

Spatstat Quick Reference

September 4, 2014

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 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 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 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 process models (by Waagepetersen's two-step fitting procedure and minimum contrast, or by composite likelihood)

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

Getting Started

For a quick introduction to **spatstat**, 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. (This is now somewhat out-of-date but it will get you started.)

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**.

See the Vignette *Summary of recent updates*, installed with **spatstat**, which describes the main changes to **spatstat** since the workshop notes were published in 2010,

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:

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

To create a point pattern:

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

To simulate a random point pattern:

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

To randomly change an existing point pattern:

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

Standard point pattern datasets:

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

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

<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>clmfires</code>	Castilla-La Mancha forest fires
<code>copper</code>	Berman-Huntington copper deposits data
<code>demohyper</code>	Synthetic point patterns
<code>demopat</code>	Synthetic point pattern
<code>finpines</code>	Finnish Pines data
<code>flu</code>	Influenza virus proteins
<code>gordon</code>	People in Gordon Square, London
<code>gorillas</code>	Gorilla nest sites
<code>hamster</code>	Aherne's hamster tumour data
<code>humberside</code>	North Humberside childhood leukaemia data

hyytiala	Mixed forest in Hyytialä, Finland
japanesePines	Japanese Pines data
lansing	Lansing Woods data
longleaf	Longleaf Pines data
mucosa	Cells in gastric mucosa
murchison	Murchison gold deposits
nbfires	New Brunswick fires data
nztrees	Mark-Esler-Ripley trees data
osteo	Osteocyte lacunae (3D, replicated)
paracou	Kimboto trees in Paracou, French Guiana
ponderosa	Getis-Franklin ponderosa pine trees data
pyramidal	Pyramidal neurons from 31 brains
redwood	Strauss-Ripley redwood saplings data
redwoodfull	Strauss redwood saplings data (full set)
residualspaper	Data from Baddeley et al (2005)
shapley	Galaxies in an astronomical survey
simdat	Simulated point pattern (inhomogeneous, with interaction)
sporophores	Mycorrhizal fungi around a tree
spruces	Spruce trees in Saxonia
swedishPines	Strand-Ripley Swedish pines data
urkiola	Urkiola Woods data
waka	Trees in Waka national park
waterstriders	Insects on water surface

To manipulate a point pattern:

plot.ppp	plot a point pattern (e.g. plot(X))
iplot	plot a point pattern interactively
[.ppp	extract or replace a subset of a point pattern
	pp[subset] or pp[subwindow]
subset.ppp	extract subset of point pattern satisfying a condition
superimpose	combine several point patterns
by.ppp	apply a function to sub-patterns of a point pattern
cut.ppp	classify the points in a point pattern
split.ppp	divide pattern into sub-patterns
unmark	remove marks
npoints	count the number of points
coords	extract coordinates, change coordinates
marks	extract marks, change marks or attach marks
rotate	rotate pattern
shift	translate pattern
flipxy	swap x and y coordinates
reflect	reflect in the origin
periodify	make several translated copies
affine	apply affine transformation
scalardilate	apply scalar dilation
density.ppp	kernel estimation of point pattern intensity
Smooth.ppp	kernel smoothing of marks of point pattern
nnmark	mark value of nearest data point

<code>sharpen.ppp</code>	data sharpening
<code>identify.ppp</code>	interactively identify points
<code>unique.ppp</code>	remove duplicate points
<code>duplicated.ppp</code>	determine which points are duplicates
<code>connected.ppp</code>	find clumps of points
<code>dirichlet</code>	compute Dirichlet-Voronoi tessellation
<code>delaunay</code>	compute Delaunay triangulation
<code>delaunay.distance</code>	graph distance in Delaunay triangulation
<code>convexhull</code>	compute convex hull
<code>discretise</code>	discretise coordinates
<code>pixellate.ppp</code>	approximate point pattern by pixel image
<code>as.im.ppp</code>	approximate point pattern by pixel image

See `spatstat.options` to control plotting behaviour.

To create a window:

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

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

To manipulate a window:

<code>plot.owin</code>	plot a window. <code>plot(W)</code>
<code>boundingbox</code>	Find a tight bounding box for the window
<code>erosion</code>	erode window by a distance r
<code>dilation</code>	dilate window by a distance r
<code>closing</code>	close window by a distance r
<code>opening</code>	open window by a distance r
<code>border</code>	difference between window and its erosion/dilation
<code>complement.owin</code>	invert (swap inside and outside)
<code>simplify.owin</code>	approximate a window by a simple polygon
<code>rotate</code>	rotate window
<code>flipxy</code>	swap x and y coordinates
<code>shift</code>	translate window

<code>periodify</code>	make several translated copies
<code>affine</code>	apply affine transformation

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

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](#).

