be used for each event.
`adaptive`	A Boolean, indicating if an adaptive bandwidth must be used. Both spatial and temporal bandwidths are adapted but separately.
`adaptive_separate`	A boolean indicating if the adaptive bandwidths for the time and the network dimensions must be calculated separately (TRUE) or in interaction (FALSE)
`trim_bw_net`	A float, indicating the maximum value for the adaptive network bandwidth
`trim_bw_time`	A float, indicating the maximum value for the adaptive time bandwidth
`method`	The method to use when calculating the NKDE, must be one of simple / discontinuous / continuous (see nkde details for more information)
`div`	The divisor to use for the kernel. Must be "n" (the number of events within the radius around each sampling point), "bw" (the bandwith) "none" (the simple sum).
`diggle_correction`	A Boolean indicating if the correction factor for edge effect must be used.
`study_area`	A feature collection of polygons representing the limits of the study area.
`max_depth`	when using the continuous and discontinuous methods, the calculation time and memory use can go wild if the network has many small edges (area with many of intersections and many events). To avoid it, it is possible to set here a maximum depth. Considering that the kernel is divided at intersections, a value of 10 should yield good estimates in most cases. A larger value can be used without a problem for the discontinuous method. For the continuous method, a larger value will strongly impact calculation speed.
`digits`	The number of digits to retain from the spatial coordinates. It ensures that topology is good when building the network. Default is 3. Too high a precision (high number of digits) might break some connections

`tol`	A float indicating the minimum distance between the events and the lines' extremities when adding the point to the network. When points are closer, they are added at the extremity of the lines.
`agg`	A double indicating if the events must be aggregated within a distance. If NULL, the events are aggregated only by rounding the coordinates.
`sparse`	A Boolean indicating if sparse or regular matrices should be used by the Rcpp functions. These matrices are used to store edge indices between two nodes in a graph. Regular matrices are faster, but require more memory, in particular with multiprocessing. Sparse matrices are slower (a bit), but require much less memory.
`grid_shape`	A vector of two values indicating how the study area must be split when performing the calculus. Default is c(1,1) (no split). A finer grid could reduce memory usage and increase speed when a large dataset is used. When using multiprocessing, the work in each grid is dispatched between the workers.
`verbose`	A Boolean, indicating if the function should print messages about the process.
`check`	A Boolean indicating if the geometry checks must be run before the operation. This might take some times, but it will ensure that the CRS of the provided objects are valid and identical, and that geometries are valid.

Details

Temporal Network Kernel Density Estimate

The TNKDE is an extension of the NKDE considering both the location of events on the network and in time. Thus, density estimation (density sampling) can be done along lines of the network and at different time. It can be used with the three NKDE (simple, discontinuous and continuous).

density in time and space

Two bandwidths must be provided, one for the network distance and one for the time distance. They are both used to calculate the contribution of each event to each sampling point. Let us consider one event E and a sample S. dnet(E,S) is the contribution to network density of E at S location and dtime(E,S) is the contribution to time density of E at S time. The total contribution is thus dnet(E,S) * dtime(E,S). If one of the two densities is 0, then the total density is 0 because the sampling point is out of the covered area by the event in time or in the network space.

adaptive bandwidth

It is possible to use an adaptive bandwidth both on the network and in time. Adaptive bandwidths are calculated using the Abramson's smoothing regimen (Abramson 1982). To do so, the original fixed bandwidths must be specified (bw_net and bw_time parameters). The maximum size of the two local bandwidths can be limited with the parameters trim_bw_net and trim_bw_time.

Diggle correction factor

A set of events can be limited in both space (limits of the study area) and time (beginning and ending of the data collection period). These limits induce lower densities at the border of the set of events, because they are not sampled outside the limits. It is possible to apply the Diggle correction factor (Diggle 1985) in both the network and time spaces to minimize this effect.

Separated or simultaneous adaptive bandwidth
When the parameter adaptive is TRUE, one can choose between using separated calculation of network and temporal bandwidths, and calculating them simultaneously. In the first case (default), the network bandwidths are determined for each event by considering only their locations and the time bandwidths are determined by considering only there time stamps. In the second case, for each event, the spatio-temporal density at its location on the network and in time is estimated and used to determine both the network and temporal bandwidths. This second approach must be preferred if the events are characterized by a high level of spatio-temporal autocorrelation.

