**Spatstat Quick Reference guide**

June 15, 2018

---

**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. Its main focus is the analysis of spatial patterns of points in two-dimensional space. The points may carry auxiliary data (‘marks’), and the spatial region in which the points were recorded may have arbitrary shape.

The package 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.

Getting Started

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

- visit \texttt{cran.r-project.org/web/packages/spatstat} and click on \textit{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 \texttt{beginner} for a beginner’s introduction, or \texttt{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 \texttt{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, \texttt{spatstat.github.io/book}.

For information about handling data in \texttt{shapefiles}, see Chapter 3, or the Vignette \textit{Handling shapefiles in the spatstat package}, installed with \texttt{spatstat}, accessible as \texttt{vignette('shapefiles')}.

Updates

New versions of \texttt{spatstat} are released every 8 weeks. Users are advised to update their installation of \texttt{spatstat} regularly.

Type \texttt{latest.news} to read the news documentation about changes to the current installed version of \texttt{spatstat}.

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

Type \texttt{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 \texttt{spatstat} package. Alternatively an alphabetical list of all functions and datasets is available by typing \texttt{library(help=spatstat)}.

For further information on any of these, type \texttt{help(name)} or \texttt{?name} where name is the name of the function or dataset.
I. CREATING AND MANIPULATING DATA

Types of spatial data:

The main types of spatial data supported by \texttt{spatstat} are:

\begin{verbatim}
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
\end{verbatim}

To create a point pattern:

\begin{verbatim}
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 \texttt{ppp} object
clickppp interactively add points to a plot
marks<-, %mark% attach/reassign marks to a point pattern
\end{verbatim}

To simulate a random point pattern:

\begin{verbatim}
runifpoint generate \textit{n} independent uniform random points
rpoint generate \textit{n} independent random points
rmpoint generate \textit{n} independent multitype random points
rpoispp simulate the (in)homogeneous Poisson point process
rmpoispp simulate the (in)homogeneous multitype Poisson point process
runifdisc generate \textit{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
rStrauss simulate Strauss process (perfect simulation)
rHardcore simulate Hard Core process (perfect simulation)
\end{verbatim}
To randomly change an existing point pattern:

- `rshift` random shifting of points
- `rjitter` apply random displacements to points in a pattern
- `rthin` random thinning
- `rlabel` random (re)labelling of a multitype point pattern
- `quadratresample` block resampling

Standard point pattern datasets:

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

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

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

- `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
To manipulate a point pattern:

- plot.ppp: plot a point pattern (e.g. plot(X))
- iplot: plot a point pattern interactively
- edit.ppp: interactive text editor
- [.ppp: extract or replace a subset of a point pattern
- 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
Smooth.ppp  kernel smoothing of marks of point pattern
nnmark     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
connected.ppp find clumps of points
dirichlet   compute Dirichlet-Voronoi tessellation
delaunay    compute Delaunay triangulation
delaunayDistance graph distance in Delaunay triangulation
cvxhull     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
    cvxhull      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.
        plot(W)
    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:

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

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

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

**Line segment patterns**

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>psp</code></td>
<td>create a line segment pattern</td>
</tr>
<tr>
<td><code>as.psp</code></td>
<td>convert other data into a line segment pattern</td>
</tr>
<tr>
<td><code>edges</code></td>
<td>extract edges of a window</td>
</tr>
</tbody>
</table>
The `spatstat-package` includes several functions for working with line segment patterns and tessellations.

### Line Segment Patterns
- **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
- **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
- **nncross**: 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
- **quantess**: quantile tessellation
- **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
- **chop.tess**: or restrict a tessellation to a window
- **dirichlet**: subdivide a tessellation by a line
- **delaunay**: compute Dirichlet-Voronoi tessellation of points
- **rpoislinetess**: compute Delaunay triangulation of points
- **rpoislinetess**: generate tessellation using Poisson line process
tile.areas  area of each tile in tessellation
bdist.tiles 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".