im	create a pixel image
as.im	convert other data to a pixel image
pixellate	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
rotate.im	rotate pixel image
shift.im	apply vector shift to pixel image
affine.im	apply affine transformation to image
X	print very basic information about image X
summary(X)	summary of image X
hist.im	histogram of image
mean.im	mean pixel value of image
integral.im	integral of pixel values
quantile.im	quantiles of image
cut.im	convert numeric image to factor image
is.im	test whether an object is a pixel image
interp.im	interpolate a pixel image
blur	apply Gaussian blur to image
Smooth.im	apply Gaussian blur to image
connected.im	find connected components
compatible.im	test whether two images have compatible dimensions
harmonise.im	make images compatible
commonGrid	find a common pixel grid for images
eval.im	evaluate any expression involving images
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
convolve.im	spatial convolution of images
transect.im	line transect of image
pixelcentres	extract centres of pixels

Line segment patterns

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

psp	create a line segment pattern
as.psp	convert other data into a line segment pattern
edges	extract edges of a window

<code>is.psp</code>	determine whether a dataset has class "psp"
<code>plot.psp</code>	plot a line segment pattern
<code>print.psp</code>	print basic information
<code>summary.psp</code>	print summary information
<code>[.psp</code>	extract a subset of a line segment pattern
<code>as.data.frame.psp</code>	convert line segment pattern to data frame
<code>marks.psp</code>	extract marks of line segments
<code>marks<-.psp</code>	assign new marks to line segments
<code>unmark.psp</code>	delete marks from line segments
<code>midpoints.psp</code>	compute the midpoints of line segments
<code>endpoints.psp</code>	extract the endpoints of line segments
<code>lengths.psp</code>	compute the lengths of line segments
<code>angles.psp</code>	compute the orientation angles of line segments
<code>superimpose</code>	combine several line segment patterns
<code>flipxy</code>	swap x and y coordinates
<code>rotate.psp</code>	rotate a line segment pattern
<code>shift.psp</code>	shift a line segment pattern
<code>periodify</code>	make several shifted copies
<code>affine.psp</code>	apply an affine transformation
<code>pixellate.psp</code>	approximate line segment pattern by pixel image
<code>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>hextess</code>	create a tessellation of hexagons
<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>de launay</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>subset.pp3</code>	extract subset of 3-D point pattern
<code>unitname.pp3</code>	name of unit of length
<code>npoints</code>	count the number of points
<code>runifpoint3</code>	generate uniform random points in 3-D
<code>rpoispp3</code>	generate Poisson random points in 3-D
<code>envelope.pp3</code>	generate simulation envelopes for 3-D pattern
<code>box3</code>	create a 3-D rectangular box
<code>as.box3</code>	convert data to 3-D rectangular box
<code>unitname.box3</code>	name of unit of length
<code>diameter.box3</code>	diameter of box
<code>volume.box3</code>	volume of box
<code>shortside.box3</code>	shortest side of box
<code>eroded.volumes</code>	volumes of erosions of box

Multi-dimensional space-time point patterns

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

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

Point patterns on a linear network

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

<code>linnet</code>	create a linear network
<code>clickjoin</code>	interactively join vertices in network
<code>simplenet</code>	simple example of network
<code>lineardisc</code>	disc in a linear network
<code>methods.linnet</code>	methods for linnet 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 lpp objects
<code>subset.lpp</code>	method for subset
<code>rpoislpp</code>	simulate Poisson points on linear network
<code>runiflpp</code>	simulate random points on a linear network
<code>chicago</code>	Chicago 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
<code>[.layered</code>	extract subset of layered object

Colour maps

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

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

II. EXPLORATORY DATA ANALYSIS

Inspection of data:

<code>summary(X)</code>	print useful summary of point pattern <code>X</code>
<code>X</code>	print basic description of point pattern <code>X</code>
<code>any(duplicated(X))</code>	check for duplicated points in pattern <code>X</code>
<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>	Morisita Index plot

Smoothing:

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

Modern exploratory tools:

<code>clusterset</code>	Allard-Fraley feature detection
<code>nnclean</code>	Byers-Raftery feature detection
<code>sharpen.ppp</code>	Choi-Hall data sharpening
<code>rho2hat</code>	Kernel estimate of covariate effect
<code>rho2hat</code>	Kernel estimate of covariate effect
<code>spatialcdf</code>	Spatial cumulative distribution function

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

<code>intensity</code>	Mean intensity
<code>quadratcount</code>	Quadrat counts
<code>intensity.quadratcount</code>	Mean intensity in quadrats
<code>Fest</code>	empty space function F
<code>Gest</code>	nearest neighbour distribution function G
<code>Jest</code>	J -function $J = (1 - G)/(1 - F)$
<code>Kest</code>	Ripley's K -function
<code>Lest</code>	Besag L -function

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
envelope	simulation envelopes for a summary function
varblock	variances and confidence intervals for a summary function
lohboot	bootstrap for a summary function

