Package ‘spatstat’

June 23, 2021

Version 2.2-0
Date 2021-06-23
Title Spatial Point Pattern Analysis, Model-Fitting, Simulation, Tests
Maintainer Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>
Depends R (>= 3.5.0), spatstat.data (>= 2.1-0), spatstat.geom (>= 2.2-0), spatstat.core (>= 2.2-0), spatstat.linnet (>= 2.2-0), utils
Imports spatstat.utils (>= 2.2-0)
Suggests

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 2000 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(), dpmp() 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],
        Rolf Turner [aut],
        Ege Rubak [aut]

Repository  CRAN

Date/Publication  2021-06-23 07:40:02 UTC

R topics documented:

- spatstat-package .................................................. 2
- beginner ............................................................ 30
- bugfixes ........................................................... 31
- foo ................................................................. 32
- latest.news ......................................................... 33

Index  35

---

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 \texttt{lm} and \texttt{glm}. Fitted models can be printed, plotted, predicted, simulated and so on.

\textbf{Getting Started}

For a quick introduction to spatstat, read the package vignette \textit{Getting started with spatstat} installed with \texttt{spatstat}. To read that document, you can either

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

Once you have installed \texttt{spatstat}, start R and type \texttt{library(spatstat)}. Then type beginner for a beginner’s introduction, or demo(spatstat) for a demonstration of the package’s capabilities.


The \texttt{spatstat} package includes over 50 datasets, which can be useful when learning the package. Type \texttt{demo(data)} to see plots of all datasets available in the package. Type \texttt{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/](https://book.spatstat.org/).

For information about handling data in `shapefiles`, see Chapter 3, or the Vignette *Handling shapefiles in the spatstat package*, installed with `spatstat`, accessible as `vignette('shapefiles')`.

**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.geom` containing functionality for geometrical operations, and defining the main classes of spatial objects
- `spatstat.core` containing the main functions for statistical analysis and modelling of 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:

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

- `ppp(x, y)`: create a point pattern from \((x, y)\)
- `ppp(x, y, xlim, ylim)`: for rectangular window
- `ppp(x, y, poly)`: for polygonal window
- `ppp(x, y, mask)`: for binary image window

- `as.ppp()`: convert other types of data to a `ppp` object
- `clickppp()`: interactively add points to a plot
- `marks<-`, `%mark%`: attach/reassign marks to a point pattern

To simulate a random point pattern:

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

To randomly change an existing point pattern:

- `rshift()`: random shifting of points
- `rjitter()`: apply random displacements to points in a pattern
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.

amacrine  Austin Hughes' rabbit amacrine cells
anemones  Upton-Fingleton sea anemones data
ants      Harkness-Isham ant nests data
bdspots   Breakdown spots in microelectrodes
bei       Tropical rainforest trees
betacells Waessle et al. cat retinal ganglia data
bramblecanes  Bramble Canes data
bronzeﬁlter  Bronze Filter Section data
cells     Crick-Ripley biological cells data
chicago   Chicago crimes
chorley   Chorley-Ribble cancer data
clmfires  Castilla-La Mancha forest fires
copper   Berman-Huntington copper deposits data
dendrite  Dendritic spines
demohyper Synthetic point patterns
demopat   Synthetic point pattern
ﬁnpines  Finnish Pines data
flu       Inﬂuenza virus proteins
gordon   People in Gordon Square, London
gorillas  Gorilla nest sites
hamster  Aherne's hamster tumour data
humberside North Humberside childhood leukaemia data
hyyttiala Mixed forest in Hyyttiälä, Finland
japanesepines Japanese Pines data
lansing  Lansing Woods data
longleaf  Longleaf Pines data
mucosa   Cells in gastric mucosa
murchison Murchison gold deposits
nbﬁres  New Brunswick fires data
nztrees  Mark-Esler-Ripley trees data
osteo    Osteocyte lacunae (3D, replicated)
paracou  Kimboto trees in Paracou, French Guiana
ponderosa Getis-Franklin ponderosa pine trees data
pyramidal Pyramidal neurons from 31 brains
redwood  Strauss-Ripley redwood saplings data
redwoodfull  Strauss redwood saplings data (full set)
residualspaper Data from Baddeley et al (2005)
shapley Galaxies in an astronomical survey
simdat Simulated point pattern (inhomogeneous, with interaction)
spiders Spider webs on mortar lines of brick wall
sporophores Mycorrhizal fungi around a tree
spruces Spruce trees in Saxonia
swedishpines Strand-Ripley Swedish pines data
urkiola Urtiola Woods data
waka Trees in Waka national park
waterstriders Insects on water surface