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
everloppp3 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
npoints count the number of points
runifpointx generate uniform random points
rpoisppx generate Poisson random points
boxx define multidimensional box
diameter.boxx diameter of box
volume.boxx volume of box
shortside.boxx shortest side of box
eroded.volumes.boxx 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).

linnet create a linear network
clickjoin interactively join vertices in network
iplot.linnet interactively plot network
simplenet simple example of network
lineardisc disc in a linear network
delaunayNetwork network of Delaunay triangulation
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.

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

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

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
- `istat(X)` Interactive exploratory analysis
- `View(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
- `relrisk` 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.scott` Scott’s rule of thumb for density estimation
- `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:

- `clusterhset` Allard-Fraley feature detection
- `nnclean` Byers-Raftery feature detection
- `sharpen.ppp` Choi-Hall data sharpening
- `rhohat` 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

Kinhom  
$K$ for inhomogeneous point patterns

Linhom  
$L$ for inhomogeneous point patterns

pcfinhom  
pair correlation for inhomogeneous patterns

Finhom  
$F$ 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

localKinhom  
local $K$ for inhomogeneous point patterns

localLinhom  
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

evelope  
simulation envelopes for a summary function

varblock  
variances and confidence intervals

for a summary function

lohboot  
bootstrap for a summary function

Related facilities:

plot.fv  
plot a summary function

eval.fv  
evaluate any expression involving summary functions

harmonise.fv  
make functions compatible

eval.fasp  
evaluate any expression involving an array of functions

with.fv  
evaluate an expression for a summary function

Smooth.fv  
apply smoothing to a summary function

deriv.fv  
calculate derivative of a summary function

pool.fv  
pool several estimates of a summary function

nndist  
nearest neighbour distances

nnwhich  
find nearest neighbours

pairdist  
distances between all pairs of points

crossdist  
distances between points in two patterns

nncross  
nearest neighbours between two point patterns

exactdt  
distance from any location to nearest data point

distmap  
distance map image

distfun  
distance map function

nnmap  
nearest point image

nnfun  
nearest point function

density.ppp  
kernel smoothed density

Smooth.ppp  
spatial interpolation of marks

relrisk  
kernel estimate of relative risk

sharpen.ppp  
data sharpening

rknn  
theoretical distribution of nearest neighbour distance

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

relrisk  
kern...
scan.test  spatial scan test of elevated risk
Gcross,Gdot,Gmulti  multitype nearest neighbour distributions \( G_{ij}, G_i \cdot \)
Kcross,Kdot,Kmulti  multitype \( K \)-functions \( K_{ij}, K_i \cdot \)
Lcross,Ldot  multitype \( L \)-functions \( L_{ij}, L_i \cdot \)
Jcross,Jdot,Jmulti  multitype \( J \)-functions \( J_{ij}, J_i \cdot \)
pcf  multitype pair correlation function \( g_{ij} \)
pcfdot  multitype pair correlation function \( g_{i \cdot} \)
pcfmulti  general pair correlation function
markconnect  marked connection function \( p_{ij} \)
alltypes  estimates of the above for all \( i, j \) pairs
Iest  multitype \( I \)-function
Kcross.inhom,Kdot.inhom  inhomogeneous counterparts of \( K_{ij}, K_i \cdot \)
Lcross.inhom,Ldot.inhom  inhomogeneous counterparts of \( L_{ij}, L_i \cdot \)
pcf.inhom,pcfdot.inhom  inhomogeneous counterparts of \( g_{ij}, g_{i \cdot} \)

**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
- `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
- **ncross.lpp**  nearest neighbour distances
- **nnwhich.lpp**  find nearest neighbours
- **nnfun.lpp**  find nearest data point
- **density.lpp**  kernel smoothing estimator of intensity
- **distfun.lpp**  distance transform
- **envelope.lpp**  simulation envelopes
- **rpoislpp**  simulate Poisson points on linear network
- **runiflpp**  simulate random points on a linear network

