# Package 'spNetwork'

January 22, 2021

**Type** Package

**Title** Spatial Analysis on Network

**Version** 0.1.1

**Description** Perform spatial analysis on network.
Allow to calculate Network Kernel Density Estimate, and to build
spatial matrices ('listw' objects like in 'spdep' package) to conduct any kind of traditional
spatial analysis with spatial weights based on reticular distances. K functions on network
are also available but still experimental. References: Okabe et al (2019) <doi:10.1080/13658810802475491>;
Okabe et al (2012, ISBN:978-0470770818);Baddeley el al (2015, ISBN:9781482210200).

**License** GPL-2

**Encoding** UTF-8

**LazyData** true

**Imports** spdep (>= 1.1.2), maptools (>= 0.9-5), rgeos (>= 0.5-2), sp (>= 1.3-1), sf (>= 0.9-0), raster (>= 3.0-12), igraph (>= 1.2.4.2), 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), SearchTrees (>= 0.5.2), Rcpp (>= 1.0.4.6)

**Depends** R (>= 3.6)

**Suggests** future (>= 1.16.0), testthat (>= 3.0.0), knitr (>= 1.28), kableExtra (>= 1.1.0), rmarkdown (>= 2.1), RColorBrewer (>= 1.1-2), classInt (>= 0.4-3), reshape2 (>= 1.4.3), dplyr (>= 0.8.3), rlang (>= 0.4.6), rgdal (>= 1.5-18), plot3D (>= 1.3)

**RoxygenNote** 7.1.1

**VignetteBuilder** knitr

**URL** https://github.com/JeremyGelb/spNetwork

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

**LinkingTo** Rcpp, RcppProgress, RcppArmadillo

**SystemRequirements** C++11

**NeedsCompilation** yes

**Author** Jeremy Gelb [aut, cre],
Philippe Apparicio [ctb]

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

**Repository** CRAN

**Date/Publication** 2021-01-21 23:30:02 UTC

# R topics documented:

build_quadtree . . . 3
closest_points . . . 3
continuousfunction . . . 4
continuousWorker_int . . . 5
continuousWorker_sparse . . . 6
corrfactor_continuous . . . 7
corrfactor_continuous_sparse . . . 7
corrfactor_discontinuous . . . 8
corrfactor_discontinuous_sparse . . . 8
cosine_kernel . . . 9
cross_kfunctions . . . 10
cross_kfunctions.mc . . . 12
discontinuousfunction . . . 13
discontinuousWorker_int . . . 15
discontinuousWorker_sparse . . . 15
epanechnikov_kernel . . . 16
gaussian_kernel . . . 17
gaussian_kernel_scaled . . . 17
graph_checking . . . 18
kfunctions . . . 18
kfunctions.mc . . . 20
lines_center . . . 22
lines_points_along . . . 23
lixelize_lines . . . 24
lixelize_lines.mc . . . 24
network_listw . . . 25
network_listw.mc . . . 27
nkde . . . 29
nkde.mc . . . 33
quartic_kernel . . . 35
spatial_request . . . 36
triangle_kernel . . . 36
tricube_kernel . . . 37
triweight_kernel . . . 37
uniform_kernel . . . 38

**Index** **39**

## build_quadtree *Build a quadtree*

### Description

Generate quadtree object from package SearchTrees, useful to speed up spatial requesting.

### Usage

```
build_quadtree(data)
```

### Arguments

| | |
|---|---|
| data | a SpatialLinesDataFrame or a SpatialPointsDataFrame |

### Value

quadtree object from package SearchTrees

### Examples

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

## closest_points *Find closest points*

### Description

build a quadtree index and solve the nearest neighbour problem for two SpatialPointsDataFrame.