To manipulate a point pattern:

plot.ppp plot a point pattern (e.g. plot(X))
spatstat.gui::iplot plot a point pattern interactively
edit.ppp interactive text editor
[.ppp extract or replace a subset of a point pattern
pp[subset] or pp[subwindow]
subset.ppp extract subset of point pattern satisfying a condition
superimpose combine several point patterns
by.ppp apply a function to sub-patterns of a point pattern
cut.ppp classify the points in a point pattern
split.ppp divide pattern into sub-patterns
unmark remove marks
npoints count the number of points
coords extract coordinates, change coordinates
marks extract marks, change marks or attach marks
rotate rotate pattern
shift translate pattern
flipxy swap x and y coordinates
reflect reflect in the origin
periodify make several translated copies
affine apply affine transformation
scalardilate apply scalar dilation
density.ppp kernel estimation of point pattern intensity
densityHeat.ppp diffusion kernel estimation of point pattern intensity
Smooth.ppp kernel smoothing of marks of point pattern
mmmark mark value of nearest data point
sharpen.ppp data sharpening
identify.ppp interactively identify points
unique.ppp remove duplicate points
duplicated.ppp determine which points are duplicates
uniquemap.ppp map duplicated points to unique points
connected.ppp find clumps of points
dirichlet compute Dirichlet-Voronoi tessellation
delaunay compute Delaunay triangulation
delaunayDistance graph distance in Delaunay triangulation
convexhull compute convex hull
discretise discretise coordinates
**Pixellate.ppp**  
approximate point pattern by pixel image

**as.im.ppp**  
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).

- **owin**  
  Create a window object  
  `owin(xlim, ylim)` for rectangular window  
  `owin(poly)` for polygonal window  
  `owin(mask)` for binary image window

- **Window**  
  Extract window of another object

- **Frame**  
  Extract the containing rectangle ('frame') of another object

- **as.owin**  
  Convert other data to a window object

- **square**  
  make a square window

- **disc**  
  make a circular window

- **ellipse**  
  make an elliptical window

- **ripras**  
  Ripley-Rasson estimator of window, given only the points

- **convexhull**  
  compute convex hull of something

- **letterR**  
  polygonal window in the shape of the R logo

- **clickpoly**  
  interactively draw a polygonal window

- **clickbox**  
  interactively draw a rectangle

**To manipulate a window:**

- **plot.owin**  
  plot a window.

- **boundingbox**  
  Find a tight bounding box for the window

- **erosion**  
  erode window by a distance r

- **dilation**  
  dilate window by a distance r

- **closing**  
  close window by a distance r

- **opening**  
  open window by a distance r

- **border**  
  difference between window and its erosion/dilation

- **complement.owin**  
  invert (swap inside and outside)

- **simplify.owin**  
  approximate a window by a simple polygon

- **rotate**  
  rotate window

- **flipxy**  
  swap x and y coordinates

- **shift**  
  translate window

- **periodify**  
  make several translated copies

- **affine**  
  apply affine transformation

- **as.data.frame.owin**  
  convert window to data frame

**Digital approximations:**

- **as.mask**  
  Make a discrete pixel approximation of a given window

- **as.im.owin**  
  convert window to pixel image

- **pixellate.owin**  
  convert window to pixel image
**commonGrid**  
find common pixel grid for windows

**nearest.raster.point**  
map continuous coordinates to raster locations

**raster.x**  
raster x coordinates

**raster.y**  
raster y coordinates

**raster.xy**  
raster x and y coordinates

**as.polygonal**  
convert pixel mask to polygonal window

See **spatstat.options** to control the approximation

**Geometrical computations with windows:**

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

**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**.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>im</strong></td>
<td>create a pixel image</td>
</tr>
<tr>
<td><strong>as.im</strong></td>
<td>convert other data to a pixel image</td>
</tr>
<tr>
<td><strong>pixellate</strong></td>
<td>convert other data to a pixel image</td>
</tr>
</tbody>
</table>
spatstat-package

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

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

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

Tessellations
An object of class "tess" represents a tessellation.

tess create a tessellation
quadrats create a tessellation of rectangles
hextess create a tessellation of hexagons
polartess tessellation using polar coordinates
quantess quantile tessellation
venn.tess Venn diagram tessellation
dirichlet compute Dirichlet-Voronoi tessellation of points
delaunay compute Delaunay triangulation of points
as.tess convert other data to a tessellation
plot.tess  plot a tessellation
tiles    extract all the tiles of a tessellation
[.tess   extract some tiles of a tessellation
[<-.tess change some tiles of a tessellation
intersect.tess intersect two tessellations
or restrict a tessellation to a window
chop.tess subdivide a tessellation by a line
rpoislinetess generate tessellation using Poisson line process
tile.areas area of each tile in tessellation
bdist.tiles boundary distance for each tile in tessellation
connected.tess find connected components of tiles
shift.tess shift a tessellation
rotate.tess rotate a tessellation
reflect.tess reflect about the origin
flipxy.tess reflect about the diagonal
affine.tess 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".

