spatstat Quick Reference 1.0.1

Type `demo(spatstat)` for an overall demonstration.

Creation, manipulation and plotting of point patterns

An object of class `"ppp"` describes a point pattern. If the points have marks, these are included as a component vector `marks`.

To create a point pattern:

`ppp`	create a point pattern from (x, y) and window information
	`ppp(x, y, xlim, ylim)` for rectangular window
	`ppp(x, y, poly)` for polygonal window
	`ppp(x, y, mask)` for binary image window
`as.ppp`	convert other types of data to a `ppp` object

To simulate a random point pattern:

`runifpoint`	generate n independent uniform random points
`rpoispp`	simulate the (in)homogeneous Poisson point process
`rMaternI`	simulate the Matérn Model I inhibition process
`rMaternII`	simulate the Matérn Model II inhibition process
`rSSI`	simulate Simple Sequential Inhibition
`rNeymanScott`	simulate a general Neyman-Scott process
`rMatClust`	simulate the Matérn Cluster process
`rThomas`	simulate the Thomas process
`rmh`	simulate Gibbs point process using Metropolis-Hastings

Standard point pattern datasets:

Remember to say `data(bramblecanes)` etc.

`bramblecanes`	Bramble Canes data
`catHughes`	Austin Hughes' cat retina data
`catWaessle`	Wässle et al. cat retina data
`cells`	Crick-Ripley biological cells data
`lansing`	Lansing Woods data
`longleaf`	Longleaf Pines data
`nztrees`	Mark-Esler-Ripley trees data
`redwood`	Strauss-Ripley redwood saplings data
`swedishpines`	Strand-Ripley swedish pines data

To manipulate a point pattern:

`plot.ppp`	plot a point pattern `plot(X)`

`"[.ppp"`	extract a subset of a point.pattern
	`pp[subset]`
	`pp[, subwindow]`

`superimpose`	superimpose any number of point patterns
`unmark`	remove marks
`rotate`	rotate pattern
`shift`	translate pattern
`affine`	apply affine transformation

To create a window:

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

`owin`	Create a window object `owin(xlim, ylim)` for rectangular window `owin(poly)` for polygonal window `owin(mask)` for binary image window
`as.owin`	Convert other data to a window object

To manipulate a window:

`plot.owin`	plot a window. `plot(W)`
`bounding.box`	Find a tight bounding box for the window
`erode.owin`	erode window by a distance r
`complement.owin`	invert (inside $\leftrightarrow$ outside)
`rotate`	rotate window
`shift`	translate window
`affine`	apply affine transformation

Digital approximations:

`as.mask`	Make a discrete pixel approximation of a given window
`nearest.raster.point`	map continuous coordinates to raster locations
`raster.x`	raster x coordinates
`raster.y`	raster y coordinates

Geometrical computations with windows:

`inside.owin`	determine whether a point is inside a window
`area.owin`	compute window's area
`diameter`	compute window frame's diameter
`eroded.areas`	compute areas of eroded windows
`bdist.points`	compute distances from data points to window boundary
`bdist.pixels`	compute distances from all pixels to window boundary

Exploratory Data Analysis

Summary statistics for a point pattern:

`Fest`	empty space function F
`Gest`	nearest neighbour distribution function G
`Kest`	Ripley's K-function
`Jest`	J-function $J = (1 - G)/(1 - F)$
`allstats`	all four functions F, G, J, K
`pcf`	pair correlation function
`Kinhom`	K for inhomogeneous point patterns

Summary statistics for a marked point pattern:

A multitype point pattern is represented by an object `X` of class `"ppp"` with a component `X$marks` which is a factor.

`Gcross,Gdot,Gmulti`	multitype nearest neighbour distributions $G_{ij}, G_{i\bullet}$
`Kcross,Kdot, Kmulti`	multitype K-functions $K_{ij}, K_{i\bullet}$
`Jcross,Jdot,Jmulti`	multitype J-functions $J_{ij}, J_{i\bullet}$
`alltypes`	estimates of the above for all i, j pairs

Model Fitting

To fit a point process model:

Model fitting in `spatstat` version 1.0 is performed by the function `mpl`. Its result is an object of class `ppm`.

`mpl`	Fit a point process model to a two-dimensional point pattern
`plot.ppm`	Plot the fitted model
`predict.ppm`	Compute the spatial trend and conditional intensity of the fitted point process model

To specify a point process model:

The first order "trend" of the model is written as an S language formula.

`~1`	No trend (stationary)
`~x`	First order term $\lambda(x, y) = \exp(\alpha + \beta x)$ where x, y are Cartesian coordinates
`~polynom(x,y,3)`	Log-cubic polynomial trend

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

`Poisson()`	the Poisson point process
`Strauss()`	the Strauss process
`StraussHard()`	the Strauss/hard core point process
`Softcore()`	pairwise interaction, soft core potential
`PairPiece()`	pairwise interaction, piecewise constant
`Pairwise()`	pairwise interaction, user-supplied potential
`Geyer()`	Geyer's saturation process
`Saturated()`	Saturated pair model, user-supplied potential
`OrdThresh()`	Ord process, threshold potential
`Ord()`	Ord model, user-supplied potential
`MultiStrauss()`	multitype Strauss process
`MultiStraussHard()`	multitype Strauss/hard core process

Finer control over model fitting:

A quadrature scheme is represented by an object of class `"quad"`.

`quadscheme`	generate a Berman-Turner quadrature scheme for use by `mpl`
`default.dummy`	default pattern of dummy points
`gridcentres`	dummy points in a rectangular grid
`stratrand`	stratified random dummy pattern
`spokes`	radial pattern of dummy points
`corners`	dummy points at corners of the window
`gridweights`	quadrature weights by the grid-counting rule
`dirichlet.weights`	quadrature weights are Dirichlet tile areas

`$Revision: 1.2 $ $Date: 2002/01/21 01:34:48 $`