### Usage

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

### Arguments

| | |
|---|---|
| origins | a SpatialPointsDataFrame |
| targets | a SpatialPointsDataFrame |

### Value

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

### Examples

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

`continuousfunction` *The main function to calculate continuous NKDE (with ARMADILO and sparse matrix)*

## Description

The main function to calculate continuous NKDE (with ARMADILO and sparse matrix)

The main function to calculate continuous NKDE (with ARMADILO and Integer matrix)

## Usage

```
continuous_nkde_cpp_arma_sparse(
  neighbour_list,
  events,
  weights,
  samples,
  bws,
  kernel_name,
  nodes,
  line_list,
  max_depth,
  verbose
)

continuous_nkde_cpp_arma(
  neighbour_list,
  events,
  weights,
  samples,
  bws,
  kernel_name,
  nodes,
  line_list,
  max_depth,
  verbose
)
```

## Arguments

| | |
|---|---|
| `neighbour_list` | a list of the neighbours of each node |
| `events` | a numeric vector of the node id of each event |
| `weights` | a numeric vector of the weight of each event |
| `samples` | a DataFrame of the samples (with spatial coordinates and belonging edge) |
| `bws` | the kernel bandwidths for each event |
| `kernel_name` | the name of the kernel to use |

| | |
|---|---|
| nodes | a DataFrame representing the nodes of the graph (with spatial coordinates) |
| line_list | a DataFrame representing the lines of the graph |
| max_depth | the maximum recursion depth (after which recursion is stopped) |
| verbose | a boolean indicating if the function must print its progress |

## Value

a DataFrame with two columns : the kernel values (sum_k) and the number of events for each sample (n)

a DataFrame with two columns : the kernel values (sum_k) and the number of events for each sample (n)

---

`continuousWorker_int` *The worker function to calculate continuous NKDE (with ARMADILLO and Integer matrix)*

---

## Description

The worker function to calculate continuous NKDE (with ARMADILLO and Integer matrix)

## Arguments

| | |
|---|---|
| kernel_func | a cpp pointer function (selected with the kernel name) |
| samples_k | a numeric vector of the actual kernel values, updates at each recursion |
| neighbour_list | a List, providing an IntegerVector for each node with its neighbours |
| edge_mat | matrix, to find the id of each edge given two neighbours. |
| v | the actual node to consider for the recursion (int) |
| v1 | the connected node to consider for the recursion (int) |
| l1 | the edge connecting v and v1 (int) |
| d | the actual distance traveled before the recursion |
| alpha | the actual alpha value before the recursion |
| bw | the kernel bandwidth |
| line_weights | a vector with the length of the edges |
| samples_edgeid | a vector associating each sample to an edge |
| samples_x | a vector with x coordinates of each sample |
| samples_y | a vector with y coordinates of each sample |
| nodes_x | a vector with x coordinates of each node |
| nodes_y | a vector with y coordinates of each node |
| depth | the actual recursion depth |
| max_depth | the maximum recursion depth |

## Value

a vector with the kernel values calculated for each samples from the first node given

`continuousWorker_sparse` *The worker function to calculate continuous NKDE (with ARMADILLO and sparse matrix)*

## Description

The worker function to calculate continuous NKDE (with ARMADILLO and sparse matrix)

## Arguments

| | |
|---|---|
| `kernel_func` | a cpp pointer function (selected with the kernel name) |
| `samples_k` | a numeric vector of the actual kernel values, updates at each recursion |
| `neighbour_list` | a List, giving for each node an IntegerVector with its neighbours |
| `edge_mat` | matrix, to find the id of each edge given two neighbours. |
| `v` | the actual node to consider for the recursion (int) |
| `v1` | the connected node to consider for the recursion (int) |
| `l1` | the edge connecting v and v1 (int) |
| `d` | the actual distance traveled before the recursion |
| `alpha` | the actual alpha value before the recursion |
| `bw` | the kernel bandwidth |
| `line_weights` | a vector with the length of the edges |
| `samples_edgeid` | a vector associating each sample to an edge |
| `samples_x` | a vector with x coordinates of each sample |
| `samples_y` | a vector with y coordinates of each sample |
| `nodes_x` | a vector with x coordinates of each node |
| `nodes_y` | a vector with y coordinates of each node |
| `depth` | the actual recursion depth |
| `max_depth` | the maximum recursion depth |

## Value

a vector with the kernel values calculated for each samples from the first node given

## corrfactor_continuous *A function to calculate the necessary information to apply the Diggle correction factor with a continuous method*

### Description

A function to calculate the necessary information to apply the Diggle correction factor with a continuous method