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

ppx      create a multidimensional space-time point pattern
coords   extract coordinates
as.hyperframe extract coordinates
subset.ppx extract subset
unitname.ppx name of unit of length
Point patterns on a linear network

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

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

```
lpp create a point pattern on a linear network
methods.lpp methods for lpp objects
subset.lpp method for subset
rpoislpp simulate Poisson points on linear network
runiflpp simulate random points on a linear network
chicago Chicago crime data
dendrite Dendritic spines data
spiders 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.

```
hyperframe create a hyperframe
as.hyperframe convert data to hyperframe
plot.hyperframe plot hyperframe
with.hyperframe evaluate expression using each row of hyperframe
```
Layered objects
A layered object represents data that should be plotted in successive layers, for example, a background and a foreground.

```
layered          create layered object
plot.layered    plot layered object
[.layered        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.

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

II. EXPLORATORY DATA ANALYSIS

Inspection of data:

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

Classical exploratory tools:

```
clarkevans    Clark and Evans aggregation index
fryplot       Fry plot
miplot        Morisita Index plot
```

Smoothing:

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

Modern exploratory tools:

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

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

- `intensity`  Mean intensity
- `quadratcount`  Quadrat counts
- `intensity.quadratcount`  Mean intensity in quadrats
- `Fest`  empty space function $F$
- `Gest`  nearest neighbour distribution function $G$
- `Jest`  $J$-function $J = (1 - G) / (1 - F)$
- `Kest`  Ripley’s $K$-function
- `Lest`  Besag $L$-function
- `Tstat`  Third order $T$-function
- `allstats`  all four functions $F, G, J, K$
- `pcf`  pair correlation function
- `Kininhom`  $K$ for inhomogeneous point patterns
- `Lininhom`  $L$ for inhomogeneous point patterns
- `pcfinhom`  pair correlation for inhomogeneous patterns
- `Fininhom`  $F$ for inhomogeneous point patterns
- `Ginhom`  $G$ for inhomogeneous point patterns
- `Jinhom`  $J$ for inhomogeneous point patterns
- `localL`  Getis-Franklin neighbourhood density function
- `localK`  neighbourhood $K$-function
- `localpcf`  local pair correlation function
- `localKininhom`  local $K$ for inhomogeneous point patterns
- `localLininhom`  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

loohboot

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

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 $g_{i\bullet}$

pcfmulti
general pair correlation function

markconnect
marked connection function $p_{ij}$

alltypes
estimates of the above for all $i, j$ pairs

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

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

For marks of any type, there are the following:

- `Gmulti`: multitype nearest neighbour distribution
- `Kmulti`: multitype \(K\) function
- `Jmulti`: multitype \(J\) function

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

Programming tools:

- `applynbd`: apply function to every neighbourhood in a point pattern
- `markstat`: apply function to the marks of neighbours in a point pattern
- `marktable`: tabulate the marks of neighbours in a point pattern
- `pppdist`: 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:

- `linearK`: \(K\) function on linear network
- `linearKinhom`: inhomogeneous \(K\) function on linear network
- `linearpcf`: pair correlation function on linear network
- `linearpcfinhom`: inhomogeneous pair correlation on linear network
For multitype patterns:

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

Related facilities:

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

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

**Summary statistics for a three-dimensional point pattern:**

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

- `F3est` empty space function \( F \)
- `G3est` nearest neighbour function \( G \)
- `K3est` \( K \)-function
- `pcf3est` pair correlation function

Related facilities:

- `envelope.pp3` simulation envelopes
- `pairdist.pp3` distances between all pairs of points
- `crossdist.pp3` distances between points in two patterns
- `nndist.pp3` nearest neighbour distances
- `nnwhich.pp3` find nearest neighbours
- `nncross.pp3` find nearest neighbours in another pattern

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

- `pairdist.ppx`: distances between all pairs of points
- `crossdist.ppx`: distances between points in two patterns
- `nndist.ppx`: nearest neighbour distances
- `nnwhich.ppx`: find nearest neighbours

Summary statistics for random sets:
These work for point patterns (class ppp), line segment patterns (class psp) or windows (class owin).

- `Hest`: spherical contact distribution $H$
- `Gfox`: Foxall $G$-function
- `Jfox`: 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.

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

- `lgcp.estK`: fit a log-Gaussian Cox process model
- `lgcp.estpcf`: fit a log-Gaussian Cox process model
- `thomas.estK`: fit the Thomas process model
- `thomas.estpcf`: fit the Thomas process model
**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"):

- `ppm(X)` Complete Spatial Randomness
- `ppm(X ~ 1)` Complete Spatial Randomness
- `ppm(X ~ x)` Poisson process with intensity loglinear in `x` coordinate
- `ppm(X ~ 1, Strauss(0.1))` Stationary Strauss process
- `ppm(X ~ x, Strauss(0.1))` Strauss process with conditional intensity loglinear in `x`

