Package ‘spatstat.linnet’

May 23, 2025

Version 3.2-6

Date 2025-05-23

Title Linear Networks Functionality of the 'spatstat' Family

Maintainer Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>

Depends R (>= 3.5.0), stats, graphics, grDevices, methods, utils, spatstat.data (>= 3.1-4), spatstat.univar (>= 3.1-3), spatstat.geom (>= 3.4-1), spatstat.random (>= 3.4-1), spatstat.explore (>= 3.4-3), spatstat.model (>= 3.3-5)

Imports Matrix, spatstat.utils (>= 3.1-4), spatstat.sparse (>= 3.1-0)

Suggests goftest, locfit, spatstat (>= 3.0)

Description Defines types of spatial data on a linear network and provides functionality for geometrical operations, data analysis and modelling of data on a linear network, in the 'spatstat' family of packages.

Contains definitions and support for linear networks, including creation of networks, geometrical measurements, topological connectivity, geometrical operations such as inserting and deleting vertices, intersecting a network with another object, and interactive editing of networks.

Data types defined on a network include point patterns, pixel images, functions, and tessellations.

Exploratory methods include kernel estimation of intensity on a network, K-functions and pair correlation functions on a network, simulation envelopes, nearest neighbour distance and empty space distance, relative risk estimation with cross-validated bandwidth selection. Formal hypothesis tests of random pattern (chi-squared, Kolmogorov-Smirnov, Monte Carlo, Diggle-Cressie-Loosmore-Ford, Dao-Genton, two-stage Monte Carlo) and tests for covariate effects (Cox-Berman-Waller-Lawson, Kolmogorov-Smirnov, ANOVA) are also supported.
Parametric models can be fitted to point pattern data using the function lppm() similar to glm(). Only Poisson models are implemented so far. Models may involve dependence on covariates and dependence on marks. Models are fitted by maximum likelihood. Fitted point process models can be simulated, automatically. Formal hypothesis tests of a fitted model are supported (likelihood ratio test, analysis of deviance, Monte Carlo tests) along with basic tools for model selection (stepwise(), AIC()) and variable selection (sdr). Tools for validating the fitted model include simulation envelopes, residuals, residual plots and Q-Q plots, leverage and influence diagnostics, partial residuals, and added variable plots.

Random point patterns on a network can be generated using a variety of models.

License GPL (>= 2)

URL http://spatstat.org/

NeedsCompilation yes

ByteCompile true

BugReports https://github.com/spatstat/spatstat.linnet/issues

Author Adrian Baddeley [aut, cre, cph] (ORCID:
<https://orcid.org/0000-0001-9499-8382>),
Rolf Turner [aut, cph] (ORCID: <https://orcid.org/0000-0001-5521-5218>),
Ege Rubak [aut, cph] (ORCID: <https://orcid.org/0000-0002-6675-533X>),
Greg McSwiggan [aut, cph],
Tilman Davies [ctb, cph],
Mehdi Moradi [ctb, cph],
Suman Rakshit [ctb, cph],
Ottmar Cronie [ctb]

Repository CRAN

Date/Publication 2025-05-23 02:50:01 UTC

Contents

`spatstat.linnet-package`

The spatstat.linnet Package

Description

The **spatstat.linnet** package belongs to the **spatstat** family of packages. It contains the functionality for analysing spatial data on a linear network.

Details

spatstat is a family of R packages for the statistical analysis of spatial data. Its main focus is the analysis of spatial patterns of points in two-dimensional space.

The original **spatstat** package has now been split into several sub-packages.

This sub-package **spatstat.linnet** contains the user-level functions from **spatstat** that are concerned with spatial data on a linear network.

Structure of the spatstat family

The orginal **spatstat** package grew to be very large. It has now been divided into several **sub-packages**:

- **spatstat.utils** containing basic utilities
- **spatstat.sparse** containing linear algebra utilities
- **spatstat.data** containing datasets
- **spatstat.univar** containing functions for estimating probability distributions of random variables
- **spatstat.geom** containing geometrical objects and geometrical operations
- **spatstat.explore** containing the main functionality for exploratory and non-parametric analysis of spatial data
- **spatstat.model** containing the main functionality for statistical modelling and inference for spatial data
- **spatstat.linnet** containing functions for spatial data on a linear network
- **spatstat**, which simply loads the other sub-packages listed above, and provides documentation.

When you install **spatstat**, these sub-packages are also installed. Then if you load the **spatstat** package by typing `library(spatstat)`, the other sub-packages listed above will automatically be loaded or imported. For an overview of all the functions available in these sub-packages, see the help file for **spatstat** in the **spatstat** package,

Additionally there are several **extension packages:**

- **spatstat.gui** for interactive graphics
- **spatstat.local** for local likelihood (including geographically weighted regression)
- **spatstat.Knet** for additional, computationally efficient code for linear networks
- **spatstat.sphere** (under development) for spatial data on a sphere, including spatial data on the earth's surface

The extension packages must be installed separately and loaded explicitly if needed. They also have separate documentation.

Overview of spatstat.linnet

A linear network is a subset of the two-dimensional plane composed of straight line segments. It could represent a road network, for example. Our code requires that, if two segments intersect each other, then the intersection is a single point, and the intersection point is treated as a vertex of the network.

The **spatstat.linnet** package supports spatial data analysis on a linear network. The primary aim is to analyse spatial patterns of points on a network. The points could represent road accidents on a road network, for example.

The **spatstat.linnet** package provides code for handling

- `linear networks`

- `point patterns` on a `linear network`
- `pixel images` on a `linear network` (where the network is divided into small segments and a numerical value is assigned to each segment)
- `functions` on a `linear network` (i.e. functions that are defined at every location along the network)
- `tessellations` of a `linear network` (where the network is subdivided into disjoint subsets with different labels)
- `point process models` on a `linear network`

Here is a list of the main functionality provided in **spatstat.linnet**.

Linear networks

An object of class `"linnet"` represents a linear network. Examples of such objects include the dataset `simplenet` provided in the package.

Linear network objects can be created by the following functions:

`linnet`	create a linear network
`as.linnet`	convert other data to a network
`delaunayNetwork`	network of Delaunay triangulation
`dirichletNetwork`	network of Dirichlet edges

Utilities for manipulating networks include:

`[.linnet`	extract subset of linear network
`clickjoin`	interactively join vertices in network
`joinVertices`	join existing vertices in a network
`insertVertices`	insert new vertices at positions along network
`addVertices`	add new vertices, extending a network
`thinNetwork`	remove vertices or lines from a network
`repairNetwork`	repair internal format
`vertices.linnet`	extract the vertices of network
`terminalvertices`	find terminal vertices of network
`affine.linnet`	apply affine transformation
`shift.linnet`	apply vector translation
`rotate.linnet`	apply rotation
`rescale.linnet`	rescale the unit of length
`scalardilate.linnet`	physically rescale the network
`diameter.linnet`	diameter of linear network
`is.connected.linnet`	determine whether network is connected
`lineardisc`	compute disc of given radius in network
`marks.linnet`	extract marks of a network
`marks<-.linnet`	assign marks to a network
`plot.linnet`	plot a network
`as.owin.linnet`	extract window containing network
`as.psp.linnet`	extract line segments comprising network
`nsegments.linnet`	number of segments in network

`nvertices.linnet`	number of vertices in network
`pixellate.linnet`	convert network to 2D pixel image
`print.linnet`	print basic information
`summary.linnet`	print summary information
`unitname.linnet`	extract name of unit of length
`unitname<-.linnet`	assign name of unit of length
`vertexdegree`	number of segments meeting each vertex
`volume.linnet`	total length of network
`Window.linnet`	extract window containing network
`density.linnet`	smoothed 2D spatial density of lines

A network is called a tree if it has no closed loops. The following functions support the creation and manipulation of trees:

`begins`	check start of character string
`branchlabelfun`	tree branch membership labelling function
`deletebranch`	delete a branch of a tree
`extractbranch`	extract a branch of a tree
`treebranchlabels`	label vertices of a tree by branch membership
`treeprune`	prune tree to given level

Point patterns on a linear network

An object of class `"lpp"` represents a point pattern on a linear network (for example, road accidents on a road network).

Examples of such objects include the following datasets provided in the **spatstat.data** package:

`chicago`	Chicago crime data
`dendrite`	Dendritic spines data
`spiders`	Spider webs on mortar lines of brick wall

There is also a dataset provided in the extension package **spatstat.Knet**:

`wacrashes`	Road accidents in Western Australia

Point patterns on a network can be created by the following functions:

`lpp`	create a point pattern on a linear network
`as.lpp`	convert other data to point pattern on network
`clicklpp`	interactively add points on a linear Network
`crossing.linnet`	crossing points between network and other lines

Point patterns on a network can be generated randomly using the following functions:

`rpoislpp`	Poisson points on linear network

`runiflpp`	uniform random points on a linear network
`rlpp`	random points on a linear network
`rSwitzerlpp`	simulate Switzer-type point process on linear network
`rThomaslpp`	simulate Thomas process on linear network
`rcelllpp`	simulate cell process on linear network
`rjitter.lpp`	randomly perturb a point pattern on a network

Functions for manipulating a point pattern on a network include the following. An object of class `"lpp"` also belongs to the class `"ppx"`, for which additional support is available.

`as.ppp.lpp`	convert to 2D point pattern
`as.psp.lpp`	extract line segments
`marks.ppx`	extract marks associated with points
`marks<-.ppx`	assign marks to points on network
`nsegments.lpp`	count number of segments
`print.lpp`	print basic information
`summary.lpp`	print summary information
`unitname.lpp`	extract name of unit of length
`unitname<-.lpp`	assign name of unit of length
`unmark.lpp`	remove marks
`subset.lpp`	subset of points satisfying a condition
`[.lpp`	extract subset of point pattern
`Window.lpp`	extract window containing network
`as.owin.lpp`	extract window containing network
`affine.lpp`	apply affine transformation
`shift.lpp`	apply vector translation
`rotate.lpp`	apply rotation
`rescale.lpp`	rescale the unit of length
`scalardilate.lpp`	physically rescale the network and points
`connected.lpp`	find connected components of point pattern on network
`cut.lpp`	classify points in a Point Pattern on a Network
`distfun.lpp`	distance map (function)
`distmap.lpp`	distance map (image)
`domain.lpp`	extract the linear network
`identify.lpp`	interactively identify points
`is.multitype.lpp`	recognize whether point pattern is multitype
`nncross.lpp`	nearest neighbours
`nndist.lpp`	nearest neighbour distances
`nnfromvertex`	nearest data point from each vertex
`nnfun.lpp`	nearest neighbour map
`nnwhich.lpp`	identify nearest neighbours
`pairdist.lpp`	pairwise shortest-path distances
`plot.lpp`	plot point pattern on linear Network
`points.lpp`	draw points on existing plot
`superimpose.lpp`	superimpose several point patterns
`text.lpp`	add text labels
`unstack.lpp`	separate multiple columns of marks

Pixel images on a network

An object of class `"linim"` represents a pixel image on a linear network. Effectively, the network is divided into small segments (lixels) and each small segment is assigned a value, which could be numeric, factor, logical or complex values.

Pixel images on a network can be created using the following functions:

`linim`	create pixel image on linear network
`as.linim`	convert other data to pixel image on network

Functions for manipulating a pixel image on a network include:

`[.linim`	extract subset of pixel image on linear network
`[<-.linim`	reset values in subset of image on linear network
`Math.linim`	S3 group generic methods for images on a linear network
`eval.linim`	evaluate expression involving pixel images on linear network
`as.linnet.linim`	extract linear network
`integral.linim`	integral of pixel image on a linear network
`mean.linim`	mean of pixel values
`median.linim`	median of pixel values
`quantile.linim`	quantiles of pixel values
`as.data.frame.linim`	convert to data frame
`print.linim`	print basic information
`summary.linim`	print summary information
`affine.linim`	apply affine transformation
`scalardilate.linim`	apply scalar dilation
`shift.linim`	apply vector translation
`pairs.linim`	scatterplot matrix for images
`persp.linim`	perspective view of pixel image on network
`plot.linim`	plot pixel image on linear network

Functions on a linear network

An object of class `"linfun"` represents a function defined at any location along the network. Objects of this class are created by the following functions:

`linfun`	create function on a linear network
`as.linfun`	convert other data to function on network

The following supporting code is available:

`print.linfun`	print basic information
`summary.linfun`	print summary information

`plot.linfun`	plot function on network
`persp.linfun`	perspective view of function on network
`as.data.frame.linfun`	convert to data frame
`as.owin.linfun`	extract window containing network
`as.function.linfun`	convert to ordinary R function

Tessellations of a linear network

An object of class `"lintess"` represents a tessellation of the network, that is, a subdivision of the network into disjoint subsets called 'tiles'. Objects of this class are created by the following functions:

`lintess`	create tessellation of network
`chop.linnet`	divide a linear network into tiles using infinite lines
`divide.linnet`	divide linear network at cut points
`lineardirichlet`	Dirichlet tessellation on a linear network

The following functions are provided for manipulating a tessellation on a network:

`as.data.frame.lintess`	convert to data frame
`intersect.lintess`	intersection of two tessellations on network
`lineartileindex`	determine which tile contains each given point on network
`marks.lintess`	extract marks of each tile
`marks<-.lintess`	assign marks to each tile
`plot.lintess`	plot tessellation on network
`tile.lengths`	compute lengths of tiles
`tilenames.lintess`	names of tiles
`as.linfun.lintess`	convert tessellation to a function

Smoothing a point pattern on a linear network:

Given a point pattern dataset on a linear network, it is often desired to estimate the spatially-varying density or intensity of points along the network. For example if the points represent road accidents, then we may wish to estimate the spatially-varying density of accidents per unit length (over a given period of time).

Related tasks include estimation of relative risk, and smoothing of of values observed at the data points.

`density.lpp`	kernel estimate of intensity
`densityEqualSplit`	kernel estimate of intensity using equal-split algorithm
`densityHeat.lpp`	kernel estimate of intensity using heat equation
`densityQuick.lpp`	kernel estimate of intensity using a 2D kernel
`densityVoronoi.lpp`	intensity estimate using Voronoi-Dirichlet Tessellation
`densityfun.lpp`	kernel estimate of intensity as a function
`bw.lppl`	Bandwidth selection for kernel estimate of intensity

`bw.voronoi`	bandwidth selection for Voronoi estimator
`relrisk.lpp`	kernel estimate of relative risk
`bw.relrisk.lpp`	Bandwidth selection for relative risk
`Smooth.lpp`	spatial smoothing of observations at points

Exploration of dependence on a covariate:

Another task is to investigate how the spatially-varying intensity of points depends on an explanatory variable (covariate). The covariate may be given as a pixel image on the network (class `"linim"`) or as a function on the network (class `"linfun"`).

`rhohat.lpp`	nonparametric estimate of intensity as function of a covariate
`roc.lpp`	Receiver Operating Characteristic for data on a network
`auc.lpp`	Area Under ROC Curve for data on a network
`cdf.test.lpp`	spatial distribution test for points on a linear network
`berman.test.lpp`	Berman's tests for point pattern on a network
`sdr.lpp`	Sufficient Dimension Reduction for a point pattern on a linear network

Summary statistics for a point pattern on a linear network:

These are for point patterns on a linear network (class `lpp`). For unmarked patterns:

`linearK`	K function on linear network
`linearKinhom`	inhomogeneous K function on linear network
`linearpcf`	pair correlation function on linear network
`linearpcfinhom`	inhomogeneous pair correlation on linear network
`linearJinhom`	inhomogeneous J function on linear network
`linearKEuclid`	K function on linear network using Euclidean distance
`linearKEuclidInhom`	inhomogeneous K function on linear network using Euclidean distance
`linearpcfEuclid`	pair correlation function on linear network using Euclidean distance
`linearpcfEuclidInhom`	inhomogeneous pair correlation on linear network using Euclidean distance

For multitype patterns:

`linearKcross`	K function between two types of points
`linearKdot`	K function from one type to any type
`linearKcross.inhom`	Inhomogeneous version of `linearKcross`
`linearKdot.inhom`	Inhomogeneous version of `linearKdot`
`linearmarkconnect`	Mark connection function on linear network
`linearmarkequal`	Mark equality function on linear network
`linearpcfcross`	Pair correlation between two types of points
`linearpcfdot`	Pair correlation from one type to any type
`linearpcfcross.inhom`	Inhomogeneous version of `linearpcfcross`
`linearpcfdot.inhom`	Inhomogeneous version of `linearpcfdot`

Related facilities:

`pairdist.lpp`	distances between pairs
`crossdist.lpp`	distances between pairs
`nndist.lpp`	nearest neighbour distances
`nncross.lpp`	nearest neighbour distances
`nnwhich.lpp`	find nearest neighbours
`nnfun.lpp`	find nearest data point
`density.lpp`	kernel smoothing estimator of intensity
`distfun.lpp`	distance transform
`envelope.lpp`	simulation envelopes
`rpoislpp`	simulate Poisson points on linear network
`runiflpp`	simulate random points on a linear network

It is also possible to fit point process models to `lpp` objects.

Point process models on a linear network:

An object of class `"lpp"` represents a pattern of points on a linear network. Point process models can also be fitted to these objects. Currently only Poisson models can be fitted.

`lppm`	point process model on linear network
`anova.lppm`	analysis of deviance for point process model on linear network
`envelope.lppm`	simulation envelopes for point process model on linear network
`fitted.lppm`	fitted intensity values
`predict.lppm`	model prediction on linear network
`data.lppm`	extract original data
`berman.test.lppm`	Berman's tests of goodness-of-fit
`is.marked.lppm`	Recognise whether model is marked
`is.multitype.lppm`	Recognise whether model is multitype
`is.stationary.lppm`	Recognise whether model is stationary
`model.frame.lppm`	Extract the variables in model
`model.images.lppm`	Compute images of constructed covariates
`model.matrix.lppm`	Extract design matrix
`plot.lppm`	Plot fitted point process model
`pseudoR2.lppm`	Calculate Pseudo-R-Squared for model
`simulate.lppm`	simulate fitted point process model

Licence

Acknowledgements

Ottmar Cronie, Tilman Davies, Greg McSwiggan and Suman Rakshit made substantial contributions of code.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>, Rolf Turner <`rolfturner@posteo.net`> and Ege Rubak <`rubak@math.aau.dk`>.

`addVertices` *Add New Vertices to a Linear Network*

Description

Adds new vertices to a linear network at specified locations outside the network.

Usage

```
addVertices(L, X, join=NULL, joinmarks=NULL)
```

Arguments

`L`	Existing linear network (object of class `"linnet"`) or point pattern on a linear network (object of class `"lpp"`).
`X`	Point pattern (object of class `"ppp"`) specifying the new vertices.
`join`	Optional information specifying how to join the new vertices `X` to the existing network. See Details. If `join=NULL` (the default), the new vertices are simply added to the list of network vertices without being joined to the rest of the network.
`joinmarks`	Optional vector or data frame of marks associated with the new edges specified by `join`.

Details

This function adds new vertices to an existing linear network `L`, at specified locations `X` outside the network.

The argument `L` can be either a linear network (class `"linnet"`) or some other object that includes a linear network.

The new vertex locations are points outside the network, specified as a point pattern `X` (object of class `"ppp"`).

The argument `join` specifies how to join the new vertices to the existing network.

- If `join=NULL` (the default), the new vertices are simply added to the list of network vertices without being joined to the rest of the network.
- If `join` is a vector of integers, then these are taken to be indices of existing vertices of `L` in the order given in `V = vertices(L)`. Then each new vertex `X[i]` will be joined to an existing vertex `V[j]` where `j = join[i]`. Each new vertex is joined to exactly one existing vertex.
- If `join="vertices"` then each new vertex `X[i]` is joined to the nearest existing vertex `V[j]`. Each new vertex is joined to exactly one existing vertex.

- If `join="nearest"` then each new vertex is projected to the nearest location along on the network; these locations are inserted as new vertices of `L`; and then each vertex `X[i]` is joined to the corresponding projected point. Each new vertex is joined to exactly one newly-inserted vertex.
- If `join` is a point pattern on a network (class `"lpp"`), it must be defined on the same network as `L` and it must consist of the same number of points as `X`. The points of `join` will be inserted as new vertices of `L`, and then each vertex `X[i]` is joined to the corresponding point `join[i]`. Each new vertex is joined to exactly one newly-inserted vertex.

The result is the modified object, with an attribute `"id"` such that the `i`th added vertex has become the `id[i]`th vertex of the new network.

Value

An object of the same class as `L` representing the result of adding the new vertices. The result also has an attribute `"id"` as described in Details.

Author(s)

Adrian Baddeley

See Also

`insertVertices` to insert vertices along an existing network.

`as.lpp`, `linnet`, `methods.linnet`, `joinVertices`, `thinNetwork`.

Examples

```
opa <- par(mfrow=c(1,3))
L <- simplenet
X <- runifpoint(20, Window(simplenet))
plot(L)
plot(X, add=TRUE, cols="green", pch=16, cex=2)
plot(addVertices(L, X, "nearest"), col="red")
plot(L, add=TRUE, col="grey", lwd=3)
plot(X, add=TRUE, cols="green", pch=16, cex=2)
plot(addVertices(L, X, "vertices"), col="red")
plot(L, add=TRUE, col="grey", lwd=3)
plot(X, add=TRUE, cols="green", pch=16, cex=2)
par(opa)
```

`affine.linnet` *Apply Geometrical Transformations to a Linear Network*

Description

Apply geometrical transformations to a linear network.

Usage

```
  ## S3 method for class 'linnet'
affine(X, mat=diag(c(1,1)), vec=c(0,0), ...)

  ## S3 method for class 'linnet'
shift(X, vec=c(0,0), ..., origin=NULL)

  ## S3 method for class 'linnet'
rotate(X, angle=pi/2, ..., centre=NULL)

  ## S3 method for class 'linnet'
scalardilate(X, f, ...)

  ## S3 method for class 'linnet'
rescale(X, s, unitname)
```

Arguments

`X`	Linear network (object of class `"linnet"`).
`mat`	Matrix representing a linear transformation.
`vec`	Vector of length 2 representing a translation.
`angle`	Rotation angle in radians.
`f`	Scalar dilation factor.
`s`	Unit conversion factor: the new units are `s` times the old units.
`...`	Arguments passed to other methods.
`origin`	Character string determining a location that will be shifted to the origin. Options are `"centroid"`, `"midpoint"` and `"bottomleft"`. Partially matched.
`centre`	Centre of rotation. Either a vector of length 2, or a character string (partially matched to `"centroid"`, `"midpoint"` or `"bottomleft"`). The default is the coordinate origin `c(0,0)`.
`unitname`	Optional. New name for the unit of length. A value acceptable to the function `unitname<-`

Details

These functions are methods for the generic functions `affine`, `shift`, `rotate`, `rescale` and `scalardilate` applicable to objects of class `"linnet"`.

All of these functions perform geometrical transformations on the object `X`, except for `rescale`, which simply rescales the units of length.

Value

Another linear network (of class `"linnet"`) representing the result of applying the geometrical transformation.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>

and Rolf Turner <`rolfturner@posteo.net`>

See Also

`linnet` and `as.linnet`.

Generic functions `affine`, `shift`, `rotate`, `scalardilate`, `rescale`.

Examples

```
U <- rotate(simplenet, pi)
stretch <- diag(c(2,3))
Y <- affine(simplenet, mat=stretch)
shear <- matrix(c(1,0,0.6,1),ncol=2, nrow=2)
Z <- affine(simplenet, mat=shear, vec=c(0, 1))
```

`affine.lpp` *Apply Geometrical Transformations to Point Pattern on a Linear Network*

Description

Apply geometrical transformations to a point pattern on a linear network.

Usage

```
  ## S3 method for class 'lpp'
affine(X, mat=diag(c(1,1)), vec=c(0,0), ...)

  ## S3 method for class 'lpp'
shift(X, vec=c(0,0), ..., origin=NULL)

  ## S3 method for class 'lpp'
rotate(X, angle=pi/2, ..., centre=NULL)

  ## S3 method for class 'lpp'
scalardilate(X, f, ...)

  ## S3 method for class 'lpp'
rescale(X, s, unitname)
```

Arguments

`X`	Point pattern on a linear network (object of class `"lpp"`).
`mat`	Matrix representing a linear transformation.
`vec`	Vector of length 2 representing a translation.
`angle`	Rotation angle in radians.
`f`	Scalar dilation factor.
`s`	Unit conversion factor: the new units are `s` times the old units.
`...`	Arguments passed to other methods.
`origin`	Character string determining a location that will be shifted to the origin. Options are `"centroid"`, `"midpoint"` and `"bottomleft"`. Partially matched.
`centre`	Centre of rotation. Either a vector of length 2, or a character string (partially matched to `"centroid"`, `"midpoint"` or `"bottomleft"`). The default is the coordinate origin `c(0,0)`.
`unitname`	Optional. New name for the unit of length. A value acceptable to the function `unitname<-`

Details

These functions are methods for the generic functions `affine`, `shift`, `rotate`, `rescale` and `scalardilate` applicable to objects of class `"lpp"`.

All of these functions perform geometrical transformations on the object `X`, except for `rescale`, which simply rescales the units of length.

Value

Another point pattern on a linear network (object of class `"lpp"`) representing the result of applying the geometrical transformation.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`> and Rolf Turner <`rolfturner@posteo.net`>

See Also

`lpp`.

Generic functions `affine`, `shift`, `rotate`, `scalardilate`, `rescale`.

Examples

```
X <- rpoislpp(2, simplenet)
U <- rotate(X, pi)
V <- shift(X, c(0.1, 0.2))
stretch <- diag(c(2,3))
Y <- affine(X, mat=stretch)
shear <- matrix(c(1,0,0.6,1),ncol=2, nrow=2)
Z <- affine(X, mat=shear, vec=c(0, 1))
```

`anova.lppm` *ANOVA for Fitted Point Process Models on Linear Network*

Description

Performs analysis of deviance for two or more fitted point process models on a linear network.

Usage

```
  ## S3 method for class 'lppm'
anova(object, ..., test=NULL)
```

Arguments

`object`	A fitted point process model on a linear network (object of class `"lppm"`).
`...`	One or more fitted point process models on the same linear network.
`test`	Character string, partially matching one of `"Chisq"`, `"F"` or `"Cp"`.

Details

This is a method for `anova` for fitted point process models on a linear network (objects of class `"lppm"`, usually generated by the model-fitting function `lppm`).

If the fitted models are all Poisson point processes, then this function performs an Analysis of Deviance of the fitted models. The output shows the deviance differences (i.e. 2 times log likelihood ratio), the difference in degrees of freedom, and (if `test="Chi"`) the two-sided p-values for the chi-squared tests. Their interpretation is very similar to that in `anova.glm`.

If some of the fitted models are *not* Poisson point processes, then the deviance difference is replaced by the adjusted composite likelihood ratio (Pace et al, 2011; Baddeley et al, 2014).

Value

An object of class `"anova"`, or `NULL`.

Errors and warnings

models not nested: There may be an error message that the models are not “nested”. For an Analysis of Deviance the models must be nested, i.e. one model must be a special case of the other. For example the point process model with formula `~x` is a special case of the model with formula `~x+y`, so these models are nested. However the two point process models with formulae `~x` and `~y` are not nested.

If you get this error message and you believe that the models should be nested, the problem may be the inability of R to recognise that the two formulae are nested. Try modifying the formulae to make their relationship more obvious.

different sizes of dataset: There may be an error message from `anova.glmlist` that “models were not all fitted to the same size of dataset”. This generally occurs when the point process models are fitted on different linear networks.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>

References

Ang, Q.W. (2010) *Statistical methodology for events on a network.* Master's thesis, School of Mathematics and Statistics, University of Western Australia.

Ang, Q.W., Baddeley, A. and Nair, G. (2012) Geometrically corrected second-order analysis of events on a linear network, with applications to ecology and criminology. *Scandinavian Journal of Statistics* **39**, 591–617.

Baddeley, A., Turner, R. and Rubak, E. (2015) Adjusted composite likelihood ratio test for Gibbs point processes. *Journal of Statistical Computation and Simulation* **86** (5) 922–941. DOI: 10.1080/00949655.2015.1044530.

McSwiggan, G., Nair, M.G. and Baddeley, A. (2012) Fitting Poisson point process models to events on a linear network. Manuscript in preparation.

Pace, L., Salvan, A. and Sartori, N. (2011) Adjusting composite likelihood ratio statistics. *Statistica Sinica* **21**, 129–148.

See Also

`lppm`

Examples

```
X <- runiflpp(10, simplenet)
mod0 <- lppm(X ~1)
modx <- lppm(X ~x)
anova(mod0, modx, test="Chi")
```

`as.data.frame.lintess` *Convert Network Tessellation to Data Frame*

Description

Converts a tessellation on a linear network into a data frame.

Usage

```
## S3 method for class 'lintess'
as.data.frame(x, ...)
```

Arguments

`x`	Tessellation on a linear network (object of class `"lintess"`).
`...`	Further arguments passed to `as.data.frame.default` to determine the row names and other features.

Details

A tessellation on a linear network is a partition of the network into non-overlapping pieces (tiles). Each tile consists of one or more line segments which are subsets of the line segments making up the network. A tile can consist of several disjoint pieces.

This function converts the tessellation `x` to a data frame. Each row of the data frame specifies one sub-segment of the network, and allocates it to a particular tile. The data frame has the following columns:

- The `seg` column specifies which line segment of the network contains the sub-segment. Values of `seg` are integer indices for the network segments in `as.psp(as.linnet(x))`.
- The `t0` and `t1` columns specify the start and end points of the sub-segment. They are numeric values between 0 and 1 inclusive, where the values 0 and 1 representing the network vertices that are joined by this network segment.
- The `tile` column specifies which tile of the tessellation includes this sub-segment. It is a factor whose levels are the names of the tiles.

The tessellation may have marks, which are attached to the *tiles* of the tessellation. If marks are present, the resulting data frame includes columns containing, for each sub-segment, the mark value of the corresponding tile.

Value

A data frame with columns named `seg`, `t0`, `t1`, `tile`, and possibly other columns.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`, Rolf Turner `<rolfturner@posteo.net>` and Ege Rubak `<rubak@math.aau.dk>`.

See Also

`lintess`

Examples

```
X <- lineardirichlet(runiflpp(3, simplenet))
marks(X) <- letters[1:3]
as.data.frame(X)
```

`as.linfun` *Convert Data to a Function on a Linear Network*

Description

Convert some kind of data to an object of class `"linfun"` representing a function on a linear network.

Usage

```
  as.linfun(X, ...)

  ## S3 method for class 'linim'
as.linfun(X, ...)

  ## S3 method for class 'lintess'
as.linfun(X, ..., values=marks(X), navalue=NA)
```

Arguments

`X`	Some kind of data to be converted.
`...`	Other arguments passed to methods.
`values`	Optional. Vector of function values, one entry associated with each tile of the tessellation.
`navalue`	Optional. Function value associated with locations that do not belong to a tile of the tessellation.

Details

An object of class `"linfun"` represents a function defined on a linear network.

The function `as.linfun` is generic. The method `as.linfun.linim` converts objects of class `"linim"` (pixel images on a linear network) to functions on the network.

The method `as.linfun.lintess` converts a tessellation on a linear network into a function with a different value on each tile of the tessellation. The function values are specified by the argument `values`. It should be a vector with one entry for each tile of the tessellation; any point lying in tile number `i` will return the value `v[i]`. If `values` is missing, the marks of the tessellation are taken as the function values. If `values` is missing and the tessellation has no marks, or if `values` is given as `NULL`, then the function returns factor values identifying which tile contains each given point.

Value

Object of class `"linfun"`.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>, Rolf Turner <`rolfturner@posteo.net`> and Ege Rubak <`rubak@math.aau.dk`>.

See Also

`linfun`

Examples

```
X <- runiflpp(2, simplenet)
Y <- runiflpp(5, simplenet)

# image on network
D <- density(Y, 0.1)

f <- as.linfun(D)
f
f(X)

# tessellation on network
Z <- lineardirichlet(Y)
g <- as.linfun(Z)
g(X)
h <- as.linfun(Z, values = runif(5))
h(X)
```

`as.linim` *Convert to Pixel Image on Linear Network*

Description

Converts various kinds of data to a pixel image on a linear network.

Usage

```
  as.linim(X, ...)

  ## S3 method for class 'linim'
as.linim(X, ...)

  ## Default S3 method:
as.linim(X, L, ...,
                              eps = NULL, dimyx = NULL, xy = NULL,
                              rule.eps=c("adjust.eps",
                                         "grow.frame", "shrink.frame"),
                              delta=NULL, nd=NULL)

  ## S3 method for class 'linfun'
as.linim(X, L=domain(X), ...,
                             eps = NULL, dimyx = NULL, xy = NULL,
                             rule.eps=c("adjust.eps",
                                        "grow.frame", "shrink.frame"),
                             delta=NULL, nd=NULL)
```

Arguments

`X`	Data to be converted to a pixel image on a linear network.
`L`	Linear network (object of class `"linnet"`).
`...`	Additional arguments passed to `X` when `X` is a function.
`eps`, `dimyx`, `xy`, `rule.eps`	Optional arguments passed to `as.mask` to control the pixel resolution.
`delta`	Optional. Numeric value giving the approximate distance (in coordinate units) between successive sample points along each segment of the network.
`nd`	Optional. Integer giving the (approximate) number of sample points on the network. Ignored if `delta` is given.

Details

This function converts the data `X` into a pixel image on a linear network, an object of class `"linim"` (see `linim`).

The argument `X` may be any of the following:

- a function on a linear network, an object of class `"linfun"`.
- a pixel image on a linear network, an object of class `"linim"`.
- a pixel image, an object of class `"im"`.
- any type of data acceptable to `as.im`, such as a function, numeric value, or window.

First `X` is converted to a pixel image object `Y` (object of class `"im"`). The conversion is performed by `as.im`. The arguments `eps`, `dimyx`, `xy` and `rule.eps` determine the pixel resolution.

Next `Y` is converted to a pixel image on a linear network using `linim`. The argument `L` determines the linear network. If `L` is missing or `NULL`, then `X` should be an object of class `"linim"`, and `L` defaults to the linear network on which `X` is defined.

In addition to converting the function to a pixel image, the algorithm also generates a fine grid of sample points evenly spaced along each segment of the network (with spacing at most `delta` coordinate units). The function values at these sample points are stored in the resulting object as a data frame (the argument `df` of `linim`). This mechanism allows greater accuracy for some calculations (such as `integral.linim`).

Value

An image object on a linear network; an object of class `"linim"`.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>, Rolf Turner <`rolfturner@posteo.net`> and Ege Rubak <`rubak@math.aau.dk`>

See Also

`as.im`

Examples

```
f <- function(x,y){ x + y }
plot(as.linim(f, simplenet))
```

`as.linnet.linim` *Extract Linear Network from Data on a Linear Network*

Description

Given some kind of data on a linear network, the command `as.linnet` extracts the linear network itself.

Usage

```
 ## S3 method for class 'linim'
as.linnet(X, ...)

 ## S3 method for class 'linfun'
as.linnet(X, ...)

 ## S3 method for class 'lintess'
as.linnet(X, ...)

 ## S3 method for class 'lpp'
as.linnet(X, ..., fatal=TRUE, sparse)
```

Arguments

`X`	Data on a linear network. A point pattern (class `"lpp"`), pixel image (class `"linim"`), function (class `"linfun"`) or tessellation (class `"lintess"`) on a linear network.
`...`	Ignored.
`fatal`	Logical value indicating whether data in the wrong format should lead to an error (`fatal=TRUE`) or a warning (`fatal=FALSE`).
`sparse`	Logical value indicating whether to use a sparse matrix representation, as explained in `linnet`. Default is to keep the same representation as in `X`.

Details

These are methods for the generic `as.linnet` for various classes.

The network on which the data are defined is extracted.

Value

A linear network (object of class `"linnet"`).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

See Also

`linnet`, `methods.linnet`.

Examples

```
# make some data
xcoord <- linfun(function(x,y,seg,tp) { x }, simplenet)
as.linnet(xcoord)
X <- as.linim(xcoord)
as.linnet(X)
```

---

as.linnet.psp *Convert Line Segment Pattern to Linear Network*

Description

Converts a line segment pattern to a linear network.

Usage

```
 ## S3 method for class 'psp'
as.linnet(X, ..., eps, sparse=FALSE)
```

Arguments

`X`	Line segment pattern (object of class "psp").
`...`	Ignored.
`eps`	Optional. Distance threshold. If two segment endpoints are closer than eps units apart, they will be treated as the same point, and will become a single vertex in the linear network.
`sparse`	Logical value indicating whether to use a sparse matrix representation, as explained in `linnet`.

Details

This command converts any collection of line segments into a linear network by guessing the connectivity of the network, using the distance threshold eps.

If any segments in `X` cross over each other, they are first cut into pieces using `selfcut.psp`.

Then any pair of segment endpoints lying closer than eps units apart, is treated as a single vertex. The linear network is then constructed using `linnet`.

It would be wise to check the result by plotting the degree of each vertex, as shown in the Examples.

If `X` has marks, then these are stored in the resulting linear network `Y <- as.linnet(X)`, and can be extracted as `marks(as.psp(Y))` or `marks(Y$lines)`.

Value

A linear network (object of class `"linnet"`).

The result also has an attribute `"camefrom"` indicating the provenance of each line in the resulting network. For example `camefrom[3]=2` means that the third line segment in the result is a piece of the second segment of `X`.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>, Rolf Turner <`rolfturner@posteo.net`> and Ege Rubak <`rubak@math.aau.dk`>.

See Also

`linnet`, `selfcut.psp`, `methods.linnet`.

Examples

```
# make some data
A <- psp(0.09, 0.55, 0.79, 0.80, window=owin())
B <- superimpose(A, as.psp(simplenet))

# convert to a linear network
L <- as.linnet(B)

# check validity
L
plot(L)
text(vertices(L), labels=vertexdegree(L))

# show the pieces that came from original segment number 1
S <- as.psp(L)
(camefrom <- attr(L, "camefrom"))
parts <- which(camefrom == 1)
plot(S[parts], add=TRUE, col="green", lwd=2)
```

`as.lpp` *Convert Data to a Point Pattern on a Linear Network*

Description

Convert various kinds of data to a point pattern on a linear network.

Usage

```
as.lpp(x=NULL, y=NULL, seg=NULL, tp=NULL, ...,
       marks=NULL, L=NULL, check=FALSE, sparse)
```

Arguments

`x, y`	Vectors of cartesian coordinates, or any data acceptable to `xy.coords`. Alternatively `x` can be a point pattern on a linear network (object of class `"lpp"`) or a planar point pattern (object of class `"ppp"`).
`seg, tp`	Optional local coordinates. Vectors of the same length as `x,y`. See Details.
`...`	Ignored.
`marks`	Optional marks for the point pattern. A vector or factor with one entry for each point, or a data frame or hyperframe with one row for each point.
`L`	Linear network (object of class `"linnet"`) on which the points lie.
`check`	Logical. Whether to check the validity of the spatial coordinates.
`sparse`	Optional logical value indicating whether to store the linear network data in a sparse matrix representation or not. See `linnet`.

Details

This function converts data in various formats into a point pattern on a linear network (object of class `"lpp"`).

The possible formats are:

- `x` is already a point pattern on a linear network (object of class `"lpp"`). Then `x` is returned unchanged.
- `x` is a planar point pattern (object of class `"ppp"`). Then `x` is converted to a point pattern on the linear network `L` using `lpp`.
- `x,y,seg,tp` are vectors of equal length. These specify that the `i`th point has Cartesian coordinates `(x[i],y[i])`, and lies on segment number `seg[i]` of the network `L`, at a fractional position `tp[i]` along that segment (with `tp=0` representing one endpoint and `tp=1` the other endpoint of the segment).
- `x,y` are missing and `seg,tp` are vectors of equal length as described above.
- `seg,tp` are `NULL`, and `x,y` are data in a format acceptable to `xy.coords` specifying the Cartesian coordinates.
- Only the arguments `x` and `L` are given, and `x` is a data frame with one of the following types:
  - two columns labelled `seg,tp` interpreted as local coordinates on the network.
  - two columns labelled `x,y` interpreted as Cartesian coordinates.
  - four columns labelled `x,y,seg,tp` interpreted as Cartesian coordinates and local coordinates.

Value

A point pattern on a linear network (object of class `"lpp"`).

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>

and Rolf Turner <`rolfturner@posteo.net`>

See Also

`lpp`.

Examples

```
A <- as.psp(simplenet)
X <- runifpointOnLines(10, A)
is.ppp(X)
Y <- as.lpp(X, L=simplenet)
```

`as.owin.lpp` *Convert Data on a Network to class owin*

Description

Converts data on a linear network into an object of class `"owin"`.

Usage

```
 ## S3 method for class 'lpp'
as.owin(W, ..., fatal=TRUE)

 ## S3 method for class 'lppm'
as.owin(W, ..., fatal=TRUE)
```

Arguments

`W`	Data specifying an observation window, in any of several formats described under *Details* below.
`fatal`	Logical value determining what to do if the data cannot be converted to an observation window. See Details.
`...`	Ignored.

Details

The class `"owin"` is a way of specifying the observation window for a point pattern. See `owin.object` for an overview.

The function `as.owin` converts data in any of several formats into an object of class `"owin"` for use by the **spatstat** package. The function `as.owin` is generic, with methods for different classes of objects, and a default method.

A long list of methods for `as.owin` is documented in the help file for `as.owin` in the **spatstat.geom** package.

This help file documents additional methods applicable when `W` is

- an object of class `"lpp"` representing a point pattern on a linear network. In this case, `as.owin` extracts the linear network and returns a window containing this network.

- an object of class `"lppm"` representing a fitted point process model on a linear network. In this case, `as.owin` extracts the linear network and returns a window containing this network.

If the argument `W` cannot be converted to a window, then an error will be generated (if `fatal=TRUE`) or a value of `NULL` will be returned (if `fatal=FALSE`).

Value

An object of class `"owin"` (see `owin.object`) specifying an observation window.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`, Rolf Turner `<rolfturner@posteo.net>` and Ege Rubak `<rubak@math.aau.dk>`.

See Also

`as.owin`, `owin.object`, `owin`.

Additional methods for `as.owin` may be provided by other packages outside the **spatstat** family.

Examples

```
as.owin(simplenet)
```

`auc.lpp` *Area Under ROC Curve for Data on a Network*

Description

Compute the AUC (area under the Receiver Operating Characteristic curve) for a fitted point process model on a linear network.

Usage

```
## S3 method for class 'lpp'
auc(X, covariate, ..., high = TRUE)

## S3 method for class 'lppm'
auc(X, ...)
```

Arguments

`X`	Point pattern (object of class `"ppp"` or `"lpp"`) or fitted point process model (object of class `"ppm"` or `"kppm"` or `"lppm"`).
`covariate`	Spatial covariate. Either a `function(x,y)`, a pixel image (object of class `"im"` or `"linim"`), or one of the strings `"x"` or `"y"` indicating the Cartesian coordinates.
`...`	Arguments passed to `as.mask` controlling the pixel resolution for calculations.

`high`	Logical value indicating whether the threshold operation should favour high or low values of the covariate.

Details

This command computes the AUC, the area under the Receiver Operating Characteristic curve. The ROC itself is computed by `roc`.

The function `auc` is generic, with methods for `"ppp"` and `"ppm"` described in the help file for `auc`.

This help file describes the methods for classes `"lpp"` and `"lppm"`.

For a point pattern `X` and a covariate `Z`, the AUC is a numerical index that measures the ability of the covariate to separate the spatial domain into areas of high and low density of points. Let x_i be a randomly-chosen data point from `X` and U a randomly-selected location in the study region. The AUC is the probability that $Z(x_i) > Z(U)$ assuming `high=TRUE`. That is, AUC is the probability that a randomly-selected data point has a higher value of the covariate `Z` than does a randomly-selected spatial location. The AUC is a number between 0 and 1. A value of 0.5 indicates a complete lack of discriminatory power.

For a fitted point process model `X`, the AUC measures the ability of the fitted model intensity to separate the spatial domain into areas of high and low density of points. Suppose $\lambda(u)$ is the intensity function of the model. The AUC is the probability that $\lambda(x_i) > \lambda(U)$. That is, AUC is the probability that a randomly-selected data point has higher predicted intensity than does a randomly-selected spatial location. The AUC is **not** a measure of the goodness-of-fit of the model (Lobo et al, 2007).

Value

Numeric. For `auc.lpp`, the result is a single number giving the AUC value. For `auc.lppm`, the result is a numeric vector of length 2 giving the AUC value and the theoretically expected AUC value for this model.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`, Rolf Turner `<rolfturner@posteo.net>` and Ege Rubak `<rubak@math.aau.dk>`.

References

Lobo, J.M., Jiménez-Valverde, A. and Real, R. (2007) AUC: a misleading measure of the performance of predictive distribution models. *Global Ecology and Biogeography* **17**(2) 145–151.

Nam, B.-H. and D'Agostino, R. (2002) Discrimination index, the area under the ROC curve. Pages 267–279 in Huber-Carol, C., Balakrishnan, N., Nikulin, M.S. and Mesbah, M., *Goodness-of-fit tests and model validity*, Birkhäuser, Basel.

See Also

`auc`, `roc`, `roc.lpp`

Examples

```
auc(spiders, "x")
fit <- lppm(spiders ~ x + y)
auc(fit)
```

`begins` *Check Start of Character String*

Description

Checks whether a character string begins with a particular prefix.

Usage

```
begins(x, firstbit)
```

Arguments

`x`	Character string, or vector of character strings, to be tested.
`firstbit`	A single character string.

Details

This simple wrapper function checks whether (each entry in) `x` begins with the string `firstbit`, and returns a logical value or logical vector with one entry for each entry of `x`. This function is useful mainly for reducing complexity in model formulae.

Value

Logical vector of the same length as `x`.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>

Rolf Turner <`rolfturner@posteo.net`>

and Ege Rubak <`rubak@math.aau.dk`>

Examples

```
begins(c("Hello", "Goodbye"), "Hell")
begins("anything", "")
```

`berman.test.lpp` *Berman's Tests for Point Process Model on a Network*

Description

Tests the goodness-of-fit of a Poisson point process model on a linear network, using the approach of Berman (1986).