It is also possible to fit point process models to **lpp** objects. 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
- **ncross.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
ndist.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 the log-Gaussian Cox process model
thomas.estK    fit the Thomas process model
thomas.estpcf  fit the Thomas process model
matclust.estK  fit the Matern Cluster process model
matclust.estpcf fit the Matern Cluster process model
cauchy.estK    fit a Neyman-Scott Cauchy cluster process
cauhcy.estpcf  fit a Neyman-Scott Cauchy cluster process
vargamma.estK  fit a Neyman-Scott Variance Gamma process
vargamma.estpcf fit a Neyman-Scott Variance Gamma process
mincontrast    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"):

<table>
<thead>
<tr>
<th>command</th>
<th>model</th>
</tr>
</thead>
<tbody>
<tr>
<td>ppm(X)</td>
<td>Complete Spatial Randomness</td>
</tr>
<tr>
<td>ppm(X ~ 1)</td>
<td>Complete Spatial Randomness</td>
</tr>
<tr>
<td>ppm(X ~ x)</td>
<td>Poisson process with intensity loglinear in x coordinate</td>
</tr>
<tr>
<td>ppm(X ~ 1, Strauss(0.1))</td>
<td>Stationary Strauss process</td>
</tr>
<tr>
<td>ppm(X ~ x, Strauss(0.1))</td>
<td>Strauss process with conditional intensity loglinear in x</td>
</tr>
</tbody>
</table>

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

Manipulating the fitted model:

<table>
<thead>
<tr>
<th>function</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>plot.ppm</td>
<td>Plot the fitted model</td>
</tr>
<tr>
<td>predict.ppm</td>
<td>Compute the spatial trend and conditional intensity of the fitted point process model</td>
</tr>
<tr>
<td>coef.ppm</td>
<td>Extract the fitted model coefficients</td>
</tr>
<tr>
<td>parameters</td>
<td>Extract all model parameters</td>
</tr>
<tr>
<td>formula.ppm</td>
<td>Extract the trend formula</td>
</tr>
<tr>
<td>intensity.ppm</td>
<td>Compute fitted intensity</td>
</tr>
<tr>
<td>Kmodel.ppm</td>
<td>K function of fitted model</td>
</tr>
<tr>
<td>pcfmodel.ppm</td>
<td>pair correlation of fitted model</td>
</tr>
<tr>
<td>fitted.ppm</td>
<td>Compute fitted conditional intensity at quadrature points</td>
</tr>
<tr>
<td>residuals.ppm</td>
<td>Compute point process residuals at quadrature points</td>
</tr>
<tr>
<td>update.ppm</td>
<td>Update the fit</td>
</tr>
<tr>
<td>vcov.ppm</td>
<td>Variance-covariance matrix of estimates</td>
</tr>
<tr>
<td>rmh.ppm</td>
<td>Simulate from fitted model</td>
</tr>
<tr>
<td>simulate.ppm</td>
<td>Simulate from fitted model</td>
</tr>
<tr>
<td>print.ppm</td>
<td>Print basic information about a fitted model</td>
</tr>
<tr>
<td>summary.ppm</td>
<td>Summarise a fitted model</td>
</tr>
<tr>
<td>effectfun</td>
<td>Compute the fitted effect of one covariate</td>
</tr>
<tr>
<td>logLik.ppm</td>
<td>log-likelihood or log-pseudolikelihood</td>
</tr>
<tr>
<td>anova.ppm</td>
<td>Analysis of deviance</td>
</tr>
<tr>
<td>model.frame.ppm</td>
<td>Extract data frame used to fit model</td>
</tr>
<tr>
<td>model.images</td>
<td>Extract spatial data used to fit 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`.