Related facilities:

plot.fv	plot a summary function
eval.fv	evaluate any expression involving summary functions
harmonise.fv	make functions compatible
eval.fasp	evaluate any expression involving an array of functions
with.fv	evaluate an expression for a summary function
Smooth.fv	apply smoothing to a summary function
deriv.fv	calculate derivative of a summary function
nnlist	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 `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>pcfmulti</code>	general pair correlation function
<code>markconnect</code>	marked connection function p_{ij}
<code>alltypes</code>	estimates of the above for all i, j pairs
<code>Iest</code>	multitype I -function
<code>Kcross.inhom,Kdot.inhom</code>	inhomogeneous counterparts of <code>Kcross, 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`). For unmarked patterns:

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

For multitype patterns:

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

Related facilities:

<code>pairdist.lpp</code>	distances between pairs
<code>crossdist.lpp</code>	distances between pairs
<code>ndist.lpp</code>	nearest neighbour distances
<code>nncross.lpp</code>	nearest neighbour distances
<code>nnwhich.lpp</code>	find nearest neighbours
<code>nnfun.lpp</code>	find nearest data point
<code>distfun.lpp</code>	distance transform
<code>envelope.lpp</code>	simulation envelopes
<code>rpoislpp</code>	simulate Poisson points on linear network
<code>runiflpp</code>	simulate random points on a linear network

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

Summary statistics for a three-dimensional point pattern:

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

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

Related facilities:

<code>envelope.pp3</code>	simulation envelopes
<code>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
<code>nncross.pp3</code>	find nearest neighbours in another pattern

Computations for multi-dimensional point pattern:

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

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

Summary statistics for random sets:

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

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

III. MODEL FITTING (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>fitted.kppm</code>	Compute fitted intensity
<code>predict.kppm</code>	Compute fitted intensity
<code>update.kppm</code>	Update the model
<code>improve.kppm</code>	Refine the estimate of trend
<code>simulate.kppm</code>	Generate simulated realisations
<code>vcov.kppm</code>	Variance-covariance matrix of coefficients
<code>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>intensity.ppm</code>	Compute fitted intensity
<code>Kmodel.ppm</code>	K function of fitted model
<code>pcfmodel.ppm</code>	pair correlation of fitted model
<code>fitted.ppm</code>	Compute fitted conditional intensity at quadrature points
<code>residuals.ppm</code>	Compute point process residuals at quadrature points

<code>update.ppm</code>	Update the fit
<code>vcov.ppm</code>	Variance-covariance matrix of estimates
<code>rmh.ppm</code>	Simulate from fitted model
<code>simulate.ppm</code>	Simulate from fitted model
<code>print.ppm</code>	Print basic information about a fitted model
<code>summary.ppm</code>	Summarise a fitted model
<code>effectfun</code>	Compute the fitted effect of one covariate
<code>loglik.ppm</code>	log-likelihood or log-pseudolikelihood
<code>anova.ppm</code>	Analysis of deviance
<code>model.frame.ppm</code>	Extract data frame used to fit model
<code>model.images</code>	Extract spatial data used to fit model
<code>model.depends</code>	Identify variables in the model
<code>as.interact</code>	Interpoint interaction component of model
<code>fitin</code>	Extract fitted interpoint interaction
<code>is.hybrid</code>	Determine whether the model is a hybrid
<code>valid.ppm</code>	Check the model is a valid point process
<code>project.ppm</code>	Ensure the model is a valid point process

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

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

To specify a point process model:

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

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

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

<code>Poisson()</code>	the Poisson point process
<code>AreaInter()</code>	Area-interaction process
<code>BadGey()</code>	multiscale Geyer process
<code>Concom()</code>	connected component interaction
<code>DiggleGratton()</code>	Diggle-Gratton potential
<code>DiggleGatesStibbard()</code>	Diggle-Gates-Stibbard potential
<code>Fiksel()</code>	Fiksel pairwise interaction process
<code>Geyer()</code>	Geyer’s saturation process
<code>Hardcore()</code>	Hard core process
<code>Hybrid()</code>	Hybrid of several interactions
<code>LennardJones()</code>	Lennard-Jones potential

<code>MultiHard()</code>	multitype hard core process
<code>MultiStrauss()</code>	multitype Strauss process
<code>MultiStraussHard()</code>	multitype Strauss/hard core process
<code>OrdThresh()</code>	Ord process, threshold potential
<code>Ord()</code>	Ord model, user-supplied potential
<code>PairPiece()</code>	pairwise interaction, piecewise constant
<code>Pairwise()</code>	pairwise interaction, user-supplied potential
<code>SatPiece()</code>	Saturated pair model, piecewise constant potential
<code>Saturated()</code>	Saturated pair model, user-supplied potential
<code>Softcore()</code>	pairwise interaction, soft core potential
<code>Strauss()</code>	Strauss process
<code>StraussHard()</code>	Strauss/hard core point process
<code>Triplets()</code>	Geyer triplets process

Note that it is also possible to combine several such interactions using [Hybrid](#).

Finer control over model fitting:

A quadrature scheme is represented by an object of class "quad". To create a quadrature scheme, typically use [quadscheme](#).

