

R documentation

of ‘spatstat/man/spatstat-package.Rd’

February 5, 2012

spatstat-package *The Spatstat Package*

Description

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

Details

spatstat is a package for the statistical analysis of spatial data. Currently, it deals mainly with the analysis of patterns of points in the plane. The points may carry auxiliary data (‘marks’), and the spatial region in which the points were recorded may have arbitrary shape.

The package supports

- creation, manipulation and plotting of point patterns
- exploratory data analysis
- simulation of point process models
- parametric model-fitting
- hypothesis tests and model diagnostics

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

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 or Huang-Ogata approximate maximum likelihood)
- Cox/cluster process models (by Waagepetersen’s two-step fitting procedure and minimum contrast)

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**, see the package vignette *Getting started with spatstat* installed with **spatstat**. (To see this document online, start R, type `help.start()` to open the help browser, and navigate to `Packages > spatstat > Vignettes`).

For a complete 2-day course on using **spatstat**, see the workshop notes by Baddeley (2010), available on the internet.

Type `demo(spatstat)` for a demonstration of the package's capabilities. Type `demo(data)` to see all the datasets available in the package.

For information about handling data in **shapefiles**, see the Vignette *Handling shapefiles in the spatstat package* installed with **spatstat**.

To learn about spatial point process methods, see the short book by Diggle (2003) and the handbook Gelfand et al (2010).

Updates

New versions of **spatstat** are produced about once a month. 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**. 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)` where `name` is the name of the function or dataset.

CONTENTS:

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

| | |
|-------------------|---|
| <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 <code>ppp</code> 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 Mat\`ern Model I inhibition process |
| <code>rMaternII</code> | simulate the Mat\`ern 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>rDiggleGratton</code> | simulate Diggle-Gratton process (perfect simulation) |
| <code>rDGS</code> | simulate Diggle-Gates-Stibbard process (perfect simulation) |
| <code>rNeymanScott</code> | simulate a general Neyman-Scott process |
| <code>rPoissonCluster</code> | simulate a general Neyman-Scott process |
| <code>rNeymanScott</code> | simulate a general Neyman-Scott process |
| <code>rMatClust</code> | simulate the Mat\`ern 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.

| | |
|-----------------------------|---|
| <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>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 street crimes |
| <code>chorley</code> | Chorley-Ribble cancer data |
| <code>copper</code> | Berman-Huntington copper deposits data |
| <code>demopat</code> | Synthetic point pattern |
| <code>finpines</code> | Finnish Pines data |
| <code>flu</code> | Influenza virus proteins |
| <code>gorillas</code> | Gorilla nest sites |
| <code>hamster</code> | Aherne's hamster tumour data |
| <code>humberside</code> | North Humberside childhood leukaemia data |
| <code>japanesepines</code> | Japanese Pines data |
| <code>lansing</code> | Lansing Woods data |
| <code>longleaf</code> | Longleaf Pines data |
| <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>ponderosa</code> | Getis-Franklin ponderosa pine trees data |
| <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>spruces</code> | Spruce trees in Saxonia |
| <code>swedishpines</code> | Strand-Ripley swedish pines data |
| <code>urkiola</code> | Urkiola Woods data |

To manipulate a point pattern:

| | |
|--------------------------|---|
| <code>plot.ppp</code> | plot a point pattern (e.g. <code>plot(X)</code>) |
| <code>iplot</code> | plot a point pattern interactively |
| <code>[.ppp]</code> | extract or replace a subset of a point pattern <code>pp[subset]</code> or <code>pp[subwindow]</code> |
| <code>superimpose</code> | 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 |
| unmark | remove marks |
| npoints | count the number of points |
| coords | extract coordinates, change coordinates |
| marks | extract marks, change marks or attach marks |
| split.ppp | divide pattern into sub-patterns |
| rotate | rotate pattern |
| shift | translate pattern |
| flipxy | swap x and y coordinates |
| periodify | make several translated copies |
| affine | apply affine transformation |
| density.ppp | kernel smoothing of point pattern |
| smooth.ppp | smooth the marks attached to points |
| sharpen.ppp | data sharpening |
| identify.ppp | interactively identify points |
| unique.ppp | remove duplicate points |
| duplicated.ppp | determine which points are duplicates |
| dirichlet | compute Dirichlet-Voronoi tessellation |
| delaulay | compute 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 |
| as.owin | Convert other data to a window object |
| square | make a square window |
| disc | make a circular 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 |