Usage

```
## S3 method for class 'lpp'
berman.test(X, covariate,
                         which = c("Z1", "Z2"),
        alternative = c("two.sided", "less", "greater"), ...)

## S3 method for class 'lppm'
berman.test(model, covariate,
                         which = c("Z1", "Z2"),
               alternative = c("two.sided", "less", "greater"), ...)
```

Arguments

`X`	A point pattern (object of class `"lpp"`).
`model`	A fitted point process model (object of class `"lppm"`).
`covariate`	The spatial covariate on which the test will be based. An image (object of class `"im"` or `"linim"`) or a function.
`which`	Character string specifying the choice of test.
`alternative`	Character string specifying the alternative hypothesis.
`...`	Additional arguments controlling the pixel resolution (arguments `dimyx` and `eps` passed to `as.mask`) or other undocumented features.

Details

These functions perform a goodness-of-fit test of a Poisson point process model fitted to point pattern data. The observed distribution of the values of a spatial covariate at the data points, and the predicted distribution of the same values under the model, are compared using either of two test statistics Z_1 and Z_2 proposed by Berman (1986). The Z_1 test is also known as the Lawson-Waller test.

The function `berman.test` is generic, with methods for point patterns (`"ppp"` or `"lpp"`) and point process models (`"ppm"` or `"lppm"`).

See the help file for `berman.test` for information on the generic function and the methods for data in two-dimensional space, classes `"ppp"` and `"ppm"`.

This help file describes the methods for data on a linear network, classes `"lpp"` and `"lppm"`.

- If `X` is a point pattern dataset (object of class `"ppp"` or `"lpp"`), then `berman.test(X, ...)` performs a goodness-of-fit test of the uniform Poisson point process (Complete Spatial Randomness, CSR) for this dataset.
- If `model` is a fitted point process model (object of class `"ppm"` or `"lppm"`) then `berman.test(model, ...)` performs a test of goodness-of-fit for this fitted model. In this case, `model` should be a Poisson point process.

The test is performed by comparing the observed distribution of the values of a spatial covariate at the data points, and the predicted distribution of the same covariate under the model. Thus, you must nominate a spatial covariate for this test.

The argument `covariate` should be either a `function(x,y)` or a pixel image (object of class `"im"` containing the values of a spatial function. If `covariate` is an image, it should have numeric values, and its domain should cover the observation window of the `model`. If `covariate` is a function, it should expect two arguments `x` and `y` which are vectors of coordinates, and it should return a numeric vector of the same length as `x` and `y`.

First the original data point pattern is extracted from `model`. The values of the `covariate` at these data points are collected.

Next the values of the `covariate` at all locations in the observation window are evaluated. The point process intensity of the fitted model is also evaluated at all locations in the window.

- If `which="Z1"`, the test statistic Z_1 is computed as follows. The sum S of the covariate values at all data points is evaluated. The predicted mean μ and variance σ^2 of S are computed from the values of the covariate at all locations in the window. Then we compute $Z_1 = (S - \mu)/\sigma$. Closely-related tests were proposed independently by Waller et al (1993) and Lawson (1993) so this test is often termed the Lawson-Waller test in epidemiological literature.
- If `which="Z2"`, the test statistic Z_2 is computed as follows. The values of the `covariate` at all locations in the observation window, weighted by the point process intensity, are compiled into a cumulative distribution function F. The probability integral transformation is then applied: the values of the `covariate` at the original data points are transformed by the predicted cumulative distribution function F into numbers between 0 and 1. If the model is correct, these numbers are i.i.d. uniform random numbers. The standardised sample mean of these numbers is the statistic Z_2.

In both cases the null distribution of the test statistic is the standard normal distribution, approximately.

The return value is an object of class `"htest"` containing the results of the hypothesis test. The print method for this class gives an informative summary of the test outcome.

Value

An object of class `"htest"` (hypothesis test) and also of class `"bermantest"`, containing the results of the test. The return value can be plotted (by `plot.bermantest`) or printed to give an informative summary of the test.

Warning

The meaning of a one-sided test must be carefully scrutinised: see the printed output.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>, Rolf Turner <`rolfturner@posteo.net`> and Ege Rubak <`rubak@math.aau.dk`>.

References

Berman, M. (1986) Testing for spatial association between a point process and another stochastic process. *Applied Statistics* **35**, 54–62.

Lawson, A.B. (1993) On the analysis of mortality events around a prespecified fixed point. *Journal of the Royal Statistical Society, Series A* **156** (3) 363–377.

Waller, L., Turnbull, B., Clark, L.C. and Nasca, P. (1992) Chronic Disease Surveillance and testing of clustering of disease and exposure: Application to leukaemia incidence and TCE-contaminated dumpsites in upstate New York. *Environmetrics* **3**, 281–300.

See Also

`cdf.test`, `quadrat.test`, `ppm` `lppm`

Examples

```
#' test of complete randomness
berman.test(spiders, "x")
#' test of fitted model
fit <- lppm(spiders ~ x)
berman.test(fit, "y", "Z2")
```

`branchlabelfun` *Tree Branch Membership Labelling Function*

Description

Creates a function which returns the tree branch membership label for any location on a linear network.

Usage

```
branchlabelfun(L, root = 1)
```

Arguments

`L`	Linear network (object of class `"linnet"`). The network must have no loops.
`root`	Root of the tree. An integer index identifying which point in `vertices(L)` is the root of the tree.

Details

The linear network `L` must be an acyclic graph (i.e. must not contain any loops) so that it can be interpreted as a tree.

The result of `f <- branchlabelfun(L, root)` is a function `f` which gives, for each location on the linear network `L`, the tree branch label at that location.

Tree branch labels are explained in `treebranchlabels`.

The result `f` also belongs to the class `"linfun"`. It can be called using several different kinds of data, as explained in the help for `linfun`. The values of the function are character strings.

Value

A function (of class `"linfun"`).

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>

Rolf Turner <`rolfturner@posteo.net`>

and Ege Rubak <`rubak@math.aau.dk`>

See Also

`treebranchlabels`, `linfun`

Examples

```
# make a simple tree
m <- simplenet$m
m[8,10] <- m[10,8] <- FALSE
L <- linnet(vertices(simplenet), m)
# make function
f <- branchlabelfun(L, 1)
plot(f)
X <- runiflpp(5, L)
f(X)
```

`bw.lppl` *Likelihood Cross Validation Bandwidth Selection for Kernel Density on a Linear Network*

Description

Uses likelihood cross-validation to select a smoothing bandwidth for the kernel estimation of point process intensity on a linear network.

Usage

```
bw.lppl(X, ..., srange=NULL, ns=16, sigma=NULL, weights=NULL,
        distance=c("euclidean", "path"), shortcut=TRUE, warn=TRUE)
```

Arguments

`X`	A point pattern on a linear network (object of class `"lpp"`).
`srange`	Optional numeric vector of length 2 giving the range of values of bandwidth to be searched.
`ns`	Optional integer giving the number of values of bandwidth to search.
`sigma`	Optional. Vector of values of the bandwidth to be searched. Overrides the values of `ns` and `srange`.
`weights`	Optional. Numeric vector of weights for the points of `X`. Argument passed to `density.lpp`.
`distance`	Argument passed to `density.lpp` controlling the type of kernel estimator.
`...`	Additional arguments passed to `density.lpp`.
`shortcut`	Logical value indicating whether to speed up the calculation by omitting the integral term in the cross-validation criterion.
`warn`	Logical. If `TRUE`, issue a warning if the maximum of the cross-validation criterion occurs at one of the ends of the search interval.

Details

This function selects an appropriate bandwidth `sigma` for the kernel estimator of point process intensity computed by `density.lpp`.

The argument `X` should be a point pattern on a linear network (class `"lpp"`).

The bandwidth σ is chosen to maximise the point process likelihood cross-validation criterion

$$\mathrm{LCV}(\sigma) = \sum_i \log \hat\lambda_{-i}(x_i) - \int_L \hat\lambda(u)\,\mathrm{d}u$$

where the sum is taken over all the data points x_i, where $\hat\lambda_{-i}(x_i)$ is the leave-one-out kernel-smoothing estimate of the intensity at x_i with smoothing bandwidth σ, and $\hat\lambda(u)$ is the kernel-smoothing estimate of the intensity at a spatial location u with smoothing bandwidth σ. See Loader(1999, Section 5.3).

The value of $\mathrm{LCV}(\sigma)$ is computed directly, using `density.lpp`, for `ns` different values of σ between `srange[1]` and `srange[2]`.

The result is a numerical value giving the selected bandwidth. The result also belongs to the class `"bw.optim"` which can be plotted to show the (rescaled) mean-square error as a function of `sigma`.

If `shortcut=TRUE`, the computation is accelerated by omitting the integral term in the equation above. This is valid because the integral is approximately constant.

Value

A single numerical value giving the selected bandwidth. The result also belongs to the class `"bw.optim"` (see `bw.optim.object`) which can be plotted to show the bandwidth selection criterion as a function of `sigma`.

Author(s)

Greg McSwiggan, Suman Rakshit and Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`.

References

Loader, C. (1999) *Local Regression and Likelihood.* Springer, New York.

McSwiggan, G., Baddeley, A. and Nair, G. (2019) Estimation of relative risk for events on a linear network. *Statistics and Computing* **30** (2) 469–484.

See Also

`density.lpp`, `bw.scott`.

`bw.optim.object`.

For point patterns in two-dimensional space, use `bw.ppl`.

Examples

```
if(interactive()) {
  b <- bw.lppl(spiders)
  plot(b, main="Likelihood cross validation for spiders")
  plot(density(spiders, b, distance="e"))
} else {
  b1 <- bw.lppl(spiders, ns=2)
  b2 <- bw.lppl(spiders, ns=2, shortcut=FALSE)
}
```

`bw.relrisk.lpp` *Cross Validated Bandwidth Selection for Relative Risk Estimation on a Network*

Description

Uses cross-validation to select a smoothing bandwidth for the estimation of relative risk on a linear network.

Usage

```
  ## S3 method for class 'lpp'
bw.relrisk(X, ...,
    method = c("likelihood", "leastsquares", "KelsallDiggle", "McSwiggan"),
    distance=c("path", "euclidean"),
    hmin = NULL, hmax = NULL, nh = NULL,
    fast = TRUE, fastmethod = "onestep",
    floored = TRUE, reference = c("thumb", "uniform", "sigma"),
    allow.infinite = TRUE, epsilon = 1e-20, fudge = 0,
    verbose = FALSE, warn = TRUE)
```

Arguments

`X`	A multitype point pattern on a linear network (object of class `"lpp"` which has factor-valued marks).
`...`	Arguments passed to `density.lpp` to control the resolution of the algorithm.
`method`	Character string (partially matched) determining the cross-validation method. See Details.
`distance`	Character string (partially matched) specifying the type of smoothing kernel. See `density.lpp`.
`hmin`, `hmax`	Optional. Numeric values. Range of trial values of smoothing bandwith `sigma` to consider. There is a sensible default.
`nh`	Number of trial values of smoothing bandwidth `sigma` to consider.
`fast`	Logical value specifying whether the leave-one-out density estimates should be computed using a fast approximation (`fast=TRUE`, the default) or exactly (`fast=FALSE`).
`fastmethod`, `floored`	Developer use only.
`reference`	Character string (partially matched) specifying the bandwidth for calculating the reference intensities used in the McSwiggan method (modified Kelsall-Diggle method). `reference="sigma"` means the maximum bandwidth considered, which is given by the argument `sigma`. `reference="thumb"` means the bandwidths selected by Scott's rule of thumb `bw.scott.iso`. `reference="uniform"` means infinite bandwidth corresponding to uniform intensity.
`allow.infinite`	Logical value indicating whether an infinite bandwidth (corresponding to a constant relative risk) should be permitted as a possible choice of bandwidth.
`epsilon`	A small constant value added to the reference density in some of the cross-validation calculations, to improve performance.
`fudge`	Fudge factor to prevent very small density estimates in the leave-one-out calculation. If `fudge > 0`, then the lowest permitted value for a leave-one-out estimate of intensity is `fudge/L`, where `L` is the total length of the network.
`verbose`	Logical value indicating whether to print progress reports,
`warn`	Logical. If `TRUE`, issue a warning if the minimum of the cross-validation criterion occurs at one of the ends of the search interval.

Details

This function is a method for the generic `bw.relrisk`. It computes an optimal value of smoothing bandwidth for the nonparametric estimation of relative risk on a linear network using `relrisk.lpp`. The optimal value is found by minimising a cross-validation criterion.

The cross-validation criterion is selected by the argument `method`:

`method="likelihood"`	(negative) likelihood cross-validation
`method="leastsquares"`	least squares cross-validation
`method="KelsallDiggle"`	Kelsall and Diggle (1995) density ratio cross-validation
`method="McSwiggan"`	McSwiggan et al (2019) modified density ratio cross-validation

See McSwiggan et al (2019) for details.

The result is a numerical value giving the selected bandwidth `sigma`. The result also belongs to the class `"bw.optim"` allowing it to be printed and plotted. The plot shows the cross-validation criterion as a function of bandwidth. The 'optimal' bandwidth is the value of bandwidth which minimises the cross-validation criterion.

The range of values for the smoothing bandwidth `sigma` is set by the arguments `hmin, hmax`. There is a sensible default, based on the linear network version of Scott's rule `bw.scott.iso`.

If the optimal bandwidth is achieved at an endpoint of the interval `[hmin, hmax]`, the algorithm will issue a warning (unless `warn=FALSE`). If this occurs, then it is probably advisable to expand the interval by changing the arguments `hmin, hmax`.

The cross-validation procedure is based on kernel estimates of intensity, which are computed by `density.lpp`. Any arguments `...` are passed to `density.lpp` to control the kernel estimation procedure. This includes the argument `distance` which specifies the type of kernel. The default is `distance="path"`; the fastest option is `distance="euclidean"`.

Value

A single numerical value giving the selected bandwidth. The result also belongs to the class `"bw.optim"` (see `bw.optim.object`) which can be plotted to show the bandwidth selection criterion as a function of `sigma`.

Author(s)

Greg McSwiggan and Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`.

References

Kelsall, J.E. and Diggle, P.J. (1995) Kernel estimation of relative risk. *Bernoulli* **1**, 3–16.

McSwiggan, G., Baddeley, A. and Nair, G. (2019) Estimation of relative risk for events on a linear network. *Statistics and Computing* **30** (2) 469–484.

See Also

`relrisk.lpp`, `bw.relrisk`, `bw.optim.object`

Examples

```
set.seed(2020)
X <- superimpose(A=runiflpp(20, simplenet),
                 B=runifpointOnLines(20, as.psp(simplenet)[1]))
plot(bw.relrisk(X, hmin=0.13, hmax=0.22, method="McSwiggan"))
plot(bw.relrisk(X, hmin=0.1, hmax=0.2, nh=8, distance="euclidean"))
```

`bw.voronoi` *Cross Validated Bandwidth Selection for Voronoi Estimator of Intensity on a Network*

Description

Uses cross-validation to select a smoothing bandwidth for the Voronoi estimate of point process intensity on a linear network.

Usage

```
bw.voronoi(X, ..., probrange = c(0.2, 0.8), nprob = 10,
           prob = NULL, nrep = 100, verbose = TRUE, warn=TRUE)
```

Arguments

`X`	Point pattern on a linear network (object of class `"lpp"`).
`...`	Ignored.
`probrange`	Numeric vector of length 2 giving the range of bandwidths (retention probabilities) to be assessed.
`nprob`	Integer. Number of bandwidths to be assessed.
`prob`	Optional. A numeric vector of bandwidths (retention probabilities) to be assessed. Entries must be probabilities between 0 and 1. Overrides `nprob` and `probrange`.
`nrep`	Number of simulated realisations to be used for the computation.
`verbose`	Logical value indicating whether to print progress reports.
`warn`	Logical. If `TRUE`, issue a warning if the maximum of the cross-validation criterion occurs at one of the ends of the search interval.

Details

This function uses likelihood cross-validation to choose the optimal value of the thinning fraction `f` (the retention probability) to be used in the smoothed Voronoi estimator of point process intensity `densityVoronoi.lpp`.

Value

A single numerical value giving the selected bandwidth. The result also belongs to the class `"bw.optim"` (see `bw.optim.object`) which can be plotted to show the bandwidth selection criterion as a function of `sigma`.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`, Rolf Turner `<rolfturner@posteo.net>` and Ege Rubak `<rubak@math.aau.dk>` and Mehdi Moradi `<m2.moradi@yahoo.com>`.

References

Moradi, M., Cronie, 0., Rubak, E., Lachieze-Rey, R., Mateu, J. and Baddeley, A. (2019) Resample-smoothing of Voronoi intensity estimators. *Statistics and Computing* **29** (5) 995–1010.

See Also

`densityVoronoi.lpp`, `bw.optim.object`

Examples

```
np <- if(interactive()) 10 else 3
nr <- if(interactive()) 100 else 2
b <- bw.voronoi(spiders, nprob=np, nrep=nr)
b
plot(b)
```

`cdf.test.lpp` *Spatial Distribution Test for Points on a Linear Network*

Description

Performs a test of goodness-of-fit of a point process model on a linear network. The observed and predicted distributions of the values of a spatial covariate are compared using either the Kolmogorov-Smirnov test, Cramér-von Mises test or Anderson-Darling test. For non-Poisson models, a Monte Carlo test is used.

Usage

```
## S3 method for class 'lpp'
cdf.test(X, covariate,  test=c("ks", "cvm", "ad"), ...,
        interpolate=TRUE, jitter=TRUE)

## S3 method for class 'lppm'
cdf.test(model, covariate,  test=c("ks", "cvm", "ad"), ...,
          interpolate=TRUE, jitter=TRUE, nsim=99, verbose=TRUE)
```

Arguments

`X`	A point pattern on a linear network (object of class `"lpp"`).
`model`	A fitted point process model on a linear network (object of class `"lppm"`)
`covariate`	The spatial covariate on which the test will be based. A function, a pixel image (object of class `"im"` or `"linim"`), a list of pixel images, or one of the characters `"x"` or `"y"` indicating the Cartesian coordinates.
`test`	Character string identifying the test to be performed: `"ks"` for Kolmogorov-Smirnov test, `"cvm"` for Cramér-von Mises test or `"ad"` for Anderson-Darling test.

`...`	Arguments passed to `ks.test` (from the **stats** package) or `cvm.test` or `ad.test` (from the **goftest** package) to control the test.
`interpolate`	Logical flag indicating whether to interpolate pixel images. If `interpolate=TRUE`, the value of the covariate at each point of `X` will be approximated by interpolating the nearby pixel values. If `interpolate=FALSE`, the nearest pixel value will be used.
`jitter`	Logical flag. If `jitter=TRUE`, values of the covariate will be slightly perturbed at random, to avoid tied values in the test.
`nsim`	Number of simulated realisations from the `model` to be used for the Monte Carlo test, when `model` is not a Poisson process.
`verbose`	Logical value indicating whether to print progress reports when performing a Monte Carlo test.

Details

These functions perform a goodness-of-fit test of a Poisson point process model fitted to point pattern data on a linear network. The observed distribution of the values of a spatial covariate at the data points, and the predicted distribution of the same values under the model, are compared using the Kolmogorov-Smirnov test, the Cramér-von Mises test or the Anderson-Darling test. For Gibbs models, a Monte Carlo test is performed using these test statistics.

The function `cdf.test` is generic, with methods for point patterns (`"ppp"` or `"lpp"`), point process models (`"ppm"` or `"lppm"`) and spatial logistic regression models (`"slrm"`).

See the help file for `cdf.test` for information on the generic function and the methods for data in two-dimensional space, classes `"ppp"`, `"ppm"` and `"slrm"`.

This help file describes the methods for data on a linear network, classes `"lpp"` and `"lppm"`.

- If `X` is a point pattern on a linear network (object of class `"lpp"`), then `cdf.test(X, ...)` performs a goodness-of-fit test of the uniform Poisson point process (Complete Spatial Randomness, CSR) for this dataset. For a multitype point pattern, the uniform intensity is assumed to depend on the type of point (sometimes called Complete Spatial Randomness and Independence, CSRI).
- If `model` is a fitted point process model on a network (object of class `"lppm"`) then `cdf.test(model, ...)` performs a test of goodness-of-fit for this fitted model.

The test is performed by comparing the observed distribution of the values of a spatial covariate at the data points, and the predicted distribution of the same covariate under the model, using a classical goodness-of-fit test. Thus, you must nominate a spatial covariate for this test.

If `X` is a point pattern that does not have marks, the argument `covariate` should be either a `function(x,y)` or a pixel image (object of class `"im"` or `"linim"`) containing the values of a spatial function, or one of the characters `"x"` or `"y"` indicating the Cartesian coordinates. If `covariate` is an image, it should have numeric values, and its domain should cover the observation window of the `model`. If `covariate` is a function, it should expect two arguments `x` and `y` which are vectors of coordinates, and it should return a numeric vector of the same length as `x` and `y`.

If `X` is a multitype point pattern, the argument `covariate` can be either a `function(x,y,marks)`, or a pixel image, or a list of pixel images corresponding to each possible mark value, or one of the characters `"x"` or `"y"` indicating the Cartesian coordinates.

First the original data point pattern is extracted from `model`. The values of the `covariate` at these data points are collected.

The predicted distribution of the values of the `covariate` under the fitted `model` is computed as follows. The values of the `covariate` at all locations in the observation window are evaluated, weighted according to the point process intensity of the fitted model, and compiled into a cumulative distribution function F using `ewcdf`.

The probability integral transformation is then applied: the values of the `covariate` at the original data points are transformed by the predicted cumulative distribution function F into numbers between 0 and 1. If the model is correct, these numbers are i.i.d. uniform random numbers. The A goodness-of-fit test of the uniform distribution is applied to these numbers using `stats::ks.test`, `goftest::cvm.test` or `goftest::ad.test`.

This test was apparently first described (in the context of two-dimensional spatial data, and using Kolmogorov-Smirnov) by Berman (1986). See also Baddeley et al (2005).

If `model` is not a Poisson process, then a Monte Carlo test is performed, by generating `nsim` point patterns which are simulated realisations of the `model`, re-fitting the model to each simulated point pattern, and calculating the test statistic for each fitted model. The Monte Carlo p value is determined by comparing the simulated values of the test statistic with the value for the original data.

The return value is an object of class `"htest"` containing the results of the hypothesis test. The print method for this class gives an informative summary of the test outcome.

The return value also belongs to the class `"cdftest"` for which there is a plot method `plot.cdftest`. The plot method displays the empirical cumulative distribution function of the covariate at the data points, and the predicted cumulative distribution function of the covariate under the model, plotted against the value of the covariate.

The argument `jitter` controls whether covariate values are randomly perturbed, in order to avoid ties. If the original data contains any ties in the covariate (i.e. points with equal values of the covariate), and if `jitter=FALSE`, then the Kolmogorov-Smirnov test implemented in `ks.test` will issue a warning that it cannot calculate the exact p-value. To avoid this, if `jitter=TRUE` each value of the covariate will be perturbed by adding a small random value. The perturbations are normally distributed with standard deviation equal to one hundredth of the range of values of the covariate. This prevents ties, and the p-value is still correct. There is a very slight loss of power.

Value

An object of class `"htest"` containing the results of the test. See `ks.test` for details. The return value can be printed to give an informative summary of the test.

The value also belongs to the class `"cdftest"` for which there is a plot method.

Warning

The outcome of the test involves a small amount of random variability, because (by default) the coordinates are randomly perturbed to avoid tied values. Hence, if `cdf.test` is executed twice, the p-values will not be exactly the same. To avoid this behaviour, set `jitter=FALSE`.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>` and Rolf Turner `<rolfturner@posteo.net>`

References

Baddeley, A., Turner, R., Møller, J. and Hazelton, M. (2005) Residual analysis for spatial point processes. *Journal of the Royal Statistical Society, Series B* **67**, 617–666.

Berman, M. (1986) Testing for spatial association between a point process and another stochastic process. *Applied Statistics* **35**, 54–62.

See Also

`plot.cdftest`, `quadrat.test`, `berman.test`, `ks.test`, `goftest::cvm.test`, `goftest::ad.test`, `lppm`

Examples

```
op <- options(useFancyQuotes=FALSE)

# test of CSR using x coordinate
cdf.test(spiders, "x")

# fit inhomogeneous Poisson model and test
model <- lppm(spiders ~x)
cdf.test(model, "y")

# test of CSR using a function of x and y
fun <- function(x,y){2* x + y}
cdf.test(spiders, fun)

# test of CSR using an image covariate
fim <- as.linim(fun, domain(spiders))
cdf.test(spiders, fim)

options(op)
```

`chop.linnet` *Divide a Linear Network into Tiles Using Infinite Lines*

Description

Given a linear network and a set of infinite lines, divide the network into tiles demarcated by the lines. The result is a tessellation of the network.

Usage

```
chop.linnet(X, L)
```

Arguments

`X`	Linear network (object of class `"linnet"`) or data acceptable to `as.linnet`.
`L`	Infinite line or lines (object of class `"infline"`).

Details

The first line of `L` divides `X` into two tiles. Subsequent lines divide each of these tiles. The result is a tessellation of `X`. Tiles are not necessarily connected sets.

Value

Tessellation on a linear network (object of class `"lintess"`).

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`.

See Also

`crossing.linnet` to determine the crossing points between the lines and the network.

`divide.linnet` to divide a network into a tessellation using arbitrary cut points.

Examples

```
L <- infline(p=runif(3), theta=runif(3, max=pi/2))
Y <- chop.linnet(simplenet, L)
plot(Y, main="")
plot(L, col="red")
```

`clickjoin` *Interactively join vertices on a plot*

Description

Given a point pattern representing a set of vertices, this command gives a point-and-click interface allowing the user to join pairs of selected vertices by edges.

Usage

```
clickjoin(X, ..., add = TRUE, m = NULL, join = TRUE)
```

Arguments

`X`	Point pattern of vertices. An object of class `"ppp"`.
`...`	Arguments passed to `segments` to control the plotting of the new edges.
`add`	Logical. Whether the point pattern `X` should be added to the existing plot (`add=TRUE`) or a new plot should be created (`add=FALSE`).
`m`	Optional. Logical matrix specifying an initial set of edges. There is an edge between vertices `i` and `j` if `m[i,j] = TRUE`.
`join`	Optional. If `TRUE`, then each user click will join a pair of vertices. If `FALSE`, then each user click will delete an existing edge. This is only relevant if `m` is supplied.

Details

This function makes it easier for the user to create a linear network or a planar graph, given a set of vertices.

The function first displays the point pattern `X`, then repeatedly prompts the user to click on a pair of points in `X`. Each selected pair of points will be joined by an edge. The function returns a logical matrix which has entries equal to `TRUE` for each pair of vertices joined by an edge.

The selection of points is performed using `identify.ppp` which typically expects the user to click the left mouse button. This point-and-click interaction continues until the user terminates it, by pressing the middle mouse button, or pressing the right mouse button and selecting `stop`.

The return value can be used in `linnet` to create a linear network.

Value

Logical matrix `m` with value `m[i,j] = TRUE` for every pair of vertices `X[i]` and `X[j]` that should be joined by an edge.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`.

See Also

`linnet`, `clickppp`

`clicklpp` *Interactively Add Points on a Linear Network*

Description

Allows the user to create a point pattern on a linear network by point-and-click in the display.

Usage

```
clicklpp(L, n=NULL, types=NULL, ...,
         add=FALSE, main=NULL, hook=NULL)
```

Arguments

`L`	Linear network on which the points will be placed. An object of class `"linnet"`.
`n`	Number of points to be added (if this is predetermined).
`types`	Vector of types, when creating a multitype point pattern.
`...`	Optional extra arguments to be passed to `locator` to control the display.
`add`	Logical value indicating whether to create a new plot (`add=FALSE`) or draw over the existing plot (`add=TRUE`).
`main`	Main heading for plot.
`hook`	For internal use only. Do not use this argument.

Details

This function allows the user to create a point pattern on a linear network by interactively clicking on the screen display.

First the linear network `L` is plotted on the current screen device. Then the user is prompted to point the mouse at any desired locations and click the left mouse button to add each point. Interactive input stops after `n` clicks (if `n` was given) or when the middle mouse button is pressed.

The return value is a point pattern on the network `L`, containing the locations of all the clicked points, after they have been projected onto the network `L`. Any points that were clicked outside the bounding window of the network will be ignored.

If the argument `types` is given, then a multitype point pattern will be created. The user is prompted to input the locations of points of type `type[i]`, for each successive index `i`. (If the argument `n` was given, there will be `n` points of *each* type.) The return value is a multitype point pattern on a linear network.

This function uses the R command `locator` to input the mouse clicks. It only works on screen devices such as 'X11', 'windows' and 'quartz'. Arguments that can be passed to `locator` through `...` include `pch` (plotting character), `cex` (character expansion factor) and `col` (colour). See `locator` and `par`.

Value

A point pattern (object of class `"lpp"`).

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>, Rolf Turner <`rolfturner@posteo.net`> and Ege Rubak <`rubak@math.aau.dk`>, based on an idea by Dominic Schuhmacher.

See Also

`clickppp`, `identify.lpp`, `locator`, `clickpoly`, `clickbox`, `clickdist`

`connected.linnet` *Connected Components of a Linear Network*

Description

Find the topologically-connected components of a linear network.

Usage

```
## S3 method for class 'linnet'
connected(X, ..., what = c("labels", "components"))
```

Arguments

`X`	A linear network (object of class `"linnet"`).
`...`	Ignored.
`what`	Character string specifying the kind of result.

Details

The function `connected` is generic. This is the method for linear networks (objects of class `"linnet"`).

Two vertices of the network are connected if they are joined by a path in the network. This function divides the network into subsets, such that all points in a subset are connected to each other.

If `what="labels"` the return value is a factor with one entry for each vertex of `X`, identifying which connected component the vertex belongs to.

If `what="components"` the return value is a list of linear networks, which are the connected components of `X`.

Value

If `what="labels"`, a factor. If `what="components"`, a list of linear networks.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`> and Suman Rakshit.

See Also

`thinNetwork`

Examples

```
# remove some edges from a network to make it disconnected
plot(simplenet, col="grey", main="", lty=2)
A <- thinNetwork(simplenet, retainedges=-c(3,5))
plot(A, add=TRUE, lwd=2)
# find the connected components
connected(A)
cA <- connected(A, what="components")
plot(cA[[1]], add=TRUE, col="green", lwd=2)
plot(cA[[2]], add=TRUE, col="blue", lwd=2)
```

`connected.lpp` *Connected Components of a Point Pattern on a Linear Network*

Description

Finds the topologically-connected components of a point pattern on a linear network, when all pairs of points closer than a threshold distance are joined.

Usage

```
## S3 method for class 'lpp'
connected(X, R=Inf, ..., dismantle=TRUE)
```

Arguments

`X`	A linear network (object of class `"lpp"`).
`R`	Threshold distance. Pairs of points will be joined together if they are closer than `R` units apart, measured by the shortest path in the network. The default `R=Inf` implies that points will be joined together if they are mutually connected by any path in the network.
`dismantle`	Logical. If `TRUE` (the default), the network itself will be divided into its path-connected components using `connected.linnet`.
`...`	Ignored.

Details

The function `connected` is generic. This is the method for point patterns on a linear network (objects of class `"lpp"`). It divides the point pattern `X` into one or more groups of points.

If `R=Inf` (the default), then `X` is divided into groups such that any pair of points in the same group can be joined by a path in the network.

If `R` is a finite number, then two points of `X` are declared to be *R-close* if they lie closer than `R` units apart, measured by the length of the shortest path in the network. Two points are *R-connected* if they can be reached by a series of steps between R-close pairs of points of `X`. Then `X` is divided into groups such that any pair of points in the same group is R-connected.

If `dismantle=TRUE` (the default) the algorithm first checks whether the network is connected (i.e. whether any pair of vertices can be joined by a path in the network), and if not, the network is decomposed into its connected components.

Value

A point pattern (of class `"lpp"`) with marks indicating the grouping, or a list of such point patterns.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>.

See Also

thinNetwork

Examples

```
## behaviour like connected.ppp
U <- runiflpp(20, simplenet)
plot(connected(U, 0.15, dismantle=FALSE))

## behaviour like connected.owin
## remove some edges from a network to make it disconnected
plot(simplenet, col="grey", main="", lty=2)
A <- thinNetwork(simplenet, retainedges=-c(3,5))
plot(A, add=TRUE, lwd=2)
X <- runiflpp(10, A)
## find the connected components
cX <- connected(X)
plot(cX[[1]], add=TRUE, col="blue", lwd=2)
```

`crossdist.lpp` *Pairwise distances between two point patterns on a linear network*

Description

Computes the distances between pairs of points taken from two different point patterns on the same linear network.

Usage

```
  ## S3 method for class 'lpp'
crossdist(X, Y, ..., method="C", check=TRUE)
```

Arguments

`X, Y`	Point patterns on a linear network (objects of class `"lpp"`). They must lie on the *same* network.
`...`	Ignored.
`method`	String specifying which method of calculation to use when the network data use the non-sparse representation. Values are `"C"` and `"interpreted"`.
`check`	Logical value specifying whether to check that `X` and `Y` are defined on the same network. Default is `check=TRUE`. Setting `check=FALSE` will save time, but should only be used if it is certain that the two networks are identical.

Details

Given two point patterns on a linear network, this function computes the distance from each point in the first pattern to each point in the second pattern, measuring distance by the shortest path along the network.

This is a method for the generic function `crossdist` for the class of point patterns on a linear network (objects of class `"lpp"`).

This function expects two point pattern objects `X` and `Y` on the *same* linear network, and returns the matrix whose `[i,j]` entry is the shortest-path distance from `X[i]` to `Y[j]`.

If two points cannot be joined by a path, the distance between them is infinite (`Inf`).

The argument `method` is not normally used. It is retained only for developers to check the validity of the software.

Value

A matrix whose `[i,j]` entry is the distance from the `i`-th point in `X` to the `j`-th point in `Y`. Matrix entries are nonnegative numbers or infinity (`Inf`).

Algorithms and accuracy

Distances are accurate within the numerical tolerance of the network, `summary(X)$toler`.

For network data stored in the non-sparse representation described in `linnet`, then pairwise distances are computed using the matrix of path distances between vertices of the network, using R code if `method = "interpreted"`, or using C code if `method="C"` (the default).

For networks stored in the sparse representation, the argument `method` has no effect, and the distances are computed using an efficient C algorithm.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>.

See Also

`crossdist`, `crossdist.ppp`, `pairdist`, `nndist`

Examples

```
v <- split(chicago)
X <- v$cartheft
Y <- v$burglary
d <- crossdist(X, Y)
d[1:3,1:4]
```

`crossing.linnet` *Crossing Points between Linear Network and Other Lines*

Description

Find all the crossing-points between a linear network and another pattern of lines or line segments.

Usage

```
crossing.linnet(X, Y)
```

Arguments

`X`	Linear network (object of class `"linnet"`).
`Y`	A linear network, or a spatial pattern of line segments (class `"psp"`) or infinite lines (class `"infline"`).

Details

All crossing-points between `X` and `Y` are determined. The result is a point pattern on the network `X`.

Value

Point pattern on a linear network (object of class `"lpp"`).

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`.

See Also

`crossing.psp`

Examples

```
plot(simplenet, main="")
L <- infline(p=runif(3), theta=runif(3, max=pi/2))
plot(L, col="red")
Y <- crossing.linnet(simplenet, L)
plot(Y, add=TRUE, cols="blue")
```

`cut.lpp` *Classify Points in a Point Pattern on a Network*

Description

For a point pattern on a linear network, classify the points into distinct types according to the numerical marks in the pattern, or according to another variable.

Usage

```
  ## S3 method for class 'lpp'
cut(x, z=marks(x), ...)
```

Arguments

`x`	A point pattern on a linear network (object of class `"lpp"`).
`z`	Data determining the classification. A numeric vector, a factor, a pixel image on a linear network (class `"linim"`), a function on a linear network (class `"linfun"`), a tessellation on a linear network (class `"lintess"`), a string giving the name of a column of marks, or one of the coordinate names `"x"`, `"y"`, `"seg"` or `"tp"`.
`...`	Arguments passed to `cut.default`. They determine the breakpoints for the mapping from numerical values in `z` to factor values in the output. See `cut.default`.

Details

This function has the effect of classifying each point in the point pattern `x` into one of several possible types. The classification is based on the dataset `z`, which may be either

- a factor (of length equal to the number of points in `z`) determining the classification of each point in `x`. Levels of the factor determine the classification.
- a numeric vector (of length equal to the number of points in `z`). The range of values of `z` will be divided into bands (the number of bands is determined by `...`) and `z` will be converted to a factor using `cut.default`.
- a pixel image on a network (object of class `"linim"`). The value of `z` at each point of `x` will be used as the classifying variable.
- a function on a network (object of class `"linfun"`, see `linfun`). The value of `z` at each point of `x` will be used as the classifying variable.
- a tessellation on a network (object of class `"lintess"`, see `lintess`). Each point of `x` will be classified according to the tile of the tessellation into which it falls.
- a character string, giving the name of one of the columns of `marks(x)`, if this is a data frame.
- a character string identifying one of the coordinates: the spatial coordinates `"x"`, `"y"` or the segment identifier `"seg"` or the fractional coordinate along the segment, `"tp"`.

The default is to take `z` to be the vector of marks in `x` (or the first column in the data frame of marks of `x`, if it is a data frame). If the marks are numeric, then the range of values of the numerical marks is divided into several intervals, and each interval is associated with a level of a factor. The result is a marked point pattern, on the same linear network, with the same point locations as `x`, but with the numeric mark of each point discretised by replacing it by the factor level. This is a convenient way to transform a marked point pattern which has numeric marks into a multitype point pattern, for example to plot it or analyse it. See the examples.

To select some points from `x`, use the subset operators `[.lpp` or `subset.lpp` instead.

Value

A multitype point pattern on the same linear network, that is, a point pattern object (of class `"lpp"`) with a `marks` vector that is a factor.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`, Rolf Turner `<rolfturner@posteo.net>` and Ege Rubak `<rubak@math.aau.dk>`.

See Also

`cut`, `lpp`, `lintess`, `linfun`, `linim`

Examples

```
X <- runiflpp(20, simplenet)
f <- linfun(function(x,y,seg,tp) { x }, simplenet)
plot(cut(X, f, breaks=4))
plot(cut(X, "x", breaks=4))
plot(cut(X, "seg"))
```

`data.lppm` *Extract Original Data from a Fitted Point Process Model on a Network*

Description

Given a fitted point process model on a linear network, this function extracts the original point pattern dataset to which the model was fitted.

Usage

```
data.lppm(object)
```

Arguments

`object`	fitted point process model on a linear network (an object of class `"lppm"`).

Details

An object of class `"lppm"` represents a point process model that has been fitted to a point pattern dataset on a linear network. It is typically produced by the model-fitting algorithm `lppm`. The object contains complete information about the original data point pattern to which the model was fitted. This function extracts the original data pattern.

Value

A point pattern on a linear network (object of class `"lpp"`).

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`, Rolf Turner `<rolfturner@posteo.net>` and Ege Rubak `<rubak@math.aau.dk>`.

See Also

`lppm`, `data.ppm`

Examples

```
fit <- lppm(spiders ~ x)
X <- data.lppm(fit)
# 'X' is identical to 'spiders'
```

`delaunayNetwork` *Linear Network of Delaunay Triangulation or Dirichlet Tessellation*

Description

Computes the edges of the Delaunay triangulation or Dirichlet tessellation of a point pattern, and returns the result as a linear network object.

Usage

```
delaunayNetwork(X)

dirichletNetwork(X, ...)
```

Arguments

`X`	A point pattern (object of class "ppp").
`...`	Arguments passed to `as.linnet.psp`

Details

For `delaunayNetwork`, points of `X` which are neighbours in the Delaunay triangulation (see `delaunay`) will be joined by a straight line. The result will be returned as a linear network (object of class `"linnet"`).

For `dirichletNetwork`, the Dirichlet tessellation is computed (see `dirichlet`) and the edges of the tiles of the tessellation are extracted. This is converted to a linear network using `as.linnet.psp`.

Value

Linear network (object of class `"linnet"`) or `NULL`.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>

Rolf Turner <`rolfturner@posteo.net`>

and Ege Rubak <`rubak@math.aau.dk`>

See Also

`delaunay`, `dirichlet`, `delaunayDistance`

Examples

```
LE <- delaunayNetwork(cells)
LI <- dirichletNetwork(cells)
```

`deletebranch` *Delete or Extract a Branch of a Tree*

Description

Deletes or extracts a given branch of a tree.

Usage

```
deletebranch(X, ...)

## S3 method for class 'linnet'
deletebranch(X, code, labels, ...)

## S3 method for class 'lpp'
deletebranch(X, code, labels, ...)

extractbranch(X, ...)

## S3 method for class 'linnet'
```

```
extractbranch(X, code, labels, ..., which=NULL)

## S3 method for class 'lpp'
extractbranch(X, code, labels, ..., which=NULL)
```

Arguments

`X`	Linear network (object of class `"linnet"`) or point pattern on a linear network (object of class `"lpp"`).
`code`	Character string. Label of the branch to be deleted or extracted.
`labels`	Vector of character strings. Branch labels for the vertices of the network, usually obtained from `treebranchlabels`.
`...`	Arguments passed to methods.
`which`	Logical vector indicating which vertices of the network should be extracted. Overrides `code` and `labels`.

Details

The linear network `L <- X` or `L <- as.linnet(X)` must be a tree, that is, it has no loops.

The argument `labels` should be a character vector giving tree branch labels for each vertex of the network. It is usually obtained by calling `treebranchlabels`.

The branch designated by the string `code` will be deleted or extracted.

The return value is the result of deleting or extracting this branch from `X` along with any data associated with this branch (such as points or marks).

Value

Another object of the same type as `X` obtained by deleting or extracting the specified branch.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>

See Also

`treebranchlabels`, `branchlabelfun`, `linnet`

Examples

```
  # make a simple tree
  m <- simplenet$m
  m[8,10] <- m[10,8] <- FALSE
  L <- linnet(vertices(simplenet), m)
  plot(L, main="")
  # compute branch labels
  tb <- treebranchlabels(L, 1)
  tbc <- paste0("[", tb, "]")
```

```
text(vertices(L), labels=tbc, cex=2)

# delete branch B
LminusB <- deletebranch(L, "b", tb)
plot(LminusB, add=TRUE, col="green")

# extract branch B
LB <- extractbranch(L, "b", tb)
plot(LB, add=TRUE, col="red")
```

`density.linnet` *Kernel Smoothing of Linear Network*

Description

Compute a kernel smoothed intensity function for the line segments of a linear network.

Usage

```
  ## S3 method for class 'linnet'
density(x, ...)
```

Arguments

`x`	Linear network (object of class `"linnet"`)
`...`	Arguments passed to `density.psp` to control the amount of smoothing and the spatial resolution of the result.

Details

This is the method for the generic function `density` for the class `"linnet"` (linear networks).

The network `x` is first converted to a line segment pattern (object of class `"psp"`). Then the method `density.psp` is applied to the segment pattern.

A kernel estimate of the intensity of the line segment pattern is computed. The result is the convolution of the isotropic Gaussian kernel, of standard deviation `sigma`, with the line segments.

The intensity of a line segment pattern is the (spatially-varying) amount of segment length per unit area, expressed in the same units as the coordinates of `x`. If the units of `x` are in metres, then an intensity value of 3 means that there are 3 metres of segment length per square metre of spatial domain.

See `density.psp` for more details.

Value

A pixel image in two dimensions (object of class `"im"`) or a numeric vector.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>, Rolf Turner <`rolfturner@posteo.net`> and Ege Rubak <`rubak@math.aau.dk`>.

See Also

`density.psp`, `im.object`, `density`.

Examples

```
D <- density(simplenet, 0.1)
plot(D)
plot(simplenet, add=TRUE, col="white")
## compare with average intensity
volume(simplenet)/area(Window(simplenet))
```

`density.lpp` *Kernel Estimate of Intensity on a Linear Network*

Description

Estimates the intensity of a point process on a linear network by applying kernel smoothing to the point pattern data.

Usage

```
## S3 method for class 'lpp'
density(x, sigma=NULL, ...,
        weights=NULL,
        distance=c("path", "euclidean"),
        continuous=TRUE,
        kernel="gaussian")

## S3 method for class 'splitppx'
density(x, sigma=NULL, ...)
```

Arguments

`x`	Point pattern on a linear network (object of class `"lpp"`) to be smoothed.
`sigma`	Smoothing bandwidth (standard deviation of the kernel). A single numerical value in the same units as the spatial coordinates of `x`. Alternatively `sigma` may be a function which selects a bandwidth when applied to `X`, for example, `bw.scott.iso` or `bw.lppl`. There is a sensible default.
`...`	Additional arguments controlling the algorithm and the spatial resolution of the result. These arguments are passed either to `densityQuick.lpp`, `densityHeat.lpp` or `densityEqualSplit` depending on the algorithm chosen.

`weights`	Optional. Numeric vector of weights associated with the points of `x`. Weights may be positive, negative or zero.
`distance`	Character string (partially matched) specifying whether to use a kernel based on paths in the network (`distance="path"`, the default) or a two-dimensional kernel (`distance="euclidean"`).
`kernel`	Character string specifying the smoothing kernel. See `dkernel` for possible options.
`continuous`	Logical value indicating whether to compute the "equal-split continuous" smoother (`continuous=TRUE`, the default) or the "equal-split discontinuous" smoother (`continuous=FALSE`). Applies only when `distance="path"`.

Details

Kernel smoothing is applied to the points of `x` using either a kernel based on path distances in the network, or a two-dimensional kernel. The result is a pixel image on the linear network (class `"linim"`) which can be plotted.