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

To inspect a quadrature scheme:

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

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

<code>default.dummy</code>	default pattern of dummy points
<code>gridcentres</code>	dummy points in a rectangular grid
<code>rstrat</code>	stratified random dummy pattern
<code>spokes</code>	radial pattern of dummy points
<code>corners</code>	dummy points at corners of the window

To compute weights:

<code>gridweights</code>	quadrature weights by the grid-counting rule
<code>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
<code>eval.linim</code>	evaluate expression involving images
<code>linfun</code>	function defined on linear network
<code>methods.linfun</code>	conversion facilities

V. MODEL FITTING (SPATIAL LOGISTIC REGRESSION)

Logistic regression

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

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

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

Fitting a spatial logistic regression

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

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

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

Manipulating a fitted spatial logistic regression

<code>anova.slrm</code>	Analysis of deviance
-------------------------	----------------------

<code>coef.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>rmipoispp</code>	simulate the (in)homogeneous multitype Poisson point process
<code>runifdisc</code>	generate n independent uniform random points in disc
<code>rstrat</code>	stratified random sample of points
<code>rsyst</code>	systematic random sample (grid) of points
<code>rMaternI</code>	simulate the Matérn Model I inhibition process
<code>rMaternII</code>	simulate the Matérn Model II inhibition process
<code>rSSI</code>	simulate Simple Sequential Inhibition process
<code>rStrauss</code>	simulate Strauss process (perfect simulation)
<code>rNeymanScott</code>	simulate a general Neyman-Scott process
<code>rMatClust</code>	simulate the Matérn Cluster process
<code>rThomas</code>	simulate the Thomas process
<code>rLGCP</code>	simulate the log-Gaussian Cox process
<code>rGaussPoisson</code>	simulate the Gauss-Poisson cluster process
<code>rCauchy</code>	simulate Neyman-Scott process with Cauchy clusters
<code>rVarGamma</code>	simulate Neyman-Scott process with Variance Gamma clusters
<code>rcell</code>	simulate the Baddeley-Silverman cell process
<code>runifpointOnLines</code>	generate n random points along specified line segments
<code>rpoisppOnLines</code>	generate Poisson random points along specified line segments

Resampling a point pattern:

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

See also `varblock` for estimating the variance of a summary statistic by block resampling, and

`lohboot` for another bootstrap technique.

Fitted point process models:

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

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

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

Other random patterns:

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

Simulation-based inference

<code>envelope</code>	critical envelope for Monte Carlo test of goodness-of-fit
<code>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>cdf.test</code>	Spatial distribution goodness-of-fit test
<code>berman.test</code>	Berman's goodness-of-fit tests
<code>envelope</code>	critical envelope for Monte Carlo test of goodness-of-fit
<code>scan.test</code>	spatial scan statistic/test
<code>dclf.test</code>	Diggle-Cressie-Loosmore-Ford test
<code>mad.test</code>	Mean Absolute Deviation test
<code>dclf.progress</code>	Progress plot for DCLF test
<code>mad.progress</code>	Progress plot for MAD 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
<code>dfbetas.ppm</code>	Parameter influence

Diagnostics for covariate effect:

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

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

Residual diagnostics:

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

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

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

Resampling and randomisation procedures

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

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

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, Dominic Schuhmacher and Rasmus Waagepetersen made substantial contributions of code.

Additional contributions by Ang Qi Wei, Sandro Azaele, Malissa Baddeley, Colin Beale, Melanie Bell, Thomas Bendtsen, Ricardo Bernhardt, Andrew Bevan, Brad Biggerstaff, Leanne Bischof, Roger Bivand, Jose M. Blanco Moreno, Florent Bonneu, Julian Burgos, Simon Byers, Ya-Mei Chang, Jianbao Chen, Igor Chernayavsky, Y.C. Chin, Bjarke Christensen, Jean-Francois Coeurjolly, Robin Corria Ainslie, Marcelino de la Cruz, Peter Dalgaard, Sourav Das, Peter Diggle, Patrick Donnelly, Ian Dryden, Stephen Eglen, Olivier Flores, David Ford, Peter Forbes, Janet Franklin, Funwi-Gabga Neba, Oscar Garcia, Agnes Gault, Marc Genton, Julian Gilbey, Jason Goldstick, Pavel Grabarnik, C. Graf, Ute Hahn, Andrew Hardegen, Mandy Hering, Martin Bøgested Hansen, Martin Hazelton, Juha Heikkinen, Kurt Hornik, Philipp Hunziker, Ross Ihaka, Aruna Jammalamadaka, Robert John-Chandran, Devin Johnson, Mike Kuhn, Jeff Laake, Frederic Lavancier, Tom Lawrence, Robert Lamb, Jonathan Lee, George Leser, Li Haitao, George Limitsios, Ben Madin, Kiran Marchikanti, Jeff Marcus, Robert Mark, Peter McCullagh, Monia Mahling, Jorge Mateu Mahiques, Ulf Mehlig, Sebastian Wastl Meyer, Mi Xiangcheng, Lore De Middeleer, Robin Milne, Jesper Møller, Erika Mudrak, Gopalan Nair, Linda Stougaard Nielsen, Felipe Nunes, Jens Oehlschlägel, Thierry Onkelinx, Sean O’Riordan, Evgeni Parilov, Jeff Picka, Nicolas Picard, Mike Porter, Sergiy Protsiv, Adrian Raftery, Suman Rakshit, Pablo Ramon, Xavier Raynaud, Matt Reiter, Tom Richardson, Brian Ripley, Ted Rosenbaum, Barry Rowlingson, Jason Rudokas, John Rudge, Christopher Ryan, Farzaneh Safavianesh, Aila Särkkä, 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, Thordis Linda Thorarinsdottir, Berwin Turlach, Kevin Ummer, Medha Uppala, Andrew van Burgel, Tobias Verbeke, Mikko Vihtakari, Alexandre Villers, Fabrice Vinatier, Hao Wang, H. Wendrock, Jan Wild, Selene Wong, Mike Zamboni and Achim Zeileis.