Value

A matrix with the estimated density for each sample point (rows) at each timestamp (columns). If adaptive = TRUE, the function returns a list with two slots: k (the matrix with the density values) and events (a feature collection of points with the local bandwidths).

Examples

```
# loading the data
data(mtl_network)
data(bike_accidents)

# converting the Date field to a numeric field (counting days)
bike_accidents$Time <- as.POSIXct(bike_accidents$Date, format = "%Y/%m/%d")
start <- as.POSIXct("2016/01/01", format = "%Y/%m/%d")
bike_accidents$Time <- difftime(bike_accidents$Time, start, units = "days")
bike_accidents$Time <- as.numeric(bike_accidents$Time)

# creating sample points
lixels <- lixelize_lines(mtl_network, 50)
sample_points <- lines_center(lixels)

# choosing sample in times (every 10 days)
sample_time <- seq(0, max(bike_accidents$Time), 10)

# calculating the densities
tnkde_densities <- tnkde(lines = mtl_network,
    events = bike_accidents, time_field = "Time",
    w = rep(1, nrow(bike_accidents)),
    samples_loc = sample_points,
    samples_time = sample_time,
    kernel_name = "quartic",
    bw_net = 700, bw_time = 60, adaptive = TRUE,
    trim_bw_net = 900, trim_bw_time = 80,
    method = "discontinuous", div = "bw",
    max_depth = 10, digits = 2, tol = 0.01,
    agg = 15, grid_shape = c(1,1),
    verbose  = FALSE)
```

tnkde.mc *Temporal Network Kernel density estimate (multicore)*

Description

Calculate the Temporal Network Kernel Density Estimate based on a network of lines, sampling points in space and times, and events in space and time with multicore support.

Usage

```
tnkde.mc(
  lines,
  events,
  time_field,
  w,
  samples_loc,
  samples_time,
  kernel_name,
  bw_net,
  bw_time,
  adaptive = FALSE,
  adaptive_separate = TRUE,
  trim_bw_net = NULL,
  trim_bw_time = NULL,
  method,
  div = "bw",
  diggle_correction = FALSE,
  study_area = NULL,
  max_depth = 15,
  digits = 5,
  tol = 0.1,
  agg = NULL,
  sparse = TRUE,
  grid_shape = c(1, 1),
  verbose = TRUE,
  check = TRUE
)
```

Arguments

`lines`	A feature collection of linestrings representing the underlying network. The geometries must be simple Linestrings (may crash if some geometries are invalid) without MultiLineSring.
`events`	events A feature collection of points representing the events on the network. The points will be snapped on the network to their closest line.
`time_field`	The name of the field in events indicating when the events occurred. It must be a numeric field

