

Package ‘spatstat’

September 22, 2024

Version 3.2-1

Date 2024-09-22

Title Spatial Point Pattern Analysis, Model-Fitting, Simulation, Tests

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

Depends R (>= 3.5.0), spatstat.data (>= 3.1-0), spatstat.univar (>= 3.0-0), spatstat.geom (>= 3.3-3), spatstat.random (>= 3.3-2), spatstat.explore (>= 3.3-2), spatstat.model (>= 3.3-2), spatstat.linnet (>= 3.2-2), utils

Imports spatstat.utils (>= 3.1-0)

Description Comprehensive open-source toolbox for analysing Spatial Point Patterns. Focused mainly on two-dimensional point patterns, including multitype/marked points, in any spatial region. Also supports three-dimensional point patterns, space-time point patterns in any number of dimensions, point patterns on a linear network, and patterns of other geometrical objects. Supports spatial covariate data such as pixel images. Contains over 3000 functions for plotting spatial data, exploratory data analysis, model-fitting, simulation, spatial sampling, model diagnostics, and formal inference. Data types include point patterns, line segment patterns, spatial windows, pixel images, tessellations, and linear networks. Exploratory methods include quadrat counts, K-functions and their simulation envelopes, nearest neighbour distance and empty space statistics, Fry plots, pair correlation function, kernel smoothed intensity, relative risk estimation with cross-validated bandwidth selection, mark correlation functions, segregation indices, mark dependence diagnostics, and kernel estimates of covariate effects. 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 functions `ppm()`, `kppm()`, `slrm()`, `dppm()` similar to `glm()`. Types of models include Poisson, Gibbs and Cox point processes, Neyman-Scott cluster processes, and determinantal point processes. Models may involve dependence on covariates, inter-point interaction, cluster formation and dependence on marks. Models are fitted by maximum likelihood, logistic regression, minimum contrast, and composite likelihood methods. A model can be fitted to a list of point patterns (replicated point pattern data) using the function `mppm()`. The model can include random effects and fixed effects depending on the experimental design, in addition to all the features listed above.

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.

License GPL (>= 2)

URL <http://spatstat.org/>

NeedsCompilation yes

ByteCompile true

BugReports <https://github.com/spatstat/spatstat/issues>

Author Adrian Baddeley [aut, cre] (<<https://orcid.org/0000-0001-9499-8382>>),
Rolf Turner [aut] (<<https://orcid.org/0000-0001-5521-5218>>),
Ege Rubak [aut] (<<https://orcid.org/0000-0002-6675-533X>>)

Repository CRAN

Date/Publication 2024-09-22 04:20:02 UTC

Contents

spatstat-package	2
beginner	30
bugfixes	31
foo	33
latest.changes	34
latest.news	35
spatstat.family	37

Index **39**

spatstat-package *The Spatstat Package*

Description

This is a summary of the features of **spatstat**, a family of R packages for the statistical analysis of spatial point patterns.

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.

spatstat is designed to support a complete statistical analysis of spatial data. It supports

- creation, manipulation and plotting of point patterns;

- exploratory data analysis;
- spatial random sampling;
- simulation of point process models;
- parametric model-fitting;
- non-parametric smoothing and regression;
- formal inference (hypothesis tests, confidence intervals);
- model diagnostics.

Apart from two-dimensional point patterns and point processes, **spatstat** also supports point patterns in three dimensions, point patterns in multidimensional space-time, point patterns on a linear network, patterns of line segments in two dimensions, and spatial tessellations and random sets in two dimensions.

The package can fit several types of point process models to a point pattern dataset:

- Poisson point process models (by Berman-Turner approximate maximum likelihood or by spatial logistic regression)
- Gibbs/Markov point process models (by Baddeley-Turner approximate maximum pseudolikelihood, Coeurjolly-Rubak logistic likelihood, or Huang-Ogata approximate maximum likelihood)
- Cox/cluster point process models (by Waagepetersen's two-step fitting procedure and minimum contrast, composite likelihood, or Palm likelihood)
- determinantal point process models (by Waagepetersen's two-step fitting procedure and minimum contrast, composite likelihood, or Palm likelihood)

The models may include spatial trend, dependence on covariates, and complicated interpoint interactions. Models are specified by a formula in the R language, and are fitted using a function analogous to `lm` and `glm`. Fitted models can be printed, plotted, predicted, simulated and so on.

Getting Started

For a quick introduction to **spatstat**, read the package vignette *Getting started with spatstat* installed with **spatstat**. To read that document, you can either

- visit <https://cran.r-project.org/package=spatstat> and click on Getting Started with Spatstat
- start R, type `library(spatstat)` and `vignette('getstart')`
- start R, type `help.start()` to open the help browser, and navigate to Packages > spatstat > Vignettes.

Once you have installed **spatstat**, start R and type `library(spatstat)`. Then type `beginner` for a beginner's introduction, or `demo(spatstat)` for a demonstration of the package's capabilities.

For a complete course on **spatstat**, and on statistical analysis of spatial point patterns, read the book by Baddeley, Rubak and Turner (2015). Other recommended books on spatial point process methods are Diggle (2014), Gelfand et al (2010) and Illian et al (2008).

The **spatstat** package includes over 50 datasets, which can be useful when learning the package. Type `demo(data)` to see plots of all datasets available in the package. Type `vignette('datasets')` for detailed background information on these datasets, and plots of each dataset.

For information on converting your data into **spatstat** format, read Chapter 3 of Baddeley, Rubak and Turner (2015). This chapter is available free online, as one of the sample chapters at the book companion website, <https://book.spatstat.org/>.

Structure of the spatstat family

The original **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 functionality for geometrical operations, and defining the main classes of spatial objects
- **spatstat.explore** containing the main functions for exploratory analysis of spatial data
- **spatstat.model** containing the main functions for parametric statistical modelling and analysis, and formal 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.

The breakup has been done in such a way that the user should not notice any difference. Source code that worked with the old **spatstat** package should work with the new **spatstat** family. Code that is documented in our books, journal articles and vignettes should still work.

When you install **spatstat**, the sub-packages listed above 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.

This help file covers all the functionality and datasets that are provided in the sub-packages listed above.

Extension packages

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.

Updates

New versions of **spatstat** are released every 8 weeks. Users are advised to update their installation of **spatstat** regularly.

Type `latest.news` to read the news documentation about changes to the current installed version of **spatstat**.

See the Vignette *Summary of recent updates*, installed with **spatstat**, which describes the main changes to **spatstat** since the book (Baddeley, Rubak and Turner, 2015) was published. It is accessible as `vignette('updates')`.

Type `news(package="spatstat")` to read news documentation about all previous versions of the package.

FUNCTIONS AND DATASETS

Following is a summary of the main functions and datasets in the **spatstat** package. Alternatively an alphabetical list of all functions and datasets is available by typing `library(help=spatstat)`.

For further information on any of these, type `help(name)` or `?name` where `name` is the name of the function or dataset.

CONTENTS:

- I. Creating and manipulating data
- II. Exploratory Data Analysis
- III. Model fitting (Cox and cluster models)
- IV. Model fitting (Poisson and Gibbs models)
- V. Model fitting (determinantal point processes)
- VI. Model fitting (spatial logistic regression)
- VII. Simulation
- VIII. Tests and diagnostics
- IX. Documentation

I. CREATING AND MANIPULATING DATA

Types of spatial data:

The main types of spatial data supported by **spatstat** are:

<code>ppp</code>	point pattern
<code>owin</code>	window (spatial region)
<code>im</code>	pixel image
<code>psp</code>	line segment pattern
<code>tess</code>	tessellation
<code>pp3</code>	three-dimensional point pattern
<code>ppx</code>	point pattern in any number of dimensions
<code>lpp</code>	point pattern on a linear network

To create a point pattern:

<code>ppp</code>	create a point pattern from (x, y) and window information <code>ppp(x, y, xlim, ylim)</code> for rectangular window <code>ppp(x, y, poly)</code> for polygonal window <code>ppp(x, y, mask)</code> for binary image window
<code>as.ppp</code>	convert other types of data to a ppp object
<code>clickppp</code>	interactively add points to a plot
<code>marks<-, %mark%</code>	attach/reassign marks to a point pattern

To simulate a random point pattern:

<code>runifpoint</code>	generate n independent uniform random points
<code>rpoint</code>	generate n independent random points
<code>rmpoint</code>	generate n independent multitype random points
<code>rpoispp</code>	simulate the (in)homogeneous Poisson point process
<code>rmpoispp</code>	simulate the (in)homogeneous multitype Poisson point process
<code>runifdisc</code>	generate n independent uniform random points in disc
<code>rstrat</code>	stratified random sample of points
<code>rsyst</code>	systematic random sample of points
<code>rjitter</code>	apply random displacements to points in a pattern
<code>rMaternI</code>	simulate the Matérn Model I inhibition process
<code>rMaternII</code>	simulate the Matérn Model II inhibition process
<code>rSSI</code>	simulate Simple Sequential Inhibition process
<code>rStrauss</code>	simulate Strauss process (perfect simulation)
<code>rHardcore</code>	simulate Hard Core process (perfect simulation)
<code>rStraussHard</code>	simulate Strauss-hard core process (perfect simulation)
<code>rDiggleGratton</code>	simulate Diggle-Gratton process (perfect simulation)
<code>rDGS</code>	simulate Diggle-Gates-Stibbard process (perfect simulation)
<code>rPenttinen</code>	simulate Penttinen process (perfect simulation)
<code>rNeymanScott</code>	simulate a general Neyman-Scott process
<code>rPoissonCluster</code>	simulate a general Poisson cluster process
<code>rMatClust</code>	simulate the Matérn Cluster process
<code>rThomas</code>	simulate the Thomas process
<code>rGaussPoisson</code>	simulate the Gauss-Poisson cluster process
<code>rCauchy</code>	simulate Neyman-Scott Cauchy cluster process
<code>rVarGamma</code>	simulate Neyman-Scott Variance Gamma cluster process
<code>rthin</code>	random thinning
<code>rcell</code>	simulate the Baddeley-Silverman cell process
<code>rmh</code>	simulate Gibbs point process using Metropolis-Hastings
<code>simulate.ppm</code>	simulate Gibbs point process using Metropolis-Hastings
<code>runifpointOnLines</code>	generate n random points along specified line segments
<code>rpoisppOnLines</code>	generate Poisson random points along specified line segments

To randomly change an existing point pattern:

<code>rshift</code>	random shifting of points
<code>rjitter</code>	apply random displacements to points in a pattern

<code>rthin</code>	random thinning
<code>rlabel</code>	random (re)labelling of a multitype point pattern
<code>quadratresample</code>	block resampling

Standard point pattern datasets:

Datasets in **spatstat** are lazy-loaded, so you can simply type the name of the dataset to use it; there is no need to type `data(amacrine)` etc.

Type `demo(data)` to see a display of all the datasets installed with the package.

Type `vignette('datasets')` for a document giving an overview of all datasets, including background information, and plots.