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

**Manipulating the fitted model:**

<table>
<thead>
<tr>
<th>command</th>
<th>model</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>plot.ppm</code></td>
<td>Plot the fitted model</td>
</tr>
<tr>
<td><code>predict.ppm</code></td>
<td>Compute the spatial trend and conditional intensity of the fitted point process model</td>
</tr>
<tr>
<td><code>coef.ppm</code></td>
<td>Extract the fitted model coefficients</td>
</tr>
<tr>
<td><code>parameters</code></td>
<td>Extract all model parameters</td>
</tr>
<tr>
<td><code>formula.ppm</code></td>
<td>Extract the trend formula</td>
</tr>
<tr>
<td><code>intensity.ppm</code></td>
<td>Compute fitted intensity</td>
</tr>
<tr>
<td><code>Kmodel.ppm</code></td>
<td>$K$ function of fitted model</td>
</tr>
</tbody>
</table>
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.

- \( X \sim 1 \)  
  No trend (stationary)
- \( X \sim x \)  
  Loglinear trend \( \lambda(x, y) = \exp(\alpha + \beta x) \)  
  where \( x, y \) are Cartesian coordinates
- \( X \sim \text{polynom}(x, y, 3) \)  
  Log-cubic polynomial trend
- \( X \sim \text{harmonic}(x, y, 2) \)  
  Log-harmonic polynomial trend
- \( X \sim Z \)  
  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:

- `Poisson()`  
  the Poisson point process
- `AreaInter()`  
  Area-interaction process
- `BadGey()`  
  multiscale Geyer process
- `Concom()`  
  connected component interaction
- `DiggleGratton()`  
  Diggle-Gratton potential
- `DiggleGatesStibbard()`  
  Diggle-Gates-Stibbard potential
- `Fiksel()`  
  Fiksel pairwise interaction process
- `Geyer()`  
  Geyer’s saturation process
Hardcore()  Hard core process
HierHard()  Hierarchical multitype hard core process
HierStrauss()  Hierarchical multitype Strauss process
HierStraussHard()  Hierarchical multitype Strauss-hard core process
Hybrid()  Hybrid of several interactions
LennardJones()  Lennard-Jones potential
MultiHard()  multitype hard core process
MultiStrauss()  multitype Strauss process
MultiStraussHard()  multitype Strauss-hard core process
OrdThresh()  Ord process, threshold potential
Ord()  Ord model, user-supplied potential
PairPiece()  pairwise interaction, piecewise constant
Pairwise()  pairwise interaction, user-supplied potential
Penttinen()  Penttinen pairwise interaction
SatPiece()  Saturated pair model, piecewise constant potential
Saturated()  Saturated pair model, user-supplied potential
Softcore()  pairwise interaction, soft core potential
Strauss()  Strauss process
StraussHard()  Strauss/hard core point process
Triplets()  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.

```r
quadscheme  default quadrature scheme
            using rectangular cells or Dirichlet cells
pixelquad  quadrature scheme based on image pixels
quad  create an object of class "quad"
```

To inspect a quadrature scheme:

```r
plot(Q)  plot quadrature scheme Q
print(Q)  print basic information about quadrature scheme Q
summary(Q)  summary of quadrature scheme Q
```

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

```r
default.dummy  default pattern of dummy points
gridcentres  dummy points in a rectangular grid
rstrat  stratified random dummy pattern
spokes  radial pattern of dummy points
corners  dummy points at corners of the window
```

To compute weights:
Simulation and goodness-of-fit for fitted models:

- `rmh.ppm`: simulate realisations of a fitted model
- `simulate.ppm`: simulate realisations of a fitted model
- `envelope`: 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.

- `lppm`: point process model on linear network
- `anova.lppm`: analysis of deviance for point process model on linear network
- `envelope.lppm`: simulation envelopes for point process model on linear network
- `fitted.lppm`: fitted intensity values
- `predict.lppm`: model prediction on linear network
- `linim`: pixel image on linear network
- `plot.linim`: plot a pixel image on linear network
- `eval.linim`: evaluate expression involving images
- `linfun`: function defined on linear network
- `methods.linfun`: 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"):

<table>
<thead>
<tr>
<th>command</th>
<th>model</th>
</tr>
</thead>
<tbody>
<tr>
<td>slrm(X ~ 1)</td>
<td>Complete Spatial Randomness</td>
</tr>
<tr>
<td>slrm(X ~ x)</td>
<td>Poisson process with intensity loglinear in ( x ) coordinate</td>
</tr>
<tr>
<td>slrm(X ~ Z)</td>
<td>Poisson process with intensity loglinear in covariate ( Z )</td>
</tr>
</tbody>
</table>