`w`	A vector representing the weight of each event
`samples_loc`	A feature collection of points representing the locations for which the densities will be estimated.
`samples_time`	A numeric vector indicating when the densities will be sampled
`kernel_name`	The name of the kernel to use. Must be one of triangle, gaussian, tricube, cosine, triweight, quartic, epanechnikov or uniform.
`bw_net`	The network kernel bandwidth (using the scale of the lines), can be a single float or a numeric vector if a different bandwidth must be used for each event.
`bw_time`	The time kernel bandwidth, can be a single float or a numeric vector if a different bandwidth must be used for each event.
`adaptive`	A Boolean, indicating if an adaptive bandwidth must be used. Both spatial and temporal bandwidths are adapted but separately.
`adaptive_separate`	A boolean indicating if the adaptive bandwidths for the time and the network dimensions must be calculated separately (TRUE) or in interaction (FALSE)
`trim_bw_net`	A float, indicating the maximum value for the adaptive network bandwidth
`trim_bw_time`	A float, indicating the maximum value for the adaptive time bandwidth
`method`	The method to use when calculating the NKDE, must be one of simple / discontinuous / continuous (see nkde details for more information)
`div`	The divisor to use for the kernel. Must be "n" (the number of events within the radius around each sampling point), "bw" (the bandwith) "none" (the simple sum).
`diggle_correction`	A Boolean indicating if the correction factor for edge effect must be used.
`study_area`	A feature collection of polygons representing the limits of the study area.
`max_depth`	when using the continuous and discontinuous methods, the calculation time and memory use can go wild if the network has many small edges (area with many of intersections and many events). To avoid it, it is possible to set here a maximum depth. Considering that the kernel is divided at intersections, a value of 10 should yield good estimates in most cases. A larger value can be used without a problem for the discontinuous method. For the continuous method, a larger value will strongly impact calculation speed.
`digits`	The number of digits to retain from the spatial coordinates. It ensures that topology is good when building the network. Default is 3. Too high a precision (high number of digits) might break some connections
`tol`	A float indicating the minimum distance between the events and the lines' extremities when adding the point to the network. When points are closer, they are added at the extremity of the lines.
`agg`	A double indicating if the events must be aggregated within a distance. If NULL, the events are aggregated only by rounding the coordinates.
`sparse`	A Boolean indicating if sparse or regular matrices should be used by the Rcpp functions. These matrices are used to store edge indices between two nodes in a graph. Regular matrices are faster, but require more memory, in particular with multiprocessing. Sparse matrices are slower (a bit), but require much less memory.

`grid_shape`	A vector of two values indicating how the study area must be split when performing the calculus. Default is c(1,1) (no split). A finer grid could reduce memory usage and increase speed when a large dataset is used. When using multiprocessing, the work in each grid is dispatched between the workers.
`verbose`	A Boolean, indicating if the function should print messages about the process.
`check`	A Boolean indicating if the geometry checks must be run before the operation. This might take some times, but it will ensure that the CRS of the provided objects are valid and identical, and that geometries are valid.

Details

For details, see help(tnkde) and help(nkde)

Value

A matrix with the estimated density for each sample point (rows) at each timestamp (columns). If adaptive = TRUE, the function returns a list with two slots: k (the matrix with the density values) and events (a feature collection of points with the local bandwidths).

Examples

```
# loading the data
data(mtl_network)
data(bike_accidents)

# converting the Date field to a numeric field (counting days)
bike_accidents$Time <- as.POSIXct(bike_accidents$Date, format = "%Y/%m/%d")
start <- as.POSIXct("2016/01/01", format = "%Y/%m/%d")
bike_accidents$Time <- difftime(bike_accidents$Time, start, units = "days")
bike_accidents$Time <- as.numeric(bike_accidents$Time)

# creating sample points
lixels <- lixelize_lines(mtl_network, 50)
sample_points <- lines_center(lixels)

# choosing sample in times (every 10 days)
sample_time <- seq(0, max(bike_accidents$Time), 10)

future::plan(future::multisession(workers=1))

# calculating the densities
tnkde_densities <- tnkde.mc(lines = mtl_network,
    events = bike_accidents, time_field = "Time",
    w = rep(1, nrow(bike_accidents)),
    samples_loc = sample_points,
    samples_time = sample_time,
    kernel_name = "quartic",
    bw_net = 700, bw_time = 60, adaptive = TRUE,
    trim_bw_net = 900, trim_bw_time = 80,
    method = "discontinuous", div = "bw",
    max_depth = 10, digits = 2, tol = 0.01,
    agg = 15, grid_shape = c(1,1),
    verbose  = FALSE)

## make sure any open connections are closed afterward
if (!inherits(future::plan(), "sequential")) future::plan(future::sequential)
```

tnkde_get_loo_values *The exposed function to calculate TNKDE likelihood cv*

Description

The exposed function to calculate TNKDE likelihood cv (INTERNAL)

Usage

```
tnkde_get_loo_values(
  method,
  neighbour_list,
  sel_events,
  sel_events_wid,
  sel_events_time,
  events,
  events_wid,
  events_time,
  weights,
  bws_net,
  bws_time,
  kernel_name,
  line_list,
  max_depth,
  min_tol
)
```