<code>amacrine</code>	Austin Hughes' rabbit amacrine cells
<code>anemones</code>	Upton-Fingleton sea anemones data
<code>ants</code>	Harkness-Isham ant nests data
<code>bdspots</code>	Breakdown spots in microelectrodes
<code>bei</code>	Tropical rainforest trees
<code>betacells</code>	Waessle et al. cat retinal ganglia data
<code>bramblecanes</code>	Bramble Canes data
<code>bronzefilter</code>	Bronze Filter Section data
<code>cells</code>	Crick-Ripley biological cells data
<code>chicago</code>	Chicago crimes
<code>chorley</code>	Chorley-Ribble cancer data
<code>clmfires</code>	Castilla-La Mancha forest fires
<code>copper</code>	Berman-Huntington copper deposits data
<code>dendrite</code>	Dendritic spines
<code>demohyper</code>	Synthetic point patterns
<code>demopat</code>	Synthetic point pattern
<code>finpines</code>	Finnish Pines data
<code>flu</code>	Influenza virus proteins
<code>gordon</code>	People in Gordon Square, London
<code>gorillas</code>	Gorilla nest sites
<code>hamster</code>	Aherne's hamster tumour data
<code>humberside</code>	North Humberside childhood leukaemia data
<code>hyytiala</code>	Mixed forest in Hyytiälä, Finland
<code>japanesepines</code>	Japanese Pines data
<code>lansing</code>	Lansing Woods data
<code>longleaf</code>	Longleaf Pines data
<code>mucosa</code>	Cells in gastric mucosa
<code>murchison</code>	Murchison gold deposits
<code>nbfires</code>	New Brunswick fires data
<code>nztrees</code>	Mark-Esler-Ripley trees data
<code>osteo</code>	Osteocyte lacunae (3D, replicated)
<code>paracou</code>	Kimboto trees in Paracou, French Guiana
<code>ponderosa</code>	Getis-Franklin ponderosa pine trees data
<code>pyramidal</code>	Pyramidal neurons from 31 brains
<code>redwood</code>	Strauss-Ripley redwood saplings data
<code>redwoodfull</code>	Strauss redwood saplings data (full set)

<code>residualspaper</code>	Data from Baddeley et al (2005)
<code>shapley</code>	Galaxies in an astronomical survey
<code>simdat</code>	Simulated point pattern (inhomogeneous, with interaction)
<code>spiders</code>	Spider webs on mortar lines of brick wall
<code>sporophores</code>	Mycorrhizal fungi around a tree
<code>spruces</code>	Spruce trees in Saxonia
<code>swedishpines</code>	Strand-Ripley Swedish pines data
<code>urkiola</code>	Urkiola Woods data
<code>waka</code>	Trees in Waka national park
<code>waterstriders</code>	Insects on water surface

To display a 2D point pattern:

<code>plot.ppp</code>	plot a point pattern (e.g. <code>plot(X)</code>)
<code>spatstat.gui::iplot</code>	plot a point pattern interactively
<code>persp.ppp</code>	perspective plot of marked point pattern

To manipulate a 2D point pattern:

<code>edit.ppp</code>	interactive text editor
<code>[.ppp]</code>	extract or replace a subset of a point pattern <code>pp[subset]</code> or <code>pp[subwindow]</code>
<code>subset.ppp</code>	extract subset of point pattern satisfying a condition
<code>superimpose</code>	combine several point patterns
<code>by.ppp</code>	apply a function to sub-patterns of a point pattern
<code>cut.ppp</code>	classify the points in a point pattern
<code>split.ppp</code>	divide pattern into sub-patterns
<code>unmark</code>	remove marks
<code>npoints</code>	count the number of points
<code>coords</code>	extract coordinates, change coordinates
<code>marks</code>	extract marks, change marks or attach marks
<code>rotate</code>	rotate pattern
<code>shift</code>	translate pattern
<code>flipxy</code>	swap x and y coordinates
<code>reflect</code>	reflect in the origin
<code>periodify</code>	make several translated copies
<code>affine</code>	apply affine transformation
<code>scalardilate</code>	apply scalar dilation
<code>density.ppp</code>	kernel estimation of point pattern intensity
<code>densityHeat.ppp</code>	diffusion kernel estimation of point pattern intensity
<code>Smooth.ppp</code>	kernel smoothing of marks of point pattern
<code>nnmark</code>	mark value of nearest data point
<code>sharpen.ppp</code>	data sharpening
<code>identify.ppp</code>	interactively identify points
<code>unique.ppp</code>	remove duplicate points
<code>duplicated.ppp</code>	determine which points are duplicates
<code>uniquemap.ppp</code>	map duplicated points to unique points
<code>connected.ppp</code>	find clumps of points

<code>dirichlet</code>	compute Dirichlet-Voronoi tessellation
<code>delaunay</code>	compute Delaunay triangulation
<code>delaunayDistance</code>	graph distance in Delaunay triangulation
<code>convexhull</code>	compute convex hull
<code>discretise</code>	discretise coordinates
<code>pixellate.ppp</code>	approximate point pattern by pixel image
<code>as.im.ppp</code>	approximate point pattern by pixel image

See `spatstat.options` to control plotting behaviour.

To create a window:

An object of class "owin" describes a spatial region (a window of observation).

<code>owin</code>	Create a window object <code>owin(xlim, ylim)</code> for rectangular window <code>owin(poly)</code> for polygonal window <code>owin(mask)</code> for binary image window
<code>Window</code>	Extract window of another object
<code>Frame</code>	Extract the containing rectangle ('frame') of another object
<code>as.owin</code>	Convert other data to a window object
<code>square</code>	make a square window
<code>disc</code>	make a circular window
<code>ellipse</code>	make an elliptical window
<code>ripras</code>	Ripley-Rasson estimator of window, given only the points
<code>convexhull</code>	compute convex hull of something
<code>letterR</code>	polygonal window in the shape of the R logo
<code>clickpoly</code>	interactively draw a polygonal window
<code>clickbox</code>	interactively draw a rectangle

To manipulate a window:

<code>plot.owin</code>	plot a window. <code>plot(W)</code>
<code>boundingbox</code>	Find a tight bounding box for the window
<code>erosion</code>	erode window by a distance r
<code>dilation</code>	dilate window by a distance r
<code>closing</code>	close window by a distance r
<code>opening</code>	open window by a distance r
<code>border</code>	difference between window and its erosion/dilation
<code>complement.owin</code>	invert (swap inside and outside)
<code>simplify.owin</code>	approximate a window by a simple polygon
<code>rotate</code>	rotate window
<code>flipxy</code>	swap x and y coordinates
<code>shift</code>	translate window
<code>periodify</code>	make several translated copies
<code>affine</code>	apply affine transformation
<code>as.data.frame.owin</code>	convert window to data frame

Digital approximations:

<code>as.mask</code>	Make a discrete pixel approximation of a given window
<code>as.im.owin</code>	convert window to pixel image
<code>pixellate.owin</code>	convert window to pixel image
<code>commonGrid</code>	find common pixel grid for windows
<code>nearest.raster.point</code>	map continuous coordinates to raster locations
<code>raster.x</code>	raster x coordinates
<code>raster.y</code>	raster y coordinates
<code>raster.xy</code>	raster x and y coordinates
<code>as.polygonal</code>	convert pixel mask to polygonal window

See `spatstat.options` to control the approximation

Geometrical computations with windows:

<code>edges</code>	extract boundary edges
<code>intersect.owin</code>	intersection of two windows
<code>union.owin</code>	union of two windows
<code>setminus.owin</code>	set subtraction of two windows
<code>inside.owin</code>	determine whether a point is inside a window
<code>area.owin</code>	compute area
<code>perimeter</code>	compute perimeter length
<code>diameter.owin</code>	compute diameter
<code>incircle</code>	find largest circle inside a window
<code>inradius</code>	radius of incircle
<code>connected.owin</code>	find connected components of window
<code>eroded.areas</code>	compute areas of eroded windows
<code>dilated.areas</code>	compute areas of dilated windows
<code>bdist.points</code>	compute distances from data points to window boundary
<code>bdist.pixels</code>	compute distances from all pixels to window boundary
<code>bdist.tiles</code>	boundary distance for each tile in tessellation
<code>distmap.owin</code>	distance transform image
<code>distfun.owin</code>	distance transform
<code>centroid.owin</code>	compute centroid (centre of mass) of window
<code>is.subset.owin</code>	determine whether one window contains another
<code>is.convex</code>	determine whether a window is convex
<code>convexhull</code>	compute convex hull
<code>triangulate.owin</code>	decompose into triangles
<code>as.mask</code>	pixel approximation of window
<code>as.polygonal</code>	polygonal approximation of window
<code>is.rectangle</code>	test whether window is a rectangle
<code>is.polygonal</code>	test whether window is polygonal
<code>is.mask</code>	test whether window is a mask
<code>setcov</code>	spatial covariance function of window
<code>pixelcentres</code>	extract centres of pixels in mask
<code>clickdist</code>	measure distance between two points clicked by user

Pixel images: An object of class "im" represents a pixel image. Such objects are returned by some of the functions in **spatstat** including `Kmeasure`, `setcov` and `density.ppp`.

<code>im</code>	create a pixel image
<code>as.im</code>	convert other data to a pixel image
<code>pixellate</code>	convert other data to a pixel image
<code>as.matrix.im</code>	convert pixel image to matrix
<code>as.data.frame.im</code>	convert pixel image to data frame
<code>as.function.im</code>	convert pixel image to function
<code>plot.im</code>	plot a pixel image on screen as a digital image
<code>contour.im</code>	draw contours of a pixel image
<code>persp.im</code>	draw perspective plot of a pixel image
<code>rgbim</code>	create colour-valued pixel image
<code>hsvim</code>	create colour-valued pixel image
<code>[.im</code>	extract a subset of a pixel image
<code>[<-.im</code>	replace a subset of a pixel image
<code>rotate.im</code>	rotate pixel image
<code>shift.im</code>	apply vector shift to pixel image
<code>affine.im</code>	apply affine transformation to image
<code>X</code>	print very basic information about image X
<code>summary(X)</code>	summary of image X
<code>hist.im</code>	histogram of image
<code>mean.im</code>	mean pixel value of image
<code>integral.im</code>	integral of pixel values
<code>quantile.im</code>	quantiles of image
<code>cut.im</code>	convert numeric image to factor image
<code>is.im</code>	test whether an object is a pixel image
<code>interp.im</code>	interpolate a pixel image
<code>blur</code>	apply Gaussian blur to image
<code>Smooth.im</code>	apply Gaussian blur to image
<code>connected.im</code>	find connected components
<code>compatible.im</code>	test whether two images have compatible dimensions
<code>harmonise.im</code>	make images compatible
<code>commonGrid</code>	find a common pixel grid for images
<code>eval.im</code>	evaluate any expression involving images
<code>im.apply</code>	evaluate a function of several images
<code>scaletointerval</code>	rescale pixel values
<code>zapsmall.im</code>	set very small pixel values to zero
<code>levelset</code>	level set of an image
<code>solutionset</code>	region where an expression is true
<code>imcov</code>	spatial covariance function of image
<code>convolve.im</code>	spatial convolution of images
<code>transect.im</code>	line transect of image
<code>pixelcentres</code>	extract centres of pixels
<code>transmat</code>	convert matrix of pixel values to a different indexing convention
<code>rnoise</code>	random pixel noise

Line segment patterns

An object of class "psp" represents a pattern of straight line segments.

<code>psp</code>	create a line segment pattern
<code>as.psp</code>	convert other data into a line segment pattern
<code>edges</code>	extract edges of a window
<code>is.psp</code>	determine whether a dataset has class "psp"
<code>plot.psp</code>	plot a line segment pattern
<code>print.psp</code>	print basic information
<code>summary.psp</code>	print summary information
<code>[.psp</code>	extract a subset of a line segment pattern
<code>subset.psp</code>	extract subset of line segment pattern
<code>as.data.frame.psp</code>	convert line segment pattern to data frame
<code>marks.psp</code>	extract marks of line segments
<code>marks<- .psp</code>	assign new marks to line segments
<code>unmark.psp</code>	delete marks from line segments
<code>midpoints.psp</code>	compute the midpoints of line segments
<code>endpoints.psp</code>	extract the endpoints of line segments
<code>lengths_psp</code>	compute the lengths of line segments
<code>angles.psp</code>	compute the orientation angles of line segments
<code>superimpose</code>	combine several line segment patterns
<code>flipxy</code>	swap x and y coordinates
<code>rotate.psp</code>	rotate a line segment pattern
<code>shift.psp</code>	shift a line segment pattern
<code>periodify</code>	make several shifted copies
<code>affine.psp</code>	apply an affine transformation
<code>pixellate.psp</code>	approximate line segment pattern by pixel image
<code>psp2mask</code>	approximate line segment pattern by binary mask
<code>distmap.psp</code>	compute the distance map of a line segment pattern
<code>distfun.psp</code>	compute the distance map of a line segment pattern
<code>density.psp</code>	kernel smoothing of line segments
<code>selfcrossing.psp</code>	find crossing points between line segments
<code>selfcut.psp</code>	cut segments where they cross
<code>crossing.psp</code>	find crossing points between two line segment patterns
<code>extrapolate.psp</code>	extrapolate line segments to infinite lines
<code>nncross</code>	find distance to nearest line segment from a given point
<code>nearestsegment</code>	find line segment closest to a given point
<code>project2segment</code>	find location along a line segment closest to a given point
<code>pointsOnLines</code>	generate points evenly spaced along line segment
<code>rpoisline</code>	generate a realisation of the Poisson line process inside a window
<code>rlinegrid</code>	generate a random array of parallel lines through a window

Tessellations

An object of class "tess" represents a tessellation.