To manipulate a window:

| | |
|-----------------|--|
| plot.owin | plot a window. plot(W) |
| bounding.box | 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) |

| | |
|----------------------------|--|
| <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 |

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>as.polygonal</code> | convert pixel mask to polygonal window |

See `spatstat.options` to control the approximation

Geometrical computations with windows:

| | |
|-----------------------------|---|
| <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>connected</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>as.mask</code> | pixel approximation of window |
| <code>as.polygonal</code> | polygonal approximation of window |
| <code>setcov</code> | spatial covariance function of window |

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 |

| | |
|------------------|--|
| as.matrix.im | convert pixel image to matrix |
| as.data.frame.im | convert pixel image to data frame |
| 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 |
| shift.im | apply vector shift to pixel 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 |
| connected | 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 |
| scaletointerval | 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 |

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 |
| 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 |
| 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 |

| | |
|-------------------------------|--|
| <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>as.mask.psp</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>crossing.psp</code> | find crossing points between two line segment patterns |
| <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>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>dirichlet</code> | compute Dirichlet-Voronoi tessellation of points |
| <code>delaunay</code> | compute Delaunay triangulation of points |
| <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 |

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>unitname.pp3</code> | name of unit of length |
| <code>npoints</code> | count the number of points |
| <code>runiformpoint3</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>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>simplenet</code> | simple example of network |
| <code>lineardisc</code> | disc in a linear network |
| <code>methods.linnet</code> | methods for <code>linnet</code> objects |

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 <code>lpp</code> objects |
| <code>rpoislpp</code> | simulate Poisson points on linear network |
| <code>runiflpp</code> | simulate random points on a linear network |
| <code>chicago</code> | Chicago street crime data |

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 |

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 |

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>istat(X)</code> | Interactive exploratory analysis |

Classical exploratory tools:

| | |
|-------------------------|-----------------------------------|
| <code>clarkevans</code> | Clark and Evans aggregation index |
| <code>fryplot</code> | Fry plot |
| <code>miplot</code> | Morishita Index plot |

Modern exploratory tools:

| | |
|--------------------------|--|
| <code>nnclean</code> | Byers-Raftery feature detection |
| <code>sharpen.ppp</code> | Choi-Hall data sharpening |
| <code>rhohat</code> | Smoothing estimate of covariate effect |

Summary statistics for a point pattern:

| | |
|----------------------------|---|
| <code>quadratcount</code> | Quadrat counts |
| <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>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 |
| <code>localpcfinhom</code> | local pair correlation for inhomogeneous patterns |
| <code>Kest.fft</code> | fast K -function using FFT for large datasets |
| <code>Kmeasure</code> | reduced second moment measure |
| <code>envelope</code> | simulation envelopes for a summary function |
| <code>varblock</code> | variances and confidence intervals |

for a summary function

Related facilities:

| | |
|--------------------------|---|
| <code>plot.fv</code> | plot a summary function |
| <code>eval.fv</code> | evaluate any expression involving summary functions |
| <code>eval.fasp</code> | evaluate any expression involving an array of functions |
| <code>with.fv</code> | evaluate an expression for a summary function |
| <code>smooth.fv</code> | apply smoothing to a summary function |
| <code>nndist</code> | nearest neighbour distances |
| <code>nnwhich</code> | find nearest neighbours |
| <code>pairdist</code> | distances between all pairs of points |
| <code>crossdist</code> | distances between points in two patterns |
| <code>nncross</code> | nearest neighbours between two point patterns |
| <code>exactdt</code> | distance from any location to nearest data point |
| <code>distmap</code> | distance map image |
| <code>distfun</code> | distance map function |
| <code>density.ppp</code> | kernel smoothed density |
| <code>smooth.ppp</code> | spatial interpolation of marks |
| <code>relrisk</code> | kernel estimate of relative risk |
| <code>sharpen.ppp</code> | data sharpening |
| <code>rknn</code> | 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 `marks(X)` is a factor.

| | |
|--|--|
| <code>relrisk</code> | kernel estimation of relative risk |
| <code>scan.test</code> | spatial scan test of elevated risk |
| <code>Gcross,Gdot,Gmulti</code> | multitype nearest neighbour distributions $G_{ij}, G_{i\bullet}$ |
| <code>Kcross,Kdot,Kmulti</code> | multitype K -functions $K_{ij}, K_{i\bullet}$ |
| <code>Lcross,Ldot</code> | multitype L -functions $L_{ij}, L_{i\bullet}$ |
| <code>Jcross,Jdot,Jmulti</code> | multitype J -functions $J_{ij}, J_{i\bullet}$ |
| <code>pcfcross</code> | multitype pair correlation function g_{ij} |
| <code>pcfdot</code> | multitype pair correlation function $g_{i\bullet}$ |
| <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, Kdot</code> |
| <code>Lcross.inhom,Ldot.inhom</code> | inhomogeneous counterparts of <code>Lcross, Ldot</code> |
| <code>pcfcross.inhom,pcfdot.inhom</code> | inhomogeneous counterparts of <code>pcfcross, pcfdot</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>markvario</code> | mark variogram |
| <code>markcorrint</code> | mark correlation integral |
| <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`).

| | |
|-----------------------------|--|
| <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 |

Related facilities:

| | |
|---------------------------|--|
| <code>pairdist.lpp</code> | shortest path distances |
| <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>pairdist.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 |

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 (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>predict.kppm</code> | Compute fitted intensity |
| <code>update.kppm</code> | Update the model |
| <code>simulate.kppm</code> | Generate simulated realisations |
| <code>vcov.kppm</code> | Variance-covariance matrix of coefficients |
| <code>Kmodel.kppm</code> | K function of fitted model |
| <code>pcfmodel.kppm</code> | Pair correlation of fitted model |

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 Matern Cluster process model |
| <code>matclust.estpcf</code> | fit the Matern 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>formula.ppm</code> | Extract the trend formula |
| <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>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.

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

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>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>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>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 |