### Usage

```
corrfactor_continuous(neighbour_list, events, line_list, bws, max_depth)
```

### Arguments

| | |
|---|---|
| neighbour_list | a list of the neighbours of each node |
| events | a numeric vector of the node id of each event |
| line_list | a DataFrame representing the lines of the graph |
| bws | the kernel bandwidth for each event |
| max_depth | the maximum recursion depth (after which recursion is stopped) |

### Value

a list of dataframes, used to calculate the Diggel correction factor

## corrfactor_continuous_sparse *A function to calculate the necessary information to apply the Diggle correction factor with a continuous method (sparse)*

### Description

A function to calculate the necessary information to apply the Diggle correction factor with a continuous method (sparse)

### Usage

```
corrfactor_continuous_sparse(neighbour_list, events, line_list, bws, max_depth)
```

### Arguments

| | |
|---|---|
| neighbour_list | a list of the neighbours of each node |
| events | a numeric vector of the node id of each event |
| line_list | a DataFrame representing the lines of the graph |
| bws | the kernel bandwidth for each event |
| max_depth | the maximum recursion depth (after which recursion is stopped) |

### Value

a list of dataframes, used to calculate the Diggel correction factor

---

`corrfactor_discontinuous`

*A function to calculate the necessary informations to apply the Diggle correction factor with a discontinuous method*

---

### Description

A function to calculate the necessary informations to apply the Diggle correction factor with a discontinuous method

### Arguments

| | |
|---|---|
| `neighbour_list` | a list of the neighbours of each node |
| `events` | a numeric vector of the node id of each event |
| `line_list` | a DataFrame representing the lines of the graph |
| `bws` | the kernel bandwidth for each event |
| `max_depth` | the maximum recursion depth (after which recursion is stopped) |

### Value

a list of dataframes, used to calculate the Diggel correction factor

---

`corrfactor_discontinuous_sparse`

*A function to calculate the necessary information to apply the Diggle correction factor with a discontinuous method (sparse)*

---

### Description

A function to calculate the necessary information to apply the Diggle correction factor with a discontinuous method (sparse)

### Usage

```
corrfactor_discontinuous_sparse(
  neighbour_list,
  events,
  line_list,
  bws,
  max_depth
)
```

### Arguments

| | |
|---|---|
| neighbour_list | a list of the neighbours of each node |
| events | a numeric vector of the node id of each event |
| line_list | a DataFrame representing the lines of the graph |
| bws | the kernel bandwidth for each event |
| max_depth | the maximum recursion depth (after which recursion is stopped) |

### Value

a list of dataframes, used to calculate the Diggel correction factor

---

## 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_kfunctions` *Network cross k and g functions (experimental)*

## Description

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

## 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
)
```

## Arguments

| | |
|---|---|
| `lines` | A SpatialLinesDataFrame with the sampling points. The geometries must be a SpatialLinesDataFrame (may crash if some geometries are invalid) |
| `pointsA` | A SpatialPointsDataFrame representing the points to which the distances are calculated. |
| `pointsB` | A SpatialPointsDataFrame representing the points from which the distances are calculated. |
| `start` | A double, the start value for evaluating the k and g functions |
| `end` | A double, the last value for evaluating the k and g functions |
| `step` | A double, the jump between two evaluations of the k and g function |
| `width` | The width of each donut for the g-function |
| `nsim` | An integer indicating the number of Monte Carlo simulations required |
| `conf_int` | A double indicating the width confidence interval (default = 0.05) |
| `digits` | An integer indicating the number of digits to retain for 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 are selected vertices on the new network. |
| agg | A double indicating if the events must be aggregated within a distance. if NULL, then the events are aggregated by rounding the coordinates. |
| verbose | A Boolean indicating if progress messages should be displayed |

## 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.

## Value

A list with the folowing values :

- plotkA ggplot2 object representing the values of the cross k-function
- plotgA ggplot2 object representing the values of the cross g-function
- valuesA DataFrame with the values used to build the plots

## Examples

```
## Not run:
networkgpkg <- system.file("extdata", "networks.gpkg", package = "spNetwork", mustWork = TRUE)
eventsgpkg <- system.file("extdata", "events.gpkg", package = "spNetwork", mustWork = TRUE)
main_network_mtl <- rgdal::readOGR(networkgpkg,layer="main_network_mtl", verbose=FALSE)
mtl_libraries <- rgdal::readOGR(eventsgpkg,layer="mtl_libraries", verbose=FALSE)
mtl_theatres <- rgdal::readOGR(eventsgpkg,layer="mtl_theatres", verbose=FALSE)
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)