<code>tess</code>	create a tessellation
<code>quadrats</code>	create a tessellation of rectangles
<code>hextess</code>	create a tessellation of hexagons
<code>polar tess</code>	tessellation using polar coordinates
<code>quantess</code>	quantile tessellation
<code>venn . tess</code>	Venn diagram tessellation

<code>dirichlet</code>	compute Dirichlet-Voronoi tessellation of points
<code>del aunay</code>	compute Delaunay triangulation of points
<code>as.tess</code>	convert other data to a tessellation
<code>plot.tess</code>	plot a tessellation
<code>tiles</code>	extract all the tiles of a tessellation
<code>[.tess</code>	extract some tiles of a tessellation
<code>[<-.tess</code>	change some tiles of a tessellation
<code>intersect.tess</code>	intersect two tessellations or restrict a tessellation to a window
<code>chop.tess</code>	subdivide a tessellation by a line
<code>rpoislinetess</code>	generate tessellation using Poisson line process
<code>tile.areas</code>	area of each tile in tessellation
<code>bdist.tiles</code>	boundary distance for each tile in tessellation
<code>connected.tess</code>	find connected components of tiles
<code>shift.tess</code>	shift a tessellation
<code>rotate.tess</code>	rotate a tessellation
<code>reflect.tess</code>	reflect about the origin
<code>flipxy.tess</code>	reflect about the diagonal
<code>affine.tess</code>	apply affine transformation

Three-dimensional point patterns

An object of class "pp3" represents a three-dimensional point pattern in a rectangular box. The box is represented by an object of class "box3".

<code>pp3</code>	create a 3-D point pattern
<code>plot.pp3</code>	plot a 3-D point pattern
<code>coords</code>	extract coordinates
<code>as.hyperframe</code>	extract coordinates
<code>subset.pp3</code>	extract subset of 3-D point pattern
<code>unitname.pp3</code>	name of unit of length
<code>npoints</code>	count the number of points
<code>runifpoint3</code>	generate uniform random points in 3-D
<code>rpoispp3</code>	generate Poisson random points in 3-D
<code>envelope.pp3</code>	generate simulation envelopes for 3-D pattern
<code>box3</code>	create a 3-D rectangular box
<code>as.box3</code>	convert data to 3-D rectangular box
<code>unitname.box3</code>	name of unit of length
<code>diameter.box3</code>	diameter of box
<code>volume.box3</code>	volume of box
<code>shortside.box3</code>	shortest side of box
<code>eroded.volumes</code>	volumes of erosions of box

Multi-dimensional space-time point patterns

An object of class "ppx" represents a point pattern in multi-dimensional space and/or time.

<code>ppx</code>	create a multidimensional space-time point pattern
<code>coords</code>	extract coordinates

<code>as.hyperframe</code>	extract coordinates
<code>subset.ppx</code>	extract subset
<code>unitname.ppx</code>	name of unit of length
<code>npoints</code>	count the number of points
<code>runifpointx</code>	generate uniform random points
<code>rpoisppx</code>	generate Poisson random points
<code>boxx</code>	define multidimensional box
<code>diameter.boxx</code>	diameter of box
<code>volume.boxx</code>	volume of box
<code>shortside.boxx</code>	shortest side of box
<code>eroded.volumes.boxx</code>	volumes of erosions of box

Point patterns on a linear network

An object of class "linnet" represents a linear network (for example, a road network).

<code>linnet</code>	create a linear network
<code>clickjoin</code>	interactively join vertices in network
<code>spatstat.gui::iplot.linnet</code>	interactively plot network
<code>simplenet</code>	simple example of network
<code>lineardisc</code>	disc in a linear network
<code>delaunayNetwork</code>	network of Delaunay triangulation
<code>dirichletNetwork</code>	network of Dirichlet edges
<code>methods.linnet</code>	methods for linnet objects
<code>vertices.linnet</code>	nodes of network
<code>joinVertices</code>	join existing vertices in a network
<code>insertVertices</code>	insert new vertices at positions along a network
<code>addVertices</code>	add new vertices, extending a network
<code>thinNetwork</code>	remove vertices or lines from a network
<code>repairNetwork</code>	repair internal format
<code>pixellate.linnet</code>	approximate by pixel image

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

<code>lpp</code>	create a point pattern on a linear network
<code>methods.lpp</code>	methods for lpp objects
<code>subset.lpp</code>	method for subset
<code>rpoislpp</code>	simulate Poisson points on linear network
<code>runiflpp</code>	simulate random points on a linear network
<code>chicago</code>	Chicago crime data
<code>dendrite</code>	Dendritic spines data
<code>spiders</code>	Spider webs on mortar lines of brick wall

Hyperframes

A hyperframe is like a data frame, except that the entries may be objects of any kind.

<code>hyperframe</code>	create a hyperframe
<code>as.hyperframe</code>	convert data to hyperframe
<code>plot.hyperframe</code>	plot hyperframe
<code>with.hyperframe</code>	evaluate expression using each row of hyperframe
<code>cbind.hyperframe</code>	combine hyperframes by columns
<code>rbind.hyperframe</code>	combine hyperframes by rows
<code>as.data.frame.hyperframe</code>	convert hyperframe to data frame
<code>subset.hyperframe</code>	method for subset
<code>head.hyperframe</code>	first few rows of hyperframe
<code>tail.hyperframe</code>	last few rows of hyperframe

Layered objects

A layered object represents data that should be plotted in successive layers, for example, a background and a foreground.

<code>layered</code>	create layered object
<code>plot.layered</code>	plot layered object
<code>[.layered</code>	extract subset of layered object

Colour maps

A colour map is a mechanism for associating colours with data. It can be regarded as a function, mapping data to colours. Using a colourmap object in a plot command ensures that the mapping from numbers to colours is the same in different plots.

<code>colourmap</code>	create a colour map
<code>plot.colourmap</code>	plot the colour map only
<code>tweak.colourmap</code>	alter individual colour values
<code>interp.colourmap</code>	make a smooth transition between colours
<code>beachcolourmap</code>	one special colour map

II. EXPLORATORY DATA ANALYSIS

Inspection of data:

<code>summary(X)</code>	print useful summary of point pattern X
<code>X</code>	print basic description of point pattern X
<code>any(duplicated(X))</code>	check for duplicated points in pattern X
<code>spatstat.gui::istat(X)</code>	Interactive exploratory analysis
<code>spatstat.gui::View.ppp(X)</code>	spreadsheet-style viewer

Classical exploratory tools:

<code>clarkevans</code>	Clark and Evans aggregation index
<code>fryplot</code>	Fry plot
<code>miplot</code>	Morisita Index plot

Smoothing:

<code>density.ppp</code>	kernel smoothed density/intensity
<code>relrisk</code>	kernel estimate of relative risk
<code>Smooth.ppp</code>	spatial interpolation of marks
<code>bw.diggle</code>	cross-validated bandwidth selection for <code>density.ppp</code>
<code>bw.ppl</code>	likelihood cross-validated bandwidth selection for <code>density.ppp</code>
<code>bw.CvL</code>	Cronie-Van Lieshout bandwidth selection for density estimation
<code>bw.scott</code>	Scott's rule of thumb for density estimation
<code>bw.abram</code>	Abramson's rule for adaptive bandwidths
<code>bw.relrisk</code>	cross-validated bandwidth selection for <code>relrisk</code>
<code>bw.smoothppp</code>	cross-validated bandwidth selection for <code>Smooth.ppp</code>
<code>bw.frac</code>	bandwidth selection using window geometry
<code>bw.stoyan</code>	Stoyan's rule of thumb for bandwidth for <code>pcf</code>

Modern exploratory tools:

<code>clusterset</code>	Allard-Fraley feature detection
<code>nnclean</code>	Byers-Raftery feature detection
<code>sharpen.ppp</code>	Choi-Hall data sharpening
<code>rhohat</code>	Kernel estimate of covariate effect
<code>rho2hat</code>	Kernel estimate of effect of two covariates
<code>spatialcdf</code>	Spatial cumulative distribution function
<code>roc</code>	Receiver operating characteristic curve

Summary statistics for a point pattern: Type `demo(sumfun)` for a demonstration of many of the summary statistics.

<code>intensity</code>	Mean intensity
<code>quadratcount</code>	Quadrat counts
<code>intensity.quadratcount</code>	Mean intensity in quadrats
<code>Fest</code>	empty space function F
<code>Gest</code>	nearest neighbour distribution function G
<code>Jest</code>	J -function $J = (1 - G)/(1 - F)$
<code>Kest</code>	Ripley's K -function
<code>Lest</code>	Besag L -function
<code>Tstat</code>	Third order T -function
<code>allstats</code>	all four functions F, G, J, K
<code>pcf</code>	pair correlation function
<code>Kinhom</code>	K for inhomogeneous point patterns
<code>Linhom</code>	L for inhomogeneous point patterns
<code>pcfinhom</code>	pair correlation for inhomogeneous patterns
<code>Finhom</code>	F for inhomogeneous point patterns
<code>Ginhom</code>	G for inhomogeneous point patterns
<code>Jinhom</code>	J for inhomogeneous point patterns
<code>locall</code>	Getis-Franklin neighbourhood density function
<code>localK</code>	neighbourhood K -function
<code>localpcf</code>	local pair correlation function
<code>localKinhom</code>	local K for inhomogeneous point patterns
<code>localLinhom</code>	local L for inhomogeneous point patterns

localpcfinhom	local pair correlation for inhomogeneous patterns
Ksector	Directional K -function
Kscaled	locally scaled K -function
Kest.fft	fast K -function using FFT for large datasets
Kmeasure	reduced second moment measure
envelope	simulation envelopes for a summary function
varblock	variances and confidence intervals for a summary function
lohboot	bootstrap for a summary function

Related facilities:

plot.fv	plot a summary function
eval.fv	evaluate any expression involving summary functions
harmonise.fv	make functions compatible
eval.fasp	evaluate any expression involving an array of functions
with.fv	evaluate an expression for a summary function
Smooth.fv	apply smoothing to a summary function
deriv.fv	calculate derivative of a summary function
pool.fv	pool several estimates of a summary function
nnDIST	nearest neighbour distances
nnwhich	find nearest neighbours
pairdist	distances between all pairs of points
crossdist	distances between points in two patterns
nncross	nearest neighbours between two point patterns
exactdt	distance from any location to nearest data point
distmap	distance map image
distfun	distance map function
nnmap	nearest point image
nnfun	nearest point function
density.ppp	kernel smoothed density
densityHeat.ppp	diffusion kernel smoothed density
Smooth.ppp	spatial interpolation of marks
relrisk	kernel estimate of relative risk
sharpen.ppp	data sharpening
rknn	theoretical distribution of nearest neighbour distance

Summary statistics for a multitype point pattern: A multitype point pattern is represented by an object X of class "ppp" such that $\text{marks}(X)$ is a factor.

relrisk	kernel estimation of relative risk
scan.test	spatial scan test of elevated risk
Gcross, Gdot, Gmulti	multitype nearest neighbour distributions $G_{ij}, G_{i\bullet}$
Kcross, Kdot, Kmulti	multitype K -functions $K_{ij}, K_{i\bullet}$
Lcross, Ldot	multitype L -functions $L_{ij}, L_{i\bullet}$
Jcross, Jdot, Jmulti	multitype J -functions $J_{ij}, J_{i\bullet}$
pcfcross	multitype pair correlation function g_{ij}
pcfdot	multitype pair correlation function $g_{i\bullet}$