**Manipulating a fitted spatial logistic regression**

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

- **runifpoint** generate \( n \) independent uniform random points
- **rpoint** generate \( n \) independent random points
- **rmpoint** generate \( n \) independent multitype random points
- **rpoispp** simulate the (in)homogeneous Poisson point process
- **rmpoispp** simulate the (in)homogeneous multitype Poisson point process
- **runifdisc** generate \( n \) independent uniform random points in disc
- **rstrat** stratified random sample of points
- **rsyst** systematic random sample (grid) of points
- **rMaternI** simulate the Matérn Model I inhibition process
- **rMaternII** simulate the Matérn Model II inhibition process
- **rSSI** simulate Simple Sequential Inhibition process
- **rHardcore** simulate hard core process (perfect simulation)
- **rStrauss** simulate Strauss process (perfect simulation)
- **rStraussHard** simulate Strauss-hard core process (perfect simulation)
- **rDiggleGratton** simulate Diggle-Gratton process (perfect simulation)
- **rDGS** simulate Diggle-Gates-Stibbard process (perfect simulation)
- **rPenttinen** simulate Penttinen process (perfect simulation)
- **rNeymanScott** simulate a general Neyman-Scott process
rMatClust simulate the Matérn Cluster process
rThomas simulate the Thomas process
rLGCP simulate the log-Gaussian Cox process
rGaussPoisson simulate the Gauss-Poisson cluster process
rCauchy simulate Neyman-Scott process with Cauchy clusters
rVarGamma simulate Neyman-Scott process with Variance Gamma clusters
rcell simulate the Baddeley-Silverman cell process
runifpointOnLines generate $n$ random points along specified line segments
rpoisppOnLines generate Poisson random points along specified line segments

Resampling a point pattern:

- quadratresample block resampling
- rjitter apply random displacements to points in a pattern
- rshift random shifting of (subsets of) points
- rthin 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:

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

Simulation-based inference

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

VIII. TESTS AND DIAGNOSTICS

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

More recently-developed tests:

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

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

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

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

parres Partial residual plot
addvar Added variable plot
rholat Kernel estimate of covariate effect
rho2hat 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.

diagnose.ppm diagnostic plots for spatial trend
qqplot.ppm diagnostic Q-Q plot for interpoint interaction
residualspaper examples from Baddeley et al (2005)
Kcom model compensator of $K$ function
Gcom model compensator of $G$ function
Kres score residual of $K$ function
**Gres** score residual of \( G \) function

**psst** pseudoscore residual of summary function

**pssta** pseudoscore residual of empty space function

**psstG** pseudoscore residual of \( G \) function

**compareFit** 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:

- **quadratresample** block resampling
- **rjitter** apply random displacements to points in a pattern
- **rshift** random shifting of (subsets of) points
- **rthin** 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 Aström, Robert Aue, Marcel Austenfeld, Sandro Azaele, Malissa Baddeley, Guy Bayegnak, Colin Beale, Melanie Bell, Thomas Bendtsen, Ricardo Bernhardt, Andrew Bevan, Brad Biggerstaff, Anders Bilgraau, Leanne Bischof, Christophe Biscio, Roger Bivand, Jose M. Blanco Moreno, Florent Bonneu, Jordan Brown, Ian Buller, Julian Burgos, Simon Byers, Ya-Mei Chang, Jianbao Chen, Igor Chernayavsky, Y.C. Chin, Bjärke 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,

Author(s)
Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

References


---

**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 <r.turner@auckland.ac.nz>

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` package.

#### Usage

```
bugfixes(sinceversion = NULL, sincedate = NULL,
         package = "spatstat", show = TRUE)
```

#### Arguments

- `sinceversion`: Earliest version of package for which bugs should be listed. The default is the current installed version.
- `sincedate`: Earliest release date of package for which bugs should be listed. A character string or a date-time object.
- `package`: Character string. The name of the package for which bugs are to be listed.
- `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 `sincedate` should be a character string like "2015-05-27", or a date-time object.

If `sinceversion="all"` or `sincedate="all"` then all recorded bugs will be listed.
If package="spatstat" (the default) then sinceversion="book" and sincedate="book" are interpreted to mean sinceversion="1.42-1", 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.

Value

A data frame, belonging to the class "bugtable", which has its own print method.

Author(s)

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

References


See Also

latest.news, news.

Examples

bugfixes
  ## show all bugs reported after publication of the spatstat book
if(interactive()) bugfixes(sinceversion="1.42-1")
  ## equivalent to bugfixes(sinceversion="book")

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

x  Ignored.
... 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 <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

beginner

Examples

foo

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", doBrowse=FALSE, major=TRUE)
Arguments

package Name of package for which the latest news should be printed.
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 the spatstat package.