Author(s)

Adrian Baddeley <Adrian.Baddeley@uwa.edu.au> <http://www.maths.uwa.edu.au/~adrian/>,
Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

References

- Baddeley, A. (2010) *Analysing spatial point patterns in R*. Workshop notes. Version 4.1. CSIRO online technical publication. URL: www.uwa.edu.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. Groggi, J. Mateu, R. Stoica, and D. Stoyan, editors, *Case Studies in Spatial Point Pattern Modelling*, Lecture Notes in Statistics number 185. Pages 23–74. Springer-Verlag, New York, 2006. ISBN: 0-387-28311-0.
- Baddeley, A., Turner, R., Møller, J. and Hazelton, M. (2005) Residual analysis for spatial point processes. *Journal of the Royal Statistical Society, Series B* **67**, 617–666.
- Baddeley, A., Rubak, E. and Møller, J. (2011) Score, pseudo-score and residual diagnostics for spatial point process models. *Statistical Science* **26**, 613–646.

Baddeley, A., Turner, R., Mateu, J. and Bevan, A. (2013) Hybrids of Gibbs point process models and their implementation. *Journal of Statistical Software* **55**:11, 1–43. <http://www.jstatsoft.org/v55/i11/>

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, 8
[.layered, 11
[.ppp, 5
[.psp, 9
[.tess, 9
[<-.im, 8
[<-.tess, 9

addvar, 23
affine, 5, 7
affine.im, 8
affine.psp, 9
AIC, 18
allstats, 13
alltypes, 14
amacrine, 4
anemones, 4
angles.psp, 9
anova.lppm, 20
anova.ppm, 18, 22
anova.slm, 20
ants, 4
applynbd, 14
area.owin, 7
AreaInter, 18
as.box3, 10
as.data.frame.hyperframe, 11
as.data.frame.im, 8
as.data.frame.psp, 9
as.hyperframe, 10, 11
as.im, 8
as.im.owin, 7
as.im.ppp, 6
as.interact, 18
as.mask, 7
as.mask.psp, 9
as.matrix.im, 8
as.owin, 6
as.polygonal, 7
as.ppp, 3
as.psp, 8
as.tess, 9

BadGey, 18
bdist.pixels, 7
bdist.points, 7
bdist.tiles, 7, 10
beachcolourmap, 11
bei, 4
berman.test, 22
betacells, 4
blur, 8
border, 6
boundingbox, 6
box3, 10
boxx, 10
bramblecanes, 4
bronzefilter, 4
bw.diggle, 12
bw.frac, 12
bw.ppl, 12
bw.relrisk, 12
bw.scott, 12
bw.smoothppp, 12
bw.stoyan, 12
by.ppp, 5

cauchy.estK, 17
cauchy.estpcf, 17
cbind.hyperframe, 11
cdf.test, 22
cells, 4
centroid.owin, 7
chicago, 4, 11
chop.tess, 9
chorley, 4

clarkevans, [12](#)
clarkevans.test, [22](#)
clickbox, [6](#)
clickjoin, [11](#)
clickpoly, [6](#)
clickppp, [3](#)
clmfires, [4](#)
closing, [6](#)
cluster.set, [12](#)
coef.ppm, [17](#)
coef.slm, [21](#)
colourmap, [11](#)
commonGrid, [7](#), [8](#)
compareFit, [23](#)
compatible.im, [8](#)
complement.owin, [6](#)
Concom, [18](#)
connected.im, [8](#)
connected.owin, [7](#)
connected.ppp, [6](#)
contour.im, [8](#)
convexhull, [6](#), [7](#)
convolve.im, [8](#)
coords, [5](#), [10](#)
copper, [4](#)
corners, [19](#)
crossdist, [13](#)
crossdist.lpp, [15](#)
crossdist.pp3, [16](#)
crossdist.ppx, [16](#)
crossing.psp, [9](#)
cut.im, [8](#)
cut.ppp, [5](#), [14](#)