<code>pcfmulti</code>	general pair correlation function
<code>markconnect</code>	marked connection function p_{ij}
<code>alltypes</code>	estimates of the above for all i, j pairs
<code>Iest</code>	multitype I -function
<code>Kcross.inhom, Kdot.inhom</code>	inhomogeneous counterparts of <code>Kcross</code> , <code>Kdot</code>
<code>Lcross.inhom, Ldot.inhom</code>	inhomogeneous counterparts of <code>Lcross</code> , <code>Ldot</code>
<code>pcfcross.inhom, pcfdot.inhom</code>	inhomogeneous counterparts of <code>pcfcross</code> , <code>pcfdot</code>
<code>localKcross, localKdot</code>	local counterparts of <code>Kcross</code> , <code>Kdot</code>
<code>localLcross, localLdot</code>	local counterparts of <code>Lcross</code> , <code>Ldot</code>
<code>localKcross.inhom, localLcross.inhom</code>	local counterparts of <code>Kcross.inhom</code> , <code>Lcross.inhom</code>

Summary statistics for a marked point pattern: A marked point pattern is represented by an object `X` of class "ppp" with a component `X$marks`. The entries in the vector `X$marks` may be numeric, complex, string or any other atomic type. For numeric marks, there are the following functions:

<code>markmean</code>	smoothed local average of marks
<code>markvar</code>	smoothed local variance of marks
<code>markcorr</code>	mark correlation function
<code>markcrosscorr</code>	mark cross-correlation function
<code>markvario</code>	mark variogram
<code>markmarkscatter</code>	mark-mark scatterplot
<code>Kmark</code>	mark-weighted K function
<code>Emark</code>	mark independence diagnostic $E(r)$
<code>Vmark</code>	mark independence diagnostic $V(r)$
<code>nnmean</code>	nearest neighbour mean index
<code>nnvario</code>	nearest neighbour mark variance index

For marks of any type, there are the following:

<code>Gmulti</code>	multitype nearest neighbour distribution
<code>Kmulti</code>	multitype K -function
<code>Jmulti</code>	multitype J -function

Alternatively use `cut.ppp` to convert a marked point pattern to a multitype point pattern.

Programming tools:

<code>applynbd</code>	apply function to every neighbourhood in a point pattern
<code>markstat</code>	apply function to the marks of neighbours in a point pattern
<code>marktable</code>	tabulate the marks of neighbours in a point pattern
<code>pppdist</code>	find the optimal match between two point patterns

Summary statistics for a point pattern on a linear network:

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

<code>lineark</code>	K function on linear network
----------------------	--------------------------------

<code>linearKinhom</code>	inhomogeneous K function on linear network
<code>linearpcf</code>	pair correlation function on linear network
<code>linearpcfinhom</code>	inhomogeneous pair correlation on linear network

For multitype patterns:

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

Related facilities:

<code>pairedist.lpp</code>	distances between pairs
<code>crossdist.lpp</code>	distances between pairs
<code>nndist.lpp</code>	nearest neighbour distances
<code>nncross.lpp</code>	nearest neighbour distances
<code>nnwhich.lpp</code>	find nearest neighbours
<code>nnfun.lpp</code>	find nearest data point
<code>density.lpp</code>	kernel smoothing estimator of intensity
<code>densityHeat.lpp</code>	diffusion kernel estimate
<code>distfun.lpp</code>	distance transform
<code>envelope.lpp</code>	simulation envelopes
<code>rpoislpp</code>	simulate Poisson points on linear network
<code>runiflpp</code>	simulate random points on a linear network

It is also possible to fit point process models to lpp objects. See Section IV.

Summary statistics for a three-dimensional point pattern:

These are for 3-dimensional point pattern objects (class pp3).

<code>F3est</code>	empty space function F
<code>G3est</code>	nearest neighbour function G
<code>K3est</code>	K -function
<code>pcf3est</code>	pair correlation function

Related facilities:

<code>envelope.pp3</code>	simulation envelopes
<code>pairedist.pp3</code>	distances between all pairs of points
<code>crossdist.pp3</code>	distances between points in two patterns
<code>nndist.pp3</code>	nearest neighbour distances

<code>nnwhich.pp3</code>	find nearest neighbours
<code>nncross.pp3</code>	find nearest neighbours in another pattern

Computations for multi-dimensional point pattern:

These are for multi-dimensional space-time point pattern objects (class `ppx`).

<code>pairdist.ppx</code>	distances between all pairs of points
<code>crossdist.ppx</code>	distances between points in two patterns
<code>nndist.ppx</code>	nearest neighbour distances
<code>nnwhich.ppx</code>	find nearest neighbours

Summary statistics for random sets:

These work for point patterns (class `ppp`), line segment patterns (class `psp`) or windows (class `owin`).

<code>Hest</code>	spherical contact distribution H
<code>Gfox</code>	Foxall G -function
<code>Jfox</code>	Foxall J -function

III. MODEL FITTING (COX AND CLUSTER MODELS)

Cluster process models (with homogeneous or inhomogeneous intensity) and Cox processes can be fitted by the function `kppm`. Its result is an object of class "kppm". The fitted model can be printed, plotted, predicted, simulated and updated.

<code>kppm</code>	Fit model
<code>plot.kppm</code>	Plot the fitted model
<code>summary.kppm</code>	Summarise the fitted model
<code>fitted.kppm</code>	Compute fitted intensity
<code>predict.kppm</code>	Compute fitted intensity
<code>update.kppm</code>	Update the model
<code>improve.kppm</code>	Refine the estimate of trend
<code>simulate.kppm</code>	Generate simulated realisations
<code>vcov.kppm</code>	Variance-covariance matrix of coefficients
<code>coef.kppm</code>	Extract trend coefficients
<code>formula.kppm</code>	Extract trend formula
<code>parameters</code>	Extract all model parameters
<code>clusterfield.kppm</code>	Compute offspring density
<code>clusterradius.kppm</code>	Radius of support of offspring density
<code>Kmodel.kppm</code>	K function of fitted model
<code>pcfmodel.kppm</code>	Pair correlation of fitted model

For model selection, you can also use the generic functions `step`, `drop1` and `AIC` on fitted point process models. For variable selection, see `sdr`.

The theoretical models can also be simulated, for any choice of parameter values, using `rThomas`, `rMatClust`, `rCauchy`, `rVarGamma`, and `rLGCP`.

Lower-level fitting functions include:

<code>lgcp.estK</code>	fit a log-Gaussian Cox process model
<code>lgcp.estpcf</code>	fit a log-Gaussian Cox process model
<code>thomas.estK</code>	fit the Thomas process model
<code>thomas.estpcf</code>	fit the Thomas process model
<code>matclust.estK</code>	fit the Matérn Cluster process model
<code>matclust.estpcf</code>	fit the Matérn Cluster process model
<code>cauchy.estK</code>	fit a Neyman-Scott Cauchy cluster process
<code>cauchy.estpcf</code>	fit a Neyman-Scott Cauchy cluster process
<code>vargamma.estK</code>	fit a Neyman-Scott Variance Gamma process
<code>vargamma.estpcf</code>	fit a Neyman-Scott Variance Gamma process
<code>mincontrast</code>	low-level algorithm for fitting models by the method of minimum contrast

IV. MODEL FITTING (POISSON AND GIBBS MODELS)

Types of models

Poisson point processes are the simplest models for point patterns. A Poisson model assumes that the points are stochastically independent. It may allow the points to have a non-uniform spatial density. The special case of a Poisson process with a uniform spatial density is often called Complete Spatial Randomness.

Poisson point processes are included in the more general class of Gibbs point process models. In a Gibbs model, there is *interaction* or dependence between points. Many different types of interaction can be specified.

For a detailed explanation of how to fit Poisson or Gibbs point process models to point pattern data using **spatstat**, see Baddeley and Turner (2005b) or Baddeley (2008).

To fit a Poisson or Gibbs point process model:

Model fitting in **spatstat** is performed mainly by the function `ppm`. Its result is an object of class "ppm".

Here are some examples, where X is a point pattern (class "ppp"):

<i>command</i>	<i>model</i>
<code>ppm(X)</code>	Complete Spatial Randomness
<code>ppm(X ~ 1)</code>	Complete Spatial Randomness
<code>ppm(X ~ x)</code>	Poisson process with intensity loglinear in x coordinate
<code>ppm(X ~ 1, Strauss(0.1))</code>	Stationary Strauss process
<code>ppm(X ~ x, Strauss(0.1))</code>	Strauss process with conditional intensity loglinear in x

It is also possible to fit models that depend on other covariates.

Manipulating the fitted model:

<code>plot.ppm</code>	Plot the fitted model
<code>predict.ppm</code>	Compute the spatial trend and conditional intensity of the fitted point process model
<code>coef.ppm</code>	Extract the fitted model coefficients

<code>parameters</code>	Extract all model parameters
<code>formula.ppm</code>	Extract the trend formula
<code>intensity.ppm</code>	Compute fitted intensity
<code>Kmodel.ppm</code>	K function of fitted model
<code>pcfmodel.ppm</code>	pair correlation of fitted model
<code>fitted.ppm</code>	Compute fitted conditional intensity at quadrature points
<code>residuals.ppm</code>	Compute point process residuals at quadrature points
<code>update.ppm</code>	Update the fit
<code>vcov.ppm</code>	Variance-covariance matrix of estimates
<code>rmh.ppm</code>	Simulate from fitted model
<code>simulate.ppm</code>	Simulate from fitted model
<code>print.ppm</code>	Print basic information about a fitted model
<code>summary.ppm</code>	Summarise a fitted model
<code>effectfun</code>	Compute the fitted effect of one covariate
<code>logLik.ppm</code>	log-likelihood or log-pseudolikelihood
<code>anova.ppm</code>	Analysis of deviance
<code>model.frame.ppm</code>	Extract data frame used to fit model
<code>model.images</code>	Extract spatial data used to fit model
<code>model.depends</code>	Identify variables in the model
<code>as.interact</code>	Interpoint interaction component of model
<code>fitin</code>	Extract fitted interpoint interaction
<code>is.hybrid</code>	Determine whether the model is a hybrid
<code>valid.ppm</code>	Check the model is a valid point process
<code>project.ppm</code>	Ensure the model is a valid point process

For model selection, you can also use the generic functions `step`, `drop1` and `AIC` on fitted point process models. For variable selection, see `sdr`.

See `spatstat.options` to control plotting of fitted model.

To specify a point process model:

The first order “trend” of the model is determined by an R language formula. The formula specifies the form of the *logarithm* of the trend.

<code>X ~ 1</code>	No trend (stationary)
<code>X ~ x</code>	Loglinear trend $\lambda(x, y) = \exp(\alpha + \beta x)$ where x, y are Cartesian coordinates
<code>X ~ polynom(x, y, 3)</code>	Log-cubic polynomial trend
<code>X ~ harmonic(x, y, 2)</code>	Log-harmonic polynomial trend
<code>X ~ Z</code>	Loglinear function of covariate Z $\lambda(x, y) = \exp(\alpha + \beta Z(x, y))$

The higher order (“interaction”) components are described by an object of class “interact”. Such objects are created by:

<code>Poisson()</code>	the Poisson point process
<code>AreaInter()</code>	Area-interaction process
<code>BadGey()</code>	multiscale Geyer process
<code>Concom()</code>	connected component interaction

<code>DiggleGratton()</code>	Diggle-Gratton potential
<code>DiggleGatesStibbard()</code>	Diggle-Gates-Stibbard potential
<code>Fiksel()</code>	Fiksel pairwise interaction process
<code>Geyer()</code>	Geyer's saturation process
<code>Hardcore()</code>	Hard core process
<code>HierHard()</code>	Hierarchical multitype hard core process
<code>HierStrauss()</code>	Hierarchical multitype Strauss process
<code>HierStraussHard()</code>	Hierarchical multitype Strauss-hard core process
<code>Hybrid()</code>	Hybrid of several interactions
<code>LennardJones()</code>	Lennard-Jones potential
<code>MultiHard()</code>	multitype hard core process
<code>MultiStrauss()</code>	multitype Strauss process
<code>MultiStraussHard()</code>	multitype Strauss/hard core process
<code>OrdThresh()</code>	Ord process, threshold potential
<code>Ord()</code>	Ord model, user-supplied potential
<code>PairPiece()</code>	pairwise interaction, piecewise constant
<code>Pairwise()</code>	pairwise interaction, user-supplied potential
<code>Penttinen()</code>	Penttinen pairwise interaction
<code>SatPiece()</code>	Saturated pair model, piecewise constant potential
<code>Saturated()</code>	Saturated pair model, user-supplied potential
<code>Softcore()</code>	pairwise interaction, soft core potential
<code>Strauss()</code>	Strauss process
<code>StraussHard()</code>	Strauss/hard core point process
<code>Triplets()</code>	Geyer triplets process

Note that it is also possible to combine several such interactions using `Hybrid`.