Arguments

`method`	a string, one of "simple", "continuous", "discontinuous"
`neighbour_list`	a List, giving for each node an IntegerVector with its neighbours
`sel_events`	a Numeric vector indicating the selected events (id of nodes)
`sel_events_wid`	a Numeric Vector indicating the unique if of the selected events
`sel_events_time`	a Numeric Vector indicating the time of the selected events
`events`	a NumericVector indicating the nodes in the graph being events
`events_wid`	a NumericVector indicating the unique id of all the events
`events_time`	a NumericVector indicating the timestamp of each event

`weights`	a cube with the weights associated with each event for each bws_net and bws_time.
`bws_net`	an arma::vec with the network bandwidths to consider
`bws_time`	an arma::vec with the time bandwidths to consider
`kernel_name`	a string with the name of the kernel to use
`line_list`	a DataFrame describing the lines
`max_depth`	the maximum recursion depth
`min_tol`	a double indicating by how much 0 in density values must be replaced

Value

a matrix with the CV score for each pair of bandiwdths

Examples

```
# no example provided, this is an internal function
```

`tnkde_get_loo_values2` *The exposed function to calculate TNKDE likelihood cv*

Description

The exposed function to calculate TNKDE likelihood cv (INTERNAL) when an adaptive bandwidth is used

Usage

```
tnkde_get_loo_values2(
  method,
  neighbour_list,
  sel_events,
  sel_events_wid,
  sel_events_time,
  events,
  events_wid,
  events_time,
  weights,
  bws_net,
  bws_time,
  kernel_name,
  line_list,
  max_depth,
  min_tol
)
```

Arguments

`method`	a string, one of "simple", "continuous", "discontinuous"
`neighbour_list`	a List, giving for each node an IntegerVector with its neighbours
`sel_events`	a Numeric vector indicating the selected events (id of nodes)
`sel_events_wid`	a Numeric Vector indicating the unique if of the selected events
`sel_events_time`	a Numeric Vector indicating the time of the selected events
`events`	a NumericVector indicating the nodes in the graph being events
`events_wid`	a NumericVector indicating the unique id of all the events
`events_time`	a NumericVector indicating the timestamp of each event
`weights`	a cube with the weights associated with each event for each bws_net and bws_time.
`bws_net`	an arma::cube of three dimensions with the network bandwidths calculated for each observation for each global time and network bandwidths
`bws_time`	an arma::cube of three dimensions with the time bandwidths calculated for each observation for each global time and network bandwidths
`kernel_name`	a string with the name of the kernel to use
`line_list`	a DataFrame describing the lines
`max_depth`	the maximum recursion depth
`min_tol`	a double indicating by how much 0 in density values must be replaced

Value

a matrix with the CV score for each pair of global bandiwdths

Examples

```
# no example provided, this is an internal function
```

`tnkde_worker_bw_sel` *Worker function fo Bandwidth selection by likelihood cross validation for temporal NKDE*

Description

Calculate for multiple network and time bandwidths the cross validation likelihood to select an appropriate bandwidth in a data-driven approach (INTERNAL)

Usage

```
tnkde_worker_bw_sel(
  lines,
  quad_events,
  events_loc,
  events,
  w,
  kernel_name,
  bws_net,
  bws_time,
  method,
  div,
  digits,
  tol,
  sparse,
  max_depth,
  verbose = FALSE,
  cvl = FALSE
)
```

Arguments

`lines`	A feature collection of linestrings representing the underlying network
`quad_events`	a feature collection of points indicating for which events the densities must be calculated
`events_loc`	A feature collection of points representing the location of the events
`events`	A feature collection of points representing the events. Multiple events can share the same location. They are linked by the goid column
`w`	A numeric array with the weight of the events for each pair of bandwidth
`kernel_name`	The name of the kernel to use (string)
`bws_net`	A numeric vector with the network bandwidths. Could also be an array if an adaptive bandwidth is calculated.
`bws_time`	A numeric vector with the time bandwidths. Could also be an array if an adaptive bandwidth is calculated.
`method`	The type of NKDE to use (string)
`div`	The type of divisor (not used currently)
`digits`	The number of digits to retain from the spatial coordinates. It ensures that topology is good when building the network. Default is 3. Too high a precision (high number of digits) might break some connections
`tol`	A float indicating the minimum distance between the events and the lines' extremities when adding the point to the network. When points are closer, they are added at the extremity of the lines.
`sparse`	A Boolean indicating if sparse or regular matrices should be used by the Rcpp functions. These matrices are used to store edge indices between two nodes in a graph. Regular matrices are faster, but require more memory, in particular