data, [4](#)
dclf.progress, [22](#)
dclf.test, [22](#)
default.dummy, [19](#)
delaunay, [6](#), [10](#)
delaunay.distance, [6](#)
demohyper, [4](#)
demopat, [4](#)
density.ppp, [5](#), [8](#), [12](#), [13](#)
density.psp, [9](#)
deriv.fv, [13](#)
dfbetas.ppm, [22](#)
diagnose.ppm, [23](#)
diameter.box3, [10](#)
diameter.boxx, [10](#)

diameter.owin, [7](#)
DiggleGatesStibbard, [18](#)
DiggleGratton, [18](#)
dilated.areas, [7](#)
dilation, [6](#)
dirichlet, [6](#), [9](#)
dirichlet.weights, [19](#)
disc, [6](#)
discretise, [6](#)
distfun, [13](#)
distfun.lpp, [15](#)
distfun.owin, [7](#)
distfun.psp, [9](#)
distmap, [13](#)
distmap.owin, [7](#)
distmap.psp, [9](#)
drop1, [18](#)
duplicated.ppp, [6](#)

edges, [7](#), [8](#)
effectfun, [18](#)
Emark, [14](#)
endpoints.psp, [9](#)
envelope, [13](#), [20](#), [22](#)
envelope.lpp, [15](#)
envelope.lppm, [20](#)
envelope.pp3, [10](#), [15](#)
eroded.areas, [7](#)
eroded.volumes, [10](#)
eroded.volumes.boxx, [10](#)
erosion, [6](#)
eval.fasp, [13](#)
eval.fv, [13](#)
eval.im, [8](#)
eval.linim, [20](#)
exactdt, [13](#)

F3est, [15](#)
Fest, [12](#)
Fiksel, [18](#)
Finhom, [13](#)
finpines, [4](#)
fitin, [18](#)
fitted.kppm, [16](#)
fitted.ppm, [17](#)
fitted.slm, [21](#)
flipxy, [5](#), [6](#), [9](#)
flu, [4](#)
formula.ppm, [17](#)

Frame, [6](#)
 fryplot, [12](#)

 G3est, [15](#)
 Gcom, [23](#)
 Gcross, [14](#)
 Gdot, [14](#)
 Gest, [12](#)
 Geyer, [18](#)
 Gfox, [16](#)
 Ginhom, [13](#)
 glm, [2](#)
 Gmulti, [14](#)
 gordon, [4](#)
 gorillas, [4](#)
 Gres, [23](#)
 gridcentres, [19](#)
 gridweights, [19](#)

 hamster, [4](#)
 Hardcore, [18](#)
 harmonise.fv, [13](#)
 harmonise.im, [8](#)
 Hest, [16](#)
 hextess, [9](#)
 hist.im, [8](#)
 hsvim, [8](#)
 humberside, [4](#)
 Hybrid, [18](#), [19](#)
 hyperframe, [11](#)
 hyytiala, [5](#)

 identify.ppp, [6](#)
 Iest, [14](#)
 im, [3](#), [8](#)
 imcov, [8](#)
 improve.kppm, [16](#)
 incircle, [7](#)
 influence.ppm, [22](#)
 inside.owin, [7](#)
 integral.im, [8](#)
 intensity, [12](#)
 intensity.ppm, [17](#)
 intensity.quadratcount, [12](#)
 interp.colourmap, [11](#)
 interp.im, [8](#)
 intersect.owin, [7](#)
 intersect.tess, [9](#)
 iplot, [5](#)

 is.convex, [7](#)
 is.hybrid, [18](#)
 is.im, [8](#)
 is.mask, [7](#)
 is.polygonal, [7](#)
 is.psp, [9](#)
 is.rectangle, [7](#)
 is.subset.owin, [7](#)
 istat, [12](#)

 japanesepines, [5](#)
 Jcross, [14](#)
 Jdot, [14](#)
 Jest, [12](#)
 Jfox, [16](#)
 Jinhom, [13](#)
 Jmulti, [14](#)

 K3est, [15](#)
 Kcom, [23](#)
 Kcross, [14](#)
 Kcross.inhom, [14](#)
 Kdot, [14](#)
 Kdot.inhom, [14](#)
 Kest, [12](#)
 Kest.fft, [13](#)
 Kinhom, [13](#)
 Kmeasure, [8](#), [13](#)
 Kmodel.kppm, [16](#)
 Kmodel.ppm, [17](#)
 Kmulti, [14](#)
 kppm, [16](#), [22](#)
 Kres, [23](#)
 Kscaled, [13](#)
 Ksector, [13](#)

 lansing, [5](#)
 layered, [11](#)
 Lcross, [14](#)
 Lcross.inhom, [14](#)
 Ldot, [14](#)
 Ldot.inhom, [14](#)
 lengths.psp, [9](#)
 LennardJones, [18](#)
 Lest, [12](#)
 letterR, [6](#)
 levelset, [8](#)
 leverage.ppm, [22](#)
 lgcp.estK, [16](#)