Finer control over model fitting:

A quadrature scheme is represented by an object of class "quad". To create a quadrature scheme, typically use `quadscheme`.

<code>quadscheme</code>	default quadrature scheme using rectangular cells or Dirichlet cells
<code>pixelquad</code>	quadrature scheme based on image pixels
<code>quad</code>	create an object of class "quad"

To inspect a quadrature scheme:

<code>plot(Q)</code>	plot quadrature scheme Q
<code>print(Q)</code>	print basic information about quadrature scheme Q
<code>summary(Q)</code>	summary of quadrature scheme Q

A quadrature scheme consists of data points, dummy points, and weights. To generate dummy points:

<code>default.dummy</code>	default pattern of dummy points
<code>gridcentres</code>	dummy points in a rectangular grid
<code>rstrat</code>	stratified random dummy pattern

<code>spokes</code>	radial pattern of dummy points
<code>corners</code>	dummy points at corners of the window

To compute weights:

<code>gridweights</code>	quadrature weights by the grid-counting rule
<code>dirichletWeights</code>	quadrature weights are Dirichlet tile areas

Simulation and goodness-of-fit for fitted models:

<code>rmh.ppm</code>	simulate realisations of a fitted model
<code>simulate.ppm</code>	simulate realisations of a fitted model
<code>envelope</code>	compute simulation envelopes for a fitted model

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.

<code>lppm</code>	point process model on linear network
<code>anova.lppm</code>	analysis of deviance for point process model on linear network
<code>envelope.lppm</code>	simulation envelopes for point process model on linear network
<code>fitted.lppm</code>	fitted intensity values
<code>predict.lppm</code>	model prediction on linear network
<code>linim</code>	pixel image on linear network
<code>plot.linim</code>	plot a pixel image on linear network
<code>eval.linim</code>	evaluate expression involving images
<code>linfun</code>	function defined on linear network
<code>methods.linfun</code>	conversion facilities

V. MODEL FITTING (DETERMINANTAL POINT PROCESS MODELS)

Code for fitting *determinantal point process models* has recently been added to **spatstat**.

For information, see the help file for `dppm`.

VI. MODEL FITTING (SPATIAL LOGISTIC REGRESSION)

Logistic regression

Pixel-based spatial logistic regression is an alternative technique for analysing spatial point patterns that is widely used in Geographical Information Systems. It is approximately equivalent to fitting a Poisson point process model.

In pixel-based logistic regression, the spatial domain is divided into small pixels, the presence or absence of a data point in each pixel is recorded, and logistic regression is used to model the presence/absence indicators as a function of any covariates.

Facilities for performing spatial logistic regression are provided in **spatstat** for comparison purposes.

Fitting a spatial logistic regression

Spatial logistic regression is performed by the function `slrm`. Its result is an object of class "slrm". There are many methods for this class, including methods for `print`, `fitted`, `predict`, `simulate`, `anova`, `coef`, `logLik`, `terms`, `update`, `formula` and `vcov`.

For example, if X is a point pattern (class "ppp"):

<i>command</i>	<i>model</i>
<code>slrm(X ~ 1)</code>	Complete Spatial Randomness
<code>slrm(X ~ x)</code>	Poisson process with intensity loglinear in x coordinate
<code>slrm(X ~ Z)</code>	Poisson process with intensity loglinear in covariate Z

Manipulating a fitted spatial logistic regression

<code>anova.slrm</code>	Analysis of deviance
<code>coef.slrm</code>	Extract fitted coefficients
<code>vcov.slrm</code>	Variance-covariance matrix of fitted coefficients
<code>fitted.slrm</code>	Compute fitted probabilities or intensity
<code>logLik.slrm</code>	Evaluate loglikelihood of fitted model
<code>plot.slrm</code>	Plot fitted probabilities or intensity
<code>predict.slrm</code>	Compute predicted probabilities or intensity with new data
<code>simulate.slrm</code>	Simulate model

There are many other undocumented methods for this class, including methods for `print`, `update`, `formula` and `terms`. Stepwise model selection is possible using `step` or `stepAIC`. For variable selection, see `sdr`.

VII. SIMULATION

There are many ways to generate a random point pattern, line segment pattern, pixel image or tessellation in `spatstat`.

Random point patterns:

<code>runifpoint</code>	generate n independent uniform random points
<code>rpoint</code>	generate n independent random points
<code>rmpoint</code>	generate n independent multitype random points
<code>rpoispp</code>	simulate the (in)homogeneous Poisson point process
<code>rmpoispp</code>	simulate the (in)homogeneous multitype Poisson point process
<code>runifdisc</code>	generate n independent uniform random points in disc
<code>rstrat</code>	stratified random sample of points
<code>rsyst</code>	systematic random sample (grid) of points
<code>rMaternI</code>	simulate the Matérn Model I inhibition process
<code>rMaternII</code>	simulate the Matérn Model II inhibition process
<code>rSSI</code>	simulate Simple Sequential Inhibition process
<code>rHardcore</code>	simulate hard core process (perfect simulation)
<code>rStrauss</code>	simulate Strauss process (perfect simulation)

<code>rStraussHard</code>	simulate Strauss-hard core process (perfect simulation)
<code>rDiggleGratton</code>	simulate Diggle-Gratton process (perfect simulation)
<code>rDGS</code>	simulate Diggle-Gates-Stibbard process (perfect simulation)
<code>rPenttinen</code>	simulate Penttinen process (perfect simulation)
<code>rNeymanScott</code>	simulate a general Neyman-Scott process
<code>rMatClust</code>	simulate the Matérn Cluster process
<code>rThomas</code>	simulate the Thomas process
<code>rLGCP</code>	simulate the log-Gaussian Cox process
<code>rGaussPoisson</code>	simulate the Gauss-Poisson cluster process
<code>rCauchy</code>	simulate Neyman-Scott process with Cauchy clusters
<code>rVarGamma</code>	simulate Neyman-Scott process with Variance Gamma clusters
<code>rcell</code>	simulate the Baddeley-Silverman cell process
<code>runifpointOnLines</code>	generate n random points along specified line segments
<code>rpoisppOnLines</code>	generate Poisson random points along specified line segments

Resampling a point pattern:

<code>quadratresample</code>	block resampling
<code>rjitter</code>	apply random displacements to points in a pattern
<code>rshift</code>	random shifting of (subsets of) points
<code>rthin</code>	random thinning

See also `varblock` for estimating the variance of a summary statistic by block resampling, and `lohboot` for another bootstrap technique.

Fitted point process models:

If you have fitted a point process model to a point pattern dataset, the fitted model can be simulated.

Cluster process models are fitted by the function `kppm` yielding an object of class "kppm". To generate one or more simulated realisations of this fitted model, use `simulate.kppm`.

Gibbs point process models are fitted by the function `ppm` yielding an object of class "ppm". To generate a simulated realisation of this fitted model, use `rmh`. To generate one or more simulated realisations of the fitted model, use `simulate.ppm`.

Other random patterns:

<code>rlinegrid</code>	generate a random array of parallel lines through a window
<code>rpoisline</code>	simulate the Poisson line process within a window
<code>rpoislinetess</code>	generate random tessellation using Poisson line process
<code>rMosaicSet</code>	generate random set by selecting some tiles of a tessellation
<code>rMosaicField</code>	generate random pixel image by assigning random values in each tile of a tessellation

Simulation-based inference

<code>envelope</code>	critical envelope for Monte Carlo test of goodness-of-fit
<code>bits.envelope</code>	critical envelope for balanced two-stage Monte Carlo test
<code>qqplot.ppm</code>	diagnostic plot for interpoint interaction
<code>scan.test</code>	spatial scan statistic/test
<code>studpermu.test</code>	studentised permutation test

`segregation.test` test of segregation of types

VIII. TESTS AND DIAGNOSTICS

Hypothesis tests:

<code>quadrat.test</code>	χ^2 goodness-of-fit test on quadrat counts
<code>clarkevans.test</code>	Clark and Evans test
<code>cdf.test</code>	Spatial distribution goodness-of-fit test
<code>berman.test</code>	Berman's goodness-of-fit tests
<code>envelope</code>	critical envelope for Monte Carlo test of goodness-of-fit
<code>scan.test</code>	spatial scan statistic/test
<code>dclf.test</code>	Diggle-Cressie-Loosmore-Ford test
<code>mad.test</code>	Mean Absolute Deviation test
<code>anova.ppm</code>	Analysis of Deviance for point process models

More recently-developed tests:

<code>dg.test</code>	Dao-Genton test
<code>bits.test</code>	Balanced independent two-stage test
<code>dclf.progress</code>	Progress plot for DCLF test
<code>mad.progress</code>	Progress plot for MAD test

Sensitivity diagnostics:

Classical measures of model sensitivity such as leverage and influence have been adapted to point process models.

<code>leverage.ppm</code>	Leverage for point process model
<code>influence.ppm</code>	Influence for point process model
<code>dfbetas.ppm</code>	Parameter influence
<code>dffit.ppm</code>	Effect change diagnostic

Diagnostics for covariate effect:

Classical diagnostics for covariate effects have been adapted to point process models.

<code>parres</code>	Partial residual plot
<code>addvar</code>	Added variable plot
<code>rhohat</code>	Kernel estimate of covariate effect
<code>rho2hat</code>	Kernel estimate of covariate effect (bivariate)

Residual diagnostics:

Residuals for a fitted point process model, and diagnostic plots based on the residuals, were introduced in Baddeley et al (2005) and Baddeley, Rubak and Møller (2011).

Type `demo(diagnose)` for a demonstration of the diagnostics features.

<code>diagnose.ppm</code>	diagnostic plots for spatial trend
<code>qqplot.ppm</code>	diagnostic Q-Q plot for interpoint interaction
<code>residualspaper</code>	examples from Baddeley et al (2005)
<code>Kcom</code>	model compensator of K function
<code>Gcom</code>	model compensator of G function
<code>Kres</code>	score residual of K function
<code>Gres</code>	score residual of G function
<code>psst</code>	pseudoscore residual of summary function
<code>psstA</code>	pseudoscore residual of empty space function
<code>psstG</code>	pseudoscore residual of G function
<code>compareFit</code>	compare compensators of several fitted models

Resampling and randomisation procedures

You can build your own tests based on randomisation and resampling using the following capabilities:

<code>quadratresample</code>	block resampling
<code>rjitter</code>	apply random displacements to points in a pattern
<code>rshift</code>	random shifting of (subsets of) points
<code>rthin</code>	random thinning

IX. DOCUMENTATION

The online manual entries are quite detailed and should be consulted first for information about a particular function.

The book Baddeley, Rubak and Turner (2015) is a complete course on analysing spatial point patterns, with full details about **spatstat**.

Older material (which is now out-of-date but is freely available) includes Baddeley and Turner (2005a), a brief overview of the package in its early development; Baddeley and Turner (2005b), a more detailed explanation of how to fit point process models to data; and Baddeley (2010), a complete set of notes from a 2-day workshop on the use of **spatstat**.

Type `citation("spatstat")` to get a list of these references.

Licence

This library and its documentation are usable under the terms of the "GNU General Public License", a copy of which is distributed with the package.

Acknowledgements

Kasper Klitgaard Berthelsen, Ottmar Cronie, Tilman Davies, Yongtao Guan, Ute Hahn, Abdollah Jalilian, Marie-Colette van Lieshout, Greg McSwiggan, Tuomas Rajala, Suman Rakshit, Dominic Schuhmacher, Rasmus Waagepetersen and Hangsheng Wang made substantial contributions of code.