- If `distance="path"` (the default) then the smoothing is performed using a kernel based on path distances in the network, as described in described in Okabe and Sugihara (2012) and McSwiggan et al (2016).
  - If `continuous=TRUE` (the default), smoothing is performed using the "equal-split continuous" rule described in Section 9.2.3 of Okabe and Sugihara (2012). The resulting function is continuous on the linear network.
  - If `continuous=FALSE`, smoothing is performed using the "equal-split discontinuous" rule described in Section 9.2.2 of Okabe and Sugihara (2012). The resulting function is continuous except at the network vertices.
  - In the default case (where `distance="path"` and `continuous=TRUE` and `kernel="gaussian"`, computation is performed rapidly by solving the classical heat equation on the network, as described in McSwiggan et al (2016). The arguments are passed to `densityHeat.lpp` which performs the computation. Computational time is short, but increases quadratically with `sigma`.
  - In all other cases, computation is performed by path-tracing as described in Okabe and Sugihara (2012); the arguments are passed to `densityEqualSplit` which performs the computation. Computation time can be extremely long, and increases exponentially with `sigma`.
- If `distance="euclidean"`, the smoothing is performed using a two-dimensional kernel. The arguments are passed to `densityQuick.lpp` to perform the computation. Computation time is very short. See the help for `densityQuick.lpp` for further details.

There is also a method for split point patterns on a linear network (class `"splitppx"`) which will return a list of pixel images.

The argument `sigma` specifies the smoothing bandwidth. If `sigma` is missing or `NULL`, the default is one-eighth of the length of the shortest side of the bounding box of `x`. If `sigma` is a function in the R language, it is assumed to be a bandwidth selection rule, and it will be applied to `x` to compute the bandwidth value.

Value

A pixel image on the linear network (object of class `"linim"`), or in some cases, a numeric vector of length equal to `npoints(x)`.

Infinite bandwidth

If `sigma=Inf`, the resulting density estimate is constant over all locations, and is equal to the average density of points per unit length. (If the network is not connected, then this rule is applied separately to each connected component of the network).

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`> and Greg McSwiggan.

References

McSwiggan, G., Baddeley, A. and Nair, G. (2016) Kernel density estimation on a linear network. *Scandinavian Journal of Statistics* **44**, 324–345.

Okabe, A. and Sugihara, K. (2012) *Spatial analysis along networks*. Wiley.

See Also

`lpp`, `linim`, `densityQuick.lpp`, `densityHeat.lpp`, `densityVoronoi.lpp`

Examples

```
X <- runiflpp(3, simplenet)
D <- density(X, 0.2, verbose=FALSE)
plot(D, style="w", main="", adjust=2)
Dq <- density(X, 0.2, distance="euclidean")
plot(Dq, style="w", main="", adjust=2)
Dw <- density(X, 0.2, weights=c(1,2,-1), verbose=FALSE)
De <- density(X, 0.2, kernel="epanechnikov", verbose=FALSE)
Ded <- density(X, 0.2, kernel="epanechnikov", continuous=FALSE, verbose=FALSE)
```

`densityEqualSplit` *Equal-Split Algorithm for Kernel Density on a Network*

Description

Computes a kernel density estimate on a linear network using the Okabe-Sugihara equal-split algorithms.

Usage

```
densityEqualSplit(x, sigma = NULL, ...,
                  at = c("pixels", "points"),
                  leaveoneout=TRUE,
                  weights = NULL,
                  kernel = "epanechnikov", continuous = TRUE,
              epsilon = 1e-06, verbose = TRUE, debug = FALSE, savehistory = TRUE)
```

Arguments

`x`	Point pattern on a linear network (object of class `"lpp"`) to be smoothed.
`sigma`	Smoothing bandwidth (standard deviation of the kernel). A numeric value in the same units as the spatial coordinates of `x`. Alternatively `sigma` may be a function which selects a bandwidth when applied to `X`, for example, `bw.scott.iso` or `bw.lppl`. There is a sensible default.
`...`	Arguments passed to `as.mask` determining the resolution of the result.
`at`	String (partially matched) specifying whether to compute the intensity values at a fine grid of locations on the network (`at="pixels"`, the default) or only at the points of `x` (`at="points"`).
`leaveoneout`	Logical value indicating whether to compute a leave-one-out estimator. Applicable only when `at="points"`.
`weights`	Optional. Numeric vector of weights associated with the points of `x`. Weights may be positive, negative or zero.
`kernel`	Character string specifying the smoothing kernel. See `dkernel` for possible options.
`continuous`	Logical value indicating whether to compute the “equal-split continuous” smoother (`continuous=TRUE`, the default) or the “equal-split discontinuous” smoother (`continuous=FALSE`).
`epsilon`	Tolerance value. A tail of the kernel with total mass less than `epsilon` may be deleted.
`verbose`	Logical value indicating whether to print progress reports.
`debug`	Logical value indicating whether to print debugging information.
`savehistory`	Logical value indicating whether to save the entire history of the algorithm, for the purposes of evaluating performance.

Details

Kernel smoothing is applied to the points of `x` using a kernel based on path distances in the network. The result is a pixel image on the linear network (class `"linim"`) which can be plotted.

Smoothing is performed using one of the “equal-split” rules described in Okabe and Sugihara (2012).

- If `continuous=TRUE` (the default), smoothing is performed using the “equal-split continuous” rule described in Section 9.2.3 of Okabe and Sugihara (2012). The resulting function is continuous on the linear network.

- If `continuous=FALSE`, smoothing is performed using the “equal-split discontinuous” rule described in Section 9.2.2 of Okabe and Sugihara (2012). The resulting function is not continuous.

Computation is performed by path-tracing as described in Okabe and Sugihara (2012).

It is advisable to choose a kernel with bounded support such as `kernel="epanechnikov"`. With a Gaussian kernel, computation time can be long, and increases exponentially with `sigma`.

Faster algorithms are available through `density.lpp`.

The argument `sigma` specifies the smoothing bandwidth. If `sigma` is missing or `NULL`, the default is one-eighth of the length of the shortest side of the bounding box of `x`. If `sigma` is a function in the R language, it is assumed to be a bandwidth selection rule, and it will be applied to `x` to compute the bandwidth value.

Value

If `at="pixels"` (the default), a pixel image on the linear network (object of class `"linim"`).

If `at="points"`, a numeric vector with one entry for each point of `x`.

Infinite bandwidth

If `sigma=Inf`, the resulting density estimate is constant over all locations, and is equal to the average density of points per unit length. (If the network is not connected, then this rule is applied separately to each connected component of the network).

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`> and Greg McSwiggan.

References

Okabe, A. and Sugihara, K. (2012) *Spatial analysis along networks*. Wiley.

See Also

`density.lpp`

Examples

```
X <- runiflpp(3, simplenet)
De <- density(X, 0.2, kernel="epanechnikov", verbose=FALSE)
Ded <- density(X, 0.2, kernel="epanechnikov", continuous=FALSE, verbose=FALSE)
```

`densityfun.lpp` *Kernel Estimate of Intensity on a Linear Network as a Spatial Function*

Description

Computes a kernel estimate of the intensity of a point process on a linear network, and returns the intensity estimate as a function of spatial location.

Usage

```
## S3 method for class 'lpp'
densityfun(X, sigma, ..., weights=NULL, nsigma=1, verbose=FALSE)
```

Arguments

`X`	Point pattern on a linear network (object of class `"lpp"`).
`sigma`	Bandwidth of kernel (standard deviation of Gaussian kernel), in the same units of length as `X`.
`...`	Arguments passed to `density.lpp` to control the discretisation.
`weights`	Optional numeric vector of weights associated with the points of `X`.
`nsigma`	Integer. The number of different bandwidths for which a result should be returned. If `nsigma=1` (the default), the result is a function giving kernel estimate with bandwidth `sigma`. If `nsigma` > `1`, the result is a function with an additional argument `k` containing the kernel estimates for the `nsigma+1` equally-spaced time steps from `0` to `sigma^2`.
`verbose`	Logical value indicating whether to print progress reports.

Details

Kernel smoothing is applied to the points of `X` using the diffusion algorithm of McSwiggan et al (2016). The result is a function on the linear network (object of class `"linfun"`) that can be printed, plotted and evaluated at any location.

This is a method for the generic function `densityfun` for the class `"lpp"` of point patterns on a linear network.

Value

Function on a linear network (object of class `"linfun"`).

If `nsigma=1` (the default), the result is a function giving kernel estimate with bandwidth `sigma`.

If `nsigma` > `1`, the result is a function with an additional argument `k`. If `k` is specified, the function returns the kernel estimate for bandwidth `tau` = `sigma` * `sqrt(k/nsigma)`. If `k` is not specified, results are returned for all `k` = `1, 2, ..., nsigma`.

The result also has attributes

- `attr(result, "dt")` giving the time step Δt;

- `attr(result, "dx")` giving the spacing Δx between sample points in the numerical algorithm;
- `attr(result, "sigma")` giving the smoothing bandwidth σ used (or the successive bandwidths used at each sampled time step, if `nsigma` > `1`).

Author(s)

Greg McSwiggan, with tweaks by Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`.

References

McSwiggan, G., Baddeley, A. and Nair, G. (2016) Kernel Density Estimation on a Linear Network. *Scandinavian Journal of Statistics* **44**, 324–345.

See Also

`density.lpp` which returns a pixel image on the linear network.

`methods.linfun` for methods applicable to `"linfun"` objects.

Examples

```
X <- unmark(chicago)
# single bandwidth
g <- densityfun(X, 30)
plot(g)
Y <- X[1:5]
g(Y)
# weighted
gw <- densityfun(X, 30, weights=runif(npoints(X)))
# sequence of bandwidths
g10 <- densityfun(X, 30, nsigma=10)
g10(Y, k=10)
g10(Y)
plot(as.linim(g10, k=5))
```

`densityHeat.lpp` *Kernel Density on a Network using Heat Equation*

Description

Given a point pattern on a linear network, compute a kernel estimate of intensity, by solving the heat equation.

Usage

```
  ## S3 method for class 'lpp'
densityHeat(x, sigma=NULL, ...,
              at=c("pixels", "points"), leaveoneout=TRUE,
              weights = NULL,
              dx = NULL, dt = NULL, iterMax = 1e+06,
              finespacing = TRUE, verbose=FALSE)
```

Arguments

`x`	Point pattern on a linear network (object of class `"lpp"`) to be smoothed.
`sigma`	Smoothing bandwidth (standard deviation of the kernel). A numeric value in the same units as the spatial coordinates of `x`. Alternatively `sigma` may be a function which selects a bandwidth when applied to `X`, for example, `bw.scott.iso` or `bw.lppl`. There is a sensible default.
`...`	Arguments passed to `as.mask` determining the resolution of the result. (Any other arguments are ignored.)
`at`	String specifying whether to compute the intensity values at a fine grid of pixel locations on the network (`at="pixels"`, the default) or only at the data points of `x` (`at="points"`).
`leaveoneout`	Logical value indicating whether to compute a leave-one-out estimator. Applicable only when `at="points"`.
`weights`	Optional. Numeric vector of weights associated with the points of `x`. Weights may be positive, negative or zero.
`dx`	Optional. Spacing of the sampling points along the network. A single number giving a distance value in the same units as `x`.
`dt`	Optional. Time step in the heat equation solver. A single number.
`iterMax`	Maximum number of iterations.
`finespacing`	Logical value specifying whether the discrete approximation is required to be accurate along every segment of the network, no matter how short the segment is. See the section on Discretisation.
`verbose`	Logical value specifying whether to print progress reports.

Details

The function `densityHeat` is generic. This is the method for the class `"lpp"` of points on a linear network.

Kernel smoothing is applied to the points of `x` using a kernel based on path distances in the network. If `at="pixels"` (the default), the result is a pixel image on the linear network (class `"linim"`) which can be plotted. If `at="points"` the result is a numeric vector giving the density estimates at the data points of `x`.

The smoothing operation is equivalent to the “equal-split continuous” rule described in Section 9.2.3 of Okabe and Sugihara (2012). However, the actual computation is performed rapidly, by solving the classical time-dependent heat equation on the network, as described in McSwiggan et al (2016). Computational time is short, but increases quadratically with `sigma`.

If `at="points"` and `leaveoneout=TRUE`, a leave-one-out estimate is computed at each data point (that is, the estimate at each data point `x[i]` is based on all of the points except `x[i]`) using the truncated series approximation of McSwiggan et al (2019).

The argument `sigma` specifies the smoothing bandwidth. If `sigma` is missing or `NULL`, the default is one-eighth of the length of the shortest side of the bounding box of `x`. If `sigma` is a function in the R language, it is assumed to be a bandwidth selection rule, and it will be applied to `x` to compute the bandwidth value.

Value

If `at="pixels"` (the default), a pixel image on the linear network (object of class `"linim"`).

If `at="points"`, a numeric vector with one entry for each point of `x`.

Infinite bandwidth

If `sigma=Inf`, the resulting density estimate is constant over all locations, and is equal to the average density of points per unit length. (If the network is not connected, then this rule is applied separately to each connected component of the network).

Discretisation and Error Messages

The arguments `dx`, `dt` and `iterMax` determine the discretisation of the network, according to a set of rules. The argument `finespacing` determines which rule will be applied.

The arguments `dx`, `dt`, `iterMax` are connected by several constraints; specifying one of these arguments will affect the default values of the other two arguments.

The argument `finespacing` specifies whether a very fine spacing of sample points is required, in order to attain hihg accuracy.

- If `finespacing=TRUE` (the default), then the sample point spacing `dx` must not exceed one-third of the length of the shortest segment of the network. This ensures that the discrete approximation is accurate along every segment, no matter how short the segment is. However, this may not be feasible if it implies a very large number of sample points, or a large number of iterations: in such cases, the code may terminate with an error about illegal values of `dx`, `dt` or `iterMax`.
- If `finespacing=FALSE`, then the sample point spacing `dx` will be about one-half the width of a pixel in the default pixellation of the window of `x`. This is usually a much coarser resolution than the one selected by `finespacing=TRUE`. If it is too coarse, the pixel resolution can be refined using the arguments `dimyx`, `eps` or `xy` passed to `as.mask`. For example, `dimyx=512` would specify a 512 x 512 pixel grid. The default pixel resolution can be changed for the remainder of the R session by `spatstat.options('npixel')`.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>` and Greg McSwiggan.

References

McSwiggan, G., Baddeley, A. and Nair, G. (2016) Kernel density estimation on a linear network. *Scandinavian Journal of Statistics* **44**, 324–345.

McSwiggan, G., Baddeley, A. and Nair, G. (2019) Estimation of relative risk for events on a linear network. *Statistics and Computing* **30**, 469–484.

Okabe, A. and Sugihara, K. (2012) *Spatial analysis along networks*. Wiley.

See Also

`density.lpp`

Examples

```
X <- runiflpp(3, simplenet)
D <- densityHeat(X, 0.2)
plot(D, style="w", main="", adjust=2)
densityHeat.lpp(X, 0.2, at="points")
Dw <- densityHeat(X, 0.2, weights=c(1,2,-1))
```

`densityQuick.lpp` *Kernel Estimation of Intensity on a Network using a 2D Kernel*

Description

Estimates the intensity of a point process on a linear network using a two-dimensional smoothing kernel.

Usage

```
densityQuick.lpp(x, sigma=NULL, ...,
         kernel="gaussian",
         at = c("pixels", "points"),
         what = c("estimate", "se", "var"),
         leaveoneout = TRUE,
         diggle = FALSE,
         edge2D = FALSE,
         weights = NULL,
         positive = FALSE)
```

Arguments

`x`	Point pattern on a linear network (object of class `"lpp"`).
`sigma`	Smoothing bandwidth. A single numeric value, in the same units as the coordinates of `x`. Alternatively `sigma` may be a function which selects a bandwidth when applied to `x`, for example, `bw.scott.iso` or `bw.lppl`. There is a sensible default.

`...`	Additional arguments passed to `as.mask` to determine the pixel resolution, or arguments passed to `sigma` if it is a function.
`kernel`	String (partially matched) specifying the smoothing kernel. Current options are `"gaussian"`, `"epanechnikov"`, `"quartic"` or `"disc"`.
`at`	String (partially matched) specifying whether to compute the intensity values at a fine grid of locations on the network (`at="pixels"`, the default) or only at the points of `x` (`at="points"`).
`what`	String (partially matched) specifying whether to calculate the intensity estimate, or its estimated standard error, or its estimated variance.
`leaveoneout`	Logical value indicating whether to compute a leave-one-out estimator. Applicable only when `at="points"`.
`diggle`	Logical value specifying whether to use the 'Diggle' correction.
`edge2D`	Logical value specifying whether to apply the usual two-dimensional edge correction procedure to the numerator and denominator of the estimate.
`weights`	Optional weights to be attached to the points. A numeric vector, an `expression`, or a pixel image.
`positive`	Logical value indicating whether to force the resulting values to be positive. Default is `FALSE` for the sake of speed.

Details

Kernel smoothing is applied to the points of `x` using a two-dimensional Gaussian kernel, as described in Rakshit et al (2019). The result is a pixel image on the linear network (class `"linim"`) which can be plotted.

Other techniques for kernel smoothing on a network are implemented in `density.lpp`. The main advantages of using a two-dimensional kernel are very fast computation and insensitivity to changes in the network geometry. The main disadvantage is that it ignores the connectivity of the network. See Rakshit et al (2019) for further explanation.

The argument `sigma` specifies the smoothing bandwidth. If `sigma` is missing or `NULL`, the default is one-eighth of the length of the shortest side of the bounding box of `x`. If `sigma` is a function in the R language, it is assumed to be a bandwidth selection rule, and it will be applied to `x` to compute the bandwidth value.

Value

If `at="pixels"` (the default), a pixel image on the linear network (object of class `"linim"`).

If `at="points"`, a numeric vector with one entry for each point of `x`.

Infinite bandwidth

If `sigma=Inf`, the resulting density estimate is constant over all locations, and is equal to the average density of points per unit length. (If the network is not connected, then this rule is applied separately to each connected component of the network).

Author(s)

Adrian Baddeley, Suman Rakshit and Tilman Davies

References

Rakshit, S., Davies, T., Moradi, M., McSwiggan, G., Nair, G., Mateu, J. and Baddeley, A. (2019) Fast kernel smoothing of point patterns on a large network using 2D convolution. *International Statistical Review* **87** (3) 531–556. DOI: 10.1111/insr.12327.

See Also

`density.lpp`, the main function for density estimation on a network.

`bw.scott`, `bw.scott.iso`, `bw.lppl`for bandwidth selection.

Examples

```
X <- unmark(chicago)
plot(densityQuick.lpp(X, 500))
plot(densityQuick.lpp(X, 500, diggle=TRUE))
plot(densityQuick.lpp(X, bw.scott.iso))
plot(densityQuick.lpp(X, 500, what="se"))
```

`densityVoronoi.lpp` *Intensity Estimate of Point Pattern on Linear Network Using Voronoi-Dirichlet Tessellation*

Description

Computes an adaptive estimate of the intensity function of a point pattern on a linear network, using the Dirichlet-Voronoi tessellation on the network.

Usage

```
## S3 method for class 'lpp'
densityVoronoi(X, f = 1, ..., nrep = 1, verbose = TRUE)
```

Arguments

`X`	Point pattern on a linear network (object of class `"lpp"`).
`f`	Fraction (between 0 and 1 inclusive) of the data points that will be used to build a tessellation for the intensity estimate.
`...`	Arguments passed to `linim` determining the pixel resolution of the result.
`nrep`	Number of independent repetitions of the randomised procedure.
`verbose`	Logical value indicating whether to print progress reports.

Details

This function is an alternative to `density.lpp`. It computes an estimate of the intensity function of a point pattern dataset on a linear network. The result is a pixel image on the network, giving the estimated intensity.

This function is a method for the generic `densityVoronoi` for the class `"lpp"` of point patterns on a linear network.

If `f=1` (the default), the Voronoi estimate (Barr and Schoenberg, 2010) is computed: the point pattern `X` is used to construct a Voronoi/Dirichlet tessellation on the network (see `lineardirichlet`); the lengths of the Dirichlet tiles are computed; the estimated intensity in each tile is the reciprocal of the tile length. The result is a pixel image of intensity estimates which are constant on each tile of the tessellation.

If `f=0`, the intensity estimate at every location is equal to the average intensity (number of points divided by network length). The result is a pixel image of intensity estimates which are constant.

If `f` is strictly between 0 and 1, the smoothed Voronoi estimate (Moradi et al, 2019) is computed. The dataset `X` is randomly thinned by deleting or retaining each point independently, with probability `f` of retaining a point. The thinned pattern is used to construct a Dirichlet tessellation and form the Voronoi estimate, which is then adjusted by a factor `1/f`. This procedure is repeated `nrep` times and the results are averaged to obtain the smoothed Voronoi estimate.

The value `f` can be chosen automatically by bandwidth selection using `bw.voronoi`.

Value

Pixel image on a linear network (object of class `"linim"`).

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`, Rolf Turner `<rolfturner@posteo.net>` and Ege Rubak `<rubak@math.aau.dk>` and Mehdi Moradi `<m2.moradi@yahoo.com>`.

References

Moradi, M., Cronie, 0., Rubak, E., Lachieze-Rey, R., Mateu, J. and Baddeley, A. (2019) Resample-smoothing of Voronoi intensity estimators. *Statistics and Computing* **29** (5) 995–1010.

See Also

`densityVoronoi` is the generic, with a method for class `"ppp"`.

`lineardirichlet` computes the Dirichlet-Voronoi tessellation on a network.

`bw.voronoi` performs bandwidth selection of the fraction `f`.

See also `density.lpp`.

Examples

```
nr <- if(interactive()) 100 else 3
plot(densityVoronoi(spiders, 0.1, nrep=nr))
```

`diameter.linnet` *Diameter and Bounding Radius of a Linear Network*

Description

Compute the diameter or bounding radius of a linear network measured using the shortest path distance.

Usage

```
## S3 method for class 'linnet'
diameter(x)

## S3 method for class 'linnet'
boundingradius(x, ...)
```

Arguments

`x`	Linear network (object of class `"linnet"`).
`...`	Ignored.

Details

The diameter of a linear network (in the shortest path distance) is the maximum value of the shortest-path distance between any two points u and v on the network.

The bounding radius of a linear network (in the shortest path distance) is the minimum value, over all points u on the network, of the maximum shortest-path distance from u to another point v on the network.

The functions `boundingradius` and `diameter` are generic; the functions `boundingradius.linnet` and `diameter.linnet` are the methods for objects of class `linnet`.

Value

A single numeric value.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>

See Also

`boundingradius`, `diameter`, `linnet`

Examples

```
diameter(simplenet)
boundingradius(simplenet)
```

`distfun.lpp` *Distance Map on Linear Network*

Description

Compute the distance function of a point pattern on a linear network.

Usage

```
  ## S3 method for class 'lpp'
distfun(X, ..., k=1)
```

Arguments

`X`	A point pattern on a linear network (object of class `"lpp"`).
`k`	An integer. The distance to the kth nearest point will be computed.
`...`	Extra arguments are ignored.

Details

On a linear network L, the "geodesic distance function" of a set of points A in L is the mathematical function f such that, for any location s on L, the function value `f(s)` is the shortest-path distance from s to A.

The command `distfun.lpp` is a method for the generic command `distfun` for the class `"lpp"` of point patterns on a linear network.

If `X` is a point pattern on a linear network, `f <- distfun(X)` returns a *function* in the R language that represents the distance function of `X`. Evaluating the function `f` in the form `v <- f(x,y)`, where `x` and `y` are any numeric vectors of equal length containing coordinates of spatial locations, yields the values of the distance function at these locations. More efficiently `f` can be called in the form `v <- f(x, y, seg, tp)` where `seg` and `tp` are the local coordinates on the network. It can also be called as `v <- f(x)` where `x` is a point pattern on the same linear network.

The function `f` obtained from `f <- distfun(X)` also belongs to the class `"linfun"`. It can be printed and plotted immediately as shown in the Examples. It can be converted to a pixel image using `as.linim`.

Value

A `function` with arguments `x,y` and optional arguments `seg,tp`. It also belongs to the class `"linfun"` which has methods for `plot`, `print` etc.

Distance values

The values returned by the distance function `f <- distfun(X)` are distances, expressed as multiples of the unit of length of the spatial coordinates in `X`. The unit of length is given by `unitname(X)`.

Note that, if the unit of length in `X` is a composite expression such as '2 microns', then the values of `f` are expressed as multiples of 2 microns, rather than being expressed in microns.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>, Rolf Turner <`rolfturner@posteo.net`> and Ege Rubak <`rubak@math.aau.dk`>.

See Also

`linfun`, `methods.linfun`.

To identify *which* point is the nearest neighbour, see `nnfun.lpp`.

Examples

```
X <- runiflpp(3, simplenet)
f <- distfun(X)
f
plot(f)

# using a distfun as a covariate in a point process model:
Y <- runiflpp(4, simplenet)
fit <- lppm(Y ~D, covariates=list(D=f))

f(Y)
```

`distmap.lpp` *Distance Map of Point Pattern on Linear Network*

Description

Computes the distance from each pixel to the nearest point in the given point pattern on a linear network.

Usage

```
  ## S3 method for class 'lpp'
distmap(X, ..., k=1)
```

Arguments

`X`	A point pattern on a linear network (object of class `"lpp"`).
`k`	Integer. The distance to the `k`-th nearest data point will be computed.
`...`	Arguments passed to `as.linim.linfun` to control pixel resolution.

Details

This is a method for the generic function `distmap`. It computes the distance map of the point pattern `X` as a pixel image on the network.

At a pixel u, the greyscale value equals the distance from u to the nearest point of the pattern `X` (or the k-th nearest point of `X`).

Value

A pixel image on the network (object of class `"linim"`) whose greyscale values are the values of the distance map.

Distance values

The pixel values in the image `distmap(X)` are distances, expressed as multiples of the unit of length of the spatial coordinates in `X`. The unit of length is given by `unitname(X)`.

Note that, if the unit of length in `X` is a composite expression such as ‘2 microns’, then the values in `distmap(X)` are expressed as multiples of 2 microns, rather than being expressed in microns.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>, Rolf Turner <`rolfturner@posteo.net`> and Ege Rubak <`rubak@math.aau.dk`>.

See Also

Generic function `distmap` and methods.

Examples

```
plot(distmap(spiders))
```

`divide.linnet` *Divide Linear Network at Cut Points*

Description

Make a tessellation of a linear network by dividing it into pieces demarcated by the points of a point pattern.

Usage

```
divide.linnet(X)
```

Arguments

`X`	Point pattern on a linear network (object of class `"lpp"`).

Details

The points `X` are interpreted as dividing the linear network `L=as.linnet(X)` into separate pieces.

Two locations on `L` belong to the same piece if and only if they can be joined by a path in `L` that does not cross any of the points of `X`.

The result is a tessellation of the network (object of class `"lintess"`) representing the division of `L` into pieces.

Value

A tessellation on a linear network (object of class `"lintess"`).

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`, Rolf Turner `<rolfturner@posteo.net>` and Ege Rubak `<rubak@math.aau.dk>` and Greg McSwiggan.

See Also

`linnet`, `lintess`.

Examples

```
X <- runiflpp(5, simplenet)
plot(divide.linnet(X))
plot(X, add=TRUE, pch=16, show.network=FALSE)
```

`domain.lpp` *Extract the Linear Network on which Spatial Data are Defined*

Description

Given a spatial object representing data on a linear network, extract the network.

Usage

```
## S3 method for class 'lpp'
domain(X, ...)

## S3 method for class 'lppm'
domain(X, ...)

## S3 method for class 'linfun'
domain(X, ...)

## S3 method for class 'lintess'
domain(X, ...)
```

Arguments

`X`	A spatial object representing data on a linear network. An object of class `"lpp"`, `"lppm"`, `"linfun"` or `"lintess"`.
`...`	Extra arguments. They are ignored by all the methods listed here.

Details

The function `domain` is generic, with methods for many classes.

For a spatial object `X` `domain(X)` extracts the spatial domain in which `X` is defined.

For a two-dimensional object `X`, typically `domain(X)` is the same as `Window(X)`.

The exception is that, if `X` is a point pattern on a linear network (class `"lpp"`) or a point process model on a linear network (class `"lppm"`), then `domain(X)` is the linear network on which the points lie, while `Window(X)` is the two-dimensional window containing the linear network.

Value

A linear network (object of class `"linnet"`).

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>, Rolf Turner <`rolfturner@posteo.net`> and Ege Rubak <`rubak@math.aau.dk`>.

See Also

`domain`, `domain.rmhmodel`, `domain.ppm`.

`Window`, `Frame`

Examples

```
domain(chicago)
```

`envelope.lpp` *Envelope for Point Patterns on Linear Network*

Description

Enables envelopes to be computed for point patterns on a linear network.

Usage

```
  ## S3 method for class 'lpp'
envelope(Y, fun=linearK, nsim=99, nrank=1, ...,
  funargs=list(), funYargs=funargs,
  simulate=NULL, fix.n=FALSE, fix.marks=FALSE, verbose=TRUE,
  transform=NULL,global=FALSE,ginterval=NULL,use.theory=NULL,
  alternative=c("two.sided", "less", "greater"),
  scale=NULL, clamp=FALSE,
  savefuns=FALSE, savepatterns=FALSE,
  nsim2=nsim, VARIANCE=FALSE, nSD=2, Yname=NULL,
  maxnerr=nsim, rejectNA=FALSE, silent=FALSE,
  do.pwrong=FALSE, envir.simul=NULL)
```

```
  ## S3 method for class 'lppm'
envelope(Y, fun=linearK, nsim=99, nrank=1, ...,
  funargs=list(), funYargs=funargs,
  simulate=NULL, fix.n=FALSE, fix.marks=FALSE, verbose=TRUE,
  transform=NULL,global=FALSE,ginterval=NULL,use.theory=NULL,
  alternative=c("two.sided", "less", "greater"),
  scale=NULL, clamp=FALSE,
  savefuns=FALSE, savepatterns=FALSE,
  nsim2=nsim, VARIANCE=FALSE, nSD=2, Yname=NULL,
  maxnerr=nsim, rejectNA=FALSE, silent=FALSE,
  do.pwrong=FALSE, envir.simul=NULL)
```

Arguments

`Y`	A point pattern on a linear network (object of class `"lpp"`) or a fitted point process model on a linear network (object of class `"lppm"`).
`fun`	Function that is to be computed for each simulated pattern.
`nsim`	Number of simulations to perform.
`nrank`	Integer. Rank of the envelope value amongst the `nsim` simulated values. A rank of 1 means that the minimum and maximum simulated values will be used.
`...`	Extra arguments passed to `fun`.
`funargs`	A list, containing extra arguments to be passed to `fun`.
`funYargs`	Optional. A list, containing extra arguments to be passed to `fun` when applied to the original data `Y` only.
`simulate`	Optional. Specifies how to generate the simulated point patterns. If `simulate` is an expression in the R language, then this expression will be evaluated `nsim` times, to obtain `nsim` point patterns which are taken as the simulated patterns from which the envelopes are computed. If `simulate` is a function, then this function will be repeatedly applied to the data pattern `Y` to obtain `nsim` simulated patterns. If `simulate` is a list of point patterns, then the entries in this list will be treated as the simulated patterns from which the envelopes are computed. Alternatively `simulate` may be an object produced by the `envelope` command: see Details.
`fix.n`	Logical. If `TRUE`, simulated patterns will have the same number of points as the original data pattern.
`fix.marks`	Logical. If `TRUE`, simulated patterns will have the same number of points *and* the same marks as the original data pattern. In a multitype point pattern this means that the simulated patterns will have the same number of points *of each type* as the original data.
`verbose`	Logical flag indicating whether to print progress reports during the simulations.
`transform`	Optional. A transformation to be applied to the function values, before the envelopes are computed. An expression object (see Details).
`global`	Logical flag indicating whether envelopes should be pointwise (`global=FALSE`) or simultaneous (`global=TRUE`).

`ginterval`	Optional. A vector of length 2 specifying the interval of r values for the simultaneous critical envelopes. Only relevant if `global=TRUE`.
`use.theory`	Logical value indicating whether to use the theoretical value, computed by `fun`, as the reference value for simultaneous envelopes. Applicable only when `global=TRUE`.
`alternative`	Character string determining whether the envelope corresponds to a two-sided test (`side="two.sided"`, the default) or a one-sided test with a lower critical boundary (`side="less"`) or a one-sided test with an upper critical boundary (`side="greater"`).
`scale`	Optional. Scaling function for global envelopes. A function in the R language which determines the relative scale of deviations, as a function of distance r, when computing the global envelopes. Applicable only when `global=TRUE`. Summary function values for distance `r` will be *divided* by `scale(r)` before the maximum deviation is computed. The resulting global envelopes will have width proportional to `scale(r)`.
`clamp`	Logical value indicating how to compute envelopes when `alternative="less"` or `alternative="greater"`. Deviations of the observed summary function from the theoretical summary function are initially evaluated as signed real numbers, with large positive values indicating consistency with the alternative hypothesis. If `clamp=FALSE` (the default), these values are not changed. If `clamp=TRUE`, any negative values are replaced by zero.
`savefuns`	Logical flag indicating whether to save all the simulated function values.
`savepatterns`	Logical flag indicating whether to save all the simulated point patterns.
`nsim2`	Number of extra simulated point patterns to be generated if it is necessary to use simulation to estimate the theoretical mean of the summary function. Only relevant when `global=TRUE` and the simulations are not based on CSR.
`VARIANCE`	Logical. If `TRUE`, critical envelopes will be calculated as sample mean plus or minus `nSD` times sample standard deviation.
`nSD`	Number of estimated standard deviations used to determine the critical envelopes, if `VARIANCE=TRUE`.
`Yname`	Character string that should be used as the name of the data point pattern `Y` when printing or plotting the results.
`maxnerr`	Maximum number of rejected patterns. If `fun` yields a fatal error when applied to a simulated point pattern (for example, because the pattern is empty and `fun` requires at least one point), the pattern will be rejected and a new random point pattern will be generated. If this happens more than `maxnerr` times, the algorithm will give up.
`rejectNA`	Logical value specifying whether to reject a simulated pattern if the resulting values of `fun` are all equal to `NA`, `NaN` or infinite. If `FALSE` (the default), then simulated patterns are rejected only when `fun` gives a fatal error.
`silent`	Logical value specifying whether to print a report each time a simulated pattern is rejected.
`do.pwrong`	Logical. If `TRUE`, the algorithm will also estimate the true significance level of the “wrong” test (the test that declares the summary function for the data to be significant if it lies outside the *pointwise* critical boundary at any point). This estimate is printed when the result is printed.

`envir.simul`	Environment in which to evaluate the expression `simulate`, if not the current environment.

Details

This is a method for the generic function `envelope` applicable to point patterns on a linear network.

The argument `Y` can be either a point pattern on a linear network, or a fitted point process model on a linear network. The function `fun` will be evaluated for the data and also for `nsim` simulated point patterns on the same linear network. The upper and lower envelopes of these evaluated functions will be computed as described in `envelope`.

The type of simulation is determined as follows.

- if `Y` is a point pattern (object of class `"lpp"`) and `simulate` is missing or `NULL`, then random point patterns will be generated according to a Poisson point process on the linear network on which `Y` is defined, with intensity estimated from `Y`.
- if `Y` is a fitted point process model (object of class `"lppm"`) and `simulate` is missing or `NULL`, then random point patterns will be generated by simulating from the fitted model.
- If `simulate` is present, it specifies the type of simulation as explained below.
- If `simulate` is an expression (typically including a call to a random generator), then the expression will be repeatedly evaluated, and should yield random point patterns on the same linear network as `Y`.
- If `simulate` is a function (typically including a call to a random generator), then the function will be repeatedly applied to the original point pattern `Y`, and should yield random point patterns on the same linear network as `Y`.
- If `simulate` is a list of point patterns, then these will be taken as the simulated point patterns. They should be on the same linear network as `Y`.

The function `fun` should accept as its first argument a point pattern on a linear network (object of class `"lpp"`) and should have another argument called `r` or a `...` argument.

Value

Function value table (object of class `"fv"`) with additional information, as described in `envelope`.

Author(s)

Ang Qi Wei <`aqw07398@hotmail.com`> and Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>

References

Ang, Q.W. (2010) *Statistical methodology for events on a network.* Master's thesis, School of Mathematics and Statistics, University of Western Australia.

Ang, Q.W., Baddeley, A. and Nair, G. (2012) Geometrically corrected second-order analysis of events on a linear network, with applications to ecology and criminology. *Scandinavian Journal of Statistics* **39**, 591–617.

Okabe, A. and Yamada, I. (2001) The K-function method on a network and its computational implementation. *Geographical Analysis* **33**, 271-290.

See Also

envelope, linearK

Examples

```
if(interactive()) {
  ns <- 39
  np <- 40
} else { ns <- np <- 3 }
X <- runiflpp(np, simplenet)

# uniform Poisson: random numbers of points
envelope(X, nsim=ns)

# uniform Poisson: conditional on observed number of points
envelope(X, fix.n=TRUE, nsim=ns)

# nonuniform Poisson
fit <- lppm(X ~x)
envelope(fit, nsim=ns)

#multitype
marks(X) <- sample(letters[1:2], np, replace=TRUE)
envelope(X, nsim=ns)
```

`eval.linim` *Evaluate Expression Involving Pixel Images on Linear Network*

Description

Evaluates any expression involving one or more pixel images on a linear network, and returns a pixel image on the same linear network.

Usage

```
eval.linim(expr, envir, harmonize=TRUE, warn=TRUE)
```

Arguments

`expr`	An expression in the R language, involving the names of objects of class `"linim"`.
`envir`	Optional. The environment in which to evaluate the expression.
`harmonize`	Logical. Whether to resolve inconsistencies between the pixel grids.
`warn`	Logical. Whether to issue a warning if the pixel grids were inconsistent.

Details

This function a wrapper to make it easier to perform pixel-by-pixel calculations. It is one of several functions whose names begin with `eval` which work on objects of different types. This particular function is designed to work with objects of class `"linim"` which represent pixel images on a linear network.

Suppose `X` is a pixel image on a linear network (object of class `"linim"`. Then `eval.linim(X+3)` will add 3 to the value of every pixel in `X`, and return the resulting pixel image on the same linear network.

Suppose `X` and `Y` are two pixel images on the same linear network, with compatible pixel dimensions. Then `eval.linim(X + Y)` will add the corresponding pixel values in `X` and `Y`, and return the resulting pixel image on the same linear network.

In general, `expr` can be any expression in the R language involving (a) the *names* of pixel images, (b) scalar constants, and (c) functions which are vectorised. See the Examples.

First `eval.linim` determines which of the *variable names* in the expression `expr` refer to pixel images. Each such name is replaced by a matrix containing the pixel values. The expression is then evaluated. The result should be a matrix; it is taken as the matrix of pixel values.

The expression `expr` must be vectorised. There must be at least one linear pixel image in the expression.

All images must have compatible dimensions. If `harmonize=FALSE`, images that are incompatible will cause an error. If `harmonize=TRUE`, images that have incompatible dimensions will be resampled so that they are compatible; if `warn=TRUE`, a warning will be issued.

Value

An image object of class `"linim"`.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>` and Rolf Turner `<rolfturner@posteo.net>`

See Also

`eval.im`, `linim`

Examples

```
M <- psp2mask(as.psp(simplenet))
Z <- as.im(function(x,y) {x-y}, W=M)
X <- linim(simplenet, Z)
X

Y <- linfun(function(x,y,seg,tp){y^2+x}, simplenet)
Y <- as.linim(Y)

eval.linim(X + 3)
eval.linim(X - Y)
eval.linim(abs(X - Y))
Z <- eval.linim(sin(X * pi) + Y)
```

`Extract.linim` *Extract Subset of Pixel Image on Linear Network*

Description

Extract a subset of a pixel image on a linear network.

Usage

```
  ## S3 method for class 'linim'
x[i, ..., drop=TRUE]
```

Arguments

`x`	A pixel image on a linear network (object of class `"linim"`).
`i`	Spatial window defining the subregion. Either a spatial window (an object of class `"owin"`), or a logical-valued pixel image, or any type of index that applies to a matrix, or a point pattern (an object of class `"lpp"` or `"ppp"`), or something that can be converted to a point pattern by `as.lpp` (using the network on which `x` is defined).
`...`	Additional arguments passed to `[.im`.
`drop`	Logical value indicating whether `NA` values should be omitted from the result.

Details

This function is a method for the subset operator `"["` for pixel images on linear networks (objects of class `"linim"`).

The pixel image `x` will be restricted to the domain specified by `i`.

Pixels outside the domain of `x` are assigned the value `NA`; if `drop=TRUE` (the default) such `NA` values are deleted from the result; if `drop=FALSE`, then `NA` values are retained.

If `i` is a window (or a logical-valued pixel image) then `x[i]` is another pixel image of class `"linim"`, representing the restriction of `x` to the spatial domain specified by `i`.

If `i` is a point pattern, then `x[i]` is the vector of pixel values of `x` at the locations specified by `i`.

Value

Another pixel image on a linear network (object of class `"linim"`) or a vector of pixel values.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>

See Also

`thinNetwork` to extract the data lying on selected edges of the network.

`linim` to make a pixel image on a network.

Examples

```
M <- psp2mask(as.psp(simplenet))
Z <- as.im(function(x,y){x}, W=M)
Y <- linim(simplenet, Z)
X <- runiflpp(4, simplenet)
Y[X]
Y[square(c(0.3, 0.6))]
```

`Extract.linnet` *Extract Subset of Linear Network*

Description

Extract a subset of a linear network.

Usage

```
  ## S3 method for class 'linnet'
x[i, ..., snip=TRUE]
```

Arguments

`x`	A linear network (object of class `"linnet"`).
`i`	Spatial window defining the subregion. An object of class `"owin"`.
`snip`	Logical. If `TRUE` (the default), segments of `x` which cross the boundary of `i` will be cut by the boundary. If `FALSE`, these segments will be deleted.
`...`	Ignored.

Details

This function computes the intersection between the linear network `x` and the domain specified by `i`.

This function is a method for the subset operator `"["` for linear networks (objects of class `"linnet"`). It is provided mainly for completeness.

The index `i` should be a window.

The argument `snip` specifies what to do with segments of `x` which cross the boundary of `i`. If `snip=FALSE`, such segments are simply deleted. If `snip=TRUE` (the default), such segments are cut into pieces by the boundary of `i`, and those pieces which lie inside the window `i` are included in the resulting network.

Value

Another linear network (object of class `"linnet"`).

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>, Rolf Turner <`rolfturner@posteo.net`>, Ege Rubak <`rubak@math.aau.dk`> and Suman Rakshit.

Examples

```
p <- par(mfrow=c(1,2), mar=0.2+c(0,0,1,0))
B <- owin(c(0.1,0.7),c(0.19,0.6))

plot(simplenet, main="x[w, snip=TRUE]")
plot(simplenet[B], add=TRUE, col="green", lwd=3)
plot(B, add=TRUE, border="red", lty=3)

plot(simplenet, main="x[w, snip=FALSE]")
plot(simplenet[B, snip=FALSE], add=TRUE, col="green", lwd=3)
plot(B, add=TRUE, border="red", lty=3)

par(p)
```

`Extract.lpp` *Extract Subset of Point Pattern on Linear Network*

Description

Extract a subset of a point pattern on a linear network.

Usage

```
  ## S3 method for class 'lpp'
x[i, j, drop=FALSE, ..., snip=TRUE]
```

Arguments

`x`	A point pattern on a linear network (object of class `"lpp"`).
`i`	Subset index. A valid subset index in the usual R sense, indicating which points should be retained.
`j`	Spatial window (object of class `"owin"`) delineating the region that should be retained.
`drop`	Logical value indicating whether to remove unused levels of the marks, if the marks are a factor.
`snip`	Logical. If `TRUE` (the default), segments of the network which cross the boundary of the window `j` will be cut by the boundary. If `FALSE`, these segments will be deleted.
`...`	Ignored.

Details

This function extracts a designated subset of a point pattern on a linear network.

The function `[.lpp` is a method for `[` for the class `"lpp"`. It extracts a designated subset of a point pattern. The argument `i` should be a subset index in the usual R sense: either a numeric vector of positive indices (identifying the points to be retained), a numeric vector of negative indices (identifying the points to be deleted) or a logical vector of length equal to the number of points in the point pattern `x`. In the latter case, the points `(x$x[i], x$y[i])` for which `subset[i]=TRUE` will be retained, and the others will be deleted.

The argument `j`, if present, should be a spatial window. The pattern inside the region will be retained. *Line segments that cross the boundary of the window are deleted* in the current implementation.

The argument `drop` determines whether to remove unused levels of a factor, if the point pattern is multitype (i.e. the marks are a factor) or if the marks are a data frame or hyperframe in which some of the columns are factors.

The argument `snip` specifies what to do with segments of the network which cross the boundary of the window `j`. If `snip=FALSE`, such segments are simply deleted. If `snip=TRUE` (the default), such segments are cut into pieces by the boundary of `j`, and those pieces which lie inside the window `ji` are included in the resulting network.

Use `unmark` to remove all the marks in a marked point pattern, and `subset.lpp` to remove only some columns of marks.

Value

A point pattern on a linear network (of class `"lpp"`).

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>` and Rolf Turner `<rolfturner@posteo.net>`

See Also

`lpp`, `subset.lpp`

Examples

```
# Chicago crimes data - remove cases of assault
chicago[marks(chicago) != "assault"]
# equivalent to subset(chicago, select=-assault)

# spatial window subset
B <- owin(c(350, 700), c(600, 1000))
plot(chicago)
plot(B, add=TRUE, lty=2, border="red", lwd=3)
op <- par(mfrow=c(1,2), mar=0.6+c(0,0,1,0))
plot(B, main="chicago[B, snip=FALSE]", lty=3, border="red")
plot(chicago[, B, snip=FALSE], add=TRUE)
plot(B, main="chicago[B, snip=TRUE]", lty=3, border="red")
plot(chicago[, B, snip=TRUE], add=TRUE)
par(op)
```

`fitted.lppm` *Fitted Intensity for Point Process on Linear Network*

Description

Given a point process model fitted to a point pattern on a linear network, compute the fitted intensity of the model at the points of the pattern, or at the points of the quadrature scheme used to fit the model.

Usage

```
## S3 method for class 'lppm'
fitted(object, ...,
                      dataonly = FALSE, new.coef = NULL,
      leaveoneout = FALSE)
```

Arguments

`object`	Fitted point process model on a linear network (object of class `"lppm"`).
`...`	Ignored.
`dataonly`	Logical value indicating whether to computed fitted intensities at the points of the original point pattern dataset (`dataonly=TRUE`) or at all the quadrature points of the quadrature scheme used to fit the model (`dataonly=FALSE`, the default).
`new.coef`	Numeric vector of parameter values to replace the fitted model parameters `coef(object)`.
`leaveoneout`	Logical. If `TRUE` the fitted value at each data point will be computed using a leave-one-out method. See Details.