## End(Not run)
```

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

## Description

Calculate the cross k and g functions for a set of points on a network with multicore support. (experimental)

## 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
)
```

## Arguments

| | |
|---|---|
| `lines` | A SpatialLinesDataFrame with the sampling points. The geometries must be a SpatialLinesDataFrame (may crash if some geometries are invalid) |
| `pointsA` | A SpatialPointsDataFrame representing the points to which the distances are calculated. |
| `pointsB` | A SpatialPointsDataFrame representing the points from which the distances are calculated. |
| `start` | A double, the start value for evaluating the k and g functions |
| `end` | A double, the last value for evaluating the k and g functions |
| `step` | A double, the jump between two evaluations of the k and g function |
| `width` | The width of each donut for the g-function |
| `nsim` | An integer indicating the number of Monte Carlo simulations required |
| `conf_int` | A double indicating the width confidence interval (default = 0.05) |
| `digits` | An integer indicating the number of digits to retain for 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 are selected vertices on the new network. |
| agg | A double indicating if the events must be aggregated within a distance. If NULL, the events are aggregated by rounding the coordinates. |
| verbose | A Boolean indicating if progress messages should be displayed |

## Value

A list with the following values :

- plotkA ggplot2 object representing the values of the cross k-function
- plotgA ggplot2 object representing the values of the cross g-function
- valuesA DataFrame with the values used to build the plots

## Examples

```
## Not run:
networkgpkg <- system.file("extdata", "networks.gpkg", package = "spNetwork", mustWork = TRUE)
eventsgpkg <- system.file("extdata", "events.gpkg", package = "spNetwork", mustWork = TRUE)
main_network_mtl <- rgdal::readOGR(networkgpkg,layer="main_network_mtl", verbose=FALSE)
mtl_libraries <- rgdal::readOGR(eventsgpkg,layer="mtl_libraries", verbose=FALSE)
mtl_theatres <- rgdal::readOGR(eventsgpkg,layer="mtl_theatres", verbose=FALSE)
future::plan(future::multisession(workers=2))
result <- cross_kfunctions.mc(main_network_mtl, mtl_libraries, mtl_theatres,
                           start = 0, end = 2500, step = 10, width = 250,
                           nsim = 50, conf_int = 0.05, digits = 2,
                           tol = 0.1, agg = NULL, verbose = TRUE)
## make sure any open connections are closed afterward
if (!inherits(future::plan(), "sequential")) future::plan(future::sequential)

## End(Not run)
```

---

# discontinuousfunction *The main function to calculate discontinuous NKDE (ARMA and sparse matrix)*

---

## Description

The main function to calculate discontinuous NKDE (ARMA and sparse matrix)

The main function to calculate discontinuous NKDE (ARMA and Integer matrix)

## Usage

```
discontinuous_nkde_cpp_arma_sparse(
  neighbour_list,
  events,
  weights,
  samples,
  bws,
  kernel_name,
  nodes,
  line_list,
  max_depth,
  verbose
)

discontinuous_nkde_cpp_arma(
  neighbour_list,
  events,
  weights,
  samples,
  bws,
  kernel_name,
  nodes,
  line_list,
  max_depth,
  verbose
)
```

## Arguments

| | |
|---|---|
| neighbour_list | a list of the neighbours of each node |
| events | a numeric vector of the node id of each event |
| weights | a numeric vector of the weight of each event |
| samples | a DataFrame of the samples (with spatial coordinates and belonging edge) |
| bws | the kernel bandwidth for each event |
| kernel_name | the name of the kernel function to use |
| nodes | a DataFrame representing the nodes of the graph (with spatial coordinates) |
| line_list | a DataFrame representing the lines of the graph |
| max_depth | the maximum recursion depth (after which recursion is stopped) |
| verbose | a boolean indicating if the function must print its progress |

## Value

a DataFrame with two columns : the kernel values (sum_k) and the number of events for each sample (n)

a DataFrame with two columns : the kernel values (sum_k) and the number of events for each sample (n)