lgcp.estpcf, 16
 lineardisc, 11
 linearK, 15
 linearKcross, 15
 linearKcross.inhom, 15
 linearKdot, 15
 linearKdot.inhom, 15
 linearKinhom, 15
 linearmarkconnect, 15
 linearmarkequal, 15
 linearpcf, 15
 linearpcfcross, 15
 linearpcfcross.inhom, 15
 linearpcfdot, 15
 linearpcfdot.inhom, 15
 linearpcfinhom, 15
 linfun, 20
 Linhom, 13
 linim, 20
 linnet, 11
 lm, 2
 localK, 13
 localKinhom, 13
 localL, 13
 localLinhom, 13
 localpcf, 13
 localpcfinhom, 13
 logLik.ppm, 18
 logLik.slm, 21
 lohboot, 13, 22
 longleaf, 5
 lpp, 3, 11
 lppm, 20

 mad.progress, 22
 mad.test, 22
 markconnect, 14
 markcorr, 14
 markcorrint, 14
 markmean, 14
 marks, 5
 marks.psp, 9
 marks<-, 3
 marks<- .psp, 9
 markstat, 14
 marktable, 14
 markvar, 14
 markvario, 14
 matclust.estK, 16
 matclust.estpcf, 16
 mean.im, 8
 methods.linfun, 20
 methods.linnet, 11
 methods.lpp, 11
 midpoints.psp, 9
 mincontrast, 17
 miplot, 12
 model.depends, 18
 model.frame.ppm, 18
 model.images, 18
 mucosa, 5
 MultiHard, 19
 MultiStrauss, 19
 MultiStraussHard, 19
 murchison, 5

 nbfires, 5
 nearest.raster.point, 7
 nearestsegment, 9
 nnclean, 12
 nncross, 9, 13
 nncross.lpp, 15
 nncross.pp3, 16
 nndist, 13
 nndist.lpp, 15
 nndist.pp3, 16
 nndist.ppx, 16
 nnfun, 13
 nnfun.lpp, 15
 nnmap, 13
 nnmark, 5
 nnmean, 14
 nnvario, 14
 nnwhich, 13
 nnwhich.lpp, 15
 nnwhich.pp3, 16
 nnwhich.ppx, 16
 npoints, 5, 10
 nztrees, 5

 opening, 6
 Ord, 19
 OrdThresh, 19
 osteo, 5
 owin, 3, 6

 pairedist, 13
 pairedist.lpp, 15

pairdist.pp3, [15](#)
 pairdist.ppx, [16](#)
 PairPiece, [19](#)
 Pairwise, [19](#)
 paracou, [5](#)
 parres, [23](#)
 pcf, [12](#), [13](#)
 pcf3est, [15](#)
 pcfcross, [14](#)
 pcfcross.inhom, [14](#)
 pcfdot, [14](#)
 pcfdot.inhom, [14](#)
 pcfinhom, [13](#)
 pcfmodel.kppm, [16](#)
 pcfmodel.ppm, [17](#)
 pcfmulti, [14](#)
 perimeter, [7](#)
 periodify, [5](#), [7](#), [9](#)
 persp.im, [8](#)
 pixelcentres, [7](#), [8](#)
 pixellate, [8](#)
 pixellate.owin, [7](#)
 pixellate.ppp, [6](#)
 pixellate.psp, [9](#)
 pixelquad, [19](#)
 plot.colourmap, [11](#)
 plot.fv, [13](#)
 plot.hyperframe, [11](#)
 plot.im, [8](#)
 plot.kppm, [16](#)
 plot.layered, [11](#)
 plot.linim, [20](#)
 plot.owin, [6](#)
 plot.pp3, [10](#)
 plot.ppm, [17](#)
 plot.ppp, [5](#)
 plot.psp, [9](#)
 plot.slr, [21](#)
 plot.tess, [9](#)
 pointsOnLines, [9](#)
 Poisson, [18](#)
 ponderosa, [5](#)
 pp3, [3](#), [10](#)
 ppm, [17](#), [22](#)
 ppp, [3](#)
 pppdist, [14](#)
 ppx, [3](#), [10](#)
 predict.kppm, [16](#)
 predict.lppm, [20](#)
 predict.ppm, [17](#)
 predict.slr, [21](#)
 print.ppm, [18](#)
 print.psp, [9](#)
 project.ppm, [18](#)
 project2segment, [9](#)
 psp, [3](#), [8](#)
 psst, [23](#)
 psstA, [23](#)
 psstG, [23](#)
 pyramidal, [5](#)

 qqplot.ppm, [22](#), [23](#)
 quad, [19](#)
 quadrat.test, [22](#)
 quadratcount, [12](#)
 quadratresample, [4](#), [21](#), [23](#)
 quadrats, [9](#)
 quadscheme, [19](#)
 quantile.im, [8](#)