Details

This is a method for the generic function `fitted` for the class `"lppm"` of fitted point process models on a linear network.

The locations u at which the fitted conditional intensity/trend is evaluated, are the points of the quadrature scheme used to fit the model in `ppm`. They include the data points (the points of the original point pattern dataset `x`) and other "dummy" points in the window of observation.

If `leaveoneout=TRUE`, fitted values will be computed for the data points only, using a 'leave-one-out' rule: the fitted value at `X[i]` is effectively computed by deleting this point from the data and re-fitting the model to the reduced pattern `X[-i]`, then predicting the value at `X[i]`. (Instead of literally performing this calculation, we apply a Taylor approximation using the influence function computed in `dfbetas.ppm`.

Value

A vector containing the values of the fitted spatial trend.

Entries in this vector correspond to the quadrature points (data or dummy points) used to fit the model. The quadrature points can be extracted from `object` by `union.quad(quad.ppm(object))`.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>, Rolf Turner <`rolfturner@posteo.net`> and Ege Rubak <`rubak@math.aau.dk`>.

See Also

`lppm`, `predict.lppm`

Examples

```
fit <- lppm(spiders~x+y)
a <- fitted(fit)
b <- fitted(fit, dataonly=TRUE)
```

`heatkernelapprox` *Approximation to Heat Kernel on Linear Network at Source Point*

Description

Computes an approximation to the value of the heat kernel on a network evaluated at its source location.

Usage

```
heatkernelapprox(X, sigma, nmax = 20, floored=TRUE)
```

Arguments

`X`	Point pattern on a linear network (object of class `"lpp"`).
`sigma`	Numeric. Bandwidth for kernel.
`nmax`	Number of terms to be used in the sum.
`floored`	Logical. If `TRUE`, all values are constrained to be greater than or equal to $1/L$ where L is the total length of the network. This the exact value of the heat kernel when the bandwidth is infinite.

Details

For each point `X[i]` in the pattern `X`, this algorithm computes an approximation to the value of the heat kernel with source point `X[i]` evaluated at the same location.

The heat kernel $\kappa(u,v)$ for a source location u evaluated at location v can be expressed as an infinite sum of contributions from all possible paths from u to v. This algorithm applies to the special case $u = v$ where the source point and the query point are the same.

The algorithm computes an approximation to $\kappa(u,u)$ by taking only the contributions from paths which (a) remain in the line segment containing the point u and (b) visit a vertex at most `nmax` times.

Value

Numeric vector with one entry for each point in `X`.

Author(s)

Greg McSwiggan and Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`.

See Also

`hotrod`

Examples

```
X <- runiflpp(3,simplenet)
heatkernelapprox(X, 0.5)
```

`identify.lpp` *Identify Points in a Point Pattern on a Linear Network*

Description

If a point pattern on a network is plotted in the graphics window, this function will find the point of the pattern which is nearest to the mouse position, and print its mark value (or its serial number if there is no mark).

Usage

```
 ## S3 method for class 'lpp'
identify(x, ...)
```

Arguments

`x`	A point pattern on a linear network (object of class `"lpp"`).
`...`	Arguments passed to `identify.default`.

Details

This is a method for the generic function `identify` for point patterns on a linear network (objects of class `"lpp"`).

The point pattern `x` should first be plotted using `plot.lpp`. Then `identify(x)` reads the position of the graphics pointer each time the left mouse button is pressed. It then finds the point of the pattern `x` closest to the mouse position. If this closest point is sufficiently close to the mouse pointer, its index (and its mark if any) will be returned as part of the value of the call.

Each time a point of the pattern is identified, text will be displayed next to the point, showing its serial number (if `x` is unmarked) or its mark value (if `x` is marked).

Value

If `x` is unmarked, the result is a vector containing the serial numbers of the points in the pattern `x` that were identified. If `x` is marked, the result is a 2-column matrix, the first column containing the serial numbers and the second containing the marks for these points.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`, Rolf Turner `<rolfturner@posteo.net>` and Ege Rubak `<rubak@math.aau.dk>`.

See Also

`identify.ppp`, `identify`, `clicklpp`

`insertVertices` *Insert New Vertices in a Linear Network*

Description

Adds new vertices to a linear network at specified locations along the network.

Usage

```
insertVertices(L, ...)
```

Arguments

`L`	Linear network (object of class `"linnet"`) or point pattern on a linear network (object of class `"lpp"`).
`...`	Additional arguments passed to `as.lpp` specifying the positions of the new vertices along the network.

Details

This function adds new vertices at locations along an existing linear network.

The argument `L` can be either a linear network (class `"linnet"`) or some other object that includes a linear network.

The new vertex locations can be specified either as a point pattern (class `"lpp"` or `"ppp"`) or using coordinate vectors `x,y` or `seg,tp` or `x,y,seg,tp` as explained in the help for `as.lpp`.

This function breaks the existing line segments of `L` into pieces at the locations specified by the coordinates `seg,tp` and creates new vertices at these locations.

The result is the modified object, with an attribute `"id"` such that the `i`th added vertex has become the `id[i]`th vertex of the new network.

Value

An object of the same class as `L` representing the result of adding the new vertices. The result also has an attribute `"id"` as described in Details.

Author(s)

Adrian Baddeley

See Also

`addVertices` to create new vertices at locations which are not yet on the network.

`as.lpp`, `linnet`, `methods.linnet`, `joinVertices`, `thinNetwork`.

Examples

```
opa <- par(mfrow=c(1,3), mar=rep(0,4))
simplenet

plot(simplenet, main="")
plot(vertices(simplenet), add=TRUE)

# add two new vertices at specified local coordinates
L <- insertVertices(simplenet, seg=c(3,7), tp=c(0.2, 0.5))
L
plot(L, main="")
plot(vertices(L), add=TRUE)
id <- attr(L, "id")
id
plot(vertices(L)[id], add=TRUE, pch=16)

# add new vertices at three randomly-generated points
X <- runiflpp(3, simplenet)
LL <- insertVertices(simplenet, X)
plot(LL, main="")
plot(vertices(LL), add=TRUE)
ii <- attr(LL, "id")
plot(vertices(LL)[ii], add=TRUE, pch=16)
par(opa)
```

`integral.linim` *Integral on a Linear Network*

Description

Computes the integral (total value) of a function or pixel image over a linear network.

Usage

```
## S3 method for class 'linim'
integral(f, domain=NULL, weight=NULL, ...)

## S3 method for class 'linfun'
integral(f, domain=NULL, weight=NULL, ..., delta, nd)
```

Arguments

`f`	A pixel image on a linear network (class `"linim"`) or a function on a linear network (class `"linfun"`).
`domain`	Optional window specifying the domain of integration. Alternatively a tessellation.
`weight`	Optional numerical weight function for the integration. A pixel image (object of class `"linim"` or `"im"`), a function (object of class `"linfun"`, `"funxy"` or a a `function(x,y)`) or anything acceptable to `as.linim`.
`...`	Ignored.
`delta`	Optional. The step length (in coordinate units) for computing the approximate integral. A single positive number.
`nd`	Optional. Integer giving the approximate number of sample points on the network.

Details

The integral (total value of the function over the network) is calculated.

If `domain` is a window (class `"owin"`) then the integration will be restricted to this window. If `domain` is a tessellation (class `"tess"`) then the integral of `f` in each tile of `domain` will be computed.

If `weight` is given, effectively the integral of `weight * f` is computed.

Value

A single numeric or complex value (or a vector of such values if `domain` is a tessellation).

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>, Rolf Turner <`rolfturner@posteo.net`> and Ege Rubak <`rubak@math.aau.dk`>.

See Also

`linim`, `integral.im`

Examples

```
# make some data
xcoord <- linfun(function(x,y,seg,tp) { x }, simplenet)
integral(xcoord)
X <- as.linim(xcoord)
integral(X)

# integrals inside each tile of a tessellation
A <- quadrats(Frame(simplenet), 3)
integral(X, A)
```

`intensity.lpp` *Empirical Intensity of Point Pattern on Linear Network*

Description

Computes the average number of points per unit length in a point pattern on a linear network.

Usage

```
## S3 method for class 'lpp'
intensity(X, ...)
```

Arguments

`X`	A point pattern on a linear network (object of class `"lpp"`).
`...`	Ignored.

Details

This is a method for the generic function `intensity` It computes the empirical intensity of a point pattern on a linear network (object of class `"lpp"`), i.e. the average density of points per unit length.

If the point pattern is multitype, the intensities of the different types are computed separately.

Value

A numeric value (giving the intensity) or numeric vector (giving the intensity for each possible type).

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`> and Rolf Turner <`rolfturner@posteo.net`>

See Also

`intensity`, `intensity.ppp`

Examples

```
intensity(chicago)
```

`intersect.lintess` *Intersection of Tessellations on a Linear Network*

Description

Yields the intersection (common refinement) of two tessellations on a linear network.

Usage

```
intersect.lintess(X, Y)
```

Arguments

`X, Y`	Tessellations (objects of class `"lintess"`) on the same linear network, or data that define such tessellations. See Details.

Details

`X` and `Y` should be tessellations on a linear network (objects of class `"lintess"`) and should be defined on the same network. The algorithm finds the common refinement of the two tessellations. Each tile in the resulting tessellation is the intersection of a tile of `X` with a tile of `Y`.

Alternatively, one of the arguments `X` or `Y` can be a two-dimensional tessellation (object of class `"tess"`) while the other argument is a network or a tessellation on a network. The two-dimensional tessellation will be intersected with the network to produce a tessellation on the network, then intersected with the other tessellation on the network.

Value

Another tessellation (object of class `"lintess"`) on the same linear network as `X` and `Y`.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>.

See Also

`lintess`, `divide.linnet`, `chop.linnet`

Examples

```
X <- divide.linnet(runiflpp(4, simplenet))
Y <- divide.linnet(runiflpp(3, simplenet))
opa <- par(mfrow=c(1,3))
plot(X)
plot(Y)
plot(intersect.lintess(X,Y))
par(opa)
```

`is.connected.linnet` *Determine Whether a Linear Network is Connected*

Description

Determine whether a linear network is topologically connected.

Usage

```
## S3 method for class 'linnet'
is.connected(X, ...)
```

Arguments

`X`	A linear network (object of class `"linnet"`).
`...`	Arguments passed to `connected.linnet` to determine the connected components.

Details

The command `is.connected(X)` returns `TRUE` if the network `X` consists of a single, topologically-connected piece, and returns `FALSE` if `X` consists of several pieces which are not joined together.

The function `is.connected` is generic, with methods for several classes. This help file documents the method for linear networks, `is.connected.linnet`.

Value

A logical value.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>, Rolf Turner <`rolfturner@posteo.net`> and Ege Rubak <`rubak@math.aau.dk`>.

See Also

`is.connected`, `connected`, `connected.lpp`.

Examples

```
is.connected(simplenet)
```

---

is.marked.lppm *Test Whether A Point Process Model is Marked*

Description

Tests whether a fitted point process model on a network involves “marks” attached to the points.

Usage

```
  ## S3 method for class 'lppm'
is.marked(X, ...)
```

Arguments

`X`	Fitted point process model on a linear networ (object of class `"lppm"`) usually obtained from `lppm`.
`...`	Ignored.

Details

“Marks” are observations attached to each point of a point pattern. For example the `chicago` dataset contains the locations of crimes, each crime location being marked by the type of crime.

The argument `X` is a fitted point process model on a network (an object of class `"lppm"`) typically obtained by fitting a model to point pattern data using `lppm`.

This function returns `TRUE` if the *original data* (to which the model `X` was fitted) were a marked point pattern.

Note that this is not the same as testing whether the model involves terms that depend on the marks (i.e. whether the fitted model ignores the marks in the data). See the Examples for a trick to do this.

If this function returns `TRUE`, the implications are (for example) that any simulation of this model will require simulation of random marks as well as random point locations.

Value

Logical value, equal to `TRUE` if `X` is a model that was fitted to a marked point pattern dataset.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>` and Rolf Turner `<rolfturner@posteo.net>`

See Also

`is.marked`.

Examples

```
fit <- lppm(chicago ~ x)
is.marked(fit)
## result is TRUE, i.e. the data are marked

## To check whether the model involves marks:
"marks" %in% spatstat.utils::variablesinformula(formula(fit))
```

`is.multitype.lpp` *Test Whether A Point Pattern on a Network is Multitype*

Description

Tests whether a point pattern on a network has "marks" attached to the points which classify the points into several types.

Usage

```
  ## S3 method for class 'lpp'
is.multitype(X, na.action="warn", ...)
```

Arguments

`X`	Point pattern on a linear networl (object of class `"lpp"`).
`na.action`	String indicating what to do if `NA` values are encountered amongst the marks. Options are `"warn"`, `"fatal"` and `"ignore"`.
`...`	Ignored.

Details

"Marks" are observations attached to each point of a point pattern. For example the `chicago` dataset contains the locations of crimes, each crime location being marked by the type of crime.

This function tests whether the point pattern `X` contains or involves marked points, **and** that the marks are a factor. It is a method for the generic function `is.multitype`.

The argument `na.action` determines what action will be taken if the point pattern has a vector of marks but some or all of the marks are `NA`. Options are `"fatal"` to cause a fatal error; `"warn"` to issue a warning and then return `TRUE`; and `"ignore"` to take no action except returning `TRUE`.

Value

Logical value, equal to `TRUE` if `X` is a multitype point pattern.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>` and Rolf Turner `<rolfturner@posteo.net>`

See Also

`is.multitype`, `is.multitype.lppm`

Examples

```
is.multitype(chicago)
```

`is.multitype.lppm` *Test Whether A Point Process Model is Multitype*

Description

Tests whether a fitted point process model on a network involves “marks” attached to the points that classify the points into several types.

Usage

```
  ## S3 method for class 'lppm'
is.multitype(X, ...)
```

Arguments

`X`	Fitted point process model on a linear network (object of class `"lppm"`) usually obtained from `lppm`.
`...`	Ignored.

Details

“Marks” are observations attached to each point of a point pattern. For example the `chicago` dataset contains the locations of crimes, each crime location being marked by the type of crime.

The argument `X` is a fitted point process model on a network (an object of class `"lppm"`) typically obtained by fitting a model to point pattern data on a network using `lppm`.

This function returns `TRUE` if the *original data* (to which the model `X` was fitted) were a multitype point pattern.

Note that this is not the same as testing whether the model involves terms that depend on the marks (i.e. whether the fitted model ignores the marks in the data). See the Examples for a trick for doing this.

If this function returns `TRUE`, the implications are (for example) that any simulation of this model will require simulation of random marks as well as random point locations.

Value

Logical value, equal to `TRUE` if `X` is a model that was fitted to a multitype point pattern dataset.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>` and Rolf Turner `<rolfturner@posteo.net>`

See Also

is.multitype, is.multitype.lpp

Examples

```
fit <- lppm(chicago ~ x)
is.multitype(fit)
# TRUE because chicago data are multitype

## To check whether the model involves marks:
"marks" %in% spatstat.utils::variablesinformula(formula(fit))
```

---

is.stationary.lppm *Recognise Stationary and Poisson Point Process Models on a Network*

Description

Given a point process model that has been fitted to data on a network, determine whether the model is a stationary point process, and whether it is a Poisson point process.

Usage

```
## S3 method for class 'lppm'
is.stationary(x)

## S3 method for class 'lppm'
is.poisson(x)
```

Arguments

`x`	A fitted spatial point process model on a linear network (object of class `"lppm"`).

Details

The argument `x` represents a fitted spatial point process model on a linear network.

`is.stationary(x)` returns `TRUE` if `x` represents a stationary point process, and `FALSE` if not.

`is.poisson(x)` returns `TRUE` if `x` represents a Poisson point process, and `FALSE` if not.

The functions `is.stationary` and `is.poisson` are generic, with methods for many classes of models.

Value

A logical value.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>, Rolf Turner <`rolfturner@posteo.net`> and Ege Rubak <`rubak@math.aau.dk`>.

See Also

`is.marked` to determine whether a model is a marked point process.

`is.stationary`, `is.poisson` for generics.

`summary.lppm` for detailed information.

Model-fitting function `lppm`.

Examples

```
fit <- lppm(spiders ~ x)
is.stationary(fit)
is.poisson(fit)
```

`joinVertices` *Join Vertices in a Network*

Description

Join the specified vertices in a linear network, creating a new network.

Usage

```
joinVertices(L, from, to, marks=NULL)
```

Arguments

`L`	A linear network (object of class `"linnet"`) or point pattern on a linear network (object of class `"lpp"`).
`from, to`	Integers, or integer vectors of equal length, specifying the vertices which should be joined. Alternatively `from` can be a 2-column matrix of integers and `to` is missing or `NULL`.
`marks`	Optional vector or data frame of values associated with the new edges.

Details

Vertices of the network are numbered by their order of appearance in the point pattern `vertices(L)`.

If `from` and `to` are single integers, then the pair of vertices numbered `from` and `to` will be joined to make a new segment of the network. If `from` and `to` are vectors of integers, then vertex `from[i]` will be joined to vertex `to[i]` for each `i = 1,2,...`

If `L` is a network (class `"linnet"`), the result is another network, created by adding new segments. If `L` is a point pattern on a network (class `"lpp"`), the result is another point pattern object, created by adding new segments to the underlying network, and retaining the points.

In the resulting object, the new line segments are appended to the existing list of line segments.

Value

A linear network (object of class `"linnet"`) or point pattern on a linear network (object of class `"lpp"`).

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`, Rolf Turner `<rolfturner@posteo.net>` and Ege Rubak `<rubak@math.aau.dk>`.

See Also

`linnet`, `methods.linnet`, `thinNetwork`

Examples

```
snet <- joinVertices(simplenet, 4, 8)
plot(solist(simplenet, snet), main="")
X <- runiflpp(3, simplenet)
Y <- joinVertices(X, 4, 8)
```

`lineardirichlet` *Dirichlet Tessellation on a Linear Network*

Description

Given a point pattern on a linear network, compute the Dirichlet (or Voronoi or Thiessen) tessellation induced by the points.

Usage

```
lineardirichlet(X)
```

Arguments

`X`	Point pattern on a linear network (object of class `"lpp"`).

Details

The Dirichlet tessellation induced by a point pattern `X` on a linear network `L` is a partition of `L` into subsets. The subset `L[i]` associated with the data point `X[i]` is the part of `L` lying closer to `X[i]` than to any other data point `X[j]`, where distance is measured by the shortest path.

Value

A tessellation on a linear network (object of class `"lintess"`).

Missing tiles

If the linear network is not connected, and if one of the connected components contains no data points, then the Dirichlet tessellation is mathematically undefined inside this component. The resulting tessellation object includes a tile with label `NA`, which contains this component of the network. A plot of the tessellation will not show this tile.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`.

See Also

`lintess`.

For the Dirichlet tessellation in two-dimensional space, see `dirichlet`.

Examples

```
X <- runiflpp(5, simplenet)
plot(lineardirichlet(X), lwd=3)
points(X)
```

`lineardisc` *Compute Disc of Given Radius in Linear Network*

Description

Computes the 'disc' of given radius and centre in a linear network.

Usage

```
lineardisc(L, x = locator(1), r, plotit = TRUE,
           cols=c("blue", "red","green"), add=TRUE)

lineardisclength(L, x = locator(1), r)

countends(L, x = locator(1), r, toler=NULL, internal=list())
```

Arguments

`L`	Linear network (object of class `"linnet"`).
`x`	Location of centre of disc. Either a point pattern (object of class `"ppp"`) containing exactly 1 point, or a numeric vector of length 2.
`r`	Radius of disc.
`plotit`	Logical. Whether to plot the disc.

`add`	Logical. If `add=TRUE` (the default), the disc will be plotted on the current plot frame. If `add=FALSE`, a new plot frame will be started, the entire network will be displayed, and then the disc will be plotted over this.
`cols`	Colours for plotting the disc. A numeric or character vector of length 3 specifying the colours of the disc centre, disc lines and disc endpoints respectively.
`toler`	Optional. Distance threshold for `countends`. See Details. There is a sensible default.
`internal`	Argument for internal use by the package.

Details

The ‘disc’ $B(u, r)$ of centre x and radius r in a linear network L is the set of all points u in L such that the shortest path distance from x to u is less than or equal to r. This is a union of line segments contained in L.

The *relative boundary* of the disc $B(u, r)$ is the set of points v such that the shortest path distance from x to u is *equal* to r.

The function `lineardisc` computes the disc of radius r and its relative boundary, optionally plots them, and returns them. The faster function `lineardisclength` computes only the total length of the disc, and `countends` computes only the number of endpoints of the disc.

Note that `countends` requires the linear network `L` to be given in the non-sparse matrix format (see the argument `sparse` in `linnet` or `as.linnet`) while `lineardisc` and `lineardisclength` accept both sparse and non-sparse formats.

The optional threshold `toler` is used to suppress numerical errors in `countends`. If the distance from u to a network vertex v is between `r-toler` and `r+toler`, the vertex will be treated as lying on the relative boundary.

Value

The value of `lineardisc` is a list with two entries:

`lines`	Line segment pattern (object of class `"psp"`) representing the interior disc
`endpoints`	Point pattern (object of class `"ppp"`) representing the relative boundary of the disc.

The value of `lineardisclength` is a single number giving the total length of the disc.

The value of `countends` is an integer giving the number of points in the relative boundary.

Author(s)

Ang Qi Wei <`aqw07398@hotmail.com`> and Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>

References

Ang, Q.W. (2010) *Statistical methodology for events on a network.* Master’s thesis, School of Mathematics and Statistics, University of Western Australia.

Ang, Q.W., Baddeley, A. and Nair, G. (2012) Geometrically corrected second-order analysis of events on a linear network, with applications to ecology and criminology. *Scandinavian Journal of Statistics* **39**, 591–617.

See Also

linnet

Examples

```
 # letter 'A'
 v <- ppp(x=(-2):2, y=3*c(0,1,2,1,0), c(-3,3), c(-1,7))
 edg <- cbind(1:4, 2:5)
 edg <- rbind(edg, c(2,4))
 letterA <- linnet(v, edges=edg)
 plot(letterA)

di <- lineardisc(letterA, c(0,3), 1.6)
di

# count the endpoints more efficiently
countends(letterA, c(0,3), 1.6)
# cross-check
npoints(di$endpoints)

# measure the length more efficiently
lineardisclength(letterA, c(0,3), 1.6)
# cross-check
sum(lengths_psp(di$lines))
```

`linearJinhom` *Inhomogeneous Linear J-function for Point Processes on Linear Networks*

Description

Computes an estimate of the inhomogeneous linear J-function for a point pattern on a linear network.

Usage

```
linearJinhom(X, lambda = NULL, lmin=NULL,
              ...,
              r=NULL, rmax=NULL,
              distance=c("path","euclidean"),
              densitymethod=c("kernel", "Voronoi"),
              sigma=bw.scott.iso,
              f=0.2, nrep=200, ngrid=256)
```

Arguments

`X`	Point pattern on linear network (object of class `"lpp"`).

`lambda`	Intensity values for the point pattern. Either a numeric vector, a `function`, a pixel image (object of class `"im"` or `"linim"`) or a fitted point process model (object of class `"ppm"` or `"lppm"`).
`lmin`	Optional. The minimum possible value of the intensity over the network. A positive numerical value.
`r`	Optional. Numeric vector of values of the function argument r. There is a sensible default.
`rmax`	Optional. Numeric value specifying the largest desired value of r. There is a sensible default.
`distance`	A string (partially matched) specifying the metric that will be used to measure distances between points on the network: `distance="path"` is the shortest-path distance, and `distance="euclidean"` is the Euclidean distance.
`densitymethod`	String (partially matched) specifying the method that will be used to estimate the intensity `lambda`, if `lambda` is not given: `densitymethod="kernel"` specifies kernel smoothing and `densitymethod="Voronoi"` specifies Voronoi estimation. See Details.
`sigma`	Smoothing bandwidth used to estimate `lambda` by kernel smoothing, if `lambda` is not given and `densitymethod="kernel"`. Either a numeric value, or a function that can be applied to `X` to compute the bandwidth.
`f`, `nrep`	Arguments passed to the algorithm for estimating the intensity by Voronoi estimation, if `lambda` is not given and `densitymethod="Voronoi"`.
`...`	Additional arguments passed to the algorithms that estimate the intensity, if `lambda` is not given.
`ngrid`	Integer specifying the number of sample points on the network that will be used to estimate the inhomogeneous empty space function F.

Details

This function computes the geometrically corrected inhomogeneous linear J-function for point processes on linear networks defined by Cronie et al (2020).

The argument `lambda` is the (estimated) intensity of the underlying point process. It should be either a numeric vector (giving intensity values at the points of `X`), a `function`, a pixel image (object of class `"im"` or `"linim"`) or a fitted point process model (object of class `"ppm"` or `"lppm"`).

If `lambda` is not given, it will be estimated from the observed point pattern `X` as follows:

- If `densitymethod="kernel"`, the intensity will be estimated by kernel smoothing, using the fast estimator `densityQuick.lpp` introduced by Rakshit et al (2019). The smoothing bandwidth `sigma` is required. It may be specified as a numeric value, or as a function that can be applied to `X` to obtain a bandwidth value. Examples of the latter include `bw.scott.iso` and `bw.lppl`. Additional arguments `...` will be passed to `sigma` and to `densityQuick.lpp`.
- If `densitymethod = "Voronoi"`, the intensity will be estimated using the resample-smoothed Voronoi estimator `densityVoronoi.lpp` introduced by Moradi et al (2019). The arguments `f` and `nrep` are passed to `densityVoronoi.lpp` and determine the retention probability and the number of replicates, respectively. Additional arguments `...` will be passed to `densityVoronoi.lpp`.

Value

Function value table (object of class `"fv"`).

Author(s)

Mehdi Moradi <`m2.moradi@yahoo.com`> and Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>.

References

Cronie, O., Moradi, M., and Mateu, J. (2020) Inhomogeneous higher-order summary statistics for point processes on linear networks. *Statistics and Computing* **30** (6) 1221–1239.

Moradi, M., Cronie, 0., Rubak, E., Lachieze-Rey, R., Mateu, J. and Baddeley, A. (2019) Resample-smoothing of Voronoi intensity estimators. *Statistics and Computing* **29** (5) 995–1010.

Rakshit, S., Davies, T., Moradi, M., McSwiggan, G., Nair, G., Mateu, J. and Baddeley, A. (2019) Fast kernel smoothing of point patterns on a large network using 2D convolution. *International Statistical Review* **87** (3) 531–556. DOI: 10.1111/insr.12327.

See Also

`bw.scott.iso`, `bw.lppl`, `densityVoronoi.lpp`, `densityQuick.lpp`

`linearKinhom`

`Jinhom`

Examples

```
if(interactive()) {
  plot(linearJinhom(spiders))
} else {
  bottomhalf <- owin(c(0, 1125), c(0, 500))
  plot(linearJinhom(spiders[bottomhalf]))
}
```

`linearK` *Linear K Function*

Description

Computes an estimate of the linear K function for a point pattern on a linear network.

Usage

```
linearK(X, r=NULL, ..., correction="Ang", ratio=FALSE)
```

Arguments

`X`	Point pattern on linear network (object of class `"lpp"`).
`r`	Optional. Numeric vector of values of the function argument r. There is a sensible default.
`...`	Ignored.
`correction`	Geometry correction. Either `"none"` or `"Ang"`. See Details.
`ratio`	Logical. If `TRUE`, the numerator and denominator of the estimate will also be saved, for use in analysing replicated point patterns.

Details

This command computes the linear K function from point pattern data on a linear network.

If `correction="none"`, the calculations do not include any correction for the geometry of the linear network. The result is the network K function as defined by Okabe and Yamada (2001).

If `correction="Ang"`, the pair counts are weighted using Ang's correction (Ang, 2010; Ang et al, 2012).

Value

Function value table (object of class `"fv"`).

Author(s)

Ang Qi Wei <`aqw07398@hotmail.com`> and Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>.

References

Ang, Q.W. (2010) Statistical methodology for spatial point patterns on a linear network. MSc thesis, University of Western Australia.

Ang, Q.W., Baddeley, A. and Nair, G. (2012) Geometrically corrected second-order analysis of events on a linear network, with applications to ecology and criminology. *Scandinavian Journal of Statistics* **39**, 591–617.

Okabe, A. and Yamada, I. (2001) The K-function method on a network and its computational implementation. *Geographical Analysis* **33**, 271-290.

See Also

`compileK`, `lpp`

Examples

```
X <- rpoislpp(5, simplenet)
linearK(X)
linearK(X, correction="none")
```

`linearKcross` *Multitype K Function (Cross-type) for Linear Point Pattern*

Description

For a multitype point pattern on a linear network, estimate the multitype K function which counts the expected number of points of type j within a given distance of a point of type i.

Usage

```
linearKcross(X, i, j, r=NULL, ..., correction="Ang")
```

Arguments

`X`	The observed point pattern, from which an estimate of the cross type K function $K_{ij}(r)$ will be computed. An object of class `"lpp"` which must be a multitype point pattern (a marked point pattern whose marks are a factor).
`i`	Number or character string identifying the type (mark value) of the points in `X` from which distances are measured. Defaults to the first level of `marks(X)`.
`j`	Number or character string identifying the type (mark value) of the points in `X` to which distances are measured. Defaults to the second level of `marks(X)`.
`r`	numeric vector. The values of the argument r at which the K-function $K_{ij}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
`correction`	Geometry correction. Either `"none"` or `"Ang"`. See Details.
`...`	Ignored.

Details

This is a counterpart of the function `Kcross` for a point pattern on a linear network (object of class `"lpp"`).

The arguments `i` and `j` will be interpreted as levels of the factor `marks(X)`. If `i` and `j` are missing, they default to the first and second level of the marks factor, respectively.

The argument `r` is the vector of values for the distance r at which $K_{ij}(r)$ should be evaluated. The values of r must be increasing nonnegative numbers and the maximum r value must not exceed the radius of the largest disc contained in the window.

Value

An object of class `"fv"` (see `fv.object`).

Warnings

The arguments `i` and `j` are interpreted as levels of the factor `marks(X)`. Beware of the usual trap with factors: numerical values are not interpreted in the same way as character values.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>

References

Baddeley, A, Jammalamadaka, A. and Nair, G. (2014) Multitype point process analysis of spines on the dendrite network of a neuron. *Applied Statistics* (Journal of the Royal Statistical Society, Series C), **63**, 673–694.

See Also

`linearKdot`, `linearK`.

Examples

```
K <- linearKcross(chicago, "assault", "robbery")
```

`linearKcross.inhom` *Inhomogeneous multitype K Function (Cross-type) for Linear Point Pattern*

Description

For a multitype point pattern on a linear network, estimate the inhomogeneous multitype K function which counts the expected number of points of type j within a given distance of a point of type i.

Usage

```
linearKcross.inhom(X, i, j, lambdaI=NULL, lambdaJ=NULL,
                   r=NULL, ..., correction="Ang", normalise=TRUE,
                   sigma=NULL)
```

Arguments

`X`	The observed point pattern, from which an estimate of the cross type K function $K_{ij}(r)$ will be computed. An object of class `"lpp"` which must be a multitype point pattern (a marked point pattern whose marks are a factor).
`i`	Number or character string identifying the type (mark value) of the points in `X` from which distances are measured. Defaults to the first level of `marks(X)`.
`j`	Number or character string identifying the type (mark value) of the points in `X` to which distances are measured. Defaults to the second level of `marks(X)`.
`lambdaI`	Intensity values for the points of type `i`. Either a numeric vector, a `function`, a pixel image (object of class `"im"` or `"linim"`) or a fitted point process model (object of class `"ppm"` or `"lppm"`) or `NULL`.
`lambdaJ`	Intensity values for the points of type `j`. Either a numeric vector, a `function`, a pixel image (object of class `"im"` or `"linim"`) or a fitted point process model (object of class `"ppm"` or `"lppm"`) or `NULL`.

`r`	numeric vector. The values of the argument r at which the K-function $K_{ij}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
`correction`	Geometry correction. Either `"none"` or `"Ang"`. See Details.
`...`	Arguments passed to `lambdaI` and `lambdaJ` if they are functions.
`normalise`	Logical. If `TRUE` (the default), the denominator of the estimator is data-dependent (equal to the sum of the reciprocal intensities at the points of type `i`), which reduces the sampling variability. If `FALSE`, the denominator is the length of the network.
`sigma`	Smoothing bandwidth passed to `density.lpp` for estimation of intensities when either `lambdaI` or `lambdaJ` is `NULL`.

Details

This is a counterpart of the function `Kcross.inhom` for a point pattern on a linear network (object of class `"lpp"`).

The arguments `i` and `j` will be interpreted as levels of the factor `marks(X)`. If `i` and `j` are missing, they default to the first and second level of the marks factor, respectively.

The argument `r` is the vector of values for the distance r at which $K_{ij}(r)$ should be evaluated. The values of r must be increasing nonnegative numbers and the maximum r value must not exceed the radius of the largest disc contained in the window.

If `lambdaI` or `lambdaJ` is missing or `NULL`, it will be estimated by kernel smoothing using `density.lpp`.

If `lambdaI` or `lambdaJ` is a fitted point process model, the default behaviour is to update the model by re-fitting it to the data, before computing the fitted intensity. This can be disabled by setting `update=FALSE`.

Value

An object of class `"fv"` (see `fv.object`).

Warnings

The arguments `i` and `j` are interpreted as levels of the factor `marks(X)`. Beware of the usual trap with factors: numerical values are not interpreted in the same way as character values.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`.

References

Baddeley, A, Jammalamadaka, A. and Nair, G. (2014) Multitype point process analysis of spines on the dendrite network of a neuron. *Applied Statistics* (Journal of the Royal Statistical Society, Series C), **63**, 673–694.

See Also

`linearKdot`, `linearK`.

Examples

```
lam <- table(marks(chicago))/(summary(chicago)$totlength)
lamI <- function(x,y,const=lam[["assault"]]){ rep(const, length(x)) }
lamJ <- function(x,y,const=lam[["robbery"]]){ rep(const, length(x)) }

K <- linearKcross.inhom(chicago, "assault", "robbery", lamI, lamJ)

# using fitted models for the intensity
# fit <- lppm(chicago ~marks + x)
# K <- linearKcross.inhom(chicago, "assault", "robbery", fit, fit)
```

`linearKdot` *Multitype K Function (Dot-type) for Linear Point Pattern*

Description

For a multitype point pattern on a linear network, estimate the multitype K function which counts the expected number of points (of any type) within a given distance of a point of type i.

Usage

```
linearKdot(X, i, r=NULL, ..., correction="Ang")
```

Arguments

`X`	The observed point pattern, from which an estimate of the dot type K function $K_{i\bullet}(r)$ will be computed. An object of class `"lpp"` which must be a multitype point pattern (a marked point pattern whose marks are a factor).
`i`	Number or character string identifying the type (mark value) of the points in `X` from which distances are measured. Defaults to the first level of `marks(X)`.
`r`	numeric vector. The values of the argument r at which the K-function $K_{i\bullet}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
`correction`	Geometry correction. Either `"none"` or `"Ang"`. See Details.
`...`	Ignored.

Details

This is a counterpart of the function `Kdot` for a point pattern on a linear network (object of class `"lpp"`).

The argument `i` will be interpreted as levels of the factor `marks(X)`. If `i` is missing, it defaults to the first level of the marks factor.

The argument `r` is the vector of values for the distance r at which $K_{i\bullet}(r)$ should be evaluated. The values of r must be increasing nonnegative numbers and the maximum r value must not exceed the radius of the largest disc contained in the window.

Value

An object of class `"fv"` (see `fv.object`).

Warnings

The argument `i` is interpreted as a level of the factor `marks(X)`. Beware of the usual trap with factors: numerical values are not interpreted in the same way as character values.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

References

Baddeley, A, Jammalamadaka, A. and Nair, G. (2014) Multitype point process analysis of spines on the dendrite network of a neuron. *Applied Statistics* (Journal of the Royal Statistical Society, Series C), **63**, 673–694.

See Also

`Kdot`, `linearKcross`, `linearK`.

Examples

```
K <- linearKdot(chicago, "assault")
```

`linearKdot.inhom` *Inhomogeneous multitype K Function (Dot-type) for Linear Point Pattern*

Description

For a multitype point pattern on a linear network, estimate the inhomogeneous multitype K function which counts the expected number of points (of any type) within a given distance of a point of type i.

Usage

```
linearKdot.inhom(X, i, lambdaI=NULL, lambdadot=NULL, r=NULL, ...,
                 correction="Ang", normalise=TRUE, sigma=NULL)
```

Arguments

`X`	The observed point pattern, from which an estimate of the dot type K function $K_{i\bullet}(r)$ will be computed. An object of class `"lpp"` which must be a multitype point pattern (a marked point pattern whose marks are a factor).
`i`	Number or character string identifying the type (mark value) of the points in `X` from which distances are measured. Defaults to the first level of `marks(X)`.
`lambdaI`	Intensity values for the points of type `i`. Either a numeric vector, a `function`, a pixel image (object of class `"im"` or `"linim"`) or a fitted point process model (object of class `"ppm"` or `"lppm"`) or `NULL`.
`lambdadot`	Intensity values for all points of `X`. Either a numeric vector, a `function`, a pixel image (object of class `"im"` or `"linim"`) or a fitted point process model (object of class `"ppm"` or `"lppm"`) or `NULL`.
`r`	numeric vector. The values of the argument r at which the K-function $K_{i\bullet}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
`correction`	Geometry correction. Either `"none"` or `"Ang"`. See Details.
`...`	Arguments passed to `lambdaI` and `lambdadot` if they are functions.
`normalise`	Logical. If `TRUE` (the default), the denominator of the estimator is data-dependent (equal to the sum of the reciprocal intensities at the points of type `i`), which reduces the sampling variability. If `FALSE`, the denominator is the length of the network.
`sigma`	Smoothing bandwidth passed to `density.lpp` for estimation of intensities when either `lambdaI` or `lambdadot` is `NULL`.

Details

This is a counterpart of the function `Kdot.inhom` for a point pattern on a linear network (object of class `"lpp"`).

The argument `i` will be interpreted as levels of the factor `marks(X)`. If `i` is missing, it defaults to the first level of the marks factor.

The argument `r` is the vector of values for the distance r at which $K_{i\bullet}(r)$ should be evaluated. The values of r must be increasing nonnegative numbers and the maximum r value must not exceed the radius of the largest disc contained in the window.

If `lambdaI` or `lambdadot` is missing, it will be estimated by kernel smoothing using `density.lpp`.

If `lambdaI` or `lambdadot` is a fitted point process model, the default behaviour is to update the model by re-fitting it to the data, before computing the fitted intensity. This can be disabled by setting `update=FALSE`.

Value

An object of class `"fv"` (see `fv.object`).

Warnings

The argument `i` is interpreted as a level of the factor `marks(X)`. Beware of the usual trap with factors: numerical values are not interpreted in the same way as character values.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>

References

Baddeley, A, Jammalamadaka, A. and Nair, G. (2014) Multitype point process analysis of spines on the dendrite network of a neuron. *Applied Statistics* (Journal of the Royal Statistical Society, Series C), **63**, 673–694.

See Also

`linearKdot`, `linearK`.

Examples

```
lam <- table(marks(chicago))/(summary(chicago)$totlength)
lamI <- function(x,y,const=lam[["assault"]]){ rep(const, length(x)) }
lam. <- function(x,y,const=sum(lam)){ rep(const, length(x)) }

K <- linearKdot.inhom(chicago, "assault", lamI, lam.)

# using fitted models for the intensity
# fit <- lppm(chicago ~marks + x)
# linearKdot.inhom(chicago, "assault", fit, fit)
```

`linearKEuclid` *Linear K Function Using Euclidean Distance*

Description

Computes an estimate of the linear K function based on Euclidean distances, for a point pattern on a linear network.

Usage

```
linearKEuclid(X, r = NULL, ...)
```

Arguments

`X`	Point pattern on linear network (object of class `"lpp"`).
`r`	Optional. Numeric vector of values of the function argument r. There is a sensible default.
`...`	Ignored.

Details

This command computes an estimate of the linear K function based on Euclidean distances between the points, as described by Rakshit, Nair and Baddeley (2017).

This is different from the linear `K` function based on shortest-path distances, which is computed by `linearK`.

The linear K function based on Euclidean distances is defined in equation (20) of Rakshit, Nair and Baddeley (2017). The estimate is computed from the point pattern as described in equation (25).

Value

Function value table (object of class `"fv"`).

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`.

References

Rakshit. S., Nair, G. and Baddeley, A. (2017) Second-order analysis of point patterns on a network using any distance metric. *Spatial Statistics* **22** (1) 129–154.

See Also

`linearpcfEuclid`, `linearKEuclidInhom`.

See `linearK` for the corresponding function based on shortest-path distances.

Examples

```
X <- rpoislpp(5, simplenet)
K <- linearKEuclid(X)
```

`linearKEuclidInhom` *Inhomogeneous Linear K Function Based on Euclidean Distances*

Description

Computes an estimate of the inhomogeneous linear K function based on Euclidean distances, for a point pattern on a linear network.

Usage

```
linearKEuclidInhom(X, lambda = NULL, r = NULL, ...,
   normalise = TRUE, normpower = 2, update = TRUE,
   leaveoneout = TRUE, sigma=NULL)
```

Arguments

`X`	Point pattern on linear network (object of class `"lpp"`).
`lambda`	Intensity values for the point pattern. Either a numeric vector, a `function`, a pixel image (object of class `"im"` or `"linim"`) or a fitted point process model (object of class `"ppm"` or `"lppm"`) or `NULL`.
`r`	Optional. Numeric vector of values of the function argument r. There is a sensible default.
`...`	Ignored.
`normalise`	Logical. If `TRUE` (the default), the denominator of the estimator is data-dependent (equal to the sum of the reciprocal intensities at the data points, raised to `normpower`), which reduces the sampling variability. If `FALSE`, the denominator is the length of the network.
`normpower`	Integer (usually either 1 or 2). Normalisation power. See Details.
`update`	Logical value indicating what to do when `lambda` is a fitted model (class `"lppm"` or `"ppm"`). If `update=TRUE` (the default), the model will first be refitted to the data `X` (using `update.lppm` or `update.ppm`) before the fitted intensity is computed. If `update=FALSE`, the fitted intensity of the model will be computed without re-fitting it to `X`.
`leaveoneout`	Logical value specifying whether to use a leave-one-out rule when calculating the intensity. See Details.
`sigma`	Smoothing bandwidth (passed to `density.lpp`) for kernel density estimation of the intensity when `lambda=NULL`.

Details

This command computes the inhomogeneous version of the linear K function based on *Euclidean* distances, for a point pattern on a linear network.

This is different from the inhomogeneous K function based on *shortest-path* distances, which is computed by `linearKinhom`.

The inhomogeneous K function based on *Euclidean* distances is defined in equation (23) of Rakshit, Nair and Baddeley (2017). Estimation is performed as described in equation (28).

The argument `lambda` should provide estimated values of the intensity of the point process at each point of `X`.

If `lambda=NULL`, the intensity will be estimated by kernel smoothing by calling `density.lpp` with the smoothing bandwidth `sigma`, and with any other relevant arguments that might be present in `...`. A leave-one-out kernel estimate will be computed if `leaveoneout=TRUE`.

If `lambda` is given, then it is expected to provide estimated values of the intensity of the point process at each point of `X`. The argument `lambda` may be a numeric vector (of length equal to the number of points in `X`), or a `function(x,y)` that will be evaluated at the points of `X` to yield numeric values, or a pixel image (object of class `"im"`) or a fitted point process model (object of class `"ppm"` or `"lppm"`).

If `lambda` is a fitted point process model, the default behaviour is to update the model by re-fitting it to the data, before computing the fitted intensity. This can be disabled by setting `update=FALSE`.

The intensity at data points will be computed by `fitted.lppm` or `fitted.ppm`. A leave-one-out estimate will be computed if `leaveoneout=TRUE` and `update=TRUE`.

If `normalise=TRUE` (the default), then the estimate is multiplied by $c^{\text{normpower}}$ where $c = \text{length}(L)/\sum(1/\lambda(x_i))$. This rescaling reduces the variability and bias of the estimate in small samples and in cases of very strong inhomogeneity. The default value of `normpower` is 1 (for consistency with previous versions of **spatstat**) but the most sensible value is 2, which would correspond to rescaling the `lambda` values so that $\sum(1/\lambda(x_i)) = \text{area}(W)$.

Value

Function value table (object of class `"fv"`).

Warning

Older versions of `linearKEuclidInhom` interpreted `lambda=NULL` to mean that the homogeneous function `linearKEuclid` should be computed. This was changed to the current behaviour in version `3.1-0` of **spatstat.linnet**.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>.

References

Rakshit. S., Nair, G. and Baddeley, A. (2017) Second-order analysis of point patterns on a network using any distance metric. *Spatial Statistics* **22** (1) 129–154.

See Also

`linearpcfEuclidInhom`, `linearKEuclid`.

See `linearKinhom` for the corresponding function based on shortest-path distances.

Examples

```
X <- rpoislpp(5, simplenet)
fit <- lppm(X ~x)
K <- linearKEuclidInhom(X, lambda=fit)
plot(K)
```

`linearKinhom` *Inhomogeneous Linear K Function*

Description

Computes an estimate of the inhomogeneous linear K function for a point pattern on a linear network.