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> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

news, bugfixes

Examples

if(interactive()) {
  # current news
  latest.news

  # all news
  news(package="spatstat")
}

latest.news
Index

* documentation
  beginner, 30
  bugfixes, 31
  foo, 32
  latest/news, 33
* package
  spatstat-package, 2
* spatial
  spatstat-package, 2
[.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, 17
amacrine, 7
anemones, 7
angles.psp, 12
anova.lppm, 24
anova.ppm, 22, 27
anova.sirm, 25
ants, 7
aplynbdd, 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, 14
as.im, 10
as.im.owin, 9
as.im.psp, 9
as.interact, 22
as.mask, 9, 10
as.mask.psp, 12
as.matrix.im, 11
as.owin, 9
as.polygonal, 10
as.ppp, 6
as.psp, 11
as.tess, 12
BadGey, 22
bdist.pixels, 10
bdist.points, 10
bdist.tiles, 10, 13
bdspots, 7
beachcoloursmap, 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, 34
bw.abram, 16
bw.CvL, 16
bw.diggle, 16
bw.frac, 16
bw.ppl, 16
crossdist.pp3, 19
bw.relrisk, 16
crossdist.pp, 20
bw.scott, 16
cut.im, 11
bw.smoothppp, 16
cut.ppp, 8, 18
bw.stoyan, 16
data, 7
by.ppp, 8
dclf.progress, 27
cbind.hyperframe, 15
dclf.test, 27
cdf.test, 27
default.dummy, 23
cells, 7
delaunay, 8, 12
centroid.owin, 10
delaunayDistance, 8
chorley, 7
delaunayNetwork, 14
clarkevans, 15
demohyper, 7
clarkevans.test, 27
demopat, 7
clickbox, 9
dendrite, 7, 14
clickdist, 10
density.lpp, 19
clickjoin, 14
density.ppp, 8, 10, 15–17
clickpoly, 9
density.psp, 12
clickppp, 6
densityHeat.lpp, 19
clickppp, 6
densityHeat.ppp, 8, 17
clmfires, 7
deriv.fv, 17
closing, 9
dfbetas.ppm, 27
crossing.psp, 19
crossdist.ppp, 20
crossdist.pp3, 19
crossdist.ppx, 20
cbdists.psp, 12
dilated.areas, 10
density.ppp, 12
dilation, 9
density.psp, 12
dirichlet, 8, 12
densityHeat.ppp, 8, 17
dirichletNetwork, 14
density.lpp, 19
dirichletWeights, 24
density.ppp, 8, 10, 15–17
disc, 9
density.psp, 12
disc, 9
discretise, 8
distfun, 17
discretise, 8
distfun.lpp, 19
discretise, 8
distfun.psp, 12
distfun.psp, 12
distfun.psp, 12
convexhull, 8–10
distmap, 17
dconvexhull, 8–10
distmap.psp, 12
convhull, 8–10
distmap.psp, 12
coord, 8, 13
distmap.psp, 12
copper, 7
distmap.owin, 10
corners, 23
distmap.pp, 10
crossdist, 17
drop1, 20, 22
crossdist.lpp, 19
duplicated.ppp, 8
<table>
<thead>
<tr>
<th>Index Term</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>edges</td>
<td>10, 11</td>
</tr>
<tr>
<td>edit.ppp</td>
<td>8</td>
</tr>
<tr>
<td>effectfun</td>
<td>22</td>
</tr>
<tr>
<td>ellipse</td>
<td>9</td>
</tr>
<tr>
<td>Emark</td>
<td>18</td>
</tr>
<tr>
<td>endpoints.psp</td>
<td>12</td>
</tr>
<tr>
<td>envelope</td>
<td>17, 24, 26, 27</td>
</tr>
<tr>
<td>envelope.lpp</td>
<td>19</td>
</tr>
<tr>
<td>envelope.lppm</td>
<td>24</td>
</tr>
<tr>
<td>envelope.pp3</td>
<td>13, 19</td>
</tr>
<tr>
<td>eroded.areas</td>
<td>10</td>
</tr>
<tr>
<td>eroded.volumes</td>
<td>13</td>
</tr>
<tr>
<td>envelope.lppm</td>
<td>24</td>
</tr>
<tr>
<td>envelope.pp3</td>
<td>13, 19</td>
</tr>
<tr>
<td>eroded.volumes.boxx</td>
<td>14</td>
</tr>
<tr>
<td>erosion</td>
<td>9</td>
</tr>
<tr>
<td>eval.fasp</td>
<td>17</td>
</tr>
<tr>
<td>eval.fv</td>
<td>17</td>
</tr>
<tr>
<td>eval.im</td>
<td>11</td>
</tr>
<tr>
<td>eval.linim</td>
<td>24</td>
</tr>
<tr>
<td>exactdt</td>
<td>17</td>
</tr>
<tr>
<td>extrapolate.psp</td>
<td>12</td>
</tr>
<tr>
<td>F3est</td>
<td>19</td>
</tr>
<tr>
<td>Fest</td>
<td>16</td>
</tr>
<tr>
<td>Fiksel</td>
<td>22</td>
</tr>
<tr>
<td>Finhom</td>
<td>16</td>
</tr>
<tr>
<td>finpines</td>
<td>7</td>
</tr>
<tr>
<td>fitin</td>
<td>22</td>
</tr>
<tr>
<td>fitted.kppm</td>
<td>20</td>
</tr>
<tr>
<td>fitted.lppm</td>
<td>24</td>
</tr>
<tr>
<td>fitted.ppm</td>
<td>22</td>
</tr>
<tr>
<td>fitted.slrm</td>