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 “ <code>quad</code> ” |

To inspect a quadrature scheme:

```
plot(Q)      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>dirichlet.weights</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>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 |

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

VI. 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 Mat\`ern Model I inhibition process |
| <code>rMaternII</code> | simulate the Mat\`ern Model II inhibition process |
| <code>rSSI</code> | simulate Simple Sequential Inhibition process |
| <code>rStrauss</code> | simulate Strauss process (perfect simulation) |
| <code>rNeymanScott</code> | simulate a general Neyman-Scott process |
| <code>rMatClust</code> | simulate the Mat\`ern 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 |

`rpoisppOnLines` 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.

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>rpoislinetest</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>qqplot.ppm</code> | diagnostic plot for interpoint interaction |
| <code>scan.test</code> | spatial scan statistic/test |

VII. TESTS AND DIAGNOSTICS

Classical hypothesis tests:

| | |
|------------------------------|---|
| <code>quadrat.test</code> | χ^2 goodness-of-fit test on quadrat counts |
| <code>clarkevans.test</code> | Clark and Evans test |
| <code>kstest</code> | Kolmogorov-Smirnov goodness-of-fit test |
| <code>bermantest</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>anova.ppm</code> | Analysis of Deviance for point process models |

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 |

dfbetas.ppm Parameter influence

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 Moller (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 |

VIII. DOCUMENTATION

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

The paper by Baddeley and Turner (2005a) is a brief overview of the package. Baddeley and Turner (2005b) is a more detailed explanation of how to fit point process models to data. Baddeley (2010) is a complete set of notes from a 2-day workshop on the use of `spatstat`.

Type `citation("spatstat")` to get 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, Abdollah Jalilian, Marie-Colette van Lieshout, Ege Rubak, Dominic Schuhmacher and Rasmus Waagepetersen made substantial contributions of code. Additional contributions by Ang Qi Wei, Sandro Azaele, Colin Beale, Ricardo Bernhardt, Brad Biggerstaff, Roger Bivand, Florent Bonneu, Julian Burgos, Simon Byers, Ya-Mei Chang, Jianbao Chen, Igor Chernayavsky, Y.C. Chin, Bjarke Christensen, Marcelino de la Cruz, Peter Dalgaard, Peter Diggle, Ian Dryden, Stephen Eglén, Neba Funwi-Gabga,