## discontinuousWorker_int

*The worker function to calculate discontinuous NKDE (with ARMADILLO and Integer matrix)*

### Description

The worker function to calculate discontinuous NKDE (with ARMADILLO and Integer matrix)

### 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 |
| neighbour_list | a List, giving for each node an IntegerVector with its neighbours |
| v | the actual node to consider for the recursion (int) |
| bw | the kernel bandiwdth |
| line_weights | a vector with the length of the edges |
| samples_edgeid | a vector associating each sample to an edge |
| samples_x | a vector with x coordinates of each sample |
| samples_ya | vector with y coordinates of each sample |
| nodes_x | a vector with x coordinates of each node |
| nodes_y | a vector with y coordinates of each node |
| depth | the actual recursion depth |
| max_depth | the maximum recursion depth |

### Value

a vector with the kernel values calculated for each samples from the first node given

## discontinuousWorker_sparse

*The worker function to calculate discontinuous NKDE (with ARMADILLO and sparse matrix)*

### Description

The worker function to calculate discontinuous NKDE (with ARMADILLO and sparse matrix)

### 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 |
| neighbour_list | a List, giving for each node an IntegerVector with its neighbours |
| v | the actual node to consider for the recursion (int) |
| bw | the kernel bandiwdth |
| line_weights | a vector with the length of the edges |
| samples_edgeid | a vector associating each sample to an edge |
| samples_x | a vector with x coordinates of each sample |
| samples_ya | vector with y coordinates of each sample |
| nodes_x | a vector with x coordinates of each node |
| nodes_y | a vector with y coordinates of each node |
| depth | the actual recursion depth |
| max_depth | the maximum recursion depth |

### Value

a vector with the kernel values calculated for each samples from the first node given

---

## 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
```

## 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
```

## graph_checking *Topological error*

### Description

A utility function to find topological errors in a network.

### Usage

```
graph_checking(lines, digits, tol)
```

### Arguments

| | |
|---|---|
| lines | A SpatialLinesDataFrame representing the network |
| digits | An integer indicating the number of digits to retain for coordinates |
| tol | A float indicating the tolerance distance to identify a dangle node |

### Value

A list with two elements. The first is a SpatialPointsDataFrame indicating for each node of the network to which component it belongs. The second is a SpatialPointsDataFrame with the dangle nodes of the network.

### Examples

```
## Not run:
networkgpkg <- system.file("extdata", "networks.gpkg", package = "spNetwork", mustWork = TRUE)
mtl_network <- rgdal::readOGR(networkgpkg,layer="mtl_network", verbose=FALSE)
topo_errors <- graph_checking(mtl_network, 2, 2)

## End(Not run)
```

## kfunctions *Network k and g functions (experimental)*

### Description

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

## Usage

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

## Arguments

| | |
|---|---|
| lines | A SpatialLinesDataFrame with the sampling points. The geometries must be a SpatialLinesDataFrame (may crash if some geometries are invalid) |
| points | A SpatialPointsDataFrame representing the points on the network. These points will be snapped on the network. |
| start | A double, the start value for evaluating the k and g functions |
| end | A double, the last value for evaluating the k and g functions |
| step | A double, the jump between two evaluations of the k and g function |
| width | The width of each donut for the g-function |
| nsim | An integer indicating the number of Monte Carlo simulations required |
| conf_int | A double indicating the width confidence interval (default = 0.05) |
| digits | An integer indicating the number of digits to retain for 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 are selected vertices on the new network. |
| agg | A double indicating if the events must be aggregated within a distance. If NULL, the events are aggregated by rounding the coordinates. |
| verbose | A Boolean indicating if progress messages should be displayed. |

## 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.

## Value

A list with the following values :

- plotkA ggplot2 object representing the values of the k-function
- plotgA ggplot2 object representing the values of the g-function
- valuesA DataFrame with the values used to build the plots