Additional contributions and suggestions from Monsuru Adepeju, Corey Anderson, Ang Qi Wei, Ryan Arellano, Jens Åström, Robert Aue, Marcel Austenfeld, Sandro Azaele, Malissa Baddeley,

Guy Bayegnak, Colin Beale, Melanie Bell, Thomas Bendtsen, Ricardo Bernhardt, Andrew Bevan, Brad Biggerstaff, Anders Bilgrau, Leanne Bischof, Christophe Biscio, Roger Bivand, Jose M. Blanco Moreno, Florent Bonneau, Jordan Brown, Ian Buller, Julian Burgos, Simon Byers, Ya-Mei Chang, Jianbao Chen, Igor Chernayavsky, Y.C. Chin, Bjarke Christensen, Lucía Cobo Sanchez, Jean-François Coeurjolly, Kim Colyvas, Hadrien Commenges, Rochelle Constantine, Robin Corria Ainslie, Richard Cotton, Marcelino de la Cruz, Peter Dalgaard, Mario D'Antuono, Sourav Das, Peter Diggle, Patrick Donnelly, Ian Dryden, Stephen Eglen, Ahmed El-Gabbas, Belarmain Fandohan, Olivier Flores, David Ford, Peter Forbes, Shane Frank, Janet Franklin, Funwi-Gabga Neba, Oscar Garcia, Agnes Gault, Jonas Geldmann, Marc Genton, Shaaban Ghalandarayeshi, Julian Gilbey, Jason Goldstick, Pavel Grabarnik, C. Graf, Ute Hahn, Andrew Hardegen, Martin Bøgsted Hansen, Martin Hazelton, Juha Heikkinen, Mandy Hering, Markus Herrmann, Maximilian Hesselbarth, Paul Hewson, Hamidreza Heydarian, Kassel Hingee, Kurt Hornik, Philipp Hunziker, Jack Hywood, Ross Ihaka, Ācenk İçös, Aruna Jammalamadaka, Robert John-Chandran, Devin Johnson, Mahdiah Khanmohammadi, Bob Klaver, Lily Kozmian-Ledward, Peter Kovesi, Mike Kuhn, Jeff Laake, Robert Lamb, Frédéric Lavancier, Tom Lawrence, Tomas Lazauskas, Jonathan Lee, George Leser, Angela Li, Li Haitao, George Limitsios, Andrew Lister, Nestor Luambua, Bethany Macdonald, Ben Madin, Martin Maechler, Daniel Manrique-Castaño, Kiran Marchikanti, Jeff Marcus, Robert Mark, Peter McCullagh, Monia Mahling, Jorge Mateu Mahiques, Ulf Mehlhlig, Frederico Mestre, Sebastian Wastl Meyer, Mi Xiangcheng, Lore De Middeleer, Robin Milne, Enrique Miranda, Jesper Møller, Annie Mollié, Ines Moncada, Mehdi Moradi, Virginia Morera Pujol, Erika Mudrak, Gopalan Nair, Nader Najari, Nicoletta Nava, Linda Stougaard Nielsen, Felipe Nunes, Jens Randel Nyengaard, Jens Oehlschlägel, Thierry Onkelinx, Sean O'Riordan, Evgeni Parilov, Jeff Picka, Nicolas Picard, Tim Pollington, Mike Porter, Sergiy Protsiv, Adrian Raftery, Suman Rakshit, Ben Ramage, Pablo Ramon, Xavier Raynaud, Nicholas Read, Matt Reiter, Ian Renner, Tom Richardson, Brian Ripley, Yonatan Rosen, Ted Rosenbaum, Barry Rowlingson, Jason Rudokas, Tyler Rudolph, John Rudge, Christopher Ryan, Farzaneh Safavimanesh, Aila Särkkä, Cody Schank, Katja Schladitz, Sebastian Schutte, Bryan Scott, Olivia Semboli, François Sémécurbe, Vadim Shcherbakov, Shen Guochun, Shi Peijian, Harold-Jeffrey Ship, Tammy L Silva, Ida-Maria Sintorn, Yong Song, Malte Spiess, Mark Stevenson, Kaspar Stucki, Jan Sulavik, Michael Sumner, P. Surovy, Ben Taylor, Thordis Linda Thorarinsdottir, Leigh Torres, Berwin Turlach, Torben Tvedebrink, Kevin Ummer, Medha Uppala, Andrew van Burgel, Tobias Verbeke, Mikko Vihtakari, Alexandre Villers, Fabrice Vinatier, Maximilian Vogtland, Sasha Voss, Sven Wagner, Hao Wang, H. Wendrock, Jan Wild, Carl G. Witthoft, Selene Wong, Maxime Woringe, Luke Yates, Mike Zamboni and Achim Zeileis.

Author(s)

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

References

- Baddeley, A. (2010) *Analysing spatial point patterns in R*. Workshop notes, Version 4.1. Online technical publication, CSIRO. https://research.csiro.au/software/wp-content/uploads/sites/6/2015/02/Rspatialcourse_CMIS_PDF-Standard.pdf
- Baddeley, A., Rubak, E. and Turner, R. (2015) *Spatial Point Patterns: Methodology and Applications with R*. Chapman and Hall/CRC Press.
- Baddeley, A. and Turner, R. (2005a) Spatstat: an R package for analyzing spatial point patterns. *Journal of Statistical Software* **12**:6, 1–42. DOI: 10.18637/jss.v012.i06.

- Baddeley, A. and Turner, R. (2005b) 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., 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.
- Baddeley, A., Rubak, E. and Møller, J. (2011) Score, pseudo-score and residual diagnostics for spatial point process models. *Statistical Science* **26**, 613–646.
- Baddeley, A., Turner, R., Mateu, J. and Bevan, A. (2013) Hybrids of Gibbs point process models and their implementation. *Journal of Statistical Software* **55**:11, 1–43. <https://www.jstatsoft.org/v55/i11/>
- Diggle, P.J. (2003) *Statistical analysis of spatial point patterns*, Second edition. Arnold.
- Diggle, P.J. (2014) *Statistical Analysis of Spatial and Spatio-Temporal Point Patterns*, Third edition. Chapman and Hall/CRC.
- Gelfand, A.E., Diggle, P.J., Fuentes, M. and Guttorp, P., editors (2010) *Handbook of Spatial Statistics*. CRC Press.
- Huang, F. and Ogata, Y. (1999) Improvements of the maximum pseudo-likelihood estimators in various spatial statistical models. *Journal of Computational and Graphical Statistics* **8**, 510–530.
- Illian, J., Penttinen, A., Stoyan, H. and Stoyan, D. (2008) *Statistical Analysis and Modelling of Spatial Point Patterns*. Wiley.
- Waagepetersen, R. An estimating function approach to inference for inhomogeneous Neyman-Scott processes. *Biometrics* **63** (2007) 252–258.

 beginner

Print Introduction For Beginners

Description

Prints an introduction for beginners to the spatstat package, or another specified package.

Usage

```
beginner(package = "spatstat")
```

Arguments

package Name of package.

Details

This function prints an introduction for beginners to the **spatstat** package.

The function can be executed simply by typing beginner without parentheses.

If the argument package is given, then the function prints the beginner's help file BEGINNER.txt from the specified package (if it has one).

Value

Null.

Author(s)

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

See Also

[latest.news](#)

Examples

```
beginner
```

bugfixes

List Recent Bug Fixes

Description

List all bug fixes in a package, starting from a certain date or version of the package. Fixes are sorted alphabetically by the name of the affected function. The default is to list bug fixes in the latest version of the **spatstat** family of packages.

Usage

```
bugfixes(sinceversion = NULL, sinedate = NULL,  
         package = spatstat.family(), show = TRUE)
```

Arguments

sinceversion	Earliest version of package for which bugs should be listed. A character string. The default is the current installed version.
sinedate	Earliest release date of package for which bugs should be listed. A character string or a date-time object.
package	The name of the package (or packages) for which bugs are to be listed. A character string or character vector.
show	Logical value indicating whether to display the bug table on the terminal.

Details

Bug reports are extracted from the NEWS file of the specified package. Only those after a specified date, or after a specified version of the package, are retained. The bug reports are then sorted alphabetically, so that all bugs affecting a particular function are listed consecutively. Finally the table of bug reports is displayed (if `show=TRUE`) and returned invisibly.

The argument `sinceversion` should be a character string like "1.2-3". The default is the current installed version of the package.

The argument `sincedeate` should be a character string like "2015-05-27", or a date-time object.

If `sinceversion="all"` or `sincedeate="all"` then all recorded bugs will be listed.

The special options `sinceversion="book"` and `sincedeate="book"` are interpreted to mean `sincedeate="2015-06-05"`, which gives all bugs reported after publication of the book by Baddeley, Rubak and Turner (2015).

Typing `bugfixes` without parentheses will display a table of all bugs that were fixed in the current installed version of **spatstat** and its sub-packages.

By default, bugs in the *extension* packages **spatstat.local**, **spatstat.Knet**, **spatstat.gui** are *not* reported. To include these bugs as well, set `package=spatstat.family(TRUE, TRUE)`.

Value

(Invisibly) a data frame, belonging to the class "bugtable", which has a `print` method.

Author(s)

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

References

Baddeley, A., Rubak, E. and Turner, R. (2015) *Spatial Point Patterns: Methodology and Applications with R*. Chapman and Hall/CRC Press.

See Also

[latest.changes](#), [latest.news](#), [news](#).

Examples

```
bugfixes
## show all bugs reported after publication of the spatstat book
if(interactive()) bugfixes(sinceversion="book")
```

`foo`*Foo is Not a Real Name*

Description

The name `foo` is not a real name: it is a place holder, used to represent the name of any desired thing.

The functions defined here simply print an explanation of the placeholder name `foo`.

Usage

```
foo()
```

```
## S3 method for class 'foo'  
plot(x, ...)
```

Arguments

<code>x</code>	Ignored.
<code>...</code>	Ignored.

Details

The name `foo` is used by computer scientists as a *place holder*, to represent the name of any desired object or function. It is not the name of an actual object or function; it serves only as an example, to explain a concept.

However, many users misinterpret this convention, and actually type the command `foo` or `foo()`. Then they email the package author to inform them that `foo` is not defined.

To avoid this correspondence, we have now defined an object called `foo`.

The function `foo()` prints a message explaining that `foo` is not really the name of a variable.

The function can be executed simply by typing `foo` without parentheses.

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

[beginner](#)

Examples

```
foo
```

```
latest.changes
```

```
List Recent Significant Changes to a Function
```

Description

List all the changes to a particular function in a package, starting from a certain date or version of the package. The default is to list changes in the latest version of the **spatstat** family of packages.

Usage

```
latest.changes(x,
               sinceversion = NULL, sinedate = NULL,
               package = spatstat.family(), show = TRUE)
```

Arguments

x	Character string giving the name of the function of interest, or a search pattern to be matched. A character vector is permitted.
sinceversion	Earliest version of package for which changes should be listed. A character string. Default is the most recent version.
sinedate	Earliest release date of package for which changes should be listed. A character string or a date-time object. Default is the date of the most recent version.
package	The name of the package (or packages) for which changes are to be listed. A character string or character vector.
show	Logical value indicating whether to display the table of changes on the terminal.

Details

Details of changes are extracted from the NEWS file of the specified package under the heading ‘Significant User-Visible Changes’. All entries for which the first line matches x are selected. The table of changes is displayed (if show=TRUE) and returned invisibly.

The argument sinceversion should be a character string like "1.2-3". The default is the version string of the most recent version.

The argument sinedate should be a character string like "2015-05-27", or a date-time object. The default is the date of the most recent version.

If sinceversion="all" or sinedate="all" then all recorded changes will be listed.

The special options sinceversion="book" and sinedate="book" are interpreted to mean sinedate="2015-06-05", which gives all changes reported after publication of the book by Baddeley, Rubak and Turner (2015).