Usage

```
linearKinhom(X, lambda=NULL, r=NULL, ..., correction="Ang",
             normalise=TRUE, normpower=1,
     update=TRUE, leaveoneout=TRUE, sigma=NULL, ratio=FALSE)
```

Arguments

`X`	Point pattern on linear network (object of class `"lpp"`).
`lambda`	Intensity values for the point pattern. Either a numeric vector, a `function`, a pixel image (object of class `"im"` or `"linim"`) or a fitted point process model (object of class `"ppm"` or `"lppm"`) or `NULL`.
`r`	Optional. Numeric vector of values of the function argument r. There is a sensible default. Users are advised not to specify `r` in normal usage.
`...`	Ignored.
`correction`	Geometry correction. Either `"none"` or `"Ang"`. See Details.
`normalise`	Logical. If `TRUE` (the default), the denominator of the estimator is data-dependent (equal to the sum of the reciprocal intensities at the data points, raised to `normpower`), which reduces the sampling variability. If `FALSE`, the denominator is the length of the network.
`normpower`	Integer (usually either 1 or 2). Normalisation power. See Details.
`update`	Logical value indicating what to do when `lambda` is a fitted model (class `"lppm"` or `"ppm"`). If `update=TRUE` (the default), the model will first be refitted to the data `X` (using `update.lppm` or `update.ppm`) before the fitted intensity is computed. If `update=FALSE`, the fitted intensity of the model will be computed without re-fitting it to `X`.
`leaveoneout`	Logical value specifying whether to use a leave-one-out rule when calculating the intensity. See Details.
`sigma`	Smoothing bandwidth (passed to `density.lpp`) for kernel density estimation of the intensity when `lambda=NULL`.
`ratio`	Logical. If `TRUE`, the numerator and denominator of the estimate will also be saved, for use in analysing replicated point patterns.

Details

This command computes the inhomogeneous version of the linear K function from point pattern data on a linear network.

The argument `lambda` should provide estimated values of the intensity of the point process at each point of `X`.

If `lambda=NULL`, the intensity will be estimated by kernel smoothing by calling `density.lpp` with the smoothing bandwidth `sigma`, and with any other relevant arguments that might be present in `...`. A leave-one-out kernel estimate will be computed if `leaveoneout=TRUE`.

If `lambda` is given, it may be a numeric vector (of length equal to the number of points in `X`), or a `function(x,y)` that will be evaluated at the points of `X` to yield numeric values, or a pixel image (object of class `"im"`) or a fitted point process model (object of class `"ppm"` or `"lppm"`).

If `lambda` is a fitted point process model, the default behaviour is to update the model by re-fitting it to the data, before computing the fitted intensity. This can be disabled by setting `update=FALSE`. The intensity at data points will be computed by `fitted.lppm` or `fitted.ppm`. A leave-one-out estimate will be computed if `leaveoneout=TRUE` and `update=TRUE`.

If `correction="none"`, the calculations do not include any correction for the geometry of the linear network. If `correction="Ang"`, the pair counts are weighted using Ang's correction (Ang, 2010).

Each estimate is initially computed as

$$\widehat{K}_{\mathrm{inhom}}(r) = \frac{1}{\mathrm{length}(L)} \sum_i \sum_j \frac{1\{d_{ij} \le r\} e(x_i, x_j)}{\lambda(x_i)\lambda(x_j)}$$

where `L` is the linear network, d_{ij} is the distance between points x_i and x_j, and $e(x_i, x_j)$ is a weight. If `correction="none"` then this weight is equal to 1, while if `correction="Ang"` the weight is $e(x_i, x_j, r) = 1/m(x_i, d_{ij})$ where $m(u, t)$ is the number of locations on the network that lie exactly t units distant from location u by the shortest path.

If `normalise=TRUE` (the default), then the estimates described above are multiplied by $c^{\mathrm{normpower}}$ where $c = \mathrm{length}(L)/\sum(1/\lambda(x_i))$. This rescaling reduces the variability and bias of the estimate in small samples and in cases of very strong inhomogeneity. The default value of `normpower` is 1 (for consistency with previous versions of **spatstat**) but the most sensible value is 2, which would correspond to rescaling the `lambda` values so that $\sum(1/\lambda(x_i)) = \mathrm{area}(W)$.

Value

Function value table (object of class `"fv"`).

Warning

Older versions of `linearKinhom` interpreted `lambda=NULL` to mean that the homogeneous function `linearK` should be computed. This was changed to the current behaviour in version `3.1-0` of **spatstat.linnet**.

Author(s)

Ang Qi Wei `<aqw07398@hotmail.com>` and Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

References

Ang, Q.W. (2010) Statistical methodology for spatial point patterns on a linear network. MSc thesis, University of Western Australia.

Ang, Q.W., Baddeley, A. and Nair, G. (2012) Geometrically corrected second-order analysis of events on a linear network, with applications to ecology and criminology. *Scandinavian Journal of Statistics* **39**, 591–617.

See Also

`lpp`

Examples

```
X <- rpoislpp(5, simplenet)
fit <- lppm(X ~x)
K <- linearKinhom(X, lambda=fit)
plot(K)
Ke <- linearKinhom(X, sigma=bw.lppl)
plot(Ke)
```

`linearmarkconnect` *Mark Connection Function for Multitype Point Pattern on Linear Network*

Description

For a multitype point pattern on a linear network, estimate the mark connection function from points of type i to points of type j.

Usage

```
linearmarkconnect(X, i, j, r=NULL, ...)
```

Arguments

`X`	The observed point pattern, from which an estimate of the mark connection function $p_{ij}(r)$ will be computed. An object of class `"lpp"` which must be a multitype point pattern (a marked point pattern whose marks are a factor).
`i`	Number or character string identifying the type (mark value) of the points in `X` from which distances are measured. Defaults to the first level of `marks(X)`.
`j`	Number or character string identifying the type (mark value) of the points in `X` to which distances are measured. Defaults to the second level of `marks(X)`.
`r`	numeric vector. The values of the argument r at which the function $p_{ij}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
`...`	Arguments passed to `linearpcfcross` and `linearpcf`.

Details

This is a counterpart of the function `markconnect` for a point pattern on a linear network (object of class `"lpp"`).

The argument `i` will be interpreted as levels of the factor `marks(X)`. If `i` is missing, it defaults to the first level of the marks factor.

The argument `r` is the vector of values for the distance r at which $p_{ij}(r)$ should be evaluated. The values of r must be increasing nonnegative numbers and the maximum r value must not exceed the radius of the largest disc contained in the window.

Value

An object of class `"fv"` (see `fv.object`).

Warnings

The argument `i` is interpreted as a level of the factor `marks(X)`. Beware of the usual trap with factors: numerical values are not interpreted in the same way as character values.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

References

Baddeley, A, Jammalamadaka, A. and Nair, G. (2014) Multitype point process analysis of spines on the dendrite network of a neuron. *Applied Statistics* (Journal of the Royal Statistical Society, Series C), **63**, 673–694.

See Also

`linearpcfcross`, `linearpcf`, `linearmarkequal`, `markconnect`.

Examples

```
pab <- linearmarkconnect(chicago, "assault", "burglary")

# plot(alltypes(chicago, linearmarkconnect))
```

`linearmarkequal` *Mark Connection Function for Multitype Point Pattern on Linear Network*

Description

For a multitype point pattern on a linear network, estimate the mark connection function from points of type i to points of type j.

Usage

```
linearmarkequal(X, r=NULL, ...)
```

Arguments

`X`	The observed point pattern, from which an estimate of the mark connection function $p_{ij}(r)$ will be computed. An object of class `"lpp"` which must be a multitype point pattern (a marked point pattern whose marks are a factor).
`r`	numeric vector. The values of the argument r at which the function $p_{ij}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
`...`	Arguments passed to `linearpcfcross` and `linearpcf`.

Details

This is the mark equality function for a point pattern on a linear network (object of class `"lpp"`).

The argument `r` is the vector of values for the distance r at which $p_{ij}(r)$ should be evaluated. The values of r must be increasing nonnegative numbers and the maximum r value must not exceed the radius of the largest disc contained in the window.

Value

An object of class `"fv"` (see `fv.object`).

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>

References

Baddeley, A, Jammalamadaka, A. and Nair, G. (2014) Multitype point process analysis of spines on the dendrite network of a neuron. *Applied Statistics* (Journal of the Royal Statistical Society, Series C), **63**, 673–694.

See Also

`linearpcfcross`, `linearpcf`, `linearmarkconnect`, `markconnect`.

Examples

```
if(interactive()) {
  X <- chicago
} else {
  m <- sample(factor(c("A","B")), 20, replace=TRUE)
  X <- runiflpp(20, simplenet) %mark% m
}
p <- linearmarkequal(X)
```

`linearpcf` *Linear Pair Correlation Function*

Description

Computes an estimate of the linear pair correlation function for a point pattern on a linear network.

Usage

```
linearpcf(X, r=NULL, ..., correction="Ang", ratio=FALSE)
```

Arguments

`X`	Point pattern on linear network (object of class `"lpp"`).
`r`	Optional. Numeric vector of values of the function argument r. There is a sensible default.
`...`	Arguments passed to `density.default` to control the smoothing.
`correction`	Geometry correction. Either `"none"` or `"Ang"`. See Details.
`ratio`	Logical. If `TRUE`, the numerator and denominator of each estimate will also be saved, for use in analysing replicated point patterns.

Details

This command computes the linear pair correlation function from point pattern data on a linear network.

The pair correlation function is estimated from the shortest-path distances between each pair of data points, using the fixed-bandwidth kernel smoother `density.default`, with a bias correction at each end of the interval of r values. To switch off the bias correction, set `endcorrect=FALSE`.

The bandwidth for smoothing the pairwise distances is determined by arguments `...` passed to `density.default`, mainly the arguments `bw` and `adjust`. The default is to choose the bandwidth by Silverman's rule of thumb `bw="nrd0"` explained in `density.default`.

If `correction="none"`, the calculations do not include any correction for the geometry of the linear network. The result is an estimate of the first derivative of the network K function defined by Okabe and Yamada (2001).

If `correction="Ang"`, the pair counts are weighted using Ang's correction (Ang, 2010). The result is an estimate of the pair correlation function in the linear network.

Value

Function value table (object of class `"fv"`).

If `ratio=TRUE` then the return value also has two attributes called `"numerator"` and `"denominator"` which are `"fv"` objects containing the numerators and denominators of each estimate of $g(r)$.

Author(s)

Ang Qi Wei <aqw07398@hotmail.com> and Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>.

References

Ang, Q.W. (2010) Statistical methodology for spatial point patterns on a linear network. MSc thesis, University of Western Australia.

Ang, Q.W., Baddeley, A. and Nair, G. (2012) Geometrically corrected second-order analysis of events on a linear network, with applications to ecology and criminology. *Scandinavian Journal of Statistics* **39**, 591–617.

Okabe, A. and Yamada, I. (2001) The K-function method on a network and its computational implementation. *Geographical Analysis* **33**, 271-290.

See Also

`linearK`, `linearpcfinhom`, `lpp`

Examples

```
X <- rpoislpp(5, simplenet)
linearpcf(X)
linearpcf(X, correction="none")
```

---

linearpcfcross *Multitype Pair Correlation Function (Cross-type) for Linear Point Pattern*

Description

For a multitype point pattern on a linear network, estimate the multitype pair correlation function from points of type i to points of type j.

Usage

```
linearpcfcross(X, i, j, r=NULL, ..., correction="Ang")
```

Arguments

`X`	The observed point pattern, from which an estimate of the i-to-any pair correlation function $g_{ij}(r)$ will be computed. An object of class `"lpp"` which must be a multitype point pattern (a marked point pattern whose marks are a factor).
`i`	Number or character string identifying the type (mark value) of the points in `X` from which distances are measured. Defaults to the first level of `marks(X)`.
`j`	Number or character string identifying the type (mark value) of the points in `X` to which distances are measured. Defaults to the second level of `marks(X)`.

`r`	numeric vector. The values of the argument r at which the function $g_{ij}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
`correction`	Geometry correction. Either `"none"` or `"Ang"`. See Details.
`...`	Arguments passed to `density.default` to control the kernel smoothing.

Details

This is a counterpart of the function `pcfcross` for a point pattern on a linear network (object of class `"lpp"`).

The argument `i` will be interpreted as levels of the factor `marks(X)`. If `i` is missing, it defaults to the first level of the marks factor.

The argument `r` is the vector of values for the distance r at which $g_{ij}(r)$ should be evaluated. The values of r must be increasing nonnegative numbers and the maximum r value must not exceed the radius of the largest disc contained in the window.

Value

An object of class `"fv"` (see `fv.object`).

Warnings

The argument `i` is interpreted as a level of the factor `marks(X)`. Beware of the usual trap with factors: numerical values are not interpreted in the same way as character values.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>

References

Baddeley, A, Jammalamadaka, A. and Nair, G. (2014) Multitype point process analysis of spines on the dendrite network of a neuron. *Applied Statistics* (Journal of the Royal Statistical Society, Series C), **63**, 673–694.

See Also

`linearpcfdot`, `linearpcf`, `pcfcross`.

Examples

```
g <- linearpcfcross(chicago, "assault")
```

`linearpcfcross.inhom` *Inhomogeneous Multitype Pair Correlation Function (Cross-type) for Linear Point Pattern*

Description

For a multitype point pattern on a linear network, estimate the inhomogeneous multitype pair correlation function from points of type i to points of type j.

Usage

```
linearpcfcross.inhom(X, i, j, lambdaI, lambdaJ, r=NULL, ...,
                     correction="Ang", normalise=TRUE,
                     sigma=NULL, adjust.sigma=1,
                     bw="nrd0", adjust.bw=1)
```

Arguments

`X`	The observed point pattern, from which an estimate of the i-to-any pair correlation function $g_{ij}(r)$ will be computed. An object of class `"lpp"` which must be a multitype point pattern (a marked point pattern whose marks are a factor).
`i`	Number or character string identifying the type (mark value) of the points in `X` from which distances are measured. Defaults to the first level of `marks(X)`.
`j`	Number or character string identifying the type (mark value) of the points in `X` to which distances are measured. Defaults to the second level of `marks(X)`.
`lambdaI`	Intensity values for the points of type `i`. Either a numeric vector, a `function`, a pixel image (object of class `"im"` or `"linim"`) or a fitted point process model (object of class `"ppm"` or `"lppm"`).
`lambdaJ`	Intensity values for the points of type `j`. Either a numeric vector, a `function`, a pixel image (object of class `"im"` or `"linim"`) or a fitted point process model (object of class `"ppm"` or `"lppm"`).
`r`	numeric vector. The values of the argument r at which the function $g_{ij}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
`correction`	Geometry correction. Either `"none"` or `"Ang"`. See Details.
`...`	Arguments passed to `density.default` to control the kernel smoothing.
`normalise`	Logical. If `TRUE` (the default), the denominator of the estimator is data-dependent (equal to the sum of the reciprocal intensities at the points of type `i`), which reduces the sampling variability. If `FALSE`, the denominator is the length of the network.
`sigma`	Smoothing bandwidth passed to `density.lpp` for estimation of intensities when either `lambdaI` or `lambdaJ` is `NULL`.
`adjust.sigma`	Numeric value. `sigma` will be multiplied by this value.

`bw`	Smoothing bandwidth (passed to `density.default`) for one-dimensional kernel smoothing of the pair correlation function. Either a numeric value, or a character string recognised by `density.default`.
`adjust.bw`	Numeric value. `bw` will be multiplied by this value.

Details

This is a counterpart of the function `pcfcross.inhom` for a point pattern on a linear network (object of class `"lpp"`).

The argument `i` will be interpreted as levels of the factor `marks(X)`. If `i` is missing, it defaults to the first level of the marks factor.

The argument `r` is the vector of values for the distance r at which $g_{ij}(r)$ should be evaluated. The values of r must be increasing nonnegative numbers and the maximum r value must not exceed the radius of the largest disc contained in the window.

If `lambdaI` or `lambdaJ` is missing or `NULL`, it will be estimated by kernel smoothing using `density.lpp`.

If `lambdaI` or `lambdaJ` is a fitted point process model, the default behaviour is to update the model by re-fitting it to the data, before computing the fitted intensity. This can be disabled by setting `update=FALSE`.

Value

An object of class `"fv"` (see `fv.object`).

Warnings

The argument `i` is interpreted as a level of the factor `marks(X)`. Beware of the usual trap with factors: numerical values are not interpreted in the same way as character values.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

References

Baddeley, A, Jammalamadaka, A. and Nair, G. (2014) Multitype point process analysis of spines on the dendrite network of a neuron. *Applied Statistics* (Journal of the Royal Statistical Society, Series C), **63**, 673–694.

See Also

`linearpcfdot`, `linearpcf`, `pcfcross.inhom`.

Examples

```
lam <- table(marks(chicago))/(summary(chicago)$totlength)
lamI <- function(x,y,const=lam[["assault"]]){ rep(const, length(x)) }
lamJ <- function(x,y,const=lam[["robbery"]]){ rep(const, length(x)) }

g <- linearpcfcross.inhom(chicago, "assault", "robbery", lamI, lamJ)
```

```
# using fitted models for intensity
# fit <- lppm(chicago ~marks + x)
# linearpcfcross.inhom(chicago, "assault", "robbery", fit, fit)
```

`linearpcfdot` *Multitype Pair Correlation Function (Dot-type) for Linear Point Pattern*

Description

For a multitype point pattern on a linear network, estimate the multitype pair correlation function from points of type i to points of any type.

Usage

```
linearpcfdot(X, i, r=NULL, ..., correction="Ang")
```

Arguments

`X`	The observed point pattern, from which an estimate of the i-to-any pair correlation function $g_{i\bullet}(r)$ will be computed. An object of class `"lpp"` which must be a multitype point pattern (a marked point pattern whose marks are a factor).
`i`	Number or character string identifying the type (mark value) of the points in `X` from which distances are measured. Defaults to the first level of `marks(X)`.
`r`	numeric vector. The values of the argument r at which the function $g_{i\bullet}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
`correction`	Geometry correction. Either `"none"` or `"Ang"`. See Details.
`...`	Arguments passed to `density.default` to control the kernel smoothing.

Details

This is a counterpart of the function `pcfdot` for a point pattern on a linear network (object of class `"lpp"`).

The argument `i` will be interpreted as levels of the factor `marks(X)`. If `i` is missing, it defaults to the first level of the marks factor.

The argument `r` is the vector of values for the distance r at which $g_{i\bullet}(r)$ should be evaluated. The values of r must be increasing nonnegative numbers and the maximum r value must not exceed the radius of the largest disc contained in the window.

Value

An object of class `"fv"` (see `fv.object`).

Warnings

The argument `i` is interpreted as a level of the factor `marks(X)`. Beware of the usual trap with factors: numerical values are not interpreted in the same way as character values.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

References

Baddeley, A, Jammalamadaka, A. and Nair, G. (2014) Multitype point process analysis of spines on the dendrite network of a neuron. *Applied Statistics* (Journal of the Royal Statistical Society, Series C), **63**, 673–694.

See Also

`linearpcfcross`, `linearpcf`, `pcfdot`.

Examples

```
g <- linearpcfdot(chicago, "assault")
```

`linearpcfdot.inhom` *Inhomogeneous Multitype Pair Correlation Function (Dot-type) for Linear Point Pattern*

Description

For a multitype point pattern on a linear network, estimate the inhomogeneous multitype pair correlation function from points of type i to points of any type.

Usage

```
linearpcfdot.inhom(X, i, lambdaI, lambdadot, r=NULL, ...,
                   correction="Ang", normalise=TRUE,
                   sigma=NULL, adjust.sigma=1,
                   bw="nrd0", adjust.bw=1)
```

Arguments

`X`	The observed point pattern, from which an estimate of the i-to-any pair correlation function $g_{i\bullet}(r)$ will be computed. An object of class `"lpp"` which must be a multitype point pattern (a marked point pattern whose marks are a factor).
`i`	Number or character string identifying the type (mark value) of the points in `X` from which distances are measured. Defaults to the first level of `marks(X)`.
`lambdaI`	Intensity values for the points of type `i`. Either a numeric vector, a `function`, a pixel image (object of class `"im"` or `"linim"`) or a fitted point process model (object of class `"ppm"` or `"lppm"`) or `NULL`.

`lambdadot`	Intensity values for all points of `X`. Either a numeric vector, a `function`, a pixel image (object of class `"im"` or `"linim"`) or a fitted point process model (object of class `"ppm"` or `"lppm"`) or `NULL`.
`r`	numeric vector. The values of the argument r at which the function $g_{i\bullet}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
`correction`	Geometry correction. Either `"none"` or `"Ang"`. See Details.
`...`	Arguments passed to `density.default` to control the kernel smoothing.
`normalise`	Logical. If `TRUE` (the default), the denominator of the estimator is data-dependent (equal to the sum of the reciprocal intensities at the points of type `i`), which reduces the sampling variability. If `FALSE`, the denominator is the length of the network.
`sigma`	Smoothing bandwidth passed to `density.lpp` for estimation of intensities when either `lambdaI` or `lambdadot` is `NULL`.
`adjust.sigma`	Numeric value. `sigma` will be multiplied by this value.
`bw`	Smoothing bandwidth (passed to `density.default`) for one-dimensional kernel smoothing of the pair correlation function. Either a numeric value, or a character string recognised by `density.default`.
`adjust.bw`	Numeric value. `bw` will be multiplied by this value.

Details

This is a counterpart of the function `pcfdot.inhom` for a point pattern on a linear network (object of class `"lpp"`).

The argument `i` will be interpreted as levels of the factor `marks(X)`. If `i` is missing, it defaults to the first level of the marks factor.

The argument `r` is the vector of values for the distance r at which $g_{i\bullet}(r)$ should be evaluated. The values of r must be increasing nonnegative numbers and the maximum r value must not exceed the radius of the largest disc contained in the window.

If `lambdaI` or `lambdadot` is missing or `NULL`, it will be estimated by kernel smoothing using `density.lpp`.

If `lambdaI` or `lambdadot` is a fitted point process model, the default behaviour is to update the model by re-fitting it to the data, before computing the fitted intensity. This can be disabled by setting `update=FALSE`.

Value

An object of class `"fv"` (see `fv.object`).

Warnings

The argument `i` is interpreted as a level of the factor `marks(X)`. Beware of the usual trap with factors: numerical values are not interpreted in the same way as character values.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>

References

Baddeley, A, Jammalamadaka, A. and Nair, G. (2014) Multitype point process analysis of spines on the dendrite network of a neuron. *Applied Statistics* (Journal of the Royal Statistical Society, Series C), **63**, 673–694.

See Also

`linearpcfcross.inhom`, `linearpcfdot`, `pcfdot.inhom`.

Examples

```
lam <- table(marks(chicago))/(summary(chicago)$totlength)
lamI <- function(x,y,const=lam[["assault"]]){ rep(const, length(x)) }
lam. <- function(x,y,const=sum(lam)){ rep(const, length(x)) }

g <- linearpcfdot.inhom(chicago, "assault", lamI, lam.)

# using fitted models for the intensity
# fit <- lppm(chicago, ~marks + x)
# linearpcfdot.inhom(chicago, "assault", fit, fit)
```

`linearpcfEuclid` *Linear Pair Correlation Function Using Euclidean Distance*

Description

Computes an estimate of the pair correlation function based on Euclidean distances, for a point pattern on a linear network.

Usage

```
linearpcfEuclid(X, r = NULL, ...)
```

Arguments

`X`	Point pattern on linear network (object of class `"lpp"`).
`r`	Optional. Numeric vector of values of the function argument r. There is a sensible default.
`...`	Ignored.

Details

This command computes an estimate of the pair correlation function based on Euclidean distances between the points, as described by Rakshit, Nair and Baddeley (2017).

This is different from the linear pair correlation function based on shortest-path distances, which is computed by `linearpcf`.

The linear pair correlation function based on Euclidean distances is defined in equation (15) of Rakshit, Nair and Baddeley (2017). The estimate is computed from the point pattern as described in equation (31).

Value

Function value table (object of class `"fv"`).

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>.

References

Rakshit. S., Nair, G. and Baddeley, A. (2017) Second-order analysis of point patterns on a network using any distance metric. *Spatial Statistics* **22** (1) 129–154.

See Also

`linearKEuclid`, `linearpcfEuclidInhom`.

See `linearpcf` for the corresponding function based on shortest-path distances.

Examples

```
X <- rpoislpp(5, simplenet)
g <- linearpcfEuclid(X)
```

`linearpcfEuclidInhom` *Inhomogeneous Linear Pair Correlation Function Based on Euclidean Distances*

Description

Computes an estimate of the inhomogeneous pair correlation function based on Euclidean distances, for a point pattern on a linear network.

Usage

```
linearpcfEuclidInhom(X, lambda = NULL, r = NULL, ...,
   normalise = TRUE, normpower = 2,
   update = TRUE, leaveoneout = TRUE,
   sigma=NULL, adjust.sigma=1, bw="nrd0", adjust.bw=1)
```

Arguments

`X`	Point pattern on linear network (object of class `"lpp"`).
`lambda`	Intensity values for the point pattern. Either a numeric vector, a `function`, a pixel image (object of class `"im"` or `"linim"`) or a fitted point process model (object of class `"ppm"` or `"lppm"`) or `NULL`.
`r`	Optional. Numeric vector of values of the function argument r. There is a sensible default.
`...`	Ignored.

`normalise`	Logical. If `TRUE` (the default), the denominator of the estimator is data-dependent (equal to the sum of the reciprocal intensities at the data points, raised to `normpower`), which reduces the sampling variability. If `FALSE`, the denominator is the length of the network.
`normpower`	Integer (usually either 1 or 2). Normalisation power. See Details.
`update`	Logical value indicating what to do when `lambda` is a fitted model (class `"lppm"` or `"ppm"`). If `update=TRUE` (the default), the model will first be refitted to the data `X` (using `update.lppm` or `update.ppm`) before the fitted intensity is computed. If `update=FALSE`, the fitted intensity of the model will be computed without re-fitting it to `X`.
`leaveoneout`	Logical value specifying whether to use a leave-one-out rule when calculating the intensity. See Details.
`sigma`	Smoothing bandwidth (passed to `density.lpp`) for kernel density estimation of the intensity when `lambda=NULL`.
`adjust.sigma`	Numeric value. `sigma` will be multiplied by this value.
`bw`	Smoothing bandwidth (passed to `density.default`) for one-dimensional kernel smoothing of the pair correlation function. Either a numeric value, or a character string recognised by `density.default`.
`adjust.bw`	Numeric value. `bw` will be multiplied by this value.

Details

This command computes the inhomogeneous version of the pair correlation function based on *Euclidean* distances, for a point pattern on a linear network.

This is different from the inhomogeneous pair correlation function based on *shortest-path* distances, which is computed by `linearpcfinhom`.

The inhomogeneous pair correlation function based on *Euclidean* distances is defined in equation (30) of Rakshit, Nair and Baddeley (2017). Estimation is performed as described in equation (34) of Rakshit, Nair and Baddeley (2017).

The argument `lambda` should provide estimated values of the intensity of the point process at each point of `X`.

If `lambda=NULL`, the intensity will be estimated by kernel smoothing by calling `density.lpp` with the smoothing bandwidth `sigma`, and with any other relevant arguments that might be present in `...`. A leave-one-out kernel estimate will be computed if `leaveoneout=TRUE`.

If `lambda` is given, then it may be a numeric vector (of length equal to the number of points in `X`), or a `function(x,y)` that will be evaluated at the points of `X` to yield numeric values, or a pixel image (object of class `"im"`) or a fitted point process model (object of class `"ppm"` or `"lppm"`).

If `lambda` is a fitted point process model, the default behaviour is to update the model by re-fitting it to the data, before computing the fitted intensity. This can be disabled by setting `update=FALSE`. The intensity at data points will be computed by `fitted.lppm` or `fitted.ppm`. A leave-one-out estimate will be computed if `leaveoneout=TRUE` and `update=TRUE`.

If `normalise=TRUE` (the default), then the estimate is multiplied by $c^{\mathrm{normpower}}$ where $c = \mathrm{length}(L)/\sum(1/\lambda(x_i))$. This rescaling reduces the variability and bias of the estimate in small samples and in cases of very strong inhomogeneity. The default value of `normpower` is 1 (for consistency with previous versions of **spatstat**) but the most sensible value is 2, which would correspond to rescaling the `lambda` values so that $\sum(1/\lambda(x_i)) = \mathrm{area}(W)$.

Value

Function value table (object of class `"fv"`).

Warning

Older versions of `linearpcfEuclidInhom` interpreted `lambda=NULL` to mean that the homogeneous function `linearpcfEuclid` should be computed. This was changed to the current behaviour in version `3.1-0` of **spatstat.linnet**.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>.

References

Rakshit. S., Nair, G. and Baddeley, A. (2017) Second-order analysis of point patterns on a network using any distance metric. *Spatial Statistics* **22** (1) 129–154.

See Also

`linearKEuclidInhom`, `linearpcfEuclid`.

See `linearpcfinhom` for the corresponding function based on shortest-path distances.

Examples

```
X <- rpoislpp(5, simplenet)
fit <- lppm(X ~x)
g <- linearpcfEuclidInhom(X, lambda=fit)
plot(g)
```

`linearpcfinhom` *Inhomogeneous Linear Pair Correlation Function*

Description

Computes an estimate of the inhomogeneous linear pair correlation function for a point pattern on a linear network.

Usage

```
linearpcfinhom(X, lambda=NULL, r=NULL, ..., correction="Ang",
               normalise=TRUE, normpower=1,
       update = TRUE, leaveoneout = TRUE,
               sigma=NULL, adjust.sigma=1,
               bw="nrd0", adjust.bw=1,
       ratio = FALSE)
```

Arguments

`X`	Point pattern on linear network (object of class `"lpp"`).
`lambda`	Intensity values for the point pattern. Either a numeric vector, a `function`, a pixel image (object of class `"im"`) or a fitted point process model (object of class `"ppm"` or `"lppm"`).
`r`	Optional. Numeric vector of values of the function argument r. There is a sensible default.
`...`	Arguments passed to `density.default` to control the smoothing of the estimates of pair correlation.
`correction`	Geometry correction. Either `"none"` or `"Ang"`. See Details.
`normalise`	Logical. If `TRUE` (the default), the denominator of the estimator is data-dependent (equal to the sum of the reciprocal intensities at the data points, raised to `normpower`), which reduces the sampling variability. If `FALSE`, the denominator is the length of the network.
`normpower`	Integer (usually either 1 or 2). Normalisation power. See explanation in `linearKinhom`.
`update`	Logical value indicating what to do when `lambda` is a fitted model (class `"lppm"` or `"ppm"`). If `update=TRUE` (the default), the model will first be refitted to the data `X` (using `update.lppm` or `update.ppm`) before the fitted intensity is computed. If `update=FALSE`, the fitted intensity of the model will be computed without re-fitting it to `X`.
`leaveoneout`	Logical value specifying whether to use a leave-one-out rule when calculating the intensity. See Details.
`sigma`	Smoothing bandwidth (passed to `density.lpp`) for kernel density estimation of the intensity when `lambda=NULL`.
`adjust.sigma`	Numeric value. `sigma` will be multiplied by this value.
`bw`	Smoothing bandwidth (passed to `density.default`) for one-dimensional kernel smoothing of the pair correlation function. Either a numeric value, or a character string recognised by `density.default`.
`adjust.bw`	Numeric value. `bw` will be multiplied by this value.
`ratio`	Logical. If `TRUE`, the numerator and denominator of each estimate will also be saved, for use in analysing replicated point patterns.

Details

This command computes the inhomogeneous version of the linear pair correlation function from point pattern data on a linear network.

The argument `lambda` should provide estimated values of the intensity of the point process at each point of `X`.

If `lambda=NULL`, the intensity will be estimated by kernel smoothing by calling `density.lpp` with the smoothing bandwidth `sigma`, and with any other relevant arguments that might be present in `...`. A leave-one-out kernel estimate will be computed if `leaveoneout=TRUE`.

If `lambda` is given, it may be a numeric vector (of length equal to the number of points in `X`), or a `function(x,y)` that will be evaluated at the points of `X` to yield numeric values, or a pixel image (object of class `"im"`) or a fitted point process model (object of class `"ppm"` or `"lppm"`).

If `lambda` is a fitted point process model, the default behaviour is to update the model by re-fitting it to the data, before computing the fitted intensity. This can be disabled by setting `update=FALSE`. The intensity at data points will be computed by `fitted.lppm` or `fitted.ppm`. A leave-one-out estimate will be computed if `leaveoneout=TRUE` and `update=TRUE`.

If `correction="none"`, the calculations do not include any correction for the geometry of the linear network. If `correction="Ang"`, the pair counts are weighted using Ang's correction (Ang, 2010).

The bandwidth for smoothing the pairwise distances is determined by arguments `...` passed to `density.default`, mainly the arguments `bw` and `adjust`. The default is to choose the bandwidth by Silverman's rule of thumb `bw="nrd0"` explained in `density.default`.

Value

Function value table (object of class `"fv"`).

If `ratio=TRUE` then the return value also has two attributes called `"numerator"` and `"denominator"` which are `"fv"` objects containing the numerators and denominators of each estimate of $g(r)$.

Warning

Older versions of `linearpcfinhom` interpreted `lambda=NULL` to mean that the homogeneous function `linearpcf` should be computed. This was changed to the current behaviour in version `3.1-0` of **spatstat.linnet**.

Author(s)

Ang Qi Wei <`aqw07398@hotmail.com`> and Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>.

References

Ang, Q.W. (2010) Statistical methodology for spatial point patterns on a linear network. MSc thesis, University of Western Australia.

Ang, Q.W., Baddeley, A. and Nair, G. (2012) Geometrically corrected second-order analysis of events on a linear network, with applications to ecology and criminology. *Scandinavian Journal of Statistics* **39**, 591–617.

Okabe, A. and Yamada, I. (2001) The K-function method on a network and its computational implementation. *Geographical Analysis* **33**, 271-290.

See Also

`linearpcf`, `linearKinhom`, `lpp`

Examples

```
X <- rpoislpp(5, simplenet)
fit <- lppm(X ~x)
g <- linearpcfinhom(X, lambda=fit, update=FALSE)
plot(g)
ge <- linearpcfinhom(X, sigma=bw.lppl)
```

`lineartileindex` *Determine Which Tile Contains Each Given Point on a Linear Network*

Description

Given a tessellation on a linear network, and a list of points on the network, determine which tile of the tessellation contains each of the given points.

Usage

```
lineartileindex(seg, tp, Z, method = c("encode", "C", "interpreted"))
```

Arguments

`seg, tp`	Vectors of local coordinates of the query points. See Details.
`Z`	A tessellation on a linear network (object of class `"lintess"`).
`method`	Internal use only.

Details

This low-level function is the analogue of `tileindex` for linear networks. For a tessellation `Z` on a linear network, and a list of query points on the same network, the function determines which tile of the tessellation contains each query point.

Argument `Z` should be a tessellation on a linear network (object of class `"lintess"`).

The vectors `seg` and `tp` specify the locations of the query points, on the same network, using local coordinates: `seg` contains integer values specifying which segment of the network contains each query point; `tp` contains numeric values between 0 and 1 specifying the fractional position along that segment.

The result is a factor, of the same length as `seg` and `tp`, indicating which tile contains each point. The levels of the factor are the names of the tiles of `Z`.

Value

A factor, of the same length as `seg` and `tp`, whose levels are the names of the tiles of `Z`.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`, Rolf Turner `<rolfturner@posteo.net>` and Ege Rubak `<rubak@math.aau.dk>`

See Also

`lintess`.

`as.linfun.lintess` to create a function whose value is the tile index.

`cut.lpp` for a neater way to classify the points of a point pattern on a linear network according to a tessellation on the network.

Examples

```
Z <- lineardirichlet(runiflpp(15, simplenet))
X <- runiflpp(10, simplenet)
coX <- coords(X)
ii <- lineartileindex(coX$seg, coX$tp, Z)
```

`linequad` *Quadrature Scheme on a Linear Network*

Description

Generates a quadrature scheme (an object of class `"quad"`) on a linear network.

Usage

```
linequad(X, Y, ..., eps = NULL, nd = 1000, random = FALSE)
```

Arguments

`X`	Data points. An object of class `"lpp"` or `"ppp"`.
`Y`	Line segments on which the points of `X` lie. An object of class `"psp"`. Required only when `X` is a `"ppp"` object.
`...`	Ignored.
`eps`	Optional. Spacing between successive dummy points along each segment. (This is the maximum spacing; some spacings will be shorter.)
`nd`	Optional. Total number of dummy points to be generated. (Actual number may be larger.)
`random`	Logical value indicating whether the sequence of dummy points should start at a randomly-chosen position along each segment.

Details

This command generates a quadrature scheme (object of class `"quad"`) from a pattern of points on a linear network.

Normally the user does not need to call `linequad` explicitly. It is invoked by **spatstat** functions when needed. A quadrature scheme is required by `lppm` in order to fit point process models to point pattern data on a linear network. A quadrature scheme is also used by `rhohat.lpp` and other functions.

In order to create the quadrature scheme, dummy points are placed along each line segment of the network. The dummy points are evenly-spaced with spacing `eps`. The default is `eps = totlen/nd` where `totlen` is the total length of all line segments in the network.

Every line segment of the network will contain at least one dummy point. Consequently the actual number of dummy points generated will typically be greater than `nd`, especially when `nd` is small. If `eps` is specified, the number of dummy points will be greater than `totlen/eps`, especially when `eps` is large.

Value

A quadrature scheme (object of class `"quad"`).

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`, Greg McSwiggan and Suman Rakshit.

See Also

`lppm`

`linfun` *Function on a Linear Network*

Description

Create a function on a linear network.

Usage

```
linfun(f, L)
```

Arguments

`f`	A `function` in the R language.
`L`	A linear network (object of class `"linnet"`) on which `f` is defined.

Details

This creates an object of class `"linfun"`. This is a simple mechanism for handling a function defined on a linear network, to make it easier to display and manipulate.

`f` should be a `function` in the R language, with formal arguments `x,y,seg,tp` (and optional additional arguments) where `x,y` are Cartesian coordinates of locations on the linear network, `seg, tp` are the local coordinates.

The function `f` should be vectorised: that is, if `x,y,seg,tp` are numeric vectors of the same length `n`, then `v <- f(x,y,seg,tp)` should be a vector of length `n`.

`L` should be a linear network (object of class `"linnet"`) on which the function `f` is well-defined.

The result is a function `g` in the R language which belongs to the special class `"linfun"`. There are several methods for this class including `print`, `plot` and `as.linim`.

This function can be called as `g(X)` where `X` is an `"lpp"` object, or called as `g(x,y)` or `g(x,y,seg,tp)` where `x,y,seg,tp` are coordinates. If the original function `f` had additional arguments, then these may be included in the call to `g`, and will be passed to `f`.

Value

A function in the R\ language. It also belongs to the class `"linfun"` which has methods for `plot`, `print` etc.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`> and Rolf Turner <`rolfturner@posteo.net`>

See Also

`methods.linfun` for methods applicable to `"linfun"` objects.

`distfun.lpp`, `nnfun.lpp`.

Examples

```
f <- function(x,y,seg,tp) { x+y }
g <- linfun(f, simplenet)
plot(g)
X <- runiflpp(3, simplenet)
g(X)
Z <- as.linim(g)

f <- function(x,y,seg,tp, mul=1) { mul*(x+y) }
g <- linfun(f, simplenet)
plot(g)
plot(g, mul=10)
g(X, mul=10)
Z <- as.linim(g, mul=10)
```

`linim` *Create Pixel Image on Linear Network*

Description

Creates an object of class `"linim"` that represents a pixel image on a linear network.

Usage

```
linim(L, Z, ..., restrict=TRUE, df=NULL)
```

Arguments

`L`	Linear network (object of class `"linnet"`).
`Z`	Pixel image (object of class `"im"`).
`...`	Ignored.
`restrict`	Advanced use only. Logical value indicating whether to ensure that all pixels in `Z` which do not lie on the network `L` have pixel value `NA`. This condition must be satisfied, but if you set `restrict=FALSE` it will not be checked, and the code will run faster.
`df`	Advanced use only. Data frame giving full details of the mapping between the pixels of `Z` and the lines of `L`. See Details.

Details

This command creates an object of class `"linim"` that represents a pixel image defined on a linear network. Typically such objects are used to represent the result of smoothing or model-fitting on the network. Most users will not need to call `linim` directly.

The argument `L` is a linear network (object of class `"linnet"`). It gives the exact spatial locations of the line segments of the network, and their connectivity.

The argument `Z` is a pixel image object of class `"im"` that gives a pixellated approximation of the function values.

For increased efficiency, advanced users may specify the optional argument `df`. This is a data frame giving the precomputed mapping between the pixels of `Z` and the line segments of `L`. It should have columns named `xc, yc` containing the coordinates of the pixel centres, `x,y` containing the projections of these pixel centres onto the linear network, `mapXY` identifying the line segment on which each projected point lies, and `tp` giving the parametric position of `(x,y)` along the segment.

Value

Object of class `"linim"` that also inherits the class `"im"`. There is a special method for plotting this class.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>

References

Ang, Q.W. (2010) *Statistical methodology for events on a network.* Master's thesis, School of Mathematics and Statistics, University of Western Australia.

Ang, Q.W., Baddeley, A. and Nair, G. (2012) Geometrically corrected second-order analysis of events on a linear network, with applications to ecology and criminology. *Scandinavian Journal of Statistics* **39**, 591–617.

McSwiggan, G., Nair, M.G. and Baddeley, A. (2012) Fitting Poisson point process models to events on a linear network. Manuscript in preparation.

See Also

`plot.linim`, `linnet`, `eval.linim`, `Math.linim`, `im`.

Examples

```
Z <- as.im(function(x,y) {x-y}, Frame(simplenet))
X <- linim(simplenet, Z)
X
```

`linnet` *Create a Linear Network*

Description

Creates an object of class `"linnet"` representing a network of line segments.

Usage

```
linnet(vertices, m, edges, sparse=FALSE, warn=TRUE)
```

Arguments

`vertices`	Point pattern (object of class `"ppp"`) specifying the vertices of the network.
`m`	Adjacency matrix. A matrix or sparse matrix of logical values equal to `TRUE` when the corresponding vertices are joined by a line. (Specify either `m` or `edges`.)
`edges`	Edge list. A two-column matrix of integers, specifying all pairs of vertices that should be joined by an edge. (Specify either `m` or `edges`.)
`sparse`	Optional. Logical value indicating whether to use a sparse matrix representation of the network. See Details.
`warn`	Logical value indicating whether to issue a warning if the resulting network is not connected.

Details

An object of class `"linnet"` represents a network of straight line segments in two dimensions. The function `linnet` creates such an object from the minimal information: the spatial location of each vertex (endpoint, crossing point or meeting point of lines) and information about which vertices are joined by an edge.

If `sparse=FALSE` (the default), the algorithm will compute and store various properties of the network, including the adjacency matrix `m` and a matrix giving the shortest-path distances between each pair of vertices in the network. This is more efficient for small datasets. However it can require large amounts of memory and can take a long time to execute.

If `sparse=TRUE`, then the shortest-path distances will not be computed, and the network adjacency matrix `m` will be stored as a sparse matrix. This saves a lot of time and memory when creating the linear network.

If the argument `edges` is given, then it will also determine the *ordering* of the line segments when they are stored or extracted. For example, `edges[i,]` corresponds to `as.psp(L)[i]`.

Value

Object of class `"linnet"` representing the linear network.

Author(s)

Ang Qi Wei `<aqw07398@hotmail.com>` and Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

See Also

`simplenet` for an example of a linear network.

`methods.linnet` for methods applicable to `linnet` objects.

Special tools: `thinNetwork`, `insertVertices`, `joinVertices`, `connected.linnet`, `lixellate`.

`delaunayNetwork` for the Delaunay triangulation as a network.

`ppp`, `psp`.

Examples

```
# letter 'A' specified by adjacency matrix
v <- ppp(x=(-2):2, y=3*c(0,1,2,1,0), c(-3,3), c(-1,7))
m <- matrix(FALSE, 5,5)
for(i in 1:4) m[i,i+1] <- TRUE
m[2,4] <- TRUE
m <- m | t(m)
letterA <- linnet(v, m)
plot(letterA)

# letter 'A' specified by edge list
edg <- cbind(1:4, 2:5)
edg <- rbind(edg, c(2,4))
letterA <- linnet(v, edges=edg)
```

`lintess` *Tessellation on a Linear Network*

Description

Create a tessellation on a linear network.

Usage

```
lintess(L, df, marks=NULL)
```

Arguments

`L`	Linear network (object of class `"linnet"`).
`df`	Data frame of local coordinates for the pieces that make up the tiles of the tessellation. See Details.
`marks`	Vector or data frame of marks associated with the tiles of the tessellation.

Details

A tessellation on a linear network `L` is a partition of the network into non-overlapping pieces (tiles). Each tile consists of one or more line segments which are subsets of the line segments making up the network. A tile can consist of several disjoint pieces.

The data frame `df` should have columns named `seg`, `t0`, `t1` and `tile`. Any additional columns will be ignored.

Each row of the data frame specifies one sub-segment of the network and allocates it to a particular tile.

The `seg` column specifies which line segment of the network contains the sub-segment. Values of `seg` are integer indices for the segments in `as.psp(L)`.

The `t0` and `t1` columns specify the start and end points of the sub-segment. They should be numeric values between 0 and 1 inclusive, where the values 0 and 1 representing the network vertices that are joined by this network segment.

The `tile` column specifies which tile of the tessellation includes this sub-segment. It will be coerced to a factor and its levels will be the names of the tiles.

If `df` is missing or `NULL`, the result is a tessellation with only one tile, consisting of the entire network `L`.

Additional data called *marks* may be associated with each tile of the tessellation. The argument `marks` should be a vector with one entry for each tile (that is, one entry for each level of `df$tile`) or a data frame with one row for each tile. In general `df` and `marks` will have different numbers of rows.

Value

An object of class `"lintess"`. There are methods for `print`, `plot` and `summary` for this object.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`> and Greg McSwiggan.

See Also

`linnet` for linear networks.

`plot.lintess` for plotting.

`divide.linnet` to make a tessellation demarcated by given points.

`chop.linnet` to make a tessellation demarcated by infinite lines.

`lineardirichlet` to create the Dirichlet-Voronoi tessellation from a point pattern on a linear network.

`as.linfun.lintess`, `as.linnet.lintess` and `as.linim` to convert to other classes.

`tile.lengths` to compute the length of each tile in the tessellation.

The undocumented methods `Window.lintess` and `as.owin.lintess` extract the spatial window.

Examples

```
# tessellation consisting of one tile for each existing segment
ns <- nsegments(simplenet)
df <- data.frame(seg=1:ns, t0=0, t1=1, tile=letters[1:ns])
u <- lintess(simplenet, df)
u
plot(u)
S <- as.psp(simplenet)
marks(u) <- data.frame(len=lengths_psp(S), ang=angles.psp(S))
u
plot(u)
```

`lixellate` *Subdivide Segments of a Network*

Description

Each line segment of a linear network will be divided into several shorter segments (line elements or lixels).