	with multiprocessing. Sparse matrices are slower (a bit), but require much less memory.
max_depth	The maximum depth of recursion
verbose	A boolean
cvl	A boolean indicating if the cvl method (TRUE) or the loo (FALSE) method must be used

Value

An array with the CV score for each pair of bandiwdths (rows and lines) for each event (slices)

Examples

```
# no example provided, this is an internal function
```

triangle_kernel *triangle kernel*

Description

Function implementing the triangle kernel.

Usage

```
triangle_kernel(d, bw)
```

Arguments

d	The distance from the event
bw	The bandwidth used for the kernel

Value

The estimated density

Examples

```
#This is an internal function, no example provided
```

`tricube_kernel` *Tricube kernel*

Description

Function implementing the tricube kernel.

Usage

```
tricube_kernel(d, bw)
```

Arguments

`d`	The distance from the event
`bw`	The bandwidth used for the kernel

Value

The estimated density

Examples

```
#This is an internal function, no example provided
```

`triweight_kernel` *Triweight kernel*

Description

Function implementing the triweight kernel.

Usage

```
triweight_kernel(d, bw)
```

Arguments

`d`	The distance from the event
`bw`	The bandwidth used for the kernel

Value

The estimated density

Examples

```
#This is an internal function, no example provided
```

`uniform_kernel` *Uniform kernel*

Description

Function implementing the uniform kernel.

Usage

```
uniform_kernel(d, bw)
```

Arguments

`d`	The distance from the event
`bw`	The bandwidth used for the kernel

Value

The estimated density

Examples

```
#This is an internal function, no example provided
```

`worker_adaptive_bw_tnkde` *Worker function for adaptive bandwidth for TNDE*

Description

The worker function to calculate Adaptive bandwidths according to Abramson's smoothing regimen for TNKDE with a space-time interaction (INTERNAL).

Usage

```
worker_adaptive_bw_tnkde(
  lines,
  quad_events,
  events_loc,
  events,
  w,
  kernel_name,
  bw_net,
  bw_time,
  method,
  div,
  digits,
  tol,
  sparse,
  max_depth,
  verbose = FALSE
)
```

Arguments

lines	A feature collection of linestrings representing the underlying network
quad_events	a feature collection of points indicating for which events the densities must be calculated
events_loc	A feature collection of points representing the location of the events
events	A feature collection of points representing the events. Multiple events can share the same location. They are linked by the goid column
w	A numeric vector with the weight of the events
kernel_name	The name of the kernel to use (string)
bw_net	The fixed kernel bandwidth for the network dimension. Can also be a vector if several bandwidth must be used.
bw_time	The fixed kernel bandwidth for the time dimension. Can also be a vector if several bandwidth must be used.
method	The type of NKDE to use (string)
div	The divisor to use for the kernel. Must be "n" (the number of events within the radius around each sampling point), "bw" (the bandwidth) "none" (the simple sum).
digits	The number of digits to retain from the spatial coordinates. It ensures that topology is good when building the network. Default is 3. Too high a precision (high number of digits) might break some connections
tol	A float indicating the minimum distance between the events and the lines' extremities when adding the point to the network. When points are closer, they are added at the extremity of the lines.
sparse	A Boolean indicating if sparse or regular matrices should be used by the Rcpp functions. These matrices are used to store edge indices between two nodes in a graph. Regular matrices are faster, but require more memory, in particular with multiprocessing. Sparse matrices are slower (a bit), but require much less memory.
max_depth	An integer, the maximum depth to reach for continuous and discontinuous NKDE
verbose	A Boolean, indicating if the function should print messages about the process.

Value

A vector with the local bandwidths or an array if bw_net and bw_time are vectors

Examples

```
#This is an internal function, no example provided
```

Index