<td>25</td>
</tr>
<tr>
<td>flipxy</td>
<td>8, 9, 12</td>
</tr>
<tr>
<td>flipxy.tess</td>
<td>13</td>
</tr>
<tr>
<td>flu</td>
<td>7</td>
</tr>
<tr>
<td>foo</td>
<td>32</td>
</tr>
<tr>
<td>formula.kppm</td>
<td>20</td>
</tr>
<tr>
<td>formula.ppm</td>
<td>21</td>
</tr>
<tr>
<td>Frame</td>
<td>9</td>
</tr>
<tr>
<td>fryplot</td>
<td>15</td>
</tr>
<tr>
<td>G3est</td>
<td>19</td>
</tr>
<tr>
<td>Gcom</td>
<td>27</td>
</tr>
<tr>
<td>Gcross</td>
<td>17</td>
</tr>
<tr>
<td>Gdot</td>
<td>17</td>
</tr>
<tr>
<td>Gest</td>
<td>16</td>
</tr>
<tr>
<td>Geyer</td>
<td>22</td>
</tr>
<tr>
<td>Gfox</td>
<td>20</td>
</tr>
<tr>
<td>Ginhom</td>
<td>16</td>
</tr>
<tr>
<td>glm</td>
<td>3</td>
</tr>
<tr>
<td>Gmulti</td>
<td>17, 18</td>
</tr>
<tr>
<td>gordon</td>
<td>7</td>
</tr>
<tr>
<td>gorillas</td>
<td>7</td>
</tr>
<tr>
<td>Gres</td>
<td>28</td>
</tr>
<tr>
<td>gridcentres</td>
<td>23</td>
</tr>
<tr>
<td>gridweights</td>
<td>24</td>
</tr>
<tr>
<td>hamster</td>
<td>7</td>
</tr>
<tr>
<td>Hardcore</td>
<td>23</td>
</tr>
<tr>
<td>harmonise.fv</td>
<td>17</td>
</tr>
<tr>
<td>harmonise.im</td>
<td>11</td>
</tr>
<tr>
<td>head.hyperframe</td>
<td>15</td>
</tr>
<tr>
<td>Hest</td>
<td>20</td>
</tr>
<tr>
<td>hextess</td>
<td>12</td>
</tr>
<tr>
<td>HierHard</td>
<td>23</td>
</tr>
<tr>
<td>HierStrauss</td>
<td>23</td>
</tr>
<tr>
<td>HierStraussHard</td>
<td>23</td>
</tr>
<tr>
<td>hist.im</td>
<td>11</td>
</tr>
<tr>
<td>hsvim</td>
<td>11</td>
</tr>
<tr>
<td>humberside</td>
<td>7</td>
</tr>
<tr>
<td>Hybrid</td>
<td>23</td>
</tr>
<tr>
<td>hyperframe</td>
<td>14</td>
</tr>
<tr>
<td>hyyttiala</td>
<td>7</td>
</tr>
<tr>
<td>identify.ppp</td>
<td>8</td>
</tr>
<tr>
<td>Iest</td>
<td>18</td>
</tr>
<tr>
<td>im</td>
<td>5, 10</td>
</tr>
<tr>
<td>im.apply</td>
<td>11</td>
</tr>
<tr>
<td>imcov</td>
<td>11</td>
</tr>
<tr>
<td>improve.kppm</td>
<td>20</td>
</tr>
<tr>
<td>incircle</td>
<td>10</td>
</tr>
<tr>
<td>influence.ppm</td>
<td>27</td>
</tr>
<tr>
<td>inradius</td>
<td>10</td>
</tr>
<tr>
<td>insertVertices</td>
<td>14</td>
</tr>
<tr>
<td>inside.owin</td>
<td>10</td>
</tr>
<tr>
<td>integral.im</td>
<td>11</td>
</tr>
<tr>
<td>intensity</td>
<td>16</td>
</tr>
<tr>
<td>intensity.ppm</td>
<td>21</td>
</tr>
<tr>
<td>intensity.quadratcount</td>
<td>16</td>
</tr>
<tr>
<td>interp.colourmap</td>
<td>15</td>
</tr>
<tr>
<td>interp.im</td>
<td>11</td>
</tr>
<tr>
<td>intersect.owin</td>
<td>10</td>
</tr>
<tr>
<td>intersect.tess</td>
<td>13</td>
</tr>
<tr>
<td>is.convex</td>
<td>10</td>
</tr>
<tr>
<td>is.hybrid</td>
<td>22</td>
</tr>
<tr>
<td>is.im</td>
<td>11</td>
</tr>
<tr>
<td>is.mask</td>
<td>10</td>
</tr>
<tr>
<td>is.polygonal</td>
<td>10</td>
</tr>
<tr>
<td>is.psp</td>
<td>12</td>
</tr>
</tbody>
</table>
is.rectangle, 10
is.subset.owin, 10
japanese.pines, 7
Jcross, 17
Jdot, 17
Jest, 16
Jfox, 20
Jinhom, 16
Jmulti, 17, 18
joinVertices, 14
K3est, 19
Kcom, 27
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, 21
Kmulti, 17, 18
kppm, 20, 26
Kres, 27
Kscaled, 16
Ksector, 16
lansing, 7
latest.news, 31, 32, 33
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, 20
lgcp.estpcf, 20
lineardisc, 14
linearK, 18
linearKcross, 19
linearKcross.inhom, 19
linearKdot, 19
linearKdot.inhom, 19
linearKinhom, 18
linearmarkconnect, 19
linearmarkequal, 19
linearpcf, 18
linearpcfcross, 19
linearpcfcross.inhom, 19
linearpcf.inhom, 19
linearpcf.inhom, 19
linsfun, 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
localpcf.inhom, 16
logLik.ppm, 22
logLik.slrm, 25
loh.boot, 17, 26
longleaf, 7
lpp, 5, 14
lppm, 24
mad.progress, 27
mad.test, 27
markconnect, 17
markcorr, 18
markcrosscorr, 18
markmarkscatter, 18
markmean, 18
marks, 8
marks.psp, 12
marks<-, 6
marks<-.psp, 12
markstat, 18
marktable, 18
INDEX