Usage

```
lixellate(X, ..., nsplit, eps, sparse = TRUE)
```

Arguments

`X`	A linear network (object of class `"linnet"`) or a point pattern on a linear network (object of class `"lpp"`).
`...`	Ignored.
`nsplit`	Number of pieces into which *each* line segment of `X` should be divided. Either a single integer, or an integer vector with one entry for each line segment in `X`. Incompatible with `eps`.
`eps`	Maximum length of the resulting pieces of line segment. A single numeric value. Incompatible with `nsplit`.
`sparse`	Optional. Logical value specifying whether the resulting linear network should be represented using a sparse matrix. If `sparse=NULL`, then the representation will be the same as in `X`.

Details

Each line segment in `X` will be subdivided into equal pieces. The result is an object of the same kind as `X`, representing the same data as `X` except that the segments have been subdivided.

Splitting is controlled by the arguments `nsplit` and `eps`, exactly one of which should be given.

If `nsplit` is given, it specifies the number of pieces into which *each* line segment of `X` should be divided. It should be either a single integer, or an integer vector of length equal to the number of line segments in `X`.

If eps is given, it specifies the maximum length of any resulting piece of line segment.

It is strongly advisable to use `sparse=TRUE` (the default) to limit the computation time.

If `X` is a point pattern (class `"lpp"`) then the spatial coordinates and marks of each data point are unchanged, but the local coordinates will change, because they are adjusted to map them to the new subdivided network.

Value

Object of the same kind as `X`.

Author(s)

Greg McSwiggan, Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

See Also

`linnet`, `lpp`.

Examples

```
A <- lixellate(simplenet, nsplit=4)
plot(A, main="lixellate(simplenet, nsplit=4)")
points(vertices(A), pch=16)

spiders
lixellate(spiders, nsplit=3)
```

`lpp` *Create Point Pattern on Linear Network*

Description

Creates an object of class `"lpp"` that represents a point pattern on a linear network.

Usage

```
lpp(X, L, ...)
```

Arguments

`X`	Locations of the points. A matrix or data frame of coordinates, or a point pattern object (of class `"ppp"`) or other data acceptable to `as.ppp`.
`L`	Linear network (object of class `"linnet"`).
`...`	Ignored.

Details

This command creates an object of class `"lpp"` that represents a point pattern on a linear network.

Normally `X` is a point pattern. The points of `X` should lie on the lines of `L`.

Alternatively `X` may be a matrix or data frame containing at least two columns.

- Usually the first two columns of `X` will be interpreted as spatial coordinates, and any remaining columns as marks.
- An exception occurs if `X` is a data frame with columns named `x`, `y`, `seg` and `tp`. Then `x` and `y` will be interpreted as spatial coordinates, and `seg` and `tp` as local coordinates, with `seg` indicating which line segment of `L` the point lies on, and `tp` indicating how far along the segment the point lies (normalised to 1). Any remaining columns will be interpreted as marks.
- Another exception occurs if `X` is a data frame with columns named `seg` and `tp`. Then `seg` and `tp` will be interpreted as local coordinates, as above, and the spatial coordinates `x,y` will be computed from them. Any remaining columns will be interpreted as marks.

If `X` is missing or `NULL`, the result is an empty point pattern (i.e. containing no points).

Value

An object of class `"lpp"`. Also inherits the class `"ppx"`.

Note on changed format

The internal format of `"lpp"` objects was changed in **spatstat** version `1.28-0`. Objects in the old format are still handled correctly, but computations are faster in the new format. To convert an object `X` from the old format to the new format, use `X <- lpp(as.ppp(X), as.linnet(X))`.

Author(s)

Ang Qi Wei <`aqw07398@hotmail.com`> and Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>

See Also

Installed datasets which are `"lpp"` objects: `chicago`, `dendrite`, `spiders`.

See `as.lpp` for converting data to an `lpp` object.

See `methods.lpp` and `methods.ppx` for other methods applicable to `lpp` objects.

Calculations on an `lpp` object: `intensity.lpp`, `distfun.lpp`, `nndist.lpp`, `nnwhich.lpp`, `nncross.lpp`, `nnfun.lpp`.

Summary functions: `linearK`, `linearKinhom`, `linearpcf`, `linearKdot`, `linearKcross`, `linearmarkconnect`, etc.

Random point patterns on a linear network can be generated by `rpoislpp` or `runiflpp`.

See `linnet` for linear networks.

Examples

```
# letter 'A'
v <- ppp(x=(-2):2, y=3*c(0,1,2,1,0), c(-3,3), c(-1,7))
edg <- cbind(1:4, 2:5)
edg <- rbind(edg, c(2,4))
letterA <- linnet(v, edges=edg)

# points on letter A
xx <- list(x=c(-1.5,0,0.5,1.5), y=c(1.5,3,4.5,1.5))
X <- lpp(xx, letterA)

plot(X)
X
summary(X)

# empty pattern
lpp(L=letterA)
```

`lppm` *Fit Point Process Model to Point Pattern on Linear Network*

Description

Fit a point process model to a point pattern dataset on a linear network

Usage

```
lppm(X, ...)

## S3 method for class 'formula'
lppm(X, interaction=NULL, ..., data=NULL)

## S3 method for class 'lpp'
lppm(X, ..., eps=NULL, nd=1000, random=FALSE)
```

Arguments

`X`	Either an object of class `"lpp"` specifying a point pattern on a linear network, or a `formula` specifying the point process model.
`...`	Arguments passed to `ppm`.
`interaction`	An object of class `"interact"` describing the point process interaction structure, or `NULL` indicating that a Poisson process (stationary or nonstationary) should be fitted.
`data`	Optional. The values of spatial covariates (other than the Cartesian coordinates) required by the model. A list whose entries are images, functions, windows, tessellations or single numbers.
`eps`	Optional. Spacing between dummy points along each segment of the network.

`nd`	Optional. Total number of dummy points placed on the network. Ignored if `eps` is given.
`random`	Logical value indicating whether the grid of dummy points should be placed at a randomised starting position.

Details

This function fits a point process model to data that specify a point pattern on a linear network. It is a counterpart of the model-fitting function `ppm` designed to work with objects of class `"lpp"` instead of `"ppp"`.

The function `lppm` is generic, with methods for the classes `formula` and `lppp`.

In `lppm.lpp` the first argument `X` should be an object of class `"lpp"` (created by the command `lpp`) specifying a point pattern on a linear network.

In `lppm.formula`, the first argument is a `formula` in the R language describing the spatial trend model to be fitted. It has the general form `pattern ~ trend` where the left hand side `pattern` is usually the name of a point pattern on a linear network (object of class `"lpp"`) to which the model should be fitted, or an expression which evaluates to such a point pattern; and the right hand side `trend` is an expression specifying the spatial trend of the model.

Other arguments `...` are passed from `lppm.formula` to `lppm.lpp` and from `lppm.lpp` to `ppm`.

Value

An object of class `"lppm"` representing the fitted model. There are methods for `print`, `predict`, `coef` and similar functions.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`> and Greg McSwiggan.

References

Ang, Q.W. (2010) *Statistical methodology for events on a network*. Master's thesis, School of Mathematics and Statistics, University of Western Australia.

Ang, Q.W., Baddeley, A. and Nair, G. (2012) Geometrically corrected second-order analysis of events on a linear network, with applications to ecology and criminology. *Scandinavian Journal of Statistics* **39**, 591–617.

McSwiggan, G., Nair, M.G. and Baddeley, A. (2012) Fitting Poisson point process models to events on a linear network. Manuscript in preparation.

See Also

`methods.lppm`, `predict.lppm`, `ppm`, `lpp`.

Examples

```
X <- runiflpp(15, simplenet)
lppm(X ~1)
lppm(X ~x)
marks(X) <- factor(rep(letters[1:3], 5))
lppm(X ~ marks)
lppm(X ~ marks * x)
```

---

marks.linnet *Marks of a Network*

Description

Extract or change the marks attached to vertices or segments of a linear network.

Usage

```
## S3 method for class 'linnet'
marks(x, of=c("segments", "vertices"), ...)

## S3 replacement method for class 'linnet'
marks(x, of=c("segments", "vertices"), ...) <- value
```

Arguments

`x`	Linear network (object of class `"linnet"`).
`of`	Character string (partially matched) specifying whether the marks are attached to the vertices of the network (`of="vertices"`) or to the line segments of the network (`of="segments"`, the default).
`...`	Ignored.
`value`	Vector or data frame of mark values, or `NULL`.

Details

These functions extract or change the marks attached to the network `x`. They are methods for the generic functions `marks` and `marks<-` for the class `"linnet"` of linear networks.

A linear network may include a set of marks attached to the line segments, and a separate set of marks attached to the vertices. Each set of marks can be a vector, a factor, or a data frame.

The expression `marks(x, of)` extracts the marks from `x`. The assignment `marks(x, of) <- value` assigns new marks to the dataset `x`, and updates the dataset `x` in the current environment. The argument `of` specifies whether we are referring to the segments or the vertices.

For the assignment `marks(x, "segments") <- value`, the `value` should be a vector or factor of length equal to the number of segments in `x`, or a data frame with as many rows as there are segments in `x`. If `value` is a single value, or a data frame with one row, then it will be replicated so that the same marks will be attached to each segment. Similarly for `marks(x, "vertices") <- value` the number of marks must match the number of vertices.

To remove marks, use `marks(x, of) <- NULL`.

To extract the vertices (including their marks) as a point pattern, use `vertices(x)`. To extract the segments (including their marks) as a line segment pattern, use `as.psp(x)`.

Value

For `marks(x)`, the result is a vector, factor or data frame, containing the mark values attached to the vertices or the segments of `x`. If there are no marks, the result is `NULL`.

For `marks(x) <- value`, the result is the updated network `x` (with the side-effect that the dataset `x` is updated in the current environment).

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>, Rolf Turner <`rolfturner@posteo.net`> and Ege Rubak <`rubak@math.aau.dk`>.

See Also

`linnet`, `marks`, `marks<-`

Examples

```
L <- simplenet
marks(L, "vertices") <- letters[1:nvertices(L)]
marks(L, "segments") <- runif(nsegments(L))
L
marks(L, "v")
marks(L, "s")
```

`marks.lintess` *Marks of a Tessellation on a Network*

Description

Extract or change the marks attached to the tiles of a tessellation on a linear network.

Usage

```
## S3 method for class 'lintess'
marks(x, ...)

## S3 replacement method for class 'lintess'
marks(x, ...) <- value

## S3 method for class 'lintess'
unmark(X)
```

Arguments

`x, X`	Tessellation on a linear network (object of class `"lintess"`).
`...`	Ignored.
`value`	Vector or data frame of mark values, or `NULL`.

Details

These functions extract or change the marks attached to each of the tiles in the tessellation `x`. They are methods for the generic functions `marks`, `marks<-` and `unmark` for the class `"lintess"` of tessellations on a network.

The expression `marks(x)` extracts the marks of `x`. The assignment `marks(x) <- value` assigns new marks to the dataset `x`, and updates the dataset `x` in the current environment.

The marks can be a vector, a factor, or a data frame.

For the assignment `marks(x) <- value`, the `value` should be a vector or factor of length equal to the number of tiles in `x`, or a data frame with as many rows as there are tiles in `x`. If `value` is a single value, or a data frame with one row, then it will be replicated so that the same marks will be attached to each tile.

To remove marks, use `marks(x) <- NULL` or `unmark(x)`.

Value

For `marks(x)`, the result is a vector, factor or data frame, containing the mark values attached to the tiles of `x`. If there are no marks, the result is `NULL`.

For `unmark(x)`, the result is the tessellation without marks.

For `marks(x) <- value`, the result is the updated tessellation `x` (with the side-effect that the dataset `x` is updated in the current environment).

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>, Rolf Turner <`rolfturner@posteo.net`> and Ege Rubak <`rubak@math.aau.dk`>.

See Also

`lintess`, `marks`, `marks<-`

Examples

```
B <- lineardirichlet(runiflpp(5, simplenet))
marks(B) <- letters[1:5]
```

`Math.linim` *S3 Group Generic Methods for Images on a Linear Network*

Description

These are group generic methods for images of class `"linim"`, which allows for usual mathematical functions and operators to be applied directly to pixel images on a linear network. See Details for a list of implemented functions.

Usage

```
## S3 methods for group generics have prototypes:
Math(x, ...)
Ops(e1, e2)
Complex(z)
Summary(..., na.rm = FALSE)
```

Arguments

`x, z, e1, e2`	objects of class `"linim"`.
`...`	further arguments passed to methods.
`na.rm`	logical: should missing values be removed?

Details

An object of class `"linim"` represents a pixel image on a linear network. See `linim`.

Below is a list of mathematical functions and operators which are defined for these images. Not all functions will make sense for all types of images. For example, none of the functions in the `"Math"` group make sense for character-valued images. Note that the `"Ops"` group methods are implemented using `eval.linim`.

1. Group `"Math"`:
    - `abs`, `sign`, `sqrt`,
      `floor`, `ceiling`, `trunc`,
      `round`, `signif`
    - `exp`, `log`, `expm1`, `log1p`,
      `cos`, `sin`, `tan`,
      `cospi`, `sinpi`, `tanpi`,
      `acos`, `asin`, `atan`
      `cosh`, `sinh`, `tanh`,
      `acosh`, `asinh`, `atanh`
    - `lgamma`, `gamma`, `digamma`, `trigamma`
    - `cumsum`, `cumprod`, `cummax`, `cummin`
2. Group `"Ops"`:
    - `"+"`, `"-"`, `"*"`, `"/"`, `"^"`, `"%%"`, `"%/%"`

   - `"&"`, `"|"`, `"!"`
   - `"=="`, `"!="`, `"<"`, `"<="`, `">="`, `">"`
3. Group `"Summary"`:
   - `all`, `any`
   - `sum`, `prod`
   - `min`, `max`
   - `range`
4. Group `"Complex"`:
   - `Arg`, `Conj`, `Im`, `Mod`, `Re`

Value

The return value is another object of class `"linim"`, except in the following cases: `all` and `any` return a single logical value; `sum`, `prod`, `min` and `max` return a single numerical value; `range` returns a vector of two numerical values.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>, Rolf Turner <`rolfturner@posteo.net`> and Ege Rubak <`rubak@math.aau.dk`>.

See Also

`eval.linim` for evaluating expressions involving images.

Examples

```
fx <- function(x,y,seg,tp) { (x - y)^2 }
fL <- linfun(fx, simplenet)
Z <- as.linim(fL)
A <- Z+2
A <- -Z
A <- sqrt(Z)
A <- !(Z > 0.1)
```

`mean.linim` *Mean, Median, Quantiles of Pixel Values on a Linear Network*

Description

Calculates the mean, median, or quantiles of the pixel values in a pixel image on a linear network.

Usage

```
  ## S3 method for class 'linim'
mean(x, ...)

  ## S3 method for class 'linim'
median(x, ...)

  ## S3 method for class 'linim'
quantile(x, probs=seq(0,1,0.25), ...)

  ## S3 method for class 'linim'
quantilefun(x, ..., type=1)
```

Arguments

`x`	A pixel image on a linear network (object of class `"linim"`).
`probs`	Vector of probabilities for which quantiles should be calculated.
`...`	Arguments passed to other methods.
`type`	Integer specifying the type of quantiles, as explained in `quantile.default`. Only types 1 and 2 are currently implemented.

Details

These functions calculate the mean, median and quantiles of the pixel values in the image `x` on a linear network.

An object of class `"linim"` describes a pixel image on a linear network. See `linim`.

The functions described here are methods for the generic `mean`, `median` and `quantile` for the class `"linim"`.

Value

For `mean` and `median`, a single number. For `quantile`, a numeric vector of the same length as `probs`. For `quantilefun`, a function.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>, Rolf Turner <`rolfturner@posteo.net`> and Ege Rubak <`rubak@math.aau.dk`>.

See Also

`mean`, `median`, `quantile`,

`mean.im`, `quantile.im`, `quantilefun`

Examples

```
M <- psp2mask(as.psp(simplenet))
Z <- as.im(function(x,y) {x-y}, W=M)
X <- linim(simplenet, Z)
X
mean(X)
median(X)
quantile(X)
f <- quantilefun(X)
```

`methods.linfun` *Methods for Functions on Linear Network*

Description

Methods for the class "`linfun`" of functions on a linear network.

Usage

```
  ## S3 method for class 'linfun'
print(x, ...)

  ## S3 method for class 'linfun'
summary(object, ...)

  ## S3 method for class 'linfun'
plot(x, ..., L=NULL, main)

  ## S3 method for class 'linfun'
as.data.frame(x, ...)

  ## S3 method for class 'linfun'
as.owin(W, ...)

  ## S3 method for class 'linfun'
as.function(x, ...)
```

Arguments

`x`, `object`, `W`	A function on a linear network (object of class "`linfun`").
`L`	A linear network
`...`	Extra arguments passed to `as.linim`, `plot.linim`, `plot.im` or `print.default`, or arguments passed to `x` if it is a function.
`main`	Main title for plot.

Details

These are methods for the generic functions `plot`, `print`, `summary` `as.data.frame` and `as.function`, and for the **spatstat** generic function `as.owin`.

An object of class `"linfun"` represents a mathematical function that could be evaluated at any location on a linear network. It is essentially an R `function` with some extra attributes.

The method `as.owin.linfun` extracts the two-dimensional spatial window containing the linear network.

The method `plot.linfun` first converts the function to a pixel image using `as.linim.linfun`, then plots the image using `plot.linim`.

Note that a `linfun` function may have additional arguments, other than those which specify the location on the network (see `linfun`). These additional arguments may be passed to `plot.linfun`.

Value

For `print.linfun` and `summary.linfun` the result is `NULL`.

For `plot.linfun` the result is the same as for `plot.linim`.

For the conversion methods, the result is an object of the required type: `as.owin.linfun` returns an object of class `"owin"`, and so on.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>, Rolf Turner <`rolfturner@posteo.net`> and Ege Rubak <`rubak@math.aau.dk`>

Examples

```
X <- runiflpp(3, simplenet)
f <- nnfun(X)
f
plot(f)
as.function(f)
as.owin(f)
head(as.data.frame(f))
```

`methods.linim` *Methods for Images on a Linear Network*

Description

Methods for the class `"linim"` of functions on a linear network.

Usage

```
  ## S3 method for class 'linim'
print(x, ...)

  ## S3 method for class 'linim'
summary(object, ...)

  ## S3 method for class 'linim'
as.im(X, ...)

  ## S3 method for class 'linim'
as.data.frame(x, ...)

  ## S3 method for class 'linim'
shift(X, ...)

  ## S3 method for class 'linim'
scalardilate(X, f, ..., origin=NULL)

  ## S3 method for class 'linim'
affine(X, mat=diag(c(1,1)), vec=c(0,0), ...)
```

Arguments

`X, x, object`	A pixel image on a linear network (object of class `"linim"`).
`...`	Extra arguments passed to other methods.
`f`	Numeric. Scalar dilation factor.
`mat`	Numeric matrix representing the linear transformation.
`vec`	Numeric vector of length 2 specifying the shift vector.
`origin`	Character string determining a location that will be shifted to the origin. Options are `"centroid"`, `"midpoint"` and `"bottomleft"`. Partially matched.

Details

These are methods for the generic functions `print`, `summary` and `as.data.frame`, and the **spatstat** generic functions `as.im`, `shift`, `scalardilate` and `affine`.

An object of class `"linfun"` represents a pixel image defined on a linear network.

The method `as.im.linim` extracts the pixel values and returns a pixel image of class `"im"`.

The method `as.data.frame.linim` returns a data frame giving spatial locations (in cartesian and network coordinates) and corresponding function values.

The methods `shift.linim`, `scalardilate.linim` and `affine.linim` apply geometric transformations to the pixels and the underlying linear network, without changing the pixel values.

Value

For `print.linim` the result is `NULL`.

The function `summary.linim` returns an object of class `"summary.linim"`. In normal usage this summary is automatically printed by `print.summary.linim`.

For `as.im.linim` the result is an object of class `"im"`.

For the geometric transformations `shift.linim`, `scalardilate.linim` and `affine.linim`, the result is another object of class `"linim"`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>

Examples

```
M <- psp2mask(as.psp(simplenet))
Z <- as.im(function(x,y) {x-y}, W=M)
X <- linim(simplenet, Z)
## ............  print basic details .........................
X
## ............  print gory details  .........................
summary(X)
## ..........................................................
shift(X, c(1,1))
scalardilate(X, 2)
head(as.data.frame(X))
```

`methods.linnet` *Methods for Linear Networks*

Description

These are methods for the class `"linnet"` of linear networks.

Usage

```
as.linnet(X, ...)

## S3 method for class 'linnet'
as.linnet(X, ..., sparse, maxsize=30000)

## S3 method for class 'linnet'
as.owin(W, ...)

## S3 method for class 'linnet'
as.psp(x, ..., fatal=TRUE)
```

```
## S3 method for class 'linnet'
nsegments(x)

## S3 method for class 'linnet'
nvertices(x, ...)

## S3 method for class 'linnet'
pixellate(x, ...)

## S3 method for class 'linnet'
print(x, ...)

## S3 method for class 'linnet'
summary(object, ...)

## S3 method for class 'linnet'
unitname(x)

## S3 replacement method for class 'linnet'
unitname(x) <- value

vertexdegree(x)

## S3 method for class 'linnet'
vertices(w)

## S3 method for class 'linnet'
volume(x)

## S3 method for class 'linnet'
Window(X, ...)
```

Arguments

`x`, `X`, `object`, `w`, `W`	An object of class `"linnet"` representing a linear network.
`...`	Arguments passed to other methods.
`value`	A valid name for the unit of length for `x`. See `unitname`.
`fatal`	Logical value indicating whether data in the wrong format should lead to an error (`fatal=TRUE`) or a warning (`fatal=FALSE`).
`sparse`	Logical value indicating whether to use a sparse matrix representation, as explained in `linnet`. Default is to keep the same representation as in `X`.
`maxsize`	Maximum permitted number of network vertices (to prevent a system crash due to lack of memory) when creating a network with `sparse=FALSE`.

Details

The function `as.linnet` is generic. It converts data from some other format into an object of class `"linnet"`. The method `as.linnet.lpp` extracts the linear network information from an `lpp` object. The method `as.linnet.linnet` converts a linear network into another linear network with the required format.

The other functions are methods for the generic commands `as.owin`, `as.psp`, `nsegments`, `nvertices`, `pixellate`, `print`, `summary`, `unitname`, `unitname<-`, `vertices`, `volume` and `Window` for the class `"linnet"`.

The methods `as.owin.linnet` and `Window.linnet` extract the window containing the linear network, and return it as an object of class `"owin"`.

The method `as.psp.linnet` extracts the lines of the linear network as a line segment pattern (object of class `"psp"`) while `nsegments.linnet` simply counts the number of line segments.

The method `vertices.linnet` extracts the vertices (nodes) of the linear network and `nvertices.linnet` simply counts the vertices. The function `vertexdegree` calculates the topological degree of each vertex (the number of lines emanating from that vertex) and returns these values as an integer vector.

The method `pixellate.linnet` applies `as.psp.linnet` to convert the network to a collection of line segments, then invokes `pixellate.psp`.

Value

For `as.linnet` the value is an object of class `"linnet"`. For other functions, see the help file for the corresponding generic function.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

See Also

`linnet`.

Generic functions: `as.owin`, `as.psp`, `nsegments`, `nvertices`, `pixellate`, `print`, `summary`, `unitname`, `unitname<-`, `vertices`, `volume` and `Window`.

Special tools: `thinNetwork`, `insertVertices`, `joinVertices`, `connected.linnet`.

`lixellate` for dividing segments into shorter segments.

Examples

```
simplenet
summary(simplenet)
nsegments(simplenet)
nvertices(simplenet)
pixellate(simplenet)
volume(simplenet)
unitname(simplenet) <- c("cubit", "cubits")
Window(simplenet)
```

`methods.lpp` *Methods for Point Patterns on a Linear Network*

Description

These are methods specifically for the class `"lpp"` of point patterns on linear networks.

Usage

```
## S3 method for class 'lpp'
as.ppp(X, ..., fatal=TRUE)

## S3 method for class 'lpp'
as.psp(x, ..., fatal=TRUE)

## S3 replacement method for class 'lpp'
marks(x, ...) <- value

## S3 method for class 'lpp'
nsegments(x)

## S3 method for class 'lpp'
print(x, ...)

## S3 method for class 'summary.lpp'
print(x, ...)

## S3 method for class 'lpp'
summary(object, ...)

## S3 method for class 'lpp'
unitname(x)

## S3 replacement method for class 'lpp'
unitname(x) <- value

## S3 method for class 'lpp'
unmark(X)
```

Arguments

`x`, `X`, `object`	An object of class `"lpp"` representing a point pattern on a linear network.
`...`	Arguments passed to other methods.
`value`	Replacement value for the `marks` or `unitname` of `x`. See Details.
`fatal`	Logical value indicating whether data in the wrong format should lead to an error (`fatal=TRUE`) or a warning (`fatal=FALSE`).

Details

These are methods for the generic functions `as.ppp`, `as.psp`, `marks<-`, `nsegments`, `print`, `summary`, `unitname`, `unitname<-` and `unmark` for objects of the class `"lpp"`.

For `"marks<-.lpp"` the replacement `value` should be either `NULL`, or a vector of length equal to the number of points in `x`, or a data frame with one row for each point in `x`.

For `"unitname<-.lpp"` the replacement `value` should be a valid name for the unit of length, as described in `unitname`.

Value

See the documentation on the corresponding generic function.

Other methods

An object of class `"lpp"` also inherits the class `"ppx"` for which many other methods are available. See `methods.ppx`.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

See Also

`lpp`, `intensity.lpp`, `methods.ppx`

Examples

```
X <- runiflpp(10, simplenet)
unitname(X) <- c("furlong", "furlongs")
X
summary(X)
summary(chicago)
nsegments(X)
Y <- as.ppp(X)
```

`methods.lppm` *Methods for Fitted Point Process Models on a Linear Network*

Description

These are methods for the class `"lppm"` of fitted point process models on a linear network.

Usage

```
  ## S3 method for class 'lppm'
coef(object, ...)

  ## S3 method for class 'lppm'
emend(object, ...)

  ## S3 method for class 'lppm'
extractAIC(fit, ...)

  ## S3 method for class 'lppm'
formula(x, ...)

  ## S3 method for class 'lppm'
logLik(object, ...)

  ## S3 method for class 'lppm'
deviance(object, ...)

  ## S3 method for class 'lppm'
nobs(object, ...)

  ## S3 method for class 'lppm'
print(x, ...)

  ## S3 method for class 'lppm'
summary(object, ...)

  ## S3 method for class 'lppm'
terms(x, ...)

  ## S3 method for class 'lppm'
update(object, ...)

  ## S3 method for class 'lppm'
valid(object, ...)

  ## S3 method for class 'lppm'
vcov(object, ...)

  ## S3 method for class 'lppm'
as.linnet(X, ...)

  ## S3 method for class 'lppm'
response(object)
```

Arguments

`object`, `fit`, `x`, `X`	An object of class `"lppm"` representing a fitted point process model on a linear network.
`...`	Arguments passed to other methods, usually the method for the class `"ppm"`.

Details

These are methods for the R generic commands `coef`, `extractAIC`, `formula`, `logLik`, `deviance`, `nobs`, `print`, `summary`, `terms`, `update` and `vcov`, and the **spatstat** generic commands `as.linnet`, `emend`, `response` and `valid`, for the class `"lppm"`.

Value

For `as.linnet.lppm` a linear network (object of class `"linnet"`). For `emend.lppm` another fitted model of the same class `"lppm"`. For `response.lppm` a spatial point pattern on a linear network (object of class `"lpp"`). For `valid.lppm` a logical value.

For the other methods, see the help for the default methods.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>, Rolf Turner <`rolfturner@posteo.net`> and Ege Rubak <`rubak@math.aau.dk`>.

See Also

`lppm`, `plot.lppm`.

Examples

```
X <- runiflpp(15, simplenet)
fit <- lppm(X ~ x)
print(fit)
coef(fit)
formula(fit)
terms(fit)
logLik(fit)
deviance(fit)
nobs(fit)
extractAIC(fit)
update(fit, ~1)
valid(fit)
vcov(fit)
response(fit)
```

`model.frame.lppm` *Extract the Variables in a Point Process Model on a Network*

Description

Given a fitted point process model on a network, this function returns a data frame containing all the variables needed to fit the model using the Berman-Turner device.

Usage

```
 ## S3 method for class 'lppm'
model.frame(formula, ...)
```

Arguments

`formula`	A fitted point process model on a linear network. An object of class `"lppm"`.
`...`	Additional arguments passed to `model.frame.glm`.

Details

The function `model.frame` is generic. This function is a method for `model.frame` for fitted point process models on a linear network (objects of class `"lppm"`).

The first argument should be a fitted point process model; it has to be named `formula` for consistency with the generic function.

The result is a data frame containing all the variables used in fitting the model. The data frame has one row for each quadrature point used in fitting the model. The quadrature scheme can be extracted using `quad.ppm`.

Value

A `data.frame` containing all the variables used in the fitted model, plus additional variables specified in `...`. It has an additional attribute `"terms"` containing information about the model formula. For details see `model.frame.glm`.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>, Rolf Turner <`rolfturner@posteo.net`> and Ege Rubak <`rubak@math.aau.dk`>.

References

Baddeley, A. and Turner, R. (2000) Practical maximum pseudolikelihood for spatial point patterns. *Australian and New Zealand Journal of Statistics* **42**, 283–322.

See Also

`lppm`, `model.frame`, `model.matrix.ppm`

Examples

```
fit <- lppm(spiders ~ x)
mf <- model.frame(fit)
```

`model.images.lppm` *Compute Images of Constructed Covariates*

Description

For a point process model fitted to spatial point pattern data on a linear network, this function computes pixel images of the covariates in the design matrix.

Usage

```
  ## S3 method for class 'lppm'
model.images(object, L = as.linnet(object), ...)
```

Arguments

`object`	Fitted point process model on a linear network. An object of class `"lppm"`.
`L`	A linear network (object of class `"linnet"`) in which the images should be computed. Defaults to the network in which the model was fitted.
`...`	Other arguments (such as `na.action`) passed to `model.matrix.lm`.

Details

This command is similar to `model.matrix.lppm` except that it computes pixel images of the covariates, instead of computing the covariate values at certain points only.

The `object` must be a fitted spatial point process model on a linear network (object of class `"lppm"` produced by the model-fitting function `lppm`).

The spatial covariates required by the model-fitting procedure are computed at every location on the network `L`.

Note that the spatial covariates computed here are not necessarily the original covariates that were supplied when fitting the model. Rather, they are the canonical covariates, the covariates that appear in the loglinear representation of the (conditional) intensity and in the columns of the design matrix. For example, they might include dummy or indicator variables for different levels of a factor, depending on the contrasts that are in force.

The format of the result depends on whether the original point pattern data were marked or unmarked.

- If the original dataset was unmarked, the result is a named list of pixel images on the network (objects of class `"linim"`) containing the values of the spatial covariates. The names of the list elements are the names of the covariates determined by `model.matrix.lm`. The result is also of class `"solist"` so that it can be plotted immediately.

- If the original dataset was a multitype point pattern, the result is a `hyperframe` with one column for each possible type of points. Each column is a named list of pixel images on the network (objects of class `"linim"`) containing the values of the spatial covariates. The row names of the hyperframe are the names of the covariates determined by `model.matrix.lm`.

The pixel resolution is determined by the arguments `...` and `spatstat.options`.

Value

A list (of class `"solist"`) or array (of class `"hyperframe"`) containing pixel images on the network (objects of class `"linim"`).

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`, Rolf Turner `<rolfturner@posteo.net>` and Ege Rubak `<rubak@math.aau.dk>`.

See Also

`model.matrix.ppm`, `model.matrix`, `lppm`.

Examples

```
fit <- lppm(spiders ~ x + polynom(y, 2))
model.images(fit)
```

`model.matrix.lppm` *Extract Design Matrix from Point Process Model on a Network*

Description

Given a point process model that has been fitted to spatial point pattern data on a linear network, this function extracts the design matrix of the model.

Usage

```
   ## S3 method for class 'lppm'
model.matrix(object,
                              data=model.frame(object, na.action=NULL),
                              ...,
                              keepNA=TRUE)
```

Arguments

`object`	The fitted point process model. An object of class `"lppm"`.
`data`	A model frame, containing the data required for the Berman-Turner device.
`keepNA`	Logical. Determines whether rows containing NA values will be deleted or retained.
`...`	Other arguments (such as `na.action`) passed to `model.matrix.lm`.

Details

This is a method for the generic function `model.matrix`. It extracts the design matrix of a spatial point process model on a linear network (object of class `"lppm"`).

More precisely, this command extracts the design matrix of the generalised linear model associated with a spatial point process model.

The `object` must be a fitted point process model on a network (object of class `"lppm"`) produced by the model-fitting function `lppm`. The method `model.matrix.lppm` extracts the model matrix for the GLM.

The result is a matrix, with one row for every quadrature point in the fitting procedure, and one column for every canonical covariate in the design matrix.

If there are `NA` values in the covariates, the argument `keepNA` determines whether to retain or delete the corresponding rows of the model matrix. The default `keepNA=TRUE` is to retain them. Note that this differs from the default behaviour of many other methods for `model.matrix`, which typically delete rows containing `NA`.

Value

A matrix. Columns of the matrix are canonical covariates in the model. Rows of the matrix correspond to quadrature points in the fitting procedure (provided `keepNA=TRUE`).

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`, Rolf Turner `<rolfturner@posteo.net>` and Ege Rubak `<rubak@math.aau.dk>`.

See Also

`model.matrix`, `model.images.lppm`, `lppm`

Examples

```
fit <- lppm(spiders ~ x + y)
head(model.matrix(fit))
```

`nncross.lpp` *Nearest Neighbours on a Linear Network*

Description

Given two point patterns `X` and `Y` on a linear network, finds the nearest neighbour in `Y` of each point of `X` using the shortest path in the network.

Usage

```
  ## S3 method for class 'lpp'
nncross(X, Y,
          iX=NULL, iY=NULL,
          what = c("dist", "which"),
          ...,
          k = 1,
          method="C")
```

Arguments

`X, Y`	Point patterns on a linear network (objects of class `"lpp"`). They must lie on the *same* linear network.
`iX, iY`	Optional identifiers, used to determine whether a point in `X` is identical to a point in `Y`. See Details.
`what`	Character string specifying what information should be returned. Either the nearest neighbour distance (`"dist"`), the identifier of the nearest neighbour (`"which"`), or both.
`...`	Ignored.
`k`	Integer, or integer vector. The algorithm will compute the distance to the `k`th nearest neighbour, for each value of `k`.
`method`	Internal use only.

Details

Given two point patterns `X` and `Y` on the same linear network, this function finds, for each point of `X`, the nearest point of `Y`, measuring distance by the shortest path in the network. The distance between these points is also computed.

The return value is a data frame, with rows corresponding to the points of `X`. The first column gives the nearest neighbour distances (i.e. the `i`th entry is the distance from the `i`th point of `X` to the nearest element of `Y`). The second column gives the indices of the nearest neighbours (i.e.\ the `i`th entry is the index of the nearest element in `Y`.) If `what="dist"` then only the vector of distances is returned. If `what="which"` then only the vector of indices is returned.

Note that this function is not symmetric in `X` and `Y`. To find the nearest neighbour in `X` of each point in `Y`, use `nncross(Y,X)`.

The arguments `iX` and `iY` are used when the two point patterns `X` and `Y` have some points in common. In this situation `nncross(X, Y)` would return some zero distances. To avoid this, attach a unique integer identifier to each point, such that two points are identical if their identifying numbers are equal. Let `iX` be the vector of identifier values for the points in `X`, and `iY` the vector of identifiers for points in `Y`. Then the code will only compare two points if they have different values of the identifier. See the Examples.

The `k`th nearest neighbour may be undefined, for example if there are fewer than `k+1` points in the dataset, or if the linear network is not connected. In this case, the `k`th nearest neighbour distance is infinite.

Value

By default (if `what=c("dist", "which")` and `k=1`) a data frame with two columns:

`dist`	Nearest neighbour distance
`which`	Nearest neighbour index in `Y`

If `what="dist"`, a vector of nearest neighbour distances.

If `what="which"`, a vector of nearest neighbour indices.

If `k` is a vector of integers, the result is a matrix with one row for each point in `X`, giving the distances and/or indices of the kth nearest neighbours in `Y`.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>, Rolf Turner <`rolfturner@posteo.net`> and Ege Rubak <`rubak@math.aau.dk`>

See Also

`nndist.lpp` for nearest neighbour distances in a single point pattern.

`nnwhich.lpp` to identify which points are nearest neighbours in a single point pattern.

Examples

```
# two different point patterns
X <- runiflpp(3, simplenet)
Y <- runiflpp(5, simplenet)
nn <- nncross(X,Y)
nn
plot(simplenet, main="nncross")
plot(X, add=TRUE, cols="red")
plot(Y, add=TRUE, cols="blue", pch=16)
XX <- as.ppp(X)
YY <- as.ppp(Y)
i <- nn$which
arrows(XX$x, XX$y, YY[i]$x, YY[i]$y, length=0.15)

# nearest and second-nearest neighbours
nncross(X, Y, k=1:2)

# two patterns with some points in common
X <- Y[1:2]
iX <- 1:2
iY <- 1:5
nncross(X,Y, iX, iY)
```

`nndist.lpp` *Nearest neighbour distances on a linear network*

Description

Given a pattern of points on a linear network, compute the nearest-neighbour distances, measured by the shortest path in the network.

Usage

```
## S3 method for class 'lpp'
nndist(X, ..., k=1, by=NULL, method="C")
```

Arguments

`X`	Point pattern on linear network (object of class `"lpp"`).
`k`	Integer, or integer vector. The algorithm will compute the distance to the `k`th nearest neighbour.
`by`	Optional. A factor, which separates `X` into groups. The algorithm will compute the distance to the nearest point in each group.
`method`	Optional string determining the method of calculation. Either `"interpreted"` or `"C"`.
`...`	Ignored.

Details

Given a pattern of points on a linear network, this function computes the nearest neighbour distance for each point (i.e. the distance from each point to the nearest other point), measuring distance by the shortest path in the network.

If `method="C"` the distances are computed using code in the C language. If `method="interpreted"` then the computation is performed using interpreted R code. The R code is much slower, but is provided for checking purposes.

The `k`th nearest neighbour distance is infinite if the `k`th nearest neighbour does not exist. This can occur if there are fewer than `k+1` points in the dataset, or if the linear network is not connected.

If the argument `by` is given, it should be a `factor`, of length equal to the number of points in `X`. This factor effectively partitions `X` into subsets, each subset associated with one of the levels of `X`. The algorithm will then compute, for each point of `X`, the distance to the nearest neighbour *in each subset*.

Value

A numeric vector, of length equal to the number of points in `X`, or a matrix, with one row for each point in `X` and one column for each entry of `k`. Entries are nonnegative numbers or infinity (`Inf`).

Distance values

The values returned by `nndist(X)` are distances, expressed as multiples of the unit of length of the spatial coordinates in `X`. The unit of length is given by `unitname(X)`.

Note that, if the unit of length in `X` is a composite expression such as '2 microns', then the values of `nndist(X)` are expressed as multiples of 2 microns, rather than being expressed in microns.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>

See Also

`lpp`

Examples

```
X <- runiflpp(12, simplenet)
nndist(X)
nndist(X, k=2)

marks(X) <- factor(rep(letters[1:3], 4))
nndist(X, by=marks(X))
```

`nnfromvertex` *Nearest Data Point From Each Vertex in a Network*

Description

Given a point pattern on a linear network, for each vertex of the network find the nearest data point.

Usage

```
nnfromvertex(X, what = c("dist", "which"), k = 1)
```

Arguments

`X`	Point pattern on a linear network (object of class `"lpp"`).
`what`	Character string specifying whether to return the nearest-neighbour distances, nearest-neighbour identifiers, or both.
`k`	Integer, or integer vector, specifying that the `k`th nearest neighbour should be returned.

Details

For each vertex (node) of the linear network, this algorithm finds the nearest data point to the vertex, and returns either the distance from the vertex to its nearest neighbour in `X`, or the serial number of the nearest neighbour in `X`, or both.

If `k` is an integer, then the `k`-th nearest neighbour is found instead.

If `k` is an integer vector, this is repeated for each integer in `k`.

Value

A numeric vector, matrix, or data frame.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`.

See Also

`nndist.lpp`

Examples

```
X <- runiflpp(5, simplenet)
nnfromvertex(X)
nnfromvertex(X, k=1:3)
```

`nnfun.lpp` *Nearest Neighbour Map on Linear Network*

Description

Compute the nearest neighbour function of a point pattern on a linear network.

Usage

```
  ## S3 method for class 'lpp'
nnfun(X, ..., k=1, value=c("index", "mark"))
```

Arguments

`X`	A point pattern on a linear network (object of class `"lpp"`).
`k`	Integer. The algorithm finds the `k`th nearest neighbour in `X` from any spatial location.
`value`	String (partially matched) specifying whether to return the index of the neighbour (`value="index"`, the default) or the mark value of the neighbour (`value="mark"`).
`...`	Other arguments are ignored.

Details

The (geodesic) *nearest neighbour function* of a point pattern `X` on a linear network `L` tells us which point of `X` is closest to any given location.

If `X` is a point pattern on a linear network `L`, the *nearest neighbour function* of `X` is the mathematical function f defined for any location s on the network by `f(s) = i`, where `X[i]` is the closest point of `X` to the location `s` measured by the shortest path. In other words the value of `f(s)` is the identifier or serial number of the closest point of `X`.

The command `nnfun.lpp` is a method for the generic command `nnfun` for the class `"lpp"` of point patterns on a linear network.

If `X` is a point pattern on a linear network, `f <- nnfun(X)` returns a *function* in the R language, with arguments `x,y, ...`, that represents the nearest neighbour function of `X`. Evaluating the function `f` in the form `v <- f(x,y)`, where `x` and `y` are any numeric vectors of equal length containing coordinates of spatial locations, yields a vector of identifiers or serial numbers of the data points closest to these spatial locations. More efficiently `f` can take the arguments `x, y, seg, tp` where `seg` and `tp` are the local coordinates on the network.

The result of `f <- nnfun(X)` also belongs to the class `"linfun"`. It can be printed and plotted immediately as shown in the Examples. It can be converted to a pixel image using `as.linim`.

Value

A `function` in the R language, with arguments `x,y` and optional arguments `seg,tp`. It also belongs to the class `"linfun"` which has methods for `plot`, `print` etc.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`, Rolf Turner `<rolfturner@posteo.net>` and Ege Rubak `<rubak@math.aau.dk>`

See Also

`linfun`, `methods.linfun`.

To compute the *distance* to the nearest neighbour, see `distfun.lpp`.

Examples

```
X <- runiflpp(3, simplenet)
f <- nnfun(X)
f
plot(f)
plot(nnfun(chicago, value="m"))
```

`nnwhich.lpp` *Identify Nearest Neighbours on a Linear Network*

Description

Given a pattern of points on a linear network, identify the nearest neighbour for each point, measured by the shortest path in the network.

Usage

```
## S3 method for class 'lpp'
nnwhich(X, ..., k=1, method="C")
```

Arguments

`X`	Point pattern on linear network (object of class `"lpp"`).
`method`	Optional string determining the method of calculation. Either `"interpreted"` or `"C"`.
`k`	Integer, or integer vector. The algorithm will find the `k`th nearest neighbour.
`...`	Ignored.

Details

Given a pattern of points on a linear network, this function finds the nearest neighbour of each point (i.e. for each point it identifies the nearest other point) measuring distance by the shortest path in the network.

If `method="C"` the task is performed using code in the C language. If `method="interpreted"` then the computation is performed using interpreted R code. The R code is much slower, but is provided for checking purposes.

The result is `NA` if the `k`th nearest neighbour does not exist. This can occur if there are fewer than `k+1` points in the dataset, or if the linear network is not connected.

Value

An integer vector, of length equal to the number of points in `X`, identifying the nearest neighbour of each point. If `nnwhich(X)[2] = 4` then the nearest neighbour of point 2 is point 4.

Alternatively a matrix with one row for each point in `X` and one column for each entry of `k`.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

See Also

`lpp`

Examples

```
X <- runiflpp(10, simplenet)
nnwhich(X)
nnwhich(X, k=2)
```

`pairdist.lpp` *Pairwise shortest-path distances between points on a linear network*

Description

Given a pattern of points on a linear network, compute the matrix of distances between all pairs of points, measuring distance by the shortest path in the network.

Usage

```
## S3 method for class 'lpp'
pairdist(X, ..., method="C")
```

Arguments

`X`	Point pattern on linear network (object of class `"lpp"`).
`method`	Optional string determining the method of calculation. Either `"interpreted"` or `"C"`.
`...`	Ignored.

Details

Given a pattern of points on a linear network, this function computes the matrix of distances between all pairs of points, measuring distance by the shortest path in the network.

If two points cannot be joined by a path, the distance between them is infinite (`Inf`).

The argument `method` is not normally used. It is retained only for developers to check the validity of the software.

Value

A symmetric matrix, whose values are nonnegative numbers or infinity (`Inf`).

Algorithms and accuracy

Distances are accurate within the numerical tolerance of the network, `summary(X)$toler`.

For network data stored in the non-sparse representation described in `linnet`, then pairwise distances are computed using the matrix of path distances between vertices of the network, using R code if `method = "interpreted"`, or using C code if `method="C"` (the default).

For networks stored in the sparse representation, the argument `method` has no effect, and the distances are computed using an efficient C algorithm.

Distance values

The values returned by `pairdist(X)` are distances, expressed as multiples of the unit of length of the spatial coordinates in `X`. The unit of length is given by `unitname(X)`.

Note that, if the unit of length in `X` is a composite expression such as ‘2 microns’, then the values of `pairdist(X)` are expressed as multiples of 2 microns, rather than being expressed in microns.

Author(s)

Ang Qi Wei <aqw07398@hotmail.com> and Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>.

See Also

`lpp`

Examples

```
X <- runiflpp(12, simplenet)
d <- pairdist(X)
d[1:3, 1:3]
```

---

pairs.linim *Scatterplot Matrix for Pixel Images on a Linear Network*

Description

Produces a scatterplot matrix of the pixel values in two or more pixel images on a linear network.