Agnes Gault, Marc Genton, Pavel Grabarnik, C. Graf, Janet Franklin, Ute Hahn, Andrew Hardegen, Mandy Hering, Martin Bogsted Hansen, Martin Hazelton, Juha Heikkinen, Kurt Hornik, Ross Ihaka, Robert John-Chandran, Devin Johnson, Mike Kuhn, Jeff Laake, Robert Lamb, George Leser, Ben Madin, Robert Mark, Jorge Mateu Mahiques, Monia Mahling, Peter McCullagh, Ulf Mehlig, Sebastian Wastl Meyer, Mi Xiangcheng, Jesper Moller, Linda Stougaard Nielsen, Felipe Nunes, Thierry Onkelinx, Evgeni Parilov, Jeff Picka, Adrian Raftery, Matt Reiter, Tom Richardson, Brian Ripley, Barry Rowlingson, John Rudge, Aila Sarkka, Katja Schladitz, Bryan Scott, Vadim Shcherbakov, Shen Guochun, Ida-Maria Sintorn, Yong Song, Malte Spiess, Mark Stevenson, Kaspar Stucki, Michael Sumner, P. Surovy, Ben Taylor, Berwin Turlach, Andrew van Burgel, Tobias Verbeke, Alexandre Villers, Hao Wang, H. Wendrock, Jan Wild and Selene Wong.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>
and Rolf Turner <r.turner@auckland.ac.nz>

References

- Baddeley, A. (2010) *Analysing spatial point patterns in R*. Workshop notes. Version 4.1. CSIRO online technical publication. URL: www.csiro.au/resources/pf16h.html
- Baddeley, A. and Turner, R. (2005a) Spatstat: an R package for analyzing spatial point patterns. *Journal of Statistical Software* **12**:6, 1–42. URL: www.jstatsoft.org, ISSN: 1548-7660.
- 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., Moller, 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 Moller, J. (2011) Score, pseudo-score and residual diagnostics for spatial point process models. To appear in *Statistical Science*.
- Diggle, P.J. (2003) *Statistical analysis of spatial point patterns*, Second edition. Arnold.
- 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.
- Waagepetersen, R. An estimating function approach to inference for inhomogeneous Neyman-Scott processes. *Biometrics* **63** (2007) 252–258.

Index

*Topic **package**
 spatstat-package, 1

*Topic **spatial**
 spatstat-package, 1

[.im, 7
[.ppp, 4
[.psp, 7
[.tess, 8
[<-.im, 7
[<-.tess, 8

affine, 5, 6
affine.psp, 8
AIC, 15
allstats, 10
alltypes, 11
amacrine, 4
anemones, 4
angles.psp, 7
anova.lppm, 16
anova.ppm, 14, 18
anova.slm, 17
ants, 4
applynbd, 12
area.owin, 6
AreaInter, 15
as.box3, 8
as.data.frame.hyperframe, 9
as.data.frame.im, 7
as.data.frame.psp, 7
as.hyperframe, 8, 9
as.im, 6
as.im.owin, 6
as.im.ppp, 5
as.interact, 14
as.mask, 6
as.mask.psp, 8
as.matrix.im, 7
as.owin, 5
as.polygonal, 6
as.ppp, 3
as.psp, 7
as.tess, 8

BadGey, 15
bdist.pixels, 6
bdist.points, 6
bdist.tiles, 6, 8
bei, 4
bermantest, 18
betacells, 4
blur, 7
border, 5
bounding.box, 5
box3, 8
boxx, 9
bramblecanes, 4
bronzefilter, 4
by.ppp, 5

cauchy.estK, 13
cauchy.estpcf, 13
cbind.hyperframe, 9
cells, 4
centroid.owin, 6
chicago, 4, 9
chop.tess, 8
chorley, 4
clarkevans, 10
clarkevans.test, 18
clickjoin, 9
clickppp, 3
closing, 5
coef.ppm, 14
coef.slm, 17
commonGrid, 6, 7
compareFit, 19
compatible.im, 7
complement.owin, 5
connected, 6, 7
contour.im, 7
convexhull, 5, 6
coords, 5, 8, 9
copper, 4
corners, 16
crossdist, 11
crossdist.pp3, 12
crossdist.ppx, 13

- crossing.psp, 8
 cut.im, 7
 cut.ppp, 5, 12

 data, 4
 default.dummy, 16
 delaunay, 5, 8
 demopat, 4