By default, changes in the *extension* packages **spatstat.local**, **spatstat.Knet**, **spatstat.gui** are *not* reported. To include these changes as well, set package=spatstat.family(TRUE, TRUE).

Value

(Invisibly) a data frame, belonging to the class "changetable", which has a print method.

Author(s)

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

References

Baddeley, A., Rubak, E. and Turner, R. (2015) *Spatial Point Patterns: Methodology and Applications with R*. Chapman and Hall/CRC Press.

See Also

[bugfixes](#), [latest.news](#), [news](#).

Examples

```
latest.changes("plot.symbolmap")
```

```
latest.news
```

Print News About Latest Version of Package

Description

Prints the news documentation for the current version of spatstat or another specified package.

Usage

```
latest.news(package = spatstat.family(), doBrowse=FALSE, major=TRUE)
```

Arguments

package	Name of package for which the latest news should be printed. A character string, or vector of character strings.
doBrowse	Logical value indicating whether to display the results in a browser window instead of printing them.
major	Logical value. If TRUE (the default), print all information for the current major version "x.y". If FALSE, print only the information for the current minor version "x.y-z".

Details

This function prints the news documentation about changes in the current installed version of a package.

By default, it prints the latest news about all the sub-packages in the **spatstat** family.

The function can be called simply by typing its name without parentheses (see the Examples).

If `major=FALSE`, only information for the current minor version "`x.y-z`" will be printed. If `major=TRUE` (the default), all information for the current major version "`x.y`" will be printed, encompassing versions "`x.y-0`", "`x.y-1`", up to "`x.y-z`".

If `package` is given, then the function reads the news for the specified package from its NEWS file (if it has one) and prints only the entries that refer to the current version of the package.

To see the news for all previous versions as well as the current version, use the R utility [news](#). See the Examples.

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

[spatstat.family](#) lists the packages in the **spatstat** family.

[bugfixes](#) lists bug fixes.

[news](#)

Examples

```
if(interactive()) {  
  
  # current news  
  latest.news  
  
  # all news  
  # news(package="spatstat")  
  
}
```

spatstat.family	<i>Names of All Packages in the Spatstat Family</i>
-----------------	---

Description

Provides the names of all the packages belonging to the **spatstat** family of packages.

Usage

```
spatstat.family(subpackages=TRUE, extensions=FALSE)
```

Arguments

subpackages	Logical value specifying whether to include sub-packages.
extensions	Logical value specifying whether to include extension packages.

Details

This function returns a character vector containing the names of the packages that belong to the **spatstat** family.

By default, only the sub-packages are listed, and not the extension packages.

A “sub-package” is a package which is implicitly loaded or imported when the command `library(spatstat)` is issued. Currently the sub-packages are:

- spatstat.utils
- spatstat.data
- spatstat.univar
- spatstat.sparse
- spatstat.geom
- spatstat.random
- spatstat.explore
- spatstat.model
- spatstat.linnet
- spatstat

An “extension package” is a package which must be loaded explicitly. The extension packages are:

- spatstat.gui
- spatstat.local
- spatstat.Knet

Value

Character vector of package names.

Author(s)

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

See Also

[latest.news](#)

Index

* **documentation**

- beginner, [30](#)
- bugfixes, [31](#)
- foo, [33](#)
- latest.changes, [34](#)
- latest.news, [35](#)

* **package**

- spatstat-package, [2](#)

* **spatial**

- spatstat-package, [2](#)
- spatstat.family, [37](#)

[.im, [11](#)

[.layered, [15](#)

[.ppp, [8](#)

[.psp, [12](#)

[.tess, [13](#)

[<-.im, [11](#)

[<-.tess, [13](#)

addvar, [27](#)

addVertices, [14](#)

affine, [8](#), [9](#)

affine.im, [11](#)

affine.psp, [12](#)

affine.tess, [13](#)

AIC, [20](#), [22](#)

allstats, [16](#)

alltypes, [18](#)

amacrine, [7](#)

anemones, [7](#)

angles.psp, [12](#)

anova.lppm, [24](#)

anova.ppm, [22](#), [27](#)

anova.slrn, [25](#)

ants, [7](#)

applynbd, [18](#)

area.owin, [10](#)

AreaInter, [22](#)

as.box3, [13](#)

as.data.frame.hyperframe, [15](#)

as.data.frame.im, [11](#)

as.data.frame.owin, [9](#)

as.data.frame.psp, [12](#)

as.function.im, [11](#)

as.hyperframe, [13–15](#)

as.im, [11](#)

as.im.owin, [10](#)

as.im.ppp, [9](#)

as.interact, [22](#)

as.mask, [10](#)

as.matrix.im, [11](#)

as.owin, [9](#)

as.polygonal, [10](#)

as.ppp, [6](#)

as.psp, [12](#)

as.tess, [13](#)

BadGey, [22](#)

bdist.pixels, [10](#)

bdist.points, [10](#)

bdist.tiles, [10](#), [13](#)

bdspts, [7](#)

beachcolourmap, [15](#)

beginner, [30](#), [33](#)

bei, [7](#)

berman.test, [27](#)

betacells, [7](#)

bits.envelope, [26](#)

bits.test, [27](#)

blur, [11](#)

border, [9](#)

boundingbox, [9](#)

box3, [13](#)

boxx, [14](#)

bramblecanes, [7](#)

bronzefilter, [7](#)

bugfixes, [31](#), [35](#), [36](#)

bw.abram, [16](#)

bw.CvL, [16](#)

bw.diggle, [16](#)

bw.frac, [16](#)
bw.ppl, [16](#)
bw.relrisk, [16](#)
bw.scott, [16](#)
bw.smoothppp, [16](#)
bw.stoyan, [16](#)
by.ppp, [8](#)

cauchy.estK, [21](#)
cauchy.estpcf, [21](#)
cbind.hyperframe, [15](#)
cdf.test, [27](#)
cells, [7](#)
centroid.owin, [10](#)
chicago, [7](#), [14](#)
chop.tess, [13](#)
chorley, [7](#)
clarkevans, [15](#)
clarkevans.test, [27](#)
checkbox, [9](#)
clickdist, [10](#)
clickjoin, [14](#)
clickpoly, [9](#)
clickppp, [6](#)
clmfires, [7](#)
closing, [9](#)
clusterfield.kppm, [20](#)
clusterradius.kppm, [20](#)
clusterset, [16](#)
coef.kppm, [20](#)
coef.ppm, [21](#)
coef.slrn, [25](#)
colourmap, [15](#)
commonGrid, [10](#), [11](#)
compareFit, [28](#)
compatible.im, [11](#)
complement.owin, [9](#)
Concom, [22](#)
connected.im, [11](#)
connected.owin, [10](#)
connected.ppp, [8](#)
connected.tess, [13](#)
contour.im, [11](#)
convexhull, [9](#), [10](#)
convolve.im, [11](#)
coords, [8](#), [13](#)
copper, [7](#)
corners, [24](#)
crossdist, [17](#)
crossdist.lpp, [19](#)
crossdist.pp3, [19](#)
crossdist.ppx, [20](#)
crossing.psp, [12](#)
cut.im, [11](#)
cut.ppp, [8](#), [18](#)

data, [7](#)
dclf.progress, [27](#)
dclf.test, [27](#)
default.dummy, [23](#)
delaunay, [9](#), [13](#)
delaunayDistance, [9](#)
delaunayNetwork, [14](#)
demohyper, [7](#)
demopat, [7](#)
dendrite, [7](#), [14](#)
density.lpp, [19](#)
density.ppp, [8](#), [10](#), [16](#), [17](#)
density.psp, [12](#)
densityHeat.lpp, [19](#)
densityHeat.ppp, [8](#), [17](#)
deriv.fv, [17](#)
dfbetas.ppm, [27](#)
dffit.ppm, [27](#)
dg.test, [27](#)
diagnose.ppm, [28](#)
diameter.box3, [13](#)
diameter.boxx, [14](#)
diameter.owin, [10](#)
DiggleGatesStibbard, [23](#)
DiggleGratton, [23](#)
dilated.areas, [10](#)
dilation, [9](#)
dirichlet, [9](#), [13](#)
dirichletNetwork, [14](#)
dirichletWeights, [24](#)
disc, [9](#)
discretise, [9](#)
distfun, [17](#)
distfun.lpp, [19](#)
distfun.owin, [10](#)
distfun.psp, [12](#)
distmap, [17](#)
distmap.owin, [10](#)
distmap.psp, [12](#)
dppm, [24](#)
drop1, [20](#), [22](#)
duplicated.ppp, [8](#)

edges, [10](#), [12](#)
edit.ppp, [8](#)
effectfun, [22](#)
ellipse, [9](#)
Emark, [18](#)
endpoints.psp, [12](#)
envelope, [17](#), [24](#), [26](#), [27](#)
envelope.lpp, [19](#)
envelope.lppm, [24](#)
envelope.pp3, [13](#), [19](#)
eroded.areas, [10](#)
eroded.volumes, [13](#)
eroded.volumes.boxx, [14](#)
erosion, [9](#)
eval.fasp, [17](#)
eval.fv, [17](#)
eval.im, [11](#)
eval.linim, [24](#)
exactdt, [17](#)
extrapolate.psp, [12](#)

F3est, [19](#)
Fest, [16](#)
Fiksel, [23](#)
Finhom, [16](#)
finpines, [7](#)
fitin, [22](#)
fitted.kppm, [20](#)
fitted.lppm, [24](#)
fitted.ppm, [22](#)
fitted.slrn, [25](#)
flipxy, [8](#), [9](#), [12](#)
flipxy.tess, [13](#)
flu, [7](#)
foo, [33](#)
formula.kppm, [20](#)
formula.ppm, [22](#)
Frame, [9](#)
fryplot, [15](#)

G3est, [19](#)
Gcom, [28](#)
Gcross, [17](#)
Gdot, [17](#)
Gest, [16](#)
Geyer, [23](#)
Gfox, [20](#)
Ginhom, [16](#)
glm, [3](#)

Gmulti, [17](#), [18](#)
gordon, [7](#)
gorillas, [7](#)
Gres, [28](#)
gridcentres, [23](#)
gridweights, [24](#)

hamster, [7](#)
Hardcore, [23](#)
harmonise.fv, [17](#)
harmonise.im, [11](#)
head.hyperframe, [15](#)
Hest, [20](#)
hextess, [12](#)
HierHard, [23](#)
HierStrauss, [23](#)
HierStraussHard, [23](#)
hist.im, [11](#)
hsvim, [11](#)
humberside, [7](#)
Hybrid, [23](#)
hyperframe, [15](#)
hyytiala, [7](#)

identify.ppp, [8](#)
Iest, [18](#)
im, [5](#), [11](#)
im.apply, [11](#)
imcov, [11](#)
improve.kppm, [20](#)
incircle, [10](#)
influence.ppm, [27](#)
inradius, [10](#)
insertVertices, [14](#)
inside.owin, [10](#)
integral.im, [11](#)
intensity, [16](#)
intensity.ppm, [22](#)
intensity.quadratcount, [16](#)
interp.colourmap, [15](#)
interp.im, [11](#)
intersect.owin, [10](#)
intersect.tess, [13](#)
is.convex, [10](#)
is.hybrid, [22](#)
is.im, [11](#)
is.mask, [10](#)
is.polygonal, [10](#)
is.psp, [12](#)

is.rectangle, [10](#)
 is.subset.owin, [10](#)

 japanesepines, [7](#)
 Jcross, [17](#)
 Jdot, [17](#)
 Jest, [16](#)
 Jfox, [20](#)
 Jinhom, [16](#)
 Jmulti, [17](#), [18](#)
 joinVertices, [14](#)