**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.

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

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
**Spatstat-package**

**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**.

- **quadscheme**: default quadrature scheme
- **pixelquad**: quadrature scheme based on image pixels
- **quad**: create an object of class "quad"

To inspect a quadrature scheme:

- **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:

- **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:

- **gridweights**: quadrature weights by the grid-counting rule
- **dirichletWeights**: quadrature weights are Dirichlet tile areas

**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.ppm**: 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.

- **lpp**: 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  
plot.linim  
eval.linim  
linfun  
methods.linfun

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"):

command  
slrm(X ~ 1)  Complete Spatial Randomness
slrm(X ~ x)  Poisson process with intensity loglinear in x coordinate
slrm(X ~ Z)  Poisson process with intensity loglinear in covariate Z

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
- `rmppoispp` 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)
- `rDiggleGatton` simulate Diggle-Gatton 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:
**spatstat-package**

- **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
- **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

**Diagnostics for covariate effect:**

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

- **parres**: Partial residual plot
- **addvar**: Added variable plot
- **rhohat**: 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, 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, Jens Åström, Marcel Austenfeld, Sandro Azaele, Malissa Baddeley, Guy Bayegnak, Colin Beale,
spatstat-package


Author(s)

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

References


Index

*Topic package
  spatstat-package, 1
*Topic spatial
  spatstat-package, 1
[.im, 9
[.layered, 12
[.ppp, 6
[.psp, 10
[.tess, 10
[<-.im, 9
[<-.tess, 10

addvar, 23
affine, 6, 8
affine.im, 9
affine.psp, 10
AIC, 17, 19
allstats, 13
alltypes, 15
amacrine, 5
anemones, 5
angles.psp, 10
anova.lppm, 20
anova.ppm, 18, 23
anova.slrm, 21
ants, 5
applynb, 15
area.owin, 8
AreaInter, 19
as.box3, 11
as.data.frame.hyperframe, 12
as.data.frame.im, 9
as.data.frame.ppp, 8
as.data.frame.psp, 10
as.function.im, 9
as.hyperframe, 11, 12
as.im, 9
as.im.owin, 8
as.im.ppp, 7
as.interact, 19
as.mask, 8
as.mask.psp, 10
as.matrix.im, 9
as.owin, 7
as.polygonal, 8
as.ppp, 4
as.psp, 9
as.tess, 10
BadGey, 19
bdist.pixels, 8
bdist.points, 8
bdist.tiles, 8, 11
bdspots, 5
beachcolourmap, 12
bei, 5
berman.test, 23
betacells, 5
bits.test, 23
blur, 9
border, 7
boundingbox, 7
box3, 11
boxx, 11
bramblecanes, 5
bronzefilter, 5
bw.diggle, 13
bw.frac, 13
bw.ppl, 13
bw.relrisk, 13
bw.scott, 13
bw.smoothppp, 13
bw.stoyan, 13
by.ppp, 6
cauchy.estK, 17
cauchy.estpcf, 17
cbind.hyperframe, 12
cdf.test, 23
cells, 5
centroid.owin, 8
chicago, 5, 12
chop.tess, 10
chorley, 5
clarkevans, 13
clarkevans.test, 23
clickbox, 7
clickdist, 8
INDEX

formula.kppm, 17
formula.ppm, 18
Frame, 7
fryplot, 13

G3est, 16
Gcom, 24
Gcross, 15
Gdot, 15
Gest, 13
Geyer, 19
Gfox, 17
Ginhom, 14

glm, 2
Gmulti, 15
gordon, 6
gorillas, 6
Gres, 24
gridcentres, 20
gridweights, 20

hamster, 6
Hardcore, 19
harmonise.fv, 14
harmonise.im, 9
head.hyperframe, 12
Hest, 17
hextess, 10
HierHard, 19
HierStrauss, 19
HierStraussHard, 19
hist.im, 9

humberside, 6
Hybrid, 19, 20
hyperframe, 12
hyytiala, 6

identify.ppp, 7

lem, 4, 9
lemcov, 9
improve.kppm, 17
incircle, 8
influence.ppm, 23
inradius, 8
inside.owin, 8
integral.im, 9
intensity, 13

intercept.ppm, 18

interpolate.ppm, 23

K3est, 16
Kcom, 24
Kcross, 15
Kcross.inhom, 15
Kdot, 15
Kdot.inhom, 15
Kest, 13
Kest.fft, 14
Kinhom, 14
Kmulti, 15
Kmark, 15
Kmeasure, 9, 14
Kmodel.kppm, 17
Kmodel.ppm, 18
Kmulti, 15
kppm, 17, 22
Kres, 24
Kscaled, 14
Ksector, 14