 density.ppp, 5, 6, 11
 density.psp, 8
 dfbetas.ppm, 19
 diagnose.ppm, 19
 diameter.box3, 9
 diameter.boxx, 9
 diameter.owin, 6
 DiggleGatesStibbard, 15
 DiggleGratton, 15
 dilated.areas, 6
 dilation, 5
 dirichlet, 5, 8
 dirichlet.weights, 16
 disc, 5
 discretise, 5
 distfun, 11
 distfun.owin, 6
 distfun.psp, 8
 distmap, 11
 distmap.owin, 6
 distmap.psp, 8
 drop1, 15
 duplicated.ppp, 5

 effectfun, 14
 Emark, 11
 endpoints.psp, 7
 envelope, 10, 16, 18
 envelope.lpp, 12
 envelope.lppm, 16
 envelope.pp3, 8, 12
 eroded.areas, 6
 eroded.volumes, 9
 eroded.volumes.boxx, 9
 erosion, 5
 eval.fasp, 11
 eval.fv, 11
 eval.im, 7
 exactdt, 11

 F3est, 12
 Fest, 10
 Fiksel, 15
 finpines, 4
 fitin, 14

 fitted.ppm, 14
 fitted.slrn, 17
 flipxy, 5-7
 flu, 4
 formula.ppm, 14
 fryplot, 10

 G3est, 12
 Gcom, 19
 Gcross, 11
 Gdot, 11
 Gest, 10
 Geyer, 15
 Gfox, 13
 glm, 1
 Gmulti, 11, 12
 gorillas, 4
 Gres, 19
 gridcentres, 16
 gridweights, 16

 hamster, 4
 Hardcore, 15
 harmonise.im, 7
 Hest, 13
 hist.im, 7
 hsvim, 7
 humberside, 4
 hyperframe, 9

 identify.ppp, 5
 Iest, 11
 im, 2, 6
 imcov, 7
 incircle, 6
 influence.ppm, 18
 inside.owin, 6
 integral.im, 7
 interp.im, 7
 intersect.owin, 6
 intersect.tess, 8
 iplot, 4
 is.convex, 6
 is.im, 7
 is.psp, 7
 is.subset.owin, 6
 istat, 10

 japanesepines, 4
 Jcross, 11
 Jdot, 11
 Jest, 10
 Jfox, 13

- Jmulti, 11, 12
- K3est, 12
- Kcom, 19
- Kcross, 11
- Kcross.inhom, 11
- Kdot, 11
- Kdot.inhom, 11
- Kest, 10
- Kest.fft, 10
- Kinhom, 10
- Kmeasure, 6, 10
- Kmodel.kppm, 13
- Kmulti, 11, 12
- kppm, 13, 18
- Kres, 19
- kstest, 18
- lansing, 4
- layered, 10
- Lcross, 11
- Lcross.inhom, 11
- Ldot, 11
- Ldot.inhom, 11
- lengths.psp, 7
- LennardJones, 15
- Lest, 10
- letterR, 5
- levelset, 7
- leverage.ppm, 18
- lgcp.estK, 13
- lgcp.estpcf, 13
- lineardisc, 9
- linearK, 12
- linearKinhom, 12
- linearpcf, 12
- linearpcfinhom, 12
- Linhom, 10
- linim, 16
- linnet, 9
- lm, 1
- localK, 10
- localKinhom, 10
- localL, 10
- localLinhom, 10
- localpcf, 10
- localpcfinhom, 10
- logLik.ppm, 14
- logLik.slrn, 17
- longleaf, 4
- lpp, 3, 9
- lppm, 16
- markconnect, 11
- markcorr, 11
- markcorrint, 11
- markmean, 11
- marks, 5
- marks.psp, 7
- marks<-, 3
- marks<- .psp, 7
- markstat, 12
- marktable, 12
- markvar, 11
- markvario, 11
- matclust.estK, 13
- matclust.estpcf, 13
- mean.im, 7
- methods.linnet, 9
- methods.lpp, 9
- midpoints.psp, 7
- mincontrast, 13
- miplot, 10
- model.depends, 14
- model.frame.ppm, 14
- model.images, 14
- MultiHard, 15
- MultiStrauss, 15
- MultiStraussHard, 15
- murchison, 4
- nbfires, 4
- nearest.raster.point, 6
- nearestsegment, 8
- nnclean, 10
- nncross, 8, 11
- nndist, 11
- nndist.pp3, 12
- nndist.ppx, 13
- nnmean, 12
- nnvario, 12
- nnwhich, 11
- nnwhich.pp3, 12
- nnwhich.ppx, 13
- npoints, 5, 8, 9
- nztrees, 4
- opening, 5
- Ord, 15
- OrdThresh, 15
- osteo, 4
- owin, 2, 5
- pairedist, 11
- pairedist.lpp, 12
- pairedist.pp3, 12

pairdist.ppx, 13
 PairPiece, 15
 Pairwise, 15
 pcf, 10
 pcf3est, 12
 pcfcross, 11