## Examples

```
## Not run:
networkgpkg <- system.file("extdata", "networks.gpkg", package = "spNetwork", mustWork = TRUE)
eventsgpkg <- system.file("extdata", "events.gpkg", package = "spNetwork", mustWork = TRUE)
main_network_mtl <- rgdal::readOGR(networkgpkg,layer="main_network_mtl", verbose=FALSE)
mtl_libraries <- rgdal::readOGR(eventsgpkg,layer="mtl_libraries", verbose=FALSE)
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)

## End(Not run)
```

---

# kfunctions.mc *Network k and g functions (multicore, experimental)*

---

## 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. (experimental)

## Usage

```
kfunctions.mc(
  lines,
  points,
  start,
  end,
  step,
  width,
  nsim,
  conf_int = 0.05,
  digits = 2,
  tol = 0.1,
  resolution = 50,
  agg = NULL,
  verbose = TRUE
)
```

## Arguments

| | |
|---|---|
| lines | A SpatialLinesDataFrame with the sampling points. The geometries must be a SpatialLinesDataFrame (may crash if some geometries are invalid) |
| points | A SpatialPointsDataFrame representing the points on the network. These points will be snapped on the network. |
| start | A double, the start value for evaluating the k and g functions |
| end | A double, the last value for evaluating the k and g functions |
| step | A double, the jump between two evaluations of the k and g functions |
| width | The width of each donut for the g-function |
| nsim | An integer indicating the number of Monte Carlo simulations required |
| conf_int | A double indicating the width confidence interval (default = 0.05) |
| digits | An integer indicating the number of digits to retain for 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 are selected vertices on the new network. |
| agg | A double indicating if the events must be aggregated within a distance. If NULL, the events are aggregated by rounding the coordinates. |
| verbose | A Boolean indicating if progress messages should be displayed |

## Details

For details, please look at the function kfunctions.

## Value

A list with the following values :

- plotkA ggplot2 object representing the values of the k-function
- plotgA ggplot2 object representing the values of the g-function
- valuesA DataFrame with the values used to build the plots

## Examples

```
## Not run:
networkgpkg <- system.file("extdata", "networks.gpkg", package = "spNetwork", mustWork = TRUE)
eventsgpkg <- system.file("extdata", "events.gpkg", package = "spNetwork", mustWork = TRUE)
main_network_mtl <- rgdal::readOGR(networkgpkg,layer="main_network_mtl", verbose=FALSE)
mtl_libraries <- rgdal::readOGR(eventsgpkg,layer="mtl_libraries", verbose=FALSE)
future::plan(future::multisession(workers=2))
result <- kfunctions.mc(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)
## make sure any open connections are closed afterward
if (!inherits(future::plan(), "sequential")) future::plan(future::sequential)

## End(Not run)
```

---

# lines_center *Center points of lines*

---

## Description

Generate a SpatialPointsDataFrame with line center points. The points are located at center of the line based on the length of the line.

## Usage

```
lines_center(lines)
```

## Arguments

| | |
|---|---|
| lines | The SpatialLinesDataframe to use |

## Value

An object of class SpatialPointsDataFrame (package sp)

## Examples

```
## Not run:
networkgpkg <- system.file("extdata", "networks.gpkg", package = "spNetwork", mustWork = TRUE)
mtl_network <- rgdal::readOGR(networkgpkg,layer="mtl_network", verbose=FALSE)
centers <- lines_center(mtl_network)

## End(Not run)
```

---

# lines_points_along *Points along lines*

---

## Description

Generate points along the lines of a SpatialLinesDataFrame.

## Usage

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

## Arguments

| | |
|---|---|
| lines | The SpatialLinesDataframe to use |
| dist | The distance between the points along the lines |

## Value

An object of class SpatialLinesDataframe (package sp)

## Examples

```
## Not run:
networkgpkg <- system.file("extdata", "networks.gpkg", package = "spNetwork", mustWork = TRUE)
mtl_network <- rgdal::readOGR(networkgpkg,layer="mtl_network", verbose=FALSE)
new_pts <- lines_points_along(mtl_network,50)

## End(Not run)
```

## lixelize_lines *Cut lines into lixels*

### Description

Cut a SpatialLines object into lixels with a specified minimal distance may fail if the lines geometries are self intersecting.