lansing, 6
layered, 12
Lcross, 15
Lcross.inhom, 15
Ldot, 15
Ldot.inhom, 15
lengths.psp, 10
LennardJones, 19
Lest, 13
letterR, 7
levelset, 9
leverage.ppm, 23
INDEX

lgcp.estK, 17
lgcp.estpcf, 17
lineardisc, 11
lineark, 15
linearKcross, 16
linearKcross.inhom, 16
linearKdot, 16
linearKdot.inhom, 16
linearKinhom, 15
linearmarkconnect, 16
linearmarkequal, 16
linearpcfcross, 16
linearpcfcross.inhom, 16
linearpcf, 15
linearpcfdot, 16
linearpcfdot.inhom, 16
lines, 11
linfun, 21
Linhom, 14
linim, 21
linnet, 11
lm, 2
localK, 14
localKinhom, 14
locall, 14
localLinhom, 14
localpcf, 14
localpcfinhom, 14
logLik.ppm, 18
logLik.slrm, 21
lohboot, 14, 22
longleaf, 6
lpp, 4, 12
lppm, 20

mad.progress, 23
mad.test, 23
markconnect, 15
markcorr, 15
markcrosscorr, 15
markmean, 15
marks, 6
marks.psp, 10
marks<-, 4
marks<-.psp, 10
markstat, 15
marktable, 15
markvar, 15
markvario, 15
matclust.estK, 17
matclust.estpcf, 17
mean.im, 9
methods.linfun, 21
methods.linnet, 12
methods.lpp, 12
midpoints.psp, 10
mincontrast, 17
miplot, 13
model.depends, 19
model.frame.ppm, 18
model.images, 18
mucosa, 6
MultiHard, 19
MultiStrauss, 19
MultiStraussHard, 19
murchison, 6
nbfires, 6
nearest.raster.point, 8
nearestsegment, 10
nnclean, 13
nncross, 10, 14
nncross.lpp, 16
nncross.pp3, 16
nndist, 14
nndist.lpp, 16
nndist.pp3, 16
nndist.ppx, 17
nnfun, 14
nnfun.lpp, 16
nnmap, 14
nnmark, 7
nnmean, 15
nnvario, 15
nnwhich, 14
nnwhich.lpp, 16
nnwhich.pp3, 16
nnwhich.ppx, 17
npoints, 6, 11
nztrees, 6

opening, 7
Ord, 19
OrdThresh, 19
osteo, 6
owin, 4, 7

pairdist, 14
pairdist.lpp, 16
pairdist.pp3, 16
PairPiece, 19
Pairwise, 19
paracou, 6
parameters, 17, 18
parres, 23
<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>PAGES</th>
</tr>
</thead>
<tbody>
<tr>
<td>pcf, pcf3est, pcfcross, pcfmodel.kppm, pcfmodel.ppm, pcfmulti, Penttinen, perimeter, periodify, persp.im, pixelcentres, pixellate, pixellate.linnet, pixellate.owin, pixellate.ppp, pixellate.psp, pixelquad, plot.colourmap, plot.fv, plot.linim, plot.owin, plot.pp3, plot.ppm, plot.ppp, plot.psp, plot.slrm, plot.tess, pointsOnLines, Poisson, ponderosa, pool.fv, pp3, ppm, ppp, pppdist, ppx, predict.kppm, predict.lppm, predict.ppm, predict.slrm, print.ppm, print.psp, project.ppm, project2segment</td>
<td>13, 14, 16, 15, 14, 18, 20, 15, 12, 8, 12, 17, 18, 4, 4, 11, 15, 17, 20, 18, 21, 18, 10, 19, 10</td>
</tr>
<tr>
<td>psp, psst, psstA, psstG, pyramidal, qqplot.ppm, quad, quadrat.test, quadratcount, quadratresample, quadrats, quantess, quantile.im, raster.x, raster.xy, raster.y, rbind.hyperframe, rCauchy, rcell, rDGS, rDiggleGratton, redwood, redwoodfull, reflect, relrisk, residuals.ppm, residualspspaper, rGaussPoisson, rgbim, rHardcore, rho2hat, rhohat, ripras, rjitter, rknn, rLGC, rlinegrid, rMatClust, rMaternI, rMaternII, rmh, rmh.ppm, rMosaicField, rMosaicSet, rmpoint, rmpoispp, rNeymanScott, rnoise, roc</td>
<td>4, 9, 24, 24, 13, 23, 13, 24, 10, 20, 10, 7, 8, 12, 5, 17, 22, 5, 22, 5, 22, 6, 24, 5, 22, 9, 5, 22, 4, 11, 23, 23, 4, 22, 22, 22, 5, 22, 4, 17, 10, 11, 21, 12, 10, 18, 20, 23, 23, 4, 22, 4, 22, 5, 22, 13</td>
</tr>
</tbody>
</table>
INDEX