 raster.x, [7](#)
 raster.y, [7](#)
 rbind.hyperframe, [11](#)
 rCauchy, [4](#), [16](#), [21](#)
 rcell, [4](#), [21](#)
 rDGS, [3](#)
 rDiggleGratton, [3](#)
 redwood, [5](#)
 redwoodfull, [5](#)
 reflect, [5](#)
 relrisk, [12–14](#)
 residuals.ppm, [17](#)
 residualspaper, [5](#), [23](#)
 rGaussPoisson, [4](#), [21](#)
 rgbim, [8](#)
 rHardcore, [3](#)
 rho2hat, [12](#), [23](#)
 rhohat, [12](#), [23](#)
 ripras, [6](#)
 rjitter, [3](#), [4](#), [21](#), [23](#)
 rknn, [13](#)
 rlabel, [4](#)
 rLGCP, [16](#), [21](#)
 rlinegrid, [9](#), [22](#)
 rMatClust, [4](#), [16](#), [21](#)
 rMaternI, [3](#), [21](#)
 rMaternII, [3](#), [21](#)

- rmh, [4, 22](#)
- rmh.ppm, [18, 19](#)
- rMosaicField, [22](#)
- rMosaicSet, [22](#)
- rmpoint, [3, 21](#)
- rmppoispp, [3, 21](#)
- rNeymanScott, [3, 4, 21](#)
- rotate, [5, 6](#)
- rotate.im, [8](#)
- rotate.psp, [9](#)
- rpoint, [3, 21](#)
- rpoisline, [9, 22](#)
- rpoislinetest, [10, 22](#)
- rpoislpp, [11, 15](#)
- rpoispp, [3, 21](#)
- rpoispp3, [10](#)
- rpoisppOnLines, [4, 21](#)
- rpoisppx, [10](#)
- rPoissonCluster, [4](#)
- rshift, [4, 21, 23](#)
- rSSI, [3, 21](#)
- rstrat, [3, 19, 21](#)
- rStrauss, [3, 21](#)
- rsyst, [3, 21](#)
- rthin, [4, 21, 23](#)
- rThomas, [4, 16, 21](#)
- runifdisc, [3, 21](#)
- runiflpp, [11, 15](#)
- runifpoint, [3, 21](#)
- runifpoint3, [10](#)
- runifpointOnLines, [4, 21](#)
- runifpointx, [10](#)
- rVarGamma, [4, 16, 21](#)
- SatPiece, [19](#)
- Saturated, [19](#)
- scalardilate, [5](#)
- scaletointerval, [8](#)
- scan.test, [14, 22](#)
- selfcrossing.psp, [9](#)
- setcov, [7, 8](#)
- setminus.owin, [7](#)
- shapley, [5](#)
- sharpen.ppp, [6, 12, 13](#)
- shift, [5, 6](#)
- shift.im, [8](#)
- shift.psp, [9](#)
- shortside.box3, [10](#)
- shortside.boxx, [10](#)
- simdat, [5](#)
- simplenet, [11](#)
- simplify.owin, [6](#)
- simulate.kppm, [16, 22](#)
- simulate.ppm, [4, 18, 20, 22](#)
- simulate.slm, [21](#)
- slrm, [20](#)
- Smooth.fv, [13](#)
- Smooth.im, [8](#)
- Smooth.ppp, [5, 12, 13](#)
- Softcore, [19](#)
- solutionset, [8](#)
- spatialcdf, [12](#)
- spatstat (spatstat-package), [1](#)
- spatstat-package, [1](#)
- spatstat.options, [6, 7, 18](#)
- split.ppp, [5](#)
- spokes, [19](#)
- sporophores, [5](#)
- spruces, [5](#)
- square, [6](#)
- step, [18](#)
- Strauss, [19](#)
- StraussHard, [19](#)
- subset.lpp, [11](#)
- subset.pp3, [10](#)
- subset.ppp, [5](#)
- subset.ppx, [10](#)
- summary, [8, 12, 19](#)
- summary.ppm, [18](#)
- summary.psp, [9](#)
- superimpose, [5, 9](#)
- swedishpines, [5](#)
- tess, [3, 9](#)
- thomas.estK, [16](#)
- thomas.estpcf, [16](#)
- tile.areas, [10](#)
- tiles, [9](#)
- transect.im, [8](#)
- Triplets, [19](#)
- Tstat, [13](#)
- tweak.colourmap, [11](#)
- union.owin, [7](#)
- unique.ppp, [6](#)
- unitname.box3, [10](#)
- unitname.pp3, [10](#)
- unitname.ppx, [10](#)

unmark, [5](#)
unmark.psp, [9](#)
update.kppm, [16](#)
update.ppm, [18](#)
urkiola, [5](#)

valid.ppm, [18](#)
varblock, [13](#), [21](#)
vargamma.estK, [17](#)
vargamma.estpcf, [17](#)
vcov.kppm, [16](#)
vcov.ppm, [18](#)
vcov.slm, [21](#)
Vmark, [14](#)
volume.box3, [10](#)
volume.boxx, [10](#)

waka, [5](#)
waterstriders, [5](#)
Window, [6](#)
with.fv, [13](#)
with.hyperframe, [11](#)

zapsmall.im, [8](#)