Usage

```
## S3 method for class 'linim'
pairs(..., plot=TRUE, eps=NULL)
```

Arguments

`...`	Any number of arguments, each of which is either a pixel image on a linear network (object of class `"linim"`), a pixel image (object of class `"im"`), or a named argument to be passed to `pairs.default`.
`plot`	Logical. If `TRUE`, the scatterplot matrix is plotted.
`eps`	Optional. Spacing between sample points on the network. A positive number.

Details

This is a method for the generic function `pairs` for the class of pixel images on a linear network.

It produces a square array of plot panels, in which each panel shows a scatterplot of the pixel values of one image against the corresponding pixel values of another image.

At least two of the arguments `...` should be a pixel image on a linear network (object of class `"linim"`). They should be defined on the **same** linear network, but may have different pixel resolutions.

First the pixel values of each image are extracted at a set of sample points equally-spaced across the network. Then `pairs.default` is called to plot the scatterplot matrix.

Any arguments in `...` which are not pixel images will be passed to `pairs.default` to control the plot.

Value

Invisible. A `data.frame` containing the corresponding pixel values for each image. The return value also belongs to the class `plotpairsim` which has a plot method, so that it can be re-plotted.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>, Rolf Turner <`rolfturner@posteo.net`> and Ege Rubak <`rubak@math.aau.dk`>

See Also

`pairs.default`, `pairs.im`

Examples

```
fit <- lppm(chicago ~ marks * (x+y))
lam <- predict(fit)
do.call(pairs, lam)
```

`persp.linfun` *Perspective View of Function on a Linear Network*

Description

Given a function on a linear network, generate a perspective view.

Usage

```
 ## S3 method for class 'linfun'
persp(x, ..., main, eps = NULL, dimyx = NULL, xy = NULL)
```

Arguments

`x`	The function to be plotted. An object of class `"linfun"`.
`...`	Arguments passed to `persp.linim` controlling the appearance of the plot.
`main`	Main title for the plot.
`eps, dimyx, xy`	Arguments passed to `as.linim` determining the spatial resolution when the function is converted to an image.

Details

The function `x` is converted to a pixel image on the linear network using `as.linim`. Then `persp.linim` is invoked to generate the perspective plot.

This style of plot is often attributed to Okabe and Sugihara (2012).

Value

(Invisibly) the perspective transformation matrix, as described in the help for `persp.default`.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>` and Greg McSwiggan.

References

Okabe, A. and Sugihara, K. (2012) *Spatial Analysis Along Networks*. John Wiley and Sons, New York.

See Also

`persp.linim`

Examples

```
f <- linfun(function(x,y,seg,tp) { abs(sin(25*x)) + abs(sin(15*y)) }, simplenet)
persp(f, phi=20)
```

`persp.linim` *Perspective View of Pixel Image on a Linear Network*

Description

Given a pixel image on a linear network, generate a perspective view.

Usage

```
 ## S3 method for class 'linim'
persp(x, ..., main,
                         grid = TRUE, ngrid = 10,
                         col.grid = "grey", col.base = "white",
                         neg.args=list(), warncross=FALSE)
```

Arguments

`x`	Pixel image on a linear network (object of class `"linim"`).
`...`	Arguments passed to `persp.default` to control the perspective view, or passed to `segments` or `polygon` to control the appearance of the vertical panes.
`main`	Main title for the plot.
`grid`	Logical value indicating whether to draw a rectangular grid at height zero, to assist the perception of perspective.
`ngrid`	Number of grid lines to draw, if `grid=TRUE`.
`col.grid`	Colour of grid lines, if `grid=TRUE`.

`col.base`	Colour of base plane, if `grid=TRUE`.
`neg.args`	Optional list of arguments passed to `polygon` when displaying negative values of the function.
`warncross`	Logical value indicating whether to issue a warning if two segments of the network cross each other (which causes difficulty for the algorithm).

Details

The pixel values are interpreted as the spatially-varying height of a vertical surface erected on each segment of the linear network. These surfaces are drawn in perspective view.

This style of plot is often attributed to Okabe and Sugihara (2012).

Value

(Invisibly) the perspective transformation matrix, as described in the help for `persp.default`.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>` and Greg McSwiggan.

References

Okabe, A. and Sugihara, K. (2012) *Spatial Analysis Along Networks*. John Wiley and Sons, New York.

See Also

`persp.linfun`

Examples

```
if(interactive()) {
  Z <- density(chicago, 100)
} else {
  X <- runiflpp(10, simplenet)
  Z <- density(X, 0.1)
}
persp(Z, theta=30, phi=20)
```

`plot.linim` *Plot Pixel Image on Linear Network*

Description

Given a pixel image on a linear network, the pixel values are displayed either as colours or as line widths.

Usage

```
## S3 method for class 'linim'
plot(x, ..., style = c("colour", "width"),
             scale, adjust = 1, fatten = 0,
             negative.args = list(col=2),
             legend=TRUE,
             leg.side=c("right", "left", "bottom", "top"),
             leg.sep=0.1,
             leg.wid=0.1,
             leg.args=list(),
             leg.scale=1,
             zlim,
             box=FALSE,
             do.plot=TRUE)
```

Arguments

`x`	The pixel image to be plotted. An object of class `"linim"`.
`...`	Extra graphical parameters, passed to `plot.im` if `style="colour"`, or to `polygon` if `style="width"`.
`style`	Character string (partially matched) specifying the type of plot. See Details.
`scale`	Physical scale factor for representing the pixel values as line widths.
`adjust`	Adjustment factor for the conversion of pixel value to line width, when `style="width"`.
`fatten`	Distance by which the line segments should be thickened, when `style="colour"`.
`negative.args`	A list of arguments to be passed to `polygon` specifying how to plot negative values of `x` when `style="width"`.
`legend`	Logical value indicating whether to plot a legend (colour ribbon or scale bar).
`leg.side`	Character string (partially matched) indicating where to display the legend relative to the main image.
`leg.sep`	Factor controlling the space between the legend and the image.
`leg.wid`	Factor controlling the width of the legend.
`leg.scale`	Rescaling factor for annotations on the legend. The values on the numerical scale printed beside the legend will be multiplied by this rescaling factor.
`leg.args`	List of additional arguments passed to `image.default`, `axis` or `text.default` to control the display of the legend. These may override the `...` arguments.
`zlim`	The range of numerical values that should be mapped. A numeric vector of length 2. Defaults to the range of values of `x`.
`box`	Logical value indicating whether to draw a bounding box.
`do.plot`	Logical value indicating whether to actually perform the plot.

Details

This is the `plot` method for objects of class `"linim"`. Such an object represents a pixel image defined on a linear network.

If `style="colour"` (the default) then the pixel values of `x` are plotted as colours, using `plot.im`. The mapping from pixel values to colours is determined by any additional arguments `...` which are passed to `plot.im`.

If `style="width"` then the pixel values of `x` are used to determine the widths of thick lines centred on the line segments of the linear network. This style of plot is often attributed to Xie and Yan (2008). The mapping from pixel values to line widths is determined by the arguments `scale` and `adjust`. The plotting of colours and borders of the lines is controlled by the additional arguments `...` which are passed to `polygon`. A different set of colours and borders can be assigned to negative pixel values by passing a list of arguments in `negative.args` as shown in the Examples.

A legend is displayed alongside the plot if `legend=TRUE` (the default). The legend displays the relationship between pixel values and colours (if `style="colour"`) or between pixel values and line widths (if `style="width"`).

The plotting of the legend itself is controlled by the arguments `leg.side`, `leg.sep`, `leg.wid`, `leg.scale` and the list of arguments `leg.args`, which are described above. If `style="colour"`, these arguments are mapped to the arguments `ribside`, `ribsep`, `ribwid`, `ribscale` and `ribargs` respectively, which are passed to `plot.im`.

Value

If `style="colour"`, the result is an object of class `"colourmap"` specifying the colour map used. If `style="width"`, the result is a numeric value `v` giving the physical scale: one unit of pixel value is represented as `v` physical units on the plot.

The result also has an attribute `"bbox"` giving a bounding box for the plot. The bounding box includes the ribbon or scale bar, if present, but not the main title.

Thin lines

When `style="colour"` it often appears that the lines are drawn too thin. This occurs because `x` is a pixel image, in which the only pixels that have a defined value are those which lie directly over the network. To make the lines appear thicker in the plot, use the argument `fatten`. The domain of the image will be expanded by a distance equal to `fatten/2` in every direction using `dilation.owin`; the pixel values will be extrapolated to this expanded domain using `nearestValue`. This may improve the visual appearance of the plot.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

References

Ang, Q.W., Baddeley, A. and Nair, G. (2012) Geometrically corrected second-order analysis of events on a linear network, with applications to ecology and criminology. *Scandinavian Journal of Statistics* **39**, 591–617.

Xie, Z. and Yan, J. (2008) Kernel Density Estimation of traffic accidents in a network space. *Computers, Environment and Urban Systems* **32**, 396–406.

See Also

`linim`, `plot.im`, `polygon`, `default.image.colours`

Examples

```
X <- linfun(function(x,y,seg,tp){y^2+x}, simplenet)
X <- as.linim(X)

plot(X, main="Colour represents function value")
plot(X, fatten=0.02, main="fattened")

plot(X, style="width", main="Width proportional to function value")

# signed values
f <- linfun(function(x,y,seg,tp){y-x}, simplenet)
plot(f, style="w", main="Negative values in red")

plot(f, style="w", negative.args=list(density=10),
     main="Negative values are hatched")
```

`plot.linnet` *Plot a linear network*

Description

Plots a linear network

Usage

```
 ## S3 method for class 'linnet'
plot(x, ..., main=NULL, add=FALSE,
                                   do.plot=TRUE,
                                   show.vertices=FALSE, show.window=FALSE,
                                   args.vertices=list(), args.segments=list())
```

Arguments

`x`	Linear network (object of class `"linnet"`).
`...`	Graphics arguments passed to `plot.psp` and `plot.ppp`.
`main`	Main title for plot. Use `main=""` to suppress it.
`add`	Logical. If `TRUE`, superimpose the graphics over the current plot. If `FALSE`, generate a new plot.
`do.plot`	Logical value specifying whether to actually perform the plot.

`show.vertices`	Logical value specifying whether to plot the vertices as well.
`show.window`	Logical value specifying whether to plot the window containing the linear network.
`args.segments`	Optional list of arguments passed to `plot.psp` when plotting the line segments of the network. These arguments override any arguments in `...`.
`args.vertices`	Optional list of arguments passed to `plot.ppp` when plotting the vertices of the network (only when `vertices=TRUE`). These arguments override any arguments in `...`.

Details

This is the plot method for class `"linnet"`.

The line segments of the network `x` are plotted using `plot.psp`. If `show.vertices=TRUE`, the vertices of the network will also be plotted, using `plot.ppp`. If `show.window=TRUE`, the window surrounding the network will also be plotted.

If the vertices or line segments of `x` are marked, the marks are not displayed by default. To plot the marks, set `use.marks=TRUE`. To plot the marks and plot the associated legends, set `use.marks=TRUE, legend=TRUE`. To plot only the marks of the segments and not the marks of the vertices, set `args.segments=list(use.marks` and so on.

Value

An (invisible) list with two elements, `segments` and `vertices` describing the representation of the marks. The element `segments` contains the result of `plot.psp` (either a `colourmap`, a numeric value or an `owin`). The element `vertices` contains the result of `plot.ppp` (a `symbolmap`) or `NULL`.

The result also has attribute `"bbox"` giving the bounding box for the plot.

Author(s)

Ang Qi Wei <`aqw07398@hotmail.com`> and Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>

See Also

`linnet`

Examples

```
plot(simplenet)

L <- simplenet
marks(L, "vertices") <- letters[1:nvertices(L)]
marks(L, "segments") <- runif(nsegments(L))
plot(L, show.vertices=TRUE, use.marks=TRUE, legend=TRUE)
```

`plot.lintess` *Plot a Tessellation on a Linear Network*

Description

Plot a tessellation or division of a linear network into tiles.

Usage

```
## S3 method for class 'lintess'
plot(x, ...,
        main, add = FALSE,
        style = c(”colour”, ”width”, ”image”),
        col = NULL, values=marks(x),
        ribbon=TRUE, ribargs=list(), multiplot=TRUE, do.plot=TRUE)
```

Arguments

`x`	Tessellation on a linear network (object of class `"lintess"`).
`...`	Arguments passed to `segments` (if `style="segments"`) or to `plot.im` (if `style="image"`) to control the plot.
`main`	Optional main title for the plot.
`add`	Logical value indicating whether the plot is to be added to an existing plot.
`style`	Character string (partially matched) specifying the type of plot. If `style="colour"` (the default), tiles are plotted using `segments` using colours to distinguish the different tiles or values. If `style="width"`, tiles are plotted using `segments` using different segment widths to distinguish the different tiles or values. If `style="image"`, the tessellation is converted to a pixel image and plotted by `plot.im`.
`col`	Vector of colours, or colour map, determining the colours used to plot the different tiles of the tessellation.
`values`	Values associated with each tile of the tessellation, used to determine the colours or widths. A vector with one entry for each tile, or a data frame with one row for each tile. The default is `marks(x)`, or if that is null, then `tilenames(x)`.
`ribbon`	Logical value specifying whether to print an explanatory legend for the colour map or width map.
`ribargs`	Arguments passed to `plot.colourmap` controlling the display of the colour map legend.
`multiplot`	Logical value determining what should happen if `marks(x)` has more than one column. If `multiplot=TRUE` (the default), several plot panels will be generated, one panel for each column of marks. If `multiplot=FALSE`, the first column of marks will be selected.
`do.plot`	Logical value specifying whether to actually generate the plot (`do.plot=TRUE`, the default) or just to compute the colour map and return it (`do.plot=FALSE`).

Details

A tessellation on a linear network `L` is a partition of the network into non-overlapping pieces (tiles). Each tile consists of one or more line segments which are subsets of the line segments making up the network. A tile can consist of several disjoint pieces.

This function plots the tessellation on the current device. It is a method for the generic `plot`.

If `style="colour"`, each tile is plotted using `segments`, drawing segments of different colours.

If `style="width"`, each tile is plotted using `segments`, drawing segments of different widths.

If `style="image"`, the tessellation is converted to a pixel image, and plotted as a colour image using `plot.im`.

The colours or widths are determined by the `values` associated with each tile of the tessellation. If `values` is missing, the default is to use the marks of the tessellation, or if there are no marks, the names of the tiles.

Value

(Invisible) colour map.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>

See Also

`lintess`

Examples

```
X <- runiflpp(7, simplenet)
Z <- divide.linnet(X)
plot(Z, main="tessellation on network")
points(as.ppp(X))
plot(Z, main="tessellation on network",
        values=1:nobjects(Z), style="w")
```

`plot.lpp` *Plot Point Pattern on Linear Network*

Description

Plots a point pattern on a linear network. Plot method for the class `"lpp"` of point patterns on a linear network.

Usage

```
## S3 method for class 'lpp'
plot(x, ..., main, add = FALSE,
     type = c("p", "n"),
     use.marks=TRUE, which.marks=NULL,
     legend=TRUE,
     leg.side=c("left", "bottom", "top", "right"),
     leg.args=list(),
     show.all = !add, show.window=FALSE, show.network=TRUE,
     do.plot = TRUE, multiplot=TRUE)
```

Arguments

`x`	Point pattern on a linear network (object of class `"lpp"`).
`...`	Additional arguments passed to `plot.linnet` or `plot.ppp`.
`main`	Main title for plot.
`add`	Logical value indicating whether the plot is to be added to the existing plot (`add=TRUE`) or whether a new plot should be initialised (`add=FALSE`, the default).
`type`	Type of plot: either `"p"` or `"n"`. If `type="p"` (the default), both the points and the observation window are plotted. If `type="n"`, only the window is plotted.
`use.marks`	logical flag; if `TRUE`, plot points using a different plotting symbol for each mark; if `FALSE`, only the locations of the points will be plotted, using `points()`.
`which.marks`	Index determining which column of marks to use, if the marks of `x` are a data frame. A character or integer vector identifying one or more columns of marks. If `add=FALSE` then the default is to plot all columns of marks, in a series of separate plots. If `add=TRUE` then only one column of marks can be plotted, and the default is `which.marks=1` indicating the first column of marks.
`legend`	Logical value indicating whether to add a legend showing the mapping between mark values and graphical symbols (for a marked point pattern).
`leg.side`	Position of legend relative to main plot.
`leg.args`	List of additional arguments passed to `plot.symbolmap` or `symbolmap` to control the legend. In addition to arguments documented under `plot.symbolmap`, and graphical arguments recognised by `symbolmap`, the list may also include the argument `sep` giving the separation between the main plot and the legend, or `sep.frac` giving the separation as a fraction of the relevant dimension (width or height) of the main plot.
`show.all`	Logical value indicating whether to plot everything including the main title and the window containing the network.
`show.window`	Logical value indicating whether to plot the window containing the network. Overrides `show.all`.
`show.network`	Logical value indicating whether to plot the network.
`do.plot`	Logical value determining whether to actually perform the plotting.
`multiplot`	Logical value giving permission to display multiple plots.

Details

The linear network is plotted by `plot.linnet`, then the points are plotted using code equivalent to `plot.ppp`.

Commonly-used arguments include:

- `col` and `lwd` for the colour and width of lines in the linear network
- `cols` for the colour or colours of the points
- `chars` for the plot characters representing different types of points
- `shape` to control the shape of the symbol (this argument takes precedence over `chars`).

These are documented in the help file for `plot.ppp`.

If `shape="crossticks"`, the points are drawn as short line segments perpendicular to the network.

Note that the linear network will be plotted even when `add=TRUE`, unless `show.network=FALSE`.

Value

(Invisible) object of class `"symbolmap"` giving the correspondence between mark values and plotting characters.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`, Rolf Turner `<rolfturner@posteo.net>` and Ege Rubak `<rubak@math.aau.dk>`.

See Also

`lpp`.

See `plot.ppp` for options for representing the points.

See also `points.lpp`, `text.lpp`.

Examples

```
plot(chicago, cols=1:7)
plot(dendrite, shape="crossticks", cols=2:4, size=8,
               leg.side="bottom", leg.args=list(lwd=3))
```

`plot.lppm` *Plot a Fitted Point Process Model on a Linear Network*

Description

Plots the fitted intensity of a point process model on a linear network.

Usage

```
  ## S3 method for class 'lppm'
plot(x, ..., type="trend")
```

Arguments

`x`	An object of class `"lppm"` representing a fitted point process model on a linear network.
`...`	Arguments passed to `plot.linim` to control the plot.
`type`	Character string (either `"trend"` or `"cif"`) determining whether to plot the fitted first order trend or the conditional intensity.

Details

This function is the plot method for the class `"lppm"`. It computes the fitted intensity of the point process model, and displays it using `plot.linim`.

The default is to display intensity values as colours. Alternatively if the argument `style="width"` is given, intensity values are displayed as the widths of thick lines drawn over the network.

Value

Null.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>

See Also

`lppm`, `plot.linim`, `methods.lppm`, `predict.lppm`.

Examples

```
X <- runiflpp(10, simplenet)
fit <- lppm(X ~x)
plot(fit)
plot(fit, style="width")
```

`points.lpp` *Draw Points on Existing Plot*

Description

For a point pattern on a linear network, this function draws the coordinates of the points only, on the existing plot display.

Usage

```
## S3 method for class 'lpp'
points(x, ...)
```

Arguments

`x`	A point pattern on a linear network (object of class `"lpp"`).
`...`	Additional arguments passed to `points.default`.

Details

This is a method for the generic function `points` for the class `"lpp"` of point patterns on a linear network.

If `x` is a point pattern on a linear network, then `points(x)` plots the spatial coordinates of the points only, on the existing plot display, without plotting the underlying network. It is an error to call this function if a plot has not yet been initialised.

The spatial coordinates are extracted and passed to `points.default` along with any extra arguments. Arguments controlling the colours and the plot symbols are interpreted by `points.default`. For example, if the argument `col` is a vector, then the `i`th point is drawn in the colour `col[i]`.

Value

Null.

Difference from plot method

The more usual way to plot the points is using `plot.lpp`. For example `plot(x)` would plot both the points and the underlying network, while `plot(x, add=TRUE)` would plot only the points. The interpretation of arguments controlling the colours and plot symbols is different here: they determine a symbol map, as explained in the help for `plot.ppp`.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>, Rolf Turner <`rolfturner@posteo.net`> and Ege Rubak <`rubak@math.aau.dk`>.

See Also

`plot.lpp`, `points.default`

Examples

```
plot(Frame(spiders), main="Spiders on a Brick Wall")
points(spiders)
```

`predict.lppm` *Predict Point Process Model on Linear Network*

Description

Given a fitted point process model on a linear network, compute the fitted intensity or conditional intensity of the model.

Usage

```
## S3 method for class 'lppm'
predict(object, ..., type = "trend",
                        locations = NULL, covariates = NULL,
                        se = FALSE,
                        new.coef=NULL)
```

Arguments

`object`	The fitted model. An object of class `"lppm"`, see `lppm`.
`type`	Type of values to be computed. Either `"trend"` or `"cif"`. Currently ignored.
`locations`	Optional. Locations at which predictions should be computed. Either a data frame with two columns of coordinates, or a binary image mask.
`covariates`	Values of external covariates required by the model. Either a data frame, or a list of images and/or functions.
`se`	Logical value indicating whether to calculate standard errors as well.
`new.coef`	Optional. Numeric vector of model coefficients, to be used instead of the fitted coefficients `coef(object)` when calculating the prediction.
`...`	Optional arguments passed to `as.mask` to determine the pixel resolution (if `locations` is missing).

Details

This function computes the fitted point process intensity, optionally with standard errors, for a point process model on a linear network. It is a method for the generic `predict` for the class `"lppm"`.

The argument `object` should be an object of class `"lppm"` (produced by `lppm`) representing a point process model on a linear network.

Currently the argument `type` has no effect. The fitted intensity is computed in all cases. This occurs because currently all fitted models of class `"lppm"` are Poisson point processes, where the trend, intensity, and conditional intensity are the same.

Predicted values are computed at the locations given by the argument `locations`. If this argument is missing, then predicted values are computed at a fine grid of points on the linear network.

- If `locations` is missing or `NULL` (the default), the return value is a pixel image (object of class `"linim"` and `"im"`) corresponding to a discretisation of the linear network, with numeric pixel values giving the predicted values at each location on the linear network. (If the model is multitype, the result is a list of such pixel images, one for each possible type of point.)

- If `locations` is a data frame, the result is a numeric vector of predicted values at the locations specified by the data frame.
- If `locations` is a binary mask, the result is a pixel image with predicted values computed at the pixels of the mask. (If the model is multitype, the result is a list of such pixel images, one for each possible type of point.)

If `se=TRUE`, standard errors are also computed. The result is a list of two elements, each following the format described above; the first element contains the fitted estimates, and the second element contains the standard errors.

Value

If `se=FALSE` (the default), the result is a pixel image (object of class `"linim"` and `"im"`) or a list of pixel images, or a numeric vector, depending on the argument `locations`. See Details.

If `se=TRUE`, the result is a list of two elements, each with the format described above.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

References

Ang, Q.W. (2010) *Statistical methodology for events on a network.* Master's thesis, School of Mathematics and Statistics, University of Western Australia.

Ang, Q.W., Baddeley, A. and Nair, G. (2012) Geometrically corrected second-order analysis of events on a linear network, with applications to ecology and criminology. *Scandinavian Journal of Statistics* **39**, 591–617.

Rakshit, S., McSwiggan, G., Nair, G. and Baddeley, A. (2021) Variable selection using penalised likelihoods for point patterns on a linear network. *Australian and New Zealand Journal of Statistics* **63**. DOI 10.1111/anzs.12341.

Baddeley, A., Nair, G., Rakshit, S., McSwiggan, G. and Davies, T.M. (2021) Analysing point patterns on networks — a review. *Spatial Statistics* **42**, 100435.

See Also

`lpp`, `linim`

Examples

```
X <- runiflpp(12, simplenet)
fit <- lppm(X ~ x)
v <- predict(fit, type="trend")
plot(v)
```

`pseudoR2.lppm` *Calculate Pseudo-R-Squared for Point Process Model on Linear Network*

Description

Given a fitted point process model on a linear network, calculate the pseudo-R-squared value, which measures the fraction of variation in the data that is explained by the model.

Usage

```
  ## S3 method for class 'lppm'
pseudoR2(object, ..., keepoffset=TRUE)
```

Arguments

`object`	Fitted point process model on a linear network. An object of class `"lppm"`.
`keepoffset`	Logical value indicating whether to retain offset terms in the model when computing the deviance difference. See Details.
`...`	Additional arguments passed to `deviance.lppm`.

Details

The function `pseudoR2` is generic, with methods for fitted point process models of class `"ppm"` and `"lppm"`.

This function computes McFadden's pseudo-Rsquared

$$R^2 = 1 - \frac{D}{D_0}$$

where D is the deviance of the fitted model `object`, and D_0 is the deviance of the null model. Deviance is defined as twice the negative log-likelihood or log-pseudolikelihood.

The null model is usually obtained by re-fitting the model using the trend formula `~1`. However if the original model formula included `offset` terms, and if `keepoffset=TRUE` (the default), then the null model formula consists of these offset terms. This ensures that the `pseudoR2` value is non-negative.

Value

A single numeric value.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`, Rolf Turner `<rolfturner@posteo.net>` and Ege Rubak `<rubak@math.aau.dk>`.

See Also

pseudoR2, deviance.lppm.

Examples

```
X <- rpoislpp(10, simplenet)
fit <- lppm(X ~ y)
pseudoR2(fit)
```

---

rcelllpp *Simulate Cell Process on Linear Network*

Description

Generate a realisation of the cell process on a linear network.

Usage

```
rcelllpp(L, lambda, rnumgen = NULL, ..., saveid=FALSE)
```

Arguments

`L`	Either a linear network (object of class `"linnet"`) or a tessellation on a linear network (object of class `"lintess"`).
`lambda`	Intensity of the process (expected number of points per unit length),
`rnumgen`	Optional. Random number generator for the number of points in each cell.
`...`	Additional arguments to `rnumgen`.
`saveid`	Logical value indicating whether to save information about cell membership.

Details

This function generates simulated realisations of a cell point process on a network, as described in Baddeley et al (2017). This is the analogue on a linear network of the two-dimensional cell point process of Baddeley and Silverman (1988).

The argument `L` should be a tessellation on a linear network. Alternatively if `L` is a linear network, it is converted to a tessellation by treating each network segment as a tile in the tessellation.

The cell process generates a point process by generating independent point processes inside each tile of the tessellation. Within each tile, given the number of random points in the tile, the points are independent and uniformly distributed within the tile.

By default (when `rnumgen` is not given), the number of points in a tile of length `t` is a random variable with mean and variance equal to `lambda * t`, generated by calling `rcellnumber`.

If `rnumgen` is given, it should be a function with arguments `rnumgen(n, mu, ...)` where `n` is the number of random integers to be generated, `mu` is the mean value of the distribution, and `...` are additional arguments, if needed. It will be called in the form `rnumgen(1, lambda * t, ...)` to determine the number of random points falling in each tile of length `t`.

Value

Point pattern on a linear network (object of class `"lpp"`). If `saveid=TRUE`, the result has an attribute `"cellid"` which is a factor specifying the cell that contains each point.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>.

References

Baddeley, A.J. and Silverman, B.W. (1984) A cautionary example on the use of second-order methods for analyzing point patterns. *Biometrics* **40**, 1089-1094.

Baddeley, A., Nair, G., Rakshit, S. and McSwiggan, G. (2017) 'Stationary' point processes are uncommon on linear networks. *STAT* **6**, 68–78.

See Also

`rSwitzerlpp`

Examples

```
X <- rcelllpp(domain(spiders), 0.01)
plot(X)
plot(linearK(X))
```

`relrisk.lpp` *Nonparametric Estimate of Spatially-Varying Relative Risk on a Network*

Description

Given a multitype point pattern on a linear network, this function estimates the spatially-varying probability of each type of point, or the ratios of such probabilities, using kernel smoothing.

Usage

```
## S3 method for class 'lpp'
relrisk(X, sigma, ...,
          at = c("pixels", "points"),
          relative=FALSE,
          adjust=1,
          casecontrol=TRUE, control=1, case,
          finespacing=FALSE)
```

Arguments

`X`	A multitype point pattern (object of class `"lpp"` which has factor valued marks).
`sigma`	The numeric value of the smoothing bandwidth (the standard deviation of Gaussian smoothing kernel) passed to `density.lpp`. Alternatively `sigma` may be a function which can be used to select the bandwidth. See Details.
`...`	Arguments passed to `density.lpp` to control the pixel resolution.
`at`	Character string specifying whether to compute the probability values at a grid of pixel locations (`at="pixels"`) or only at the points of `X` (`at="points"`).
`relative`	Logical. If `FALSE` (the default) the algorithm computes the probabilities of each type of point. If `TRUE`, it computes the *relative risk*, the ratio of probabilities of each type relative to the probability of a control.
`adjust`	Optional. Adjustment factor for the bandwidth `sigma`.
`casecontrol`	Logical. Whether to treat a bivariate point pattern as consisting of cases and controls, and return only the probability or relative risk of a case. Ignored if there are more than 2 types of points. See Details.
`control`	Integer, or character string, identifying which mark value corresponds to a control.
`case`	Integer, or character string, identifying which mark value corresponds to a case (rather than a control) in a bivariate point pattern. This is an alternative to the argument `control` in a bivariate point pattern. Ignored if there are more than 2 types of points.
`finespacing`	Logical value specifying whether to use a finer spatial resolution (with longer computation time but higher accuracy).

Details

The command `relrisk` is generic and can be used to estimate relative risk in different ways.

This function `relrisk.lpp` is the method for point patterns on a linear network (objects of class `"lpp"`). It computes *nonparametric* estimates of relative risk by kernel smoothing.

If `X` is a bivariate point pattern (a multitype point pattern consisting of two types of points) then by default, the points of the first type (the first level of `marks(X)`) are treated as controls or non-events, and points of the second type are treated as cases or events. Then by default this command computes the spatially-varying *probability* of a case, i.e. the probability $p(u)$ that a point at location u on the network will be a case. If `relative=TRUE`, it computes the spatially-varying *relative risk* of a case relative to a control, $r(u) = p(u)/(1 - p(u))$.

If `X` is a multitype point pattern with $m > 2$ types, or if `X` is a bivariate point pattern and `casecontrol=FALSE`, then by default this command computes, for each type j, a nonparametric estimate of the spatially-varying *probability* of an event of type j. This is the probability $p_j(u)$ that a point at location u on the network will belong to type j. If `relative=TRUE`, the command computes the *relative risk* of an event of type j relative to a control, $r_j(u) = p_j(u)/p_k(u)$, where events of type k are treated as controls. The argument `control` determines which type k is treated as a control.

If `at = "pixels"` the calculation is performed for every location u on a fine pixel grid over the network, and the result is a pixel image on the network representing the function $p(u)$, or a list of pixel images representing the functions $p_j(u)$ or $r_j(u)$ for $j = 1, \dots, m$. An infinite value of relative risk (arising because the probability of a control is zero) will be returned as `NA`.

If `at = "points"` the calculation is performed only at the data points x_i. By default the result is a vector of values $p(x_i)$ giving the estimated probability of a case at each data point, or a matrix of values $p_j(x_i)$ giving the estimated probability of each possible type j at each data point. If `relative=TRUE` then the relative risks $r(x_i)$ or $r_j(x_i)$ are returned. An infinite value of relative risk (arising because the probability of a control is zero) will be returned as `Inf`.

Estimation is performed by a Nadaraja-Watson type kernel smoother (McSwiggan et al., 2019).

The smoothing bandwidth `sigma` should be a single numeric value, giving the standard deviation of the isotropic Gaussian kernel. If `adjust` is given, the smoothing bandwidth will be `adjust * sigma` before the computation of relative risk.

Alternatively, `sigma` may be a function that can be applied to the point pattern `X` to select a bandwidth; the function must return a single numerical value; examples include the functions `bw.relrisk.lpp` and `bw.scott.iso`.

Accuracy depends on the spatial resolution of the density computations. If the arguments `dx` and `dt` are present, they are passed to `density.lpp` to determine the spatial resolution. Otherwise, the spatial resolution is determined by a default rule that depends on `finespacing` and `sigma`. If `finespacing=FALSE` (the default), the spatial resolution is equal to the default resolution for pixel images. If `finespacing=TRUE`, the spatial resolution is much finer and is determined by a rule which guarantees higher accuracy, but takes a longer time.

Value

If `X` consists of only two types of points, and if `casecontrol=TRUE`, the result is a pixel image on the network (if `at="pixels"`) or a vector (if `at="points"`). The pixel values or vector values are the probabilities of a case if `relative=FALSE`, or the relative risk of a case (probability of a case divided by the probability of a control) if `relative=TRUE`.

If `X` consists of more than two types of points, or if `casecontrol=FALSE`, the result is:

- (if `at="pixels"`) a list of pixel images on the network, with one image for each possible type of point. The result also belongs to the class `"solist"` so that it can be printed and plotted.
- (if `at="points"`) a matrix of probabilities, with rows corresponding to data points x_i, and columns corresponding to types j.

The pixel values or matrix entries are the probabilities of each type of point if `relative=FALSE`, or the relative risk of each type (probability of each type divided by the probability of a control) if `relative=TRUE`.

If `relative=FALSE`, the resulting values always lie between 0 and 1. If `relative=TRUE`, the results are either non-negative numbers, or the values `Inf` or `NA`.

Author(s)

Greg McSwiggan and Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`.

References

McSwiggan, G., Baddeley, A. and Nair, G. (2019) Estimation of relative risk for events on a linear network. *Statistics and Computing* **30** (2) 469–484.

See Also

relrisk

Examples

```
## case-control data: 2 types of points
set.seed(2020)
X <- superimpose(A=runiflpp(20, simplenet),
                 B=runifpointOnLines(20, as.psp(simplenet)[5]))
plot(X)
plot(relrisk(X, 0.15))
plot(relrisk(X, 0.15, case="B"))
head(relrisk(X, 0.15, at="points"))
## cross-validated bandwidth selection
plot(relrisk(X, bw.relrisk.lpp, hmax=0.3, allow.infinite=FALSE))

## more than 2 types
if(interactive()) {
  U <- chicago
  sig <- 170
} else {
  U <- do.call(superimpose,
               split(chicago)[c("theft", "cartheft", "burglary")])
  sig <- 40
}
plot(relrisk(U, sig))
head(relrisk(U, sig, at="points"))
plot(relrisk(U, sig, relative=TRUE, control="theft"))
```

`repairNetwork` *Repair Internal Data in a Linear Network*

Description

Detect and repair inconsistencies or duplication in the internal data of a network object.

Usage

`repairNetwork(X)`

Arguments

`X`	A linear network (object of class "`linnet`") or a point pattern on a linear network (object of class "`lpp`").

Details

This function detects and repairs inconsistencies in the internal data of `X`. Currently it does the following:

- checks that different ways of calculating the number of edges give the same answer
- removes any duplicated edges of the network
- ensures that each edge is recorded as a pair of vertex indices `(from, to)` with `from` < `to`.

Value

An object of the same kind as `X`.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`.

See Also

`thinNetwork`

`Replace.linim` *Reset Values in Subset of Image on Linear Network*

Description

Reset the values in a subset of a pixel image on a linear network.

Usage

```
  ## S3 replacement method for class 'linim'
x[i, j] <- value
```

Arguments

`x`	A pixel image on a linear network. An object of class `"linim"`.
`i`	Object defining the subregion or subset to be replaced. Either a spatial window (an object of class `"owin"`), or a pixel image with logical values, or a point pattern (an object of class `"ppp"`), or any type of index that applies to a matrix, or something that can be converted to a point pattern by `as.ppp` (using the window of `x`).
`j`	An integer or logical vector serving as the column index if matrix indexing is being used. Ignored if `i` is appropriate to some sort of replacement *other than* matrix indexing.
`value`	Vector, matrix, factor or pixel image containing the replacement values. Short vectors will be recycled.

Details

This function changes some of the pixel values in a pixel image. The image `x` must be an object of class `"linim"` representing a pixel image on a linear network.

The pixel values are replaced according to the rules described in the help for `[<-.im`. Then the auxiliary data are updated.

Value

The image `x` with the values replaced.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>, Rolf Turner <`rolfturner@posteo.net`> and Ege Rubak <`rubak@math.aau.dk`>.

See Also

`[<-.im`.

Examples

```
# make a function
Y <- as.linim(distfun(runiflpp(5, simplenet)))
# replace some values
B <- square(c(0.25, 0.55))
Y[B] <- 2
plot(Y, main="")
plot(B, add=TRUE, lty=3)
X <- runiflpp(4, simplenet)
Y[X] <- 5
```

`rhohat.lpp` *Nonparametric Estimate of Intensity as Function of a Covariate*

Description

Computes a nonparametric estimate of the intensity of a point process on a linear network, as a function of a (continuous) spatial covariate.

Usage

```
## S3 method for class 'lpp'
rhohat(object, covariate, ...,
       weights=NULL,
       method=c("ratio", "reweight", "transform"),
       horvitz=FALSE,
       smoother=c("kernel", "local", "decreasing", "increasing",
                  "mountain", "valley", "piecewise"),
```

```
        subset=NULL,
        do.CI=TRUE,
        jitter=TRUE, jitterfactor=1, interpolate=TRUE,
        nd=1000, eps=NULL, random=TRUE,
        n = 512, bw = "nrd0", adjust=1, from = NULL, to = NULL,
        bwref=bw,
        covname, confidence=0.95, positiveCI, breaks=NULL)

## S3 method for class 'lppm'
rhohat(object, covariate, ...,
        weights=NULL,
        method=c("ratio", "reweight", "transform"),
        horvitz=FALSE,
        smoother=c("kernel", "local", "decreasing", "increasing",
                   "mountain", "valley", "piecewise"),
        subset=NULL,
        do.CI=TRUE,
        jitter=TRUE, jitterfactor=1, interpolate=TRUE,
        nd=1000, eps=NULL, random=TRUE,
        n = 512, bw = "nrd0", adjust=1, from = NULL, to = NULL,
        bwref=bw,
        covname, confidence=0.95, positiveCI, breaks=NULL)
```

Arguments

`object`	A point pattern on a linear network (object of class `"lpp"`), or a fitted point process model on a linear network (object of class `"lppm"`).
`covariate`	Either a `function(x,y)` or a pixel image (object of class `"im"` or `"linim"`) providing the values of the covariate at any location. Alternatively one of the strings `"x"` or `"y"` signifying the Cartesian coordinates.
`weights`	Optional weights attached to the data points. Either a numeric vector of weights for each data point, or a pixel image (object of class `"im"`) or a `function(x,y)` providing the weights.
`method`	Character string determining the estimation method. See Details.
`horvitz`	Logical value indicating whether to use Horvitz-Thompson weights. See Details.
`smoother`	Character string determining the smoothing algorithm and the type of curve that will be estimated. See Details.
`subset`	Optional. A spatial window (object of class `"owin"`) specifying a subset of the data, from which the estimate should be calculated.
`do.CI`	Logical value specifying whether to calculate standard errors and confidence bands.
`jitter`	Logical value. If `jitter=TRUE` (the default), the values of the covariate at the data points will be jittered (randomly perturbed by adding a small amount of noise) using the function `jitter`. If `jitter=FALSE`, the covariate values at the data points will not be altered. See the section on *Randomisation and discretisation.*

`jitterfactor`	Numeric value controlling the scale of jittering. Passed to `jitter` as the argument `factor`.
`interpolate`	Logical value specifying whether to use spatial interpolation to obtain the values of the covariate at the data points, when the covariate is a pixel image (object of class `"im"` or `"linim"`). If `interpolate=FALSE`, the covariate value for each data point is simply the value of the covariate image at the pixel centre that is nearest to the data point. If `interpolate=TRUE`, the covariate value for each data point is obtained by interpolating the nearest pixel values using `interp.im`.
`eps, nd, random`	Arguments controlling the pixel resolution at which the covariate will be evaluated. See Details.
`bw`	Smoothing bandwidth or bandwidth rule (passed to `density.default`).
`adjust`	Smoothing bandwidth adjustment factor (passed to `density.default`).
`n, from, to`	Arguments passed to `density.default` to control the number and range of values at which the function will be estimated.
`bwref`	Optional. An alternative value of `bw` to use when smoothing the reference density (the density of the covariate values observed at all locations in the window).
`...`	Additional arguments passed to `density.default` or `locfit::locfit`.
`covname`	Optional. Character string to use as the name of the covariate.
`confidence`	Confidence level for confidence intervals. A number between 0 and 1.
`positiveCI`	Logical value. If `TRUE`, confidence limits are always positive numbers; if `FALSE`, the lower limit of the confidence interval may sometimes be negative. Default is `FALSE` if `smoother="kernel"` and `TRUE` if `smoother="local"`. See Details.
`breaks`	Breakpoints for the piecewise-constant function computed when `smoother='piecewise'`. Either a vector of numeric values specifying the breakpoints, or a single integer specifying the number of equally-spaced breakpoints. There is a sensible default.

Details

This command estimates the relationship between point process intensity and a given spatial covariate. Such a relationship is sometimes called a *resource selection function* (if the points are organisms and the covariate is a descriptor of habitat) or a *prospectivity index* (if the points are mineral deposits and the covariate is a geological variable). This command uses nonparametric methods which do not assume a particular form for the relationship.

If `object` is a point pattern, and `baseline` is missing or null, this command assumes that `object` is a realisation of a point process with intensity function $\lambda(u)$ of the form

$$\lambda(u) = \rho(Z(u))$$

where Z is the spatial covariate function given by `covariate`, and $\rho(z)$ is the resource selection function or prospectivity index. A nonparametric estimator of the function $\rho(z)$ is computed.

If `object` is a point pattern, and `baseline` is given, then the intensity function is assumed to be

$$\lambda(u) = \rho(Z(u))B(u)$$

where $B(u)$ is the baseline intensity at location u. A nonparametric estimator of the relative intensity $\rho(z)$ is computed.

If `object` is a fitted point process model, suppose `X` is the original data point pattern to which the model was fitted. Then this command assumes `X` is a realisation of a Poisson point process with intensity function of the form

$$\lambda(u) = \rho(Z(u))\kappa(u)$$

where $\kappa(u)$ is the intensity of the fitted model `object`. A nonparametric estimator of the relative intensity $\rho(z)$ is computed.

The nonparametric estimation procedure is controlled by the arguments `smoother`, `method` and `horvitz`.

The argument `smoother` selects the type of estimation technique.

- If `smoother="kernel"` (the default), the nonparametric estimator is a *kernel smoothing estimator* of $\rho(z)$ (Guan, 2008; Baddeley et al, 2012). The estimated function $\rho(z)$ will be a smooth function of z which takes nonnegative values. If `do.CI=TRUE` (the default), confidence bands are also computed, assuming a Poisson point process. See the section on *Smooth estimates*.
- If `smoother="local"`, the nonparametric estimator is a *local regression estimator* of $\rho(z)$ (Baddeley et al, 2012) obtained using local likelihood. The estimated function $\rho(z)$ will be a smooth function of z. If `do.CI=TRUE` (the default), confidence bands are also computed, assuming a Poisson point process. See the section on *Smooth estimates*.
- If `smoother="increasing"`, we assume that $\rho(z)$ is an increasing function of z, and use the *nonparametric maximum likelihood estimator* of $\rho(z)$ described by Sager (1982). The estimated function will be a step function, that is increasing as a function of z. Confidence bands are not computed. See the section on *Monotone estimates*.
- If `smoother="decreasing"`, we assume that $\rho(z)$ is a decreasing function of z, and use the *nonparametric maximum likelihood estimator* of $\rho(z)$ described by Sager (1982). The estimated function will be a step function, that is decreasing as a function of z. Confidence bands are not computed. See the section on *Monotone estimates*.
- If `smoother="mountain"`, we assume that $\rho(z)$ is a function with an inverted U shape, with a single peak at a value z_0, so that $\rho(z)$ is an increasing function of z for $z < z_0$ and a decreasing function of z for $z > z_0$. We compute the *nonparametric maximum likelihood estimator*. The estimated function will be a step function, which is increasing and then decreasing as a function of z. Confidence bands are not computed. See the section on *Unimodal estimates*.
- If `smoother="valley"`, we assume that $\rho(z)$ is a function with a U shape, with a single minimum at a value z_0, so that $\rho(z)$ is a decreasing function of z for $z < z_0$ and an increasing function of z for $z > z_0$. We compute the *nonparametric maximum likelihood estimator*. The estimated function will be a step function, which is decreasing and then increasing as a function of z. Confidence bands are not computed. See the section on *Unimodal estimates*.
- If `smoother="piecewise"`, the estimate of $\rho(z)$ is piecewise constant. The range of covariate values is divided into several intervals (ranges or bands). The endpoints of these intervals are the breakpoints, which may be specified by the argument `breaks`; there is a sensible default. The estimate of $\rho(z)$ takes a constant value on each interval. The estimate of $\rho(z)$ in each interval of covariate values is simply the average intensity (number of points per unit length) in the relevant sub-region of the network. If `do.CI=TRUE` (the default), confidence bands are also computed, assuming a Poisson point process.

See Baddeley (2018) for a comparison of these estimation techniques for two-dimensional point patterns.

If the argument `weights` is present, then the contribution from each data point `X[i]` to the estimate of ρ is multiplied by `weights[i]`.

If the argument `subset` is present, then the calculations are performed using only the data inside this spatial region.

This technique assumes that `covariate` has continuous values. It is not applicable to covariates with categorical (factor) values or discrete values such as small integers.

The argument `covariate` should be a pixel image, or a function, or one of the strings `"x"` or `"y"` signifying the cartesian coordinates. It will be evaluated on a fine grid of locations, with spatial resolution controlled by the arguments `eps,nd,random`. The argument `nd` specifies the total number of test locations on the linear network, `eps` specifies the linear separation between test locations, and `random` specifies whether the test locations have a randomised starting position.

Value

A function value table (object of class `"fv"`) containing the estimated values of ρ (and confidence limits) for a sequence of values of Z. Also belongs to the class `"rhohat"` which has special methods for `print`, `plot` and `predict`.