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
nearest.segment, 12
news, 32, 34
nnclean, 16
nncross, 12, 17
nncross.lpp, 19
nncross.pp3, 19
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, 19
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, 21
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, 17
Penrith, 23
perimeter, 10
periodify, 8, 9, 12
persp.im, 11
pixelcentres, 10, 11
pixellate, 10
pixellate.linnet, 14
pixellate.owin, 9
pixellate.ppp, 9
pixellate.psp, 12
pixelquad, 23
plot.colourmap, 15
plot.fo (foo), 32
plot.fv, 17
plot.hyperframe, 14
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.slrm, 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.slrm, 25
print.ppm, 22
print.psp, 12
project.ppm, 22
project2segment, 12
psp, 5, 11
psst, 28
psstA, 28
psstG, 28
pyramidal, 7
qqplot.ppm, 26, 27
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, 25
rDiggleGratton, 6, 25
redwood, 7
redwoodfull, 7
reflect, 8
reflect.tess, 13
relrisk, 15–17
repairNetwork, 14
residuals.ppm, 22
residualspaper, 8, 27
rGaussPoisson, 6, 26
rgbim, 11
rHardcore, 6, 25
rho2hat, 16, 27
rhohat, 16, 27
ripnas, 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
rmppppp, 6, 25
rNeymanScott, 6, 25
rnoise, 11
roc, 16
rotate, 8, 9
rotate.im, 11
rotate.psp, 12
rotate.tess, 13
rPenttinen, 6, 25
rpoint, 6, 25
rpoisson, 12, 26
rpoisline, 12, 26
rpoislinetess, 13, 26
rpoislpp, 14, 19
rpoisp, 6, 25
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, 25
rsyst, 6, 25
rthin, 6, 7, 26, 28
rThomas, 6, 20, 26
runifdisc, 6, 25
runiflpp, 14, 19
runifpoint, 6, 25
runifpoint3, 13
INDEX

runifpointOnLines, 6, 26
runifpointx, 14
rVarGamma, 6, 20, 26
SatPiece, 23
Saturated, 23
scalardilate, 8
scaleinterval, 11
scan.test, 17, 26, 27
sdr, 20, 22, 25
segregation.test, 26
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.slrm, 25
slrm, 24
Smooth.fv, 17
Smooth.im, 11
Smooth.ppp, 8, 15–17
Softcore, 23
solutionset, 11
spatialcdf, 16
spatstat (spatstat-package), 2
spatstat-package, 2
spatstat.options, 9, 10, 22
spiders, 8, 14
split.ppp, 8
spokes, 23
sporophores, 8
spruces, 8
square, 9
step, 20, 22
Strauss, 23
StraussHard, 23
studpermu.test, 26
subset.hyperframe, 15
subset.lpp, 14
subset.pp3, 13
subset.ppp, 8
subset.ppx, 13
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, 20
thomas.estpcf, 20
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
uniqueppmap.ppp, 8
unitname.box3, 13
unitname.pp3, 13
unitname.ppx, 13
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.slrm, 25
vnn.tess, 12
vertices.linnet, 14
Vmark, 18
volume.box3, 13
volume.boxx, 14

waka, 8
waterstriders, 8
Window, 9
with.fv, 17
with.hyperframe, 14

zapsmall.im, 11