### Usage

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

### Arguments

| | |
|---|---|
| lines | The SpatialLinesDataframe 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. |
| verbose | A Boolean indicating if a progress bar should be displayed |

### Value

An object of class SpatialLinesDataFrame (package sp)

### Examples

```
## Not run:
networkgpkg <- system.file("extdata", "networks.gpkg", package = "spNetwork", mustWork = TRUE)
mtl_network <- rgdal::readOGR(networkgpkg,layer="mtl_network", verbose=FALSE)
lixels <- lixelize_lines(mtl_network,150,50)

## End(Not run)
```

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

### Description

Cut a SpatialLines object into lixels with a specified minimal distance may fail if the lines geometries are self intersecting with multicore support.

### Usage

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

### Arguments

| | |
|---|---|
| lines | The SpatialLinesDataframe 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 |
| 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

An object of class SpatialLinesDataFrame (package sp)

### Examples

```
## Not run:
networkgpkg <- system.file("extdata", "networks.gpkg", package = "spNetwork", mustWork = TRUE)
mtl_network <- rgdal::readOGR(networkgpkg,layer="mtl_network", verbose=FALSE)
future::plan(future::multisession(workers=2))
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)
}

## End(Not run)
```

---

## 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 SpatialLinesDataFrame, SpatialPointsDataFrame or SpatialPolygonsDataFrame for which the spatial neighbouring list will be built |
| lines | A SpatialLinesDataFrame 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 center 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 traveling 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, 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)

## Examples

```
## Not run:
networkgpkg <- system.file("extdata", "networks.gpkg",package = "spNetwork", mustWork = TRUE)
mtl_network <- rgdal::readOGR(networkgpkg,layer="mtl_network", verbose=FALSE)
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))

## End(Not run)
```

---

# 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 SpatialLinesDataFrame, SpatialPointsDataFrame or SpatialPolygonsDataFrame for which the spatial neighbouring list will be built |
| lines | A SpatialLinesDataFrame 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 center 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 traveling 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, 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)

## Examples

```
## Not run:
networkgpkg <- system.file("extdata", "networks.gpkg", package = "spNetwork", mustWork = TRUE)
mtl_network <- rgdal::readOGR(networkgpkg,layer="mtl_network", verbose=FALSE)
future::plan(future::multisession(workers=2))
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)

## End(Not run)
```

---

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 SpatialLinesDataFrame with the sampling points. The geometries must be a SpatialLinesDataFrame (may crash if some geometries are invalid) |
| events | A SpatialPointsDataFrame representing the events on the network. The points will be snapped on the network. |
| w | A vector representing the weight of each event |
| samples | A SpatialPointsDataFrame 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, scaled gaussian, tricube, cosine ,triweight, quartic, epanechnikov or uniform. |
| bw | The kernel bandwidth (in meters) |
| 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 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 SpatialPolygonsDataFrame or a SpatialPolygon 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 problem for the discontinuous method. For the continuous method, a larger value will strongly impact calculation speed. |
| digits | The number of digits to retain in the spatial coordinates. It ensures that topology is good when building the network. Default is 3 |
| tol | When adding the events and the sampling points to the network, 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 by rounding the coordinates. |
| sparse | A Boolean indicating if sparse or regular matrix should be used by the Rcpp functions. Regular matrix are faster, but require more memory and could lead to error, in particular with multiprocessing. Sparse matrix are slower, but require much less memory. |
| grid_shape | A vector of two values indicating how the study area must be split when performing the calculus (see details). Default is c(1,1) |
| verbose | A Boolean, indicating if the function should print messages about process. |
| check | A Boolean indicating if the geometry checks must be run before calculating the densities |

## 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.

- method="simple"This first method was presented by Xie et al. (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.
- method="discontinuous"The previous method has been criticized by Okabe et al (2008), 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
- method="continuous"If the discontinuous method is unbiased, it leads to a discontinuous kernel function which is a bit counter-intuitive. Okabe et al (2008) proposed another version of the kernel, that divide the mass of the density at intersection 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. To do so, an original fixed bandwidth must be specified (bw parameter), and is used to estimate priory densities 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, each other event falling in that buffer are aggregated to the first event, forming a new event. The coordinates of this new event are the mean 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 samplings points