 pcfcross.inhom, 11
 pcfdot, 11
 pcfdot.inhom, 11
 pcfinhom, 10
 pcfmodel.kppm, 13
 perimeter, 6
 periodify, 5, 6, 8
 persp.im, 7
 pixellate, 6
 pixellate.owin, 6
 pixellate.ppp, 5
 pixellate.psp, 8
 pixelquad, 15
 plot.fv, 11
 plot.hyperframe, 9
 plot.im, 7
 plot.kppm, 13
 plot.layered, 10
 plot.linim, 16
 plot.owin, 5
 plot.pp3, 8
 plot.ppm, 14
 plot.ppp, 4
 plot.psp, 7
 plot.slrn, 17
 plot.tess, 8
 pointsOnLines, 8
 Poisson, 15
 ponderosa, 4
 pp3, 3, 8
 ppm, 14, 18
 ppp, 2, 3
 pppdist, 12
 ppx, 3, 9
 predict.kppm, 13
 predict.lppm, 16
 predict.ppm, 14
 predict.slrn, 17
 print.ppm, 14
 print.psp, 7
 project.ppm, 14
 project2segment, 8
 psp, 3, 7
 psst, 19
 psstA, 19
 psstG, 19
 qqplot.ppm, 18, 19
 quad, 15
 quadrat.test, 18
 quadratcount, 10
 quadratresample, 4, 18, 19
 quadrats, 8
 quadscheme, 15
 quantile.im, 7

 raster.x, 6
 raster.y, 6
 rbind.hyperframe, 9
 rCauchy, 3, 13, 17
 rcell, 3, 17
 rDGS, 3
 rDiggieGratton, 3
 redwood, 4
 redwoodfull, 4
 relrisk, 11
 residuals.ppm, 14
 residualspaper, 4, 19
 rGaussPoisson, 3, 17
 rgbim, 7
 rHardcore, 3
 rhohat, 10
 ripras, 5
 rjitter, 3, 4, 18, 19
 rknn, 11
 rlabel, 4
 rLGCP, 13, 17
 rlinegrid, 8, 18
 rMatClust, 3, 13, 17
 rMaternI, 3, 17
 rMaternII, 3, 17
 rmh, 3, 18
 rmh.ppm, 14, 16
 rMosaicField, 18
 rMosaicSet, 18
 rmpoint, 3, 17
 rmpoispp, 3, 17
 rNeymanScott, 3, 17
 rotate, 5, 6
 rotate.psp, 7
 rpoint, 3, 17
 rpoisline, 8, 18
 rpoislinetess, 8, 18
 rpoislp, 9, 12
 rpoispp, 3, 17
 rpoispp3, 8
 rpoisppOnLines, 3, 18
 rpoisppx, 9
 rPoissonCluster, 3
 rshift, 3, 18, 19

- rSSI, 3, 17
- rstrat, 3, 16, 17
- rStrauss, 3, 17
- rsyst, 3, 17
- rthin, 3, 4, 18, 19
- rThomas, 3, 13, 17
- runifdisc, 3, 17
- runiflpp, 9, 12
- runifpoint, 3, 17
- runifpoint3, 8
- runifpointOnLines, 3, 17
- runifpointx, 9
- rVarGamma, 3, 13, 17

- SatPiece, 15
- Saturated, 15
- scaletointerval, 7
- scan.test, 11, 18
- selfcrossing.psp, 8
- setcov, 6
- setminus.owin, 6
- shapley, 4
- sharpen.ppp, 5, 10, 11
- shift, 5, 6
- shift.im, 7
- shift.psp, 8
- shortside.box3, 9
- shortside.boxx, 9
- simdat, 4
- simplenet, 9
- simplify.owin, 6
- simulate.kppm, 13, 18
- simulate.ppm, 3, 14, 16, 18
- simulate.slr, 17
- slr, 16
- smooth.fv, 11
- smooth.ppp, 5, 11
- Softcore, 15
- solutionset, 7
- spatstat (*spatstat-package*), 1
- spatstat-package, 1
- spatstat.options, 5, 6, 15
- split.ppp, 5
- spokes, 16
- spruces, 4
- square, 5
- step, 15
- Strauss, 15
- StraussHard, 15
- summary, 7, 10, 16
- summary.ppm, 14
- summary.psp, 7
- superimpose, 4, 7

- swedishpines, 4

- tess, 3, 8
- thomas.estK, 13
- thomas.estpcf, 13
- tile.areas, 8
- tiles, 8
- Triplets, 15
- Tstat, 10

- union.owin, 6
- unique.ppp, 5
- unitname.box3, 9
- unitname.pp3, 8
- unitname.ppx, 9
- unmark, 5
- unmark.psp, 7
- update.kppm, 13
- update.ppm, 14
- urkiola, 4

- valid.ppm, 14
- varblock, 10, 18
- vargamma.estK, 13
- vargamma.estpcf, 13
- vcov.kppm, 13
- vcov.ppm, 14
- Vmark, 12
- volume.box3, 9
- volume.boxx, 9

- with.fv, 11
- with.hyperframe, 9

- zapsmall.im, 7