Smooth estimates

Smooth estimators of $\rho(z)$ were proposed by Baddeley and Turner (2005) and Baddeley et al (2012). Similar estimators were proposed by Guan (2008) and in the literature on relative distributions (Handcock and Morris, 1999).

The estimated function $\rho(z)$ will be a smooth function of z.

The smooth estimation procedure involves computing several density estimates and combining them. The algorithm used to compute density estimates is determined by `smoother`:

- If `smoother="kernel"`, the smoothing procedure is based on fixed-bandwidth kernel density estimation, performed by `density.default`.
- If `smoother="local"`, the smoothing procedure is based on local likelihood density estimation, performed by `locfit::locfit`.

The argument `method` determines how the density estimates will be combined to obtain an estimate of $\rho(z)$:

- If `method="ratio"`, then $\rho(z)$ is estimated by the ratio of two density estimates, The numerator is a (rescaled) density estimate obtained by smoothing the values $Z(y_i)$ of the covariate Z observed at the data points y_i. The denominator is a density estimate of the reference distribution of Z. See Baddeley et al (2012), equation (8). This is similar but not identical to an estimator proposed by Guan (2008).
- If `method="reweight"`, then $\rho(z)$ is estimated by applying density estimation to the values $Z(y_i)$ of the covariate Z observed at the data points y_i, with weights inversely proportional to the reference density of Z. See Baddeley et al (2012), equation (9).

- If `method="transform"`, the smoothing method is variable-bandwidth kernel smoothing, implemented by applying the Probability Integral Transform to the covariate values, yielding values in the range 0 to 1, then applying edge-corrected density estimation on the interval $[0, 1]$, and back-transforming. See Baddeley et al (2012), equation (10).

If `horvitz=TRUE`, then the calculations described above are modified by using Horvitz-Thompson weighting. The contribution to the numerator from each data point is weighted by the reciprocal of the baseline value or fitted intensity value at that data point; and a corresponding adjustment is made to the denominator.

If `do.CI=TRUE` (the default), pointwise confidence intervals for the true value of $\rho(z)$ are also calculated for each z, and will be plotted as grey shading. The confidence intervals are derived using the central limit theorem, based on variance calculations which assume a Poisson point process. If `positiveCI=FALSE`, the lower limit of the confidence interval may sometimes be negative, because the confidence intervals are based on a normal approximation to the estimate of $\rho(z)$. If `positiveCI=TRUE`, the confidence limits are always positive, because the confidence interval is based on a normal approximation to the estimate of $\log(\rho(z))$. For consistency with earlier versions, the default is `positiveCI=FALSE` for `smoother="kernel"` and `positiveCI=TRUE` for `smoother="local"`.

Monotone estimates

The nonparametric maximum likelihood estimator of a monotone function $\rho(z)$ was described by Sager (1982). This method assumes that $\rho(z)$ is either an increasing function of z, or a decreasing function of z. The estimated function will be a step function, increasing or decreasing as a function of z.

This estimator is chosen by specifying `smoother="increasing"` or `smoother="decreasing"`. The argument `method` is ignored this case.

To compute the estimate of $\rho(z)$, the algorithm first computes several primitive step-function estimates, and then takes the maximum of these primitive functions.

If `smoother="decreasing"`, each primitive step function takes the form $\rho(z) = \lambda$ when $z \leq t$, and $\rho(z) = 0$ when $z > t$, where and λ is a primitive estimate of intensity based on the data for $Z \leq t$. The jump location t will be the value of the covariate Z at one of the data points. The primitive estimate λ is the average intensity (number of points divided by area) for the region of space where the covariate value is less than or equal to t.

If `horvitz=TRUE`, then the calculations described above are modified by using Horvitz-Thompson weighting. The contribution to the numerator from each data point is weighted by the reciprocal of the baseline value or fitted intensity value at that data point; and a corresponding adjustment is made to the denominator.

Confidence intervals are not available for the monotone estimators.

Unimodal estimators

If `smoother="valley"` then we estimate a U-shaped function. A function $\rho(z)$ is U-shaped if it is decreasing when $z < z_0$ and increasing when $z > z_0$, where z_0 is called the critical value. The nonparametric maximum likelihood estimate of such a function can be computed by profiling over z_0. The algorithm considers all possible candidate values of the critical value z_0, and estimates the function $\rho(z)$ separately on the left and right of z_0 using the monotone estimators described above. These function estimates are combined into a single function, and the Poisson point process likelihood is computed. The optimal value of z_0 is the one which maximises the Poisson point process likelihood.

If `smoother="mountain"` then we estimate a function which has an inverted U shape. A function $\rho(z)$ is inverted-U-shaped if it is increasing when $z < z_0$ and decreasing when $z > z_0$. The nonparametric maximum likelihood estimate of such a function can be computed by profiling over z_0 using the same technique *mutatis mutandis*.

Confidence intervals are not available for the unimodal estimators.

Randomisation

By default, `rhohat` adds a small amount of random noise to the data. This is designed to suppress the effects of discretisation in pixel images.

This strategy means that `rhohat` does not produce exactly the same result when the computation is repeated. If you need the results to be exactly reproducible, set `jitter=FALSE` and `random=FALSE`.

The values of the covariate *at the data points* are randomly perturbed by adding a small amount of noise using the function `jitter`. To reduce this effect, set `jitterfactor` to a number smaller than 1. To suppress this effect entirely, set `jitter=FALSE`.

The values of the covariate *along the network* are sampled at a regularly-spaced grid on the network. The grid starts from a random position on each segment of the network. To suppress this behaviour, set `random=FALSE`.

Author(s)

Smoothing algorithm by Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>, Ya-Mei Chang, Yong Song, and Rolf Turner <`rolfturner@posteo.net`>.

Nonparametric maximum likelihood algorithm by Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>.

References

Baddeley, A., Chang, Y.-M., Song, Y. and Turner, R. (2012) Nonparametric estimation of the dependence of a point process on spatial covariates. *Statistics and Its Interface* **5** (2), 221–236.

Baddeley, A. and Turner, R. (2005) Modelling spatial point patterns in R. In: A. Baddeley, P. Gregori, J. Mateu, R. Stoica, and D. Stoyan, editors, *Case Studies in Spatial Point Pattern Modelling*, Lecture Notes in Statistics number 185. Pages 23–74. Springer-Verlag, New York, 2006. ISBN: 0-387-28311-0.

Baddeley, A. (2018) A statistical commentary on mineral prospectivity analysis. Chapter 2, pages 25–65 in *Handbook of Mathematical Geosciences: Fifty Years of IAMG*, edited by B.S. Daya Sagar, Q. Cheng and F.P. Agterberg. Springer, Berlin.

Guan, Y. (2008) On consistent nonparametric intensity estimation for inhomogeneous spatial point processes. *Journal of the American Statistical Association* **103**, 1238–1247.

Handcock, M.S. and Morris, M. (1999) *Relative Distribution Methods in the Social Sciences*. Springer, New York.

Sager, T.W. (1982) Nonparametric maximum likelihood estimation of spatial patterns. *Annals of Statistics* **10**, 1125–1136.

See Also

`rho2hat`, `methods.rhohat`, `parres`.

See `lppm` for a parametric method for the same problem.

Examples

```
Y <- runiflpp(30, simplenet)
rhoY <- rhohat(Y, "y")

## do spiders prefer to be in the middle of a segment?
teepee <- linfun(function(x,y,seg,tp){ tp }, domain(spiders))
rhotee <- rhohat(spiders, teepee)
rhoteeM <- rhohat(spiders, teepee, smoother="mountain")
if(interactive()) {
  plot(rhotee, main="Spider preference for mid-segment")
  plot(rhoteeM, add=TRUE, .y ~ .x, lwd=3)
}
```

`rjitter.lpp` *Random Perturbation of a Point Pattern on a Network*

Description

Applies independent random displacements to each point in a point pattern on a network.

Usage

```
## S3 method for class 'lpp'
rjitter(X, radius, ..., nsim = 1, drop = TRUE)
```

Arguments

`X`	A point pattern on a linear network (object of class `"lpp"`).
`radius`	Scale of perturbations. A positive numerical value. Each point will be displaced by a random distance, with maximum displacement equal to this value.
`...`	Ignored.
`nsim`	Number of simulated realisations to be generated.
`drop`	Logical. If `nsim=1` and `drop=TRUE` (the default), the result will be a point pattern, rather than a list containing a point pattern.

Details

The function `rjitter` is generic. This function is the method for the class `"lpp"` of point patterns on a linear network.

Each of the points in `X` will be displaced along the network by a random amount, independently of other points. The maximum displacement distance is specified by `radius`. Each point remains on the same line segment of the network as it originally was.

Value

A point pattern on a linear network (object of class `"lpp"`) or a list of such point patterns.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>, Rolf Turner <`rolfturner@posteo.net`> and Ege Rubak <`rubak@math.aau.dk`>.

See Also

`rjitter` for point patterns in two dimensions.

Examples

```
X <- runiflpp(3, simplenet)
plot(X, pch=16)
Y <- rjitter(X, 0.1)
plot(Y, add=TRUE, cols=3)
```

`rlpp` *Random Points on a Linear Network*

Description

Generates n independent random points on a linear network with a specified probability density.

Usage

```
rlpp(n, f, ..., nsim=1, drop=TRUE)
```

Arguments

`n`	Number of random points to generate. A nonnegative integer giving the number of points, or an integer vector giving the numbers of points of each type.
`f`	Probability density (not necessarily normalised). A pixel image on a linear network (object of class `"linim"`) or a function on a linear network (object of class `"linfun"`). Alternatively, `f` can be a list of functions or pixel images, giving the densities of points of each type.
`...`	Additional arguments passed to `f` if it is a function or a list of functions.
`nsim`	Number of simulated realisations to generate.
`drop`	Logical value indicating what to do when `nsim=1`. If `drop=TRUE` (the default), the result is a point pattern. If `drop=FALSE`, the result is a list with one entry which is a point pattern.

Details

The linear network `L`, on which the points will be generated, is determined by the argument `f`.

If `f` is a function, it is converted to a pixel image on the linear network, using any additional function arguments `...`.

If `n` is a single integer and `f` is a function or pixel image, then independent random points are generated on `L` with probability density proportional to `f`.

If `n` is an integer vector and `f` is a list of functions or pixel images, where `n` and `f` have the same length, then independent random points of several types are generated on `L`, with `n[i]` points of type `i` having probability density proportional to `f[[i]]`.

Value

If `nsim = 1` and `drop=TRUE`, a point pattern on the linear network, i.e.\ an object of class `"lpp"`. Otherwise, a list of such point patterns.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

See Also

`runiflpp`

Examples

```
g <- function(x, y, seg, tp) { exp(x + 3*y) }
f <- linfun(g, simplenet)

rlpp(20, f)

plot(rlpp(20, f, nsim=3))
```

`roc.lpp` *Receiver Operating Characteristic for Data on a Network*

Description

Computes the Receiver Operating Characteristic curve for a point pattern or a fitted point process model on a linear network.

Usage

```
## S3 method for class 'lpp'
roc(X, covariate, ..., high = TRUE)

## S3 method for class 'lppm'
roc(X, ...)
```

Arguments

`X`	Point pattern on a network (object of class `"lpp"`) or fitted point process model on a network (object of class `"lppm"`).
`covariate`	Spatial covariate. Either a `function(x,y)`, a pixel image (object of class `"im"` or `"linim"`), or one of the strings `"x"` or `"y"` indicating the Cartesian coordinates.
`...`	Arguments passed to `as.mask` controlling the pixel resolution for calculations.
`high`	Logical value indicating whether the threshold operation should favour high or low values of the covariate.

Details

The command `roc` computes the Receiver Operating Characteristic curve. The area under the ROC is computed by `auc`.

The function `roc` is generic, with methods for `"ppp"` and `"ppm"` described in the help file for `roc`.

This help file describes the methods for classes `"lpp"` and `"lppm"`.

For a point pattern `X` and a covariate `Z`, the ROC is a plot showing the ability of the covariate to separate the spatial domain into areas of high and low density of points. For each possible threshold z, the algorithm calculates the fraction $a(z)$ of area in the study region where the covariate takes a value greater than z, and the fraction $b(z)$ of data points for which the covariate value is greater than z. The ROC is a plot of $b(z)$ against $a(z)$ for all thresholds z.

For a fitted point process model, the ROC shows the ability of the fitted model intensity to separate the spatial domain into areas of high and low density of points. The ROC is **not** a diagnostic for the goodness-of-fit of the model (Lobo et al, 2007).

Value

Function value table (object of class `"fv"`) which can be plotted to show the ROC curve.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>, Rolf Turner <`rolfturner@posteo.net`> and Ege Rubak <`rubak@math.aau.dk`>.

References

Lobo, J.M., Jiménez-Valverde, A. and Real, R. (2007) AUC: a misleading measure of the performance of predictive distribution models. *Global Ecology and Biogeography* **17**(2) 145–151.

Nam, B.-H. and D'Agostino, R. (2002) Discrimination index, the area under the ROC curve. Pages 267–279 in Huber-Carol, C., Balakrishnan, N., Nikulin, M.S. and Mesbah, M., *Goodness-of-fit tests and model validity*, Birkhäuser, Basel.

See Also

`auc.lpp`

Examples

```
plot(roc(spiders, "x"))
fit <- lppm(spiders ~ x)
plot(roc(fit))
```

---

rpoislpp *Poisson Point Process on a Linear Network*

Description

Generates a realisation of the Poisson point process with specified intensity on the given linear network.

Usage

```
rpoislpp(lambda, L, ..., nsim=1, drop=TRUE, ex=NULL)
```

Arguments

`lambda`	Intensity of the Poisson process. A single number, a `function(x,y)`, a pixel image (object of class `"im"`), or a vector of numbers, a list of functions, or a list of images.
`L`	A linear network (object of class `"linnet"`, see `linnet`). Can be omitted in some cases: see Details.
`...`	Arguments passed to `rpoisppOnLines`.
`nsim`	Number of simulated realisations to generate.
`drop`	Logical value indicating what to do when `nsim=1`. If `drop=TRUE` (the default), the result is a point pattern. If `drop=FALSE`, the result is a list with one entry which is a point pattern.
`ex`	Optional. A point pattern on a network (object of class `"lpp"`) which serves as an example to determine the default values of `lambda` and `L`. See Details.

Details

A random number of random points is generated on the network `L`, according to a Poisson point process with intensity `lambda` points per unit length. The random points are generated by `rpoisppOnLines`. See the help file for `rpoisppOnLines` for information.

Argument `L` can be omitted, and defaults to `as.linnet(lambda)`, when `lambda` is a function on a linear network (class `"linfun"`) or a pixel image on a linear network (`"linim"`).

If `ex` is given, then it serves as an example for determining `lambda` and `L`. The default value of `lambda` will be the average intensity (number per unit length) of points in `ex` (or the average intensity of the points of each type if `ex` is multitype). The default value of `L` will be the network on which `ex` is defined.

Value

If `nsim = 1` and `drop=TRUE`, a point pattern on the linear network, i.e.\ an object of class `"lpp"`. Otherwise, a list of such point patterns.

Author(s)

Ang Qi Wei <`aqw07398@hotmail.com`> and Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>

See Also

`rpoisppOnLines`, `runiflpp`, `rlpp`, `lpp`, `linnet`.

Examples

```
X <- rpoislpp(5, simplenet)
plot(X)
# multitype
Y <- rpoislpp(c(a=5, b=5), simplenet)
# using argument 'ex' to make a pattern like 'X'
Z <- rpoislpp(ex=X)
```

`rSwitzerlpp` *Switzer-type Point Process on Linear Network*

Description

Generate a realisation of the Switzer-type point process on a linear network.

Usage

```
rSwitzerlpp(L, lambdacut, rintens = rexp, ...,
          cuts=c("points", "lines"))
```

Arguments

`L`	Linear network (object of class `"linnet"`).
`lambdacut`	Intensity of Poisson process of breakpoints.
`rintens`	Optional. Random variable generator used to generate the random intensity in each component.
`...`	Additional arguments to `rintens`.
`cuts`	String (partially matched) specifying the type of random cuts to be generated.

Details

This function generates simulated realisations of the Switzer-type point process on a network, as described in Baddeley et al (2017).

The linear network is first divided into pieces by a random mechanism:

- if `cuts="points"`, a Poisson process of breakpoints with intensity `lambdacut` is generated on the network, and these breakpoints separate the network into connected pieces.
- if `cuts="lines"`, a Poisson line process in the plane with intensity `lambdacut` is generated; these lines divide space into tiles; the network is divided into subsets associated with the tiles. Each subset may not be a connected sub-network.

In each piece of the network, a random intensity is generated using the random variable generator `rintens` (the default is a negative exponential random variable with rate 1). Given the intensity value, a Poisson process is generated with the specified intensity.

The intensity of the final process is determined by the mean of the values generated by `rintens`. If `rintens=rexp` (the default), then the parameter `rate` specifies the inverse of the intensity.

Value

Point pattern on a linear network (object of class `"lpp"`) with an attribute `"breaks"` containing the breakpoints (if `cuts="points"`) or the random lines (if `cuts="lines"`).

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`.

References

Baddeley, A., Nair, G., Rakshit, S. and McSwiggan, G. (2017) ‘Stationary’ point processes are uncommon on linear networks. *STAT* **6**, 68–78.

See Also

`rcelllpp`

Examples

```
plot(rSwitzerlpp(domain(spiders), 0.01, rate=100))

plot(rSwitzerlpp(domain(spiders), 0.0005, rate=100, cuts="l"))
```

`rThomaslpp` *Simulate Thomas Process on Linear Network*

Description

Generate a random point pattern, a realisation of the Thomas cluster process, on a linear network.

Usage

```
rThomaslpp(kappa, scale, mu, L, ..., nsim=1, drop=TRUE)
```

Arguments

`kappa`	Intensity of the Poisson process of cluster centres. A single positive number, a `function(x,y)`, or a pixel image (object of class `"im"` or `"linim"`).
`scale`	Standard deviation of random displacement (along the network) of a point from its cluster centre.
`mu`	Mean number of points per cluster (a single positive number) or reference intensity for the cluster points (a function or a pixel image).
`L`	Linear network (object of class `"linnet"`) on which the point pattern should be generated.
`...`	Arguments passed to `rpoisppOnLines`.
`nsim`	Number of simulated realisations to generate.
`drop`	Logical value indicating what to do when `nsim=1`. If `drop=TRUE` (the default), the result is a point pattern. If `drop=FALSE`, the result is a list with one entry which is a point pattern.

Details

This function generates realisations of the Thomas cluster process on a linear network, described by Baddeley et al (2017).

Argument `L` can be omitted, and defaults to `as.linnet(kappa)`, when `kappa` is a function on a linear network (class `"linfun"`) or a pixel image on a linear network (`"linim"`).

Value

A point pattern on a network (object of class `"lpp"`) or a list of point patterns on the network.

Author(s)

Greg McSwiggan and Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>.

References

Baddeley, A., Nair, G., Rakshit, S. and McSwiggan, G. (2017) 'Stationary' point processes are uncommon on linear networks. *STAT* **6** (1) 68–78.

Baddeley, A., Nair, G., Rakshit, S., McSwiggan, G. and Davies, T.M. (2021) Analysing point patterns on networks — a review. *Spatial Statistics* **42**, 100435, DOI 10.1016/j.spasta.2020.100435.

See Also

`rpoislpp`

Examples

```
plot(rThomaslpp(4, 0.07, 5, simplenet))
```

`runiflpp` *Uniform Random Points on a Linear Network*

Description

Generates n random points, independently and uniformly distributed, on a linear network.

Usage

```
runiflpp(n, L, nsim=1, drop=TRUE, ex=NULL)
```

Arguments

`n`	Number of random points to generate. A nonnegative integer, or a vector of integers specifying the number of points of each type.
`L`	A linear network (object of class `"linnet"`, see `linnet`).
`nsim`	Number of simulated realisations to generate.
`drop`	Logical value indicating what to do when `nsim=1`. If `drop=TRUE` (the default), the result is a point pattern. If `drop=FALSE`, the result is a list with one entry which is a point pattern.
`ex`	Optional. A point pattern on a network (object of class `"lpp"`) which serves as an example to determine the default values of `n` and `L`. See Details.

Details

The specified number `n` of random points is generated with uniform distribution on the network `L`. The random points are generated using `runifpointOnLines`.

If `n` is an integer vector, then a multitype point pattern is generated, with `n[i]` random points of type `i`.

If `ex` is given, then it serves as an example for determining `n` and `L`. The default value of `n` will be the number of points in `ex` (or the number of points of each type in `ex` if it is multitype). The default value of `L` will be the network on which `ex` is defined.

Value

If `nsim` = 1 and `drop=TRUE`, a point pattern on a linear network (object of class `"lpp"`). Otherwise, a list of such point patterns.

Author(s)

Ang Qi Wei `<aqw07398@hotmail.com>` and Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

See Also

`rlpp` for non-uniform random points; `rpoislpp` for Poisson point process;

`lpp`, `linnet`

Examples

```
 X <- runiflpp(10, simplenet)
 plot(X)
# marked
 Z <- runiflpp(c(a=10, b=3), simplenet)
# using 'ex'
 U <- runiflpp(ex=Z)
```

`sdr.lpp` *Sufficient Dimension Reduction for a Point Pattern on a Linear Network*

Description

Given a point pattern on a linear network, and a set of predictors, find a minimal set of new predictors, each constructed as a linear combination of the original predictors.

Usage

```
## S3 method for class 'lpp'
sdr(X, covariates,
             method = c("DR", "NNIR", "SAVE", "SIR", "TSE"),
             Dim1 = 1, Dim2 = 1, predict=FALSE, ...)
```

Arguments

`X`	A point pattern on a linear network (object of class `"lpp"`).
`covariates`	A list of pixel images (objects of class `"im"` or `"linim"`) to serve as predictor variables.
`method`	Character string indicating which method to use. See Details.
`Dim1`	Dimension of the first order Central Intensity Subspace (applicable when `method` is `"DR"`, `"NNIR"`, `"SAVE"` or `"TSE"`).

`Dim2`	Dimension of the second order Central Intensity Subspace (applicable when `method="TSE"`).
`predict`	Logical value indicating whether to compute the new predictors as well.
`...`	Extra arguments are ignored.

Details

This is the method for `sdr` for the class `"lpp"` of point patterns on a linear network.

Given a point pattern X and predictor variables $Z_1, \dots, Z_p$, Sufficient Dimension Reduction methods (Guan and Wang, 2010) attempt to find a minimal set of new predictor variables, each constructed by taking a linear combination of the original predictors, which explain the dependence of X on $Z_1, \dots, Z_p$. The methods do not assume any particular form of dependence of the point pattern on the predictors. The predictors are assumed to be Gaussian random fields.

Available methods are:

`method="DR"`	directional regression
`method="NNIR"`	nearest neighbour inverse regression
`method="SAVE"` & sliced average variance estimation	
`method="SIR"` & sliced inverse regression	
`method="TSE"` & two-step estimation	

The result includes a matrix `B` whose columns are estimates of the basis vectors of the space of new predictors. That is, the `j`th column of `B` expresses the `j`th new predictor as a linear combination of the original predictors.

If `predict=TRUE`, the new predictors are also evaluated. They can also be evaluated using `sdrPredict`.

Value

A list with components `B, M` or `B, M1, M2` where `B` is a matrix whose columns are estimates of the basis vectors for the space, and `M` or `M1,M2` are matrices containing estimates of the kernel.

If `predict=TRUE`, the result also includes a component `Y` which is a list of pixel images giving the values of the new predictors.

Author(s)

Based on a Matlab original, for two-dimensional point patterns, by Yongtao Guan. Adapted to R, and to linear networks, by Suman Rakshit.

References

Guan, Y. and Wang, H. (2010) Sufficient dimension reduction for spatial point processes directed by Gaussian random fields. *Journal of the Royal Statistical Society, Series B*, **72**, 367–387.

See Also

`sdrPredict` to compute the new predictors from the coefficient matrix.

`dimhat` to estimate the subspace dimension.

`subspaceDistance`

Examples

```
#   sdr(bei, bei.extra)

   xim <- as.linim(function(x,y) { x }, simplenet)
   yim <- as.linim(function(x,y) { y }, simplenet)
   X <- runiflpp(30, simplenet)
   sdr(X, list(x=xim, y=yim))
```

`simulate.lppm` *Simulate a Fitted Point Process Model on a Linear Network*

Description

Generates simulated realisations from a fitted Poisson point process model on a linear network.

Usage

```
  ## S3 method for class 'lppm'
simulate(object, nsim=1, ...,
                         new.coef=NULL,
                         progress=(nsim > 1),
                         drop=FALSE)
```

Arguments

`object`	Fitted point process model on a linear network. An object of class `"lppm"`.
`nsim`	Number of simulated realisations.
`progress`	Logical flag indicating whether to print progress reports for the sequence of simulations.
`new.coef`	New values for the canonical parameters of the model. A numeric vector of the same length as `coef(object)`.
`...`	Arguments passed to `predict.lppm` to determine the spatial resolution of the image of the fitted intensity used in the simulation.
`drop`	Logical. If `nsim=1` and `drop=TRUE`, the result will be a point pattern, rather than a list containing a point pattern.

Details

This function is a method for the generic function `simulate` for the class `"lppm"` of fitted point process models on a linear network.

Only Poisson process models are supported so far.

Simulations are performed by `rpoislpp`.

Value

A list of length `nsim` containing simulated point patterns (objects of class `"lpp"`) on the same linear network as the original data used to fit the model. The result also belongs to the class `"solist"`, so that it can be plotted, and the class `"timed"`, so that the total computation time is recorded.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

, Rolf Turner `<rolfturner@posteo.net>`

and Ege Rubak `<rubak@math.aau.dk>`

See Also

`lppm`, `rpoislpp`, `simulate`

Examples

```
fit <- lppm(unmark(chicago) ~ y)
simulate(fit)[[1]]
```

`Smooth.lpp` *Spatial Smoothing of Observations on a Network*

Description

Performs spatial smoothing of numeric values observed at a set of locations on a network. Uses kernel smoothing.

Usage

```
## S3 method for class 'lpp'
Smooth(X, sigma,
                     ...,
                     at=c("pixels", "points"),
                     weights=rep(1, npoints(X)),
                     leaveoneout=TRUE)
```

Arguments

`X`	A marked point pattern on a linear network (object of class `"lpp"`).
`sigma`	Smoothing bandwidth. A single positive number. See `density.lpp`.
`...`	Further arguments passed to `density.lpp` to control the kernel smoothing and the pixel resolution of the result.
`at`	String specifying whether to compute the smoothed values at a grid of pixel locations (`at="pixels"`) or only at the points of `X` (`at="points"`).
`weights`	Optional numeric vector of weights attached to the observations.
`leaveoneout`	Logical value indicating whether to compute a leave-one-out estimator. Applicable only when `at="points"`.

Details

The function `Smooth.lpp` performs spatial smoothing of numeric values observed at a set of irregular locations on a linear network.

`Smooth.lpp` is a method for the generic function `Smooth` for the class `"lpp"` of point patterns. Thus you can type simply `Smooth(X)`.

Smoothing is performed by kernel weighting, using the Gaussian kernel by default. If the observed values are $v_1, \ldots, v_n$ at locations $x_1, \ldots, x_n$ respectively, then the smoothed value at a location u is

$$g(u) = \frac{\sum_i k(u, x_i) v_i}{\sum_i k(u, x_i)}$$

where k is the kernel. This is known as the Nadaraya-Watson smoother (Nadaraya, 1964, 1989; Watson, 1964). The type of kernel is determined by further arguments `...` which are passed to `density.lpp`

The argument `X` must be a marked point pattern on a linear network (object of class `"lpp"`). The points of the pattern are taken to be the observation locations x_i, and the marks of the pattern are taken to be the numeric values v_i observed at these locations.

The marks are allowed to be a data frame. Then the smoothing procedure is applied to each column of marks.

The numerator and denominator are computed by `density.lpp`. The arguments `...` control the smoothing kernel parameters.

The optional argument `weights` allows numerical weights to be applied to the data. If a weight w_i is associated with location x_i, then the smoothed function is (ignoring edge corrections)

$$g(u) = \frac{\sum_i k(u, x_i) v_i w_i}{\sum_i k(u, x_i) w_i}$$

Value

If `X` *has a single column of marks:*

- If `at="pixels"` (the default), the result is a pixel image on the network (object of class `"linim"`). Pixel values are values of the interpolated function.

- If `at="points"`, the result is a numeric vector of length equal to the number of points in `X`. Entries are values of the interpolated function at the points of `X`.

If `X` *has a data frame of marks:*

- If `at="pixels"` (the default), the result is a named list of pixel images on the network (objects of class `"linim"`). There is one image for each column of marks. This list also belongs to the class `"solist"`, for which there is a plot method.
- If `at="points"`, the result is a data frame with one row for each point of `X`, and one column for each column of marks. Entries are values of the interpolated function at the points of `X`.

The return value has attribute `"sigma"` which reports the smoothing bandwidth that was used.

Very small bandwidth

If the chosen bandwidth `sigma` is very small, kernel smoothing is mathematically equivalent to nearest-neighbour interpolation.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>, Rolf Turner <`rolfturner@posteo.net`> and Ege Rubak <`rubak@math.aau.dk`>.

References

Nadaraya, E.A. (1964) On estimating regression. *Theory of Probability and its Applications* **9**, 141–142.

Nadaraya, E.A. (1989) *Nonparametric estimation of probability densities and regression curves*. Kluwer, Dordrecht.

Watson, G.S. (1964) Smooth regression analysis. *Sankhya A* **26**, 359–372.

See Also

`Smooth`, `density.lpp`.

Examples

```
X <- spiders
if(!interactive()) X <- X[owin(c(0,1100), c(0, 500))]
marks(X) <- coords(X)$x
plot(Smooth(X, 50))
Smooth(X, 50, at="points")
```

`subset.lpp` *Subset of Point Pattern Satisfying A Condition*

Description

Given a point pattern on a linear network, return the subset of points which satisfy a specified condition.

Usage

```
## S3 method for class 'lpp'
subset(x, subset, select, drop=FALSE, ...)
```

Arguments

`x`	A point pattern on a linear network (object of class `"lpp"`).
`subset`	Logical expression indicating which points are to be kept. The expression may involve the names of spatial coordinates (`x`, `y`), network coordinates (`seg`, `tp`), the `marks`, and (if there is more than one column of marks) the names of individual columns of marks. Missing values are taken as false. See Details.
`select`	Expression indicating which columns of marks should be kept. The *names* of columns of marks can be used in this expression, and will be treated as if they were column indices. See Details.
`drop`	Logical value indicating whether to remove unused levels of the marks, if the marks are a factor.
`...`	Ignored.

Details

This is a method for the generic function `subset`. It extracts the subset of points of `x` that satisfy the logical expression `subset`, and retains only the columns of marks that are specified by the expression `select`. The result is always a point pattern, with the same window as `x`.

The argument `subset` determines the subset of points that will be extracted. It should be a logical expression. It may involve the variable names `x` and `y` representing the Cartesian coordinates; the names of other spatial coordinates or local coordinates; the name `marks` representing the marks; and (if there is more than one column of marks) the names of individual columns of marks. The default is to keep all points.

The argument `select` determines which columns of marks will be retained (if there are several columns of marks). It should be an expression involving the names of columns of marks (which will be interpreted as integers representing the positions of these columns). For example if there are columns of marks named `A` to `Z`, then `select=D:F` is a valid expression and means that columns `D`, `E` and `F` will be retained. Similarly `select=-(A:C)` is valid and means that columns `A` to `C` will be deleted. The default is to retain all columns.

Setting `subset=FALSE` will produce an empty point pattern (i.e. containing zero points) in the same window as `x`. Setting `select=FALSE` or `select= -marks` will remove all the marks from `x`.

The argument `drop` determines whether to remove unused levels of a factor, if the resulting point pattern is multitype (i.e. the marks are a factor) or if the marks are a data frame in which some of the columns are factors.

The result is always a point pattern, of the same class as `x`. Spatial coordinates (and local coordinates) are always retained. To extract only some columns of marks or coordinates as a data frame, use `subset(as.data.frame(x), ...)`

Value

A point pattern of the same class as `x`, in the same spatial window as `x`. The result is a subset of `x`, possibly with some columns of marks removed.

Other kinds of subset arguments

Alternatively the argument `subset` can be any kind of subset index acceptable to `[.lpp`. This argument selects which points of `x` will be retained.

Warning: if the argument `subset` is a window, this is interpreted as specifying the subset of points that fall inside that window, but the resulting point pattern has the same window as the original pattern `x`.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>, Rolf Turner <`rolfturner@posteo.net`> and Ege Rubak <`rubak@math.aau.dk`>.

See Also

`subset.ppp`, `[.lpp`.

Examples

```
v <- subset(chicago, x + y > 1100 & marks == "assault")

vv <- subset(chicago, x + y > 1100 & marks == "assault", drop=TRUE)
```

`superimpose.lpp` *Superimpose Several Point Patterns on Linear Network*

Description

Superimpose any number of point patterns on the same linear network.

Usage

```
  ## S3 method for class 'lpp'
superimpose(..., L=NULL)
```

Arguments

`...`	Any number of arguments, each of which represents a point pattern on the same linear network. Each argument can be either an object of class `"lpp"`, giving both the spatial coordinates of the points and the linear network, or a `list(x,y)` or `list(x,y,seg,tp)` giving just the spatial coordinates of the points.
`L`	Optional. The linear network. An object of class `"linnet"`. This argument is required if none of the other arguments is of class `"lpp"`.

Details

This function is used to superimpose several point patterns on the same linear network. It is a method for the generic function `superimpose`.

Each of the arguments `...` can be either a point pattern on a linear network (object of class `"lpp"` giving both the spatial coordinates of the points and the linear network), or a `list(x,y)` or `list(x,y,seg,tp)` giving just the spatial coordinates of the points. These arguments must represent point patterns on the *same* linear network.

The argument `L` is an alternative way to specify the linear network, and is required if none of the arguments `...` is an object of class `"lpp"`.

The arguments `...` may be *marked* patterns. The marks of each component pattern must have the same format. Numeric and character marks may be “mixed”. If there is such mixing then the numeric marks are coerced to character in the combining process. If the mark structures are all data frames, then these data frames must have the same number of columns and identical column names.

If the arguments `...` are given in the form `name=value`, then the `names` will be used as an extra column of marks attached to the elements of the corresponding patterns.

Value

An object of class `"lpp"` representing the combined point pattern on the linear network.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>

Rolf Turner <`rolfturner@posteo.net`>

Ege Rubak <`rubak@math.aau.dk`>

and Greg McSwiggan.

See Also

`superimpose`

Examples

```
X <- rpoislpp(5, simplenet)
Y <- rpoislpp(10, simplenet)
superimpose(X,Y) # not marked
superimpose(A=X, B=Y) # multitype with types A and B
```

`terminalvertices` *Terminal Vertices of a Linear Network*

Description

Finds the terminal vertices of a linear network.

Usage

```
terminalvertices(L)
```

Arguments

`L`	A linear network (object of class `"linnet"`).

Details

Given the linear network `L`, this function examines the vertices (segment endpoints) of the network and determines which of them are 'terminal' vertices (i.e. the endpoint of only one segment). These terminal vertices are returned as a point pattern on the network.

Value

A point pattern on the same linear network (object of class `"lpp"`).

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>, Rolf Turner <`rolfturner@posteo.net`> and Ege Rubak <`rubak@math.aau.dk`> and Mehdi Moradi <`m2.moradi@yahoo.com`>.

See Also

`vertices.linnet`.

Examples

```
B <- terminalvertices(simplenet)
plot(simplenet, main="")
plot(B, add=TRUE, pch=16, cex=2)
```

`text.lpp` *Add Text Labels to Point Pattern on a Network*

Description

Plots a text label at the location of each point, for a point pattern on a linear network.

Usage

```
## S3 method for class 'lpp'
text(x, ...)
```

Arguments

`x`	A point pattern on a linear network (class `"lpp"`).
`...`	Additional arguments passed to `text.default`.

Details

This function is a method for the generic `text`. A text label is added to the existing plot, at the location of each point in the point pattern `x`, or near the location of the midpoint of each segment in the segment pattern `x`.

Additional arguments `...` are passed to `text.default` and may be used to control the placement of the labels relative to the point locations, and the size and colour of the labels.

By default, the labels are the serial numbers 1 to `n`, where `n` is the number of points or segments in `x`. This can be changed by specifying the argument `labels`, which should be a vector of length `n`.

Value

Null.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`, Rolf Turner `<rolfturner@posteo.net>` and Ege Rubak `<rubak@math.aau.dk>`.

See Also

`text.default`, `text.ppp`

Examples

```
X <- runiflpp(5, simplenet)
plot(X)
text(X, pos=2, col="blue")
```

`thinNetwork` *Remove Vertices or Segments from a Linear Network*

Description

Delete some vertices and/or segments from a linear network or related object.

Usage

```
thinNetwork(X, retainvertices=NULL, retainedges=NULL)
```

Arguments

`X`	A linear network (object of class `"linnet"`), a point pattern on a linear network (object of class `"lpp"`) or a pixel image on a linear network (object of class `"linim"`).
`retainvertices`	Optional. Subset index specifying which vertices should be retained (not deleted).
`retainedges`	Optional. Subset index specifying which edges (segments) should be retained (not deleted).

Details

This function deletes some of the vertices and edges (segments) in the linear network.

The arguments `retainvertices` and `retainedges` can be any kind of subset index: a vector of positive integers specifying which vertices/edges should be retained; a vector of negative integers specifying which vertices/edges should be deleted; or a logical vector specifying whether each vertex/edge should be retained (`TRUE`) or deleted (`FALSE`).

Vertices are indexed in the same sequence as in `vertices(as.linnet(X))`. Segments are indexed in the same sequence as in `as.psp(as.linnet(X))`.

The argument `retainedges` has higher precedence than `retainvertices` in the sense that:

- If `retainedges` is given, then any vertex which is an endpoint of a retained edge will also be retained.
- If `retainvertices` is given and `retainedges` is **missing**, then any segment joining two retained vertices will also be retained.
- Thus, when both `retainvertices` and `retainedges` are given, it is possible that more vertices will be retained than those specified by `retainvertices`.

After the network has been altered, other consequential changes will occur, including renumbering of the segments and vertices. If `X` is a point pattern on a linear network, then data points will be deleted if they lie on a deleted edge. If `X` is a pixel image on a linear network, then the image will be restricted to the new sub-network.

Value

An object of the same kind as `X`.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`> and Suman Rakshit.

See Also

`linnet` to make a network;

`connected.linnet` to extract connected components.

`repairNetwork`.

Examples

```
L <- simplenet
plot(L, main="thinNetwork(L, retainededges=c(-3, -5))")
text(midpoints.psp(as.psp(L)), labels=1:nsegments(L), pos=3)
Lsub <- thinNetwork(L, retainededges=c(-3, -5))
plot(Lsub, add=TRUE, col="blue", lwd=2)
```

`tile.lengths` *Compute Lengths of Tiles in a Tessellation on a Network*

Description

Computes the length of each tile in a tessellation on a linear network.

Usage

```
tile.lengths(x)
```

Arguments

`x`	A tessellation on a linear network (object of class `"lintess"`).

Details

A tessellation on a linear network `L` is a partition of the network into non-overlapping pieces (tiles). Each tile consists of one or more line segments which are subsets of the line segments making up the network. A tile can consist of several disjoint pieces.

This command computes the length of each of the tiles that make up the tessellation `x`. The result is a numeric vector.

Value

A numeric vector.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>, Rolf Turner <`rolfturner@posteo.net`> and Ege Rubak <`rubak@math.aau.dk`>.

See Also

`lintess`

Examples

```
X <- runiflpp(5, simplenet)
A <- lineardirichlet(X)
plot(A)
tile.lengths(A)
```

`tilenames.lintess` *Names of Tiles in a Tessellation on a Network*

Description

Extract or Change the Names of the Tiles in a Tessellation on a Network.

Usage

```
## S3 method for class 'lintess'
tilenames(x)

## S3 replacement method for class 'lintess'
tilenames(x) <- value
```

Arguments

`x`	A tessellation on a linear network (object of class `"lintess"`).
`value`	Character vector giving new names for the tiles.

Details

These functions extract or change the names of the tiles that make up the tessellation `x`.

If the tessellation is a regular grid, the tile names cannot be changed.

Value

`tilenames` returns a character vector.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`, Rolf Turner `<rolfturner@posteo.net>` and Ege Rubak `<rubak@math.aau.dk>`.

See Also

`lintess`, `tiles`

Examples

```
B <- lineardirichlet(runiflpp(5, simplenet))
tilenames(B)
tilenames(B) <- letters[1:5]
```

`treebranchlabels` *Label Vertices of a Tree by Branch Membership*

Description

Given a linear network which is a tree (acyclic graph), this function assigns a label to each vertex, indicating its position in the tree.

Usage

```
treebranchlabels(L, root = 1)
```

Arguments

`L`	Linear network (object of class `"linnet"`). The network must have no loops.
`root`	Root of the tree. An integer index identifying which point in `vertices(L)` is the root of the tree.

Details

The network `L` should be a tree, that is, it must have no loops.

This function computes a character string label for each vertex of the network `L`. The vertex identified by `root` (that is, `vertices(L)[root]`) is taken as the root of the tree and is given the empty label `""`.

- If there are several line segments which meet at the root vertex, each of these segments is the start of a new branch of the tree; the other endpoints of these segments are assigned the labels `"a"`, `"b"`, `"c"` and so on.
- If only one segment issues from the root vertex, the other endpoint of this segment is assigned the empty label `""`.

A similar rule is then applied to each of the newly-labelled vertices. If the vertex labelled `"a"` is joined to two other unlabelled vertices, these will be labelled `"aa"` and `"ab"`. The rule is applied recursively until all vertices have been labelled.

If `L` is not a tree, the algorithm will terminate, but the results will be nonsense.

Value

A vector of character strings, with one entry for each point in `vertices(L)`.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>, Rolf Turner <`rolfturner@posteo.net`> and Ege Rubak <`rubak@math.aau.dk`>

See Also

`deletebranch`, `extractbranch`, `treeprune` for manipulating a network using the branch labels.

`linnet` for creating a network.

Examples

```
# make a simple tree
m <- simplenet$m
m[8,10] <- m[10,8] <- FALSE
L <- linnet(vertices(simplenet), m)
plot(L, main="")
# compute branch labels
tb <- treebranchlabels(L, 1)
tbc <- paste0("[", tb, "]")
text(vertices(L), labels=tbc, cex=2)
```

`treeprune` *Prune Tree to Given Level*

Description

Prune a tree by removing all the branches above a given level.

Usage

```
treeprune(X, root = 1, level = 0)
```

Arguments

`X`	Object of class `"linnet"` or `"lpp"`.
`root`	Index of the root vertex amongst the vertices of `as.linnet(X)`.
`level`	Integer specifying the level above which the tree should be pruned.

Details

The object `X` must be either a linear network, or a derived object such as a point pattern on a linear network. The linear network must be an acyclic graph (i.e. must not contain any loops) so that it can be interpreted as a tree.

This function removes all vertices for which `treebranchlabels` gives a string more than `level` characters long.

Value

Object of the same kind as `X`.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>, Rolf Turner <`rolfturner@posteo.net`> and Ege Rubak <`rubak@math.aau.dk`>

See Also

`treebranchlabels` for calculating the branch labels.

`deletebranch` for removing entire branches. `extractbranch` for extracting entire branches.

`linnet` for creating networks.

Examples

```
# make a simple tree
m <- simplenet$m
m[8,10] <- m[10,8] <- FALSE
L <- linnet(vertices(simplenet), m)
plot(L, main="")
# compute branch labels
tb <- treebranchlabels(L, 1)
tbc <- paste0("[", tb, "]")
text(vertices(L), labels=tbc, cex=2)
# prune tree
tp <- treeprune(L, root=1, 1)
plot(tp, add=TRUE, col="blue", lwd=3)
```

`unstack.lpp` *Separate Multiple Columns of Marks*

Description

Given a spatial pattern on a network, with several columns of marks, take one column at a time, and return a list of spatial patterns each having only one column of marks.

Usage

```
## S3 method for class 'lpp'
unstack(x, ...)

## S3 method for class 'lintess'
unstack(x, ...)
```

Arguments

`x`	A spatial point pattern (object of class `"lpp"`) or a tessellation on a linear network (object of class `"lintess"`).
`...`	Ignored.

Details

The functions defined here are methods for the generic `unstack`. The functions expect a spatial object `x` which has several columns of marks; they separate the columns, and return a list of spatial objects, each having only one column of marks.

If `x` has several columns of marks (i.e. `marks(x)` is a matrix, data frame or hyperframe with several columns), then `y <- unstack(x)` is a list of spatial objects, each of the same kind as `x`. The `j`th entry `y[[j]]` is equivalent to `x` except that it only includes the `j`th column of `marks(x)`.

If `x` has no marks, or has only a single column of marks, the result is a list consisting of one entry, which is `x`.

Value

A list, of class `"solist"`, whose entries are objects of the same type as `x`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

See Also

`unstack`

`unstack.ppp`, `unstack.msr`.

See also methods for the generic `split` such as `split.ppx` which applies to `"lpp"` objects.

Examples

```
X <- runiflpp(5, simplenet)
marks(X) <- data.frame(id=1:5, code=factor(letters[1:5]))
unstack(X)
```

`Window.lpp` *Extract Window of Spatial Object on a Network*

Description

Given a spatial object on a network, these functions extract the window in which the network is defined.

Usage

```
 ## S3 method for class 'lpp'
Window(X, ...)

 ## S3 method for class 'lppm'
Window(X, ...)
```

Arguments

`X`	A spatial object.
`...`	Ignored.

Details

These are methods for the generic function `Window` which extract the spatial window in which the object `X` is defined.

For the methods defined here, `X` should be a spatial object on a linear network (object of class `"lpp"` or `"lppm"`).

Value

An object of class `"owin"` (see `owin.object`) specifying an observation window.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>, Rolf Turner <`rolfturner@posteo.net`> and Ege Rubak <`rubak@math.aau.dk`>.

See Also

`Window`.

Examples

```
Window(spiders)
```

Index