## Examples

```
## Not run:
networkgpkg <- system.file("extdata", "networks.gpkg", package = "spNetwork", mustWork = TRUE)
eventsgpkg <- system.file("extdata", "events.gpkg", package = "spNetwork", mustWork = TRUE)
mtl_network <- rgdal::readOGR(networkgpkg,layer="mtl_network", verbose=FALSE)
bike_accidents <- rgdal::readOGR(eventsgpkg,layer="bike_accidents", verbose=FALSE)
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)

## End(Not run)
```

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. For details, please see the function nkde

## 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 SpatialLinesDataFrame with the sampling points. The geometries must be a SpatialLinesDataFrame (may crash if some geometries are invalid) |
| events | A SpatialPointsDataFrame representing the events on the network. The points will be snapped on the network. |
| w | A vector representing the weight of each event |
| samples | A SpatialPointsDataFrame 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 (in meters) |

| | |
|---|---|
| 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 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 SpatialPolygonsDataFrame or a SpatialPolygon 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 problem for the discontinuous method. For the continuous method, a larger value will strongly impact calculation speed. |
| digits | The number of digits to retain in the spatial coordinates. It ensures that topology is good when building the network. Default is 3 |
| tol | When adding the events and the sampling points to the network, 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 by rounding the coordinates. |
| sparse | A Boolean indicating if sparse or regular matrix should be used by the Rcpp functions. Regular matrix are faster, but require more memory and could lead to error, in particular with multiprocessing. Sparse matrix are slower, but require much less memory. |
| grid_shape | A vector of two values indicating how the study area must be split when performing the calculus (see details). Default is c(1,1) |
| verbose | A Boolean, indicating if the function should print messages about process. |
| check | A Boolean indicating if the geometry checks must be run before calculating the density. |

## Value

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

## Examples

```
## Not run:
networkgpkg <- system.file("extdata", "networks.gpkg", package = "spNetwork", mustWork = TRUE)
eventsgpkg <- system.file("extdata", "events.gpkg", package = "spNetwork", mustWork = TRUE)
mtl_network <- rgdal::readOGR(networkgpkg,layer="mtl_network", verbose=FALSE)
```

```
bike_accidents <- rgdal::readOGR(eventsgpkg,layer="bike_accidents", verbose=FALSE)
future::plan(future::multisession(workers=2))
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=FALSE)
## make sure any open connections are closed afterward
if (!inherits(future::plan(), "sequential")) future::plan(future::sequential)

## End(Not run)
```

---

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
```

## spatial_request *Spatial request*

### Description

Use a quadtree index to perform spatial request.

### Usage

```
spatial_request(geometry, tree, data)
```

### Arguments

| | |
|---|---|
| geometry | objects such as SpatialLine, SpatialPolygon or SpatialPoint |
| tree | a tree object from package SearchTrees |
| data | the original data used to build the tree object |

### Value

a subset of data, intersecting geometry

### Examples

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

## 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
```

# Index


build_quadtree, 3

closest_points, 3
continuous_nkde_cpp_arma
        *(*continuousfunction*)*, 4
continuous_nkde_cpp_arma_sparse
        *(*continuousfunction*)*, 4
continuousfunction, 4
continuousWorker_int, 5
continuousWorker_sparse, 6
corrfactor_continuous, 7
corrfactor_continuous_sparse, 7
corrfactor_discontinuous, 8
corrfactor_discontinuous_sparse, 8
cosine_kernel, 9
cross_kfunctions, 10
cross_kfunctions.mc, 12

discontinuous_nkde_cpp_arma
        *(*discontinuousfunction*)*, 13
discontinuous_nkde_cpp_arma_sparse
        *(*discontinuousfunction*)*, 13
discontinuousfunction, 13
discontinuousWorker_int, 15
discontinuousWorker_sparse, 15

epanechnikov_kernel, 16

gaussian_kernel, 17
gaussian_kernel_scaled, 17
graph_checking, 18

kfunctions, 18
kfunctions.mc, 20

lines_center, 22
lines_points_along, 23
lixelize_lines, 24
lixelize_lines.mc, 24

network_listw, 25
network_listw.mc, 27
nkde, 29
nkde.mc, 33

quartic_kernel, 35

spatial_request, 36

triangle_kernel, 36
tricube_kernel, 37
triweight_kernel, 37

uniform_kernel, 38