 K3est, [19](#)
 Kcom, [28](#)
 Kcross, [17](#)
 Kcross.inhom, [18](#)
 Kdot, [17](#)
 Kdot.inhom, [18](#)
 Kest, [16](#)
 Kest.fft, [17](#)
 Kinhom, [16](#)
 Kmark, [18](#)
 Kmeasure, [10](#), [17](#)
 Kmodel.kppm, [20](#)
 Kmodel.ppm, [22](#)
 Kmulti, [17](#), [18](#)
 kppm, [20](#), [26](#)
 Kres, [28](#)
 Kscaled, [17](#)
 Ksector, [17](#)

 lansing, [7](#)
 latest.changes, [32](#), [34](#)
 latest.news, [31](#), [32](#), [35](#), [35](#), [38](#)
 layered, [15](#)
 Lcross, [17](#)
 Lcross.inhom, [18](#)
 Ldot, [17](#)
 Ldot.inhom, [18](#)
 lengths_psp, [12](#)
 LennardJones, [23](#)
 Lest, [16](#)
 letterR, [9](#)
 levelset, [11](#)
 leverage.ppm, [27](#)
 lgcp.estK, [21](#)
 lgcp.estpcf, [21](#)
 lineardisc, [14](#)
 linearK, [18](#)
 linearKcross, [19](#)
 linearKcross.inhom, [19](#)
 linearKdot, [19](#)
 linearKdot.inhom, [19](#)
 linearKinhom, [19](#)
 linearmarkconnect, [19](#)
 linearmarkequal, [19](#)
 linearpcf, [19](#)
 linearpcfcross, [19](#)
 linearpcfcross.inhom, [19](#)
 linearpcfdot, [19](#)
 linearpcfdot.inhom, [19](#)
 linearpcfinhom, [19](#)
 linfun, [24](#)
 Linhom, [16](#)
 linim, [24](#)
 linnet, [14](#)
 lm, [3](#)
 localK, [16](#)
 localKcross, [18](#)
 localKcross.inhom, [18](#)
 localKdot, [18](#)
 localKinhom, [16](#)
 localL, [16](#)
 localLcross, [18](#)
 localLcross.inhom, [18](#)
 localLdot, [18](#)
 localLinhom, [16](#)
 localpcf, [16](#)
 localpcfinhom, [17](#)
 logLik.ppm, [22](#)
 logLik.slrn, [25](#)
 lohboot, [17](#), [26](#)
 longleaf, [7](#)
 lpp, [5](#), [14](#)
 lppm, [24](#)

 mad.progress, [27](#)
 mad.test, [27](#)
 markconnect, [18](#)
 markcorr, [18](#)
 markcrosscorr, [18](#)
 markmarkscatter, [18](#)
 markmean, [18](#)
 marks, [8](#)
 marks.psp, [12](#)
 marks<-, [6](#)
 marks<- .psp, [12](#)
 markstat, [18](#)

marktable, 18
markvar, 18
markvario, 18
matclust.estK, 21
matclust.estpcf, 21
mean.im, 11
methods.linfun, 24
methods.linnet, 14
methods.lpp, 14
midpoints.psp, 12
mincontrast, 21
miplot, 15
model.depends, 22
model.frame.ppm, 22
model.images, 22
mucosa, 7
MultiHard, 23
MultiStrauss, 23
MultiStraussHard, 23
murchison, 7

nbfires, 7
nearest.raster.point, 10
nearestsegment, 12
news, 32, 35, 36
nnclean, 16
nncross, 12, 17
nncross.lpp, 19
nncross.pp3, 20
nndist, 17
nndist.lpp, 19
nndist.pp3, 19
nndist.ppx, 20
nnfun, 17
nnfun.lpp, 19
nnmap, 17
nnmark, 8
nnmean, 18
nnvario, 18
nnwhich, 17
nnwhich.lpp, 19
nnwhich.pp3, 20
nnwhich.ppx, 20
npoints, 8, 13, 14
nztrees, 7

opening, 9
Ord, 23
OrdThresh, 23

osteo, 7
owin, 5, 9

pairdist, 17
pairdist.lpp, 19
pairdist.pp3, 19
pairdist.ppx, 20
PairPiece, 23
Pairwise, 23
paracou, 7
parameters, 20, 22
parres, 27
pcf, 16
pcf3est, 19
pcfcross, 17
pcfcross.inhom, 18
pcfdot, 17
pcfdot.inhom, 18
pcfinhom, 16
pcfmodel.kppm, 20
pcfmodel.ppm, 22
pcfmulti, 18
Penttinen, 23
perimeter, 10
periodify, 8, 9, 12
persp.im, 11
persp.ppp, 8
pixelcentres, 10, 11
pixellate, 11
pixellate.linnet, 14
pixellate.owin, 10
pixellate.ppp, 9
pixellate.psp, 12
pixelquad, 23
plot.colourmap, 15
plot.foo (foo), 33
plot.fv, 17
plot.hyperframe, 15
plot.im, 11
plot.kppm, 20
plot.layered, 15
plot.linim, 24
plot.owin, 9
plot.pp3, 13
plot.ppm, 21
plot.ppp, 8
plot.psp, 12
plot.slrn, 25
plot.tess, 13

pointsOnLines, [12](#)
 Poisson, [22](#)
 polartess, [12](#)
 ponderosa, [7](#)
 pool.fv, [17](#)
 pp3, [5](#), [13](#)
 ppm, [21](#), [26](#)
 ppp, [5](#), [6](#)
 pppdist, [18](#)
 ppx, [5](#), [13](#)
 predict.kppm, [20](#)
 predict.lppm, [24](#)
 predict.ppm, [21](#)
 predict.slrn, [25](#)
 print.ppm, [22](#)
 print.psp, [12](#)
 project.ppm, [22](#)
 project2segment, [12](#)
 psp, [5](#), [12](#)
 psp2mask, [12](#)
 psst, [28](#)
 psstA, [28](#)
 psstG, [28](#)
 pyramidal, [7](#)

 qqplot.ppm, [26](#), [28](#)
 quad, [23](#)
 quadrat.test, [27](#)
 quadratcount, [16](#)
 quadratresample, [7](#), [26](#), [28](#)
 quadrats, [12](#)
 quadscheme, [23](#)
 quantess, [12](#)
 quantile.im, [11](#)

 raster.x, [10](#)
 raster.xy, [10](#)
 raster.y, [10](#)
 rbind.hyperframe, [15](#)
 rCauchy, [6](#), [20](#), [26](#)
 rcell, [6](#), [26](#)
 rDGS, [6](#), [26](#)
 rDiggleGratton, [6](#), [26](#)
 redwood, [7](#)
 redwoodfull, [7](#)
 reflect, [8](#)
 reflect.tess, [13](#)
 relrisk, [16](#), [17](#)
 repairNetwork, [14](#)

 residuals.ppm, [22](#)
 residualspaper, [8](#), [28](#)
 rGaussPoisson, [6](#), [26](#)
 rgbim, [11](#)
 rHardcore, [6](#), [25](#)
 rho2hat, [16](#), [27](#)
 rhohat, [16](#), [27](#)
 ripras, [9](#)
 rjitter, [6](#), [26](#), [28](#)
 rknn, [17](#)
 rlabel, [7](#)
 rLGCP, [20](#), [26](#)
 rlinegrid, [12](#), [26](#)
 rMatClust, [6](#), [20](#), [26](#)
 rMaternI, [6](#), [25](#)
 rMaternII, [6](#), [25](#)
 rmh, [6](#), [26](#)
 rmh.ppm, [22](#), [24](#)
 rMosaicField, [26](#)
 rMosaicSet, [26](#)
 rmpoint, [6](#), [25](#)
 rmpoispp, [6](#), [25](#)
 rNeymanScott, [6](#), [26](#)
 rnoise, [11](#)
 roc, [16](#)
 rotate, [8](#), [9](#)
 rotate.im, [11](#)
 rotate.psp, [12](#)
 rotate.tess, [13](#)
 rPenttinen, [6](#), [26](#)
 rpoint, [6](#), [25](#)
 rpoisline, [12](#), [26](#)
 rpoislinetess, [13](#), [26](#)
 rpoislpp, [14](#), [19](#)
 rpoispp, [6](#), [25](#)
 rpoispp3, [13](#)
 rpoisppOnLines, [6](#), [26](#)
 rpoisppx, [14](#)
 rPoissonCluster, [6](#)
 rshift, [6](#), [26](#), [28](#)
 rSSI, [6](#), [25](#)
 rstrat, [6](#), [23](#), [25](#)
 rStrauss, [6](#), [25](#)
 rStraussHard, [6](#), [26](#)
 rsyst, [6](#), [25](#)
 rthin, [6](#), [7](#), [26](#), [28](#)
 rThomas, [6](#), [20](#), [26](#)
 runifdisc, [6](#), [25](#)

runiflpp, [14](#), [19](#)
 runifpoint, [6](#), [25](#)
 runifpoint3, [13](#)
 runifpointOnLines, [6](#), [26](#)
 runifpointx, [14](#)
 rVarGamma, [6](#), [20](#), [26](#)

 SatPiece, [23](#)
 Saturated, [23](#)
 scalardilate, [8](#)
 scaletointerval, [11](#)
 scan.test, [17](#), [26](#), [27](#)
 sdr, [20](#), [22](#), [25](#)
 segregation.test, [27](#)
 selfcrossing.psp, [12](#)
 selfcut.psp, [12](#)
 setcov, [10](#)
 setminus.owin, [10](#)
 shapley, [8](#)
 sharpen.ppp, [8](#), [16](#), [17](#)
 shift, [8](#), [9](#)
 shift.im, [11](#)
 shift.psp, [12](#)
 shift.tess, [13](#)
 shortside.box3, [13](#)
 shortside.boxx, [14](#)
 simdat, [8](#)
 simplenet, [14](#)
 simplify.owin, [9](#)
 simulate.kppm, [20](#), [26](#)
 simulate.ppm, [6](#), [22](#), [24](#), [26](#)
 simulate.slr, [25](#)
 slr, [25](#)
 Smooth.fv, [17](#)
 Smooth.im, [11](#)
 Smooth.ppp, [8](#), [16](#), [17](#)
 Softcore, [23](#)
 solutionset, [11](#)
 spatialcdf, [16](#)
 spatstat (spatstat-package), [2](#)
 spatstat-package, [2](#)
 spatstat.family, [36](#), [37](#)
 spatstat.options, [9](#), [10](#), [22](#)
 spiders, [8](#), [14](#)
 split.ppp, [8](#)
 spokes, [24](#)
 sporophores, [8](#)
 spruces, [8](#)
 square, [9](#)

 step, [20](#), [22](#)
 Strauss, [23](#)
 StraussHard, [23](#)
 studpermu.test, [26](#)
 subset.hyperframe, [15](#)
 subset.lpp, [14](#)
 subset.ppp, [13](#)
 subset.ppp, [8](#)
 subset.ppx, [14](#)
 subset.psp, [12](#)
 summary, [11](#), [15](#), [23](#)
 summary.kppm, [20](#)
 summary.ppm, [22](#)
 summary.psp, [12](#)
 superimpose, [8](#), [12](#)
 swedishpines, [8](#)

 tail.hyperframe, [15](#)
 tess, [5](#), [12](#)
 thinNetwork, [14](#)
 thomas.estK, [21](#)
 thomas.estpcf, [21](#)
 tile.areas, [13](#)
 tiles, [13](#)
 transect.im, [11](#)
 transmat, [11](#)
 triangulate.owin, [10](#)
 Triplets, [23](#)
 Tstat, [16](#)
 tweak.colourmap, [15](#)

 union.owin, [10](#)
 unique.ppp, [8](#)
 uniquemap.ppp, [8](#)
 unitname.box3, [13](#)
 unitname.ppp, [13](#)
 unitname.ppx, [14](#)
 unmark, [8](#)
 unmark.psp, [12](#)
 update.kppm, [20](#)
 update.ppm, [22](#)
 urkiola, [8](#)

 valid.ppm, [22](#)
 varblock, [17](#), [26](#)
 vargamma.estK, [21](#)
 vargamma.estpcf, [21](#)
 vcov.kppm, [20](#)
 vcov.ppm, [22](#)

vcov.slm, [25](#)
vonn.tess, [12](#)
vertices.linnet, [14](#)
Vmark, [18](#)
volume.box3, [13](#)
volume.boxx, [14](#)

waka, [8](#)
waterstriders, [8](#)
Window, [9](#)
with.fv, [17](#)
with.hyperframe, [15](#)

zapsmall.im, [11](#)