rotate, 6, 7
rotate.im, 9
rotate.psp, 10
rPenttinen, 5, 22
rpoint, 4, 22
rpoisline, 10, 23
rpoislinetess, 10, 23
rpoislpp, 12, 16
rpoisp, 4, 22
rpoispp, 11
rpoisppOnLines, 5, 22
rpoisppx, 11
rPoissonCluster, 5
rshift, 5, 22, 24
rSSI, 4, 22
rstrat, 4, 20, 22
rStrauss, 4, 22
rStraussHard, 5, 22
rsyst, 4, 22
rthin, 5, 22, 24
rThomas, 5, 17, 22
runifdisc, 4, 22
runiflpp, 12, 16
runifpoint, 4, 22
runifpoint3, 11
runifpointOnLines, 5, 22
runifpointx, 11
rVarGamma, 5, 17, 22
SatPiece, 19
Saturated, 19
scalardilate, 6
scaletointerval, 9
scan.test, 15, 23
sdr, 17, 19, 21
segregation.test, 23
selfcrossing.psp, 10
selfcut.psp, 10
setcov, 8, 9
setminus.owin, 8
shapley, 6
sharpen.ppp, 7, 13, 14
shift, 6, 7
shift.im, 9
shift.psp, 10
shortside.box3, 11
shortside.boxx, 11
simdat, 6
simplenet, 11
simplify.owin, 7
simulate.kppm, 17, 22
simulate.ppm, 5, 18, 20, 22
simulate.slrm, 21
slrm, 21
Smooth.fv, 14
Smooth.im, 9
Smooth.ppp, 7, 13, 14
Softcore, 19
solutionset, 9
spatialcdf, 13
spatstat (spatstat-package), 1
spatstat-package, 1
spatstat.options, 7, 8, 19
spiders, 6, 12
split.ppp, 6
spokes, 20
sporophores, 6
spruces, 6
square, 7
step, 17, 19
Strauss, 19
StraussHard, 20
studpermu.test, 23
subset.hyperframe, 12
subset.lpp, 12
subset.pp3, 11
subset.ppp, 6
subset.ppx, 11
summary, 9, 13, 20
summary.kppm, 17
summary.ppm, 18
summary.psp, 10
superimpose, 6, 10
swedishpines, 6
tail.hyperframe, 12
tess, 4, 10
thomas.estK, 17
thomas.estpcf, 17
tile.areas, 11
tiles, 10
transect.im, 9
transmat, 9
triangulate.owin, 8
Triplets, 20
Tstat, 13
tweak.colourmap, 12
union.owin, 8
unique.ppp, 7
unitname.box3, 11
unitname.ppp, 11
unitname.ppx, 11
unmark, 6
unmark.psp, 10
update.kppm, 17
INDEX

update.ppm, 18
urkiola, 6

valid.ppm, 19
varblock, 14, 22
vargamma.estK, 17
vargamma.estpcf, 17
vcov.kppm, 17
vcov.ppm, 18
vcov.slrm, 21
vertices.linnet, 12
View, 13
Vmark, 15
volume.box3, 11
volume.boxx, 11

waka, 6
waterstriders, 6
Window, 7
with.fv, 14
with.hyperframe, 12
zapsmall.im, 9